PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS
ESCOLA POLITÉCNICA
GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

GEOFENCING : MONITORAMENTO DE PROPRIEDADES RURAIS

RAFAEL MIRANDA LÔBO

GOIÂNIA
2022

RAFAEL MIRANDA LÔBO

GEOFENCING : MONITORAMENTO DE PROPRIEDADES RURAIS

Trabalho de Conclusão de Curso apresentado à Escola Politécnica, da Pontifícia Universidade Católica de Goiás, como parte dos requisitos para a obtenção do título de Bacharel em Engenharia de Computação.

Orientador(a): Prof. M.E.E. Marcelo Antonio Adad de Araújo.

GOIÂNIA

2022

RAFAEL MIRANDA LÔBO

GEOFENCING : MONITORAMENTO DE PROPRIEDADES RURAIS

Este trabalho de Conclusão de Curso julgado adequado para obtenção o título de Bacharel em Engenharia de Computação, e aprovado em sua forma final pela Escola Politécnica, da Pontifícia Universidade Católica de Goiás, em. ____/____/_____.

_____

Prof. MSc. Ludmilla Reis Pinheiro dos Santos
Coordenadora de Trabalho de Conclusão de Curso

Banca examinadora:

_____

Orientador: Prof. M.E.E. Marcelo Antonio Adad de Araújo

_____

Prof. M.E.E Carlos Alexandre Ferreira de Lima

_____

Profa. M.E Mírian Sandra Rosa Gusmão

GOIÂNIA

2022

RESUMO

Em prol de atender os produtores rurais, visto que o estado de goiás é predominantemente de agronegócio, o presente trabalho visa a construção de um aplicativo móvel capaz de gerenciar informações do âmbito rural. Com a tecnologia de georreferenciamento, a localização do usuário é monitorada verificando se está dentro de uma área marcada, a demarcação da área pode ser construídas por linhas poligonais fechadas de formas variadas, após ser verificado que o usuário entrou em uma área monitorada as informações pertinentes a ela é exibida. O usuário poderá ter o controle de mais de uma propriedade permitindo cada propriedade conter diversas áreas, para ajudar na administração das propriedades será possível o cadastro de funcionários e a atribuição de permissões, sendo elas, cadastro e edição de áreas, cadastro de novos funcionários e permissão para atribuir ou revogar permissões

Palavra-Chave: *Android*, *geofencing*, banco de dados, produção rural

ABSTRACT

In order to serve rural producers, since the state of Goiás is predominantly agribusiness, this work aims to build a mobile application capable of managing information from the rural environment. With georeferencing technology, the user's location is monitored by checking whether it is within a marked area, the demarcation of the area can be constructed by polygonal lines closed in different ways, after verifying that the user has entered a monitored area according to relevant information it will be displayed. The user will be able to have control of more than one property allowing each property containing several areas, to help in the administration of the properties it will be possible to register employees and grant permissions, namely, registration and editing of areas, registration of new employees and permission to assign or revoke permissions

Keywords: Android, geofencing, database, rural production

LISTA DE FIGURAS

LISTA DE ABREVIATURAS

Me(a)	Mestre(a)
M.E.E	Mestre em engenharia elétrica
M.E	Mestre
MSc	Master of Science
MVVM	*Model / View / ViewModel*
Prof	Professor
Profa	Professora
TCC	Trabalho de Conclusão de Curso
TCC 1	Trabalho de Conclusão de Curso 1
TCC 2	Trabalho de Conclusão de Curso 2

LISTA DE SIGLAS

API	Interface de programação de aplicativos
APP	Aplicação
C/A	*Coarse Acquisition*
DCL	Linguagem de controle de dados
DDL	Biblioteca De Vínculo Dinâmico
GPS	Sistema de Posicionamento Global
GUI	Interface Gráfica do Usuário
HTTP	Protocolo de Transferência de Hipertexto
IDE	Ambiente de Desenvolvimento Integrado
JSON	Notação de Objeto JavaScript
MHz	MegaHeartz
ms	Milissegundos
dotNET	*Framework* para internet
NoSql	Não somente linguagem de consulta estruturada
P	Precisão, próprio do GPS (*Precision Code*)
PRN	Ruído Pseudo Aleatório
RFID	Identificação por radiofrequência
SQL	Linguagem de consulta estruturada
TCL	Linguagem de controle de transações
URL	localizador padrão de recursos
XAML	*Extensible Application Markup Language*
XML	*Extensible Markup Language*

Sumário

1 INTRODUÇÃO

O presente trabalho é uma continuação do trabalho de conclusão de curso do acadêmico da Pontifícia Universidade Católica de Goiás Shirlano Candido Dias Filho, de forma a melhorar e avançar o desenvolvimento anterior em alguns quesitos, formatos poligonais na criação da geofencing para aumentar a precisão na demarcação de áreas para o monitoramento de entrada e saída de um usuário no local e possibilitar ao proprietário atribuir permissões a seus funcionários para o gerenciamento das áreas.

Para este fim, é utilizado a tecnologia *geofencing*, com o uso de tecnologias com identificação por radiofrequência (RFID), sistema de posicionamento global (GPS), conjunto de antenas fixas de celulares e até mesmo alguns sinais de *Wi-Fi* conhecidos de forma a estabelecer um perímetro geográfico virtual (BRANDÃO, 2020).

As áreas de ocupação delimitadas por propriedades rurais tiveram um aumento significativo de 17,6 milhões de hectares (+5,8%) entre 2006 e 2017, de acordo com o Censo Agropecuário 2017 (SILVEIRA; TOOGE, 2019).

A tecnologia *geofencing* trabalha juntamente com o produtor rural de forma a melhorar o controle sobre a produtividade local, administração de recursos encontrados em suas terras, controle de pragas, saneamento animal, controle de doenças tanto em animais como em culturas vegetais ou mesmo para economia de recursos na propriedade.

No presente trabalho a implementação de uma forma do usuário poder delimitar a área utilizando polígonos distintos que não somente círculos é alcançada, aumentando assim a precisão da marcação das diferentes áreas e estruturas relacionadas a respectivas propriedades rurais.

Na integração da tecnologia *geofencing* é necessário a implementação de uma *Application Programming Interface* (API) com o dispositivo mobile que utiliza o sistema operacional *Android*. Trata-se de um conjunto de rotinas e padrões estabelecidos por um determinado software para utilização ou melhora de funcionalidades a serem utilizadas por outros aplicativos (TAKEBLIP, 2019).

A aplicação conta com um sistema de armazenamento de dados próprio, para que o proprietário da área rural possa customizar as informações do campo em questão, ou criar novas áreas administrativas, permitindo que outros usuários

autorizados pelo proprietário também possam relacionar dados diversos, autorizando assim que outras pessoas possam ter acesso às informações dos terrenos, que estejam selecionados ou autorizados pelo proprietário.

1.1 Objetivos

Esta seção visa apresentar e descrever os objetivos gerais e específicos deste trabalho

1.1.1 Objetivo Geral

Desenvolver um aplicativo *Android* utilizando a tecnologia *geofencing,* com o intuito de tornar possível a criação de formas distintas de *geofences* em uma zona rural, sendo capaz de delimitar uma área de forma mais precisa e trazer informações específicas da região pré-determinada pelo usuário responsável, conter informações gerenciáveis para o usuário responsável, sendo possível adicionar novos usuários e novas propriedades com permissões e informações diferentes.

1.1.2 Objetivos Específicos

- Aplicar formas distintas de *geofences,* em diversos tipos de linhas poligonais fechadas, de forma a representar melhor as partes constituintes da localidade rural.
- Possibilitar que mais de um usuário gerencie a região demarcada, dependendo de sua permissão concedida pelo proprietário.
- Produzir um banco de dados para ser gerenciado e vinculado as propriedades rurais do proprietário.
- Desenvolver o sistema com o intuito a facilitar a gestão de várias propriedades rurais, inclusive.

1.2 Justificativa

É relevante estudar este tema porque ajudará o usuário a ter um maior controle sobre suas áreas possibilitando gerenciar seus produtos trazendo assim uma maior facilidade na administração de suas terras.

Com as *geofences* em formatos radial a precisão e um pouco limitada, ela é gerada a partir de uma localização, assim irá criar um raio em torno dela, com a delimitação sendo feita por polígonos e alcançado uma maior precisão.

1.3 Metodologia

Esta pesquisa segundo sua natureza é um resumo do assunto com implementação, pois busca sistematizar a área de aplicações de *geofencing* na zona rural. O resumo de assunto procura sistematizar uma área de conhecimento, no qual indica sua evolução histórica e seu estado da arte, ou seja, adequado para os cursos de graduação (WAZLAWICK, 2014).

A pesquisa bibliográfica envolve o estudo de artigos, teses, livros e outras publicações que geralmente são fornecidos e indexados por editoras. A pesquisa bibliográfica é qualquer trabalho científico, mas não produz nenhum novo conhecimento, ela apenas ajuda o pesquisador a ter informações relevantes e públicas que ele ainda não possuía (WAZLAWICK, 2014).

E, segundo seus procedimentos técnicos, esta pesquisa é bibliográfica e experimental, pois pretende fazer um levantamento teórico da área de aplicações de *geofencing* e implementar esta técnica. Wazlawick (2014) sugere que a pesquisa bibliográfica deve seguir os seguintes passos:

a) Listar os títulos de periódicos e eventos relevantes para o tema de pesquisa e os títulos de periódicos gerais em computação que eventualmente possam ter algum artigo na área do tema de pesquisa.

b) Obter a lista e todos os artigos publicados nos últimos cinco anos (ou mais) nesses veículos.

c) Selecionar desta lista aqueles títulos que tenham relação com o tema de pesquisa.

d) Ler o *abstract* desses artigos e, em função da leitura, classificá-los como relevância “alta”, “média” ou “baixa”.

e) Ler artigos de alta relevância e fazer fichas de leitura anotando os principais conceitos e ideias aprendidos. Anotar também títulos de outros artigos possivelmente mencionados na bibliografia de cada artigo (mesmo que com mais de cinco anos) e que pareçam relevantes para o trabalho de pesquisa. Incluir esses

artigos na lista dos que devem ser lidos (inicialmente o abstract e, se for relevante, o artigo todo).

f) Dependendo do caso, ler também os artigos de relevância média e baixa, mas iniciando sempre pelos de alta relevância.

g) Se já tem material suficiente para elaborar uma ideia de pesquisa consistente.

h) Se precisa expandir a pesquisa examinando artigos mais antigos (expandindo o passo b) ou periódicos menos relevantes (expandindo o passo a).

Para Wazlawick (2014), pesquisa experimental se dá pela manipulação de uma parte da realidade do pesquisador. Na pesquisa experimental é necessário possuir variáveis manipuláveis pelo pesquisador e variáveis de observação. A medição dessa variável de observação pode concluir se existe alguma dependência entre ela e alguma variável manipulável. No caso deste trabalho, as variáveis manipuláveis são as *geofences* e as informações da área do proprietário. As variáveis observadas são a usabilidade, confiabilidade e desempenho.

1.4 Resultados Esperados

Espera-se que os resultados deste trabalho possam AUXILIAR:

- Na gerencia de áreas rurais
- Em um controle maior na produção rural

2 REFERENCIAL TEÓRICO

Neste capítulo, é apresentada a base teórica utilizada para a concepção deste trabalho. O objetivo é fornecer orientações e informações sobre os conceitos utilizados no desenvolvimento de aplicativos *Android*, que constituem o objetivo central do trabalho, junto com a tecnologia *geofencing* e as linguagens C# e Kotlin.

2.1 Coordenadas Geográficas

Para distinguir da obra de TCC anterior a que foi dada a continuidade, as coordenadas geográficas do presente trabalho basearam-se em coordenadas polares.

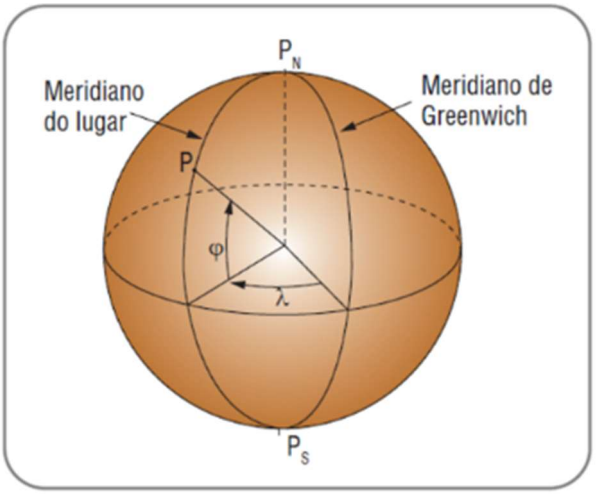
Um sistema de coordenadas é a base para representar localizações geográficas em uma superfície, seja a superfície um elipsoide, esfera ou plano. Para elipsoides ou esferas, utiliza-se sistemas de coordenadas cartesianas e curvilíneas, representadas por linhas paralelas e meridianos, e para planos, utiliza-se sistemas de coordenadas cartesianas X e Y (CARVALHO; ARAÚJO, 2008).

Com coordenadas geográficas, qualquer posição horizontal na Terra pode ser representada por duas das três coordenadas existentes em um sistema de coordenadas esféricas alinhadas com o eixo de rotação da Terra. Letras e números devem ser usados para identificar qualquer lugar na superfície da Terra. Tem-se que utilizar elementos de referência que permitam identificar qualquer lugar do planeta (CARVALHO; ARAÚJO, 2008).

Utiliza-se de cartas cartográficas ou geográficas que se resumem em um conjunto de linhas imaginárias, formadas por paralelos e meridianos, representando coordenadas geográficas (CARVALHO; ARAÚJO, 2008).

Como apresentado na Figura 1, em um *Model*o esférico da Terra, a latitude de um lugar é o ângulo entre os raios de luz solar incidentes que passam por aquele lugar e o plano equatorial. (CARVALHO; ARAÚJO, 2008).

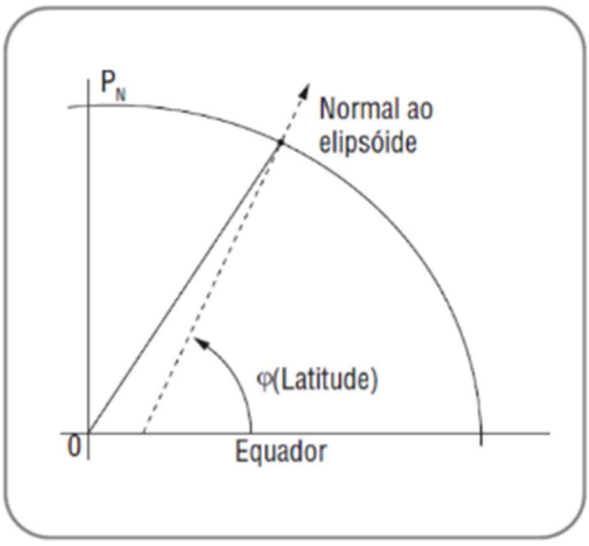
Figura 1 - Coordenadas Geográficas



Fonte: (CARVALHO; ARAUJO,2008).

Como apresentado na Figura 2 no *Model*o elipsoide da Terra, a latitude de um local é o ângulo entre a linha vertical do elipsoide perpendicular a esse local e o plano equatorial. Semelhante ao que acontece com o *Model*o esférico da Terra, as normais elipsoides em diferentes lugares não ocorrem todas simultaneamente no centro da Terra. Por outro lado, como o meridiano não é um círculo, mas uma elipse, não se pode confundir latitude com a medida do ângulo do arco meridiano entre o equador e aquele lugar, como numa esfera. A latitude do local representado no mapa é a latitude geodésica (CARVALHO; ARAÚJO, 2008).

Figura 2 - Latitude geodésica do elipsoide



Fonte: (CARVALHO;ARAUJO,2008).

Na superfície real da Terra, a latitude também pode ser definida como o ângulo entre a vertical de um lugar (ou seja, a direção do fio de prumo) e o plano do Equador. Esta modalidade de latitude (astronômica ou natural) é muitas vezes diferente da latitude indicada no mapa, ou seja, latitude geodésica, uma vez que a vertical do local geralmente não coincide com a normal da elipsoide de referência do local. Muito antes que a forma e o tamanho da Terra fossem conhecidos com precisão, como é o caso hoje, a latitude astronômica era determinada observando as estrelas, usando quadrantes, astrolábios e balestilhas. (CARVALHO; ARAUJO,2008)

2.2 Quadrante

O Quadrante foi citado e descrito em vários escritos medievais, nomeadamente *Libros del Saber* de Astronomia no século XIII. Em sua forma original, certamente era usado para medir alturas e distâncias (CAMÕES, 2002).

O quadrante era construído em madeira e tinha a forma de um quarto de círculo daí o nome. Em uma das bordas retas foi colocada uma pequena agulha com um furo, onde a "estrela foi inserida". No ápice do quadrante, um fio de prumo ligeiramente maior que o raio do instrumento é fixado em um furo. Na extremidade livre há um pequeno peso de metal. As arestas curvas são graduadas em uma escala de 0º a 90º. Para medir a altitude do sol, o usuário deve combinar a luz desta estrela com a luz que passa pelo orifício superior da asa com a luz do orifício inferior da asa (CAMÕES, 2002).

Isto só se consegue colocando o instrumento no meridiano do astro e com uma inclinação muito precisa. Nesse momento, o fio de prumo indica a altura do astro - o ângulo entre o horizonte e o astro - ou a distância do zênite - o ângulo entre o astro e o zênite do observador, dependendo apenas de como o instrumento é dimensionado de 0º A 90º ou de 90º a 0º, das arestas lisas para a aresta das pínulas. A Figura 03 apresenta um exemplo do instrumento Quadrante (CAMÕES, 2002).

Figura 3 - Quadrante Fonte

Fonte: (PUGA, 2019).

2.3 Astrolábios

Esses instrumentos provavelmente surgiram como resultado da simplificação do astrolábio planisférico, que os cosmólogos (aquele que estuda cosmologia) utilizavam para determinar as posições das estrelas no céu, a hora local da altitude do sol ou para resolver problemas geométricos (CAMÕES, 2002).

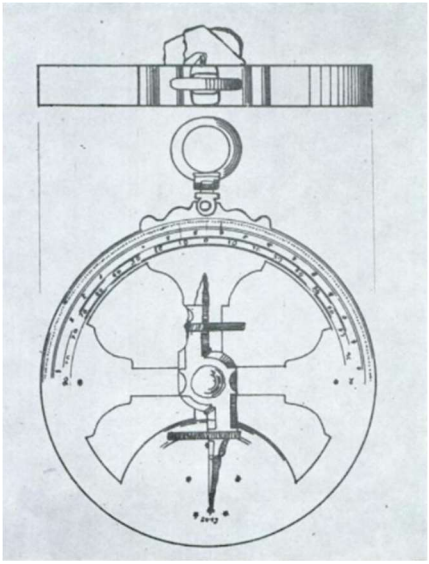
Originalmente era construído em madeira ou chapa de metal, mas logo se concluiu que não era estritamente adequado para uso em barcos. Por isso começou a ser construído em bronze. Este fato lhe confere resistência e peso suficientes para ser utilizado no mar e para minimizar os efeitos do equilíbrio do navio. Mesmo assim, é construído com um corpo volumoso, e tem a desvantagem de que a resistência ao vento é muito grande e difícil de observar. Para evitar esse problema, seu corpo foi aberto, deixando apenas dois diâmetros ortogonais (CAMÕES, 2002).

Uma mediclina (espécie de ponteiro) é girada em seu centro, onde são encontradas duas pínulas (pequena lâmina metálica colocada perpendicularmente nas duas extremidades da alidade, tendo a meio uma fenda para estabelecer os alinhamentos), como no quadrante, por onde o astro e colocado. Normalmente, nos dois quadrantes superiores do astrolábio, são gravadas escalas de 0º a 90º inicialmente uma das alturas. Mais tarde tornou-se a referência Zenith. O último

envolve a redução de cálculos na determinação da latitude através da passagem meridiana do sol (CAMÕES, 2002).

Para medir a altitude, o observador segura o astrolábio pelos anéis isso minimiza os efeitos do balanço do navio no instrumento e faz com que a luz do sol que passa pelo orifício da pínula superior coincida com o orifício da pínula inferior. Se o astrolábio estiver bem estruturado, alinhado e os orifícios forem do tamanho correto, ainda é possível ver a luz do sol caindo no convés depois de passar pelos dois orifícios. A Figura 04 apresenta um exemplo do instrumento Astrolábios (CAMÕES, 2002)

Figura 4 - Astrolábios



Fonte: (GUEDES, 1981)

2.4 Balestilha

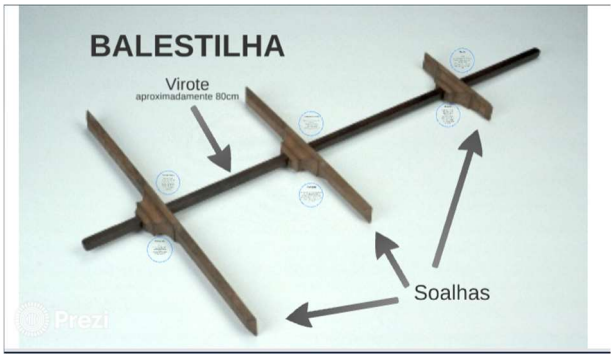
Consiste em uma vara quadrada (quatro escalas) com cerca de 80 cm de comprimento, chamada de virote. Ao longo desse pequeno pedaço de madeira, um tamanho diferente para cada escala, é chamado de soalha. Há uma escala em cada borda do virote (haste de forma quadrangular que constituía a peça principal da balestilha, e onde corria a soalha) dependendo do tamanho da soalha a ser usado (CAMÕES, 2002).

Durante a visão noturna, o observador olha através do orifício na extremidade do virote para ver a estrela tangente à borda superior da soalha e o horizonte tangente à borda inferior. Como o sol não pode ser visto diretamente, ao medir sua

altitude, as observações são feitas para trás, ou seja, de costas para o astro (CAMÕES, 2002).

Neste caso, a sombra da borda superior deve ser projetada no meio da soalha (peça móvel da balestilha) deslizante, fazendo-a coincidir com a linha horizontal. A Figura 05 mostra um exemplo do instrumento Balestilha (CAMÕES, 2002).

Figura 5 - Balestilha



Fonte: (LEANDRO, 2015).

2.5 Sistema de Posicionamento Global (GPS)

Como apresentado na figura 06, o GPS consiste em três partes, chamadas de parte espacial, parte de controle e parte do usuário. (MENDES, 2013).

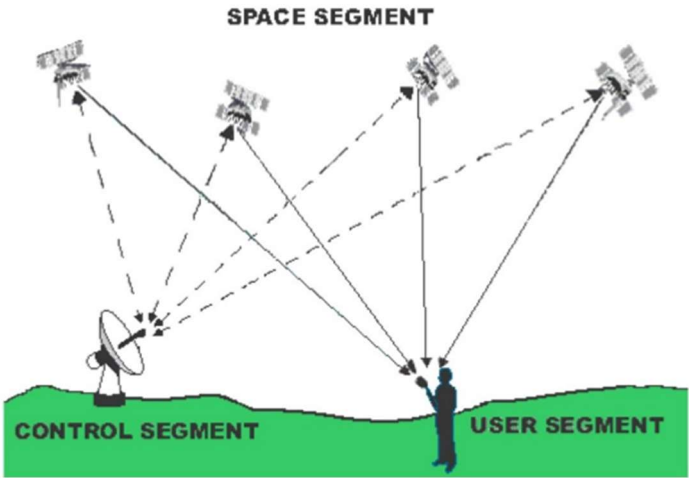
O segmento espacial consiste em uma constelação de 24 satélites distribuídos por 6 planos orbitais (4 satélites por plano). Os 6 planos orbitais estão separados por 60 graus de longitude e têm 55° de inclinação em relação ao plano equatorial da Terra (A inclinação dos satélites da parte A era de 63°). Os satélites são colocados a 20200 km acima da superfície da Terra, para que fiquem visíveis em um determinado local todos os dias por cerca de alguns minutos de antecedência. O posicionamento do satélite no plano orbital é feito de modo que pelo menos 4 satélites possam ser observados da Terra a qualquer hora (MENDES, 2013).

O segmento de controle possui as seguintes funções principais: sincronização dos relógios dos satélites com a hora do GPS; calcular as órbitas dos satélites; injetar essas informações nos satélites, para distribuí-las aos usuários e monitorar o status operacional dos satélites. O segmento de controle consiste em uma estação

principal e cinco estações de monitoramento, três das quais são responsáveis pela comunicação com os satélites (MENDES, 2013).

O segmento de usuário consiste nos receptores GPS, que decodificam e processam os sinais enviados pelos satélites, calculando a posição, velocidade e tempo do usuário. Existe uma grande variedade de antenas e receptores dependendo do tipo de aplicação a que se destinam (MENDES, 2013).

Figura 6- Segmentos do GPS



Fonte: (FERNANDES, 2022).

2.5.1 Sinal do GPS

O sinal GPS é composto por vários elementos: ondas portadoras, códigos e dados (MENDES, 2013).

As ondas portadoras, nas quais os códigos são modulados, são denominadas L1 e L2, com frequências 1575,42 MHz (Mega-Hertz) e 1227,60 MHz, respectivamente. Ambas as frequências são derivadas de uma frequência fundamental de MHz. Para comunicações com estações de monitoramento e dados, ainda são utilizadas as frequências 1783,74 MHz e 2227,5 MHz (MENDES, 2013).

Outra frequência, L3, de 1381,05 MHz, é usada para fins militares para explosões nucleares (MENDES, 2013).

Os códigos possuem características de ruído pseudoaleatório, ou seja, são sequências de zeros e de uns, que possuem características aleatórias, mas que podem ser identificadas como ambiguidade pelo receptor (MENDES, 2013).

2.5.2 Código de aquisição livre ou C/A

É um código PRN (ruído pseudo aleatório) de 1023 bits, gerado a uma taxa de 1023 MHz com um período de 1 ms (milissegundo) (MENDES, 2013).

O comprimento curto deste código permite ao receptor sintonizar rapidamente os diferentes satélites e facilitar a transição para a aquisição do código P (*Precision Code*) (MENDES, 2013).

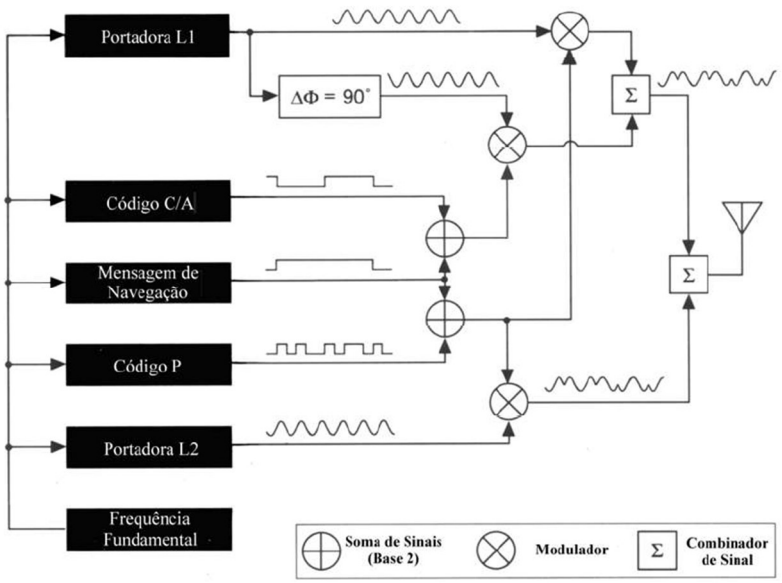
Cada satélite GPS possui seu próprio código C/A (*Coarse Acquisition Code*), selecionado a partir de um conjunto de códigos denominado *gold*. Esta série de códigos é projetada para minimizar o erro, por parte do receptor, na identificação do satélite. O código C/A só é transmitido em L1. Como apresentado na Figura 06) (MENDES, 2013).

2.5.3 Código preciso ou protegido, ou simplesmente P

Código preciso é um código PRN de 2,34x1014 bits, que gera à taxa de 10.23 MHz, com um período de 267 dias. Este código é dividido em segmentos de 7dias, são atribuídos a diferentes satélites, para que cada um possa ser identificado pelo seu código PRN (MENDES, 2013).

Usando uma técnica de multiplexação por divisão de código, torna possível que todos os satélites compartilhem a mesma frequência. O código P (*Precision Code*) é transmitido em L1 e L2. Como apresentado na Figura 07 (MENDES, 2013).

Figura 7- Estrutura do sinal GPS



Fonte: (MENDES, 2013)

2.6 Funcionamento do GPS

Um satélite é rastreado por um receptor para a recepção de seu sinal. Sinais de apenas quatro satélites são necessários para uma posição tridimensional fixa, no entanto, é desejável um receptor que realize o rastreamento dos quatro satélites simultaneamente (TAGLIANI, 2022)

No caso de o usuário estar se movimentando, o sinal de um satélite pode ser repentinamente bloqueado por algum obstáculo, mesmo assim deixando satélites suficientes para guiá-lo. A maioria dos receptores rastreia de 8 a 12 satélites ao mesmo tempo. (TAGLIANI, 2022)

Uma vez que os sinais são captados pela antena, eles são enviados para um circuito eletrônico chamado canal, que pode identificar sinais de diferentes satélites (TAGLIANI, 2022).

Um receptor com um canal lê continuamente o sinal de cada satélite até receber sinais de todos os satélites rastreados. Esta técnica é chamada de "multiplexação por divisão de tempo". Processar dados e calcular a localização leva menos de um segundo (TAGLIANI, 2022).

2.6.1 Satélites

A Terra está rodeada por 21 satélites principais (mais 3 satélites ativos em repouso) em 6 órbitas (DORE, 2019).

2.6.2 Triangulação

O sistema GPS é baseado na medição da distância entre o receptor e pelo menos 3 satélites (DORE, 2019).

a) O receptor mede a distância até o primeiro satélite. Constrói um círculo imaginário com essa distância como o raio. Até o momento a localização não e precisa (DORE, 2019).

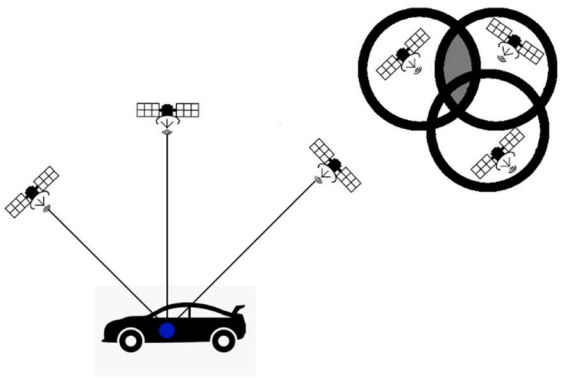
b) O receptor agora toma a distância que o separa do satélite 2 e forma um novo círculo. A localização do objeto já se encontra dentro da interseção criada pelos dois satélites (DORE, 2019).

c) O terceiro satélite cria uma nova esfera que corta a interseção dos outros dois satélites em dois pontos. A localização real está em um ponto, e a outra localização é excluída como uma localização improvável porque está longe da superfície da Terra (DORE, 2019).

2.6.3 Processamento

O receptor recebe as seguintes informações: o que os três satélites capturaram e suas características orbitais. Ele processa todos esses dados e obtém a localização exata do carro. Como apresentado na Figura 8 (DORE, 2019).

Figura 8 - Funcionamento do GPS



Fonte: Elaborada pelo autor

2.7 Geofencing

Como apresentado na figura 09 *geofencing* é uma maneira de monitorar objetos em uma área específica. O conceito básico é que para um conjunto de *geofences* específicas da região e rastreado a localização do usuário, é acionado um evento ao entrar ou sair de qualquer área. A área do *geofencing* pode ter qualquer formato, mas o mais utilizado e a forma circular. O *geofencing* é tão preciso quanto a precisão da localização fornecida pelos sensores em uso, seja integrado a um dispositivo móvel ou conectado externamente. Geralmente são utilizados os receptores GPS, mas para essa finalidade, rede *Wi-Fi* e identificação por rádio frequência (RFID) também atende (WAWRZYNIAK; HYLA, 2016)

Figura 9 - *Geofencing*

Fonte: (ACTOMIC, 2019)

Geofencing é uma análise espacial que verifica se a localização atual está dentro ou fora de uma área. Ele é otimizado e implementado no sistema operacional. Cerca de cem *geofences* podem ser monitorados por vez no *Android* e milhares no *Windows* sem degradação significativa do desempenho. Na prática, 100 são suficientes, pois nem todas as *geofences* disponíveis devem ser monitoradas a qualquer momento. Monitora-se apenas o conjunto de objetos de *geofence* mais próximos do local determinado. As *geofences* monitoradas podem ser alteradas dinamicamente (WAWRZYNIAK; HYLA, 2016).

3 Banco de dados

Um sistema de banco de dados é basicamente um sistema informatizado de manutenção de registros, ou seja, é um repositório ou container para uma coleção de arquivos de dados computadorizados. Os usuários do sistema podem realizar (ou melhor, solicitar que o sistema execute) diversas operações envolvendo tais arquivos, como: adicionar, inserir, pesquisar, alterar e excluir informações (DATE, 2004).

A informação em questão pode ser qualquer significância que ressalte aos sentidos do indivíduo ou organização a que o sistema se destina, ou seja, qualquer coisa que contribua para o processo geral das atividades do indivíduo ou da organização (DATE, 2004).

3.1 Banco de dados relacional

O *Model*o relacional (MR) proposto pelo matemático Ted Codd e pesquisadores da IBM (Máquinas de Negócios Internacionais) em 1970, é baseado em conceitos matemáticos, teoria dos conjuntos e lógica de predicados. Os primeiros sistemas comerciais baseados no *Model*o relacional foram introduzidos em 1980 e foram implementados em muitos sistemas como *Access*, *Oracle*, MySql etc. (SOUZA, 2017).

A estrutura do banco de dados relacional, é chamada também de tabela, onde os dados são organizados por um ou mais atributos que utilizam metadados para converter o tipo de dado que é armazenado (SOUZA, 2017).

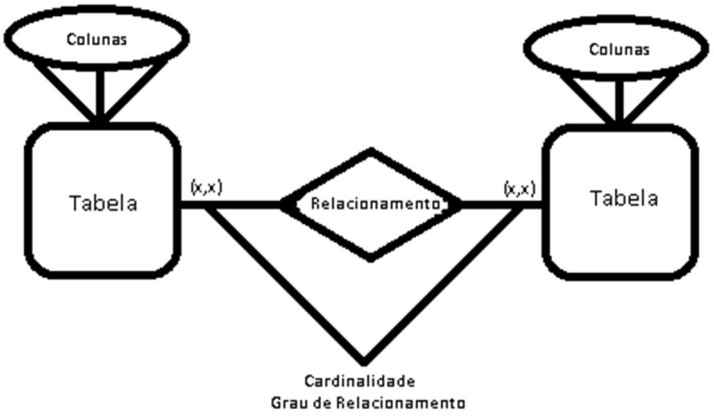
Por sua vez, os conjuntos de dados são organizados em registros (linhas ou colunas). O *Model*o relacional descreve um banco de dados como uma coleção de relacionamentos. Uma tabela é uma coleção dessas relações (SOUZA, 2017).

Como apresentado na figura 10 os relacionamentos são representados por losangos e linhas que se conectam entre eles (DATE, 2004).

Outra característica importante deste *Model*o é o uso de restrições de integridade, que utilizam chaves primárias e estrangeiras para garantir a integridade dos dados (SOUZA, 2017).

As chaves primárias permitem a identificação única de registros em um relacionamento e melhoram o desempenho da consulta, enquanto as chaves estrangeiras, por sua vez, permitem que os usuários realizem relacionamentos entre diferentes tabelas (SOUZA, 2017).

Figura 10 - *Model*o de banco de dados relacional



Fonte: Elaborada pelo autor

3.2 Banco de dados não relacional

Um banco de dados não relacional é uma abordagem não estruturada que valoriza desempenho, escalabilidade e disponibilidade. O *NoSQL (Not only Structured Query Language)* possui características importantes que diferem dos bancos de dados relacionais. Dessa forma, pode atender os requisitos de gerenciamento de grandes volumes de dados semiestruturados ou não estruturados sem utilizar linguagens nativas como *Structured Query Language* (SQL). Algumas das características que diferenciam o *NoSQL* dos bancos de dados relacionais são: escalabilidade horizontal, esquemas livres de esquema ou flexíveis, suporte a

replicação nativa e interfaces de programação de aplicativos (APIs) simples para acessar dados e consistência eventual (MEDEIROS, 2022).

Ainda assim, a indústria categoriza os bancos de dados *NoSQL* em pequenos conjuntos de áreas funcionais. Algumas dessas categorias são: bancos de dados orientados a chave/valor, bancos de dados orientados a documentos, bancos de dados orientados a gráficos e bancos de dados orientados a famílias de colunas (MEDEIROS, 2022).

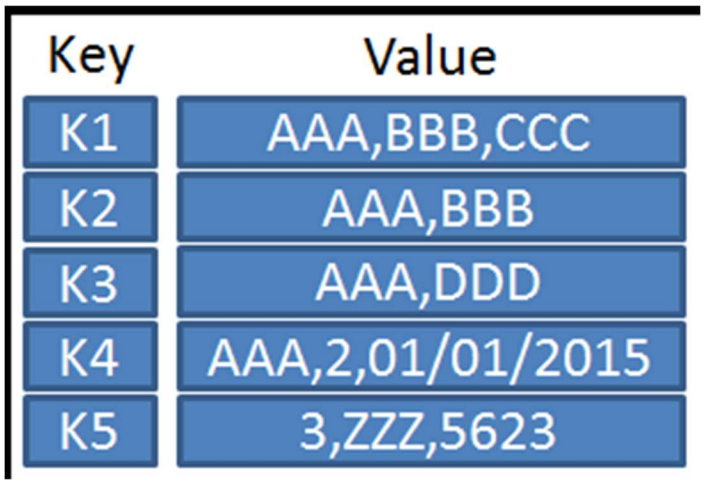
• Como apresentado na figura 11, banco de dados orientado a chave/valor: Esta abordagem implementa um mecanismo de armazenamento *NoSQL* mais simples. É considerada uma grande tabela de *hash*, ou seja, uma estrutura de dados simples que pode suportar um conjunto de pares chave/valor (MEDEIROS, 2022).

• Como apresentado na figura 12, bancos de dados orientados a documentos: semelhantes em conceito aos bancos de dados orientados a chave/valor, exceto que os dados são armazenados em documentos. Um documento é uma coleção de campos nomeados e valores associados.

• Como apresentado na figura 13, bancos de dados orientados a gráficos: armazenam entidades, mas concentram-se principalmente nos relacionamentos entre essas entidades. Ele armazena dois tipos de informações: nós (instâncias de entidades) e arestas (especificando relacionamentos entre nós).

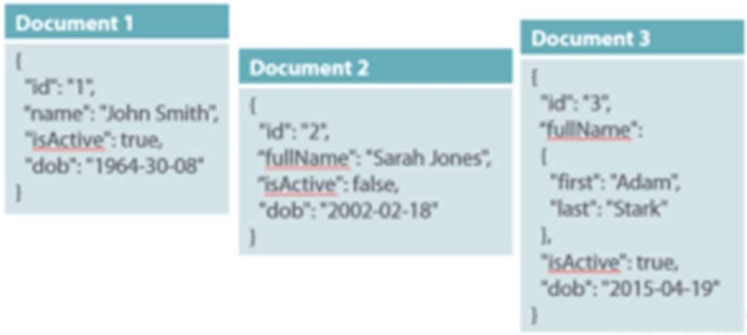
• Como apresentado na figura 14, banco de dados orientado à família de colunas: organiza os dados em linhas e colunas que se parecem muito com bancos de dados relacionais, pelo menos em conceito.

Figura 11- Orientado a chave/valor



Fonte: (MONTEIRO, 2017).

Figura 12 - Orientado a Documentos

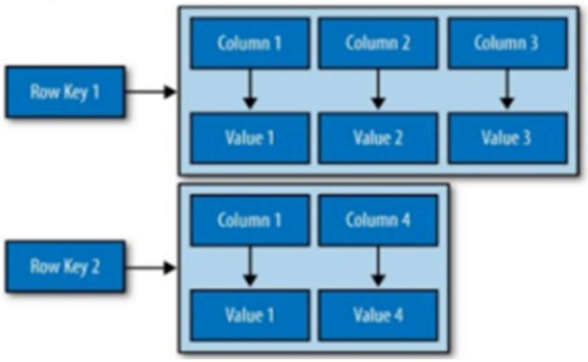


Fonte: (MONTEIRO, 2017).

Figura 13 - Orientado a gráficos

Fonte: (MONTEIRO, 2017).

Figura 14 - Orientado a família de colunas



Fonte: (MONTEIRO, 2017).

3.3 Visual *Studio Community*

O Visual *Studio* é um ambiente de programação baseado em GUI (*Graphical User Interface*) Microsoft.Net, que inclui um configurador multilíngue como C++ ou C#, F#, Ironphyton e IronRuby e Visual *Basic*.Net. No mundo, este programa é o mais utilizado para programas de desktop, pois pode entrar em uma biblioteca de links dinâmicos (DLL) como funcionalidade adicional. É um software classificado como um provedor de linguagem de programação, por isso é flexível de usar. Existem muitas versões atualizadas do Visual *Studio* (PRASTYO, 2019).

3.4 Android Studio

Criado em 2013, o *Android Studio* foi desenvolvido para ser uma ferramenta para desenvolvimento *Android* (HAGOS, 2020).

O *Android Studio* é baseado no IntelliJ da JetBrains, que possui a versão comercial e a paga (ou comunitária). A versão comunitária serviu como base para o *Android Studio* (HAGOS, 2020).

É fornecido pelo *Android Studio* ferramentas para aumentar a produtividade na criação de aplicativos, como: sistema de compilação flexível baseado em *Gradle*, emulador rápido com inúmeros recursos, ambiente unificado que possibilita o desenvolvimento para todos os dispositivos *Android*, enviar mudanças de código e recursos ao aplicativo em execução sem reiniciar o app (*Application*), *Model*os de código e integração com GitHub para ajudar a criar recursos comuns de apps e importar exemplos de código, *Frameworks* e ferramentas de teste cheios de

possibilidades, ferramentas de lint para detectar problemas de desempenho, usabilidade, compatibilidade com versões, entre outros, Compatibilidade com C++ e NDK, Compatibilidade integrada com o Google *Cloud Platform*, facilitando a integração do Google *Cloud Messaging* e do App *Engine* (DEVELOPERS, 2021).

A flexibilidade do *Gradle* permite que se faça tudo isso sem modificar os principais arquivos de origem do seu aplicativo. O arquivo de compilação do *Android Studio* é denominado *build.gradle*. Eles são arquivos de texto simples que usam a sintaxe *Groovy* para configurar a compilação usando elementos fornecidos pelo *Android Plugin for Gradle*. Cada projeto tem um arquivo de compilação de nível superior para todo o projeto e um arquivo de compilação de módulo separado para cada módulo. Quando se importa um projeto existente, o *Android Studio* gera automaticamente os arquivos de compilação necessários (DEVELOPERS, 2021).

3.5 Linguagem de Programação

Já havia relatos de Linguagens de Programação em 1940, quando as primeiras linguagens de programação e computadores modernos começaram a aparecer. A primeira linguagem de programação foi o código matemático. A ideia de uma linguagem de programação é o código específico do aplicativo. As linguagens de programação decorrem da evolução da lógica matemática, onde conceitos complexos são abstraídos da matemática e podem ser usados para resolver problemas específicos. Dois conceitos importantes em linguagens de programação são: sintaxe e semântica, podendo ser feita uma analogia com uma língua natural como o português (BERTOLINI et al, 2019).

Compilador e interpretador são as duas maneiras para converter um código de alto nível em um código que o computador possa entender e executar. A partir da tradução de um código de alto nível e gerado (BERTOLINI et al, 2019).

A tradução de um programa escrito em uma linguagem de alto nível produz um novo código chamado código objeto. O compilador converte todo o código fonte de alto nível em código objeto. Por outro lado, o interpretador traduz o código linha por linha à medida que é executado (BERTOLINI et al, 2019).

3.6 C#

No final dos anos 90, a Microsoft tinha várias linguagens para resolver diferentes tipos de problemas, porém, toda vez que um desenvolvedor mudava a linguagem para resolver tal problema, era necessário conhecer as especificações da nova linguagem, também precisava adaptar APIs e bibliotecas. Para minimizar esse problema de adaptação, foi desenvolvido o *dotNET*, uma plataforma da Microsoft na qual se baseiam todas as linguagens oferecidas pela empresa. Portanto, para um profissional mudar a linguagem, basta conhecer sua especificação, padronizar a estrutura de todas as bibliotecas e APIs. Assim, a linguagem C# foi introduzida em 2002 como parte de um projeto voltado para o desenvolvimento de tecnologias orientadas a objetos leves e fáceis de aprender (DOS REIS, 2020).

A linguagem C# é orientada a objetos e a componentes. O C# fornece construções de linguagem para suportar conceitos diretamente, tornando à uma linguagem natural para construção e utilização dos mais diversos componentes de software. Desde o início, o C# adicionou recursos para dar suporte a novas cargas de trabalho e práticas emergentes em design de softwares (SANGLARD, 2022).

3.7 Kotlin

Kotlin é uma linguagem desenvolvida pela empresa JetBrains que roda na Java Virtual *Machine* (TOLEDO, 2021).

A linguagem Kotlin visa corrigir alguns problemas da linguagem Java, assim permitindo o desenvolvimento de softwares mais seguros. Uma das novidades que a linguagem Kotlin trouxe é ser *null-safe*. Sendo assim o compilador da linguagem é capaz de detectar possíveis objetos nulos antes da execução do código, evitando possíveis erros em tempo de execução (TOLEDO, 2021).

Uma característica muito importante do Kotlin é a maneira como ele lida com objetos vazios. Em programas Java, é comum encontrar muitos códigos para poder resolver referências nulas a objetos. Em Kotlin, isso é contornado informando quando um objeto pode ter uma referência nula. Isso permite que o compilador da linguagem detecte locais onde podem ocorrer erros de acesso a objetos nulos em tempo de execução. Objetos que podem ter referências nulas são marcados com o

operador de chamada segura (representado por um ponto de interrogação "?") (TOLEDO, 2021).

3.8 SQL

SQL é uma linguagem de uso geral que permite que os usuários recuperem, armazenem, modifiquem e excluam dados, criem, modifiquem e excluam objetos de banco de dados (como tabelas, colunas, procedimentos, usuários), concedam e revoguem permissões de usuários e agrupem instruções em transações. Os comandos SQL são instruções capazes de recuperar informações de um banco de dados, denominadas de consultas. As instruções SQL consistem em cláusulas (como *SELECT*, *FROM*, *WHERE*), que contêm principalmente nomes de objetos de banco de dados; predicados (como *LIKE*, *BETWEEN*, *EXISTS*); operadores (como *AND*, *OR*, *NOT*); quantificadores (como *ANY*, *ALL*, *UNION*), e funções (como, *COUNT*, *SUM*, *AVG*). Esses conceitos são chamados quando não há necessidade de distinguir, por exemplo, cláusulas e predicados (TAIPALUS; SEPPÄNEN, 2020).

SQL é geralmente dividido em pelo menos duas sublinguagens, linguagem de manipulação de dados e linguagem de dados. Além disso, como as revisões do SQL introduziram mais funcionalidades, duas sublinguagens adicionais, linguagem de controle de dados e linguagem de transação, são algumas vezes discutidas na literatura (TAIPALUS; SEPPÄNEN, 2020).

As origens dos nomes das sublinguagens DCL (*Data Control Language*) e TCL (*Transaction Control Language*) não são claras, porque os nomes não são explicitamente mencionados no padrão SQL.

No entanto, chama-se essa divisão em quatro sublinguagens bastante intuitivas, e utiliza-se neste estudo.

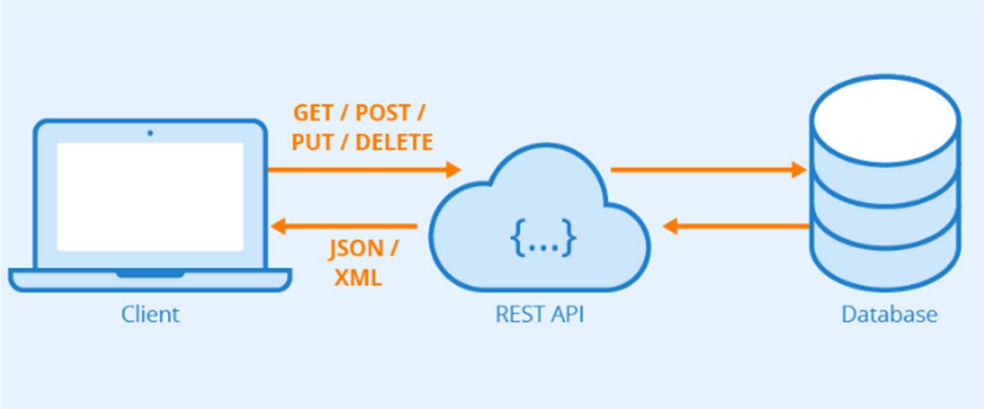
3.9 API

API, abreviação de *Application Programming Interface*, é um serviço para uso por outros aplicativos que fornece dados processados e funcionalidade de maneira transparente, abstraindo detalhes relacionados à implementação. A API é responsável por receber solicitações de outros aplicativos e retornar os resultados dessas solicitações (NETO, 2021).

Quando usada em um ambiente web, uma API define uma especificação de como uma solicitação e sua estrutura de resposta devem ser recebidas pelo protocolo HTTP (*Hyper Text Transfer Protocol*), geralmente no formato JSON (JavaScript *Object Notation*) ou XML (*Extensible Markup Language*), como apresentado na figura 15. As APIs permitem que os aplicativos cliente executem suas funções por meio de *endpoints*, que são URLs (*Uniform Resource Locator*) definidos pela API para acessar os serviços disponíveis.

A comunicação entre a API e o aplicativo cliente ocorre por meio de solicitações do aplicativo cliente para o *endpoint* da API usando o protocolo HTTP. Os principais métodos do protocolo HTTP usados para se comunicar com APIs da Web são HTTP *GET* e HTTP *POST*; ambos retornam resultados para o aplicativo solicitante, mas somente o método HTTP *POST* envia informações para a API.

Figura 15 - API



Fonte: (NAEEM, 2020).

4 Proposta de Solução

Este capítulo tem por finalidade apresentar a proposta de solução para a implementação a ser realizada no TCC2 (Trabalho de Conclusão de Curso para o segundo semestre).

4.1 Xamarin Forms

A partir de uma base de código compartilhada o Xamarin Forms permite a criação de aplicativos Xamarin Android, Xamarin IOS e Windows, Xamarin Forms é uma estrutura de interface do usuário de software livre (PROFEXORGEEK, 2022).

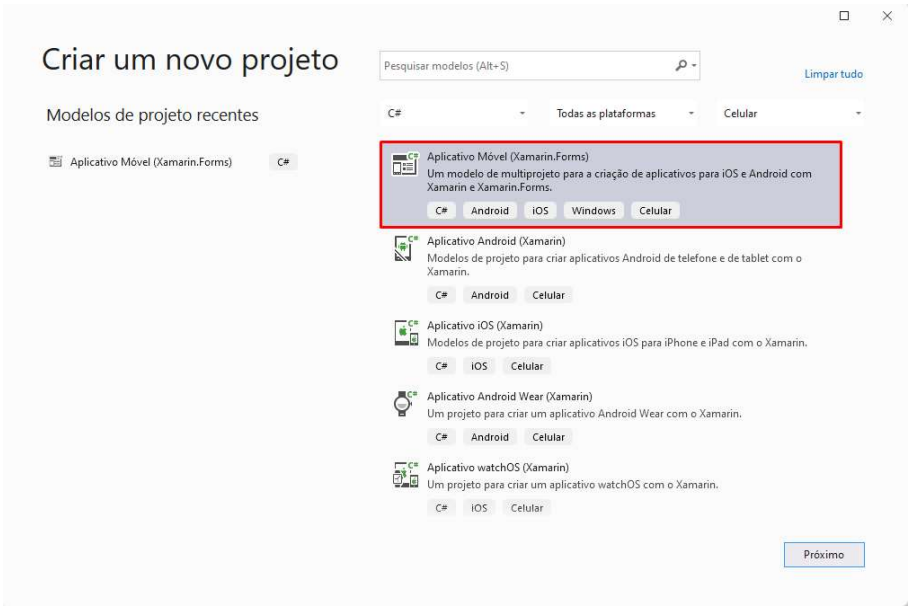
Xamarin Forms faz com que os desenvolvedores possam criar interfaces de usuário no XAML com código por trás em C#. Essas interfaces são renderizadas como controles nativos de alto desempenho em cada plataforma (PROFEXORGEEK, 2022).

A API fornecida pelo Xamarin Forms pode ser implementada em XAML ou C# para criar elementos de interface do usuário, sendo compatível com *DataBinding* para padrões como MVVM(*Model*-View-View*Model*) (PROFEXORGEEK, 2022).

Em tempo de execução o Xamarin Forms converte os elementos de interface do usuário multiplataforma em controles nativos no Xamarin Android, Xamarin IOS e UWP (Plataforma Universal do Windows), permitindo com que os desenvolvedores obtenham aparência, sensações e desempenho nativos, enquanto usufruem dos benefícios do compartilhamento de código entre plataformas (PROFEXORGEEK, 2022).

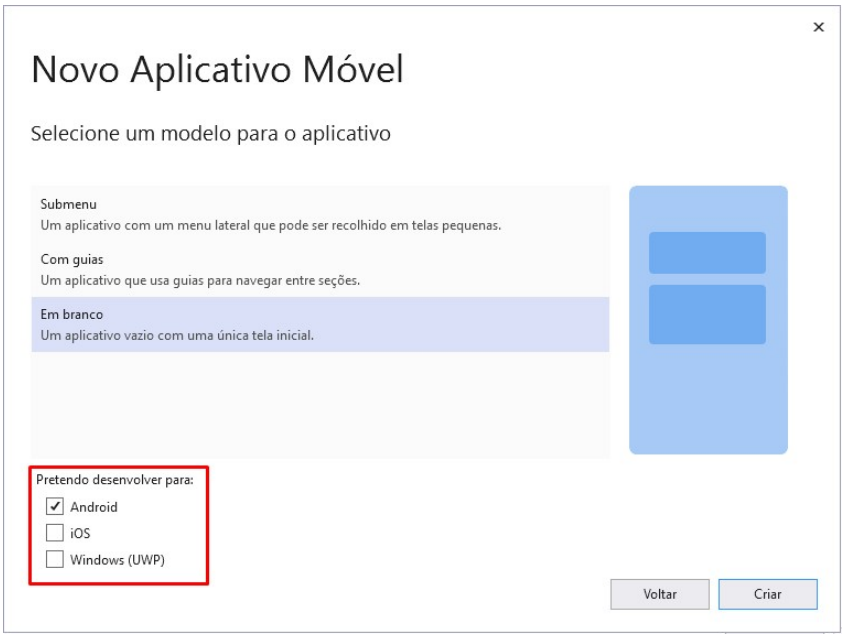
A Figura 16 e a Figura 17 mostram a criação de um projeto Xamarin Forms, e a possibilidade de desenvolver para Android, IOS e UWP utilizando a mesma solução.

Figura 16 – Como criar um Projeto Xamarin Forms



Fonte: Elaborada pelo Autor

Figura 17 - Plataformas do Xamarin Forms



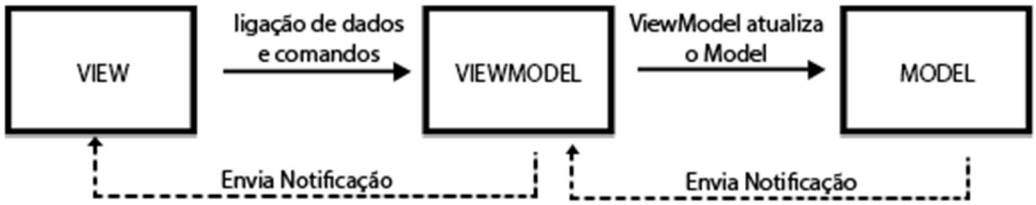
Fonte: Elaborada pelo autor

4.2 *Model*o MVVM

O padrão MVVM (*Model*-View-View*Model*) ajuda a separar claramente a lógica de negócios e a apresentação de um aplicativo de sua interface de usuário. Manter uma separação clara entre a lógica do aplicativo e a interface do usuário ajuda a resolver muitos problemas de desenvolvimento e facilita o teste, a manutenção e a evolução dos aplicativos. O modelo MVVM também melhora muito as oportunidades de reutilização ou reuso de código e permite que desenvolvedores e designers de interface do usuário colaborem com mais facilidade ao desenvolver suas respectivas partes do aplicativo. (DAVIDBRITCH, 2022).

Como apresentado na figura 18, o padrão MVVM consiste em três componentes principais: *Model*, View e View*Model*. Cada um tem um propósito diferente. (DAVIDBRITCH, 2022).

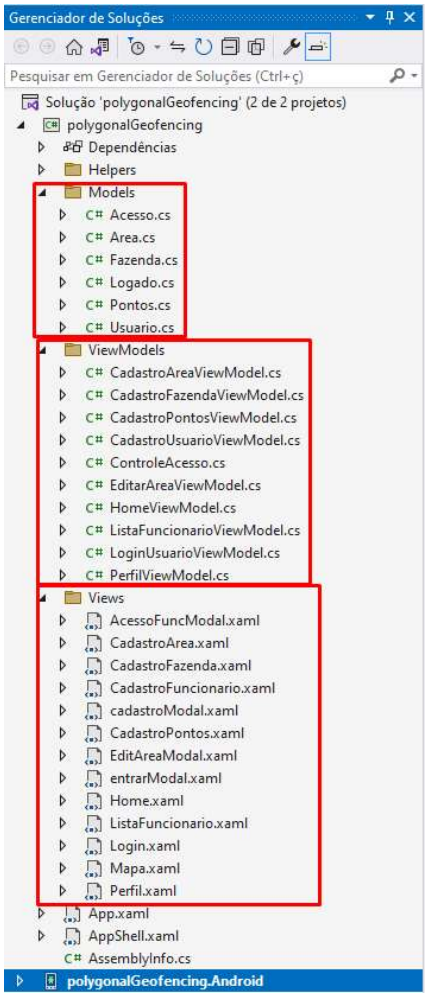
Figura 18 - Estrutura MVVM



Fonte: Elaborada pelo autor

- *View*: Responsável por definir a aparência ou estrutura que é apresentada para o usuário. A *View* se comunica com a *ViewlModel*. (DEVMEDIA, 2010).
- *ViewModel*: Responsável por disponibilizar para a *View* uma lógica de apresentação. A *ViewModel* é quem vai coordenar as interações da *View* com a *Model*. A *ViewModel* pode ser usada para implementar a lógica de validação para garantir a consistência de dados. (DEVMEDIA, 2010).
- *Model*: Responsável por encapsular a lógica de negócio e os dados. Armazena as classes de negócios que serão utilizadas em uma determinada aplicação. (DEVMEDIA, 2010).

Figura 19 - Arquivos na Estrutura MVVM



Fonte: Elaborada pelo Autor

4.3 NuGet

É um mecanismo através do qual os desenvolvedores podem criar, compartilhar e usar código útil. Esse código geralmente é fornecido em um "pacote" que contém código compilado, como uma DLL. (JONDOUGLAS, 2022).

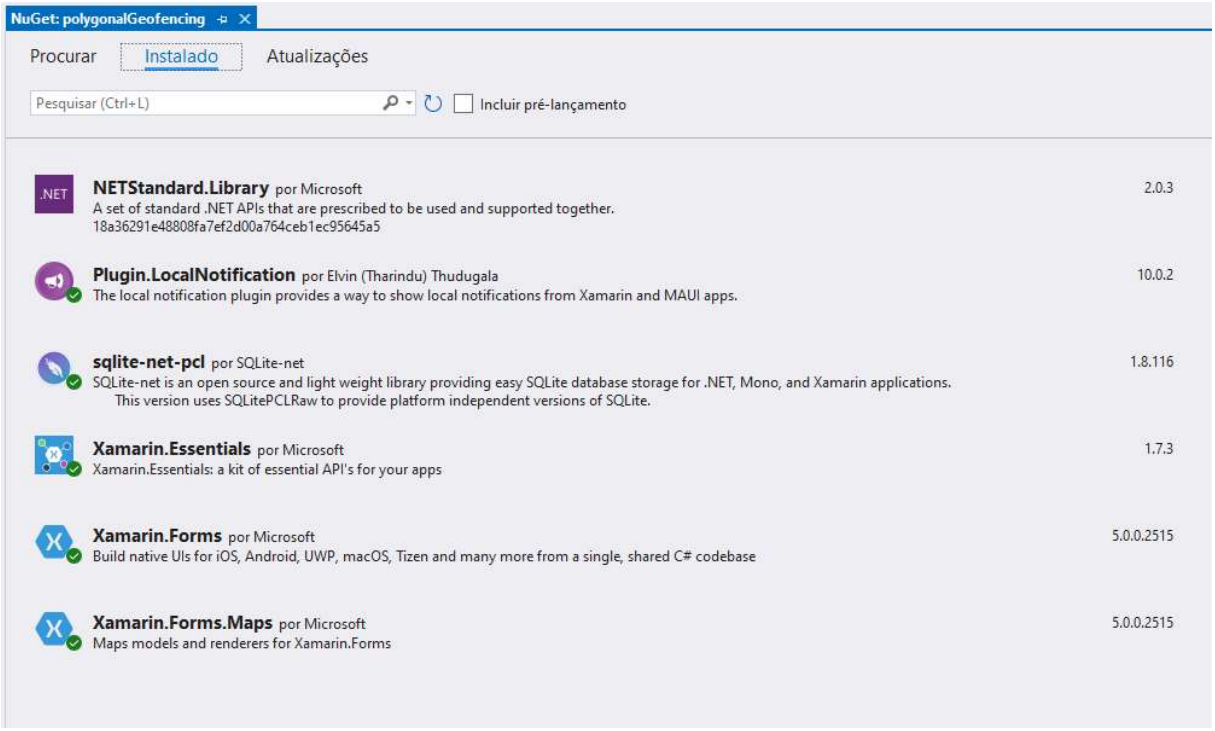
Para .NET, o mecanismo de compartilhamento de código com suporte da Microsoft é o NuGet, que define como hospedar e consumir pacotes .NET. (JONDOUGLAS, 2022).

Resumindo, um pacote NuGet é um arquivo ZIP com uma extensão nupkg que contém código compilado. (JONDOUGLAS, 2022).

Desenvolvedores com código compartilhado criam pacotes e os publicam em hosts públicos ou privados. Os consumidores de pacotes obtêm esses pacotes do

host apropriado, os adicionam ao projeto e invocam as funções do pacote no código do projeto. (JONDOUGLAS et al., 2022).

Figura 20 - Interface pacotes do NuGet



Fonte: Elaborada pelo Autor

4.4 Xamarin Essentials

Xamarin Essentials é uma biblioteca que fornece uma API multiplataforma para recursos de dispositivos nativos. Assim como o próprio Xamarin, o Xamarin Essentials é uma abstração que simplifica o acesso a utilitários nativos (JAMESMONTEMAGNO, 2022).

Xamarin Essentials abstrai o maior número possível de permissões. No entanto, cada sistema operacional possui um conjunto diferente de permissões. Além disso, existem diferenças no fornecimento de API exclusiva para algumas permissões (JAMESMONTEMAGNO et al.,2022).

A Figura 21 mostra algumas permissões atualmente disponíveis

Figura 21 - Suporte de permissões em diferentes plataformas

- ✓ – Com suporte
- ✗ - Não há suporte/necessidade

Permissão	Android	iOS	UWP	watchOS	tvOS	Tizen
CalendarRead	✓	✓	✗	✓	✗	✗
CalendarWrite	✓	✓	✗	✓	✗	✗
Câmera	✓	✓	✗	✗	✗	✓
ContactsRead	✓	✓	✓	✗	✗	✗
ContactsWrite	✓	✓	✓	✗	✗	✗
Flashlight	✓	✗	✗	✗	✗	✓
LocationWhenInUse	✓	✓	✓	✓	✓	✓
LocationAlways	✓	✓	✓	✓	✗	✓
Mídia	✗	✓	✗	✗	✗	✗
Microfone	✓	✓	✓	✗	✗	✓
Telefone	✓	✓	✗	✗	✗	✗
Fotos	✗	✓	✗	✗	✓	✗
Lembretes	✗	✓	✗	✓	✗	✗
Sensores	✓	✓	✓	✓	✗	✗
Sms	✓	✓	✗	✗	✗	✗
Fala	✓	✓	✗	✗	✗	✗
StorageRead	✓	✗	✗	✗	✗	✗
StorageWrite	✓	✗	✗	✗	✗	✗

Se uma permissão for marcada como ✗ ela sempre retornará `Granted` quando marcada ou solicitada.

Fonte: (JAMESMONTEMAGNO et al, 2022)

4.5 Xamarin Forms Permissões

Os aplicativos Android são executados em sua própria *sandbox* e, por motivos de segurança, não podem acessar determinados recursos do sistema ou hardware no dispositivo. Os usuários devem conceder explicitamente permissões de aplicativo para usar esses recursos. Por exemplo, os aplicativos não podem acessar o GPS em um dispositivo sem permissão explícita do usuário. O Android lança uma *Java.Lang.Security Exception* se um aplicativo tentar acessar um recurso protegido, sem permissão. As permissões são declaradas em AndroidManifest.xml pelo desenvolvedor do aplicativo. O Android tem dois fluxos de trabalho diferentes para obter o consentimento do usuário para essas permissões: (DAVIDORTINAU, 2022)

Para aplicativos direcionados ao Android 5.1 (API 22) ou inferior, a solicitação de permissão ocorre quando o aplicativo é instalado. Se o usuário não conceder permissão, o aplicativo não é instalado. Depois que um aplicativo é instalado, as permissões não podem ser revogadas, exceto pela desinstalação do aplicativo. A partir do Android 6.0 (API 23), os usuários ganham mais controle sobre as permissões; eles podem conceder ou revogar permissões desde que o aplicativo esteja instalado no dispositivo (DAVIDORTINAU, 2022).

Os aplicativos Android devem verificar em tempo de execução se eles têm acesso a recursos protegidos. Se o aplicativo não tiver permissões, ele deverá fazer uma solicitação usando a nova API fornecida pelo Android SDK para que o usuário conceda a permissão. As permissões são divididas em duas categorias: (DAVIDORTINAU, 2022).

- Permissões normais: Estas são permissões que representam pouco risco para a segurança ou privacidade do usuário. O Android 6.0 concederá automaticamente permissões normais na instalação
- Permissões perigosas – ao contrário das permissões normais, as permissões perigosas são aquelas que protegem a segurança ou a privacidade do usuário. Eles devem ser concedidos explicitamente pelo usuário. Enviar ou receber uma mensagem SMS é uma ação perigosa que requer permissão

4.6 SQLite

SQLite é uma biblioteca que fornece uma ferramenta de banco de dados SQL transacional independente, ou seja, capaz reverter transações ou atividades de banco de dados se não for executado corretamente (DALCOMUNE, 2021), sem a necessidade de um servidor e de configurações iniciais para a implementação do banco de dados. (SQLITE, 2019).

O SQLite possui código de domínio público, portando, é gratuito para qualquer finalidade. (SQLITE, 2019)

Ao contrário da maioria dos bancos de dados que estão no mercado, o SQLite não possui um servidor separado, tornando-o um mecanismo de Banco de Dados SQL incorporado. SQLite lê e grava os dados diretamente em arquivos. Todas as tabelas e índices estão contidos em um único arquivo de disco. Possui uma

arquitetura de arquivo multiplataforma, sendo assim o arquivo gerado é compatível com sistemas 32 bits e 64 bits (SQLITE, 2019).

4.7 Google *Maps*

A Google disponibiliza uma API de mapeamento nativo para *Android.* Esta API é mais adequada quando necessário se ter mais controle sobre a experiência de mapeamento. Com essa API e possível: (DAVIDORTINAU et.al, 2022)

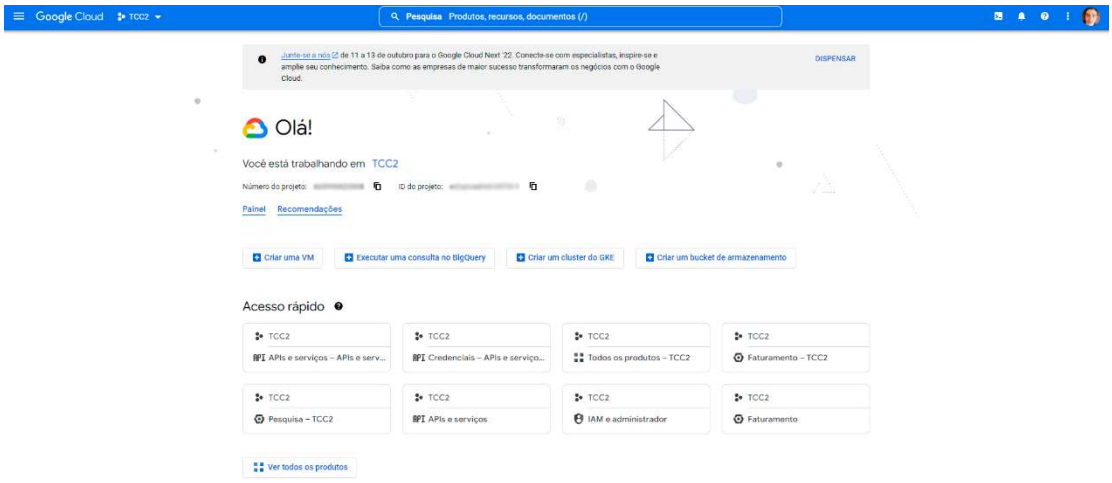
- Alterar o ponto de vista do mapa
- Adicionar marcadores personalizados
- Realizar anotações no mapa com sobreposições

Para a utilização desta API e necessário atender alguns pré-requisitos como:

- Obter a chave de API Mapas
- Instalar o pacote do Xamarin.Forms.Maps
- Especificar as permissões

A aquisição da chave de API Mapas funciona da seguinte forma: primeiro é preciso se ter um conta google. Com a conta google se tem acesso ao Google *Cloud Plataform,* nele se pode ter controle sobre as chaves APIs fornecidas pelo Google. A Figura 22 mostra a plataforma Google Cloud.

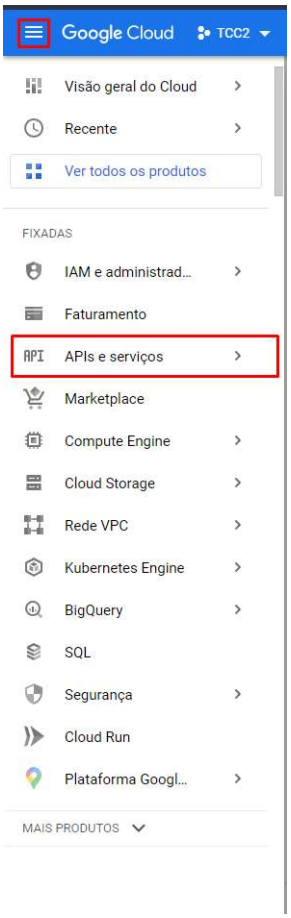
Figura 22 - Google Cloud Plataform



Fonte: Elaborada pelo autor

Como apresentado na Figura 23, ao acessar o menu do *Google Cloud Plataform* é apresentado vários serviços fornecidos pela google, entre eles o de API.

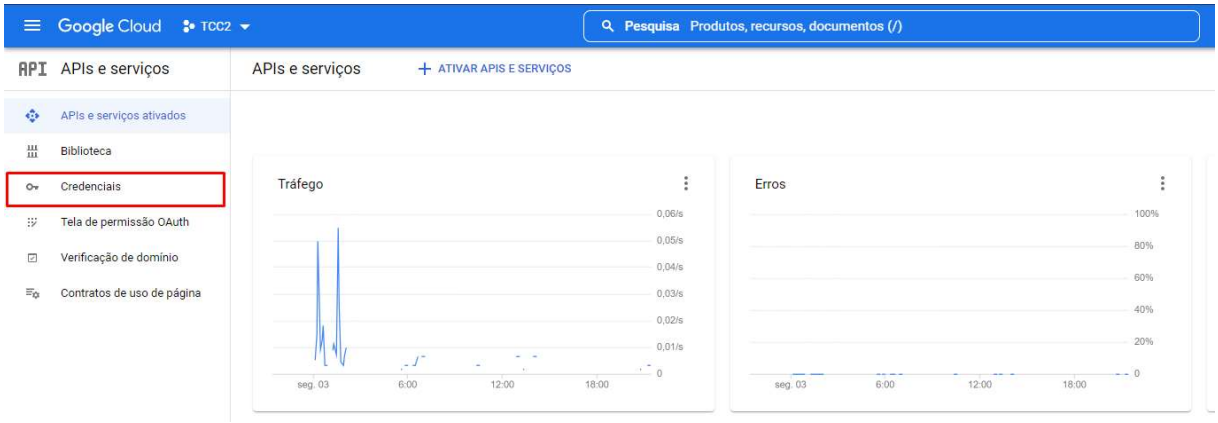
Figura 23 - Menu Google Cloud Plataform



Fonte: Elaborada pelo autor

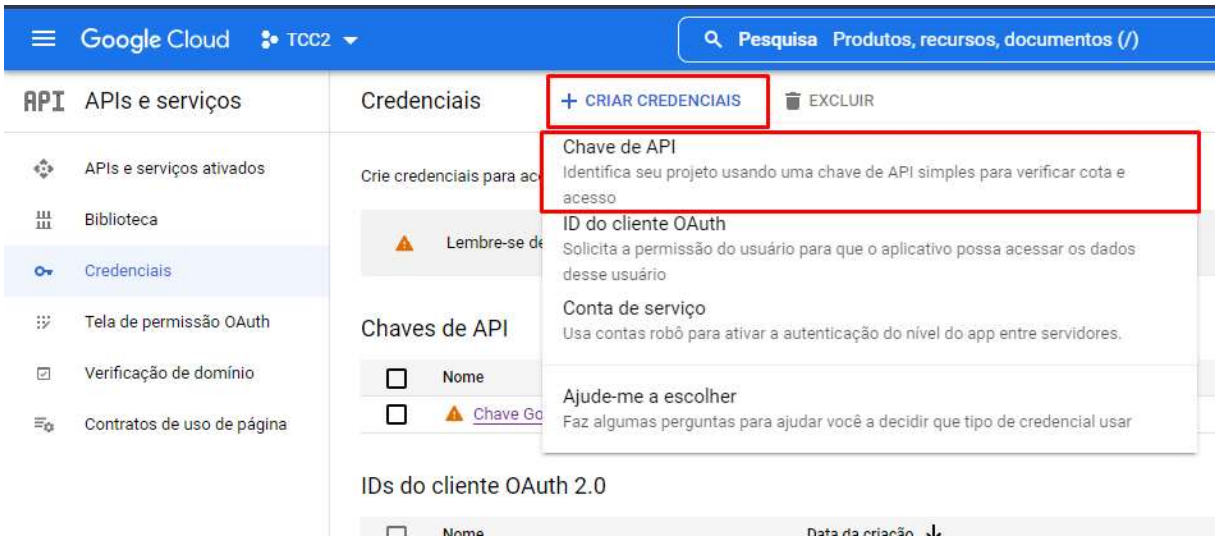
Acessando o serviço de APIs, pode-se entrar em uma página onde se tem todo o controle das APIs e acesso para criar novas credenciais, como apresentado nas figuras 24 e 25.

Figura 24 – Controle das APIs



Fonte: Elaborada pelo autor

Figura 25 - Criação da Chave API



Fonte: Elaborada pelo autor

5 Desenvolvimento da Aplicação

Este capítulo tem como objetivo mostrar o processo de criação da aplicação, expondo os procedimentos utilizados no código-fonte. O método de desenvolvimento é mostrando suas respectivas funcionalidades.

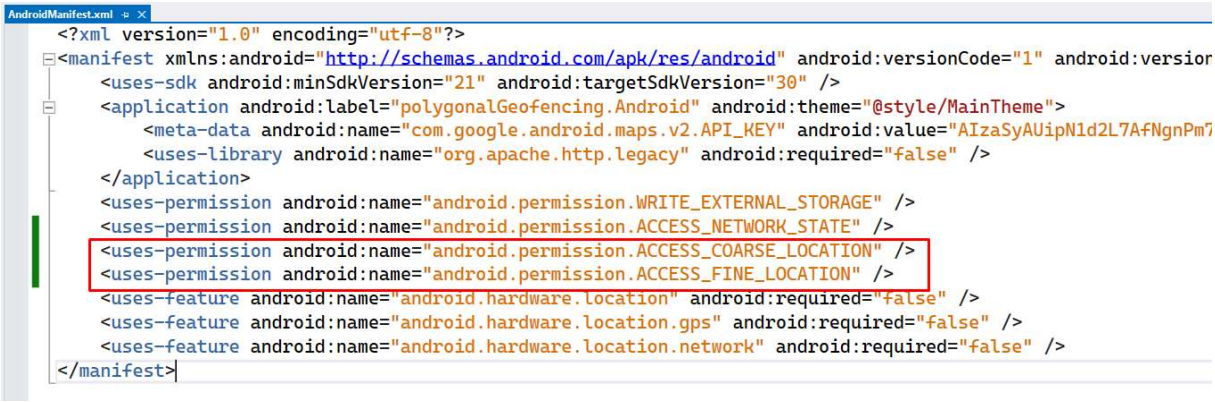
5.1 Implementação das permissões

Para se fazer o uso do aplicativo é solicitado ao usuário que o aplicativo acesse a sua localização, possibilitando-o executar suas funcionalidades.

O aplicativo solicitará permissão a localização aproximada (*ACCESS_COARSE_LOCATION*) e a localização precisa (*ACCESS_FINE_LOCATION*). (DEVELOPERS, 2022).

Os tipos de permissões deveram ser inseridos no arquivo AndroidManifest.xml, que se encontra na raiz do conjunto de origem do projeto. Todo projeto possui este arquivo e ele é responsável por descrever as informações essenciais sobre o aplicativo para as ferramentas de compilação do Android, como apresentado na Figura 26 (DEVELOPERS, 2022).

Figura 26 - Arquivo Android *Manifest*



Fonte: Elaborada pelo Autor

Ao inicializar o aplicativo, o arquivo *MainActivity.cs* é executado. Neste arquivo é feito o tratamento de algumas requisições, como o da localização.

Quando o aplicativo iniciar, a função *OnStart* é chamada, podendo ser feita qualquer verificação de permissão ou qualquer outra necessidade. Nesse caso foi

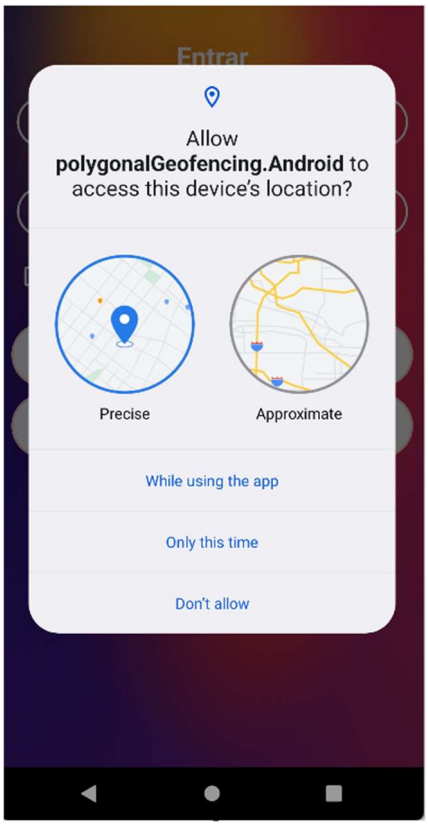
verificado se o usuário concedeu a permissão de localização, caso não tenha concedido é executada uma tratativa solicitando a permissão para acessar a localização, como apresentado nas figuras 27 e 28.

Figura 27 - Função ao inicializar o App

```
0 referências
protected override void OnStart() //ao inicializar
{
    base.OnStart();

    if ((int)Build.VERSION.SdkInt >= 23) //verifica se o sdk esta com a versão acima da 23
    {
        //verifica se a permissão de localização foi autorizada
        if (CheckSelfPermission(Manifest.Permission.AccessFineLocation) != Permission.Granted)
        {
            //solicita a permissão a ter acesso a localização
            RequestPermissions(LocationPermissions, RequestLocationId);
        }
        else
        {
            //Permissão Permitida
        }
    }
}
```

Fonte: Elaborada pelo autor

Figura 28 - Solicitação para acessar a localização

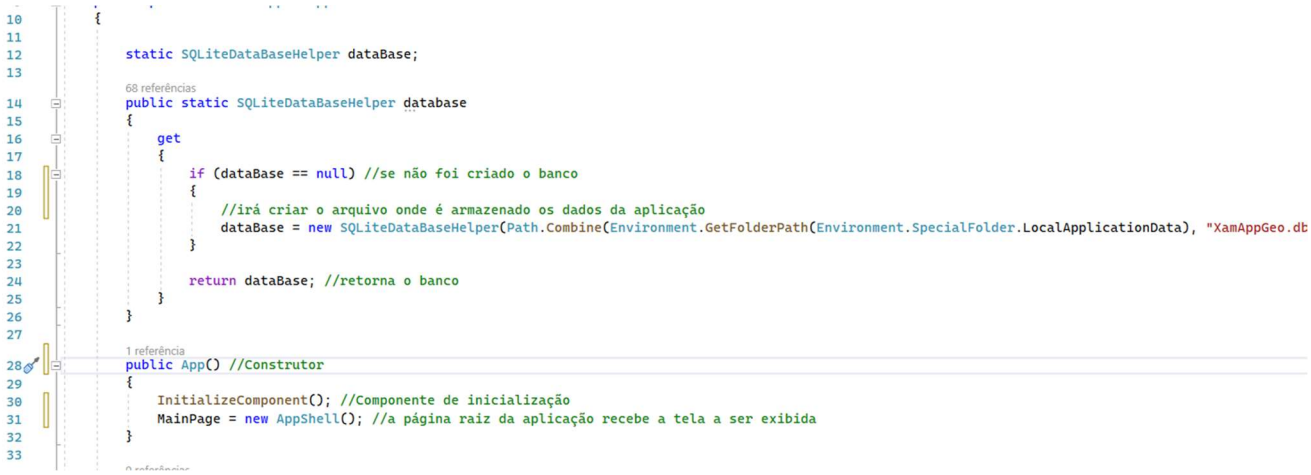


Fonte: Elaborada pelo autor

5.2 Inicialização da aplicação

Ao iniciar a aplicação, a página do App é chamada e a partir dela é criado o arquivo responsável pelo banco de dados local, SQLite. Além disso também é chamada a primeira tela a ser exibida, como apresentado na figura 29.

Figura 29 - Inicialização da Aplicação



```
10  {
11
12      static SQLiteDataBaseHelper dataBase;
13
                68 referências
14      public static SQLiteDataBaseHelper database
15      {
16          get
17          {
18              if (dataBase == null) //se não foi criado o banco
19              {
20                  //irá criar o arquivo onde é armazenado os dados da aplicação
21                  dataBase = new SQLiteDataBaseHelper(Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData), "XamAppGeo.db
22              }
23
24              return dataBase; //retorna o banco
25          }
26      }
27
                1 referência
28      public App() //Construtor
29      {
30          InitializeComponent(); //Componente de inicialização
31          MainPage = new AppShell(); //a página raiz da aplicação recebe a tela a ser exibida
32      }
33
```

Fonte: Elaborada pelo autor

A figura 29 mostra no construtor da página App, a página raiz recebendo a tela que é exibida. Nela é chamada a página AppShell, página responsável pelo roteamento da aplicação, como apresentado na figura 30.

Figura 30 - Roteamento da página

```
<!--Shell Item, recebe a nomenclatura para navegar até a página-->
<ShellItem Route="Login">
    <ShellContent ContentTemplate="{DataTemplate Views:Login}"/>
</ShellItem>

<ShellItem  Route="CadastroArea">
    <ShellContent ContentTemplate="{DataTemplate Views:CadastroArea}"/>
</ShellItem>

<ShellItem Route="CadastroFazenda">
    <ShellContent ContentTemplate="{DataTemplate Views:CadastroFazenda}"/>
</ShellItem>
<ShellItem Route="EditAreaModal">
    <ShellContent ContentTemplate="{DataTemplate Views:EditAreaModal}"/>
</ShellItem>
<ShellItem Route="CadastroPontos">
    <ShellContent ContentTemplate="{DataTemplate Views:CadastroPontos}"/>
</ShellItem>
<ShellItem Route="CadastroFuncionario">
    <ShellContent ContentTemplate="{DataTemplate Views:CadastroFuncionario}"/>
</ShellItem>

<ShellItem Route="ListaFuncionario">
    <ShellContent ContentTemplate="{DataTemplate Views:ListaFuncionario}"/>
</ShellItem>


<!--TabBar elemento guia para roteamento entre páginas -->
<TabBar>
    <Tab Title="Mapa" Icon="iconMapa" Route="Mapa">
        <ShellContent ContentTemplate="{DataTemplate Views:Mapa}" />
    </Tab>
    <Tab Title="Inicio" Route="Home" Icon="home">
        <ShellContent ContentTemplate="{DataTemplate Views:Home}" />
    </Tab>
    <Tab Title="Perfil" Icon="person" Route="Perfil">
        <ShellContent ContentTemplate="{DataTemplate Views:Perfil}" />
    </Tab>
</TabBar>

</Shell>
```

Fonte: Elaborada pelo autor

A figura 30 apresenta a TabBar, elemento gráfico que contém abas para navegação, conforme mostra a figura 31.

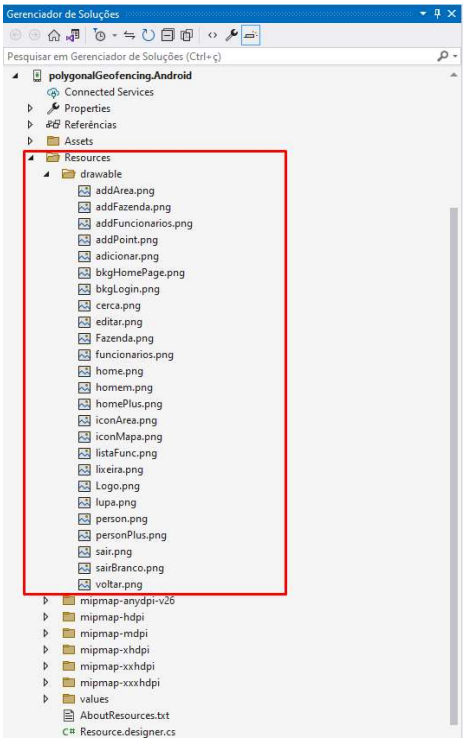
Figura 31 - TabBar

Fonte: Elaborada pelo autor

5.3 Imagens e ícones da aplicação

Nos recursos do *Android* se encontram arquivos onde são armazenadas todas as imagens e ícones que serão utilizados na aplicação. Uma vez que a imagem está dentro da pasta *drawable,* é necessário somente atribuir o nome da imagem em algum elemento capaz de recebê-la, como apresentado na figura 32 e 33.

Figura 32 - Imagens e ícones da aplicação



Fonte: Elaborada pelo autor

Figura 33 - Atribuindo a imagem

```
<!--Chamando a Logo do aplicativo-->
<Image Source="Logo" WidthRequest="90" Margin="13,-65,0,0"></Image>
```

Fonte: Elaborada pelo autor

5.4 Comandos SQL

Esta seção apresenta os comandos SQLs usados para a manipulação dos dados da aplicação. Os comandos são responsáveis pela consulta, inserção, atualização e exclusão de informação do banco de dados local.

A figura 29 mostra a verificação da existência do banco de dados local. Caso não esteja criado a função SQLiteDataBaseHelper é chamada, passando o caminho como parâmetro. Essa função então constrói uma conexão com o banco de dados local e cria as tabelas, como apresentado na figura 34.

Figura 34 - Criação do banco de dados local

```
public SQLiteDataBaseHelper(string dbPath)
{
    _db = new SQLiteAsyncConnection(dbPath);
    _db.CreateTableAsync<Area>().Wait();
    _db.CreateTableAsync<Fazenda>().Wait();
    _db.CreateTableAsync<Pontos>().Wait();
    _db.CreateTableAsync<Usuario>().Wait();
    _db.CreateTableAsync<Logado>().Wait();
    _db.CreateTableAsync<Acesso>().Wait();

}
```

Fonte: Elaborada pelo autor

Para as tabelas foram criados os comandos para selecionar, atualizar, inserir e excluir informações do banco de dados, como apresentado nas figuras 35, 36, 37, 38, 39 e 40.

Figura 35 - Comandos da tabela acesso

```
/************* Comandos Acesso ***************/

//Pega todos acessos do usuário do usuário em uma fazenda especifica ou em qualquer uma
2 referências
public Task<List<Acesso>> GetAcessoUsuario(int idUsuario,int idFazenda)
{
    string sql = "";
    //se for passado um id valido e selecionado os acessos daquela fazenda
    if (idFazenda >= 0)
    {
        sql = "SELECT * from Acesso where idUsuario = " + idUsuario + " AND idFazenda = " + idFazenda;
    }
    else // se não seleciona todos acessos
    {
        sql = "SELECT * from Acesso where idUsuario = " + idUsuario;
    }

    return _db.QueryAsync<Acesso>(sql);
}

//Insere o acesso do usuário
1 referência
public Task<List<Acesso>> InsertAcesso(int idUsuario, int idTipo, int idFazenda)
{
    string sql = "INSERT INTO Acesso (idUsuario, idTipo, idFazenda) VALUES ('" + idUsuario + "', '" + idTipo + "', '" + idFazenda + "')";
    return _db.QueryAsync<Acesso>(sql);
}

//Remove o acesso do usuário
1 referência
public Task<List<Acesso>> DeleteAcessoUsuario(int idUsuario,int idTipo,int idFazenda)
{
    string sql = "DELETE from Acesso where idUsuario = '" + idUsuario + "' AND idTipo = '" + idTipo + "' AND idFazenda = '" + idFazenda + "'";
    return _db.QueryAsync<Acesso>(sql);
}

//Remove todos acessos do banco de dados local
0 referências
public Task<List<Acesso>> DelAllAcessos()
{
    string sql = "DELETE from Acesso";
    return _db.QueryAsync<Acesso>(sql);
}

/*******************************************/
```

Fonte: Elaborada pelo autor

Figura 36 - Comandos da tabela área

```
        /************** Comandos Area **************/

        //Pega todas as áreas da fazenda
        1 referência
        public Task<List<Area>> GetAllRowsArea(int idFazenda)
        {
            string sql = "SELECT * from Area where idFazenda = " + idFazenda;
            return _db.QueryAsync<Area>(sql);
        }

        //Atualiza as informações da área
        1 referência
        public Task<List<Area>> UpdateArea(int idArea,string desc)
        {
            string sql = "UPDATE Area SET descricao = '" + desc + "' WHERE idArea = " + idArea;
            return _db.QueryAsync<Area>(sql);
        }

        //Remove todas as áreas do banco de dados
        0 referências
        public Task<List<Area>> DelArea()
        {
            string sql = "DELETE FROM Area";
            return _db.QueryAsync<Area>(sql);
        }

        //Pega todas as áreas de um usuário
        3 referências
        public Task<List<Area>> GetAreasUsuario(int id)
        {
            string sql = "SELECT a.* from Area a " +
                " Inner Join Fazenda f on f.idFazenda = a.idFazenda  where f.idUsuario = " + id + "  ";
            return _db.QueryAsync<Area>(sql);
        }
        //Pega todas as áreas das fazemdas
        3 referências
        public Task<List<Area>> GetAreasFazenda(string idFazenda)
        {
            string sql = "";
            if (idFazenda.Contains(","))
            {

                idFazenda.Replace(",", "','"); //caso seja 10,20,30 -> '10','20','30'
                sql = " SELECT * from Area a " +
                      " Inner Join Fazenda f on f.idFazenda = a.idFazenda  " +
                      " where a.idFazenda IN ('" + idFazenda + "')";
            }
            else
            {
                sql = " SELECT * from Area a " +
                      " Inner Join Fazenda f on f.idFazenda = a.idFazenda  " +
                      " where a.idFazenda = " +  Int32.Parse(idFazenda);
            }

            return _db.QueryAsync<Area>(sql);
        }


        //Seleciona a área a partir do id
        3 referências
        public Task<Area> GetByIdArea(int id)
        {

            return _db.Table<Area>().FirstAsync(i => i.idArea == id);
        }
        // Insere a área
        1 referência
        public Task<int> InsertArea(Area model)
        {
            return _db.InsertAsync(model);
        }

        //Remove a área
        1 referência
        public Task<int> DeleteArea(int idArea)
        {
            return _db.Table<Area>().DeleteAsync(i => i.idArea == idArea);
        }

        /*****************************************/
    }
```

Fonte: Elaborada pelo autor

Figura 37 - Comandos da tabela fazenda

```
/************* Comandos Fazenda *************/

//Seleciona a fazenda especificada
1 referência
public Task<Fazenda> GetByIdFazenda(int id)
{
    return _db.Table<Fazenda>().FirstAsync(i => i.idFazenda == id);
}

//Seleciona a fazenda a partir do nome
2 referências
public Task<List<Fazenda>> GetByNomeFazenda(string nome,int idDono)
{
    string sql = "SELECT * FROM Fazenda WHERE nome like '%"+ nome + "%' and idUsuario = " + idDono;
    return _db.QueryAsync<Fazenda>(sql);
}

//Deleta todas fazendas do banco de dados
0 referências
public Task<List<Fazenda>> DelFazenda()
{
    string sql = "DELETE FROM Fazenda";
    return _db.QueryAsync<Fazenda>(sql);
}

2 referências
public Task<List<Fazenda>> GetFazendaUsuario(int idDono)
{
    string sql = "SELECT * from Fazenda where idUsuario = " + idDono;
    return _db.QueryAsync<Fazenda>(sql);
}

//Pegar as fazenda que o Funcionario tem acesso
6 referências
public Task<List<Fazenda>> GetFazendaFuncionario(int idFuncionario,int idDono)
{
    string sql = "";
    if (idFuncionario == idDono)
    {
        sql = " SELECT distinct(f.idFazenda), f.nome from Fazenda f " +
              " WHERE f.idUsuario = " + idDono;
    }
    else{
         sql = " SELECT distinct(f.idFazenda), f.nome from Fazenda f " +
             " INNER JOIN Acesso a ON a.idFazenda = f.idFazenda " +
             " WHERE a.idUsuario = " + idFuncionario;
    }

    return _db.QueryAsync<Fazenda>(sql);
}

//Inserir fazenda
1 referência
public Task<int> InsertFazenda(Fazenda model)
{
    return _db.InsertAsync(model);
}

//Deletar fazenda
1 referência
public Task<int> DeleteFazenda(int idFazenda)
{
    return _db.Table<Fazenda>().DeleteAsync(i => i.idFazenda == idFazenda);
}


/********************************************/
```

Fonte: Elaborada pelo autor

Figura 38 - Comandos da tabela logado

```
/************* Comandos Usuário Logado *************/

//Seleciona o usuário logado
15 referências
public Task<List<Logado>> GetLogado()
{
    string sql = "SELECT * from Logado";
    return _db.QueryAsync<Logado>(sql);
}

//Deleta o usuário logado
4 referências
public Task<List<Logado>> DelLogado()
{
    string sql = "DELETE FROM Logado";
    return _db.QueryAsync<Logado>(sql);
}

//Insere o usuário logado
2 referências
public Task<int> InserirLogado(Logado model)
{
    return _db.InsertAsync(model);
}

/*****************************************/
```

Fonte: Elaborada pelo autor

Figura 39 - Comandos da tabela pontos

```
/************* Comandos Pontos *************/

//Deleta todos os pontos
0 referências
public Task<List<Pontos>> DelPontos()
{
    string sql = "DELETE FROM Pontos";
    return _db.QueryAsync<Pontos>(sql);
}

//Deleta todos pontos da área
1 referência
public Task<List<Pontos>> DellPontosArea(int idArea)
{
    string sql = "DELETE FROM Pontos WHERE idArea = " + idArea;
    return _db.QueryAsync<Pontos>(sql);
}

//Seleciona todos os pontos da área
5 referências
public Task<List<Pontos>> GetPontosArea(int idArea)
{
    string sql = "SELECT * from Pontos WHERE idArea = " + idArea;
    return _db.QueryAsync<Pontos>(sql);

}

//Insere os pontos
1 referência
public Task<int> InsertPonto(Pontos model)
{
    return _db.InsertAsync(model);
}

/*****************************************/
```

Fonte: Elaborada pelo autor

Figura 40 - Comandos da tabela usuário

```
/************* Comandos Usuários *************/

//Seleciona o usuário a partir do id
14 referências
public Task<Usuario> GetByIdUsuario(int id)
{
    return _db.Table<Usuario>().FirstAsync(i => i.idUsuario == id);
}

//Seleciona o Usuário a partir do e-mail e da senha
3 referências
public Task<List<Usuario>> GetUsuario(string email, string senha)
{
    string sql = "SELECT * from Usuario where email = '" + email + "' and senha = '" + senha + "' AND ativo = 1";
    return _db.QueryAsync<Usuario>(sql);
}

//Seleciona o usuário a partir do e-mail
2 referências
public Task<List<Usuario>> verificarEmail(string email)
{
    string sql = "SELECT * from Usuario where email = '" + email + "' AND ativo = 1";
    return _db.QueryAsync<Usuario>(sql);
}

//Seleciona os funcionarios da fazenda
3 referências
public Task<List<Usuario>> GetFuncionariosFazenda(string idFazenda,int idLogado, int idDono)...

//Deleta todos usuários
0 referências
public Task<List<Usuario>> DelAllUsuarios()
{
    string sql = "DELETE FROM Usuario";
    return _db.QueryAsync<Usuario>(sql);
}
//Insere o usuário no banco de dados local
2 referências
public Task<int> CadastroUsuario(Usuario model)
{
    return _db.InsertAsync(model);
}

//Atualiza informações do usuário
2 referências
public Task<List<Usuario>> UpdateDono(int idDono,int idUsuario)
{
    string sql = "UPDATE Usuario SET idDono = " + idDono + " WHERE idUsuario = " + idUsuario;
    return _db.QueryAsync<Usuario>(sql);
}

//Atualiza o status de ativo do usuário
1 referência
public Task<List<Usuario>> ExcluirUsuario(int idUsuario)
{
    string sql = "UPDATE Usuario SET ativo = 0 WHERE idUsuario = " + idUsuario;
    return _db.QueryAsync<Usuario>(sql);
}

/*********************************************/
```

Fonte: Elaborada pelo autor

5.5 Estrutura MVVM

Esta seção apresenta a estrutura MVVM implementada na aplicação, mostrando as *Models*, *Views* e as *ViewsModels* utilizadas para estruturar e manipular as informações da aplicação.

5.5.1 *Model*s

Na camada Models se encontra o *Model*o de negócio, as classes que serão utilizadas para se interagir com o banco de dados local. Serão utilizadas classes para manipulação dos dados da tabela fazenda, área, pontos, usuário, acesso e usuário logado.

Cada classe é responsável por uma tabela no banco local, que recebeu as variáveis que representam as colunas onde serão armazenadas as informações relacionadas a tabela. Na criação das chaves são especificados o tipo e o nome do campo.

Por exemplo, como foi implementado, a *Model* Fazenda irá receber o idFazenda implementado como *Auto Incremente*, ou seja, a cada registro adicionado na tabela este id receberá o valor atual dele mais 1, sendo ele um identificador único de cada registro, o idUsuario tem a função de identificar o usuário responsável pela fazenda, a variável nome e o campo que o usuário irá nomear a fazenda. Como apresentado na figura 41.

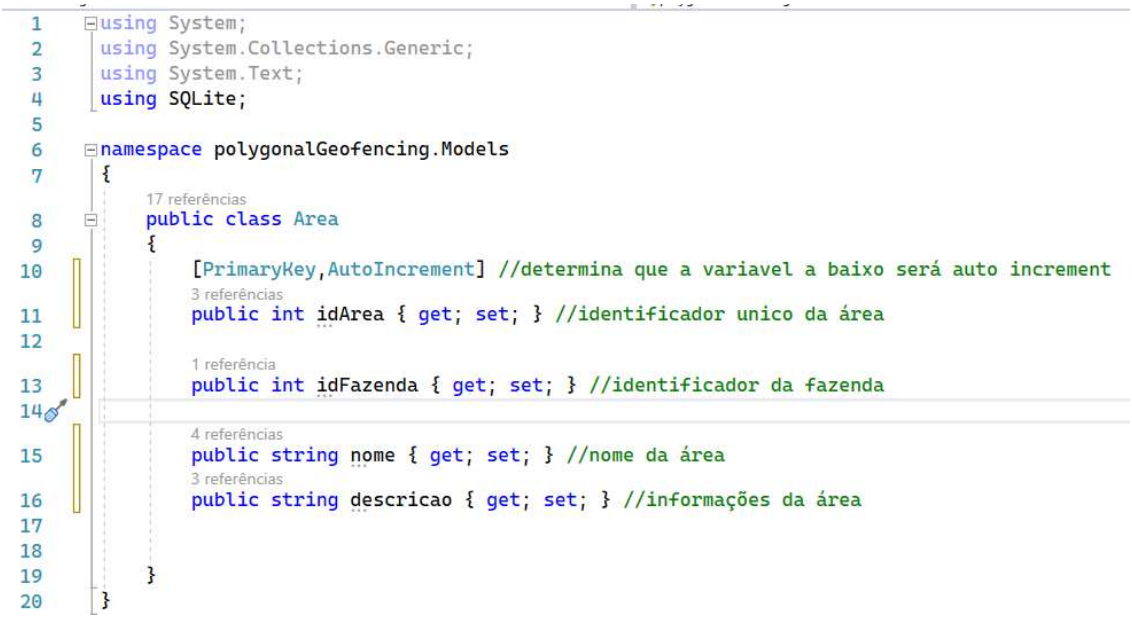
Figura 41 - *Model* Fazenda

```
using SQLite;
using System;
using System.Collections.Generic;
using System.Text;

namespace polygonalGeofencing.Models
{
    23 referências
    public class Fazenda
    {
        [PrimaryKey, AutoIncrement] //determina que a variavel a baixo será auto increment
        6 referências
        public int idFazenda { get; set; } //identificador unico da fazenda

        1 referência
        public int idUsuario { get; set; } //identificador unico do Usuario

        2 referências
        public string nome { get; set; } //nome da fazenda

    }
}
```

Fonte: Elaborada pelo autor

Por exemplo, como foi implementado, a *Model* Área é responsável por receber as informações pertinentes à área, nela terá seu identificador único idArea, uma chave secundaria idFazenda para que haja um relacionamento da tabela área com a tabela fazenda, onde uma fazenda pode conter várias áreas, o nome da área e a sua descrição, que é qualquer informação relacionado a área que o usuário julgar importante, como apresentado na figura 42.

Figura 42 - *Model* Área

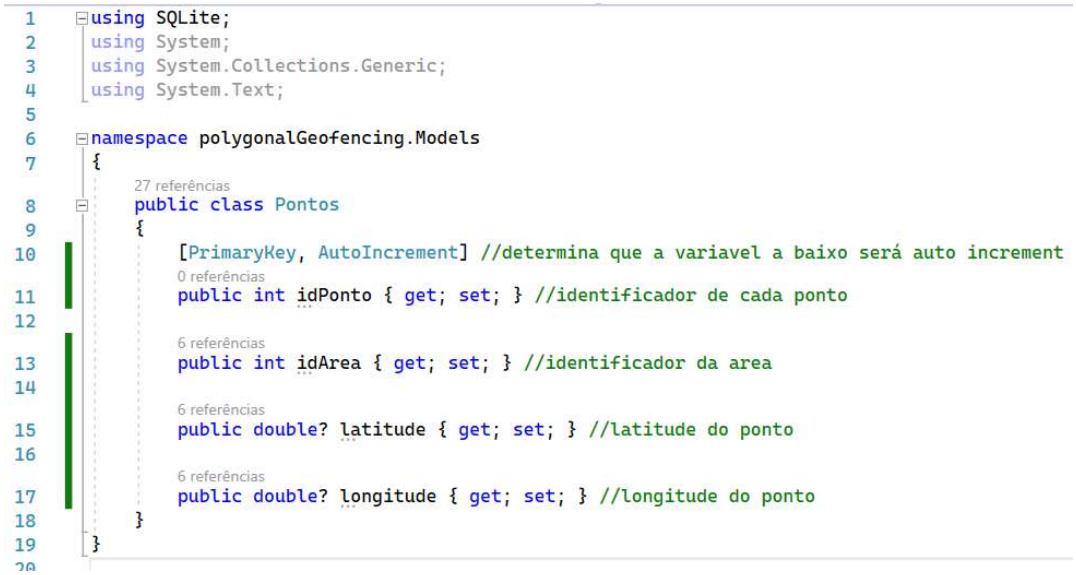


```
using System;
using System.Collections.Generic;
using System.Text;
using SQLite;

namespace polygonalGeofencing.Models
{
    17 referências
    public class Area
    {
        [PrimaryKey,AutoIncrement] //determina que a variavel a baixo será auto increment
        3 referências
        public int idArea { get; set; } //identificador unico da área

        1 referência
        public int idFazenda { get; set; } //identificador da fazenda

        4 referências
        public string nome { get; set; } //nome da área
        3 referências
        public string descricao { get; set; } //informações da área


    }
}
```

Fonte: Elaborada pelo autor

Por exemplo, como foi implementado, a *Model* Pontos é responsável por receber as informações de cada ponto, sendo o identificador de cada registro idPonto, o identificador da área que este ponto pertence e sua posição geográfica, latitude e longitude, como apresentado na figura 43 de código

Figura 43 - *Model* Pontos



```
using SQLite;
using System;
using System.Collections.Generic;
using System.Text;

namespace polygonalGeofencing.Models
{
    27 referências
    public class Pontos
    {
        [PrimaryKey, AutoIncrement] //determina que a variavel a baixo será auto increment
        0 referências
        public int idPonto { get; set; } //identificador de cada ponto

        6 referências
        public int idArea { get; set; } //identificador da area

        6 referências
        public double? latitude { get; set; } //latitude do ponto

        6 referências
        public double? longitude { get; set; } //longitude do ponto
    }
}
```

Fonte: Elaborada pelo autor

Por exemplo, como foi implementado, a *Model* Usuário encapsula as informações do usuário. O *Model*o contém o identificador único do usuário, identificador único do dono da fazenda, nome, sobrenome, e-mail, senha e uma outra variável do tipo inteiro para mostrar se o usuário está ativo ou não, como apresentado na figura 44.

Figura 44 - *Model* Usuário

```
using System;
using System.Collections.Generic;
using System.Text;
using SQLite;

namespace polygonalGeofencing.Models
{
    45 referências
    public class Usuario
    {
        [PrimaryKey, AutoIncrement] //determina que a variável a baixo será AutoIncrement
        9 referências
        public int idUsuario { get; set; } //identificador do Usuário

        8 referências
        public int idDono { get; set; } //identificador do Dono da Fazenda

        2 referências
        public int ativo { get; set; } //informa se o usuário esta ativo ou não no sistema

        5 referências
        public string nome { get; set; } //Nome do usuário

        4 referências
        public string sobrenome { get; set; } //Sobrenome do usuário

        2 referências
        public string email { get; set; } //email do usuário

        2 referências
        public string senha { get; set; } //senha do usuário
```

Fonte: Elaborada pelo autor

Por exemplo, como foi implementado, a *Model* Logado irá encapsular as regras do negócio após o usuário realizar o *login* no aplicativo, registros como identificador único do usuário e a data em que o usuário entrou na aplicação. Como apresentado na figura 45.

Figura 45 - *Model* Logado

```
1   using System;
2   using System.Collections.Generic;
3   using System.Text;
4   using SQLite;
5
6   namespace polygonalGeofencing.Models
7   {
        13 referências
8       public class Logado
9       {
10          [PrimaryKey,AutoIncrement] //determina que a variável a baixo será AutoIncrement
11
            0 referências
12          public int idLogado { get; set; } //identificador Logado
13
            6 referências
14          public int idUsuario { get; set; } //identificador do Usuário
15
            3 referências
16          public DateTime data { get; set; } //variável data
17
18      }
19  }
```

Fonte: Elaborada pelo autor

Por exemplo, como foi implementado, a *Model* Acesso é responsável pelos acessos dos usuários ao aplicativo, através das características como o identificador do usuário, o tipo de acesso e qual a fazenda onde o acesso está sendo atribuído. Com isso é capaz de gerenciar áreas de acesso do aplicativo, como apresentado na figura 46.

Figura 46 - *Model* Acesso

```
using SQLite;
using System;
using System.Collections.Generic;
using System.Text;

namespace polygonalGeofencing.Models
{
    15 referências
    public class Acesso
    {
        [PrimaryKey, AutoIncrement] //determina que a variavel a baixo será auto increment
        0 referências
        public int idAcesso { get; set; }  //identificador unico do Acesso

        0 referências
        public int idUsuario { get; set; } // identificador do usuario

        // 1 - Criar Area 2 - Add Funcionario 3 - Add Permissão
        2 referências
        public int idTipo { get; set; } // identificador do tipo de permisão

        0 referências
        public int idFazenda { get; set; } // identificador da Fazenda em que tem acesso

    }
}
```

Fonte: Elaborada pelo autor

5.5.2 Views e ViewModel

Neste subtópico é apresentada a parte responsável pela aparência e funcionalidade do aplicativo, sendo a *View* responsável pela aparência e a *ViewModel* pelas funcionalidades dos elementos da *View*.

Para a construção da interface de entrada foi utilizada a estrutura *grid,* onde são definidas as linhas e colunas para a estruturação das informações, seguido de tags como *image, labels e buttons.*

A *tag image* é responsável pelo carregamento da imagem passada como atributo, as *tags labels* são textos e *buttons* são elementos clicáveis que chamam uma função para execução da tarefa programada. O evento de chamar a função se dá pela atribuição do atributo *clicked* passando como parâmetro à função a ser chamada, como apresentado na figura 47.

Figura 47 - *View* página inicial

```
<Grid RowDefinitions="*"> <!--Grid definindo a linha como * ou sejá irá preencher toda a página-->
    <!--Elemento para organização das informações-->
    <StackLayout >

        <!--Chamando a imagem que irá compor o topo da página-->
        <Image Source="bkgLogin" Aspect="AspectFill" ></Image>


        <!--Definindo a linha 1 do grid-->
        <StackLayout Grid.Row="1">

            <!--Chamando a Logo do aplicativo-->
            <Image Source="Logo" WidthRequest="90" Margin="13,-65,0,0"></Image>

            <!-- Elemento para organizar o texto da pagina inicial -->
            <StackLayout HorizontalOptions="Center" Grid.Column="1" Orientation="Horizontal" Spacing="10">

                <Label Text="Farm" FontSize="34" FontFamily="Cabin" TextColor="#2595ef"></Label>
                <Label Text="Mapping" FontSize="34" FontFamily="Cabin" TextColor="#4A484D"></Label>
            </StackLayout>
            <Label Text="Mapeamento de Área" FontSize="11" CharacterSpacing="2" WidthRequest="130" HorizontalOptions="Center" FontFamily="Metropolis"></Lab
            <Label Text="Descubra o melh Label.Text tivo de mapeamento de áreas para monitoramento." FontSize="12" CharacterSpacing="2" HorizontalTextAlignmen

            <!-- Botões para realizar o login ou o cadastro -->
            <Button Text="Entrar" x:Name="btnEntrar" BackgroundColor="#2595ef" TextColor="white" WidthRequest="310" HeightRequest="55" HorizontalOptions
            <Button Text="Cadastrar" x:Name="btnCadastrar" BorderColor="#2595ef" BorderWidth="1" BackgroundColor="Transparent" TextColor="#2595ef" Widt
        </StackLayout>
    </StackLayout>
</Grid>
```

Fonte: Elaborada pelo autor

A figura 48 mostra a interface que o usuário irá ver após a implementação da estrutura *view* da página inicial.

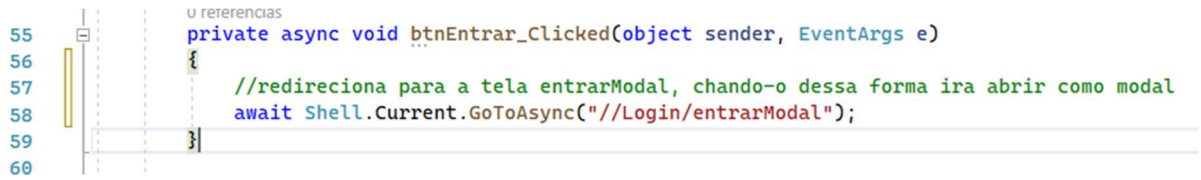
Figura 48 - Tela inicial

Fonte: Elaborada pelo autor

Como apresentado na Figura 48, ao abrir o aplicativo, o usuário terá contato com uma interface onde se dá duas opções para selecionar, sendo elas um botão para entrar no aplicativo caso já tenha conta, caso contrário a opção de se cadastrar para que futuramente possa acessar as funcionalidades da aplicação.

Ao clicar no botão entrar, o usuário é encaminhado a uma tela de *login*, como apresentado na figura 49.

Figura 49 - Evento de *click* no botão entrar



Fonte: Elaborada pelo autor

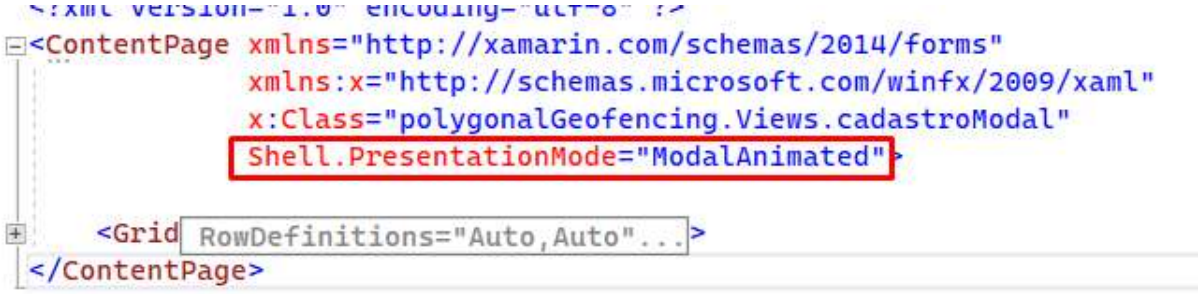
Ao clicar no botão cadastrar, o usuário é redirecionado para a tela de cadastro, como apresentado na figura 50.

Figura 50 - Evento *click* do botão cadastrar

Fonte: Elaborada pelo autor

Para que se tenha o efeito modal, é adicionado no corpo da página um atributo, como apresentado na figura 51.

Figura 51 - Modo modal da página



Fonte: Elaborada pelo autor

Na construção da interface da tela de login foi utilizado um *grid* para organizar os elementos, dois campos *entrys* que são campos para receber as informações digitadas pelo usuário e dois botões, um deles para realizar o login e outro para fazer o cadastro, como apresentado na figura 52

Figura 52 - *View* Login

```
<Grid RowDefinitions="*"> <!--Grid-->
    <StackLayout HorizontalOptions="Center">
        <Label Text="Entrar" FontSize="40" HorizontalTextAlignment="Center" FontFamily="Metropolis" TextColor="#4A484D"
        <Label Text="Insira suas informações para entrar" HorizontalTextAlignment="Center" FontSize="15" FontFamily="Metrc

        <!--Campos para o usuário preencher-->
        <Frame CornerRadius="30" BackgroundColor="#f2f2f2" HasShadow="False" WidthRequest="310" HeightRequest="15" Margi
            <Entry x:Name="EntrarNome" Text="{Binding Email }" HorizontalTextAlignment="Start" FontSize="14" Placeholder="
        </Frame>
        <Frame CornerRadius="30" BackgroundColor="#f2f2f2" HasShadow="False" WidthRequest="310" HeightRequest="15" Margi
            <Entry x:Name="EntrarSenha" Text="{Binding Senha }" HorizontalTextAlignment="Start" FontSize="14" IsPassword="
        </Frame>

    </StackLayout>
    <StackLayout Grid.Row="1" Margin="0,30,0,0" >

        <!--Botões de entrar e de se cadastrar-->
        <Button Text="Entrar" x:Name="btnEntrar" BackgroundColor="#2595ef" TextColor="white" WidthRequest="345" HeightF

        <StackLayout HorizontalOptions="Center" Grid.Column="1" Orientation="Horizontal" Spacing="10">
            <Label Text="Não possui uma conta?" FontSize="14" FontFamily="Cabin" TextColor="#4A484D"></Label>
            <Button x:Name="btnCadastrar" Text="cadastre-se" FontSize="13" FontFamily="Cabin" TextColor="#2595ef" Backgr

        </StackLayout>


    </StackLayout>
</Grid>
```

Fonte: Elaborada pelo autor

A figura 43 mostra os campos *entrys* que são campos de entrada, para o usuário digitar. Neles se encontram o comando *Binding* que serve para vincular o texto digitado com a *viewmodel*, como apresentado na figura 53.

Figura 53 - Recebimento dos campos da *view*

```
public string Email
{
    get => email;
    set
    {
        email = value;  //a variavel email criada recebera o valor digitado pelo usuario
        PropertyChanged(this, new PropertyChangedEventArgs("emailUsuario"));
    }
}

0 referências
public string Senha
{
    get => senha;
    set
    {
        senha = value; //a variavel senha criada recebera o valor digitado pelo usuario
        PropertyChanged(this, new PropertyChangedEventArgs("senhaUsuario"));
    }
}
```

Fonte: Elaborada pelo autor

A *viewmodel* receberá os campos e-mail e senha e comparar com as informações gravadas no banco. Se as informações fornecidas pelo usuário forem iguais às informações armazenadas no banco, o usuário é redirecionado para a *home page*; se as informações passadas pelo usuário não corresponderem às gravadas no banco então um alerta será disparado informando-o das informações incorretas, como apresentado na figura 54.

Figura 54 - validação das informações do usuário

```
public ICommand Entrar //recebe o comando do botão entrar
{
    get => new Command(async () => {
        try
        {
            List<Usuario> model = await App.database.GetUsuario(this.email, this.senha); //consulta no banco as credenciais informadas pelo usuário
            if (model.ToList().Count > 0) //caso a lista recebida seja maior que 0, ou seja conseguiu achar informação no banco
            {
                await App.database.DelLogado(); //deleta o registro de usuário logado
                Logado Logado = new Logado() //classe Logado, para registrar qual usuário esta logado,
                                             //para que na proxima vez que entrar no aplicativo logue direto
                {
                    idUsuario = model.ToList()[0].idUsuario, //insere o id do usuario que esta logando
                    data = DateTime.Now //adiciona a data atual para que no proximo login seja verificado se a sessão dele expirou
                };
                await App.database.InserirLogado(Logado); //insere as informações no banco
                await Shell.Current.GoToAsync("//Home");

                //App.Current.MainPage = new AppShell(); //redireciona o usuário para a homePage
            }
            else
            {
                await Application.Current.MainPage.DisplayAlert("Erro", "Email ou senha invalida", "OK"); //credenciais não encontradas no banco
            }
        }
        catch (Exception ex)
        {
            await Application.Current.MainPage.DisplayAlert("Ops", ex.Message, "ok"); //caso alguma instrução de erro
        }


    });
}
```

Fonte: Elaborada pelo autor

Quando o login é efetuado o id do usuário e a data do momento em que foi feito o login é armazenado na tabela logado. Na próxima vez em que o usuário for entrar na aplicação ele será automaticamente redirecionado para a *home page* sem a necessidade de colocar as suas credenciais. Para isso acontecer a diferença da data atual em que o usuário está tentando entrar no aplicativo e a data armazenada na tabela logado deverá ser de menos de 2 dias, caso contrário serão solicitadas as credenciais para entrar na aplicação, como apresentado na figura 55.

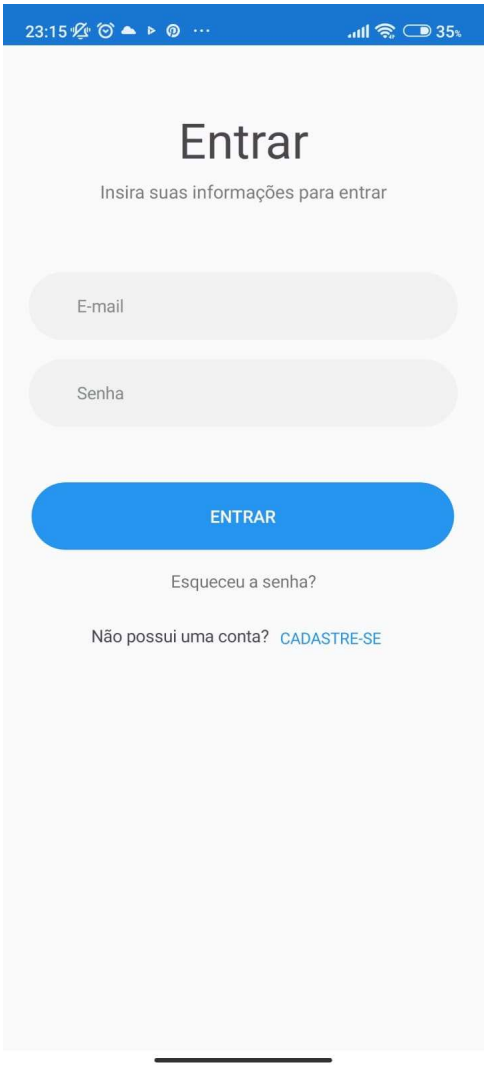
Figura 55 - Verificação de tempo logado

```
List<Logado> logado = await App.database.GetLogado(); //pega as informações da tabela logado

if (logado.Count > 0) //verifica se há registro
{
    //verifica a diferença entre a data atual e a data do banco
    int diasLogado = (int)DateTime.Today.Subtract(logado.ToList()[0].data).TotalDays;
    /*Verifica no banco se este usuário ja esta logado a mais de 2 dias, se estiver remover o registro dele,
    para que ele possa logar de novo*/
    if (diasLogado >= 2)
    {
        await App.database.DelLogado(); //deletrar as informações da tabela logado
    }
    else
    {
        //se a diferença for de menos de 2 dias redirecionar para a home page
        await Shell.Current.GoToAsync("//Home");
    }

}
```

Fonte: Elaborada pelo autor

A figura 56 mostra a página de login que o usuário terá contato.

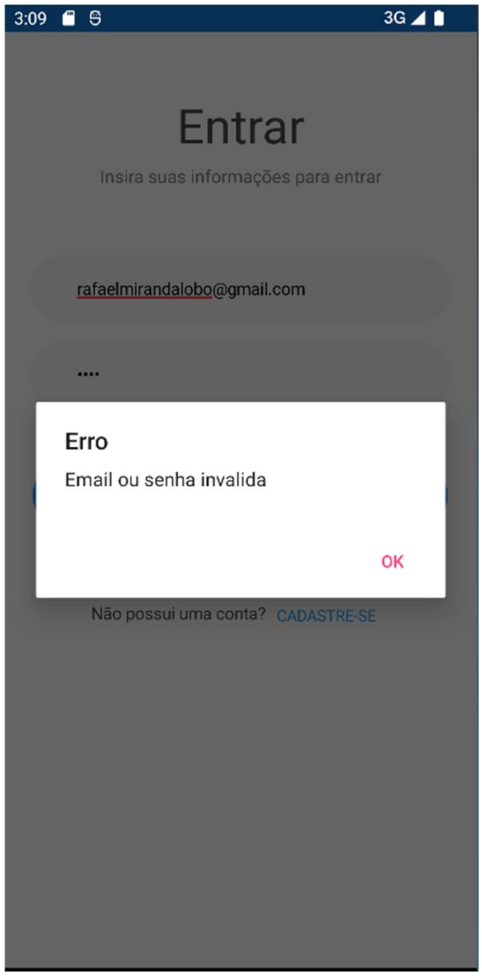
Figura 56 - Tela de *login*



Fonte: Elaborada pelo autor

O alerta emitido, caso as informações sejam incompatíveis com as armazenadas no banco de dados, é mostrada na figura 57

Figura 57 - Alerta de credenciais invalidas



Fonte: Elaborada pelo autor

Como apresentado na figura 56, na tela de login é disponibilizado para o usuário um botão de cadastro caso ele não tenha um e-mail e senha cadastrada,

Na construção da interface da tela de cadastro foi utilizado um *grid* para organizar os elementos, campos *entrys* para receber as informações como nome, sobrenome, e-mail, senha e confirmação da senha. Nos campos nome e sobrenome tem-se um método para que a primeira letra digitada sempre seja maiúscula, como apresentado na figura 58 e 59.

Figura 58 - *View* Cadastro

```
<!--Grid para organizar os elementos-->
<Grid RowDefinitions="Auto,Auto">
    <StackLayout Grid.Row="0" HorizontalOptions="Center">
        <Label Text="Cadastrar" FontSize="40" HorizontalTextAlignment="Center" FontFamily="Metropc
        <Label Text="Insira suas informações para cadastrar" HorizontalTextAlignment="Center" Font
        <!--Entrys com capitalize word, ou seja a primeira letra sempre vai ser maiuscula-->
        <Frame CornerRadius="30" BackgroundColor="#f2f2f2" HasShadow="False" WidthRequest="310"
            <Entry x:Name="cadastroNome" Text="{Binding Nome }"  HorizontalTextAlignment="Start" F
                <Entry.Keyboard>
                    <Keyboard x:FactoryMethod="Create">
                        <x:Arguments>
                            <KeyboardFlags>CapitalizeWord</KeyboardFlags>
                        </x:Arguments>
                    </Keyboard>
                </Entry.Keyboard>
            </Entry>
        </Frame>
        <!--Frame campo sobrenome-->
        <Frame CornerRadius="30" BackgroundColor="#f2f2f2" HasShadow="False" WidthRequest="310"
            <Entry x:Name="cadastroSobranome"  Text="{Binding Sobrenome }" HorizontalTextAlignment
                <Entry.Keyboard>
                    <Keyboard x:FactoryMethod="Create">
                        <x:Arguments>
                            <KeyboardFlags>CapitalizeWord</KeyboardFlags>
                        </x:Arguments>
                    </Keyboard>
                </Entry.Keyboard>
            </Entry>
        </Frame>
        <!--frane campo e-mail-->
        <Frame CornerRadius="30" BackgroundColor="#f2f2f2" HasShadow="False" WidthRequest="310" H
            <Entry x:Name="cadastroEmail" Text="{Binding Email }" HorizontalTextAlignment="Start" I
        </Frame>
        <!--Frame campo senha-->
        <Frame CornerRadius="30" BackgroundColor="#f2f2f2" HasShadow="False" WidthRequest="310" H
            <Entry x:Name="cadastroSenha" Text="{Binding Senha }" HorizontalTextAlignment="Start" I
        </Frame>
        <!--Frame campo confirmação de senha-->
        <Frame CornerRadius="30" BackgroundColor="#f2f2f2" HasShadow="False" WidthRequest="310" H
            <Entry x:Name="cadastroConfSenha" Text="{Binding confSenha }" HorizontalTextAlignment=
        </Frame>

    </StackLayout>
    <!--Botões cadastrar e logar-->
    <StackLayout Grid.Row="1" Margin="0,50,0,0"  >
        <Button x:Name="btnCadastrar" Text="Cadastrar" BackgroundColor="#2595ef" TextColor="whi


        <StackLayout HorizontalOptions="Center" Grid.Column="1" Orientation="Horizontal" Spacing="
            <Label Text="Já possui uma conta?" FontSize="14" FontFamily="Cabin" TextColor="#4A484
            <Button x:Name="btnEntrar" Text="Entrar" FontSize="13" FontFamily="Cabin" TextColor="
        </StackLayout>


    </StackLayout>
</Grid>
```

Fonte: Elaborada pelo autor

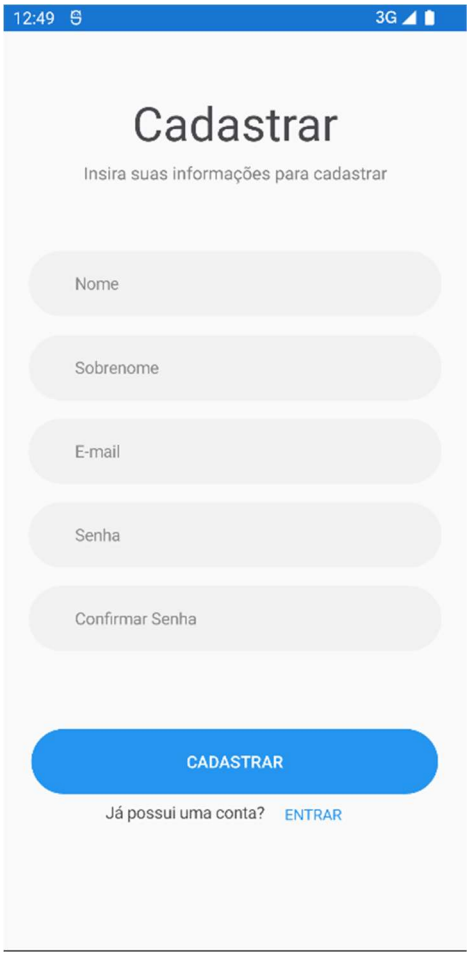
Figura 59 - Método para deixar a primeira letra maiúscula

```
<!--Entrys com capitalize word, ou seja a primeira letra sempre vai ser maiusc
<Frame CornerRadius="30" BackgroundColor="#f2f2f2" HasShadow="False" WidthReq
    <Entry x:Name="cadastroNome" Text="{Binding Nome }"  HorizontalTextAlignmer
        <Entry.Keyboard>
            <Keyboard x:FactoryMethod="Create">
                <x:Arguments>
                    <KeyboardFlags>CapitalizeWord</KeyboardFlags>
                </x:Arguments>
            </Keyboard>
        </Entry.Keyboard>
    </Entry>
</Frame>
```

Fonte: Elaborada pelo autor

A figura 60 mostra a interface de cadastro gerada para que o usuário possa interagir.

Figura 60 - *Interface* de cadastro

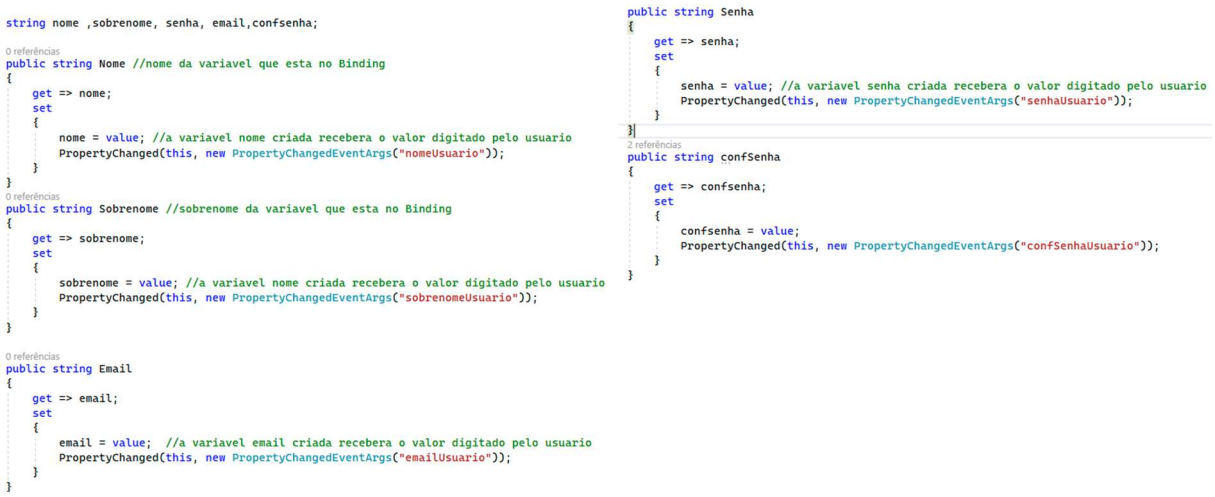


Fonte: Elaborada pelo autor

Como apresentado na figura 60, são disponibilizados ao usuário os campos necessários para o cadastro. Assim que ele preencher as informações e clicar no botão cadastrar, essas informações serão enviadas para a *viewmodel* para que possam ser validadas.

Ao clicar no botão cadastrar, a *viewmodel* irá receber os campos nome, sobrenome, e-mail, senha e confirmação de senha. Para que seja um cadastro válido, o e-mail informado não pode conter nos registros do banco de dados e as senhas digitadas deverão ser iguais. Se as informações passadas estiverem dentro das condições, o usuário é cadastrado; caso contrário, um alerta é emitido informando o que não está de acordo, como apresentado na figura 61 e 62.

Figura 61 - Recebimento dos campos digitados pelo usuário na tela de cadastro



```
string nome ,sobrenome, senha, email,confsenha;

0 referências
public string Nome //nome da variavel que esta no Binding
{
    get => nome;
    set
    {
        nome = value; //a variavel nome criada recebera o valor digitado pelo usuario
        PropertyChanged(this, new PropertyChangedEventArgs("nomeUsuario"));
    }
}
0 referências
public string Sobrenome //sobrenome da variavel que esta no Binding
{
    get => sobrenome;
    set
    {
        sobrenome = value; //a variavel nome criada recebera o valor digitado pelo usuario
        PropertyChanged(this, new PropertyChangedEventArgs("sobrenomeUsuario"));
    }
}

0 referências
public string Email
{
    get => email;
    set
    {
        email = value;  //a variavel email criada recebera o valor digitado pelo usuario
        PropertyChanged(this, new PropertyChangedEventArgs("emailUsuario"));
    }
}
```

```
public string Senha
{
    get => senha;
    set
    {
        senha = value; //a variavel senha criada recebera o valor digitado pelo usuario
        PropertyChanged(this, new PropertyChangedEventArgs("senhaUsuario"));
    }
}
2 referências
public string confSenha
{
    get => confsenha;
    set
    {
        confsenha = value;
        PropertyChanged(this, new PropertyChangedEventArgs("confSenhaUsuario"));
    }
}
```

Fonte: Elaborada pelo autor

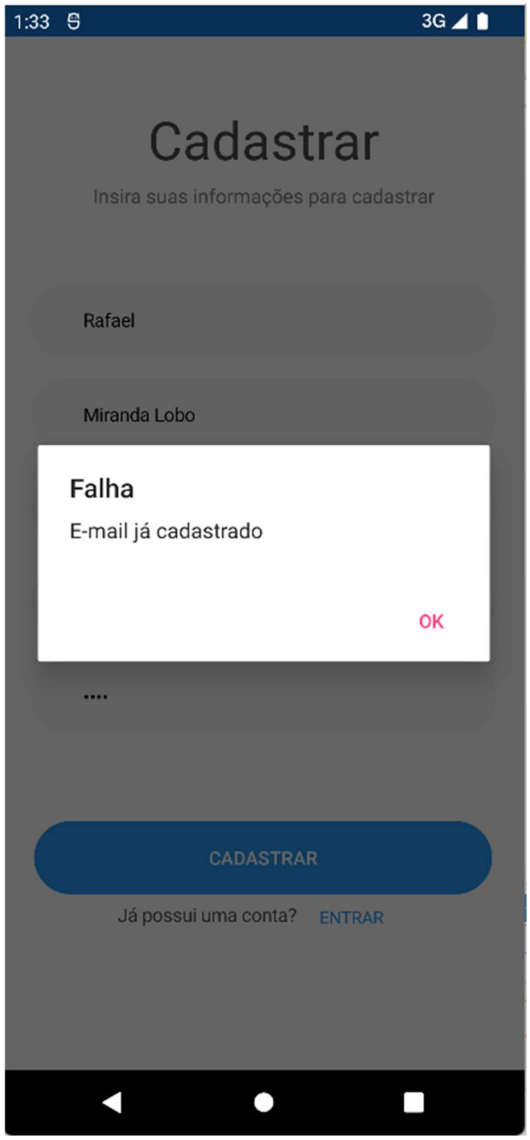
Figura 62 - Cadastro do usuário

```
0 referências
public ICommand NovoUsuario
    {
        get => new Command(async () => {

            try
            {

                Usuario model = new Usuario() //instancia um novo objeto atribuindo os valores digitados
                    {
                        nome = this.nome,
                        sobrenome = this.sobrenome,
                        senha = this.senha,
                        email = this.email,
                        ativo = 1
                    };

                //se os campos não forem nulos
                if (nome != null && sobrenome != null && senha != null && email != null && confSenha != null)
                {
                    //verifica se as senhas são compativeis
                    if(senha == confSenha)
                    {
                        //procura no banco o e-mail digitado para ver se ja existe cadastro com o e-mail
                        List<Usuario> verEmail =  await App.database.verificarEmail(email);
                        if (verEmail.Count == 0)//caso o e-mail não esteja cadastrado
                        {
                            await App.database.CadastroUsuario(model);//insere as informações do usuário no banco
                            List<Usuario> usuario = await App.database.GetUsuario(email, senha); //pega os dados do usuário cadastrado
                            int idDono = usuario.ToList()[0].idUsuario; //pega o id do Usuário cadastrado
                            await App.database.UpdateDono(idDono, idDono); //atualiza os dados do usuário colocando o id dele como dono
                bool answer = await Application.Current.MainPage.DisplayAlert("Usuário Cadastrado", "Deseja fazer o login", "Sim", "Não");
                if (answer) //caso a resposta seja sim
                {
                    await App.database.DelLogado(); //limpa a tabela logado
                    Logado Logado = new Logado()
                    {
                        idUsuario = usuario.ToList()[0].idUsuario, //insere na tabela logado o id do usuário
                        data = DateTime.Now //coloca o data e a hora do login
                    };

                    await App.database.InserirLogado(Logado);

                    await Shell.Current.GoToAsync("//Home");//usuário e redirecionado para a home page
                }
            }
            else
            {
                //caso o e-mail já for cadastrado e disparado um alerta para o usuário
                await Application.Current.MainPage.DisplayAlert("Falha", "E-mail já cadastrado", "Ok");
            }

        }
        else
        {
            //caso as senhas digitadas não forem iguais e disparado um alerta para o usuário
            await Application.Current.MainPage.DisplayAlert("Falha", "Senhas diferentes", "ok");
        }
    }
    else
    {
        await Application.Current.MainPage.DisplayAlert("Falha", "Preencha Todos os campos", "ok");
    }
```

Fonte: Elaborada pelo autor

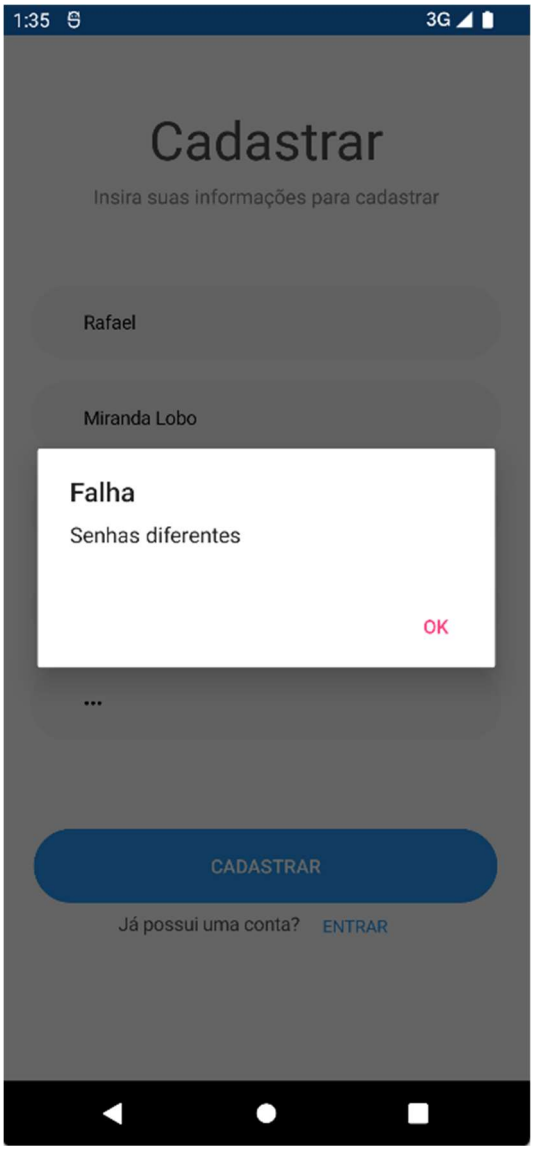
As figuras 63 e 64 mostram o alerta, caso as informações não atendam os requisitos.

Figura 63 - Alerta de e-mail já cadastrado



Fonte: Elaborada pelo autor

Figura 64 - Alerta de senhas divergentes



Fonte: Elaborada pelo autor

Assim que for realizado o cadastro, irá aparecer uma mensagem informando o sucesso do cadastro e perguntando se o usuário deseja realizar o login na aplicação, como apresentado na figura 65

Figura 65 - Mensagem após o cadastro

Fonte: Elaborada pelo autor

Caso o usuário opte pelo não, nenhuma ação será realizada e é disponibilizado o botão de *login*. Sendo sua escolha o sim, a tela é redirecionada para *Home.*

Home é a tela inicial após o usuário fazer o login na aplicação. Nessa tela o usuário poderá acessar grande parte das funcionalidades da aplicação, como, adicionar fazenda, adicionar funcionário e ver a lista de funcionários. Além dessas funcionalidades o usuário terá acesso a *TabBar* que contém os itens que o levará para visualizar o mapa, visualizar o perfil e acesso a *Home.* É disponibilizado para o usuário um botão para que ele possa sair da conta logada.

Na construção da Home foi utilizada uma grid para organização dos elementos. São eles, imagem representativa do usuário, nome completo do usuário, *card* contendo quantidades de fazendas, áreas e funcionários que o usuário tem permissão. São os cards que levarão o usuário para as outras telas, como apresentado nas figuras 66, 67 e 68.

Figura 66 - Topo da página *Home*

```
<RelativeLayout>
    <!--Imagem do topo da página-->
    <Image Source="bkgHomePage" Aspect="AspectFill" Margin="0,-137,0,0"></Image>

    <FlexLayout RelativeLayout.XConstraint="30"
                RelativeLayout.YConstraint="60"
                 >
        <!--Moldura da imagem-->
        <Frame Margin="10"
                BackgroundColor="#F1F1F1"
                CornerRadius="50"
                HeightRequest="70"
                WidthRequest="30"
               >
            <!--Imagem representantativa do usuário-->
            <Image Source="homem"
                Aspect="AspectFill"
                Margin="-11"
                />
         </Frame>

        <!--Frase de boas vindas para o usuário-->
        <StackLayout>
            <Label
                Text="Bem Vindo(a)!"
               TextColor="White"
               FontSize="17"
               Margin="0,15,0,0"
              ></Label>
          <Label  x:Name="Usuario"
            TextColor="White"
            FontSize="17"
            Margin="0,-10,0,0"
           ></Label>
        </StackLayout>

    </FlexLayout>

    <!--Botão para sair da conta logada-->
    <FlexLayout RelativeLayout.XConstraint="300"
                RelativeLayout.YConstraint="15">
        <Frame BackgroundColor="Transparent">
            <Frame.GestureRecognizers>
                <TapGestureRecognizer Tapped="TapGestureRecognizer_Tapped" Command="{Binding
            </Frame.GestureRecognizers>
        </Frame>
        <Label Text="Sair" TextColor="White" Margin="-10,0,10,0"></Label>
        <ImageButton Source="sairBranco" WidthRequest="20" HeightRequest="20" Command="{Bind:
        </ImageButton>
    </FlexLayout>
</RelativeLayout>
```

Fonte: Elaborada pelo autor

Figura 67 - Frame Visão Geral

```
<Frame Grid.Row="1"
       Margin="0,-190,0,0"
             BackgroundColor="#fff"
             WidthRequest="270"
             HeightRequest="130"
             HorizontalOptions="CenterAndExpand"
             CornerRadius="50"
             VerticalOptions="Start"
            HasShadow="True">
    <!--grid para organizar as informações de visão geral-->
    <Grid>
        <Grid.ColumnDefinitions >
            <ColumnDefinition Width="Auto"></ColumnDefinition>
            <ColumnDefinition Width="Auto"></ColumnDefinition>
            <ColumnDefinition Width="Auto"></ColumnDefinition>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"></RowDefinition>
            <RowDefinition Height="Auto"></RowDefinition>
        </Grid.RowDefinitions>
        <Label Text="Visão Geral" HorizontalTextAlignment="Center"  Grid.Row="0" Grid.Column="1"
        <!--quantidade de fazenda-->
        <StackLayout Grid.Column="0" Grid.Row="1" HorizontalOptions="Center" VerticalOptions="Cen
            <Frame Margin="10"
                   BackgroundColor="#F7F7F7"
                   CornerRadius="50"
                   HeightRequest="20"
                   WidthRequest="20"
                  >
                <Image Source="fazenda"
                 Aspect="AspectFill"
                 Margin="-9"
                 />
            </Frame>
            <Label x:Name="qtdFazenda" TextColor="Black" FontSize="17"
                   HorizontalTextAlignment="Center"></Label>
        </StackLayout>
        <!--quantidade de área-->
        <StackLayout Grid.Column="1" Grid.Row="1" HorizontalOptions="Center" VerticalOptions="Cen
            <Frame Margin="10"
                   BackgroundColor="#F7F7F7"
                   CornerRadius="50"
                   HeightRequest="20"
                   WidthRequest="20"
                  >

                <Image Source="iconArea"
                 Aspect="AspectFill"
                 Margin="-19"
                 />
            </Frame>
            <Label x:Name="qtdArea" TextColor="Black" FontSize="17"
                   HorizontalTextAlignment="Center"></Label>
        </StackLayout>
        <!--quantidade de funcionarios-->
        <StackLayout Grid.Column="2" Grid.Row="1" HorizontalOptions="Center" VerticalOptions="Cen
            <Frame Margin="10"
                   BackgroundColor="#F7F7F7"
                   CornerRadius="50"
                   HeightRequest="20"
                   WidthRequest="20"
                  >

                <Image Source="funcionarios"
                 Aspect="AspectFill"
                 Margin="-9"
                 />
            </Frame>
            <Label x:Name="qtdFuncionario" Text="0" TextColor="Black" FontSize="17"
                   HorizontalTextAlignment="Center"></Label>
        </StackLayout>
    </Grid>
</Frame>
```

Fonte: Elaborada pelo autor

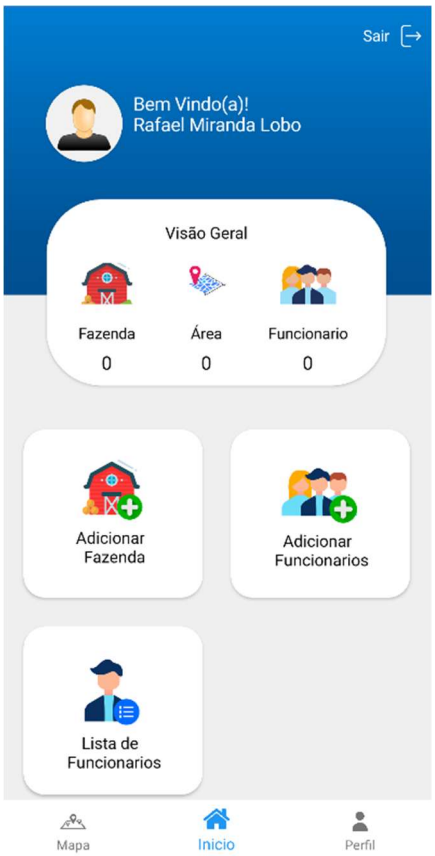
Figura 68 - *Cards* para acessar outras páginas

```
<!--grid para organização dos cards-->
<StackLayout Grid.Row="1" HorizontalOptions="Center" VerticalOptions="Center" WidthRequest="400" HeightRequest="400" >
    <ScrollView>
    <Grid Padding="10">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*"></ColumnDefinition>
            <ColumnDefinition Width="*"></ColumnDefinition>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="*"></RowDefinition>
            <RowDefinition Height="*"></RowDefinition>
        </Grid.RowDefinitions>

        <!--card para adicionar fazenda-->
            <Frame x:Name="frameFazenda"
                   Margin="10"
                   CornerRadius="20"
                   Grid.Column="0"
                   Grid.Row="0"
                   HeightRequest="100"
                   >
                <Frame.GestureRecognizers>
                    <TapGestureRecognizer  Tapped="AdicionarFazenda"/>
                </Frame.GestureRecognizers>
                <StackLayout VerticalOptions="Center" HorizontalOptions="Center">
                    <ImageButton Source="addFazenda"
                                 x:Name="cardFazenda"
                                 WidthRequest="55"
                                 HeightRequest="55"
                                 BackgroundColor="Transparent"

                    ></ImageButton>

                        <Label Text="Adicionar &#10; Fazenda" HorizontalTextAlignment="Center" TextColor="Black" FontSize=

                        </Label>
                </StackLayout>
            </Frame>
        <!--card para adicionar funcionario-->
            <Frame x:Name="frameAddFunc"
                   Margin="10"
                   CornerRadius="20"
                   Grid.Column="1"
                   Grid.Row="0">
                <Frame.GestureRecognizers>
                    <TapGestureRecognizer  Tapped="AdicionarFuncionarios"/>
                </Frame.GestureRecognizers>
                <StackLayout VerticalOptions="Center" HorizontalOptions="Center">
                    <Image Source="addFuncionarios"
                    WidthRequest="50"
                      HeightRequest="50"
                Margin="0,10,0,0"
                    ></Image>

                        <Label Text="Adicionar &#10; Funcionarios" HorizontalTextAlignment="Center" TextColor="Black" Font

                        </Label>
                </StackLayout>
            </Frame>

        <!--card para ver a lista de funcionarios-->
            <Frame x:Name="frameListaFunc"
                   Margin="10"
                   CornerRadius="20"
                   Grid.Column="0"
                   Grid.Row="1">
                <Frame.GestureRecognizers>
                    <TapGestureRecognizer  Tapped="TapGestureRecognizer_Tapped_1"/>
                </Frame.GestureRecognizers>
                <StackLayout VerticalOptions="Center" HorizontalOptions="Center">
                    <ImageButton x:Name="ListaFunc"
                                 Source="listaFunc"
                                 BackgroundColor="Transparent"
                                 WidthRequest="60"
                                 HeightRequest="60"
                                 Margin="0,10,0,0"
                    ></ImageButton>

                        <Label Text="Lista de &#10; Funcionarios" HorizontalTextAlignment="Center" TextColor="Black" FontSize="15">

                        </Label>
                </StackLayout>
            </Frame>
    </Grid>
    </ScrollView>
</StackLayout>
```

Fonte: Elaborada pelo autor

A figura 69 mostra a tela gerada após a aplicação do código mostrado nas figuras 66, 67 e 68.

Figura 69 - Tela *Home*



Fonte: Elaborada pelo autor

Ao iniciar a tela *Home,* a função de controle de acesso é chamada, como apresentado na figura 70.

Figura 70 - Ao iniciar a *Home*

```
protected override void OnAppearing() //ao inicializar a pagina, quando ela for exibida
{
    ControleAcesso();
}
```

Fonte: Elaborada pelo autor

A função controle de acesso é responsável por pegar os acessos do usuário logado e habilitar ou bloquear determinadas áreas da aplicação. Os acessos verificados são: acesso à fazenda, acesso para criar e editar áreas, acesso para criar e editar funcionários, como apresentado na figura 71.

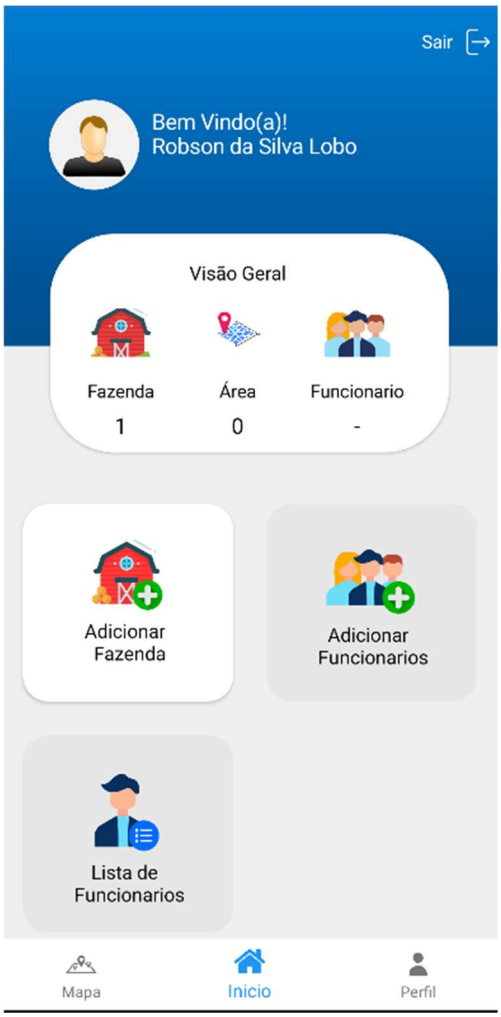
Figura 71 - Função Controle de Acesso

```
public async void ControleAcesso()
{

    //************Inicializa com os cards desabilitados***************
    frameFazenda.IsEnabled = false;
    frameFazenda.HasShadow = false;
    frameFazenda.BackgroundColor = Color.FromHex("e7e7e7");
    frameAddFunc.IsEnabled = false;
    frameAddFunc.HasShadow = false;
    frameAddFunc.BackgroundColor = Color.FromHex("e7e7e7");
    frameListaFunc.IsEnabled = false;
    frameListaFunc.HasShadow = false;
    frameListaFunc.BackgroundColor = Color.FromHex("e7e7e7");


    //*****************************************************************



    List<Logado> logado = await App.database.GetLogado(); //pega o id do usuário logado
    //pega os acessos que o usuário tem
    List<Acesso> Acessos = await App.database.GetAcessoUsuario(logado.ToList()[0].idUsuario,-1);
    //pega as informações do usuário
    Usuario usuario = await App.database.GetByIdUsuario(logado.ToList()[0].idUsuario);






    //verifica se o usuário logado e o dono
    if (usuario.idDono != usuario.idUsuario)
    {
        //coloca o nome completo do usuário na página
        Usuario.Text = usuario.nome + " " + usuario.sobrenome;
        //pega as fazendas que o usuário tem acesso
        List<Fazenda> fazendas =  await App.database.GetFazendaFuncionario(usuario.idUsuario, usuario.idDono);
        //coloca a quantidade de fazendas no quadro de visão geral
        qtdFazenda.Text = fazendas.Count().ToString();


        string str = "";
        List<Area> areas = null;
        List<Usuario> funcionarios = null;


        //colocar na variavel str todas as fazendas em que o usuário tem acesso
        foreach (Fazenda fazenda in fazendas.ToList())
        {
            str = fazenda.idFazenda + ",";
        }


        if (str != "")
        {
           areas = await App.database.GetAreasFazenda(str.Substring(0, str.Length - 1)); //Substring para retirar a ultima virgula
           funcionarios = await App.database.GetFuncionariosFazenda(str.Substring(0, str.Length - 1), usuario.idUsuario, usuario.idDono); //Subst
        }


        qtdArea.Text = "-";  //se o usuário não tiver acesso a nenhuma área
        qtdFuncionario.Text = "-";//se o usuário não tiver acesso a ver funcionario

        foreach (Acesso acesso in Acessos.ToList())
        {
        //Controle dos cards, habilitando os que o usuário tem acesso.
            switch (acesso.idTipo)
            {
                case 1:
                    frameFazenda.IsEnabled = true;
                    frameFazenda.HasShadow = true;
                    frameFazenda.BackgroundColor = Color.FromHex("fff");
                        if(areas != null)
                        {
                            qtdArea.Text = areas.Count().ToString();
                        }

                        break;
                    case 2:
                        frameAddFunc.IsEnabled = true;
                        frameAddFunc.HasShadow = true;
                        frameAddFunc.BackgroundColor = Color.FromHex("fff");
                        if (funcionarios != null)
                        {
                            qtdFuncionario.Text = funcionarios.Count().ToString();
                        }
                        break;
                    case 3:
                        frameListaFunc.IsEnabled = true;
                        frameListaFunc.HasShadow = true;
                        frameListaFunc.BackgroundColor = Color.FromHex("fff");
                        if (funcionarios != null)
                        {
                            qtdFuncionario.Text = funcionarios.Count().ToString();
                        }
                        break;

                }
            }
        }
        else
        {
            //se o usuário for o dono habilitar to
            List<Fazenda> fazendas = await App.database.GetFazendaUsuario(usuario.idDono);
            List<Area> areas = await App.database.GetAreasUsuario(usuario.idDono);
            List<Usuario> funcionarios = await App.database.GetFuncionariosFazenda("-1",usuario.idUsuario, usuario.idDono);

            Usuario.Text = usuario.nome + " " + usuario.sobrenome;
            qtdFazenda.Text = fazendas.Count().ToString();
            qtdArea.Text = areas.Count().ToString();
            qtdFuncionario.Text = funcionarios.Count().ToString();

            frameFazenda.IsEnabled = true;
            frameFazenda.HasShadow = true;
            frameFazenda.BackgroundColor = Color.FromHex("fff");
            frameAddFunc.IsEnabled = true;
            frameAddFunc.HasShadow = true;
            frameAddFunc.BackgroundColor = Color.FromHex("fff");
            frameListaFunc.IsEnabled = true;
            frameListaFunc.HasShadow = true;
            frameListaFunc.BackgroundColor = Color.FromHex("fff");


        }
    }
```

Fonte: Elaborada pelo autor

Quando o acesso não é liberado ao usuário, os cards ficam com uma coloração mais escura. No quadro de visão geral, as informações a que o usuário não tem acesso são apresentadas no lugar do valor um “-” para indicar a ausência da informação, como apresentado na figura 72.

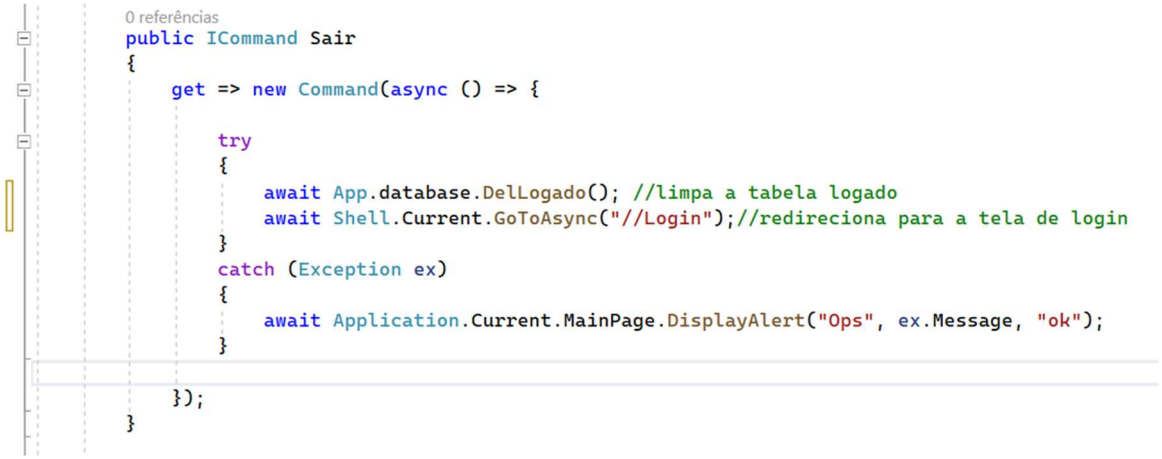
Figura 72 - Visualização de informações sem acesso



Fonte: Elaborada pelo autor

Ao clicar no botão sair, é enviado o comando para a *viewmodel* fazer o redirecionamento do usuário para a tela de login, sendo feita também a exclusão dos dados na tabela logada, como apresentado na figura 73.

Figura 73 - Comando para sair da conta



```
0 referências
public ICommand Sair
{
    get => new Command(async () => {

        try
        {
            await App.database.DelLogado(); //limpa a tabela logado
            await Shell.Current.GoToAsync("//Login");//redireciona para a tela de login
        }
        catch (Exception ex)
        {
            await Application.Current.MainPage.DisplayAlert("Ops", ex.Message, "ok");
        }

    });
}
```

Fonte: Elaborada pelo autor

Quando o usuário entrar na tela de cadastro de fazenda será exibido o título da tela, um campo para inserir o nome da fazenda, um botão para cadastrar a fazenda e um botão para voltar para a home.

A *view* é constituída por uma grid para a organizar os elementos. Nesta grid está o botão para voltar, *labels* contendo o título e o subtítulo, *entry* para que o usuário digite o nome da fazenda, um botão para o cadastro da fazenda e uma lista para a exibição das fazendas cadastradas. Na lista de fazendas, para cada item é armazenado o id do elemento respectivo, para que, ocorrendo qualquer interação com o elemento, a *viewmodel possa* identificar qual elemento está sendo interagido. Os elementos para interação são um botão para cadastro de área e outro para a exclusão da fazenda. Esses botões são *ImageButtons* ou seja, são botões com imagens, como apresentado na figura 74.

Figura 74 - *View* Cadastro Fazenda

```
<!--Grid para organizar os elementos-->
<Grid RowDefinitions="Auto,Auto,Auto">

    <!--elemento contendo o botão para voltar para a home-->
<StackLayout  VerticalOptions="Center" HorizontalOptions="Start" WidthRequest="400" HeightRequest="60" Grid.Row="0" >
    <Frame BackgroundColor="Transparent">
        <Frame.GestureRecognizers>
            <TapGestureRecognizer Tapped="TapGestureRecognizer_Tapped" Command="{Binding Voltar}"/>
        </Frame.GestureRecognizers>
    </Frame>
    <FlexLayout Margin="13,-30,0,0">
        <ImageButton Source="voltar" WidthRequest="15" HeightRequest="20" Command="{Binding Voltar}" BackgroundColor="
        <Label Text="Voltar" TextColor="Black" Margin="3,1,0,0" ></Label>
    </FlexLayout>
</StackLayout>

<StackLayout Grid.Row="1" HorizontalOptions="Center" Margin="0,-40,0,0">
    <Label Text="Cadastrar Fazenda" FontSize="40"  HorizontalTextAlignment="Center" FontFamily="Metropolis" TextColor=
    <Label Text="Insira suas informações para cadastrar" HorizontalTextAlignment="Center" FontSize="15" FontFamily="Met
   <!--campo para o usuário digitar o nome da fazenda-->
    <Frame CornerRadius="30" BackgroundColor="#f2f2f2" HasShadow="False" WidthRequest="310" HeightRequest="15" Margin=
        <Entry x:Name="cadastroNome" Text="{Binding Nome}" HorizontalTextAlignment="Start" FontSize="14" Placeholder="No
    </Frame>
</StackLayout>

    <!--Botão para cadastrar a fazenda-->
<StackLayout Grid.Row="2" Margin="0,20,0,0"  >
    <Button x:Name="NovaFazenda" Text="Cadastrar" BackgroundColor="#2595ef" TextColor="white" WidthRequest="345" Heig
</StackLayout>

<!--Lista de fazendas cadastradas-->
<ListView ItemsSource="{Binding listaFazendas}" Grid.Row="3" HasUnevenRows="True">
        <ListView.ItemTemplate>
            <DataTemplate>
                <ViewCell>

                <StackLayout  Padding="30,10" BackgroundColor="#F1F1F1">
                    <Grid RowDefinitions="100" ColumnDefinitions="Auto,*,Auto,Auto" VerticalOptions="Center">
                        <!--id da fazenda com isVisible false, o usuário não visualisa esse campo-->
                        <Label Text="{Binding idFazenda}" IsVisible="false"></Label>
                        <!--imagem do icone da fazenda-->
                        <Image Source="fazenda" Grid.Row="0" Grid.Column="0" FlowDirection="LeftToRight" HeightRequest='
                        <!--nome da fazenda cadastrada-->
                        <Label Text="{Binding nome}" Grid.Row="0" Grid.Column="1" TextColor="Black" VerticalTextAlignm
                        <!--botão para cadastrar area na fazenda-->
                        <ImageButton x:Name="btnAddArea" Source="addArea" Margin="10,0,0,0" WidthRequest="30" HeightRequ
                        <!--botão para excluir a fazenda-->
                        <ImageButton x:Name="btnExcluir" Source="lixeira" WidthRequest="15" HeightRequest="15" Backgrour
                    </Grid>
                </StackLayout>

            </ViewCell>
            </DataTemplate>
        </ListView.ItemTemplate>
 </ListView>

</Grid>
```

Fonte: Elaborada pelo autor

Os elementos como voltar, cadastrar área e excluir fazenda possuem eventos de click que ao serem acionados chamam funções para executar determinada tarefa. Os botões adicionar e excluir chamam uma função na *viewmodel* passando como parâmetro o identificador da fazenda no qual teve a interação, o botão voltar executa uma função para redirecionar de página, como apresentado na figura 75.

Figura 75 - Código por trás da *view* Cadastro de Fazenda

```
protected override void OnAppearing() //ao inicializar a pagina, quando ela for exibida
{
    var vm = (CadastroFazendaViewModel)BindingContext;
    //chama a função atualizar lista da viewmodel para carregar as fazendas cadastradas
    vm.AtualizarLista.Execute(null);
}


0 referências
private void btnExcluir_Clicked(object sender, EventArgs e)
{
    var button = sender as ImageButton;
    var fazenda = button?.BindingContext as Fazenda;
    var vm = BindingContext as CadastroFazendaViewModel;

    vm?.RemoverFazenda.Execute(fazenda);
}

0 referências
private void btnAddArea_Clicked(object sender, EventArgs e)
{
    var button = sender as ImageButton;
    //pega o id da fazenda
    var fazenda = button?.BindingContext as Fazenda;
    var vm = BindingContext as CadastroFazendaViewModel;
    //chama a função cadastrarArea passando como parametro o id da fazenda
    vm?.CadastrarArea.Execute(fazenda);
}
 private async void TapGestureRecognizer_Tapped(object sender, EventArgs e)
 {
     //botão voltar para Home
     await Shell.Current.GoToAsync("//Home");
 }
```

Fonte: Elaborada pelo autor

Com a estrutura da *view* criada, o usuário irá visualizar a tela de cadastro de fazenda, como apresentado na figura 76.

Figura 76 - Tela de cadastro de fazenda

Fonte: Elaborada pelo autor

Assim que o usuário possuir uma fazenda cadastrada a lista será atualizada, como apresentado na figura 77.

Figura 77 - Lista de fazendas cadastradas

Fonte: Elaborada pelo autor

A lista é composta pelas fazendas do usuário ou pelas fazendas em que o usuário possui acesso. Na função em que a lista é criada e verificado se o usuário logado é o dono. Caso o usuário seja o dono serão exibidas todas as fazendas no qual o idUsuario salvo na tabela fazenda for igual ao idUsuario salvo na tabela logado. Quando o usuário logado não for o dono então serão buscadas as fazendas para as quais o usuário possui permissão. Usuário que não for proprietário não poderá adicionar e nem remover fazenda, como apresentado na figura 78.

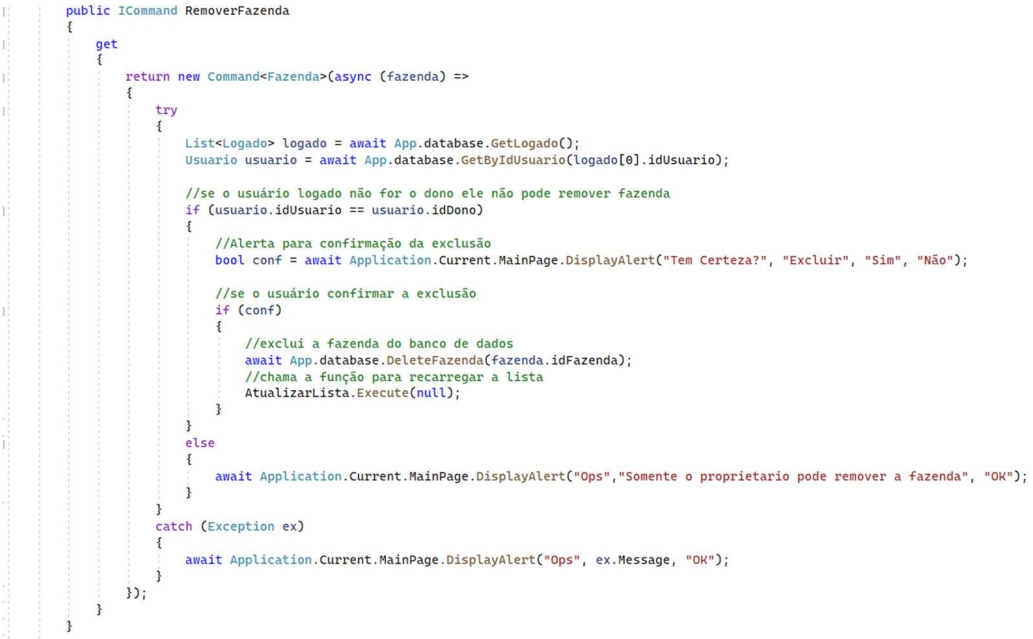
Figura 78 - Função para criar a lista de fazendas

```
public ICommand AtualizarLista
{
    get
    {
        return new Command(async () =>
        {
            try
            {

                List<Fazenda> fazendas = null;
                //pega o usuário logado
                List<Logado> logado = await App.database.GetLogado();
                //pega as informações do usuário
                Usuario usuario = await App.database.GetByIdUsuario(logado.ToList()[0].idUsuario);

                //se o usuário logado não for o dono
                if (usuario.idUsuario != usuario.idDono)
                {
                    //pega as fazendas em que o usuário tem permissão
                    fazendas  = await App.database.GetFazendaFuncionario(logado.ToList()[0].idUsuario);
                }
                else
                {
                    //pega todas as fazendas do usuário
                    fazendas = await App.database.GetFazendaUsuario(logado.ToList()[0].idUsuario);
                }



                //limpa a lista
                listaFazendas.Clear();
                if(fazendas != null)
                {
                    //cria a lista de fazendas cadastradas
                    fazendas.ForEach(i => listaFazendas.Add(i));
                }


            }
            catch (Exception ex)
            {
                await Application.Current.MainPage.DisplayAlert("Ops", ex.Message, "OK");

            }

        });
    }
}
```

Fonte: Elaborada pelo autor

Caso o usuário seja o proprietário e clicar no ícone de remover fazenda a função da RemoverFazenda é chamada. A função disparará um alerta perguntando ao usuário se ele tem certeza de que quer excluir a fazenda, caso a resposta seja sim a fazenda referenciada é excluída do banco de dados e a lista é atualizada, como apresentado na figura 79.

Figura 79 - Função remover fazenda

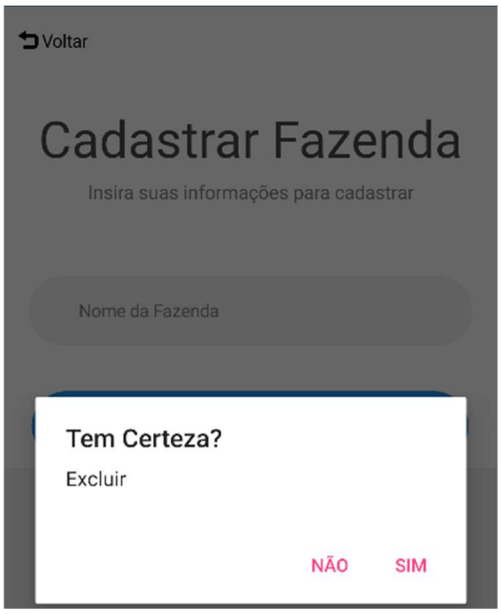


```
public ICommand RemoverFazenda
{
    get
    {
        return new Command<Fazenda>(async (fazenda) =>
        {
            try
            {
                List<Logado> logado = await App.database.GetLogado();
                Usuario usuario = await App.database.GetByIdUsuario(logado[0].idUsuario);

                //se o usuário logado não for o dono ele não pode remover fazenda
                if (usuario.idUsuario == usuario.idDono)
                {
                    //Alerta para confirmação da exclusão
                    bool conf = await Application.Current.MainPage.DisplayAlert("Tem Certeza?", "Excluir", "Sim", "Não");

                    //se o usuário confirmar a exclusão
                    if (conf)
                    {
                        //excluí a fazenda do banco de dados
                        await App.database.DeleteFazenda(fazenda.idFazenda);
                        //chama a função para recarregar a lista
                        AtualizarLista.Execute(null);
                    }
                }
                else
                {
                    await Application.Current.MainPage.DisplayAlert("Ops","Somente o proprietario pode remover a fazenda", "OK");
                }
            }
            catch (Exception ex)
            {
                await Application.Current.MainPage.DisplayAlert("Ops", ex.Message, "OK");
            }
        });
    }
}
```

Fonte: Elaborada pelo autor

A figura 80 mostra o alerta para a confirmação da exclusão da fazenda.

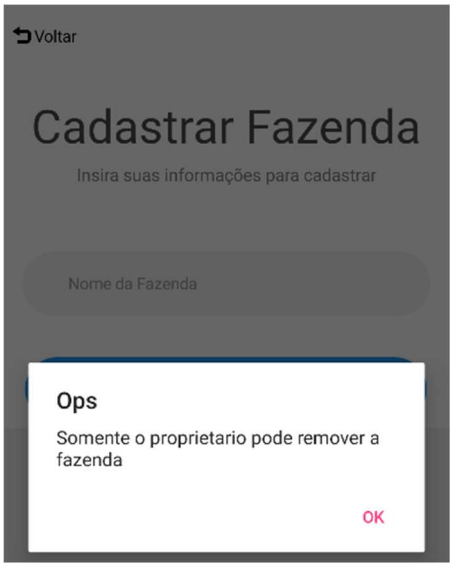
Figura 80 - Alerta de confirmação para excluir fazenda



Fonte: Elaborada pelo autor

Se o usuário não for proprietário e tentar excluir a fazenda um alerta é disparado informando que somente proprietário pode remover a fazenda, como apresentado na figura 81

Figura 81 - bloqueio ao tentar remover fazenda



Fonte: Elaborada pelo autor

Quando o usuário clicar no botão de cadastrar área, a função CadastrarArea é chamada. A função recebe como parâmetro o id da fazenda em que o botão foi clicado. Essa função irá redirecionar para a tela de cadastro de área passando como parâmetro o id da fazenda, como apresentado na figura 82.

Figura 82 - Função Cadastrar Area

```
public ICommand CadastrarArea
{
    get
    {
        return new Command<Fazenda>(async (fazenda) =>
        {
            //redirecionamento para a página de cadastro de area passando como parametro o id da fazenda
            await Shell.Current.GoToAsync($"//CadastroArea?parametro_id={fazenda.idFazenda}");
        });
    }
}
```

Fonte: Elaborada pelo autor

Ao entrar na tela de cadastro de área a função mostrarFazenda é chamada, a função recebe como parâmetro o id da fazenda no qual a área será cadastrada. A partir do id da fazenda é feita uma consulta no banco para retornar os dados da fazenda. Com os dados da fazenda a *view* recebe o nome da fazenda em uma *label* e o id da fazenda em um campo oculto, quando o usuário cadastrar a área, o id da fazenda é pego para que seja feita a referência. Após o cadastro, a função atualizarLista é chamada carregando todas as áreas criadas naquela fazenda, como apresentado na figura 83.

Figura 83 - Inicialização da página cadastro de área

```
//recebimento do id da fazenda
[QueryProperty(nameof(PegarId), "parametro_id")]
    public string PegarId
    {
        set
        {
            int id_parametro = Convert.ToInt32(Uri.UnescapeDataString(value));
            //chama a função mostrar fazenda passando como parametro o id da fazenda
            mostrarFazenda.Execute(id_parametro);
        }
    }
public ICommand mostrarFazenda
{
    get => new Command<int>(async (int id) =>
    {
        try
        {
            //pega as informações da fazenda no banco de dados
            Fazenda model = await App.database.GetByIdFazenda(id);
            /*adiciona o nome da fazenda em uma label para exibir
            para o usuário em qual fazenda ele esta adicionando a area*/
            this.nomeFazenda = "Fazenda " + model.nome;
            this.idFazenda = model.idFazenda;//coloca o id da fazenda em um campo oculto
            AtualizarLista.Execute(null); //chama a função atualizar lista
        }
        catch (Exception ex)
        {
            await App.Current.MainPage.DisplayAlert("ERRO", ex.Message, "OK");
        }

    });
}
```

Fonte: Elaborada pelo autor

A *view* do cadastro de área é constituída por uma grid para a organização dos elementos. Os elementos que estão dentro da grid são campos para inserir o nome da área e sua descrição e uma *listview* elemento para a exibição de todas as áreas já cadastradas para a fazenda em que foi referenciada. Em cada elemento da lista e exibido o nome da área e botões para interagir com a área cadastrada, como apresentado na figura 84.

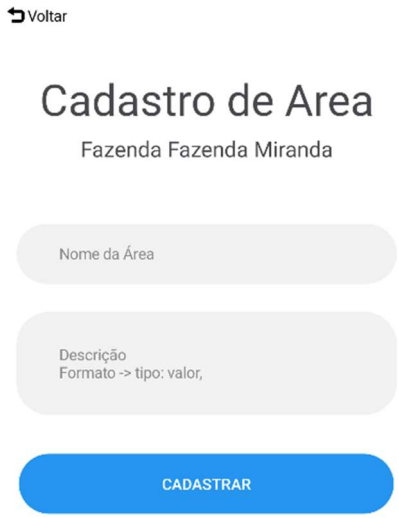
Figura 84 - *view* do cadastro de área

```
<!--grid para organizar os elementos-->
<Grid RowDefinitions="Auto,Auto,Auto">
    <!--botão voltar-->
    <StackLayout  VerticalOptions="Center" HorizontalOptions="Start" WidthRequest="400" HeightRequest="60" Grid.Row="0" ...>
    <StackLayout Grid.Row="1" HorizontalOptions="Center" Margin="0,-40,0,0">
        <Label Text="Cadastro de Area" FontSize="40"  HorizontalTextAlignment="Center" FontFamily="Metropolis" TextColor="#4A484D" Ma
        <!--Campo para receber o nome da fazenda-->
        <Label Text="{Binding nomeFazenda}" FontSize="20"  HorizontalTextAlignment="Center" FontFamily="Metropolis" TextColor="#4A484
        <!--campo para receber o id da fazenda-->
        <Label Text="{Binding idFazenda}" IsVisible="false"></Label>
        <!--campo para colocar o nome da área-->
        <Frame CornerRadius="30" BackgroundColor="#f2f2f2" HasShadow="False" WidthRequest="310" HeightRequest="15" Margin="0,50,0,10"
            <Entry x:Name="cadastroNome" Text="{Binding nomeArea}" HorizontalTextAlignment="Start" FontSize="14" Placeholder="Nome da
        </Frame>
        <!--campo para colocar a descrição da área-->
        <Frame CornerRadius="30" BackgroundColor="#f2f2f2" HasShadow="False" WidthRequest="310" HeightRequest="55" Margin="0,10,0,10"
            <Editor x:Name="cadastroDesc" Text="{Binding descArea}" AutoSize="TextChanges" FontSize="14" Placeholder="Descrição" Height
        </Frame>
    </StackLayout>
    <!--botão para salvar as informações-->
    <StackLayout Grid.Row="2" Margin="0,20,0,0"  >
        <Button x:Name="NovaArea" Text="Cadastrar" BackgroundColor="#2595ef" TextColor="white" WidthRequest="345" HeightRequest="55
    </StackLayout>

    <!--lista para mostrar as áreas cadastradas-->
    <ListView ItemsSource="{Binding listaArea}" Grid.Row="3" HasUnevenRows="True">
        <ListView.ItemTemplate>
            <DataTemplate>
                <ViewCell>
                    <StackLayout  Padding="30,10" BackgroundColor="#F1F1F1">
                        <Grid RowDefinitions="100" ColumnDefinitions="Auto,*,Auto,Auto,Auto,Auto" VerticalOptions="Center">
                            <!--id da área em um campo oculto-->
                            <Label Text="{Binding idArea}" IsVisible="false"></Label>
                            <!--imagem representativa da área-->
                            <Image Source="cerca" Grid.Row="0" Grid.Column="0" FlowDirection="LeftToRight" HeightRequest="50" WidthRequest="
                            <!--nome da área-->
                            <Label Text="{Binding nome}" Grid.Row="0" Grid.Column="1" TextColor="Black" VerticalTextAlignment="Center" Hor
                            <!--botão para editra a descrição da área-->
                            <ImageButton x:Name="btnEditArea" Source="editar" Margin="10,0,0,0" WidthRequest="25" HeightRequest="25" Backgro
                            <!--botão para adicionar a região da área-->
                            <ImageButton x:Name="btnAddGeo" Source="addPoint" Margin="0,0,10,0" WidthRequest="22" HeightRequest="22" Backgro
                            <!--botão para excluir a área-->
                            <ImageButton x:Name="btnExcluir" Source="lixeira" WidthRequest="15" HeightRequest="15" BackgroundColor="Transp

                        </Grid>
                    </StackLayout>

                </ViewCell>
            </DataTemplate>
        </ListView.ItemTemplate>
    </ListView>

</Grid>
```

Fonte: Elaborada pelo autor

A figura 85 apresenta a tela de cadastro de área que o usuário irá ter acesso.

Figura 85 - Tela de cadastro de área



Fonte: Elaborada pelo autor

O usuário quando preencher as informações e clicar no botão cadastrar, esse botão irá gerar um evento de *click,* executando assim o código que está presente

dentro do evento. Ao clicar é chamada a função novaArea, onde a classe irá receber as informações preenchidas pelo usuário e armazenar no banco de dados. Após o cadastro é emitido um alerta para usuário informando-o de que a área foi cadastrada. Então a lista de áreas é atualizada, fazendo com que o usuário possa interagir com a nova área cadastrada, como apresentado na figura 86.

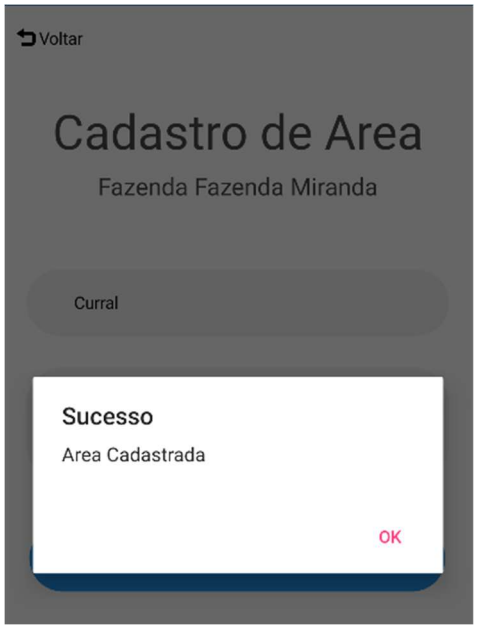
Figura 86 - Cadastro de uma nova área

```
public ICommand novaArea
{
    get => new Command(async () =>
    {
        try
        {
            //a classe recebe as informações digitadas pelo usuário
            Area model = new Area()
            {
                nome = this.txtArea,
                descricao = this.txtDesc,
                idFazenda = this.idfazenda
            };
            //insere as informações no banco de dados
            await App.database.InsertArea(model);
            //emite um alerta para usuário informando do sucesso do cadastro
            await Application.Current.MainPage.DisplayAlert("Sucesso", "Area Cadastrada", "ok");
            //chama a função para atualizar a listaao
            AtualizarLista.Execute(null);
        }
        catch (Exception ex)
        {
            await Application.Current.MainPage.DisplayAlert("Ops", ex.Message, "ok");
        }
    });
}
```

Fonte: Elaborada pelo autor

A figura 87 apresenta o alerta emitido ao usuário após o cadastro da área.

Figura 87 - Alerta emitido após cadastro da área



Fonte: Elaborada pelo autor

A figura 88 apresenta a função AtualizarLista.

Figura 88 - Função atualizar lista

```
public ICommand AtualizarLista
{
    get
    {
        return new Command(async () =>
        {
            try
            {
                //Pega todas as áreas cadastradas para a fazenda em referencia
                List<Area> tmp = await App.database.GetAllRowsArea(this.idFazenda);
                //Limpa a lista
                listaArea.Clear();
                //Para cada dado retornado do banco é adicionado na lista
                tmp.ForEach(i => listaArea.Add(i));

            }
            catch (Exception ex)
            {
                await Application.Current.MainPage.DisplayAlert("Ops", ex.Message, "OK");

            }

        });
    }
}
```

Fonte: Elaborada pelo autor

Para cada item da lista são disponibilizados três botões sendo eles para editar as informações da área, adicionar a demarcação da área e excluir a área, como apresentado na figura 89.

Figura 89 - Lista de áreas

Fonte: Elaborada pelo autor

Clicando no botão de editar área a tela é redirecionada para uma tela onde o usuário terá acesso à descrição da área, clicando no botão para adicionar a demarcação, a tela é redirecionada para um mapa para que o usuário possa selecionar a área e clicando no botão da lixeira é disparado um alerta para o usuário confirmar a exclusão. Caso seja confirmado, a área é excluída, todos os pontos que compõem a região da área são excluídos e a lista é atualizada para que o usuário veja a nova lista, como apresentado na figura 90.

Figura 90 - Funções de editar, excluir e demarcar área

```
public ICommand EditarArea
{
    get
    {
        return new Command<Area>(async (area) =>
        {
            //redireciona para a página para editar a descrição da área passando o id da área como parametro
            await Shell.Current.GoToAsync($"//EditAreaModal?parametro_id={area.idArea}");
        });
    }
}

1 referência
public ICommand AddGeo
{
    get
    {
        return new Command<Area>(async (area) =>
        {
            //redireciona para a página para selecionar a área passando o id da área como parametro
            await Shell.Current.GoToAsync($"//CadastroPontos?parametro_id={area.idArea}");
        });
    }
}

1 referência
public ICommand RemoverArea
{
    get
    {
        return new Command<Area>(async (area) =>
        {
            try
            {
                //envia um alerta para a confirmação da exclusão
                bool conf = await Application.Current.MainPage.DisplayAlert("Tem Certeza?", "Excluir", "Sim", "Não");

                if (conf)//se o usuário confirmar
                {
                    //a área e deletada do banco de dados
                    await App.database.DeleteArea(area.idArea);
                    //todos os pontos adicionados a area e excluido
                    await App.database.DellPontosArea(area.idArea);
                    //atualiza a área
                    AtualizarLista.Execute(null);
                }

            }
            catch (Exception ex)
            {
                await Application.Current.MainPage.DisplayAlert("Ops", ex.Message, "OK");

            }

        });
    }
}
```

Fonte: Elaborada pelo autor

A *view* da tela de edição da área é composta por uma grid para a organização dos elementos, sendo eles *labels* para e exibição de textos, um campo editor para exibir e editar a descrição da área e um botão editar para a confirmação das alterações, como apresentado na figura 91.

Figura 91 - *View* página editor de área

```
<Grid RowDefinitions="Auto,Auto,Auto">
    <!--botão voltar-->
    <StackLayout  VerticalOptions="Center" HorizontalOptions="Start" WidthRequest="400" HeightRequest="60" Grid.Row="0" ...>
    <!--Grid para organizar os elementos-->
    <StackLayout Grid.Row="1" HorizontalOptions="Center" Margin="0,-40,0,0">
        <!--Labels para exibição de textos-->
        <Label Text="Editar Area" FontSize="40"  HorizontalTextAlignment="Center" FontFamily="Metropolis" TextColor="#4A484D"
        <Label Text="{Binding mostrarNome}" FontSize="20"  HorizontalTextAlignment="Center" FontFamily="Metropolis" TextColor=
        <Label Text="{Binding idArea}" IsVisible="false"></Label>
        <!--Campo para exibir a descrição da área-->
        <Frame CornerRadius="30" BackgroundColor="#f2f2f2" HasShadow="False" WidthRequest="310" HeightRequest="240" Margin="0
            <Editor x:Name="cadastroDesc" Text="{Binding descArea}" AutoSize="TextChanges" FontSize="14" Placeholder="Descriçã
        </Frame>
    </StackLayout>
    <!--Botão para salvar as alterações-->
    <StackLayout Grid.Row="2" Margin="0,20,0,0"  >
        <Button x:Name="EditArea" Text="Editar" BackgroundColor="#2595ef" TextColor="white" WidthRequest="345" HeightReques
    </StackLayout>
</Grid>
```

Fonte: Elaborada pelo autor

Ao entrar na tela de edição de área, a função mostrarArea recebe o id enviado à página e traz do banco todas as informações pertinentes a ela, exibindo-as na tela para que o usuário possa editar, como apresentado na figura 92.

Figura 92 - Mostrar as informações da área para edição

```
public string PegarId
{
    set
    {
        int id_parametro = Convert.ToInt32(Uri.UnescapeDataString(value));
        //chama a função para mostrar as informações da área
        mostrarArea.Execute(id_parametro);
    }
}
1 referência
public ICommand mostrarArea
{
    get => new Command<int>(async (int id) =>
    {
        try
        {
            //pega no banco as informações da área
            Area model = await App.database.GetByIdArea(id);
            this.mostrarNome =  model.nome; //mostra na tela o nome da área
            this.idArea = model.idArea; //guarda o id da área
            //formata a descrição para a melhor visualização
            this.descArea = String.Format(model.descricao.Replace(",", "\n\n"));
        }
        catch (Exception ex)
        {
            await App.Current.MainPage.DisplayAlert("ERRO", ex.Message, "OK");
        }

    });
}
```

Fonte: Elaborada pelo autor

Somente o campo da descrição pode ser editado, o nome da área não é disponibilizado para a edição. A figura 85 mostra a formatação da descrição para a melhor visualização dos dados. A partir das vírgulas que separam as informações é realizada uma troca pelo comando ‘/n’ fazendo assim a quebra de linha, como apresentado na figura 93.

Figura 93 - Tela para editar a descrição da área

Fonte: Elaborado pelo autor

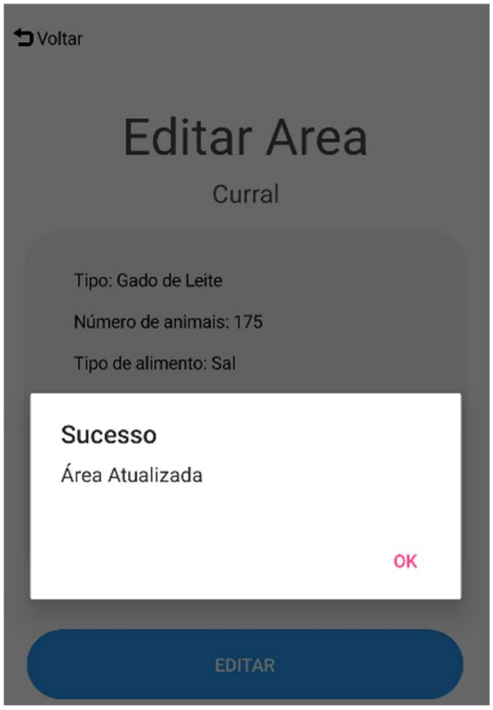
Quando o usuário clicar no botão editar, a informação alterada é recebida e formatada para armazená-la no banco de dados. As quebras de linhas são trocadas por vírgula novamente. Assim, atualizando a informação no banco de dados. É disparado um alerta informando o usuário que a alteração foi feita, como apresentado na figura 94 e 95.

Figura 94 - Edição da descrição

```
public ICommand editarArea
{
    get => new Command (async () =>
    {
        try
        {
            //recebe o campo da descrição trocando as quebras de linha por ','
            string descricao = String.Format(this.descArea.Replace("\n\n", ","));
            //atualiza a informação no banco
            await App.database.UpdateArea(this.idArea, descricao);
            //dispara um alerta informando que a alteração foi realizada
            await App.Current.MainPage.DisplayAlert("Sucesso", "Área Atualizada", "OK");
        }
        catch (Exception ex)
        {
            await App.Current.MainPage.DisplayAlert("ERRO", ex.Message, "OK");
        }

    });
}
```

Fonte: Elaborada pelo autor

Figura 95 - Alerta de atualização de descrição



Fonte: Elaborada pelo autor

Na tela para selecionar a área criada é exibido um campo para que o usuário possa procurar pelo endereço da região, e um mapa exibindo a sua localização, como apresentado na figura 96.

Figura 96 - *View* adicionar região da área

```
<ContentPage.Content>
    <StackLayout>
        <StackLayout  VerticalOptions="Center" HorizontalOptions="Start" WidthRequest="400" HeightRequest="60" Grid.Row="
            <Frame BackgroundColor="Transparent" HeightRequest="60"...>
            <StackLayout Margin="13,-30,0,15">
                <FlexLayout >
                    <ImageButton Source="voltar" WidthRequest="15" HeightRequest="20" Command="{Binding Voltar}" Backgrou
                    <Label Text="Voltar" TextColor="Black" Margin="3,1,0,0" ></Label>
                    <!--Nome da área-->
                    <Label Text="{Binding nomeArea}" Margin="45,-5,0,0" x:Name="txtArea"  FontSize="Large" TextColor="B
                    <!--id da área em um campo oculto-->
                    <Label Text="{Binding idArea}" IsVisible="false" x:Name="idarea"></Label>

                </FlexLayout>
            </StackLayout>
        </StackLayout>
        <StackLayout HorizontalOptions="Center" Orientation="Horizontal">
            <!--Campo para receber o endereço da região-->
            <Frame CornerRadius="30" BackgroundColor="#f2f2f2" HasShadow="False" WidthRequest="310" HeightRequest="15" 
                <Entry x:Name="enderecoMapa" HorizontalTextAlignment="Start" FontSize="14" Placeholder="Endereço" TextCol
            </Frame>
            <!--botão para realizar a busca pela região-->
            <ImageButton x:Name="iconLupa" Source="lupa" BackgroundColor="Transparent" WidthRequest="20" Clicked="iconLu
        </StackLayout>
        <!--mapa-->
        <maps:Map IsShowingUser="True" x:Name="map" MapClicked="map_MapClicked"/>
    </StackLayout>
</ContentPage.Content>
```

Fonte: Elaborada pelo autor

Ao iniciar a tela, a localização do usuário é adiquirida e centralizada no mapa, e verificado no banco de dados se a área em questão possui algum ponto cadastrado. Caso exista, é criado um polígono com todos os pontos. O polígono então é inserido no mapa, como apresentado na figura 97.

Figura 97 - Informações no mapa

```
protected override async void OnAppearing() //ao inicializar a pagina, quando ela for exibida
{
    var vm = (CadastroPontosViewModel)BindingContext;


    //***************Centralizar no Usuário*********************************

    var request = new GeolocationRequest(GeolocationAccuracy.Best, TimeSpan.FromSeconds(10));

    var location = await Geolocation.GetLocationAsync(request);

    Position pUsuario = new Position(location.Latitude, location.Longitude);

    MapSpan MapSpan = MapSpan.FromCenterAndRadius(pUsuario, Distance.FromKilometers(.444));
    map.MoveToRegion(MapSpan);
    //**********************************************************************

    //pega os pontos cadastrados no banco
    List<Pontos> model = await App.database.getPontos(Convert.ToInt32(idarea.Text));

    if (model.Count > 0) //se retornar algum registro do banco
    {
        Polygon polygon1 = new Polygon //cria um novo poligono
        {
            //estilo do poligono, borda e preenchimento
            StrokeWidth = 8,
            StrokeColor = Color.FromHex("#00BFFF"),
            FillColor = Color.FromHex("#87CEFA")
        };

        foreach (Pontos element in model.ToList())
        {
            //para cada ponto cadastrado no banco ele e adicionado no objeto poligono
            Position p = new Position((double)element.latitude, (double)element.longitude);
            polygon1.Geopath.Add(p);
        };
        //adiciona o objeto poligono no mapa
        map.MapElements.Add(polygon1);
    }
    else
    {
        //se não tiver registro no banco os elementos são removidos do mapa
        map.MapElements.Clear();

    }


}
```

Fonte: Elaborada pelo autor

A figura 98 mostra a tela exibida para o usuário.

Figura 98 - Tela de cadastro da geofencing

Fonte: Elaborada pelo autor

Caso o usuário queira procurar um endereço específico, é possível digitar o endereço no campo disponibilizado e clicar na lupa para fazer a busca. Ao clicar na lupa irá disparar um evento no qual receberá o endereço digitado pelo usuário e pegar a posição do endereço, latitude e longitude. Essa localização é enviada para o mapa para que ele mova até ela, como apresentado na figura 99.

Figura 99 - Busca pelo endereço

```
private async void iconLupa_Clicked(object sender, EventArgs e)
{
    //recebe o endereço fornecido pelo usuárop
    string endereco = enderecoMapa.Text.Trim();

    Geocoder geoCoder = new Geocoder();

    //pega a latitude e longitude da localização procurada
    IEnumerable<Position> approximateLocations = await geoCoder.GetPositionsForAddressAsync(endereco);
    Position position = approximateLocations.FirstOrDefault();

    //pega a posição
    Position pEndereco = new Position(position.Latitude, position.Longitude);

    //move para a localização inserida pelo usuário
    MapSpan MapSpan = MapSpan.FromCenterAndRadius(pEndereco, Distance.FromKilometers(.444));
    map.MoveToRegion(MapSpan);


}
```

Fonte: Elaborada pelo autor

Para a criação do polígono o usuário terá que clicar no mapa até compor toda a região desejada. Conforme o usuário clica no mapa, o polígono irá tomando forma imediatamente. Para cada clique no mapa é chamada uma função que recebe a latitude e longitude em que o usuário as clicou e armazena no banco de dados. Também é realizada a consulta de todos os pontos já cadastrados para que o polígono seja criado, como apresentado na figura 100 e 101.

Figura 100 - Evento de *click* no mapa

```
private async void  map_MapClicked(object sender, Xamarin.Forms.Maps.MapClickedEventArgs e)
{

    var vm = (CadastroPontosViewModel)BindingContext;
    //monta uma string contendo a latitude e longitude
    string posicao = Convert.ToString(e.Position.Latitude.ToString().Replace(",",".") + "," + e.Position.Longitude.ToString().Replace(",", "."));
    //chama a função passando a string contendo a localização como parametro
    vm.cadastrarPonto.Execute(posicao);
    //pega no banco todos os pontos cadastrados naquela área
    List<Pontos> model = await App.database.getPontos(Convert.ToInt32(idarea.Text));
    //cria um novo objeto polígono
    Polygon polygon1 = new Polygon
    {
        StrokeWidth = 8,
        StrokeColor = Color.FromHex("#00BFFF"),
        FillColor = Color.FromHex("#87CEFA")
    };
    //para cada ponto cadastrado
    foreach (Pontos element in model.ToList())
    {
        //adiciona o ponto cadastrado no polígono
        Position p = new Position((double)element.latitude, (double)element.longitude);
        polygon1.Geopath.Add(p);
    };
    //adiciona o polígono no mapa
    map.MapElements.Add(polygon1);
}
```

Fonte: Elaborada pelo autor

Figura 101 - Inserção dos pontos no banco

```
public ICommand cadastrarPonto
{
    get => new Command<string>(async (string posicao) => {
        try
        {
            //pega a string recebida por parametro e separa a latitude e longitude
            string latitude = posicao.Split(',')[0];
            string longitude = posicao.Split(',')[1];

            //insere as informações na classe
            Pontos model = new Pontos()
            {
                idArea = this.idArea,
                // Utiliza-se o CultureInfo para ponto seja considerar o . como separador decimal
                latitude = Convert.ToDouble(latitude, new CultureInfo("en-US")),
                longitude = Convert.ToDouble(longitude, new CultureInfo("en-US"))
            };
            //insere a informação no banco
            await App.database.InsertPonto(model);

        }
        catch (Exception ex)
        {
            await Application.Current.MainPage.DisplayAlert("Ops", ex.Message, "ok");
        }
    });
}
```

Fonte: Elaborada pelo autor

As figuras 102, 103, 104 e 105 mostram os processos de criação do polígono. Só é possível visualizar o desenho da área se formando após a criação de dois pontos distintos.

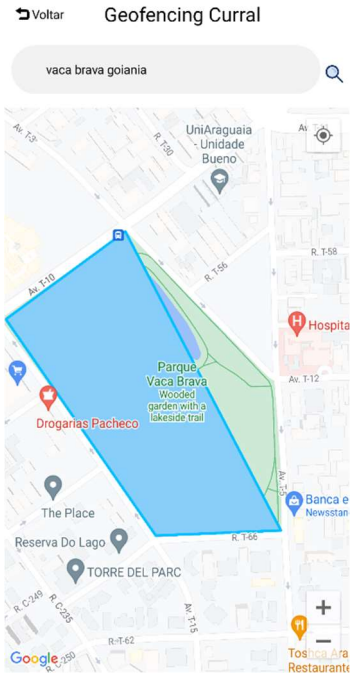
Figura 102 - Processo de criação da *geofencing* 2 pontos

Fonte: Elaborada pelo autor

Figura 103 - Processo de criação da *geofencing* 3 pontos

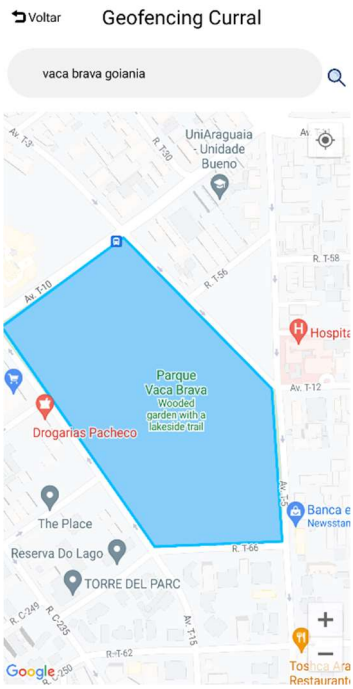
Fonte: Elaborada pelo autor

Figura 104 - Processo de criação da *geofencing* 4 pontos



Fonte: Elaborada pelo autor

Figura 105 - Processo de criação da *geofencing* 5 pontos



Fonte: Elaborada pelo autor

A tela de cadastro de funcionário é bem semelhante à tela de cadastro inicial, a *view* é composta por campos para o usuário preencher, campos como nome, sobrenome, e-mail e senha e confirmação de senha, como apresentado na figura 106.

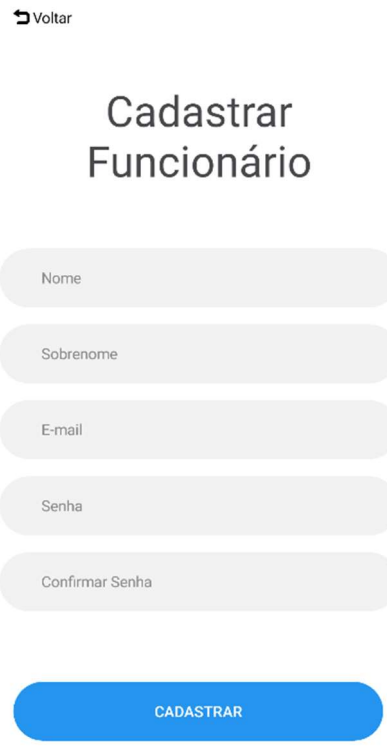
Figura 106 - *View* cadastro de funcionário

```
<StackLayout Margin="10">
    <Grid RowDefinitions="Auto,Auto">
        <!--botão voltar-->
        <StackLayout  VerticalOptions="Center" HorizontalOptions="Start" WidthRequest="400" HeightRequest="60" Grid.Row="0" ...>
        <StackLayout Grid.Row="1" HorizontalOptions="Center" Margin="0,-30,0,0" >
            <!--Titulo-->
            <Label Text="Cadastrar Funcionário" FontSize="40" HorizontalTextAlignment="Center" FontFamily="Metropolis" TextColor=[
            <!--Campo nome do funcionario-->
            <Frame CornerRadius="30" BackgroundColor="#f2f2f2" HasShadow="False" WidthRequest="310" HeightRequest="15" Margin="6
                <Entry x:Name="cadastroNome" Text="{Binding Nome}"  HorizontalTextAlignment="Start" FontSize="14" Placeholder="No
                    <Entry.Keyboard>
                        <Keyboard x:FactoryMethod="Create">
                            <x:Arguments>
                                <KeyboardFlags>Suggestions,CapitalizeWord</KeyboardFlags>
                            </x:Arguments>
                        </Keyboard>
                    </Entry.Keyboard>
                </Entry>
            </Frame>
            <!--Campo sobrenome do funcionario-->
            <Frame CornerRadius="30" BackgroundColor="#f2f2f2" HasShadow="False" WidthRequest="310" HeightRequest="15" Margin="6
                <Entry x:Name="cadastroSobranome"  Text="{Binding Sobrenome }" HorizontalTextAlignment="Start" FontSize="14" Plac
                    <Entry.Keyboard>
                        <Keyboard x:FactoryMethod="Create">
                            <x:Arguments>
                                <KeyboardFlags>Suggestions,CapitalizeWord</KeyboardFlags>
                            </x:Arguments>
                        </Keyboard>
                    </Entry.Keyboard>
                </Entry>
            </Frame>
            <!--Campo email do funcionario-->
            <Frame CornerRadius="30" BackgroundColor="#f2f2f2" HasShadow="False" WidthRequest="310" HeightRequest="15" Margin="6
                <Entry x:Name="cadastroEmail" Text="{Binding Email }" HorizontalTextAlignment="Start" FontSize="14" Placeholder="E
            </Frame>
            <!--senha do funcionario-->
            <Frame CornerRadius="30" BackgroundColor="#f2f2f2" HasShadow="False" WidthRequest="310" HeightRequest="15" Margin="6
                <Entry x:Name="cadastroSenha" Text="{Binding Senha }" HorizontalTextAlignment="Start" FontSize="14" IsPassword="T
            </Frame>
            <!--senha do funcionario-->
            <Frame CornerRadius="30" BackgroundColor="#f2f2f2" HasShadow="False" WidthRequest="310" HeightRequest="15" Margin="(
                <Entry x:Name="cadastroSenha" Text="{Binding Senha }" HorizontalTextAlignment="Start" FontSize="14" IsPassword="T
            </Frame>
            <!--confirmar senha do funcionario-->
            <Frame CornerRadius="30" BackgroundColor="#f2f2f2" HasShadow="False" WidthRequest="310" HeightRequest="15" Margin="(
                <Entry x:Name="cadastroConfSenha"  VisualElement.BackgroundColor  nha }" HorizontalTextAlignment="Start" FontSize="14" IsPas
            </Frame>

        </StackLayout>
        <!--botão para salvar as informações no banco de dados-->
        <StackLayout Grid.Row="2" Margin="0,50,0,0"  >
            <Button x:Name="btnCadastrar" Text="Cadastrar" BackgroundColor="#2595ef" TextColor="white" WidthRequest="345" Heig
        </StackLayout>
    </Grid>
</StackLayout>
```

Fonte: Elaborada pelo autor

A figura 107 mostra a tela que o usuário irá visualizar ao entrar no cadastro de funcionário.

Figura 107 - Tela de cadastro de funcionário



Fonte: Elaborada pelo autor

Assim que o usuário preencher todos os campos e clicar no botão cadastrar, a função NovoUsuario é chamada. Esta função irá validar as informações, verificar se todos os campos foram preenchidos e se as senhas são iguais, se todas as condições forem aceitas o usuário é adicionado no banco. Para que o usuário seja relacionado ao proprietário, a partir do id do usuário logado e buscado o idDono é adicionado ao cadastro do funcionário após o êxito em inserir as informações do novo usuário ao banco, é emitido um alerta para informar ao usuário o sucesso do cadastro, como apresentado na figura 108.

Figura 108 - Registro no banco do novo funcionário

```
public ICommand NovoUsuario
{
    get => new Command(async () => {

        try
        {

            //A classe usuário recebe as informações do usuário
            Usuario model = new Usuario()
            {
                nome = this.nome,
                sobrenome = this.sobrenome,
                senha = this.senha,
                email = this.email,
                ativo = 1
            };

            //verifica se os campos foram preenchidos
            if (nome != null && sobrenome != null && senha != null && email != null && confSenha != null)
            {
                //verifica se as senhas são iguais
                if (senha == confSenha)
                {
                    //verifica se o e-mail informado ja esta cadastrado
                    List<Usuario> verEmail = await App.database.verificarEmail(email);
                    if (verEmail.Count == 0)
                    {
                        //insere as informações do usuario no banco
                        await App.database.CadastroUsuario(model);

                        //pega o id do dono para inserir na informação do usuário
                        List<Logado> logado = await App.database.GetLogado();
                        Usuario idDono = await App.database.GetByIdUsuario(logado.ToList()[0].idUsuario);
                        List<Usuario> idUsuario = await App.database.GetUsuario(email,senha);
                        //adiciona o id do proprietario no registro do usuário
                        await App.database.UpdateDono(idDono.idDono, idUsuario.ToList()[0].idUsuario);

                        await App.Current.MainPage.DisplayAlert("Sucesso", "Funcionário Cadastrado", "OK");
                    }
                    else
                    {
                        await Application.Current.MainPage.DisplayAlert("Falha", "E-mail já cadastrado", "Ok");
                    }

                }
                else
                {
                    await Application.Current.MainPage.DisplayAlert("Falha", "Senhas diferentes", "ok");
                }
            }
            else
            {
                await Application.Current.MainPage.DisplayAlert("Falha", "Preencha Todos os campos", "ok");
            }


        }
        catch (Exception ex)
        {
            await Application.Current.MainPage.DisplayAlert("Ops", ex.Message, "ok");
        }


    });
}
```

Fonte: Elaborada pelo autor

A figura 109 mostra o alerta informativo após o cadastro do usuário.

Figura 109 - Alerta de funcionário cadastrado

Fonte: Elaborada pelo autor

A tela lista de funcionários é composta por uma grid para a organização dos elementos, o título e a lista onde são exibidos os funcionários. Cada item da lista é composto por uma imagem representativa do funcionário, nome do funcionário e dois botões, um para atribuição de permissão e outro para a exclusão do funcionário, como apresentado na figura 110.

Figura 110 - *View* lista de funcionário

```
<!--Grid para organizar os elementos-->
<Grid RowDefinitions="Auto,Auto,Auto">
    <!--Botão para voltar-->
    <StackLayout  VerticalOptions="Center" HorizontalOptions="Start" WidthRequest="400" HeightRequest="60" Grid.Row="0" ...>
    <!--Titulo da página-->
    <StackLayout Grid.Row="1" HorizontalOptions="Center" Margin="0,-40,0,0">
        <Label Text="Lista de Funcionarios" FontSize="30"  HorizontalTextAlignment="Center" FontFamily="Metropolis" TextColor="#4A484D" Margin="0,50,0,0"></Label>
    </StackLayout>
    <!--Lista de funcionários-->
    <ListView ItemsSource="{Binding listaFuncionario}" Grid.Row="2" HasUnevenRows="True" Margin="0,20,0,0">
        <ListView.ItemTemplate>
            <DataTemplate>
                <ViewCell>
                    <StackLayout  Padding="30,10" BackgroundColor="#F1F1F1">
                        <Grid RowDefinitions="100" ColumnDefinitions="Auto,*,Auto,Auto,Auto,Auto" VerticalOptions="Center">
                            <!--guarda o id do funcionário-->
                            <Label Text="{Binding idFuncionario}" IsVisible="false"></Label>
                            <!--imagem simbolisando os funcionários-->
                            <Image Source="funcionarios" Grid.Row="0" Grid.Column="0" FlowDirection="LeftToRight" HeightRequest="50" WidthRequest="50" ></Image>
                            <!--nome do funcionário-->
                            <Label Text="{Binding nome}" Grid.Row="0" Grid.Column="1" TextColor="Black" VerticalTextAlignment="Center" HorizontalTextAlignment="Cer
                            <!--botão para adicionar permissão-->
                            <ImageButton x:Name="btnAddPerm" Source="editar" Margin="10,0,0,0" WidthRequest="25" HeightRequest="25" BackgroundColor="Transparent" C
                            <!--botão para excluir o funcionário-->
                            <ImageButton x:Name="btnExcluir" Source="lixeira" WidthRequest="15" HeightRequest="15" BackgroundColor="Transparent" Grid.Row="0" Grid.

                        </Grid>
                    </StackLayout>

                </ViewCell>
            </DataTemplate>
        </ListView.ItemTemplate>
    </ListView>

</Grid>
```

Fonte: Elaborada pelo autor

Ao acessar a tela de lista de funcionários, a função para atualizar a lista de funcionários é chamada, como apresentado na figura 111.

Figura 111 - Chamada da função de atualizar lista de funcionários

```
protected override void OnAppearing() //ao inicializar a pagina, quando ela for exibida
{
    var vm = (ListaFuncionarioViewModel)BindingContext;
    //chama a função atualizar lista na viewmodel
    vm.AtualizarLista.Execute(null);

}
```

Fonte: Elaborada pelo autor

A função de atualizar lista é responsável por pegar todos os funcionários no qual o usuário logado possui permissão para editar e atribuir acesso e exibi-los na lista, como apresentado na figura 112.

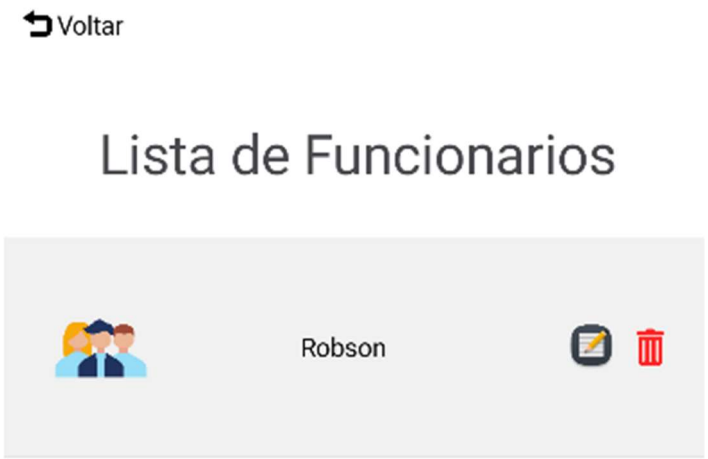
Figura 112 - Função para carregar a lista de funcionários

```
public ICommand AtualizarLista
{
    get
    {
        return new Command(async () =>
        {
            try
            {
                //pega o id do usuário logado
                List<Logado> logado = await App.database.GetLogado();
                //pega as informações do usuário
                Usuario usuario = await App.database.GetByIdUsuario(logado.ToList()[0].idUsuario);
                //pega todos os funcionarios do proprietario das fazendas em que o usuário logado tem permissão
                List<Fazenda> fazendas = await App.database.GetFazendaFuncionario(usuario.idUsuario,usuario.idDono);

                string str = "";


                //colocar na variavel str todas as fazendas em que o usuário tem acesso
                foreach (Fazenda fazenda in fazendas.ToList())
                {
                    str = fazenda.idFazenda + ",";
                }

                List<Usuario> funcionarios = await App.database.GetFuncionariosFazenda(str.Substring(0, str.Length - 1), usuario.idUsuario, usuario.idDono);
                //limpa a lista
                listaFuncionario.Clear();
                //para cada registro de funcionario no banco e adicionado na lista
                funcionarios.ForEach(i => listaFuncionario.Add(i));

            }
            catch (Exception ex)
            {
                await Application.Current.MainPage.DisplayAlert("Ops", ex.Message, "OK");

            }

        });
    }
}
```

Fonte: Elaborada pelo autor

A figura 113 apresenta a tela com os funcionários inseridos na lista.

Figura 113 - Tela com a lista de funcionários



Fonte: Elaborada pelo autor

Ao clicar na lixeira a função RemoverFuncionario e chamada. Como apresentado na figura 114

Figura 114 - Função para remover funcionário

```
public ICommand RemoverFuncionario
{
    get
    {
        return new Command<Usuario>(async (usuario) =>
        {
            try
            {
                //Alerta de confirmação para exclusão
                bool conf = await Application.Current.MainPage.DisplayAlert("Tem Certeza?", "Excluir", "Sim", "Não");

                if (conf) //se form confirmado
                {
                    //o usuário e setado como ativo = 0
                    await App.database.ExcluirUsuario(usuario.idUsuario);
                    //atualiza a lista
                    AtualizarLista.Execute(null);
                }

            }
            catch (Exception ex)
            {
                await Application.Current.MainPage.DisplayAlert("Ops", ex.Message, "OK");

            }

        });
    }
}
```

Fonte: Elaborada pelo autor

Ao clicar no ícone que tem um papel e um lápis, a tela é redirecionada para a tela de controle de permissão do funcionário.

A tela de controle de permissão é responsável por atribuir permissões ao funcionário. Essas permissões são para criar e editar área, adicionar funcionário e atribuir permissões ao funcionário.

A *view* da tela de permissão é constituída por uma grid para a organização dos elementos, os elementos são: título da tela contendo o nome do funcionário, um *picker* contendo a lista de fazendas que o usuário possui permissão, e os *switches* para atribuir ou revogar permissão do usuário, conforme apresentado na figura 115.

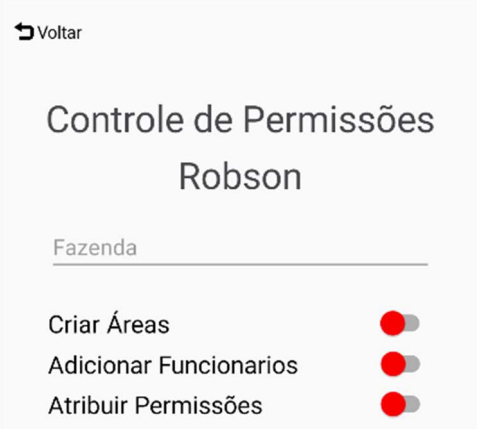
Figura 115 - *View* da tela de permissão do funcionário

```
<Grid RowDefinitions="Auto,Auto,Auto">
    <!--Botao para voltar-->
    <StackLayout  VerticalOptions="Center" HorizontalOptions="Start" WidthRequest="400" HeightRequest="60" Grid.Row="0" ...>
    <!--Titulo da tela contendo a chamada e o nome do funcionário-->
    <StackLayout Grid.Row="1" HorizontalOptions="Center" Margin="0,-40,0,0">
        <Label Text="Controle de Permissões" FontSize="30"  HorizontalTextAlignment="Center" FontFamily="Metropolis" TextColor="#4A484D" Margin="0,50,0,0"></Label>
        <Label Text="{Binding Funcionario}" FontSize="30"  HorizontalTextAlignment="Center" FontFamily="Metropolis" TextColor="#4A484D" Margin="0,0,0,0"></Label>
        <Label x:Name="idFuncionario" Text="{Binding idFuncionario}" IsVisible="false"></Label>
    </StackLayout>
    <!--área do picker com as fazenda e os switchs para atribuir ou revogar permissão-->
    <StackLayout Grid.Row="2" HeightRequest="300" WidthRequest="300" Padding="40,10">
        <Label Text="{Binding NomeFazenda}" IsVisible="false"></Label>
        <!--picker com a lista de fazendas que o usuário possui permissão-->
        <Picker Title="Fazenda"
                x:Name="listaFazenda"
                SelectedIndexChanged="listaFazenda_SelectedIndexChanged"/>

        <!--Switch para que o funcionário possa adicionar áreas na fazenda-->
            <FlexLayout Direction="Row" AlignItems="Center"
            JustifyContent="SpaceBetween" Margin="0,20,0,0">
                <Label Text="Criar Áreas" FontSize="20" TextColor="Black"></Label>


                <Switch
                    x:Name="AcessoArea"
                    Toggled="AcessoArea_Toggled"
                     >
                    <VisualStateManager.VisualStateGroups>
                        <VisualStateGroup x:Name="CommonStates">
                            <VisualState x:Name="On">
                                <VisualState.Setters>
                                    <Setter Property="ThumbColor"
                            Value="MediumSpringGreen" />
                                </VisualState.Setters>
                            </VisualState>
                            <VisualState x:Name="Off">
                                <VisualState.Setters>
                                    <Setter Property="ThumbColor"
                            Value="Red" />
                                </VisualState.Setters>
                            </VisualState>
                        </VisualStateGroup>
                    </VisualStateManager.VisualStateGroups>
                </Switch>
             </FlexLayout>

        <!--Switch para que o funcionário possa adicionar funcionários-->
        <FlexLayout Direction="Row" AlignItems="Center"
            JustifyContent="SpaceBetween">
            <Label Text="Adicionar Funcionarios" FontSize="20" TextColor="Black"></Label>
            <Switch
                x:Name="AcessoAddFunc"
                Toggled="AcessoAddFunc_Toggled">
                <VisualStateManager.VisualStateGroups>
                    <VisualStateGroup x:Name="CommonStates">
                        <VisualState x:Name="On">
                            <VisualState.Setters>
                                <Setter Property="ThumbColor"
                        Value="MediumSpringGreen" />
                            </VisualState.Setters>
                        </VisualState>
                        <VisualState x:Name="Off">
                            <VisualState.Setters>
                                <Setter Property="ThumbColor"
                        Value="Red" />
                            </VisualState.Setters>
                        </VisualState>
                    </VisualStateGroup>
                </VisualStateManager.VisualStateGroups>
            </Switch>
        </FlexLayout>

        <!--Switch para que o funcionário possa atribuir ou revogar permissões do funcionários-->
        <FlexLayout Direction="Row" AlignItems="Center"
            JustifyContent="SpaceBetween">
            <Label Text="Atribuir Permissões" FontSize="20" TextColor="Black"></Label>
            <Switch x:Name="AcessoAddPerm"
                    Toggled="AcessoAddPerm_Toggled">
                <VisualStateManager.VisualStateGroups>
                    <VisualStateGroup x:Name="CommonStates">
                        <VisualState x:Name="On">
                            <VisualState.Setters>
                                <Setter Property="ThumbColor"
                        Value="MediumSpringGreen" />
                            </VisualState.Setters>
                        </VisualState>
                        <VisualState x:Name="Off">
                            <VisualState.Setters>
                                <Setter Property="ThumbColor"
                        Value="Red" />
                            </VisualState.Setters>
                        </VisualState>
                    </VisualStateGroup>
                </VisualStateManager.VisualStateGroups>
            </Switch>
        </FlexLayout>

    </StackLayout>


</Grid>
```

Fonte: Elaborada pelo autor

A figura 116 mostra a tela em que o usuário poderá ter controle do acesso.

Figura 116 - Tela de controle de acesso



Fonte: Elaborada pelo autor

Assim que a tela é carregada, o *dropdown* de fazenda é carregado com as fazendas que o usuário possui permissão e a função ControleAcesso é chamada, como apresentado na figura 117.

Figura 117 - Ao abrir a tela de controle de acesso

```
protected override async void OnAppearing()
{
    //flag para controle de permissões
    flag = false;
    List<Logado> logado = await App.database.GetLogado();
    Usuario usuario = await App.database.GetByIdUsuario(logado.ToList()[0].idUsuario);

    //pega as fazendas que o usuário possui permissão
    List<Fazenda> fazendas = await App.database.GetFazendaFuncionario(usuario.idUsuario, usuario.idDono);

    //se o resultado da busca for maior que 0
    if(fazendas.Count > 0)
    {
        //adiciona no picker as fazendas
        foreach (Fazenda fazenda in fazendas.ToList())
        {
            listaFazenda.Items.Add(fazenda.nome);
        }

    }
    else
    {
        //se não tiver registro oculta o campo da lista de fazenda
        listaFazenda.IsVisible = false;
    }

    //chama a função de controle de acesso
    ControleAcesso();
}
```

Fonte: Elaborada pelo autor

A função controle de acesso pegará do banco de dados todos os acessos que o funcionário possui e alterará os *switches* de acordo. Se o funcionário possuir acesso, o switch é ativado, caso contrário é desativado, como apresentado na figura 118.

Figura 118 - Função controle de acesso do funcionário

```
//função de controle de acesso
2 referências
public async void ControleAcesso()
{
    //recebe a fazenda selecionada no dropdown
    var fazenda = listaFazenda.SelectedItem;
    List<Logado> logado = await App.database.GetLogado();
    Usuario usuario = await App.database.GetByIdUsuario(logado.ToList()[0].idUsuario);

    //se tiver fazenda selecionada
    if (fazenda != null)
    {
        //pega os dados da fazenda no banco
        List<Fazenda> modelFazenda = await App.database.GetByNomeFazenda(fazenda.ToString(),usuario.idDono);
        int idFazenda = modelFazenda[0].idFazenda;

        //pega os acessos do funcionário
        List<Acesso> Acessos = await App.database.GetAcessoUsuario(Int32.Parse(idFuncionario.Text), idFazenda);

        //troca o estado dos switchs de acordo com acessos do funcionário
        if(Acessos.Count > 0)
        {
            foreach (Acesso acesso in Acessos.ToList())
            {
                flag = true;
                switch (acesso.idTipo)
                {
                    case 1:
                        AcessoArea.IsToggled = true;
                        break;
                    case 2:
                        AcessoAddFunc.IsToggled = true;
                        break;
                    case 3:
                        AcessoAddPerm.IsToggled = true;
                        break;


                }
                    flag = false;
            }
        }
        else
        {
            flag = true;
            AcessoArea.IsToggled = false;
            flag = true;
            AcessoAddFunc.IsToggled = false;
            flag = true;
            AcessoAddPerm.IsToggled = false;
            flag = false;

        }
    }
    else
    {
        flag = true;
        AcessoArea.IsToggled = false;
        flag = true;
        AcessoAddFunc.IsToggled = false;
        flag = true;
        AcessoAddPerm.IsToggled = false;
        flag = false;

    }


}
```

Fonte: Elaborada pelo autor

Quando o *switch* troca de estado a função AtualizaAcesso é chamada. A função recebe como parâmetro o estado do *switch* e qual o tipo do acesso, como apresentado na figura 119.

Figura 119 - Troca de estado do switch de permissão

```
//evento chamado ao trocar o estado do switch da área
0 referências
private void AcessoArea_Toggled(object sender, ToggledEventArgs e)
{
    var troca = AcessoArea.IsToggled;
    AtualizaAcesso(troca, 1);
}

//evento chamado ao trocar o estado do switch de adicionar funcionário
0 referências
private void AcessoAddFunc_Toggled(object sender, ToggledEventArgs e)
{
    var troca = AcessoAddFunc.IsToggled;
    AtualizaAcesso(troca, 2);
}

//evento chamado ao trocar o estado do switch de adicionar permissões para o funcionário
0 referências
private void AcessoAddPerm_Toggled(object sender, ToggledEventArgs e)
{
    var troca = AcessoAddPerm.IsToggled;

    AtualizaAcesso(troca, 3);
}
```

Fonte: Elaborada pelo autor

A função AtualizaAcesso, com base nos parâmetros recebidos, irá atribuir ou revogar a permissão do funcionário na fazenda selecionada. Se o usuário não selecionar nenhuma fazenda, um alerta é emitido informando-o da necessidade de selecionar a fazenda antes de alterar a permissão, como apresentado na figura 120.

Figura 120 - Função para atualizar o acesso do funcionário

```
//função para atualizar acessos, recebe como parametro o estado do switch e o tipo de permissão
3 referências
public async void AtualizaAcesso(bool Acesso, int Tipoid)
{
    //recebe a fazenda selecionada no dropdown
    var fazenda = listaFazenda.SelectedItem;
    List<Logado> logado = await App.database.GetLogado();
    Usuario usuario = await App.database.GetByIdUsuario(logado.ToList()[0].idUsuario);

    if (!flag)
    {
        if (listaFazenda.IsVisible)
        {
            //se tiver fazenda selecionada
            if (fazenda != null)
            {
                //pega os dados da fazenda
                List<Fazenda> modelFazenda = await App.database.GetByNomeFazenda(fazenda.ToString(), usuario.idDono);
                int idFazenda = modelFazenda[0].idFazenda;

                //se o estado do switch for ativo
                if (Acesso) //Concedeu Acesso
                {
                    //adiciona o acesso para o funcionário
                    await App.database.InsertAcesso(Int32.Parse(idFuncionario.Text), Tipoid, idFazenda);
                    await App.Current.MainPage.DisplayAlert("Sucesso", "Acesso Liberado", "OK");
                }
                else //retirou acesso
                {
                    //retira o acesso do funcionário
                    await App.database.DeleteAcessoUsuario(Int32.Parse(idFuncionario.Text), Tipoid, idFazenda);
                    await App.Current.MainPage.DisplayAlert("Sucesso", "Acesso Revogado", "OK");
                }
            }
            else
            {
                //se não selecionou nenhuma fazenda e disparado um alerta para o usuário selecionar a fazenda
                await App.Current.MainPage.DisplayAlert("Ops", "Selecione a Fazenda", "OK");
                flag = true;
                //reverte a ação do usuário
                switch (Tipoid)
                {
                    case 1:
                        AcessoArea.IsToggled = !AcessoArea.IsToggled;
                        break;
                    case 2:
                        AcessoAddFunc.IsToggled = !AcessoAddFunc.IsToggled;
                        break;
                    case 3:
                        AcessoAddPerm.IsToggled = !AcessoAddPerm.IsToggled;
                        break;

                }
                flag = false;
            }
        }
        else
        {
            await Shell.Current.GoToAsync("//ListaFuncionario");
        }
    }
    else{
        flag = false;
    }
}
```

Fonte: Elaborada pelo autor

A figura 121 mostra o alerta que é emitido ao usuário, caso não seja selecionada nenhuma fazenda, e o estado *switch* é retornado.

Figura 121 - Alerta para selecionar a fazenda

Fonte: Elaborada pelo autor

Caso o usuário selecione a fazenda e altere o switch um alerta é disparado informando-o a alteração, como apresentado na figura 122.

Figura 122 - Alerta de alteração de permissão do funcionário

Fonte: Elaborada pelo autor

A tela de perfil exibe as informações básicas dos usuários como nome, sobrenome e e-mail. A *view* irá exibir uma foto representativa do usuário e campos contendo as informações. Essas informações não são editáveis, como apresentado a figura 123.

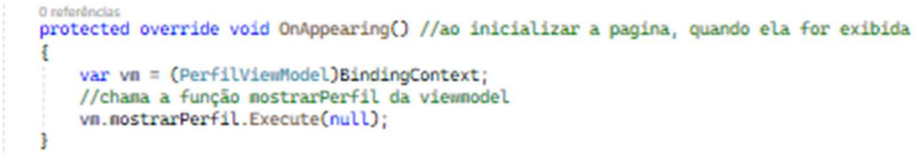
Figura 123 - *View* Perfil do usuário

```
<!--Grid para organizar os elementos-->
<Grid RowDefinitions="*,*,*">
    <StackLayout Grid.Row="0" HorizontalOptions="Center" Margin="0,-40,0,0">
        <Label Text="Perfil" FontSize="30"  HorizontalTextAlignment="Center" FontFamily="Metropolis" TextColor="#4A4B4D" Margin="0,50,0,0"></Label>
        <RelativeLayout Grid.Row="0">

            <FlexLayout RelativeLayout.XConstraint="150"
                  RelativeLayout.YConstraint="20"
                   >
                <!--Moldura da imagem-->
                <Frame Margin="10"
                  BackgroundColor="#cfcfcf"
                  CornerRadius="50"
                  HeightRequest="70"
                  WidthRequest="30"
                 >
                   <!--Imagem representantativa do usuário-->
                   <Image Source="homen"
                 Aspect="AspectFill"
                 Margin="-11"
                 />
                </Frame>

                <!--Frase de boas vindas para o usuário-->
                <StackLayout>
                    <Label
                 Text="Bem Vindo(a)!"
                TextColor="White"
                FontSize="17"
                Margin="0,15,0,0"
              ></Label>
                    <Label  x:Name="Usuario"
                 TextColor="White"
                 FontSize="17"
                 Margin="0,-10,0,0"
                ></Label>
                </StackLayout>
            </FlexLayout>
        </RelativeLayout>
    </StackLayout>

    <StackLayout Grid.Row="1" HorizontalOptions="FillAndExpand">
        <Label Text="Informação do Usuário" FontSize="19" FontFamily="Metropolis" TextColor="#5470fe" Margin="20,-50,0,0"></Label>
        <Grid RowDefinitions="*,*" ColumnDefinitions="*,*" Padding="20,10,20,0">
            <!--Campo contendo o primeiro nome do usuário-->
            <StackLayout Grid.Row="0" Grid.Column="0">
                <Label Text="Primeiro Nome" FontSize="14" TextColor="Gray"></Label>
                <Frame HasShadow="False" BorderColor="#d9d9d9" CornerRadius="20">
                    <Label Text="{Binding Nome}" FontSize="16" Margin="0,-5,0,0" TextColor="Black"></Label>
                </Frame>
            </StackLayout>
            <!--Campo contendo o sobrenome do usuário-->
            <StackLayout Grid.Row="0" Grid.Column="1" Margin="20,0,0,0">
                <Label Text="Último Nome" FontSize="14" TextColor="Gray"></Label>
                <Frame HasShadow="False" BorderColor="#d9d9d9" CornerRadius="20" >
                    <Label Text="{Binding Sobrenome}" FontSize="16" Margin="0,-5,0,0" TextColor="Black"></Label>
                </Frame>
            </StackLayout>
        </Grid>
        <!--Campo contendo o E-mail do usuário-->
        <StackLayout Margin="0,-70,0,0" Padding="20,0,20,0">
            <Label Text="E-mail" FontSize="14" TextColor="Gray"></Label>
            <Frame HasShadow="False" BorderColor="#d9d9d9" CornerRadius="20">
                <Label Text="{Binding Email}" FontSize="16" Margin="0,-5,0,0" TextColor="Black"></Label>
            </Frame>
        </StackLayout>

    </StackLayout>
</Grid>
```

Fonte: Elaborada pelo autor

Ao entrar na tela de perfil a função mostrarPerfil é chamada, como apresentado na figura 124.

Figura 124 - Função mostrarPerfil

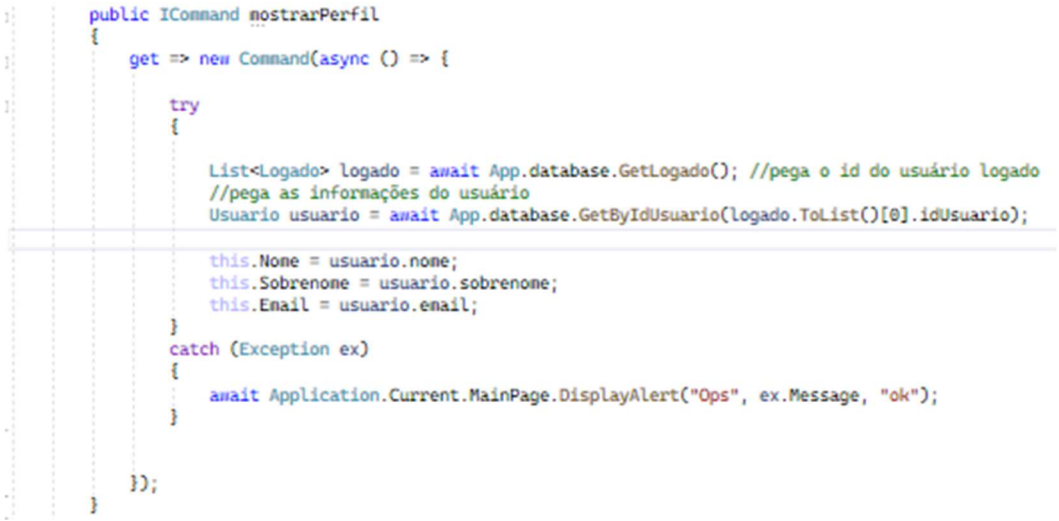


```
0 referências
protected override void OnAppearing() //ao inicializar a pagina, quando ela for exibida
{
    var vm = (PerfilViewModel)BindingContext;
    //chama a função mostrarPerfil da viewmodel
    vm.mostrarPerfil.Execute(null);
}
```

Fonte: Elaborada pelo autor

A função mostrarPerfil irá pegar o id do usuário logado para pegar as informações no banco de dados. Essas informações serão colocadas nos campos presentes na *view*, como apresentado na figura 125.

Figura 125 - Função mostrarPerfil



```
public ICommand mostrarPerfil
{
    get => new Command(async () => {

        try
        {

            List<Logado> logado = await App.database.GetLogado(); //pega o id do usuário logado
            //pega as informações do usuário
            Usuario usuario = await App.database.GetByIdUsuario(logado.ToList()[0].idUsuario);

            this.Nome = usuario.nome;
            this.Sobrenome = usuario.sobrenome;
            this.Email = usuario.email;
        }
        catch (Exception ex)
        {
            await Application.Current.MainPage.DisplayAlert("Ops", ex.Message, "ok");
        }


    });
}
```

Fonte: Elaborada pelo autor

A figura 126 apresenta a tela de perfil com os campos preenchidos pelas informações do usuário.

Figura 126 - Tela do perfil do usuário

Fonte: Elaborada pelo autor

Na navegação da TabBar tem um redirecionamento para o mapa. A página do mapa é composta por um campo para que o usuário possa digitar o endereço desejado e uma lupa para se fazer a pesquisa. Abaixo desses campos é implementado o mapa, como apresentado a figura 127.

Figura 127 - *View* página do mapa

```
<StackLayout>
    <StackLayout HorizontalOptions="Center" Orientation="Horizontal" Margin="0,30,0,0">
        <!--Campo para receber o endereço da região-->
        <Frame CornerRadius="30" BackgroundColor="#f2f2f2" HasShadow="False" WidthRequest="310" HeightRequest="15
            <Entry x:Name="enderecoMapa" HorizontalTextAlignment="Start" FontSize="14" Placeholder="Endereço" Text
        </Frame>
        <!--botão para realizar a busca pela região-->
        <ImageButton x:Name="iconLupa" Source="lupa" BackgroundColor="Transparent" WidthRequest="20" Clicked="ic
    </StackLayout>
    <!--Mapa da aplicação-->
    <maps:Map IsShowingUser="True"  x:Name="map" PropertyChanged="map_PropertyChanged"/>
</StackLayout>
```

Fonte: Elaborada pelo autor

A tela que o usuário visualizará é bem parecida com a tela de demarcação da região, onde se encontra a busca por região e o mapa, como apresentado na figura 128.

Figura 128 - Tela do mapa

Fonte: Elaborada pelo autor

Ao iniciar, a página do mapa a localização do usuário é pega para que o mapa seja centralizado na localização em que o usuário se encontra, e instanciado também as áreas em que o usuário possui acesso, como apresentado na figura 129 código.

Figura 129 - Inicialização da página do mapa

```
protected override async void OnAppearing() //ao inicializar a pagina, quando ela for exibida
{
    //limpa todos elementos do mapa
    map.MapElements.Clear();

    //****************Centralizar no Usuário**********************************

    var request = new GeolocationRequest(GeolocationAccuracy.Best, TimeSpan.FromSeconds(10));

    var location = await Geolocation.GetLocationAsync(request);

    Position pUsuario = new Position(location.Latitude, location.Longitude);

    MapSpan MapSpan = MapSpan.FromCenterAndRadius(pUsuario, Distance.FromKilometers(.444));

    map.MoveToRegion(MapSpan);

    //*************************************************************************

    //pega o usuário logado
    List<Logado> logado = await App.database.GetLogado();

    //pega as informações do usuário
    Usuario usuario = await App.database.GetByIdUsuario(logado.ToList()[0].idUsuario);

    //instancia a variavel que irá receber as areas
    List<Area> quantArea = null;

    //verifica se o usuário e o dono
    if (usuario.idDono != usuario.idUsuario)
    {
        //pega as fazendas que o usuário tem acesso
        List<Fazenda> fazendas = await App.database.GetFazendaFuncionario(usuario.idUsuario, usuario.idDono);

        string str = "";

        //colocar na variavel str todas as fazendas em que o usuário tem acesso
        foreach (Fazenda fazenda in fazendas.ToList())
        {
            str = fazenda.idFazenda + ",";
        }

        if (str != "")
        {
            //pega as áreas
            quantArea = await App.database.GetAreasFazenda(str.Substring(0, str.Length - 1)); //Substring para retirar a ultima virgula

        }
    }
    else
    {
        //pega todas as áreas
        quantArea = await App.database.GetAreasUsuario(usuario.idDono);
    }


    //se retornou algo do banco
    if (quantArea.ToList().Count > 0)
    {
        //chama a função disparaMensagem
        disparaMensagem();
        //para cada área será criado um polígono
        foreach (Area quant in quantArea.ToList())
        {
            List<Pontos> model = await App.database.GetPontosArea(quant.idArea);
            if(model.ToList().Count > 0)
            {
                Polygon polygon1 = new Polygon
                {
                    StrokeWidth = 8,
                    StrokeColor = Color.FromHex("#0035FF"),
                    FillColor = Color.FromHex("#B7C1FA")
                };

                Pin pin = new Pin
                {
                    Label = quant.nome,
                    Address = quant.descricao,
                    Type = PinType.Place,
                };


                foreach (Pontos element1 in model.ToList())
                {
                    Position p = new Position((double)element1.latitude, (double)element1.longitude);
                    polygon1.Geopath.Add(p);
                    latitude = (double)element1.latitude;
                    longitude = (double)element1.longitude;
                };

                pin.Position = new Position(latitude,longitude);
                map.Pins.Add(pin);

                map.MapElements.Add(polygon1);
            }
        };
    }
}
```

Fonte: Elaborada pelo autor

Na criação do polígono, em um dos pontos é criado um pin contendo o nome da área e sua descrição, facilitando assim a identificação da área, como apresentado na figura 130.

Figura 130 - Identificação da área

Fonte: Elaborada pelo autor

Na página do mapa tem um *timer* para que de tempos em tempos seja verificado se o usuário entrou ou saiu de alguma área. A cada segundo o timer chama a função varrerArea. A função irá carregar os pontos da área e enviar para a função que calcula se o usuário está dentro do polígono ou não. Caso o usuário adentre a área marcada uma mensagem é enviada para o dispositivo do usuário. Enquanto o usuário permanecer na área, uma variável é responsável por informar que o usuário já foi notificado, para que não ocorra disparo de mensagem a todo momento, como apresentado nas figuras 131 e 132.

Figura 131 - Timer página Mapa

```
public void disparaMensagem()
{
    //timer para que de tempos em tempos verifcar se o usuário entrou em alguma área
    Device.StartTimer(new TimeSpan(0, 0, 1), () =>
    {
        changedMap = true;
        varrerArea();
        return true;
    });
}
```

Fonte: Elaborada pelo autor

Figura 132 - Função responsavel pelo controle de mensagem

```
public async void varrerArea()
{
    try
    {
        var request = new GeolocationRequest(GeolocationAccuracy.Best, TimeSpan.FromSeconds(10));

        var location = await Geolocation.GetLocationAsync(request);

        Position pUsuario = new Position(location.Latitude, location.Longitude);

        MapSpan MapSpan = MapSpan.FromCenterAndRadius(pUsuario, Distance.FromKilometers(.444));

        if (chargedMap)
        {
            //***************Centralizar no Usuário**********************************
              map.MoveToRegion(MapSpan);
            //***********************************************************************
        }


        if (notify.Split('|')[0] == "true" && Int32.Parse(notify.Split('|')[1]) > 0) //se o Usuário já esta dentro de uma área
            {
            //pega todos os pontos para ver se o usuário ainda esta dentro da área
                List<Pontos> model = await App.database.GetPontosArea(Int32.Parse(notify.Split('|')[1]));
                Point[] pts = new Point[model.ToList().Count];
                int i = 0;
                int idArea = 0;
                foreach (Pontos element1 in model.ToList())
                {

                    pts[i] = new Point { X = (double)element1.Latitude, Y = (double)element1.Longitude };
                    i++;
                    idArea = element1.idArea;
                };
                Point posUsu = new Point { X = location.Latitude, Y = location.Longitude };

                bool dentro = intoPolygon(pts, posUsu);

            //se o usuário saiu da área notify reseta sua informação
            if (!dentro)
            {
                notify = "false|-1";
            }
        }
            else
            {
                List<Area> qtArea = await App.database.GetQuantArea(idLogado); //pega a quantidade de Area Cadastrada

                if (qtArea.ToList().Count > 0)
                {
                    foreach (Area quant in qtArea.ToList())
                    {
                        List<Pontos> model = await App.database.GetPontosArea(quant.idArea);

                        if (model.Count > 3) //so verificar se esta dentro quando tiver mais de 3 pontos cadastrados
                        {
                            Point[] pts = new Point[model.ToList().Count];
                            int i = 0;
                            int idArea = 0;
                            foreach (Pontos element1 in model.ToList())
                            {
                                pts[i] = new Point { X = (double)element1.Latitude, Y = (double)element1.Longitude };
                                i++;
                                idArea = element1.idArea;
                            };
                            Point posUsu = new Point { X = location.Latitude, Y = location.Longitude };

                            bool dentro = intoPolygon(pts, posUsu);

                            if (dentro)// se o usuário estiver dentro da área e disparado um alerta
                            {

                                notify = "true|" + idArea;
                                Area getArea = await App.database.GetByIdArea(idArea);
                                var notification = new NotificationRequest
                                {
                                    BadgeNumber = 1,
                                    NotificationId = 1337,
                                    Title = "Entrou na Area: " + getArea.nome,
                                    Description = getArea.descricao,
                                    ReturningData = "Dummy data", //Retorna dados quando toca na notificação
                                };
                                await LocalNotificationCenter.Current.Show(notification);
                                await DisplayAlert("Entrou na Area: " + getArea.nome, getArea.descricao, "OK");

                            }
                        }


                    }
                }
            }

    }
```

Fonte: Elaborada pelo autor

Para verificar se o usuário está dentro do polígono a função intoPolygon recebe como parâmetros os pontos do polígono e a posição do usuário. A partir desses parâmetros e verificado se a localização do usuário está dentro da área, a função retornará um valor booleano, *true* caso o usuário esteja dentro da área, *false* caso contrário, como apresentado na figura 133.

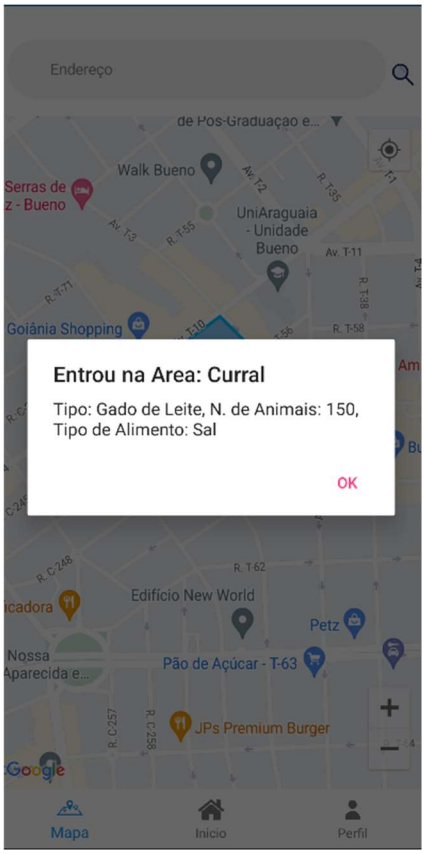
Figura 133 - Cálculo para ver se o usuário está dentro da área

```
public static bool intoPolygon(Point[] poly, Point pointUsuario)
{

    //calculo para ver se o usuário está dentro da área
    var coef = poly.Skip(1).Select((p, i) =>
                                    (pointUsuario.Y - poly[i].Y) * (p.X - poly[i].X)
                                  - (pointUsuario.X - poly[i].X) * (p.Y - poly[i].Y))
                           .ToList();

    if (coef.Any(p => p == 0))
        return true;

    for (int i = 1; i < coef.Count(); i++)
    {
        if (coef[i] * coef[i - 1] < 0)
            return false;
    }
    return true;
}
```

Fonte: Elaborada pelo autor

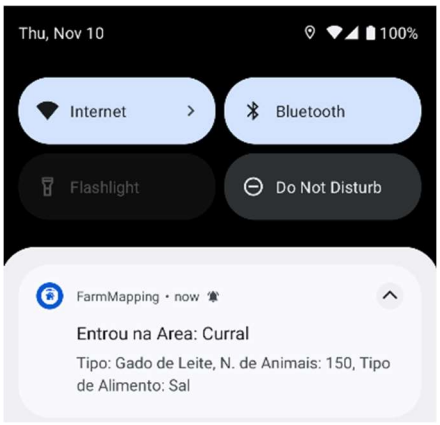
Assim que o usuário entrar na área, um alerta é emitido e uma mensagem é enviada ao dispositivo do usuário, como apresentado nas figuras 134 e 135.

Figura 134 - Alerta emitido ao entrar na área



Fonte: Elaborada pelo autor

Figura 135 - Mensagem enviada ao usuário ao entrar na área



Fonte: Elaborada pelo autor

Para que o usuário possa ter liberdade de movimentar o mapa, um evento é disparado ao interagir com ele. Nesse evento é armazenado em uma variável global booleana o valor *false*, enquanto a variável for igual a false o mapa não centraliza na localização do usuário, como apresentado na figura 136.

Figura 136 - Evento ao interagir com o mapa

```
private void map_PropertyChanged(object sender, System.ComponentModel.PropertyChangedEventArgs e)
{
    changedMap = false;
}
```

Fonte: Elaborada pelo autor

6 CONSIDERAÇÕES FINAIS

O presente trabalho propôs-se a dar prosseguimento ao trabalho feito por Shirlano Candido Dias Filho (FILHO, 2021), com base nas sugestões de trabalhos futuros.

Durante o processo de desenvolvimento fica claro a facilidade da utilização da linguagem de programação C#, linguagem com sintaxe clara e capaz de fazer coisas complexas com pouquíssimas linhas de código, uma linguagem que possui uma documentação completa tornando fácil a aprendizagem.

A IDE Visual Studio Community é uma ferramenta incrível para desenvolvimento. Com ela é possível em tempo de execução depurar o código linha a linha vendo como o código está se comportando com as trocas de informações, facilitando assim achar algum possível erros. A IDE traz também o *IntelliCode* trazendo agilidade na programação. O *IntelliCode* faz com que durante a escrita de código surja uma lista de conclusões com aquilo com maior probabilidade de se usar (STUDIO, 2022). O Visual Studio traz também bibliotecas prontas facilitando assim a implementação da aplicação, reaproveitando trechos de códigos já criados.

O banco de dados local SQLite foi de grande importância para este trabalho. Para se comunicar com o banco de dados não há a necessidade de conexão externa visto que SQLite e um arquivo armazenado no dispositivo, para fase de desenvolvimento facilitando assim a emulação para testes. Com códigos simples de SQL foi capaz de manipular os dados da aplicação, podendo inserir, selecionar, atualizar e excluir dados do aplicativo.

Com a linguagem XAML foi notória a facilidade da construção e organização das informações na tela, foi construída de forma intuitiva para que o usuário navegue pela aplicação sem grandes esforços, com poucos cliques o usuário finaliza a tarefa desejada.

Houve um grande avanço em relação a versão anterior do aplicativo, novas funcionalidades para a gestão das áreas, quando se trata de terra e muito importante que se tenha precisão na demarcação, com o desenvolvimento de geofences em formatos poligonais esse objetivo e alcançado. Além da administração das áreas agora é possível administrar funcionários, podendo liberar acessos a

determinadas atividades na fazenda como criar áreas, cadastrar funcionários e atribuir acesso a eles.

Com o desenvolvimento deste trabalho foi adquirido um vasto conhecimento de desenvolvimento de aplicativo mobile e de novas linguagens como XAML e C#. O projeto apresenta um grande potencial para ter seu desenvolvimento continuado, novas funcionalidades podem ser criadas no que diz respeito à segurança dos dados e nos relatórios de produção mostrando comparativos dentro de um prazo determinado. Os objetivos propostos foram cumpridos.

Nos apêndices encontra-se todo o código desenvolvido no trabalho, e no anexo 1 apresenta-se o termo de autorização para publicação.

6.1 Trabalhos Futuros

Propõe-se em trabalhos futuros:

- Implementar relatório de produção das áreas, mostrando histórico e comparações.
- Implementar log de modificações, sendo possível monitorar alterações identificando executor, data e o que foi modificado.
- Implementar a encriptação de dados sensíveis.
- Desenvolver um método de pegar e soltar pins, sendo assim possível a edição dos pontos criados na área.

REFÊRENCIAS

ACTOMIC. ***Polygon-geofence***. 2019. Disponível em: https://www.tislog.de/polygon-geofence-2/. Acesso em: 24 mar. 2022

BERTOLINI, Cristiano et al. **Linguagem de programação I**. 2019. Disponível em: https://repositorio.ufsm.br/bitstream/handle/1/18352/Curso_Lic-Cienc-Relig_Linguagem-Progracamacao.pdf?sequence=1&isAllowed=y Acesso em : 03 maio 2022

BRANDÃO, Bruna. **O que é *geofencing*? Para que serve? Como funciona na prática?** 2020. Disponível em: https://maplink.global/blog/o-que-e-geofencing/. Acesso em: 24 fev. 2022

CAMÕES, Instituto. **Navegações Portuguesas**. 2002. Disponível em: http://cvc.instituto-camoes.pt/navegaport/a06.html. Acesso em: 07 mar. 2022.

CARVALHO, Edilson Alves de; ARAUJO, Paulo César de. **Localização: coordenadas geográficas**. 2008. Disponível em: http://www.ead.uepb.edu.br/ava/arquivos/cursos/geografia/leituras_cartograficas/Le_Ca_A08_J_GR_260508.pdf. Acesso em: 03 mar. 2022.

DATE, Christopher J. **Introdução a sistemas de bancos de dados**. Elsevier Brasil, 2004. Disponível em: https://books.google.com.br/books?hl=pt-BR&lr=&id=xBeO9LSlK7UC&oi=fnd&pg=PP23&dq=o+que+e+um+banco+de+dados&ots=xcPFf0v89G&sig=TKxT_Lx-56KOyATOHKdAq-Uz8vM#v=onepage&q=o%20que%20e%20um%20banco%20de%20dados&f=false Acesso em: 24 mar 2022

DAVIDBRITCH et al. **O padrão *Model*-View-View*Model* - Xamarin**. 2022 Disponível em: <https://docs.microsoft.com/pt-br/xamarin/xamarin-forms/enterprise-application-patterns/mvvm>. Acesso em: 05 set 2022

DAVIDORTINAU. **Permissões no Xamarin.Android - Xamarin**. 2022 Disponível em: <https://docs.microsoft.com/pt-br/xamarin/android/app-fundamentals/permissions?tabs=windows>. Acesso em: 13 set. 2022.

DAVIDORTINAU et.al. **Usando a API do Google Mapas em seu aplicativo - Xamarin**. 2022 Disponível em: <https://learn.microsoft.com/pt-br/xamarin/android/platform/maps-and-location/maps/maps-api>. Acesso em: 3 out. 2022.

DALCOMUNE ,Monique **Sistema transacional: entenda o significado para a operação da empresa**. 15 outubro 2021 Disponível em: <https://nordware.io/blog/sistema-transacional-entenda-o-significado-para-a-operacao-da-sua-empresa/>. Acesso em: 19 set. 2022.

DEVELOPERS, *Android*. **Conheça o *Android Studio***. 17 maio 2021. Disponível em: https://developer.android.com/studio/intro?hl=pt-br. Acesso em: 9 maio 2022.

DEVELOPERS **, Manifest.permission**. 18 julho 2022 Disponível em: <https://developer.android.com/reference/android/Manifest.permission>. Acesso em: 19 set 2022

, Visão geral do manifesto do aplicativo | Desenvolvedores Android. 9 setembro 2022 Disponível em: <https://developer.android.com/guide/topics/manifest/manifest-intro?hl=pt&gclsrc=ds&gclsrc=ds#filef>. Acesso em: 19 set. 2022.

DEVMEDIA**. Entendendo o *Pattern Model View ViewModel* MVVM**. 2010 Disponível em: <https://www.devmedia.com.br/entendendo-o-pattern-*Model*-view-view*Model*-mvvm/18411>. Acesso em: 05 set 2022

DORE, Eder. **Como funciona a Geolocalização na prática? Para que serve?**. Maplink, 26 dez. 2019. Disponível em: https://maplink.global/blog/como-funciona-geolocalizacao/. Acesso em: 21 abr. 2022.

DOS REIS, Carlos Roberto et al. **Aplicação da biblioteca OpenGL em um jogo desenvolvido no C# com integração do Arduino Uno e *Ionic Framework***. Revista Eletrônica de Iniciação Científica em Computação, v. 18, n. 2, 2020. Disponível em: https://www.seer.ufrgs.br/index.php/reic/article/view/94500/57896 Acesso em 04/05/2022. Acesso em: 04 maio 2022

FERNANDES, Renato de Oliveira. **Sistema de Navegação Global por Satélite.** Disponível em: http://wiki.urca.br/dcc/lib/exe/fetch.php?media=sistema_de_navegacao_global_por_satelite.pdf. Acesso em: 07 jun 2022

GUEDES, Max Justo. **Acerca de alguns instrumentos náuticos (inclusive dois astrolábios) recuperados no naufrágio do Sacramento (1668), na Bahia**. UC Biblioteca Geral 1, 1981. Disponível em : https://books.google.com.br/books?hl=pt-BR&lr=&id=iaJxAl1Jde4C&oi=fnd&pg=PA283&dq=astrolabios&ots=GY4H7alHyA&sig=LsjJREQoqYCh9jrFFBDrWgFsG4E#v=onepage&q=astrolabios&f=false. Acesso em: 17 mar 2022

HAGOS, Ted. ***Android* Studio IDE. *In*:*Learn Android Studio* 4**. *Apress*, Berkeley, CA, 2020. p. 31-45. Disponível em: https://link.springer.com/chapter/10.1007/978-1-4842-5937-5_4. Acesso em: 9 maio 2022

JAMESMONTEMAGNO et al. **Xamarin.Essentials: permissões-Xamarin**. 2022 Disponível em: <https://docs.microsoft.com/pt-br/xamarin/essentials/permissions?tabs=android>. Acesso em: 12 set. 2022.

JAMESMONTEMAGNO et al. **Introdução com Xamarin.Essentials** 2022 Disponível em: <https://docs.microsoft.com/pt-br/xamarin/essentials/get-started?context=xamarin%2Fandroid&tabs=windows%2Candroid>. Acesso em: 12 set. 2022.

JONDOUGLAS et al. **O que é o NuGet e o que ele faz?** 2022 Disponível em: <https://docs.microsoft.com/pt-br/nuget/what-is-nuget>. Acesso em: 5 set. 2022.

LEANDRO, Julia. **BALESTILHA**. 2015. Disponível em: https://prezi.com/tbio5prapfz4/balestilha/. Acesso em: 17 mar. 2022.

MEDEIROS, Juliana Mareco. **Investigando o estado da arte na evolução de bancos de dados: um mapeamento sistemático e a criação de um *guideline*.** 2022. Disponível em: https://repositorio.unipampa.edu.br/jspui/bitstream/riu/6875/1/Juliana%20Mareco%20Medeiros%20-%202022.pdf. Acesso em: 05 abr. 2022

MENDES, V. B. **Sistema de posicionamento global**. apontamentos das aulas de Geodesia II, texto não publicado, Departamento de Engenharia Geográfica, Geofísica e Energia, Faculdade de Ciências da Universidade de Lisboa, 20013. Acesso em: 11 mar 2022

MONTEIRO, Dani. **Voce já escolheu um tipo de banco de dados.** 2017. Disponível em: http://db4beginners.com/blog/escolha-o-tipo-bd/. Acesso em: 07 jun 2022.

NAEEM, Tehreem. **Definição de API REST: Noções Básicas de APIs REST**. *Astera*, 28 Jan. 2020, Disponivel em: https://www.astera.com/pt/tipo/blog/defini%C3%A7%C3%A3o-de-api/. Acesso em: 05 jun 2022.

NETO, Anor Batista Esteves. **API GENÉRICA PARA EXTRACAO DE DADOS DE PAGINAS DA INTERNET**. 2021. Tese de Doutorado. Universidade Federal do Rio de Janeiro. Disponível em: http://www.repositorio.poli.ufrj.br/monografias/projpoli10034 360.pdf. Acesso em: 07 abr. 2022

PRASTYO, Budiman. ***Program sistem manajemen laboratorium kimia UIN Walisongo menggunakan* microsoft visual studio community 2019** berbasis pengembangan berkelanjutan. Disponivel em: http://eprints.walisongo.ac.id/id/eprint/12422/1/skripsi_1503076002_Budiman%20Prastyo.pdf. Acesso em: 9 maio 2022

PUGA,Rogério Miguel.**Quadrante**. 2019. Disponível

em: http://www.fcsh.unl.pt/devp/dictionary/quadrante/. Acesso em: 09 mar. 2022.

PROFEXORGEEK et al. **O que é o Xamarin.Forms? - Xamarin**. 2022 Disponível em: <https://docs.microsoft.com/pt-br/xamarin/get-started/what-is-xamarin-forms>. Acesso em 11 set 2022

SANGLARD, Danielly Emerick Faria. **ESTACIONE SIMPLES-SISTEMA DE GERENCIAMENTO DE ESTACIONAMENTO**. Repositório de Trabalhos de Conclusão de Curso, 2022.Disponivel em: http://pensaracademico.facig.edu.br/index.php/repositoriotcc/article/view/3432 Acesso em 04 maio 2022.

SOUZA, Fernando José de; SANTANA, Paulo Henrique Alves de. **ESTUDO DE CASO: ANÁLISE ENTRE BANCO DE DADOS RELACIONAL E NÃO RELACIONAL**. 2017.Disponivel em: http://repositorio.aee.edu.br/bitstream/aee/46/1/TCC2_2017_02_FernandoJoseDeSouza_PauloHenriqueAlvesDeSantana.pdf Acesso em: 24 mar 2022

SQLITE. **About SQLite.** 2019 Disponível em: <https://www.sqlite.org/about.html>. Acesso em: 19 set. 2022.

TAIPALUS, Toni; SEPPÄNEN, Ville. **SQL *education: A systematic mapping study and future research* agenda**. ACM *Transactions on Computing Education* (TOCE), v. 20, n. 3, p. 1-33, 2020. Disponível em: https://dl.acm.org/doi/abs/10.1145/3398377 Acesso em 07 abr. 2022

TAKEBLIP (org.). **API: conceito, exemplos de uso e importância da integração para desenvolvedores**. 2019. Disponível em: https://www.take.net/blog/tecnologia/api-conceito-e-exemplos/. Acesso em: 24 fev. 2022.

TAGLIANI, Carlos Roney Armani. **Sistema de Posicionamento Global-GPS** [Parte 1]. 2022. Disponível em: http://repositorio.furg.br/bitstream/handle/1/9780/M%c3%b3dulo%20GPS-%20Parte%201.pdf?sequence=1. Acesso em:11 mar 2022

TOLEDO, Lucas Henrique Bento. **Desenvolvimento de aplicações *Android* adaptáveis utilizando a linguagem Kotlin**. 2021.Disponivel em: https://monografias.ufop.br/bitstream/35400000/3042/6/MONOGRAFIA_DesenvolvimentoAplica%c3%a7%c3%b5esAndroid.pdf. Acesso em 09 de maio 2022

SILVEIRA, Daniel; TOOGE ,Rikardy.**Área com produção rural no Brasil cresce mais de 5% em 11 anos, diz IBGE**. 2019. Disponível em: https://g1.globo.com/economia/agronegocios/noticia/2019/10/25/area-com-producao-rural-no-brasil-cresce-mais-de-5percent-em-11-anos-diz-ibge.ghtml. Acesso em: 24 fev. 2022

STUDIO, Visual**. Visual Studio IntelliCode | Visual Studio**. Disponível em: <https://visualstudio.microsoft.com/pt-br/services/intellicode/>. Acesso em: 11 nov. 2022.

WAWRZYNIAK, Natalia; HYLA, Tomasz. ***Application of geofencing technology for the purpose of spatial analyses in inland mobile navigation***. *In*:2016 *Baltic Geodetic Congress* (BGC *Geomatics*). IEEE, 2016. p. 34-39. Disponível em: https://ieeexplore.ieee.org/abstract/document/7548001 Acesso em: 24 mar 2022

APÊNDICES

APÊNDICE A — polygonalGeofencing.Android

Arquivo – MainActivity.cs

```
using System;

using Android.App;
using Android.Content.PM;
using Android.Runtime;
using Android.OS;
using Android;
using Plugin.LocalNotification;

namespace polygonalGeofencing.Droid
{
   [Activity(Label = "FarmMapping", Icon = "@mipmap/Logo", Theme = "@style/MainTheme",
MainLauncher = true, ConfigurationChanges = ConfigChanges.ScreenSize | ConfigChanges.Orientation |
ConfigChanges.UiMode | ConfigChanges.ScreenLayout | ConfigChanges.SmallestScreenSize)]
   public class MainActivity : global::Xamarin.Forms.Platform.Android.FormsAppCompatActivity
   {
      const int RequestLocationId = 0;

      readonly string[] LocationPermissions =
      {
         Manifest.Permission.AccessCoarseLocation,
         Manifest.Permission.AccessFineLocation
      };

      protected override void OnStart() //ao inicializar
      {
         base.OnStart();

         if ((int)Build.VERSION.SdkInt >= 23) //verifica se o sdk esta com a versão acima da 23
         {
            //verifica se a permissão de localização foi autorizada
            if (CheckSelfPermission(Manifest.Permission.AccessFineLocation) != Permission.Granted)
            {
               //solicita a permissão a ter acesso a localização
               RequestPermissions(LocationPermissions, RequestLocationId);
            }
            else
            {
               //Permissão Permitida
            }
         }
      }
```

```
        protected override void OnCreate(Bundle savedInstanceState)
        {
            base.OnCreate(savedInstanceState);

            LocalNotificationCenter.NotifyNotificationTapped(Intent);

            Xamarin.Essentials.Platform.Init(this, savedInstanceState);
            global::Xamarin.Forms.Forms.Init(this, savedInstanceState);
            Xamarin.FormsMaps.Init(this, savedInstanceState);
            LoadApplication(new App());
        }
        public override void OnRequestPermissionsResult(int requestCode, string[] permissions,
[GeneratedEnum] Android.Content.PM.Permission[] grantResults)
        {
            if (requestCode == RequestLocationId)
            {
                if ((grantResults.Length == 1) && (grantResults[0] == (int)Permission.Granted))
//Permission.Granted – A permissão especificada foi concedida.
                {
                    //Permissão Permitida
                }
                else
                {
                    //Permissão Negada
                }
            }
            else
            {
                Xamarin.Essentials.Platform.OnRequestPermissionsResult(requestCode, permissions, grantResults);
            }




            base.OnRequestPermissionsResult(requestCode, permissions, grantResults);
        }
    }
}
```

Pasta Properties – AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" android:versionCode="1"
android:versionName="1.0" package="com.companyname.polygonalgeofencing"
android:installLocation="preferExternal">
        <uses-sdk android:minSdkVersion="17" android:targetSdkVersion="31" />
        <application android:label="FarmMapping" android:theme="@style/MainTheme"
android:icon="@drawable/Logo">
            <meta-data android:name="com.google.android.maps.v2.API_KEY"
android:value="AIzaSyAUipN1d2L7AfNgnPm7fqMK7I0q9IdoBfc" />
            <uses-library android:name="org.apache.http.legacy" android:required="false" />
        </application>
        <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

```
        <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
        <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
        <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
        <uses-feature android:name="android.hardware.location" android:required="false" />
        <uses-feature android:name="android.hardware.location.gps" android:required="false" />
        <uses-feature android:name="android.hardware.location.network" android:required="false" />
                                                </manifest>
```

APÊNDICE B — App.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<Application xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        x:Class="polygonalGeofencing.App">
   <Application.Resources>

   </Application.Resources>
                                                </Application>
```

APÊNDICE C — App.xaml.cs

```
using polygonalGeofencing.Helpers;
using System;
using System.IO;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
using polygonalGeofencing.Views;
namespace polygonalGeofencing
{
    public partial class App : Application
    {

        static SQLiteDataBaseHelper dataBase;

        public static SQLiteDataBaseHelper database
        {
            get
            {
                if (dataBase == null) //se não foi criado o banco
                {
                    //irá criar o arquivo onde é armazenado os dados da aplicação
                    dataBase = new
SQLiteDataBaseHelper(Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicatio
nData), "XamAppGeo.db3"));
                }

                return dataBase; //retorna o banco
            }
        }

        public App() //Construtor
        {
            InitializeComponent(); //Componente de inicialização
            MainPage = new AppShell(); //a página raiz da aplicação recebe a tela a ser exibida
        }

        protected override void OnStart()
        {
        }

        protected override void OnSleep()
        {
        }

        protected override void OnResume()
        {
        }
    }
}
```

APÊNDICE D — AppShell.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<Shell xmlns="http://xamarin.com/schemas/2014/forms"
           xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
           xmlns:Views="clr-namespace:polygonalGeofencing.Views"
           x:Class="polygonalGeofencing.AppShell">


    <!--Shell Item, recebe a nomenclatura para navegar até a página-->
    <ShellItem Route="Login">
        <ShellContent ContentTemplate="{DataTemplate Views:Login}"/>
    </ShellItem>

    <ShellItem  Route="CadastroArea">
        <ShellContent ContentTemplate="{DataTemplate Views:CadastroArea}"/>
    </ShellItem>

    <ShellItem Route="CadastroFazenda">
        <ShellContent ContentTemplate="{DataTemplate Views:CadastroFazenda}"/>
    </ShellItem>
    <ShellItem Route="EditAreaModal">
        <ShellContent ContentTemplate="{DataTemplate Views:EditAreaModal}"/>
    </ShellItem>
    <ShellItem Route="CadastroPontos">
        <ShellContent ContentTemplate="{DataTemplate Views:CadastroPontos}"/>
    </ShellItem>
    <ShellItem Route="CadastroFuncionario">
        <ShellContent ContentTemplate="{DataTemplate Views:CadastroFuncionario}"/>
    </ShellItem>

    <ShellItem Route="ListaFuncionario">
        <ShellContent ContentTemplate="{DataTemplate Views:ListaFuncionario}"/>
    </ShellItem>


    <!--TabBar elemento guia para roteamento entre páginas -->
    <TabBar>
        <Tab Title="Mapa" Icon="iconMapa" Route="Mapa">
            <ShellContent ContentTemplate="{DataTemplate Views:Mapa}" />
        </Tab>
        <Tab Title="Inicio" Route="Home" Icon="home">
            <ShellContent ContentTemplate="{DataTemplate Views:Home}" />
        </Tab>
        <Tab Title="Perfil" Icon="person" Route="Perfil">
            <ShellContent ContentTemplate="{DataTemplate Views:Perfil}" />
        </Tab>
    </TabBar>

</Shell>
```

APÊNDICE E — AppShell.xaml.cs

```
using polygonalGeofencing.Views;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;

namespace polygonalGeofencing
{
    public partial class AppShell : Shell
    {
        public AppShell()
        {
            InitializeComponent();
            Routing.RegisterRoute("entrarModal", typeof(entrarModal));
            Routing.RegisterRoute("cadastroModal", typeof(cadastroModal));
            Routing.RegisterRoute("AcessoFuncModal", typeof(AcessoFuncModal));
        }

    }
}
```

APÊNDICE F — Helpers

Arquivo – SQLiteDataBaseHelper,cs

```
using System;
using System.Collections.Generic;
using System.Text;
using SQLite;
using polygonalGeofencing.Models;
using System.Threading.Tasks;

namespace polygonalGeofencing.Helpers
{
    public class SQLiteDataBaseHelper
    {
        readonly SQLiteAsyncConnection _db;


        public SQLiteDataBaseHelper(string dbPath)
        {
            _db = new SQLiteAsyncConnection(dbPath);
            _db.CreateTableAsync<Acesso>().Wait();
            _db.CreateTableAsync<Area>().Wait();
            _db.CreateTableAsync<Fazenda>().Wait();
            _db.CreateTableAsync<Logado>().Wait();
            _db.CreateTableAsync<Pontos>().Wait();
```

```
            _db.CreateTableAsync<Usuario>().Wait();



        }



        /************* Comandos Acesso ***************/

        //Pega todos acessos do usuário do usuário em uma fazenda especifica ou em qualquer uma
        public Task<List<Acesso>> GetAcessoUsuario(int idUsuario,int idFazenda)
        {
            string sql = "";
            //se for passado um id valido e selecionado os acessos daquela fazenda
            if (idFazenda >= 0)
            {
                sql = "SELECT * from Acesso where idUsuario = " + idUsuario + " AND idFazenda = " + idFazenda;
            }
            else // se não seleciona todos acessos
            {
                sql = "SELECT * from Acesso where idUsuario = " + idUsuario;
            }

            return _db.QueryAsync<Acesso>(sql);
        }

        //Insere o acesso do usuário
        public Task<List<Acesso>> InsertAcesso(int idUsuario, int idTipo, int idFazenda)
        {
            string sql = "INSERT INTO Acesso (idUsuario, idTipo, idFazenda) VALUES ('" + idUsuario + "', '" +
idTipo + "', '" + idFazenda + "')";
            return _db.QueryAsync<Acesso>(sql);
        }

        //Remove o acesso do usuário
        public Task<List<Acesso>> DeleteAcessoUsuario(int idUsuario,int idTipo,int idFazenda)
        {
            string sql = "DELETE from Acesso where idUsuario = '" + idUsuario + "' AND idTipo = '" + idTipo + "'
AND idFazenda = '" + idFazenda + "'";
            return _db.QueryAsync<Acesso>(sql);
        }

        //Remove todos acessos do banco de dados local
        public Task<List<Acesso>> DelAllAcessos()
        {
            string sql = "DELETE from Acesso";
            return _db.QueryAsync<Acesso>(sql);
        }

        /*******************************************/
```

```
        /************* Comandos Usuários *************/

        //Seleciona o usuário a partir do id
        public Task<Usuario> GetByIdUsuario(int id)
        {
            return _db.Table<Usuario>().FirstAsync(i => i.idUsuario == id);
        }

        //Seleciona o Usuário a partir do e-mail e da senha
        public Task<List<Usuario>> GetUsuario(string email, string senha)
        {
            string sql = "SELECT * from Usuario where email = '" + email + "' and senha = '" + senha + "' AND
ativo = 1";
            return _db.QueryAsync<Usuario>(sql);
        }

        //Seleciona o usuário a partir do e-mail
        public Task<List<Usuario>> verificarEmail(string email)
        {
            string sql = "SELECT * from Usuario where email = '" + email + "' AND ativo = 1";
            return _db.QueryAsync<Usuario>(sql);
        }

        //Seleciona os funcionarios da fazenda
        public Task<List<Usuario>> GetFuncionariosFazenda(string idFazenda,int idLogado, int idDono)
        {
            string sql = "";
            if (idDono == idLogado)
            {
                sql = "SELECT * from Usuario where idDono = " + idDono + " AND idDono <> idUsuario AND
ativo = 1";
            }
            else
            {
                if (idFazenda.Contains(","))
                {
                    idFazenda.Replace(",", "','"); //caso seja 10,20,30 -> '10','20','30'
                    sql = " select distinct u.idUsuario,u.nome,u.sobrenome from usuario u " +
                        " left outer join acesso a on a.idUsuario = u.idUsuario " +
                        " where(a.idFazenda in '" + idFazenda + "' or a.idfazenda is null) and u.ativo = 1 and and
u.idDono = " + idDono + " u.idUsuario not in (" + idLogado + "," + idDono + ") ";
                }
                else
                {

                    sql = " select distinct u.idUsuario,u.nome,u.sobrenome from usuario u " +
                        " left outer join acesso a on a.idUsuario = u.idUsuario " +
                        " where(a.idFazenda = " + idFazenda + " or a.idfazenda is null) and u.ativo = 1 and u.idDono
= " + idDono + " and u.idUsuario not in (" + idLogado  + "," + idDono + ") ";
                }
            }


            return _db.QueryAsync<Usuario>(sql);
```

```
}

//Deleta todos usuários
public Task<List<Usuario>> DelAllUsuarios()
{
   string sql = "DELETE FROM Usuario";
   return _db.QueryAsync<Usuario>(sql);
}

//Insere o usuário no banco de dados local
public Task<int> CadastroUsuario(Usuario model)
{
   return _db.InsertAsync(model);
}

//Atualiza informações do usuário
public Task<List<Usuario>> UpdateDono(int idDono,int idUsuario)
{
   string sql = "UPDATE Usuario SET idDono = " + idDono + " WHERE idUsuario = " + idUsuario;
   return _db.QueryAsync<Usuario>(sql);
}

//Atualiza o status de ativo do usuário
public Task<List<Usuario>> ExcluirUsuario(int idUsuario)
{
   string sql = "UPDATE Usuario SET ativo = 0 WHERE idUsuario = " + idUsuario;
   return _db.QueryAsync<Usuario>(sql);
}

/*******************************************/



/************* Comandos Usuário Logado *************/

//Seleciona o usuário logado
public Task<List<Logado>> GetLogado()
{
   string sql = "SELECT * from Logado";
   return _db.QueryAsync<Logado>(sql);
}

//Deleta o usuário logado
public Task<List<Logado>> DelLogado()
{
   string sql = "DELETE FROM Logado";
   return _db.QueryAsync<Logado>(sql);
}

//Insere o usuário logado
public Task<int> InserirLogado(Logado model)
{
   return _db.InsertAsync(model);
}
```

```
        /*****************************************/



        /************* Comandos Fazenda *************/

        //Seleciona a fazenda especificada
        public Task<Fazenda> GetByIdFazenda(int id)
        {
            return _db.Table<Fazenda>().FirstAsync(i => i.idFazenda == id);
        }

        //Seleciona a fazenda a partir do nome
        public Task<List<Fazenda>> GetByNomeFazenda(string nome,int idDono)
        {
            string sql = "SELECT * FROM Fazenda WHERE nome like '%"+ nome + "%' and idUsuario = " +
idDono;
            return _db.QueryAsync<Fazenda>(sql);
        }

        //Deleta todas fazendas do banco de dados
        public Task<List<Fazenda>> DelFazenda()
        {
            string sql = "DELETE FROM Fazenda";
            return _db.QueryAsync<Fazenda>(sql);
        }

        public Task<List<Fazenda>> GetFazendaUsuario(int idDono)
        {
            string sql = "SELECT * from Fazenda where idUsuario = " + idDono;
            return _db.QueryAsync<Fazenda>(sql);
        }

        //Pegar as fazenda que o Funcionario tem acesso
        public Task<List<Fazenda>> GetFazendaFuncionario(int idFuncionario,int idDono)
        {
            string sql = "";
            if (idFuncionario == idDono)
            {
                sql = " SELECT distinct(f.idFazenda), f.nome from Fazenda f " +
                      " WHERE f.idUsuario = " + idDono;
            }
            else{
                 sql = " SELECT distinct(f.idFazenda), f.nome from Fazenda f " +
                    " INNER JOIN Acesso a ON a.idFazenda = f.idFazenda " +
                    " WHERE a.idUsuario = " + idFuncionario;
            }

            return _db.QueryAsync<Fazenda>(sql);
        }

        //Inserir fazenda
        public Task<int> InsertFazenda(Fazenda model)
        {
            return _db.InsertAsync(model);
```

```
}

//Deletar fazenda
public Task<int> DeleteFazenda(int idFazenda)
{
   return _db.Table<Fazenda>().DeleteAsync(i => i.idFazenda == idFazenda);
}


/*****************************************/



/************** Comandos Pontos **************/

//Deleta todos os pontos
public Task<List<Pontos>> DelPontos()
{
   string sql = "DELETE FROM Pontos";
   return _db.QueryAsync<Pontos>(sql);
}

//Deleta todos pontos da área
public Task<List<Pontos>> DellPontosArea(int idArea)
{
   string sql = "DELETE FROM Pontos WHERE idArea = " + idArea;
   return _db.QueryAsync<Pontos>(sql);
}

//Seleciona todos os pontos da área
public Task<List<Pontos>> GetPontosArea(int idArea)
{
   string sql = "SELECT * from Pontos WHERE idArea = " + idArea;
   return _db.QueryAsync<Pontos>(sql);

}

//Insere os pontos
public Task<int> InsertPonto(Pontos model)
{
   return _db.InsertAsync(model);
}

/*****************************************/


/************** Comandos Área **************/

//Pega todas as áreas da fazenda
public Task<List<Area>> GetAllRowsArea(int idFazenda)
{
   string sql = "SELECT * from Area where idFazenda = " + idFazenda;
   return _db.QueryAsync<Area>(sql);
}
```

```
//Atualiza as informações da área
public Task<List<Area>> UpdateArea(int idArea,string desc)
{
   string sql = "UPDATE Area SET descricao = '" + desc + "' WHERE idArea = " + idArea;
   return _db.QueryAsync<Area>(sql);
}

//Remove todas as áreas do banco de dados
public Task<List<Area>> DelArea()
{
   string sql = "DELETE FROM Area";
   return _db.QueryAsync<Area>(sql);
}

//Pega todas as áreas de um usuário
public Task<List<Area>> GetAreasUsuario(int id)
{
   string sql = "SELECT a.* from Area a " +
      " Inner Join Fazenda f on f.idFazenda = a.idFazenda  where f.idUsuario = " + id + "  ";
   return _db.QueryAsync<Area>(sql);
}

//Pega todas as áreas das fazemdas
public Task<List<Area>> GetAreasFazenda(string idFazenda)
{
   string sql = "";
   if (idFazenda.Contains(","))
   {

      idFazenda.Replace(",", "','"); //caso seja 10,20,30 -> '10','20','30'
      sql = " SELECT * from Area a " +
           " Inner Join Fazenda f on f.idFazenda = a.idFazenda  " +
           " where a.idFazenda IN ('" + idFazenda + "')";
   }
   else
   {
      sql = " SELECT * from Area a " +
           " Inner Join Fazenda f on f.idFazenda = a.idFazenda  " +
           " where a.idFazenda = " +  Int32.Parse(idFazenda);
   }

   return _db.QueryAsync<Area>(sql);
}


//Seleciona a área a partir do id
public Task<Area> GetByIdArea(int id)
{

   return _db.Table<Area>().FirstAsync(i => i.idArea == id);
}


// Insere a área
```

```
    public Task<int> InsertArea(Area model)
    {
        return _db.InsertAsync(model);
    }

    //Remove a área
    public Task<int> DeleteArea(int idArea)
    {
        return _db.Table<Area>().DeleteAsync(i => i.idArea == idArea);
    }

    /*******************************************/
}
                                                    }
```

APÊNDICE G — Models

Arquivo – Acesso.cs

```
using SQLite;
using System;
using System.Collections.Generic;
using System.Text;

namespace polygonalGeofencing.Models
{
    public class Acesso
    {
        [PrimaryKey, AutoIncrement] //determina que a variavel a baixo será auto increment
        public int idAcesso { get; set; }  //identificador unico do Acesso

        public int idUsuario { get; set; } // identificador do usuario

        // 1 - Criar Area 2 - Add Funcionario 3 - Add Permissão
        public int idTipo { get; set; } // identificador do tipo de permisão

        public int idFazenda { get; set; } // identificador da Fazenda em que tem acesso

    }
}
```

Arquivo – Area.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using SQLite;

namespace polygonalGeofencing.Models
{
    public class Area
    {
        [PrimaryKey,AutoIncrement] //determina que a variavel a baixo será auto increment
        public int idArea { get; set; } //identificador unico da área

        public int idFazenda { get; set; } //identificador da fazenda

        public string nome { get; set; } //nome da área
        public string descricao { get; set; } //informações da área


    }
}
```

Arquivo – Fazenda.cs

```
using SQLite;
using System;
using System.Collections.Generic;
using System.Text;

namespace polygonalGeofencing.Models
{
    public class Fazenda
    {
        [PrimaryKey, AutoIncrement] //determina que a variavel a baixo será auto increment
        public int idFazenda { get; set; } //identificador unico da fazenda

        public int idUsuario { get; set; } //identificador unico do Usuario

        public string nome { get; set; } //nome da fazenda

    }
}
```

Arquivo – Logado.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using SQLite;

namespace polygonalGeofencing.Models
{
    public class Logado
    {
        [PrimaryKey,AutoIncrement] //determina que a variável a baixo será AutoIncrement

        public int idLogado { get; set; } //identificador Logado

        public int idUsuario { get; set; } //identificador do Usuário

        public DateTime data { get; set; } //variável data

    }
}
```

Arquivo – Pontos.cs

```
using SQLite;
using System;
using System.Collections.Generic;
using System.Text;

namespace polygonalGeofencing.Models
{
    public class Pontos
    {
        [PrimaryKey, AutoIncrement] //determina que a variavel a baixo será auto increment
        public int idPonto { get; set; } //identificador de cada ponto

        public int idArea { get; set; } //identificador da area

        public double? latitude { get; set; } //latitude do ponto

        public double? longitude { get; set; } //longitude do ponto
    }
}
```

Arquivo – Usuario.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using SQLite;

namespace polygonalGeofencing.Models
{
    public class Usuario
    {
        [PrimaryKey, AutoIncrement] //determina que a variável a baixo será AutoIncrement
        public int idUsuario { get; set; } //identificador do Usuário

        public int idDono { get; set; } //identificador do Dono da Fazenda

        public int ativo { get; set; } //informa se o usuário esta ativo ou não no sistema

        public string nome { get; set; } //Nome do usuário

        public string sobrenome { get; set; } //Sobrenome do usuário

        public string email { get; set; } //email do usuário

        public string senha { get; set; } //senha do usuário

    }
}
```

APÊNDICE H — ViewModels

Arquivo – CadastroAreaViewModel.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Text;
using System.Windows.Input;
using polygonalGeofencing.Views;
using Xamarin.Forms;
using polygonalGeofencing.Models;
using System.Collections.ObjectModel;

namespace polygonalGeofencing.ViewModels
{
    //recebimento do id da fazenda
    [QueryProperty(nameof(PegarId), "parametro_id")]
    class CadastroAreaViewModel : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;


        ObservableCollection<Area> ListaArea = new ObservableCollection<Area>();


        public ObservableCollection<Area> listaArea
        {
            get => ListaArea;
            set => ListaArea = value;
        }


        string txtFazenda, txtArea, txtDesc;
        int idfazenda;

        public int idFazenda
        {
            get => idfazenda;
            set
            {
                idfazenda = value;
                PropertyChanged(this, new PropertyChangedEventArgs("idFazenda"));
            }
        }


        public string nomeArea
        {
            get => txtArea;
            set
            {
```

```
        txtArea = value;
        PropertyChanged(this, new PropertyChangedEventArgs("nomeArea"));
    }
}


public string descArea
{
    get => txtDesc;
    set
    {
        txtDesc = value;
        PropertyChanged(this, new PropertyChangedEventArgs("descArea"));
    }
}


public string nomeFazenda
{
    get => txtFazenda;
    set
    {
        txtFazenda = value;
        PropertyChanged(this, new PropertyChangedEventArgs("nomeFazenda"));
    }
}

public string PegarId
{
    set
    {
        int id_parametro = Convert.ToInt32(Uri.UnescapeDataString(value));
        //chama a função mostrar fazenda passando como parametro o id da fazenda
        mostrarFazenda.Execute(id_parametro);
    }
}


public ICommand CadastrarPontos
{
    get
    {
        return new Command<int>(async (int id) =>
        {
            await Shell.Current.GoToAsync($"//CadastroPontos?parametro_id={id}");
        });
    }
}


public ICommand EditarArea
{
    get
    {
        return new Command<Area>(async (area) =>
```

```
            {
                //redireciona para a página para editar a descrição da área passando o id da área como
parametro
                await Shell.Current.GoToAsync($"//EditAreaModal?parametro_id={area.idArea}");
            });
        }
    }

    public ICommand AddGeo
    {
        get
        {
            return new Command<Area>(async (area) =>
            {
                //redireciona para a página para selecionar a área passando o id da área como parametro
                await Shell.Current.GoToAsync($"//CadastroPontos?parametro_id={area.idArea}");
            });
        }
    }

    public ICommand RemoverArea
    {
        get
        {
            return new Command<Area>(async (area) =>
            {
                try
                {
                    //envia um alerta para a confirmação da exclusão
                    bool conf = await Application.Current.MainPage.DisplayAlert("Tem Certeza?", "Excluir", "Sim",
"Não");

                    if (conf)//se o usuário confirmar
                    {
                        //a área e deletada do banco de dados
                        await App.database.DeleteArea(area.idArea);
                        //todos os pontos adicionados a area e excluido
                        await App.database.DellPontosArea(area.idArea);
                        //atualiza a área
                        AtualizarLista.Execute(null);
                    }

                }
                catch (Exception ex)
                {
                    await Application.Current.MainPage.DisplayAlert("Ops", ex.Message, "OK");

                }

            });
        }
    }

    public ICommand mostrarFazenda
    {
```

```
    get => new Command<int>(async (int id) =>
    {
        try
        {
            //pega as informações da fazenda no banco de dados
            Fazenda model = await App.database.GetByIdFazenda(id);
            /*adiciona o nome da fazenda em uma label para exibir
            para o usuário em qual fazenda ele esta adicionando a area*/
            this.nomeFazenda = "Fazenda " + model.nome;
            this.idFazenda = model.idFazenda;//coloca o id da fazenda em um campo oculto
            AtualizarLista.Execute(null); //chama a função atualizar lista
        }
        catch (Exception ex)
        {
            await App.Current.MainPage.DisplayAlert("ERRO", ex.Message, "OK");
        }

    });
}




public ICommand novaArea
{
    get => new Command(async () =>
    {
        try
        {
            //a classe recebe as informações digitadas pelo usuário
            Area model = new Area()
            {
                nome = this.txtArea,
                descricao = this.txtDesc,
                idFazenda = this.idfazenda
            };
            //insere as informações no banco de dados
            await App.database.InsertArea(model);
            //emite um alerta para usuário informando do sucesso do cadastro
            await Application.Current.MainPage.DisplayAlert("Sucesso", "Area Cadastrada", "ok");
            //chama a função para atualizar a listaao
            AtualizarLista.Execute(null);
        }
        catch (Exception ex)
        {
            await Application.Current.MainPage.DisplayAlert("Ops", ex.Message, "ok");
        }
    });
}
public ICommand AtualizarLista
{
    get
    {
        return new Command(async () =>
        {
```

```
                try
                {
                    //Pega todas as áreas cadastradas para a fazenda em referencia
                    List<Area> tmp = await App.database.GetAllRowsArea(this.idFazenda);
                    //Limpa a lista
                    listaArea.Clear();
                    //Para cada dado retornado do banco é adicionado na lista
                    tmp.ForEach(i => listaArea.Add(i));

                }
                catch (Exception ex)
                {
                    await Application.Current.MainPage.DisplayAlert("Ops", ex.Message, "OK");

                }

            });
        }
    }



    public ICommand Voltar
    {
        get
        {
            return new Command (() =>
            {
                try
                {
                    Shell.Current.GoToAsync("//CadastroFazenda");

                }
                catch (Exception ex)
                {
                    Application.Current.MainPage.DisplayAlert("Ops", ex.Message, "OK");

                }

            });
        }
    }

}
}
```

Arquivo – CadastroFazendaViewModel.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Text;
using System.Windows.Input;
using Xamarin.Forms;
using polygonalGeofencing.Models;
using polygonalGeofencing.Views;
using System.Collections.ObjectModel;
using System.Linq;

namespace polygonalGeofencing.ViewModels
{

    class CadastroFazendaViewModel : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;


        ObservableCollection<Fazenda> ListaFazendas = new ObservableCollection<Fazenda>();




        public ObservableCollection<Fazenda> listaFazendas
        {
            get => ListaFazendas;
            set => ListaFazendas = value;
        }






        string nomeFazenda;
        int id;


        public string Nome
        {
            get => nomeFazenda;
            set
            {
                nomeFazenda = value;
                PropertyChanged(this, new PropertyChangedEventArgs("nomeFazenda"));
            }
        }

        public int Id
```

```
    {
        get => id;
        set
        {
            id = value;
            PropertyChanged(this, new PropertyChangedEventArgs("Id"));
        }
    }


    public ICommand NovaFazenda
    {
        get => new Command(async () => {

            try
            {
                List<Logado> logado = await App.database.GetLogado();

                    Fazenda model = new Fazenda()
                    {
                        nome = this.nomeFazenda,
                        idUsuario = logado.ToList()[0].idUsuario
                    };

                    await App.database.InsertFazenda(model);

                await Application.Current.MainPage.DisplayAlert("Sucesso", "Fazenda Cadastrada", "ok");
                    AtualizarLista.Execute(null);

            }
            catch(Exception ex)
            {
              await Application.Current.MainPage.DisplayAlert("Ops", ex.Message, "ok");
            }




        });
    }



    public ICommand CadastrarArea
    {
        get
        {
            return new Command<Fazenda>(async (fazenda) =>
            {
                //redirecionamento para a página de cadastro de area passando como parametro o id da
fazenda
                await Shell.Current.GoToAsync($"//CadastroArea?parametro_id={fazenda.idFazenda}");
            });
        }
    }
```

```
    public ICommand Voltar
    {
        get
        {
            return new Command (async () =>
            {
                await Shell.Current.GoToAsync("//Home");
            });
        }
    }

    public ICommand AtualizarLista
    {
        get
        {
            return new Command(async () =>
            {
                try
                {

                    List<Fazenda> fazendas = null;
                    //pega o usuário logado
                    List<Logado> logado = await App.database.GetLogado();
                    //pega as informações do usuário
                    Usuario usuario = await App.database.GetByIdUsuario(logado.ToList()[0].idUsuario);


                    fazendas = await App.database.GetFazendaFuncionario(logado.ToList()[0].idUsuario,
usuario.idDono);



                    //limpa a lista
                    listaFazendas.Clear();
                    if(fazendas != null)
                    {
                        //cria a lista de fazendas cadastradas
                        fazendas.ForEach(i => listaFazendas.Add(i));

                    }


                }
                catch (Exception ex)
                {
                    await Application.Current.MainPage.DisplayAlert("Ops", ex.Message, "OK");

                }

            });
        }
    }
```

```
        public ICommand RemoverFazenda
        {
            get
            {
                return new Command<Fazenda>(async (fazenda) =>
                {
                    try
                    {
                        List<Logado> logado = await App.database.GetLogado();
                        Usuario usuario = await App.database.GetByIdUsuario(logado[0].idUsuario);

                        //se o usuário logado não for o dono ele não pode remover fazenda
                        if (usuario.idUsuario == usuario.idDono)
                        {
                            //Alerta para confirmação da exclusão
                            bool conf = await Application.Current.MainPage.DisplayAlert("Tem Certeza?", "Excluir",
"Sim", "Não");

                            //se o usuário confirmar a exclusão
                            if (conf)
                            {
                                //exclui a fazenda do banco de dados
                                await App.database.DeleteFazenda(fazenda.idFazenda);
                                //chama a função para recarregar a lista
                                AtualizarLista.Execute(null);
                            }
                        }
                        else
                        {
                            await Application.Current.MainPage.DisplayAlert("Ops","Somente o proprietario pode
remover a fazenda", "OK");
                        }
                    }
                    catch (Exception ex)
                    {
                        await Application.Current.MainPage.DisplayAlert("Ops", ex.Message, "OK");
                    }
                });
            }
        }




    }
}
```

Arquivo – CadastroPontosViewModel.cs

```
using polygonalGeofencing.Models;
using polygonalGeofencing.Views;
using System;
using System.Collections.Generic;
```

```
using System.ComponentModel;
using System.Globalization;
using System.Text;
using System.Windows.Input;
using Xamarin.Forms;
using Xamarin.Forms.Maps;

namespace polygonalGeofencing.ViewModels
{
   [QueryProperty(nameof(PegarIdNavegacao), "parametro_id")]
    class CadastroPontosViewModel : INotifyPropertyChanged
   {
      public event PropertyChangedEventHandler PropertyChanged;

      string txtArea;
      int idarea;

      public int idArea
      {
         get => idarea;
         set
         {
            idarea = value;
            PropertyChanged(this, new PropertyChangedEventArgs("idArea"));
         }
      }


      public string nomeArea
      {
         get => txtArea;
         set
         {
            txtArea = value;
            PropertyChanged(this, new PropertyChangedEventArgs("nomeArea"));
         }
      }


      public string PegarIdNavegacao
      {
         set
         {
            int id_parametro = Convert.ToInt32(Uri.UnescapeDataString(value));
            //chama a função mostrar área passando o id da área como parametro
            mostrarArea.Execute(id_parametro);
         }
      }




      public ICommand mostrarArea
      {
```

```
    get => new Command<int>(async (int id) =>
    {
        try
        {
            //pega as informações da área no banco de dados
            Area model = await App.database.GetByIdArea(id);
            //mostra na tela o nome da área
            this.nomeArea = "Geofencing " + model.nome;
            this.idArea = model.idArea;
        }
        catch (Exception ex)
        {
            await Application.Current.MainPage.DisplayAlert("Ops", ex.Message, "ok");
        }

    });
}



public ICommand Voltar
{
    get => new Command(() =>
    {
        try
        {
            Shell.Current.GoToAsync("//CadastroArea");
        }
        catch (Exception ex)
        {
            Application.Current.MainPage.DisplayAlert("Ops", ex.Message, "ok");
        }

    });
}

public ICommand cadastrarPonto
{
    get => new Command<string>(async (string posicao) => {
        try
        {
            //pega a string recebida por parametro e separa a latitude e longitude
            string latitude = posicao.Split(',')[0];
            string longitude = posicao.Split(',')[1];

            //insere as informações na classe
            Pontos model = new Pontos()
            {
                idArea = this.idArea,
                // Utiliza-se o CultureInfo para ponto seja considerar o . como separador decimal
                latitude = Convert.ToDouble(latitude, new CultureInfo("en-US")),
                longitude = Convert.ToDouble(longitude, new CultureInfo("en-US"))
            };
            //insere a informação no banco
            await App.database.InsertPonto(model);
```

```
            }
            catch (Exception ex)
            {
                await Application.Current.MainPage.DisplayAlert("Ops", ex.Message, "ok");
            }
        });
    }


}
}
```

Arquivo – CadastroUsuarioViewModel.cs

```
using polygonalGeofencing.Models;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Windows.Input;
using Xamarin.Forms;

namespace polygonalGeofencing.ViewModels
{
    public class CadastroUsuarioViewModel : INotifyPropertyChanged
    {

      public event PropertyChangedEventHandler PropertyChanged;


       public ICommand Voltar
       {
           get => new Command(() =>
           {
               try
               {
                   Shell.Current.GoToAsync("//Home");
               }
               catch (Exception ex)
               {
                   Application.Current.MainPage.DisplayAlert("Ops", ex.Message, "ok");
               }

           });
       }

       string nome, sobrenome, senha, email, confsenha;

       public string Nome //nome da variavel que esta no Binding
       {
           get => nome;
```

```
    set
    {
        nome = value; //a variavel nome criada recebera o valor digitado pelo usuario
        PropertyChanged(this, new PropertyChangedEventArgs("nomeUsuario"));
    }
}
public string Sobrenome //sobrenome da variavel que esta no Binding
{
    get => sobrenome;
    set
    {
        sobrenome = value; //a variavel nome criada recebera o valor digitado pelo usuario
        PropertyChanged(this, new PropertyChangedEventArgs("sobrenomeUsuario"));
    }
}

public string Email
{
    get => email;
    set
    {
        email = value;  //a variavel email criada recebera o valor digitado pelo usuario
        PropertyChanged(this, new PropertyChangedEventArgs("emailUsuario"));
    }
}

public string Senha
{
    get => senha;
    set
    {
        senha = value; //a variavel senha criada recebera o valor digitado pelo usuario
        PropertyChanged(this, new PropertyChangedEventArgs("senhaUsuario"));
    }
}
public string confSenha
{
    get => confsenha;
    set
    {
        confsenha = value;
        PropertyChanged(this, new PropertyChangedEventArgs("confSenhaUsuario"));
    }
}

public ICommand NovoUsuario
{
    get => new Command(async () => {

        try
        {

            //A classe usuário recebe as informações do usuário
            Usuario model = new Usuario()
            {
```

```
                nome = this.nome,
                sobrenome = this.sobrenome,
                senha = this.senha,
                email = this.email,
                ativo = 1
            };

            //verifica se os campos foram preenchidos
            if (nome != null && sobrenome != null && senha != null && email != null && confSenha !=
null)
            {
                //verifica se as senhas são iguais
                if (senha == confSenha)
                {
                    //verifica se o e-mail informado ja esta cadastrado
                    List<Usuario> verEmail = await App.database.verificarEmail(email);
                    if (verEmail.Count == 0)
                    {
                        //insere as informações do usuario no banco
                        await App.database.CadastroUsuario(model);

                        //pega o id do dono para inserir na informação do usuário
                        List<Logado> logado = await App.database.GetLogado();
                        Usuario idDono = await App.database.GetByIdUsuario(logado.ToList()[0].idUsuario);
                        List<Usuario> idUsuario = await App.database.GetUsuario(email,senha);
                        //adiciona o id do proprietario no registro do usuário
                        await App.database.UpdateDono(idDono.idDono, idUsuario.ToList()[0].idUsuario);

                        await App.Current.MainPage.DisplayAlert("Sucesso", "Funcionário Cadastrado", "OK");
                    }
                    else
                    {
                        await Application.Current.MainPage.DisplayAlert("Falha", "E-mail já cadastrado", "Ok");
                    }

                }
                else
                {
                    await Application.Current.MainPage.DisplayAlert("Falha", "Senhas diferentes", "ok");
                }
            }
            else
            {
                await Application.Current.MainPage.DisplayAlert("Falha", "Preencha Todos os campos", "ok");
            }


        }
        catch (Exception ex)
        {
            await Application.Current.MainPage.DisplayAlert("Ops", ex.Message, "ok");
        }


    });
```

```
        }


    }
}
```

Arquivo – ControleAcesso.cs

```
using polygonalGeofencing.Models;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Windows.Input;
using Xamarin.Forms;


namespace polygonalGeofencing.ViewModels
{

    [QueryProperty(nameof(PegarId), "parametro_id")]
    public class ControleAcesso : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;


        public string PegarId
        {
            set
            {
                int id_parametro = Convert.ToInt32(Uri.UnescapeDataString(value));
                mostrarFuncionario.Execute(id_parametro);
            }
        }



        string funcionario, nomeFazenda;
        int idfuncionario;
        public string Funcionario
        {
            get => funcionario;
            set
            {
                funcionario = value;
                PropertyChanged(this, new PropertyChangedEventArgs("Funcionario"));
            }
        }

        public int idFuncionario
        {
```

```
    get => idfuncionario;
    set
    {
        idfuncionario = value;
        PropertyChanged(this, new PropertyChangedEventArgs("idFuncionario"));
    }
}


public ICommand mostrarFuncionario
{
    get
    {
        return new Command<int>(async (usuarioid) =>
        {
            try
            {

                Usuario funcionario = await App.database.GetByIdUsuario(usuarioid);

                this.Funcionario = funcionario.nome;
                this.idFuncionario = funcionario.idUsuario;
            }
            catch (Exception ex)
            {
                await Application.Current.MainPage.DisplayAlert("Ops", ex.Message, "OK");

            }

        });
    }
}



public ICommand Voltar
{
    get
    {
        return new Command(() =>
        {
            try
            {
                Shell.Current.GoToAsync("//ListaFuncionario");

            }
            catch (Exception ex)
            {
                Application.Current.MainPage.DisplayAlert("Ops", ex.Message, "OK");

            }

        });
    }
}
```

```
    }
}
```

Arquivo – EditarAreaViewModel.cs

```
using polygonalGeofencing.Models;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Text;
using System.Windows.Input;
using Xamarin.Forms;

namespace polygonalGeofencing.ViewModels
{

    [QueryProperty(nameof(PegarId), "parametro_id")]
    public class EditarAreaViewModel : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;



        string txtDesc, txtmostrarNome;
        int idarea;

        public int idArea
        {
            get => idarea;
            set
            {
                idarea = value;
                PropertyChanged(this, new PropertyChangedEventArgs("idArea"));
            }
        }


        public string mostrarNome
        {
            get => txtmostrarNome;
            set
            {
                txtmostrarNome = value;
                PropertyChanged(this, new PropertyChangedEventArgs("mostrarNome"));
            }
        }

        public string descArea
        {
            get => txtDesc;
            set
```

```
    {
        txtDesc = value;
        PropertyChanged(this, new PropertyChangedEventArgs("descArea"));
    }
}


public string PegarId
{
    set
    {
        int id_parametro = Convert.ToInt32(Uri.UnescapeDataString(value));
        //chama a função para mostrar as informações da área
        mostrarArea.Execute(id_parametro);
    }
}
public ICommand mostrarArea
{
    get => new Command<int>(async (int id) =>
    {
        try
        {
            //pega no banco as informações da área
            Area model = await App.database.GetByIdArea(id);
            this.mostrarNome =  model.nome; //mostra na tela o nome da área
            this.idArea = model.idArea; //guarda o id da área
            //formata a descrição para a melhor visualização
            this.descArea = String.Format(model.descricao.Replace(",", "\n\n"));
        }
        catch (Exception ex)
        {
            await App.Current.MainPage.DisplayAlert("ERRO", ex.Message, "OK");
        }

    });
}


public ICommand editarArea
{
    get => new Command (async () =>
    {
        try
        {
            //recebe o campo da descrição trocando as quebras de linha por ','
            string descricao = String.Format(this.descArea.Replace("\n\n", ","));
            //atualiza a informação no banco
            await App.database.UpdateArea(this.idArea, descricao);
            //dispara um alerta informando que a alteração foi realizada
            await App.Current.MainPage.DisplayAlert("Sucesso", "Área Atualizada", "OK");
        }
        catch (Exception ex)
        {
            await App.Current.MainPage.DisplayAlert("ERRO", ex.Message, "OK");
        }
```

```
        });
    }

    public ICommand Voltar
    {
        get => new Command(() =>
        {
            try
            {
                Shell.Current.GoToAsync("//CadastroArea");
            }
            catch (Exception ex)
            {
                App.Current.MainPage.DisplayAlert("ERRO", ex.Message, "OK");
            }

        });
    }



  }
}
```

Arquivo – HomeViewModel.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Text;
using System.Windows.Input;
using polygonalGeofencing.Views;
using Xamarin.Forms;
using polygonalGeofencing.Models;
using System.Collections.ObjectModel;

namespace polygonalGeofencing.ViewModels
{

  class HomeViewModel : INotifyPropertyChanged
  {
    public event PropertyChangedEventHandler PropertyChanged;

    public ICommand Sair
    {
        get => new Command(async () => {

            try
            {
                await App.database.DelLogado(); //limpa a tabela logado
                await Shell.Current.GoToAsync("//Login");//redireciona para a tela de login
            }
```

```
            catch (Exception ex)
            {
                await Application.Current.MainPage.DisplayAlert("Ops", ex.Message, "ok");
            }

        });
    }


}
}
```

Arquivo – ListaFuncionarioViewModel.cs

```
using polygonalGeofencing.Models;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Windows.Input;
using Xamarin.Forms;

namespace polygonalGeofencing.ViewModels
{

    public class ListaFuncionarioViewModel : INotifyPropertyChanged
    {

        public event PropertyChangedEventHandler PropertyChanged;


        ObservableCollection<Usuario> ListaFuncionario = new ObservableCollection<Usuario>();


        public ObservableCollection<Usuario> listaFuncionario
        {
            get => ListaFuncionario;
            set => ListaFuncionario = value;
        }




        public ICommand EditarFuncionario
        {
            get
            {
                return new Command<Usuario>(async (funcionario) =>
                {
                    List<Logado> logado = await App.database.GetLogado();
                    Usuario usuario = await App.database.GetByIdUsuario(logado.ToList()[0].idUsuario);
```

```
            List<Fazenda> fazendas = await App.database.GetFazendaUsuario(usuario.idDono);

            if (fazendas.Count > 0)
            {
                await
Shell.Current.GoToAsync($"//ListaFuncionario/AcessoFuncModal?parametro_id={funcionario.idUsuario}");
            }
            else
            {
                await App.Current.MainPage.DisplayAlert("Ops", "Nenhuma Fazenda Criada", "OK");
            }


        });
    }
}


public ICommand AtualizarLista
{
    get
    {
        return new Command(async () =>
        {
            try
            {
                //pega o id do usuário logado
                List<Logado> logado = await App.database.GetLogado();
                //pega as informações do usuário
                Usuario usuario = await App.database.GetByIdUsuario(logado.ToList()[0].idUsuario);
                //pega todos os funcionarios do proprietario das fazendas em que o usuário logado tem
permissão
                List<Fazenda> fazendas = await
App.database.GetFazendaFuncionario(usuario.idUsuario,usuario.idDono);

                string str = "";


                //colocar na variavel str todas as fazendas em que o usuário tem acesso
                foreach (Fazenda fazenda in fazendas.ToList())
                {
                    str = fazenda.idFazenda + ",";
                }

                List<Usuario> funcionarios = await App.database.GetFuncionariosFazenda(str.Substring(0,
str.Length - 1), usuario.idUsuario, usuario.idDono);
                //limpa a lista
                listaFuncionario.Clear();
                //para cada registro de funcionario no banco e adicionado na lista
                funcionarios.ForEach(i => listaFuncionario.Add(i));

            }
            catch (Exception ex)
```

```
                {
                    await Application.Current.MainPage.DisplayAlert("Ops", ex.Message, "OK");

                }

            });
        }
    }


    public ICommand RemoverFuncionario
    {
        get
        {
            return new Command<Usuario>(async (usuario) =>
            {
                try
                {
                    //Alerta de confirmação para exclusão
                    bool conf = await Application.Current.MainPage.DisplayAlert("Tem Certeza?", "Excluir", "Sim",
"Não");

                    if (conf) //se form confirmado
                    {
                        //o usuário e setado como ativo = 0
                        await App.database.ExcluirUsuario(usuario.idUsuario);
                        //atualiza a lista
                        AtualizarLista.Execute(null);
                    }

                }
                catch (Exception ex)
                {
                    await Application.Current.MainPage.DisplayAlert("Ops", ex.Message, "OK");

                }

            });
        }
    }

    public ICommand Voltar
    {
        get
        {
            return new Command(() =>
            {
                try
                {
                     Shell.Current.GoToAsync("//Home");

                }
                catch (Exception ex)
                {
                     Application.Current.MainPage.DisplayAlert("Ops", ex.Message, "OK");
```

```
                }

            });
        }
    }



}
}
```

Arquivo – LoginUsuarioViewModel.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows.Input;
using Xamarin.Forms;
using polygonalGeofencing.Views;
using System.ComponentModel;
using polygonalGeofencing.Models;
using System.Linq;
using System.Threading.Tasks;

namespace polygonalGeofencing.ViewModels
{
    public class LoginUsuarioViewModel : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;

        string nome ,sobrenome, senha, email,confsenha;

        public string Nome //nome da variavel que esta no Binding
        {
            get => nome;
            set
            {
                nome = value; //a variavel nome criada recebera o valor digitado pelo usuario
                PropertyChanged(this, new PropertyChangedEventArgs("nomeUsuario"));
            }
        }
        public string Sobrenome //sobrenome da variavel que esta no Binding
        {
            get => sobrenome;
            set
            {
                sobrenome = value; //a variavel nome criada recebera o valor digitado pelo usuario
                PropertyChanged(this, new PropertyChangedEventArgs("sobrenomeUsuario"));
            }
        }

        public string Email
```

```
    {
        get => email;
        set
        {
            email = value;  //a variavel email criada recebera o valor digitado pelo usuario
            PropertyChanged(this, new PropertyChangedEventArgs("emailUsuario"));
        }
    }

    public string Senha
    {
        get => senha;
        set
        {
            senha = value; //a variavel senha criada recebera o valor digitado pelo usuario
            PropertyChanged(this, new PropertyChangedEventArgs("senhaUsuario"));
        }
    }
    public string confSenha
    {
        get => confsenha;
        set
        {
            confsenha = value;
            PropertyChanged(this, new PropertyChangedEventArgs("confSenhaUsuario"));
        }
    }




    public ICommand Entrar //recebe o comando do botão entrar
    {
        get => new Command(async () => {
            try
            {
                List<Usuario> model = await App.database.GetUsuario(this.email, this.senha); //consulta no
banco as credenciais informadas pelo usuário
                if (model.ToList().Count > 0) //caso a lista recebida seja maior que 0, ou seja conseguiu achar
informação no banco
                {
                    await App.database.DelLogado(); //deleta o registro de usuário logado
                    Logado Logado = new Logado() //classe Logado, para registrar qual usuário esta logado,
                                        //para que na proxima vez que entrar no aplicativo logue direto
                    {
                        idUsuario = model.ToList()[0].idUsuario, //insere o id do usuario que esta logando
                        data = DateTime.Now //adiciona a data atual para que no proximo login seja verificado se
a sessão dele expirou
                    };
                    await App.database.InserirLogado(Logado); //insere as informações no banco
                    await Shell.Current.GoToAsync("//Home");

                    //App.Current.MainPage = new AppShell(); //redireciona o usuário para a homePage
                }
                else
```

```
                {
                    await Application.Current.MainPage.DisplayAlert("Erro", "Email ou senha invalida", "OK");
//credenciais não encontradas no banco
                }
            }
            catch (Exception ex)
            {
                await Application.Current.MainPage.DisplayAlert("Ops", ex.Message, "ok"); //caso alguma
instrução de erro
            }




        });
    }


    public ICommand NovoUsuario
        {
            get => new Command(async () => {

                try
                {

                    Usuario model = new Usuario() //instancia um novo objeto atribuindo os valores digitados
                        {
                            nome = this.nome,
                            sobrenome = this.sobrenome,
                            senha = this.senha,
                            email = this.email,
                            ativo = 1
                        };

                    //se os campos não forem nulos
                    if (nome != null && sobrenome != null && senha != null && email != null && confSenha !=
null)
                    {
                        //verifica se as senhas são compativeis
                        if(senha == confSenha)
                        {
                            //procura no banco o e-mail digitado para ver se ja existe cadastro com o e-mail
                            List<Usuario> verEmail =  await App.database.verificarEmail(email);
                            if (verEmail.Count == 0)//caso o e-mail não esteja cadastrado
                            {
                                await App.database.CadastroUsuario(model);//insere as informações do usuário no
banco
                                List<Usuario> usuario = await App.database.GetUsuario(email, senha); //pega os
dados do usuário cadastrado
                                int idDono = usuario.ToList()[0].idUsuario; //pega o id do Usuário cadastrado
                                await App.database.UpdateDono(idDono, idDono); //atualiza os dados do usuário
colocando o id dele como dono

                                //com o sucesso do cadastro do usuário e perguntado a ele se ele ja deseja fazer o
login
```

```
                        bool answer = await Application.Current.MainPage.DisplayAlert("Usuário Cadastrado",
"Deseja fazer o login", "Sim", "Não");
                        if (answer) //caso a resposta seja sim
                        {
                            await App.database.DelLogado(); //limpa a tabela logado
                            Logado Logado = new Logado()
                            {
                                idUsuario = usuario.ToList()[0].idUsuario, //insere na tabela logado o id do
usuário
                                data = DateTime.Now //coloca o data e a hora do login
                            };

                            await App.database.InserirLogado(Logado);

                            await Shell.Current.GoToAsync("//Home");//usuário e redirecionado para a home
page
                        }
                    }
                    else
                    {
                        //caso o e-mail já for cadastrado e disparado um alerta para o usuário
                        await Application.Current.MainPage.DisplayAlert("Falha", "E-mail já cadastrado",
"Ok");
                    }

                }
                else
                {
                    //caso as senhas digitadas não forem iguais e disparado um alerta para o usuário
                    await Application.Current.MainPage.DisplayAlert("Falha", "Senhas diferentes", "ok");
                }
            }
            else
            {
                await Application.Current.MainPage.DisplayAlert("Falha", "Preencha Todos os campos",
"ok");
            }


        }
        catch(Exception ex)
        {
            //se ocorrer algum erro durante o processo de cadastro e disparado um alerta com o erro
            await Application.Current.MainPage.DisplayAlert("Ops", ex.Message, "ok");
        }




    });
}


}
}
```

Arquivo – PerfilViewModel.cs

```
using polygonalGeofencing.Models;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Windows.Input;
using Xamarin.Forms;

namespace polygonalGeofencing.ViewModels
{
    public class PerfilViewModel : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;

        string nome, sobrenome, email;

        public string Nome //nome da variavel que esta no Binding
        {
            get => nome;
            set
            {
                nome = value; //a variavel nome criada recebera o valor digitado pelo usuario
                PropertyChanged(this, new PropertyChangedEventArgs("Nome"));
            }
        }
        public string Sobrenome //sobrenome da variavel que esta no Binding
        {
            get => sobrenome;
            set
            {
                sobrenome = value; //a variavel nome criada recebera o valor digitado pelo usuario
                PropertyChanged(this, new PropertyChangedEventArgs("Sobrenome"));
            }
        }

        public string Email
        {
            get => email;
            set
            {
                email = value;  //a variavel email criada recebera o valor digitado pelo usuario
                PropertyChanged(this, new PropertyChangedEventArgs("Email"));
            }
        }

        public ICommand mostrarPerfil
        {
            get => new Command(async () => {

                try
                {
```

```
                List<Logado> logado = await App.database.GetLogado(); //pega o id do usuário logado
                //pega as informações do usuário
                Usuario usuario = await App.database.GetByIdUsuario(logado.ToList()[0].idUsuario);

                this.Nome = usuario.nome;
                this.Sobrenome = usuario.sobrenome;
                this.Email = usuario.email;
            }
            catch (Exception ex)
            {
                await Application.Current.MainPage.DisplayAlert("Ops", ex.Message, "ok");
            }


        });
    }

  }
}
```

APÊNDICE I — View

Arquivo – AcessoFuncModal.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             Shell.NavBarIsVisible="False"
             xmlns:viewModels="clr-
namespace:polygonalGeofencing.ViewModels;assembly=polygonalGeofencing"
             Shell.PresentationMode="ModalAnimated"
             x:Class="polygonalGeofencing.Views.AcessoFuncModal">


    <ContentPage.BindingContext>
        <viewModels:ControleAcesso/>
    </ContentPage.BindingContext>

    <Grid RowDefinitions="Auto,Auto,Auto">
        <!--Botao para voltar-->
        <StackLayout  VerticalOptions="Center" HorizontalOptions="Start" WidthRequest="400"
HeightRequest="60" Grid.Row="0" >
            <Frame BackgroundColor="Transparent">
                <Frame.GestureRecognizers>
                    <TapGestureRecognizer Tapped="Voltar_Tapped" Command="{Binding Voltar}"/>
                </Frame.GestureRecognizers>
            </Frame>
            <FlexLayout Margin="13,-30,0,0">
                <ImageButton Source="voltar" x:Name="Voltar" WidthRequest="15" HeightRequest="20"
Command="{Binding Voltar}"  BackgroundColor="Transparent"></ImageButton>
                <Label Text="Voltar" TextColor="Black" Margin="3,1,0,0" ></Label>
```

```
        </FlexLayout>
    </StackLayout>
    <!--Título da tela contendo a chamada e o nome do funcionário-->
    <StackLayout Grid.Row="1" HorizontalOptions="Center" Margin="0,-40,0,0">
        <Label Text="Controle de Permissões" FontSize="30"  HorizontalTextAlignment="Center"
FontFamily="Metropolis" TextColor="#4A484D" Margin="0,50,0,0"></Label>
        <Label Text="{Binding Funcionario}" FontSize="30"  HorizontalTextAlignment="Center"
FontFamily="Metropolis" TextColor="#4A484D" Margin="0,0,0,0"></Label>
        <Label x:Name="idFuncionario" Text="{Binding idFuncionario}" IsVisible="false"></Label>
    </StackLayout>
    <!--área do picker com as fazenda e os switchs para atribuir ou revogar permissão-->
    <StackLayout Grid.Row="2" HeightRequest="300" WidthRequest="300" Padding="40,10">
        <Label Text="{Binding NomeFazenda}" IsVisible="false"></Label>
        <!--picker com a lista de fazendas que o usuário possui permissão-->
        <Picker Title="Fazenda"
                x:Name="listaFazenda"
                SelectedIndexChanged="listaFazenda_SelectedIndexChanged"/>

        <!--Switch para que o funcionário possa adicionar áreas na fazenda-->
            <FlexLayout Direction="Row" AlignItems="Center"
            JustifyContent="SpaceBetween" Margin="0,20,0,0">
                <Label Text="Criar Áreas" FontSize="20" TextColor="Black"></Label>


                <Switch
                    x:Name="AcessoArea"
                    Toggled="AcessoArea_Toggled"
                     >
                    <VisualStateManager.VisualStateGroups>
                        <VisualStateGroup x:Name="CommonStates">
                            <VisualState x:Name="On">
                                <VisualState.Setters>
                                    <Setter Property="ThumbColor"
                                Value="MediumSpringGreen" />
                                </VisualState.Setters>
                            </VisualState>
                            <VisualState x:Name="Off">
                                <VisualState.Setters>
                                    <Setter Property="ThumbColor"
                                Value="Red" />
                                </VisualState.Setters>
                            </VisualState>
                        </VisualStateGroup>
                    </VisualStateManager.VisualStateGroups>
                </Switch>
             </FlexLayout>

        <!--Switch para que o funcionário possa adicionar funcionários-->
        <FlexLayout Direction="Row" AlignItems="Center"
            JustifyContent="SpaceBetween">
            <Label Text="Adicionar Funcionarios" FontSize="20" TextColor="Black"></Label>
            <Switch
                x:Name="AcessoAddFunc"
                Toggled="AcessoAddFunc_Toggled">
                <VisualStateManager.VisualStateGroups>
```

```
            <VisualStateGroup x:Name="CommonStates">
              <VisualState x:Name="On">
                <VisualState.Setters>
                  <Setter Property="ThumbColor"
              Value="MediumSpringGreen" />
                </VisualState.Setters>
              </VisualState>
              <VisualState x:Name="Off">
                <VisualState.Setters>
                  <Setter Property="ThumbColor"
              Value="Red" />
                </VisualState.Setters>
              </VisualState>
            </VisualStateGroup>
          </VisualStateManager.VisualStateGroups>
        </Switch>
      </FlexLayout>

      <!--Switch para que o funcionário possa atribuir ou revogar permissões do funcionários-->
      <FlexLayout Direction="Row" AlignItems="Center"
        JustifyContent="SpaceBetween">
        <Label Text="Atribuir Permissões" FontSize="20" TextColor="Black"></Label>
        <Switch x:Name="AcessoAddPerm"
              Toggled="AcessoAddPerm_Toggled">
          <VisualStateManager.VisualStateGroups>
            <VisualStateGroup x:Name="CommonStates">
              <VisualState x:Name="On">
                <VisualState.Setters>
                  <Setter Property="ThumbColor"
              Value="MediumSpringGreen" />
                </VisualState.Setters>
              </VisualState>
              <VisualState x:Name="Off">
                <VisualState.Setters>
                  <Setter Property="ThumbColor"
              Value="Red" />
                </VisualState.Setters>
              </VisualState>
            </VisualStateGroup>
          </VisualStateManager.VisualStateGroups>
        </Switch>
      </FlexLayout>

    </StackLayout>


</Grid>


                                        </ContentPage>
```

Arquivo – AcessoFuncModal.xaml.cs

```
using polygonalGeofencing.Models;
using polygonalGeofencing.ViewModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace polygonalGeofencing.Views
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class AcessoFuncModal : ContentPage
    {
        bool flag = false;
        public AcessoFuncModal()
        {
            InitializeComponent();
            BindingContext = new ControleAcesso();

        }

        protected override async void OnAppearing()
        {
            //flag para controle de permissões
            flag = false;
            List<Logado> logado = await App.database.GetLogado();
            Usuario usuario = await App.database.GetByIdUsuario(logado.ToList()[0].idUsuario);

            //pega as fazendas que o usuário possui permissão
            List<Fazenda> fazendas = await App.database.GetFazendaFuncionario(usuario.idUsuario,
usuario.idDono);

            //se o resultado da busca for maior que 0
            if(fazendas.Count > 0)
            {
                //adiciona no picker as fazendas
                foreach (Fazenda fazenda in fazendas.ToList())
                {
                    listaFazenda.Items.Add(fazenda.nome);
                }

            }
            else
            {
                //se não tiver registro oculta o campo da lista de fazenda
                listaFazenda.IsVisible = false;
            }

            //chama a função de controle de acesso
```

```
    ControleAcesso();
}

private void Voltar_Tapped(object sender, EventArgs e)
{

}


//evento chamado ao trocar o estado do switch da área
private void AcessoArea_Toggled(object sender, ToggledEventArgs e)
{
    var troca = AcessoArea.IsToggled;
    AtualizaAcesso(troca, 1);
}

//evento chamado ao trocar o estado do switch de adicionar funcionário
private void AcessoAddFunc_Toggled(object sender, ToggledEventArgs e)
{
    var troca = AcessoAddFunc.IsToggled;
    AtualizaAcesso(troca, 2);
}

//evento chamado ao trocar o estado do switch de adicionar permissões para o funcionário
private void AcessoAddPerm_Toggled(object sender, ToggledEventArgs e)
{
    var troca = AcessoAddPerm.IsToggled;

    AtualizaAcesso(troca, 3);
}


//evento ao trocar de item no dropdown das fazendas
private void listaFazenda_SelectedIndexChanged(object sender, EventArgs e)
{
    ControleAcesso();
}

//função para atualizar acessos, recebe como parametro o estado do switch e o tipo de permissão
public async void AtualizaAcesso(bool Acesso, int Tipoid)
{
    //recebe a fazenda selecionada no dropdown
    var fazenda = listaFazenda.SelectedItem;
    List<Logado> logado = await App.database.GetLogado();
    Usuario usuario = await App.database.GetByIdUsuario(logado.ToList()[0].idUsuario);

    if (!flag)
    {
        if (listaFazenda.IsVisible)
        {
            //se tiver fazenda selecionada
            if (fazenda != null)
            {
                //pega os dados da fazenda
```

```
                List<Fazenda> modelFazenda = await App.database.GetByNomeFazenda(fazenda.ToString(),
usuario.idDono);
                int idFazenda = modelFazenda[0].idFazenda;

                //se o estado do switch for ativo
                if (Acesso) //Concedeu Acesso
                {
                    //adiciona o acesso para o funcionário
                    await App.database.InsertAcesso(Int32.Parse(idFuncionario.Text), Tipoid, idFazenda);
                    await App.Current.MainPage.DisplayAlert("Sucesso", "Acesso Liberado", "OK");
                }
                else //retirou acesso
                {
                    //retira o acesso do funcionário
                    await App.database.DeleteAcessoUsuario(Int32.Parse(idFuncionario.Text), Tipoid,
idFazenda);
                    await App.Current.MainPage.DisplayAlert("Sucesso", "Acesso Revogado", "OK");
                }
            }
            else
            {
                //se não selecionou nenhuma fazenda e disparado um alerta para o usuário selecionar a
fazenda
                await App.Current.MainPage.DisplayAlert("Ops", "Selecione a Fazenda", "OK");
                flag = true;
                //reverte a ação do usuário
                switch (Tipoid)
                {
                    case 1:
                        AcessoArea.IsToggled = !AcessoArea.IsToggled;
                        break;
                    case 2:
                        AcessoAddFunc.IsToggled = !AcessoAddFunc.IsToggled;
                        break;
                    case 3:
                        AcessoAddPerm.IsToggled = !AcessoAddPerm.IsToggled;
                        break;

                }
                flag = false;
            }
        }
        else
        {
            await Shell.Current.GoToAsync("//ListaFuncionario");
        }
    }
    else{
        flag = false;
    }
}

//função de controle de acesso
public async void ControleAcesso()
{
```

```
        //recebe a fazenda selecionada no dropdown
        var fazenda = listaFazenda.SelectedItem;
        List<Logado> logado = await App.database.GetLogado();
        Usuario usuario = await App.database.GetByIdUsuario(logado.ToList()[0].idUsuario);

        //se tiver fazenda selecionada
        if (fazenda != null)
        {
            //pega os dados da fazenda no banco
            List<Fazenda> modelFazenda = await
App.database.GetByNomeFazenda(fazenda.ToString(),usuario.idDono);
            int idFazenda = modelFazenda[0].idFazenda;

            //pega os acessos do funcionário
            List<Acesso> Acessos = await App.database.GetAcessoUsuario(Int32.Parse(idFuncionario.Text),
idFazenda);

            //troca o estado dos switchs de acordo com acessos do funcionário
            if(Acessos.Count > 0)
            {
                foreach (Acesso acesso in Acessos.ToList())
                {

                    switch (acesso.idTipo)
                    {
                        case 1:
                            flag = true;
                            AcessoArea.IsToggled = true;
                            break;
                        case 2:
                            flag = true;
                            AcessoAddFunc.IsToggled = true;
                            break;
                        case 3:
                            flag = true;
                            AcessoAddPerm.IsToggled = true;
                            break;


                    }
                        flag = false;
                }
            }
            else
            {
                flag = true;
                AcessoArea.IsToggled = false;
                flag = true;
                AcessoAddFunc.IsToggled = false;
                flag = true;
                AcessoAddPerm.IsToggled = false;
                flag = false;

            }
        }
```

```
        else
        {
            flag = true;
            AcessoArea.IsToggled = false;
            flag = true;
            AcessoAddFunc.IsToggled = false;
            flag = true;
            AcessoAddPerm.IsToggled = false;
            flag = false;

        }


    }


}
                                                    }
```

Arquivo – CadastroArea.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="polygonalGeofencing.Views.CadastroArea"
             xmlns:viewModels="clr-
namespace:polygonalGeofencing.ViewModels;assembly=polygonalGeofencing"
             Shell.NavBarIsVisible="False"
             x:Name="Pagina">


    <ContentPage.BindingContext>
        <viewModels:CadastroAreaViewModel/>
    </ContentPage.BindingContext>

    <!--grid para organizar os elementos-->
    <Grid RowDefinitions="Auto,Auto,Auto">
        <!--botão voltar-->
        <StackLayout  VerticalOptions="Center" HorizontalOptions="Start" WidthRequest="400"
HeightRequest="60" Grid.Row="0" >
            <Frame BackgroundColor="Transparent">
                <Frame.GestureRecognizers>
                    <TapGestureRecognizer Tapped="TapGestureRecognizer_Tapped" Command="{Binding
Voltar}"/>
                </Frame.GestureRecognizers>
            </Frame>
            <FlexLayout Margin="13,-30,0,0">
                <ImageButton Source="voltar" WidthRequest="15" HeightRequest="20" Command="{Binding
Voltar}" BackgroundColor="Transparent"></ImageButton>
                <Label Text="Voltar" TextColor="Black" Margin="3,1,0,0" ></Label>
            </FlexLayout>
        </StackLayout>
        <StackLayout Grid.Row="1" HorizontalOptions="Center" Margin="0,-40,0,0">
```

```
        <Label Text="Cadastro de Area" FontSize="40"  HorizontalTextAlignment="Center"
FontFamily="Metropolis" TextColor="#4A484D" Margin="0,50,0,0"></Label>
        <!--Campo para receber o nome da fazenda-->
        <Label Text="{Binding nomeFazenda}" FontSize="20"  HorizontalTextAlignment="Center"
FontFamily="Metropolis" TextColor="#4A484D" ></Label>
        <!--campo para receber o id da fazenda-->
        <Label Text="{Binding idFazenda}" IsVisible="false"></Label>
        <!--campo para colocar o nome da área-->
        <Frame CornerRadius="30" BackgroundColor="#f2f2f2" HasShadow="False" WidthRequest="310"
HeightRequest="15" Margin="0,50,0,10">
          <Entry x:Name="cadastroNome" Text="{Binding nomeArea}" HorizontalTextAlignment="Start"
FontSize="14" Placeholder="Nome da Área" HeightRequest="50" TextColor="Black"
PlaceholderColor="#868787" Margin="15,-30,0,-30"></Entry>
        </Frame>
        <!--campo para colocar a descrição da área-->
        <Frame CornerRadius="30" BackgroundColor="#f2f2f2" HasShadow="False" WidthRequest="310"
HeightRequest="55" Margin="0,10,0,10">
          <Editor x:Name="cadastroDesc" Text="{Binding descArea}" AutoSize="TextChanges"
FontSize="14" Placeholder="Descrição&#10;Formato -> tipo: valor,  " HeightRequest="70"
TextColor="Black" PlaceholderColor="#868787" Margin="15,0,0,-30"></Editor>
        </Frame>
      </StackLayout>
      <!--botão para salvar as informações-->
      <StackLayout Grid.Row="2" Margin="0,20,0,0"  >
        <Button x:Name="NovaArea" Text="Cadastrar" BackgroundColor="#2595ef" TextColor="white"
WidthRequest="345" HeightRequest="55" HorizontalOptions="Center" CornerRadius="30"
Margin="0,0,0,0" Command="{Binding novaArea}"></Button>
      </StackLayout>

      <!--lista para mostrar as áreas cadastradas-->
      <ListView ItemsSource="{Binding listaArea}" Grid.Row="3" HasUnevenRows="True">
        <ListView.ItemTemplate>
          <DataTemplate>
            <ViewCell>
              <StackLayout  Padding="30,10" BackgroundColor="#F1F1F1">
                <Grid RowDefinitions="100" ColumnDefinitions="Auto,*,Auto,Auto,Auto,Auto"
VerticalOptions="Center">
                  <!--id da área em um campo oculto-->
                  <Label Text="{Binding idArea}" IsVisible="false"></Label>
                  <!--imagem representativa da área-->
                  <Image Source="cerca" Grid.Row="0" Grid.Column="0" FlowDirection="LeftToRight"
HeightRequest="50" WidthRequest="50" ></Image>
                  <!--nome da área-->
                  <Label Text="{Binding nome}" Grid.Row="0" Grid.Column="1" TextColor="Black"
VerticalTextAlignment="Center" HorizontalTextAlignment="Center"></Label>
                  <!--botão para editra a descrição da área-->
                  <ImageButton x:Name="btnEditArea" Source="editar" Margin="10,0,0,0"
WidthRequest="25" HeightRequest="25" BackgroundColor="Transparent" Grid.Row="0" Grid.Column="3"
Clicked="btnEditArea_Clicked" />
                  <!--botão para adicionar a região da área-->
                  <ImageButton x:Name="btnAddGeo" Source="addPoint" Margin="0,0,10,0"
WidthRequest="22" HeightRequest="22" BackgroundColor="Transparent" Grid.Row="0" Grid.Column="4"
Clicked="btnAddGeo_Clicked" />
                  <!--botão para excluir a área-->
```

```
                        <ImageButton x:Name="btnExcluir" Source="lixeira" WidthRequest="15"
HeightRequest="15" BackgroundColor="Transparent" Grid.Row="0" Grid.Column="5"
Clicked="btnExcluir_Clicked" />

                    </Grid>
                </StackLayout>

            </ViewCell>
        </DataTemplate>
    </ListView.ItemTemplate>
</ListView>

</Grid>


</ContentPage>
```

Arquivo – CadastroArea.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
using polygonalGeofencing.ViewModels;
using polygonalGeofencing.Models;

namespace polygonalGeofencing.Views
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class CadastroArea : ContentPage
    {

        public CadastroArea()
        {
            InitializeComponent();
            BindingContext = new CadastroAreaViewModel();
        }
        protected override void OnAppearing() //ao inicializar a pagina, quando ela for exibida
        {
            var vm = (CadastroAreaViewModel)BindingContext;
        }

        private void btnExcluir_Clicked(object sender, EventArgs e)
        {
            var button = sender as ImageButton;
            var area = button?.BindingContext as Area;
            var vm = BindingContext as CadastroAreaViewModel;

            vm?.RemoverArea.Execute(area);
        }


        private void btnEditArea_Clicked(object sender, EventArgs e)
        {
            var button = sender as ImageButton;
            var area = button?.BindingContext as Area;
            var vm = BindingContext as CadastroAreaViewModel;

            vm?.EditarArea.Execute(area);
        }


        private void btnAddGeo_Clicked(object sender, EventArgs e)
        {
            var button = sender as ImageButton;
            var area = button?.BindingContext as Area;
            var vm = BindingContext as CadastroAreaViewModel;
```

```
            vm?.AddGeo.Execute(area);
        }


        private void TapGestureRecognizer_Tapped(object sender, EventArgs e)
        {

        }
    }
                                                    }
```

Arquivo –CadastroFazenda.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="polygonalGeofencing.Views.CadastroFazenda"
             x:Name="Pagina"
             xmlns:viewModels="clr-
namespace:polygonalGeofencing.ViewModels;assembly=polygonalGeofencing"
             Shell.NavBarIsVisible="False">

    <ContentPage.BindingContext>
        <viewModels:CadastroFazendaViewModel/>
    </ContentPage.BindingContext>

        <!--Grid para organizar os elementos-->
        <Grid RowDefinitions="Auto,Auto,Auto">

            <!--elemento contendo o botão para voltar para a home-->
        <StackLayout  VerticalOptions="Center" HorizontalOptions="Start" WidthRequest="400"
HeightRequest="60" Grid.Row="0" >
            <Frame BackgroundColor="Transparent">
                <Frame.GestureRecognizers>
                    <TapGestureRecognizer Tapped="TapGestureRecognizer_Tapped" Command="{Binding
Voltar}"/>
                </Frame.GestureRecognizers>
            </Frame>
            <FlexLayout Margin="13,-30,0,0">
                <ImageButton Source="voltar" WidthRequest="15" HeightRequest="20" Command="{Binding
Voltar}" BackgroundColor="Transparent"></ImageButton>
                <Label Text="Voltar" TextColor="Black" Margin="3,1,0,0" ></Label>
            </FlexLayout>
        </StackLayout>

        <StackLayout Grid.Row="1" HorizontalOptions="Center" Margin="0,-40,0,0">
            <Label Text="Cadastrar Fazenda" FontSize="40"  HorizontalTextAlignment="Center"
FontFamily="Metropolis" TextColor="#4A484D" Margin="0,50,0,0"></Label>
            <Label Text="Insira suas informações para cadastrar" HorizontalTextAlignment="Center"
FontSize="15" FontFamily="Metropolis" TextColor="#7C7D7E" Margin="0,0,0,0"></Label>
           <!--campo para o usuário digitar o nome da fazenda-->
            <Frame CornerRadius="30" BackgroundColor="#f2f2f2" HasShadow="False" WidthRequest="310"
HeightRequest="15" Margin="0,50,0,10">
```

```
            <Entry x:Name="cadastroNome" Text="{Binding Nome}" HorizontalTextAlignment="Start"
FontSize="14" Placeholder="Nome da Fazenda" HeightRequest="50" TextColor="Black"
PlaceholderColor="#868787" Margin="15,-30,0,-30"></Entry>
         </Frame>
      </StackLayout>

         <!--Botão para cadastrar a fazenda-->
      <StackLayout Grid.Row="2" Margin="0,20,0,0"  >
         <Button x:Name="NovaFazenda" Text="Cadastrar" BackgroundColor="#2595ef" TextColor="white"
WidthRequest="345" HeightRequest="55" HorizontalOptions="Center" CornerRadius="30"
Margin="0,0,0,0" Command="{Binding NovaFazenda}"></Button>
      </StackLayout>

      <!--Lista de fazendas cadastradas-->
      <ListView ItemsSource="{Binding listaFazendas}" Grid.Row="3" HasUnevenRows="True">
            <ListView.ItemTemplate>
               <DataTemplate>
                  <ViewCell>

                  <StackLayout  Padding="30,10" BackgroundColor="#F1F1F1">
                     <Grid RowDefinitions="100" ColumnDefinitions="Auto,*,Auto,Auto"
VerticalOptions="Center">
                        <!--id da fazenda com isVisible false, o usuário não visualisa esse campo-->
                        <Label Text="{Binding idFazenda}" IsVisible="false"></Label>
                        <!--imagem do icone da fazenda-->
                        <Image Source="fazenda" Grid.Row="0" Grid.Column="0" FlowDirection="LeftToRight"
HeightRequest="30" WidthRequest="30" ></Image>
                        <!--nome da fazenda cadastrada-->
                        <Label Text="{Binding nome}" Grid.Row="0" Grid.Column="1" TextColor="Black"
VerticalTextAlignment="Center" HorizontalTextAlignment="Center" FlowDirection="RightToLeft"></Label>
                        <!--botão para cadastrar area na fazenda-->
                        <ImageButton x:Name="btnAddArea" Source="addArea" Margin="10,0,0,0"
WidthRequest="30" HeightRequest="30" BackgroundColor="Transparent" Grid.Row="0" Grid.Column="2"
Clicked="btnAddArea_Clicked" />
                        <!--botão para excluir a fazenda-->
                        <ImageButton x:Name="btnExcluir" Source="lixeira" WidthRequest="15"
HeightRequest="15" BackgroundColor="Transparent" Grid.Row="0" Grid.Column="3"
Clicked="btnExcluir_Clicked" />
                     </Grid>
                  </StackLayout>

               </ViewCell>
               </DataTemplate>
            </ListView.ItemTemplate>
      </ListView>

   </Grid>


                                    </ContentPage>
```

Arquivo –CadastroFazenda.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Input;
using polygonalGeofencing.Models;
using polygonalGeofencing.ViewModels;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace polygonalGeofencing.Views
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class CadastroFazenda : ContentPage
    {
        public CadastroFazenda()
        {
            InitializeComponent();
            BindingContext = new CadastroFazendaViewModel();
        }


        protected async override void OnAppearing() //ao inicializar a pagina, quando ela for exibida
        {

            List<Logado> logado = await App.database.GetLogado();
            Usuario usuario = await App.database.GetByIdUsuario(logado[0].idUsuario);

            //se o usuário logado não for o dono ele não pode cadastrar fazenda
            if(usuario.idUsuario != usuario.idDono)
            {
                cadastroNome.IsEnabled = false;
                NovaFazenda.IsEnabled = false;

            }
            else
            {
                cadastroNome.IsEnabled = true;
                NovaFazenda.IsEnabled = true;
            }

            var vm = (CadastroFazendaViewModel)BindingContext;
            //chama a função atualizar lista da viewmodel para carregar as fazendas cadastradas
            vm.AtualizarLista.Execute(null);
        }



        private void btnExcluir_Clicked(object sender, EventArgs e)
        {
            var button = sender as ImageButton;
```

```
        var fazenda = button?.BindingContext as Fazenda;
        var vm = BindingContext as CadastroFazendaViewModel;

        vm?.RemoverFazenda.Execute(fazenda);
    }

    private void btnAddArea_Clicked(object sender, EventArgs e)
    {
        var button = sender as ImageButton;
        //pega o id da fazenda
        var fazenda = button?.BindingContext as Fazenda;
        var vm = BindingContext as CadastroFazendaViewModel;
        //chama a função cadastrarArea passando como parametro o id da fazenda
        vm?.CadastrarArea.Execute(fazenda);
    }


    private async void TapGestureRecognizer_Tapped(object sender, EventArgs e)
    {
        //botão voltar para Home
        await Shell.Current.GoToAsync("//Home");
    }
}
                                                    }
```

Arquivo –CadastroFuncionario.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             Shell.NavBarIsVisible="False"
             x:Class="polygonalGeofencing.Views.CadastroFuncionario">


    <StackLayout Margin="10">
        <Grid RowDefinitions="Auto,Auto">
            <!--botão voltar-->
            <StackLayout  VerticalOptions="Center" HorizontalOptions="Start" WidthRequest="400"
HeightRequest="60" Grid.Row="0" >
                <Frame BackgroundColor="Transparent">
                    <Frame.GestureRecognizers>
                        <TapGestureRecognizer Tapped="TapGestureRecognizer_Tapped" Command="{Binding
Voltar}"/>
                    </Frame.GestureRecognizers>
                </Frame>
                <FlexLayout Margin="13,-30,0,0">
                    <ImageButton Source="voltar" WidthRequest="15" HeightRequest="20" Command="{Binding
Voltar}" BackgroundColor="Transparent"></ImageButton>
                    <Label Text="Voltar" TextColor="Black" Margin="3,1,0,0" ></Label>
                </FlexLayout>
            </StackLayout>
            <StackLayout Grid.Row="1" HorizontalOptions="Center" Margin="0,-30,0,0" >
                <!--Titulo-->
```

```
            <Label Text="Cadastrar Funcionário" FontSize="40" HorizontalTextAlignment="Center"
FontFamily="Metropolis" TextColor="#4A484D" Margin="0,50,0,0"></Label>
            <!--Campo nome do funcionario-->
            <Frame CornerRadius="30" BackgroundColor="#f2f2f2" HasShadow="False"
WidthRequest="310" HeightRequest="15" Margin="0,50,0,10">
                <Entry x:Name="cadastroNome" Text="{Binding Nome}"  HorizontalTextAlignment="Start"
FontSize="14" Placeholder="Nome" HeightRequest="50" TextColor="Black" PlaceholderColor="#868787"
Margin="15,-30,0,-30">
                    <Entry.Keyboard>
                        <Keyboard x:FactoryMethod="Create">
                            <x:Arguments>
                                <KeyboardFlags>Suggestions,CapitalizeWord</KeyboardFlags>
                            </x:Arguments>
                        </Keyboard>
                    </Entry.Keyboard>
                </Entry>
            </Frame>
            <!--Campo sobrenome do funcionario-->
            <Frame CornerRadius="30" BackgroundColor="#f2f2f2" HasShadow="False"
WidthRequest="310" HeightRequest="15" Margin="0,0,0,10">
                <Entry x:Name="cadastroSobranome"  Text="{Binding Sobrenome }"
HorizontalTextAlignment="Start" FontSize="14" Placeholder="Sobrenome" HeightRequest="50"
TextColor="Black" PlaceholderColor="#868787" Margin="15,-30,0,-30">
                    <Entry.Keyboard>
                        <Keyboard x:FactoryMethod="Create">
                            <x:Arguments>
                                <KeyboardFlags>Suggestions,CapitalizeWord</KeyboardFlags>
                            </x:Arguments>
                        </Keyboard>
                    </Entry.Keyboard>
                </Entry>
            </Frame>
            <!--Campo email do funcionario-->
            <Frame CornerRadius="30" BackgroundColor="#f2f2f2" HasShadow="False"
WidthRequest="310" HeightRequest="15" Margin="0,0,0,10">
                <Entry x:Name="cadastroEmail" Text="{Binding Email }" HorizontalTextAlignment="Start"
FontSize="14" Placeholder="E-mail" HeightRequest="50" TextColor="Black" PlaceholderColor="#868787"
Margin="15,-30,0,-30"></Entry>
            </Frame>
            <!--senha do funcionario-->
            <Frame CornerRadius="30" BackgroundColor="#f2f2f2" HasShadow="False"
WidthRequest="310" HeightRequest="15" Margin="0,0,0,10">
                <Entry x:Name="cadastroSenha" Text="{Binding Senha }" HorizontalTextAlignment="Start"
FontSize="14" IsPassword="True" Placeholder="Senha" TextColor="Black" PlaceholderColor="#868787"
Margin="15,-30,0,-30"></Entry>
            </Frame>
            <!--confirmar senha do funcionario-->
            <Frame CornerRadius="30" BackgroundColor="#f2f2f2" HasShadow="False"
WidthRequest="310" HeightRequest="15" Margin="0,0,0,10">
                <Entry x:Name="cadastroConfSenha" Text="{Binding confSenha }"
HorizontalTextAlignment="Start" FontSize="14" IsPassword="True" Placeholder="Confirmar Senha"
TextColor="Black" PlaceholderColor="#868787" Margin="15,-30,0,-30"></Entry>
            </Frame>

        </StackLayout>
```

```
        <!--botão para salvar as informações no banco de dados-->
        <StackLayout Grid.Row="2" Margin="0,50,0,0"  >
            <Button x:Name="btnCadastrar" Text="Cadastrar" BackgroundColor="#2595ef"
TextColor="white" WidthRequest="345" HeightRequest="55" HorizontalOptions="Center"
CornerRadius="30" Margin="0,0,0,0" Command="{Binding NovoUsuario}"></Button>
        </StackLayout>
      </Grid>
   </StackLayout>
                                                    </ContentPage>
```

Arquivo – CadastroFuncionario.xaml.cs

```
using polygonalGeofencing.ViewModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace polygonalGeofencing.Views
{
   [XamlCompilation(XamlCompilationOptions.Compile)]
   public partial class CadastroFuncionario : ContentPage
   {
      public CadastroFuncionario()
      {
         InitializeComponent();
         BindingContext = new CadastroUsuarioViewModel();
      }



      private void TapGestureRecognizer_Tapped(object sender, EventArgs e)
      {

      }
   }
                                                        }
```

Arquivo – cadastroModal.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
          xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
          x:Class="polygonalGeofencing.Views.cadastroModal"
          Shell.PresentationMode="ModalAnimated">
```

```
    <!--Grid para organizar os elementos-->
    <Grid RowDefinitions="Auto,Auto">
        <StackLayout Grid.Row="0" HorizontalOptions="Center">
            <Label Text="Cadastrar" FontSize="40" HorizontalTextAlignment="Center" FontFamily="Metropolis"
TextColor="#4A484D" Margin="0,50,0,0"></Label>
            <Label Text="Insira suas informações para cadastrar" HorizontalTextAlignment="Center"
FontSize="15" FontFamily="Metropolis" TextColor="#7C7D7E" Margin="0,0,0,0"></Label>
            <!--Entrys com capitalize word, ou seja a primeira letra sempre vai ser maiuscula-->
            <Frame CornerRadius="30" BackgroundColor="#f2f2f2" HasShadow="False" WidthRequest="310"
HeightRequest="15" Margin="0,50,0,10">
                <Entry x:Name="cadastroNome" Text="{Binding Nome }"  HorizontalTextAlignment="Start"
FontSize="14" Placeholder="Nome" HeightRequest="50" TextColor="Black" PlaceholderColor="#868787"
Margin="15,-30,0,-30">
                    <Entry.Keyboard>
                        <Keyboard x:FactoryMethod="Create">
                            <x:Arguments>
                                <KeyboardFlags>CapitalizeWord</KeyboardFlags>
                            </x:Arguments>
                        </Keyboard>
                    </Entry.Keyboard>
                </Entry>
            </Frame>
            <!--Frame campo sobrenome-->
            <Frame CornerRadius="30" BackgroundColor="#f2f2f2" HasShadow="False" WidthRequest="310"
HeightRequest="15" Margin="0,0,0,10">
                <Entry x:Name="cadastroSobranome"  Text="{Binding Sobrenome }"
HorizontalTextAlignment="Start" FontSize="14" Placeholder="Sobrenome" HeightRequest="50"
TextColor="Black" PlaceholderColor="#868787" Margin="15,-30,0,-30">
                    <Entry.Keyboard>
                        <Keyboard x:FactoryMethod="Create">
                            <x:Arguments>
                                <KeyboardFlags>CapitalizeWord</KeyboardFlags>
                            </x:Arguments>
                        </Keyboard>
                    </Entry.Keyboard>
                </Entry>
            </Frame>
            <!--frane campo e-mail-->
            <Frame CornerRadius="30" BackgroundColor="#f2f2f2" HasShadow="False" WidthRequest="310"
HeightRequest="15" Margin="0,0,0,10">
                <Entry x:Name="cadastroEmail" Text="{Binding Email }" HorizontalTextAlignment="Start"
FontSize="14" Placeholder="E-mail" HeightRequest="50" TextColor="Black" PlaceholderColor="#868787"
Margin="15,-30,0,-30"></Entry>
            </Frame>
            <!--Frame campo senha-->
            <Frame CornerRadius="30" BackgroundColor="#f2f2f2" HasShadow="False" WidthRequest="310"
HeightRequest="15" Margin="0,0,0,10">
                <Entry x:Name="cadastroSenha" Text="{Binding Senha }" HorizontalTextAlignment="Start"
FontSize="14" IsPassword="True" Placeholder="Senha" TextColor="Black" PlaceholderColor="#868787"
Margin="15,-30,0,-30"></Entry>
            </Frame>
            <!--Frame campo confirmação de senha-->
            <Frame CornerRadius="30" BackgroundColor="#f2f2f2" HasShadow="False" WidthRequest="310"
HeightRequest="15" Margin="0,0,0,10">
```

```
            <Entry x:Name="cadastroConfSenha" Text="{Binding confSenha }"
HorizontalTextAlignment="Start" FontSize="14" IsPassword="True" Placeholder="Confirmar Senha"
TextColor="Black" PlaceholderColor="#868787" Margin="15,-30,0,-30"></Entry>
        </Frame>

      </StackLayout>
      <!--Botões cadastrar e logar-->
      <StackLayout Grid.Row="1" Margin="0,50,0,0"  >
        <Button x:Name="btnCadastrar" Text="Cadastrar" BackgroundColor="#2595ef" TextColor="white"
WidthRequest="345" HeightRequest="55" HorizontalOptions="Center" CornerRadius="30"
Margin="0,0,0,0" Command="{Binding NovoUsuario}"></Button>


        <StackLayout HorizontalOptions="Center" Grid.Column="1" Orientation="Horizontal"
Spacing="10">
          <Label Text="Já possui uma conta?" FontSize="14" FontFamily="Cabin"
TextColor="#4A484D"></Label>
          <Button x:Name="btnEntrar" Text="Entrar" FontSize="13" FontFamily="Cabin"
TextColor="#2595ef" BackgroundColor="Transparent" Margin="-13,-13,0,0" Clicked="btnEntrar_Clicked"
></Button>
        </StackLayout>


      </StackLayout>
    </Grid>
                                                    </ContentPage>
```

Arquivo – cadastroModal.xaml.cs

```
using polygonalGeofencing.ViewModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace polygonalGeofencing.Views
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class cadastroModal : ContentPage
    {
        public cadastroModal()
        {
            InitializeComponent();
            BindingContext = new LoginUsuarioViewModel();
        }

        private async void btnEntrar_Clicked(object sender, EventArgs e)
        {
```

```
        await Shell.Current.GoToAsync("//Login/entrarModal");
    }
}
                                                                }
```

Arquivo – cadastroPontos.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        xmlns:maps="clr-namespace:Xamarin.Forms.Maps;assembly=Xamarin.Forms.Maps"
        x:Class="polygonalGeofencing.Views.CadastroPontos"
        Shell.NavBarIsVisible="False"
        x:Name="PaginaPontos">

   <ContentPage.Content>
      <StackLayout>
         <StackLayout  VerticalOptions="Center" HorizontalOptions="Start" WidthRequest="400"
HeightRequest="60" Grid.Row="0"  >
            <Frame BackgroundColor="Transparent" HeightRequest="60">
               <Frame.GestureRecognizers>
                  <TapGestureRecognizer Tapped="TapGestureRecognizer_Tapped" Command="{Binding
Voltar}"/>
               </Frame.GestureRecognizers>
            </Frame>
            <StackLayout Margin="13,-30,0,15">

               <FlexLayout >

                  <ImageButton Source="voltar" WidthRequest="15" HeightRequest="20"
Command="{Binding Voltar}" BackgroundColor="Transparent"></ImageButton>
                  <Label Text="Voltar" TextColor="Black" Margin="3,1,0,0" ></Label>
                  <!--Nome da área-->
                  <Label Text="{Binding nomeArea}" Margin="45,-5,0,0" x:Name="txtArea"  FontSize="Large"
TextColor="Black"/>
                  <!--id da área em um campo oculto-->
                  <Label Text="{Binding idArea}" IsVisible="false" x:Name="idarea"></Label>

               </FlexLayout>
            </StackLayout>
         </StackLayout>
         <StackLayout HorizontalOptions="Center" Orientation="Horizontal">
            <!--Campo para receber o endereço da região-->
            <Frame CornerRadius="30" BackgroundColor="#f2f2f2" HasShadow="False"
WidthRequest="310" HeightRequest="15" Margin="0,0,0,10">
               <Entry x:Name="enderecoMapa" HorizontalTextAlignment="Start" FontSize="14"
Placeholder="Endereço" TextColor="Black" PlaceholderColor="#868787" Margin="15,-30,0,-30"></Entry>
            </Frame>
            <!--botão para realizar a busca pela região-->
            <ImageButton x:Name="iconLupa" Source="lupa" BackgroundColor="Transparent"
WidthRequest="20" Clicked="iconLupa_Clicked"></ImageButton>
         </StackLayout>
         <!--mapa-->
         <maps:Map IsShowingUser="True" x:Name="map" MapClicked="map_MapClicked" />
      </StackLayout>
   </ContentPage.Content>


</ContentPage>
```

Arquivo – cadastroPontos.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using polygonalGeofencing.ViewModels;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
using polygonalGeofencing.Models;
using Xamarin.Forms.Maps;
using Xamarin.Essentials;

namespace polygonalGeofencing.Views
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class CadastroPontos : ContentPage
    {
        public CadastroPontos()
        {
            InitializeComponent();
            BindingContext = new CadastroPontosViewModel();
        }

        protected override async void OnAppearing() //ao inicializar a pagina, quando ela for exibida, carregar
os pontos cadastrados
        {
            var vm = (CadastroPontosViewModel)BindingContext;


            //*****************Centralizar no Usuário*********************************

            var request = new GeolocationRequest(GeolocationAccuracy.Best, TimeSpan.FromSeconds(10));

            var location = await Geolocation.GetLocationAsync(request);

            Position pUsuario = new Position(location.Latitude, location.Longitude);

            MapSpan MapSpan = MapSpan.FromCenterAndRadius(pUsuario, Distance.FromKilometers(.444));
            map.MoveToRegion(MapSpan);
            //**********************************************************************

            //pega os pontos cadastrados no banco
            List<Pontos> model = await App.database.GetPontosArea(Convert.ToInt32(idarea.Text));

            if (model.Count > 0) //se retornar algum registro do banco
            {
                Polygon polygon1 = new Polygon //cria um novo poligono
                {
                    //estilo do poligono, borda e preenchimento
                    StrokeWidth = 8,
                    StrokeColor = Color.FromHex("#00BFFF"),
                    FillColor = Color.FromHex("#87CEFA")
```

```
            };

            foreach (Pontos element in model.ToList())
            {
                //para cada ponto cadastrado no banco ele e adicionado no objeto poligono
                Position p = new Position((double)element.latitude, (double)element.longitude);
                polygon1.Geopath.Add(p);
            };
            //adiciona o objeto poligono no mapa
            map.MapElements.Add(polygon1);
        }
        else
        {
            //se não tiver registro no banco os elementos são removidos do mapa
            map.MapElements.Clear();

        }


    }

    private async void  map_MapClicked(object sender, Xamarin.Forms.Maps.MapClickedEventArgs e)
    {

        var vm = (CadastroPontosViewModel)BindingContext;
        //monta uma string contendo a latitude e longitude
        string posicao = Convert.ToString(e.Position.Latitude.ToString().Replace(",",".") + "," +
e.Position.Longitude.ToString().Replace(",", "."));
        //chama a função passando a string contendo a localização como parametro
        vm.cadastrarPonto.Execute(posicao);
        //pega no banco todos os pontos cadastrados naquela área
        List<Pontos> model = await App.database.GetPontosArea(Convert.ToInt32(idarea.Text));
        //cria um novo objeto polígono
        Polygon polygon1 = new Polygon
        {
            StrokeWidth = 8,
            StrokeColor = Color.FromHex("#00BFFF"),
            FillColor = Color.FromHex("#87CEFA")
        };
        //para cada ponto cadastrado
        foreach (Pontos element in model.ToList())
        {
            //adiciona o ponto cadastrado no polígono
            Position p = new Position((double)element.latitude, (double)element.longitude);
            polygon1.Geopath.Add(p);
        };
        //adiciona o polígono no mapa
        map.MapElements.Add(polygon1);
    }

    private void TapGestureRecognizer_Tapped(object sender, EventArgs e)
    {

    }
```

```
        private async void iconLupa_Clicked(object sender, EventArgs e)
        {
            //recebe o endereço fornecido pelo usuárop
            string endereco = enderecoMapa.Text.Trim();

            Geocoder geoCoder = new Geocoder();

            //pega a latitude e longitude da localização procurada
            IEnumerable<Position> approximateLocations = await
geoCoder.GetPositionsForAddressAsync(endereco);
            Position position = approximateLocations.FirstOrDefault();

            //pega a posição
            Position pEndereco = new Position(position.Latitude, position.Longitude);

            //move para a localização inserida pelo usuário
            MapSpan MapSpan = MapSpan.FromCenterAndRadius(pEndereco, Distance.FromKilometers(.444));
            map.MoveToRegion(MapSpan);


        }
    }
                                                    }
```

Arquivo – EditAreaModal.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             Shell.NavBarIsVisible="False"
             x:Class= "polygonalGeofencing.Views.EditAreaModal">

    <Grid RowDefinitions="Auto,Auto,Auto">
        <!--botão voltar-->
        <StackLayout  VerticalOptions="Center" HorizontalOptions="Start" WidthRequest="400"
HeightRequest="60" Grid.Row="0" >
            <Frame BackgroundColor="Transparent">
                <Frame.GestureRecognizers>
                    <TapGestureRecognizer Tapped="TapGestureRecognizer_Tapped" Command="{Binding
Voltar}"/>
                </Frame.GestureRecognizers>
            </Frame>
            <FlexLayout Margin="13,-30,0,0">
                <ImageButton Source="voltar" WidthRequest="15" HeightRequest="20" Command="{Binding
Voltar}" BackgroundColor="Transparent"></ImageButton>
                <Label Text="Voltar" TextColor="Black" Margin="3,1,0,0" ></Label>
            </FlexLayout>
        </StackLayout>
        <!--Grid para organizar os elementos-->
        <StackLayout Grid.Row="1" HorizontalOptions="Center" Margin="0,-40,0,0">
          <!--Labels para exibição de textos-->
            <Label Text="Editar Area" FontSize="40"  HorizontalTextAlignment="Center"
FontFamily="Metropolis" TextColor="#4A484D" Margin="0,50,0,0"></Label>
```

```
        <Label Text="{Binding mostrarNome}" FontSize="20"  HorizontalTextAlignment="Center"
FontFamily="Metropolis" TextColor="#4A484D" ></Label>
        <Label Text="{Binding idArea}" IsVisible="false"></Label>
        <!--Campo para exibir a descrição da área-->
        <Frame CornerRadius="30" BackgroundColor="#f2f2f2" HasShadow="False" WidthRequest="310"
HeightRequest="240" Margin="0,10,0,10">
            <Editor x:Name="cadastroDesc" Text="{Binding descArea}" AutoSize="TextChanges"
FontSize="14" Placeholder="Descrição" HeightRequest="280" TextColor="Black"
PlaceholderColor="#868787" Margin="15,0,0,-30"></Editor>
        </Frame>
      </StackLayout>
      <!--Botão para salvar as alterações-->
      <StackLayout Grid.Row="2" Margin="0,20,0,0"  >
        <Button x:Name="EditArea" Text="Editar" BackgroundColor="#2595ef" TextColor="white"
WidthRequest="345" HeightRequest="55" HorizontalOptions="Center" CornerRadius="30"
Margin="0,0,0,0" Command="{Binding editarArea}"></Button>
      </StackLayout>
    </Grid>
                                                </ContentPage>
```

Arquivo – EditAreaModal.xaml.cs

```
using polygonalGeofencing.ViewModels;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace polygonalGeofencing.Views
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class EditAreaModal : INotifyPropertyChanged
    {
        public EditAreaModal()
        {
            InitializeComponent();
            BindingContext = new EditarAreaViewModel();
        }


        protected override void OnAppearing() //ao inicializar a pagina, quando ela for exibida
        {
            var vm = (EditarAreaViewModel)BindingContext;
        }

        private void TapGestureRecognizer_Tapped(object sender, EventArgs e)
        {
```

```
    }
}
                                                    }
```

Arquivo – entrarModal.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="polygonalGeofencing.Views.entrarModal"
             Shell.PresentationMode="ModalAnimated">


    <Grid RowDefinitions="*"> <!--Grid-->
        <StackLayout HorizontalOptions="Center">
            <Label Text="Entrar" FontSize="40" HorizontalTextAlignment="Center" FontFamily="Metropolis"
TextColor="#4A484D" Margin="0,50,0,0"></Label>
            <Label Text="Insira suas informações para entrar" HorizontalTextAlignment="Center" FontSize="15"
FontFamily="Metropolis" TextColor="#7C7D7E" Margin="0,0,0,0"></Label>

            <!--Campos para o usuário preencher-->
            <Frame CornerRadius="30" BackgroundColor="#f2f2f2" HasShadow="False" WidthRequest="310"
HeightRequest="15" Margin="0,50,0,10">
                <Entry x:Name="EntrarNome" Text="{Binding Email }" HorizontalTextAlignment="Start"
FontSize="14" Placeholder="E-mail" HeightRequest="50" TextColor="Black" PlaceholderColor="#868787"
Margin="15,-30,0,-30"></Entry>
            </Frame>
            <Frame CornerRadius="30" BackgroundColor="#f2f2f2" HasShadow="False" WidthRequest="310"
HeightRequest="15" Margin="0,0,0,10">
                <Entry x:Name="EntrarSenha" Text="{Binding Senha }" HorizontalTextAlignment="Start"
FontSize="14" IsPassword="True" Placeholder="Senha" TextColor="Black" PlaceholderColor="#868787"
Margin="15,-30,0,-30"></Entry>
            </Frame>

        </StackLayout>
        <StackLayout Grid.Row="1" Margin="0,30,0,0" >

            <!--Botões de entrar e de se cadastrar-->
            <Button Text="Entrar" x:Name="btnEntrar" BackgroundColor="#2595ef" TextColor="white"
WidthRequest="345" HeightRequest="55" HorizontalOptions="Center" CornerRadius="30"
Margin="0,0,0,0" Command="{Binding Entrar}"></Button>

            <StackLayout HorizontalOptions="Center" Grid.Column="1" Orientation="Horizontal"
Spacing="10">
                <Label Text="Não possui uma conta?" FontSize="14" FontFamily="Cabin"
TextColor="#4A484D"></Label>
                <Button x:Name="btnCadastrar" Text="cadastre-se" FontSize="13" FontFamily="Cabin"
TextColor="#2595ef" BackgroundColor="Transparent" Margin="-13,-13,0,0"
Clicked="btnCadastrar_Clicked"></Button>

            </StackLayout>


        </StackLayout>
    </Grid>
                                                </ContentPage>
```

Arquivo – entrarModal.xaml.cs

```
using polygonalGeofencing.ViewModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace polygonalGeofencing.Views
{
   [XamlCompilation(XamlCompilationOptions.Compile)]
   public partial class entrarModal : ContentPage
   {
      public entrarModal()
      {
         InitializeComponent();
         BindingContext = new LoginUsuarioViewModel();
      }
      private async void btnCadastrar_Clicked(object sender, EventArgs e)
      {
         await Shell.Current.GoToAsync("//Login/cadastroModal");
      }
   }
                                                                  }
```

Arquivo – Home.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
   <ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
            xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
            xmlns:maps="clr-namespace:Xamarin.Forms.Maps;assembly=Xamarin.Forms.Maps"
            xmlns:xct="http://xamarin.com/schemas/2020/toolkit"
            Shell.NavBarIsVisible="False"
            x:Class="polygonalGeofencing.Views.Home">



   <!--Grid encapsulando todo o conteudo da página-->
   <Grid RowDefinitions="*" BackgroundColor="#EFEFEF">
      <RelativeLayout>
         <!--Imagem do topo da página-->
         <Image Source="bkgHomePage" Aspect="AspectFill" Margin="0,-137,0,0"></Image>

         <FlexLayout RelativeLayout.XConstraint="30"
                     RelativeLayout.YConstraint="60"
                      >
            <!--Moldura da imagem-->
            <Frame Margin="10"
```

```
                    BackgroundColor="#F1F1F1"
                    CornerRadius="50"
                    HeightRequest="70"
                    WidthRequest="30"
                   >
                <!--Imagem representantativa do usuário-->
                <Image Source="homem"
                    Aspect="AspectFill"
                    Margin="-11"
                    />
             </Frame>

            <!--Frase de boas vindas para o usuário-->
            <StackLayout>
                <Label
                    Text="Bem Vindo(a)!"
                   TextColor="White"
                   FontSize="17"
                   Margin="0,15,0,0"
                  ></Label>
                <Label  x:Name="Usuario"
                   TextColor="White"
                   FontSize="17"
                   Margin="0,-10,0,0"
                  ></Label>
            </StackLayout>

        </FlexLayout>

        <!--Botão para sair da conta logada-->
        <FlexLayout RelativeLayout.XConstraint="300"
                RelativeLayout.YConstraint="15">
            <Frame BackgroundColor="Transparent">
                <Frame.GestureRecognizers>
                    <TapGestureRecognizer Tapped="TapGestureRecognizer_Tapped" Command="{Binding
Sair}"/>
                </Frame.GestureRecognizers>
            </Frame>
            <Label Text="Sair" TextColor="White" Margin="-10,0,10,0"></Label>
            <ImageButton Source="sairBranco" WidthRequest="20" HeightRequest="20"
Command="{Binding Sair}" BackgroundColor="Transparent">
            </ImageButton>
        </FlexLayout>
    </RelativeLayout>

    <Frame Grid.Row="1"
        Margin="0,-190,0,0"
            BackgroundColor="#fff"
            WidthRequest="270"
            HeightRequest="130"
            HorizontalOptions="CenterAndExpand"
            CornerRadius="50"
            VerticalOptions="Start"
           HasShadow="True">
        <!--grid para organizar as informações de visão geral-->
```

```
        <Grid>
            <Grid.ColumnDefinitions >
                <ColumnDefinition Width="Auto"></ColumnDefinition>
                <ColumnDefinition Width="Auto"></ColumnDefinition>
                <ColumnDefinition Width="Auto"></ColumnDefinition>
            </Grid.ColumnDefinitions>
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto"></RowDefinition>
                <RowDefinition Height="Auto"></RowDefinition>
            </Grid.RowDefinitions>
            <Label Text="Visão Geral" HorizontalTextAlignment="Center"  Grid.Row="0" Grid.Column="1"
FontSize="15" TextColor="Black"></Label>
            <!--quantidade de fazenda-->
            <StackLayout Grid.Column="0" Grid.Row="1" HorizontalOptions="Center"
VerticalOptions="Center">

                <Frame
                    BackgroundColor="Transparent"
                    HeightRequest="30"
                    WidthRequest="30"
                    >
                   <Image Source="fazenda"
                    Aspect="AspectFill"
                    Margin="-9"
                    />
                </Frame>
                <Label Text="Fazenda" TextColor="Black" HorizontalTextAlignment="Center"></Label>
                <Label x:Name="qtdFazenda" TextColor="Black" FontSize="17"
                    HorizontalTextAlignment="Center"></Label>
            </StackLayout>
            <!--quantidade de área-->
            <StackLayout Grid.Column="1" Grid.Row="1" HorizontalOptions="Center"
VerticalOptions="Center">

                <Frame Margin="10"
                    BackgroundColor="Transparent"
                    HeightRequest="40"
                    WidthRequest="40"
                    >

                   <Image Source="iconArea"
                    Aspect="AspectFill"
                    Margin="-19"
                    />
                </Frame>
                <Label Text="Área" TextColor="Black" HorizontalTextAlignment="Center"></Label>
                <Label x:Name="qtdArea" TextColor="Black" FontSize="17"
                    HorizontalTextAlignment="Center"></Label>
            </StackLayout>
            <!--quantidade de funcionarios-->
            <StackLayout Grid.Column="2" Grid.Row="1" HorizontalOptions="Center"
VerticalOptions="Center">
                <Frame
                    BackgroundColor="Transparent"
                    HeightRequest="30"
```

```
                    WidthRequest="30"
                    >

                  <Image Source="funcionarios"
                   Aspect="AspectFill"
                   Margin="-9"
                   />
                </Frame>
                <Label Text="Funcionario" TextColor="Black" HorizontalTextAlignment="Center"></Label>
                <Label x:Name="qtdFuncionario" Text="0" TextColor="Black" FontSize="17"
                     HorizontalTextAlignment="Center"></Label>
              </StackLayout>
            </Grid>
          </Frame>
          <!--grid para organização dos cards-->
          <StackLayout Grid.Row="1" HorizontalOptions="Center" VerticalOptions="Center"
WidthRequest="400" HeightRequest="400" >
            <ScrollView>
            <Grid Padding="10">
              <Grid.ColumnDefinitions>
                <ColumnDefinition Width="*"></ColumnDefinition>
                <ColumnDefinition Width="*"></ColumnDefinition>
              </Grid.ColumnDefinitions>
              <Grid.RowDefinitions>
                <RowDefinition Height="*"></RowDefinition>
                <RowDefinition Height="*"></RowDefinition>
              </Grid.RowDefinitions>

              <!--card para adicionar fazenda-->
                <Frame x:Name="frameFazenda"
                     Margin="10"
                     CornerRadius="20"
                     Grid.Column="0"
                     Grid.Row="0"
                     HeightRequest="100"
                     >
                  <Frame.GestureRecognizers>
                    <TapGestureRecognizer  Tapped="AdicionarFazenda"/>
                  </Frame.GestureRecognizers>
                  <StackLayout VerticalOptions="Center" HorizontalOptions="Center">
                    <ImageButton Source="addFazenda"
                          x:Name="cardFazenda"
                          WidthRequest="55"
                          HeightRequest="55"
                          BackgroundColor="Transparent"

                   ></ImageButton>

                    <Label Text="Adicionar &#10; Fazenda" HorizontalTextAlignment="Center"
TextColor="Black" FontSize="15">

                    </Label>
                  </StackLayout>
                </Frame>
              <!--card para adicionar funcionario-->
```

```
            <Frame x:Name="frameAddFunc"
                   Margin="10"
                   CornerRadius="20"
                   Grid.Column="1"
                   Grid.Row="0">
                <Frame.GestureRecognizers>
                    <TapGestureRecognizer  Tapped="AdicionarFuncionarios"/>
                </Frame.GestureRecognizers>
                <StackLayout VerticalOptions="Center" HorizontalOptions="Center">
                    <Image Source="addFuncionarios"
                   WidthRequest="50"
                       HeightRequest="50"
                  Margin="0,10,0,0"
                   ></Image>

                    <Label Text="Adicionar &#10; Funcionarios" HorizontalTextAlignment="Center"
TextColor="Black" FontSize="15">

                    </Label>
                </StackLayout>
            </Frame>
            <Frame x:Name="frameListaFunc"
                   Margin="10"
                   CornerRadius="20"
                   Grid.Column="0"
                   Grid.Row="1">
            <Frame.GestureRecognizers>
                <TapGestureRecognizer  Tapped="TapGestureRecognizer_Tapped_1"/>
            </Frame.GestureRecognizers>
            <StackLayout VerticalOptions="Center" HorizontalOptions="Center">
                <ImageButton x:Name="ListaFunc"
                             Source="listaFunc"
                             BackgroundColor="Transparent"
                             WidthRequest="60"
                             HeightRequest="60"
                             Margin="0,10,0,0"
                   ></ImageButton>

                <Label Text="Lista de &#10; Funcionarios" HorizontalTextAlignment="Center"
TextColor="Black" FontSize="15">

                </Label>
            </StackLayout>
        </Frame>
    </Grid>
    </ScrollView>
  </StackLayout>
</Grid>


</ContentPage>
```

Arquivo – Home.xaml.cs

```
using polygonalGeofencing.Models;
using polygonalGeofencing.ViewModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace polygonalGeofencing.Views
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class Home : ContentPage
    {

        public Home()
        {
            InitializeComponent();
            BindingContext = new HomeViewModel();
        }

        protected override void OnAppearing() //ao inicializar a pagina, quando ela for exibida
        {
            ControleAcesso();
        }



        private async void AdicionarFazenda(object sender, EventArgs e)
        {
            await Shell.Current.GoToAsync("//CadastroFazenda");
        }

        private async void AdicionarFuncionarios(object sender, EventArgs e)
        {
            await Shell.Current.GoToAsync("//CadastroFuncionario");
        }

        private void TapGestureRecognizer_Tapped(object sender, EventArgs e)
        {

        }

        private void TapGestureRecognizer_Tapped_2(object sender, EventArgs e)
        {

        }



        private async void TapGestureRecognizer_Tapped_1(object sender, EventArgs e)
        {
            await Shell.Current.GoToAsync("//ListaFuncionario");
```

```
    }

    public async void ControleAcesso()
    {

        //************Inicializa com os cards desabilitados***************
        frameFazenda.IsEnabled = false;
        frameFazenda.HasShadow = false;
        frameFazenda.BackgroundColor = Color.FromHex("e7e7e7");
        frameAddFunc.IsEnabled = false;
        frameAddFunc.HasShadow = false;
        frameAddFunc.BackgroundColor = Color.FromHex("e7e7e7");
        frameListaFunc.IsEnabled = false;
        frameListaFunc.HasShadow = false;
        frameListaFunc.BackgroundColor = Color.FromHex("e7e7e7");


        //*****************************************************************



        List<Logado> logado = await App.database.GetLogado(); //pega o id do usuário logado
        //pega os acessos que o usuário tem
        List<Acesso> Acessos = await App.database.GetAcessoUsuario(logado.ToList()[0].idUsuario,-1);
        //pega as informações do usuário
        Usuario usuario = await App.database.GetByIdUsuario(logado.ToList()[0].idUsuario);







        //verifica se o usuário logado e o dono
        if (usuario.idDono != usuario.idUsuario)
        {
            //coloca o nome completo do usuário na página
            Usuario.Text = usuario.nome + " " + usuario.sobrenome;
            //pega as fazendas que o usuário tem acesso
            List<Fazenda> fazendas =  await App.database.GetFazendaFuncionario(usuario.idUsuario,
usuario.idDono);

            //coloca a quantidade de fazendas no quadro de visão geral
            qtdFazenda.Text = fazendas.Count().ToString();


            string str = "";
            List<Area> areas = null;
            List<Usuario> funcionarios = null;


            //colocar na variavel str todas as fazendas em que o usuário tem acesso
            foreach (Fazenda fazenda in fazendas.ToList())
            {
```

```
            str = fazenda.idFazenda + ",";
        }


        if (str != "")
        {
          areas = await App.database.GetAreasFazenda(str.Substring(0, str.Length - 1)); //Substring para
retirar a ultima virgula
          funcionarios = await App.database.GetFuncionariosFazenda(str.Substring(0, str.Length - 1),
usuario.idUsuario, usuario.idDono); //Substring para retirar a ultima virgula
        }


        qtdArea.Text = "-";  //se o usuário não tiver acesso a nenhuma área
        qtdFuncionario.Text = "-";//se o usuário não tiver acesso a ver funcionario

        foreach (Acesso acesso in Acessos.ToList())
        {
        //Controle dos cards, habilitando os que o usuário tem acesso.
          switch (acesso.idTipo)
          {
            case 1:
              frameFazenda.IsEnabled = true;
              frameFazenda.HasShadow = true;
              frameFazenda.BackgroundColor = Color.FromHex("fff");

              if(areas != null)
              {
                qtdArea.Text = areas.Count().ToString();
              }

              break;
            case 2:
              frameAddFunc.IsEnabled = true;
              frameAddFunc.HasShadow = true;
              frameAddFunc.BackgroundColor = Color.FromHex("fff");
              if (funcionarios != null)
              {
                qtdFuncionario.Text = funcionarios.Count().ToString();
              }
              break;
            case 3:
              frameListaFunc.IsEnabled = true;
              frameListaFunc.HasShadow = true;
              frameListaFunc.BackgroundColor = Color.FromHex("fff");
              if (funcionarios != null)
              {
                qtdFuncionario.Text = funcionarios.Count().ToString();
              }
              break;

          }
        }
      }
      else
```

```
        {
            //se o usuário for o dono habilitar to
            List<Fazenda> fazendas = await App.database.GetFazendaUsuario(usuario.idDono);
            List<Area> areas = await App.database.GetAreasUsuario(usuario.idDono);
            List<Usuario> funcionarios = await App.database.GetFuncionariosFazenda("-1",usuario.idUsuario,
usuario.idDono);

            Usuario.Text = usuario.nome + " " + usuario.sobrenome;
            qtdFazenda.Text = fazendas.Count().ToString();
            qtdArea.Text = areas.Count().ToString();
            qtdFuncionario.Text = funcionarios.Count().ToString();

            frameFazenda.IsEnabled = true;
            frameFazenda.HasShadow = true;
            frameFazenda.BackgroundColor = Color.FromHex("fff");
            frameAddFunc.IsEnabled = true;
            frameAddFunc.HasShadow = true;
            frameAddFunc.BackgroundColor = Color.FromHex("fff");
            frameListaFunc.IsEnabled = true;
            frameListaFunc.HasShadow = true;
            frameListaFunc.BackgroundColor = Color.FromHex("fff");


        }

    }


}
                                                        }
```

Arquivo – ListaFuncionario.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             Shell.NavBarIsVisible="False"
             x:Class="polygonalGeofencing.Views.ListaFuncionario">


    <!--Grid para organizar os elementos-->
    <Grid RowDefinitions="Auto,Auto,Auto">
        <!--Botão para voltar-->
        <StackLayout  VerticalOptions="Center" HorizontalOptions="Start" WidthRequest="400"
HeightRequest="60" Grid.Row="0" >
            <Frame BackgroundColor="Transparent">
                <Frame.GestureRecognizers>
                    <TapGestureRecognizer Tapped="TapGestureRecognizer_Tapped" Command="{Binding
Voltar}"/>
                </Frame.GestureRecognizers>
            </Frame>
            <FlexLayout Margin="13,-30,0,0">
                <ImageButton Source="voltar" WidthRequest="15" HeightRequest="20" Command="{Binding
Voltar}" BackgroundColor="Transparent"></ImageButton>
                <Label Text="Voltar" TextColor="Black" Margin="3,1,0,0" ></Label>
            </FlexLayout>
        </StackLayout>
        <!--Titulo da página-->
        <StackLayout Grid.Row="1" HorizontalOptions="Center" Margin="0,-40,0,0">
            <Label Text="Lista de Funcionarios" FontSize="30"  HorizontalTextAlignment="Center"
FontFamily="Metropolis" TextColor="#4A484D" Margin="0,50,0,0"></Label>
        </StackLayout>
        <!--Lista de funcionários-->
        <ListView ItemsSource="{Binding listaFuncionario}" Grid.Row="2" HasUnevenRows="True"
Margin="0,20,0,0">
            <ListView.ItemTemplate>
                <DataTemplate>
                    <ViewCell>
                        <StackLayout  Padding="30,10" BackgroundColor="#F1F1F1">
                            <Grid RowDefinitions="100" ColumnDefinitions="Auto,*,Auto,Auto,Auto,Auto"
VerticalOptions="Center">
                                <!--guarda o id do funcionário-->
                                <Label Text="{Binding idFuncionario}" IsVisible="false"></Label>
                                <!--imagem simbolisando os funcionários-->
                                <Image Source="funcionarios" Grid.Row="0" Grid.Column="0"
FlowDirection="LeftToRight" HeightRequest="50" WidthRequest="50" ></Image>
                                <!--nome do funcionário-->
                                <Label Text="{Binding nome}" Grid.Row="0" Grid.Column="1" TextColor="Black"
VerticalTextAlignment="Center" HorizontalTextAlignment="Center"></Label>
                                <!--botão para adicionar permissão-->
                                <ImageButton x:Name="btnAddPerm" Source="editar" Margin="10,0,0,0"
WidthRequest="25" HeightRequest="25" BackgroundColor="Transparent" Grid.Row="0" Grid.Column="3"
Clicked="btnAddPerm_Clicked" />
                                <!--botão para excluir o funcionário-->
```

```
                        <ImageButton x:Name="btnExcluir" Source="lixeira" WidthRequest="15"
HeightRequest="15" BackgroundColor="Transparent" Grid.Row="0" Grid.Column="5"
Clicked="btnExcluir_Clicked" />

                    </Grid>
                </StackLayout>

            </ViewCell>
        </DataTemplate>
    </ListView.ItemTemplate>
</ListView>

</Grid>


                                        </ContentPage>
```

Arquivo – ListaFuncionario.xaml.cs

```
using polygonalGeofencing.Models;
using polygonalGeofencing.ViewModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace polygonalGeofencing.Views
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class ListaFuncionario : ContentPage
    {
        public ListaFuncionario()
        {
            InitializeComponent();
            BindingContext = new ListaFuncionarioViewModel();
        }

        protected override void OnAppearing() //ao inicializar a pagina, quando ela for exibida
        {
            var vm = (ListaFuncionarioViewModel)BindingContext;
            //chama a função atualizar lista na viewmodel
            vm.AtualizarLista.Execute(null);

        }

        private void btnAddPerm_Clicked(object sender, EventArgs e)
        {
            var button = sender as ImageButton;
            var area = button?.BindingContext as Usuario;
```

```
        var vm = BindingContext as ListaFuncionarioViewModel;

        vm?.EditarFuncionario.Execute(area);
    }

    private void btnExcluir_Clicked(object sender, EventArgs e)
    {
        var button = sender as ImageButton;
        var area = button?.BindingContext as Usuario;
        var vm = BindingContext as ListaFuncionarioViewModel;

        vm?.RemoverFuncionario.Execute(area);
    }

    private void TapGestureRecognizer_Tapped(object sender, EventArgs e)
    {

    }
}
                                                    }
```

Arquivo – Login.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="polygonalGeofencing.Views.Login"
             NavigationPage.HasNavigationBar="False"
             Shell.NavBarIsVisible="False">



  <Grid RowDefinitions="*"> <!--Grid definindo a linha como * ou sejá irá preencher toda a página-->
    <!--Elemento para organização das informações-->
    <StackLayout >

      <!--Chamando a imagem que irá compor o topo da página-->
      <Image Source="bkgLogin" Aspect="AspectFill" ></Image>


      <!--Definindo a linha 1 do grid-->
      <StackLayout Grid.Row="1">

        <!--Chamando a Logo do aplicativo-->
        <Image Source="Logo" WidthRequest="90" Margin="13,-65,0,0"></Image>

        <!-- Elemento para organizar o texto da pagina inicial -->
        <StackLayout HorizontalOptions="Center" Grid.Column="1" Orientation="Horizontal"
Spacing="10">

          <Label Text="Farm" FontSize="34" FontFamily="Cabin" TextColor="#2595ef"></Label>
          <Label Text="Mapping" FontSize="34" FontFamily="Cabin" TextColor="#4A484D"></Label>
        </StackLayout>
```

```
            <Label Text="Mapeamento de Área" FontSize="11" CharacterSpacing="2" WidthRequest="130"
HorizontalOptions="Center" FontFamily="Metropolis"></Label>
            <Label Text="Descubra o melhor aplicativo de mapeamento de áreas para monitoramento."
FontSize="12" CharacterSpacing="2" HorizontalTextAlignment="Center" Margin="20,10,20,20"
HorizontalOptions="Center" FontFamily="Metropolis"></Label>

            <!-- Botões para realizar o login ou o cadastro -->
            <Button Text="Entrar" x:Name="btnEntrar" BackgroundColor="#2595ef" TextColor="white"
WidthRequest="310" HeightRequest="55" HorizontalOptions="Center" CornerRadius="30"
Margin="0,0,0,10" Clicked="btnEntrar_Clicked"></Button>
            <Button Text="Cadastrar" x:Name="btnCadastrar" BorderColor="#2595ef" BorderWidth="1"
BackgroundColor="Transparent" TextColor="#2595ef" WidthRequest="310" HeightRequest="55"
HorizontalOptions="Center" CornerRadius="30" Clicked="btnCadastrar_Clicked"></Button>
        </StackLayout>
    </StackLayout>
</Grid>

                                                    </ContentPage>
```

Arquivo – Login.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using polygonalGeofencing.Models;
using polygonalGeofencing.ViewModels;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace polygonalGeofencing.Views
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class Login : ContentPage
    {
        public Login()
        {
            InitializeComponent();

        }

        protected override async void OnAppearing() //ao inicializar a pagina, quando ela for exibida
        {


            List<Logado> logado = await App.database.GetLogado(); //pega as informações da tabela logado

            if (logado.Count > 0) //verifica se há registro
            {
                //verifica a diferença entre a data atual e a data do banco
                int diasLogado = (int)DateTime.Today.Subtract(logado.ToList()[0].data).TotalDays;
```

```
            /*Verifica no banco se este usuário ja esta logado a mais de 2 dias, se estiver remover o registro
dele,
            para que ele possa logar de novo*/
            if (diasLogado >= 2)
            {
               await App.database.DelLogado(); //deletrar as informações da tabela logado
            }
            else
            {
               //se a diferença for de menos de 2 dias redirecionar para a home page
               await Shell.Current.GoToAsync("//Home");
            }

         }

      }

      private async void btnEntrar_Clicked(object sender, EventArgs e)
      {
         //redireciona para a tela entrarModal, chando-o dessa forma ira abrir como modal
         await Shell.Current.GoToAsync("//Login/entrarModal");
      }

      private async  void btnCadastrar_Clicked(object sender, EventArgs e)
      {
         await Shell.Current.GoToAsync("//Login/cadastroModal");
      }
   }
                                                      }
```

Arquivo – Mapa.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:maps="clr-namespace:Xamarin.Forms.Maps;assembly=Xamarin.Forms.Maps"
             Shell.NavBarIsVisible="False"
             x:Class="polygonalGeofencing.Views.Mapa">
   <ContentPage.Content>
      <StackLayout>
         <StackLayout HorizontalOptions="Center" Orientation="Horizontal" Margin="0,30,0,0">
            <!--Campo para receber o endereço da região-->
            <Frame CornerRadius="30" BackgroundColor="#f2f2f2" HasShadow="False"
WidthRequest="310" HeightRequest="15" Margin="0,0,0,10">
               <Entry x:Name="enderecoMapa" HorizontalTextAlignment="Start" FontSize="14"
Placeholder="Endereço" TextColor="Black" PlaceholderColor="#868787" Margin="15,-30,0,-30"></Entry>
            </Frame>
            <!--botão para realizar a busca pela região-->
            <ImageButton x:Name="iconLupa" Source="lupa" BackgroundColor="Transparent"
WidthRequest="20" Clicked="iconLupa_Clicked"></ImageButton>
         </StackLayout>
         <!--Mapa da aplicação-->
         <maps:Map IsShowingUser="True"  x:Name="map" PropertyChanged="map_PropertyChanged"/>
```

```
    </StackLayout>
</ContentPage.Content>
                                        </ContentPage>
```

Arquivo – Mapa.xaml.cs

```
using polygonalGeofencing.Models;
using Plugin.LocalNotification;
using Plugin.LocalNotification.EventArgs;
//using Plugin.LocalNotification;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Essentials;
using Xamarin.Forms;
using Xamarin.Forms.Maps;
using Xamarin.Forms.Xaml;


namespace polygonalGeofencing.Views
{

    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class Mapa : ContentPage
    {
        int idLogado = 0;
        string notify = "false|-1";
        bool changedMap = false;
        double latitude = 0;
        double longitude = 0;

        public Mapa()
        {
            InitializeComponent();
            //LocalNotificationCenter.Current.NotificationActionTapped += OnNotificationActionTapped;


        }


        //private void OnNotificationActionTapped(NotificationEventArgs e)
        //{
        //    DisplayAlert(e.Request.Title, e.Request.Description,"OK");
        //}


        protected override async void OnAppearing() //ao inicializar a pagina, quando ela for exibida
        {
            //limpa todos elementos do mapa
            map.MapElements.Clear();

            //****************Centralizar no Usuário*********************************

            var request = new GeolocationRequest(GeolocationAccuracy.Best, TimeSpan.FromSeconds(10));
```

```
        var location = await Geolocation.GetLocationAsync(request);

        Position pUsuario = new Position(location.Latitude, location.Longitude);

        MapSpan MapSpan = MapSpan.FromCenterAndRadius(pUsuario, Distance.FromKilometers(.444));

        map.MoveToRegion(MapSpan);

        //*********************************************************************

        //pega o usuário logado
        List<Logado> logado = await App.database.GetLogado();

        //pega as informações do usuário
        Usuario usuario = await App.database.GetByIdUsuario(logado.ToList()[0].idUsuario);

        //instancia a variavel que irá receber as areas
        List<Area> quantArea = null;

        //verifica se o usuário e o dono
        if (usuario.idDono != usuario.idUsuario)
        {
            //pega as fazendas que o usuário tem acesso
            List<Fazenda> fazendas = await App.database.GetFazendaFuncionario(usuario.idUsuario,
usuario.idDono);

            string str = "";

            //colocar na variavel str todas as fazendas em que o usuário tem acesso
            foreach (Fazenda fazenda in fazendas.ToList())
            {
                str = fazenda.idFazenda + ",";
            }

            if (str != "")
            {
                //pega as áreas
                quantArea = await App.database.GetAreasFazenda(str.Substring(0, str.Length - 1)); //Substring
para retirar a ultima virgula

            }
        }
        else
        {
            //pega todas as áreas
            quantArea = await App.database.GetAreasUsuario(usuario.idDono);

        }


        //se retornou algo do banco
        if (quantArea.ToList().Count > 0)
        {
            //chama a função disparaMensagem
```

```
        disparaMensagem();
        //para cada área será criado um polígono
        foreach (Area quant in quantArea.ToList())
        {
            List<Pontos> model = await App.database.GetPontosArea(quant.idArea);
            if(model.ToList().Count > 0)
            {
                Polygon polygon1 = new Polygon
                {
                    StrokeWidth = 8,
                    StrokeColor = Color.FromHex("#00BFFF"),
                    FillColor = Color.FromHex("#87CEFA")
                };

                Pin pin = new Pin
                {
                    Label = quant.nome,
                    Address = quant.descricao,
                    Type = PinType.Place,
                };


                foreach (Pontos element1 in model.ToList())
                {
                    Position p = new Position((double)element1.latitude, (double)element1.longitude);
                    polygon1.Geopath.Add(p);
                    latitude = (double)element1.latitude;
                    longitude = (double)element1.longitude;
                };

                pin.Position = new Position(latitude,longitude);
                map.Pins.Add(pin);

                map.MapElements.Add(polygon1);
            }
        };
    }
}
```

```
public static bool intoPolygon(Point[] poly, Point pointUsuario)
{

    //calculo para ver se o usuário está dentro da área
    var coef = poly.Skip(1).Select((p, i) =>
                                (pointUsuario.Y - poly[i].Y) * (p.X - poly[i].X)
                              - (pointUsuario.X - poly[i].X) * (p.Y - poly[i].Y))
                        .ToList();
```

```
        if (coef.Any(p => p == 0))
            return true;

        for (int i = 1; i < coef.Count(); i++)
        {
            if (coef[i] * coef[i - 1] < 0)
                return false;
        }
        return true;
    }


    public void disparaMensagem()
    {
        //timer para que de tempos em tempos verifcar se o usuário entrou em alguma área
        Device.StartTimer(new TimeSpan(0, 0, 1), () =>
        {
            changedMap = true;
            varrerArea();
            return true;
        });
    }




    public async void varrerArea()
    {
        try
        {
            var request = new GeolocationRequest(GeolocationAccuracy.Best, TimeSpan.FromSeconds(10));

            var location = await Geolocation.GetLocationAsync(request);

            Position pUsuario = new Position(location.Latitude, location.Longitude);

            MapSpan MapSpan = MapSpan.FromCenterAndRadius(pUsuario, Distance.FromKilometers(.444));

            if (changedMap)
            {
                //****************Centralizar no Usuário*********************************
                  map.MoveToRegion(MapSpan);
                //*********************************************************************
            }



            if (notify.Split('|')[0] == "true" && Int32.Parse(notify.Split('|')[1]) > 0) //se o Usuário já esta dentro
de uma área
                {
                //pega todos os pontos para ver se o usuário ainda esta dentro da área
                    List<Pontos> model = await App.database.GetPontosArea(Int32.Parse(notify.Split('|')[1]));
                    Point[] pts = new Point[model.ToList().Count];
                    int i = 0;
```

```
                int idArea = 0;
                foreach (Pontos element1 in model.ToList())
                {

                    pts[i] = new Point { X = (double)element1.latitude, Y = (double)element1.longitude };
                    i++;
                    idArea = element1.idArea;
                };
                Point posUsu = new Point { X = location.Latitude, Y = location.Longitude };

                bool dentro = intoPolygon(pts, posUsu);

            //se o usuário saiu da área notify reseta sua informação
            if (!dentro)
            {
                notify = "false|-1";
            }
        }
            else
            {
            //pega o usuário logado
            List<Logado> logado = await App.database.GetLogado();

            //pega as informações do usuário
            Usuario usuario = await App.database.GetByIdUsuario(logado.ToList()[0].idUsuario);

            //instancia a variavel que irá receber as areas
            List<Area> qtArea = null;

            //verifica se o usuário e o dono
            if (usuario.idDono != usuario.idUsuario)
            {
                //pega as fazendas que o usuário tem acesso
                List<Fazenda> fazendas = await App.database.GetFazendaFuncionario(usuario.idUsuario,
usuario.idDono);

                string str = "";

                //colocar na variavel str todas as fazendas em que o usuário tem acesso
                foreach (Fazenda fazenda in fazendas.ToList())
                {
                    str = fazenda.idFazenda + ",";
                }

                if (str != "")
                {
                    //pega as áreas
                    qtArea = await App.database.GetAreasFazenda(str.Substring(0, str.Length - 1)); //Substring
para retirar a ultima virgula

                }
            }
            else
            {
                //pega todas as áreas
```

```
                qtArea = await App.database.GetAreasUsuario(usuario.idDono);

            }

            if (qtArea.ToList().Count > 0)
                {
                    foreach (Area quant in qtArea.ToList())
                    {
                        List<Pontos> model = await App.database.GetPontosArea(quant.idArea);

                        if (model.Count > 3) //so verificar se esta dentro quando tiver mais de 3 pontos
cadastrados
                        {
                            Point[] pts = new Point[model.ToList().Count];
                            int i = 0;
                            int idArea = 0;
                            foreach (Pontos element1 in model.ToList())
                            {
                                pts[i] = new Point { X = (double)element1.latitude, Y = (double)element1.longitude
};
                                i++;
                                idArea = element1.idArea;
                            };
                            Point posUsu = new Point { X = location.Latitude, Y = location.Longitude };

                            bool dentro = intoPolygon(pts, posUsu);

                            if (dentro)// se o usuário estiver dentro da área e disparado um alerta
                            {

                                notify = "true|" + idArea;
                                Area getArea = await App.database.GetByIdArea(idArea);
                                var notification = new NotificationRequest
                                {
                                    BadgeNumber = 1,
                                    NotificationId = 1337,
                                    Title = "Entrou na Area: " + getArea.nome,
                                    Description = getArea.descricao,
                                    ReturningData = "Dummy data", //Retorna dados quando toca na notificação
                                };
                                await LocalNotificationCenter.Current.Show(notification);
                                await DisplayAlert("Entrou na Area: " + getArea.nome, getArea.descricao, "OK");

                            }
                        }


                    }
                }
            }

         }
        catch (FeatureNotSupportedException fnsEx)
        {
            // Handle not supported on device exception
```

```
            Console.WriteLine(fnsEx.InnerException.Message);
        }
        catch (FeatureNotEnabledException fneEx)
        {
            // Handle not enabled on device exception
            Console.WriteLine(fneEx);
        }
        catch (PermissionException pEx)
        {
            // Handle permission exception
            Console.WriteLine(pEx);
        }
        catch (Exception ex)
        {
            // Unable to get location
            Console.WriteLine(ex);
        }
    }



    private void map_PropertyChanged(object sender,
System.ComponentModel.PropertyChangedEventArgs e)
    {
        changedMap = false;
    }

    private async void iconLupa_Clicked(object sender, EventArgs e)
    {
        //recebe o endereço fornecido pelo usuárop
        string endereco = enderecoMapa.Text.Trim();

        Geocoder geoCoder = new Geocoder();

        //pega a latitude e longitude da localização procurada
        IEnumerable<Position> approximateLocations = await
geoCoder.GetPositionsForAddressAsync(endereco);
        Position position = approximateLocations.FirstOrDefault();

        //pega a posição
        Position pEndereco = new Position(position.Latitude, position.Longitude);

        //move para a localização inserida pelo usuário
        MapSpan MapSpan = MapSpan.FromCenterAndRadius(pEndereco, Distance.FromKilometers(.444));
        map.MoveToRegion(MapSpan);
    }
  }
}
```

Arquivo – Perfil.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             Shell.NavBarIsVisible="False"
             x:Class="polygonalGeofencing.Views.Perfil">
    <ContentPage.Content>


        <!--Grid para organizar os elementos-->
        <Grid RowDefinitions="*,*,*">
            <StackLayout Grid.Row="0" HorizontalOptions="Center" Margin="0,-40,0,0">
                <Label Text="Perfil" FontSize="30"  HorizontalTextAlignment="Center" FontFamily="Metropolis"
TextColor="#4A484D" Margin="0,50,0,0"></Label>
                <RelativeLayout Grid.Row="0">

                    <FlexLayout RelativeLayout.XConstraint="150"
                          RelativeLayout.YConstraint="20"
                           >
                        <!--Moldura da imagem-->
                        <Frame Margin="10"
                          BackgroundColor="#cfcfcf"
                          CornerRadius="50"
                          HeightRequest="70"
                          WidthRequest="30"
                         >
                            <!--Imagem representantativa do usuário-->
                            <Image Source="homem"
                         Aspect="AspectFill"
                         Margin="-11"
                         />
                        </Frame>

                        <!--Frase de boas vindas para o usuário-->
                        <StackLayout>
                            <Label
                       Text="Bem Vindo(a)!"
                      TextColor="White"
                      FontSize="17"
                      Margin="0,15,0,0"
                     ></Label>
                            <Label  x:Name="Usuario"
                      TextColor="White"
                      FontSize="17"
                      Margin="0,-10,0,0"
                     ></Label>
                        </StackLayout>
                    </FlexLayout>
                </RelativeLayout>
            </StackLayout>

            <StackLayout Grid.Row="1" HorizontalOptions="FillAndExpand">
```

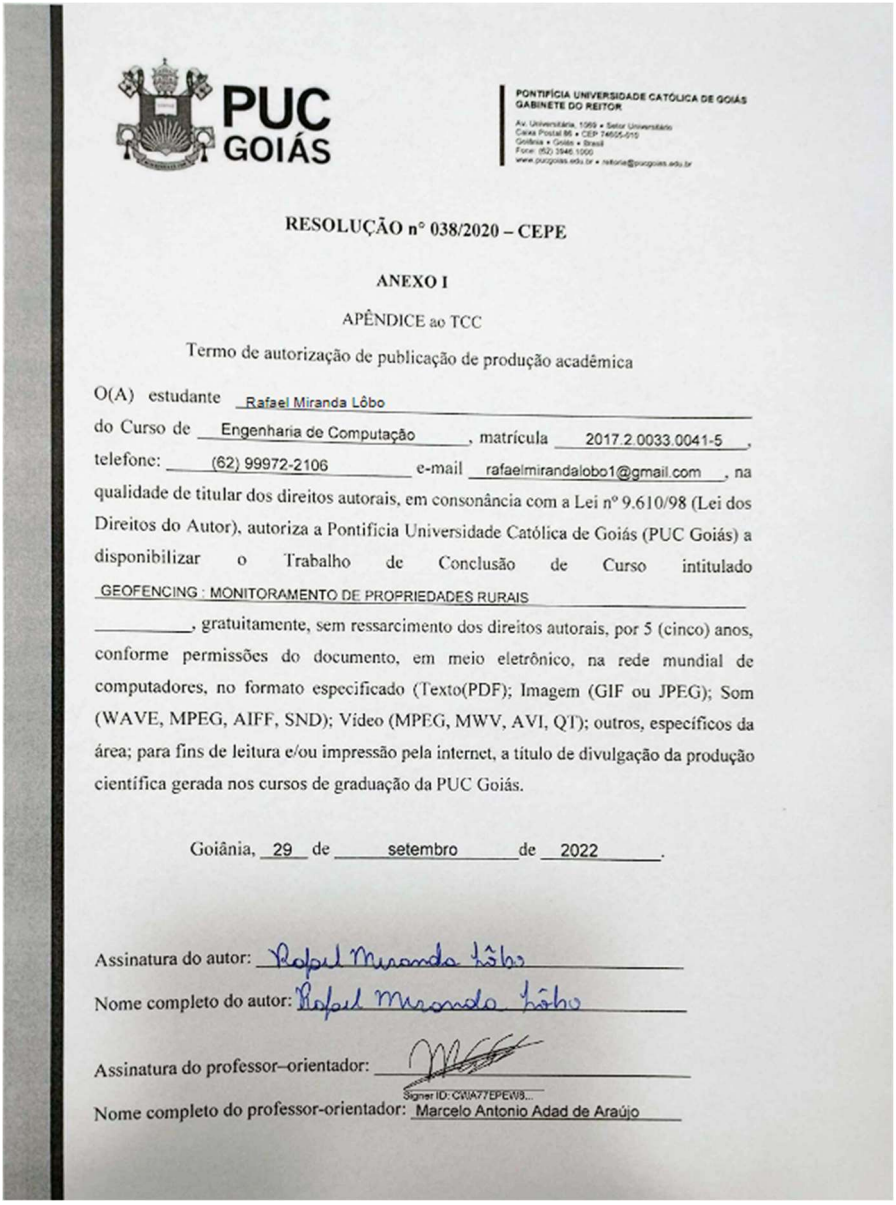
```
            <Label Text="Informação do Usuário" FontSize="19" FontFamily="Metropolis"
TextColor="#5470fe" Margin="20,-50,0,0"></Label>
            <Grid RowDefinitions="*,*" ColumnDefinitions="*,*" Padding="20,10,20,0">
               <!--Campo contendo o primeiro nome do usuário-->
               <StackLayout Grid.Row="0" Grid.Column="0">
                  <Label Text="Primeiro Nome" FontSize="14" TextColor="Gray"></Label>
                  <Frame HasShadow="False" BorderColor="#d9d9d9" CornerRadius="20">
                     <Label Text="{Binding Nome}" FontSize="16" Margin="0,-5,0,0"
TextColor="Black"></Label>
                  </Frame>
               </StackLayout>
               <!--Campo contendo o sobrenome do usuário-->
               <StackLayout Grid.Row="0" Grid.Column="1" Margin="20,0,0,0">
                  <Label Text="Último Nome" FontSize="14" TextColor="Gray"></Label>
                  <Frame HasShadow="False" BorderColor="#d9d9d9" CornerRadius="20" >
                     <Label Text="{Binding Sobrenome}" FontSize="16" Margin="0,-5,0,0"
TextColor="Black"></Label>
                  </Frame>
               </StackLayout>
            </Grid>
            <!--Campo contendo o E-mail do usuário-->
            <StackLayout Margin="0,-70,0,0" Padding="20,0,20,0">
               <Label Text="E-mail" FontSize="14" TextColor="Gray"></Label>
               <Frame HasShadow="False" BorderColor="#d9d9d9" CornerRadius="20">
                  <Label Text="{Binding Email}" FontSize="16" Margin="0,-5,0,0" TextColor="Black"></Label>
               </Frame>
            </StackLayout>

         </StackLayout>
      </Grid>




   </ContentPage.Content>
                                          </ContentPage>
```

Arquivo – Perfil.xaml.cs

```
using polygonalGeofencing.ViewModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace polygonalGeofencing.Views
{
```

```
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class Perfil : ContentPage
    {
        public Perfil()
        {
            InitializeComponent();
            BindingContext = new PerfilViewModel();
        }

        protected override void OnAppearing() //ao inicializar a pagina, quando ela for exibida
        {
            var vm = (PerfilViewModel)BindingContext;
            //chama a função mostrarPerfil da viewmodel
            vm.mostrarPerfil.Execute(null);
        }


        private async void TapGestureRecognizer_Tapped(object sender, EventArgs e)
        {
            await Shell.Current.GoToAsync("//Home");
        }
    }
}
```

ANEXO 1



PUC GOIÁS

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS
GABINETE DO REITOR
Av. Universitária, 1069 • Setor Universitário
Caixa Postal 86 • CEP 74605-010
Goiânia • Goiás • Brasil
Fone: (62) 3946.1000
www.pucgoias.edu.br • reitoria@pucgoias.edu.br

RESOLUÇÃO n° 038/2020 – CEPE

ANEXO I

APÊNDICE ao TCC

Termo de autorização de publicação de produção acadêmica

O(A) estudante Rafael Miranda Lôbo
do Curso de Engenharia de Computação, matrícula 2017.2.0033.0041-5,
telefone: (62) 99972-2106 e-mail rafaelmirandalobo1@gmail.com, na qualidade de titular dos direitos autorais, em consonância com a Lei nº 9.610/98 (Lei dos Direitos do Autor), autoriza a Pontifícia Universidade Católica de Goiás (PUC Goiás) a disponibilizar o Trabalho de Conclusão de Curso intitulado GEOFENCING : MONITORAMENTO DE PROPRIEDADES RURAIS, gratuitamente, sem ressarcimento dos direitos autorais, por 5 (cinco) anos, conforme permissões do documento, em meio eletrônico, na rede mundial de computadores, no formato especificado (Texto(PDF); Imagem (GIF ou JPEG); Som (WAVE, MPEG, AIFF, SND); Vídeo (MPEG, MWV, AVI, QT); outros, específicos da área; para fins de leitura e/ou impressão pela internet, a título de divulgação da produção científica gerada nos cursos de graduação da PUC Goiás.

Goiânia, 29 de setembro de 2022.

Assinatura do autor: Rafael Miranda Lôbo

Nome completo do autor: Rafael Miranda Lôbo

Assinatura do professor–orientador:

Signer ID: CWA77EPEW8...

Nome completo do professor-orientador: Marcelo Antonio Adad de Araújo