

**PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS
ESCOLA POLITÉCNICA
GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO**



EDUARDO MIRANDA DE OLIVEIRA

**VIVER BIKE: UM APLICATIVO MOBILE PARA GERENCIAMENTO
DE CICLISMO EM GRUPO**

GOIÂNIA
2021

EDUARDO MIRANDA DE OLIVEIRA

**VIVER BIKE: UM APLICATIVO MOBILE PARA GERENCIAMENTO
DE CICLISMO EM GRUPO**

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

Orientador: Ms. Fernando Gonçalves Abadia

Goiânia
2021

EDUARDO MIRANDA DE OLIVEIRA

**VIVER BIKE: UM APLICATIVO MOBILE PARA GERENCIAMENTO
DE CICLISMO EM GRUPO**

Trabalho de Conclusão de Curso julgado adequado para obtenção do título de Bacharel em Ciências da Computação e aprovado em sua forma final pela Escola de Ciências Exatas e da Computação, da Pontifícia Universidade Católica de Goiás, em ____/____/____.

Profa. Ma. Ludmilla Reis Pinheiro dos Santos
Coordenador(a) de Trabalho de Conclusão de Curso

Banca examinadora:

Orientador(a): Fernando Gonçalves Abadia

Prof. Me. Max Gontijo de Oliveira

Prof. Me Rafael Leal Martins

GOIÂNIA
2021

RESUMO

O universo do ciclismo é de longe um dos que mais cresce no Brasil e, no meio de tanto desenvolvimento na área, o nascimento de uma aplicação que dê suporte completo às pessoas que estão querendo organizar seus pedais traz mais interação ao grupo. Este projeto estuda a viabilização e cria um modelo de aplicação que se empenha em dar total suporte aos usuários que querem ter mais conexões no meio do esporte. Durante o projeto, foram implementadas telas de autenticação de usuário com Entrar e Se inscrever, além da tela principal do usuário, onde pode ser visualizada uma lista de salas com nome, tamanho da sala e localização do pedal e telas de interação dentro das salas com menu, mapa, chat e apoio. O principal desafio do trabalho foi o levantamento de requisitos os quais atendessem às reais necessidades dos usuários e, para sua solução, foram feitos estudos com ciclistas veteranos e pessoas que desejavam iniciar no esporte. O projeto ainda é um protótipo e será apresentado apenas o módulo principal da aplicação no trabalho, mas haverá outras funcionalidades na versão final para fins de implantação e liberação do aplicativo nas lojas.

Palavra-Chave: Ciclismo, Suporte Completo, Modelo de aplicação, Esporte, Salas, Estudos com ciclistas e Protótipo.

ABSTRACT

The world of cycling is one of the fastest growing sports in Brazil. The beginning of an application, within this development area, that fully supports people who are wanting to organize their pedals brings more interaction to the group. This project studies the feasibility and creates an application model that strives to give full support to users who wants to have more connections in the sport. The user authentication screen was implemented during the project with Login and Subscribe buttons, and it was also implemented the main user screen, where it is possible to view a list of rooms with name, room size and pedal location. In addition, it was made the interaction screens that corresponds to a room with menu, map, chat, and user support. The main challenge of this work was the requirements gathering, which is important for user's needs. Studies with veteran cyclists and people who wanted start in this sport were carried out for these solutions. The project is a prototype and only the main module of the application will be presented at this work. There will be other functionalities in the final version of this app, for the purposes of deployment and release in stores.

Keywords: Cycling, Full Support, Application Model, Sport, Rooms, Studies with Cyclists and Prototype.

LISTA DE FIGURAS

Figura 1 - Comunicação client-side com server-side.....	15
Figura 2 - Visual Studio Code.....	22
Figura 3 - Emulador Android	23
Figura 4 - Seleção de emulador	24
Figura 5 - Firebase	25
Figura 6 - Cloud Firestore	25
Figura 7 - Tela inicial Google Cloud	26
Figura 8 - API's do Google	27
Figura 9 - Código Visual Tela Login	29
Figura 10 - AuthContext.Provider	31
Figura 11 - Regra de autenticação I	34
Figura 12 - Regra de autenticação II	34
Figura 13 - Regra de autenticação III	35
Figura 14 - Regra de autenticação IV	35
Figura 15 - Regra de autenticação V	35
Figura 16 - Código Componentes Tela de Cadastro	36
Figura 17 - Código Componentes Tela de Cadastro II	37
Figura 18 - Código Tela Inicial.....	38
Figura 19 - Código Tela de Criação de Salas I.....	41
Figura 20 - Código Tela de Criação de Sala II	41
Figura 21 - Código Tela de Criação de Sala III	44
Figura 22 - Tela de Login	46
Figura 23 - Cadastro Viver Bike	47
Figura 24 - Tela Inicial.....	48
Figura 25 - Tela de Criação de Sala.....	49
Figura 26 - Tela do Mapa com Rota.....	50
Figura 27 - Tela do Chat	51
Figura 28 - Tela de Apoio.....	52

SUMÁRIO

1. INTRODUÇÃO	8
1.1 Justificativa	9
1.2 Objetivos	9
1.2.1 <i>Objetivo Geral</i>	9
1.2.2 <i>Objetivos Específicos</i>	9
1.3 Metodologia de pesquisa	10
1.4 Organização Textual	11
2 FUNDAMENTAÇÃO TEÓRICA	12
2.1 Criação de aplicações móveis	12
2.2 Aplicativo Nativo	14
2.3 Aplicativo Híbrido	14
2.4. Linguagem <i>Web</i> no Desenvolvimento <i>Mobile</i>	15
2.5. Comparação com apps existentes	17
2.6. Application Programming Interface (API)	19
3. MATERIAIS E MÉTODOS	21
4. DESENVOLVIMENTO	28
4.1. Tela de <i>Login</i>	28
4.2. Tela de cadastro	33
4.3. Tela inicial	37
4.4. Modal para criação de salas	40
4.5. Telas dentro da sala escolhida	42
5. RESULTADOS	46
6. CONSIDERAÇÕES FINAIS	53
REFERÊNCIAS	55

1. INTRODUÇÃO

As vendas de *bikes*, no Brasil, aumentaram 118% no ano de 2020, em comparação ao mesmo período do ano anterior, segundo uma pesquisa realizada pela Associação Brasileira do Setor de Bicicletas (VOGUE GLOBO, 08/2020). Esse crescimento nas vendas nos faz pensar: como essas pessoas, usuárias dessas bikes, têm sido apoiadas no meio digital?

Strava é o aplicativo mais querido dos ciclistas. O *app* é, sem dúvida, um dos recursos mais utilizados e adorados pelos ciclistas *mountain bikers*, corredores e triatletas, justamente pela praticidade e funcionalidade que apresenta, pois mostra rotas, caminho percorrido, entre outras coisas (BR CICLISMO, 2021) e, nesse contexto, é um aplicativo que não possui grande concorrência no mercado.

Observa-se que mais aplicativos voltados para esse grupo são bem vindos, pois são necessários mais recursos e com novas ideias que não são encontrados em outros lugares como, por exemplo, o recurso de salas dinâmicas que facilita o encontro entre os usuários do aplicativo.

Sobre o ciclismo, pode-se afirmar também que:

O Ciclismo também é uma ótima fonte para perda de peso e sedentarismo para pessoas que têm dificuldade de fazer suas atividades em academias. Seja para tratar ou para prevenir a obesidade, é preciso se alimentar de forma saudável e incluir a prática de atividade física na rotina. O ciclismo é um grande aliado nesse momento. Pedalar é uma atividade indicada para pessoas de todas as idades e é indicado no combate à obesidade, pois atua no metabolismo aeróbico, quando a gordura é utilizada como fonte de energia após um certo tempo de exercício (TERRA, 2018).

Outro fator relevante, segundo Diego Salgado, colunista do UOL e ciclista, é: “Uma pesquisa com esportistas amadores apontou o ciclismo como o esporte mais seguro em meio à pandemia do novo coronavírus. A bicicleta ficou à frente de corrida de rua e musculação” (DIEGO SALGADO, UOL, 2020).

Por fim, este projeto tenta suprir necessidades de um grupo que está se tornando relevante no nosso meio, trazendo mais saúde a uma grande parte da

população e proporcionando um ecossistema intuitivo que facilita a organização de grupos de ciclismo.

1.1 Justificativa

O estudo deste tema justifica-se tendo em vista que o mundo passa por um momento complicado em âmbito nacional e internacional. A pandemia ainda é uma realidade das famílias, e o sedentarismo cresce junto com o número de contágios. Nesse contexto, as pedaladas podem salvar vidas, apoiando pessoas que querem sair desse quadro. A construção do *Viver Bike* influencia pessoas que estão se tornando mais sedentárias e que tendem a piorar seus quadros durante e, provavelmente, após a Covid a melhorar sua saúde e crescerem socialmente.

1.2 Objetivos

Este tópico tem como objetivo apresentar o objetivo geral e os objetivos específicos centrados no tema proposto no presente trabalho de conclusão de curso, sendo que o objetivo geral define o foco principal do projeto e os vários objetivos específicos apoiam o objetivo geral, promovendo maior compreensão do conteúdo.

1.2.1 Objetivo Geral

O *Viver Bike* tem como objetivo geral desenvolver um aplicativo móvel para apoiar novos ciclistas, no qual os usuários poderão criar salas de interação, selecionar uma rota e definir o ponto de saída e chegada, assim como a distância e o tempo do trajeto.

1.2.2 Objetivos Específicos

- Permitir a criação e gerenciamento de grupos de ciclismo;
- Permitir a comunicação entre ciclista por meios digitais;
- Inserção de mapas e rotas para facilitar na orientação dos ciclistas;
- Inserção de um sistema de suporte em caso de furtos e acidentes;

1.3 Metodologia de pesquisa

Esta pesquisa, segundo sua natureza, é “resumo de assunto buscando apenas sistematizar uma área de conhecimento, usualmente indicando sua evolução histórica e estado da arte” (WAZLAWICK, 1967, p.21).

Segundo seus objetivos, é uma “pesquisa exploratória, vamos examinar um conjunto de fenômenos, buscando anomalias que não sejam ainda conhecidas e que possam ser, então, a base para uma pesquisa mais elaborada”(WAZLAWICK, 1967, p.22).

Na concepção de Wazlawick, “a pesquisa bibliográfica é fundamental e necessária para qualquer trabalho científico, mas ela em si não produz qualquer conhecimento novo. Apenas supre o pesquisador de informações públicas que ele ainda não possuía” (WAZLAWICK, 1967, p.23).

Ainda, segundo seus procedimentos técnicos, esta pesquisa é bibliográfica, pois foi levantado artigos e revistas para sua compreensão, visando coletar conhecimentos existentes. Como essa técnica não gera conhecimento, apenas suprindo o autor com conhecimentos existentes (WAZLAWICK, 2014), o projeto também é um trabalho de pesquisa experimental, uma vez que tem o objetivo de desenvolver um aplicativo que permite a melhor interação entre os usuários a partir de salas organizacionais com rota, chat e apoio.

Para Wazlawick, “a preparação de uma pesquisa é uma etapa que deve ser realizada antes que se comece a escrever sobre a pesquisa” (WAZLAWICK, 1967, p.37). Esse autor afirma, ainda, que “a revisão bibliográfica de um trabalho de pesquisa em computação, em geral, deve ser um tratado sobre a área de pesquisa” (WAZLAWICK, 1967, p.37).

As áreas pesquisadas foram:

- Influência do ciclismo durante o período de pandemia;
- Influência do ciclismo na luta contra a obesidade e sedentarismo;
- Técnicas e boas práticas na linguagem de programação *Javascript*;
- Técnicas e boas práticas para utilização da ferramenta de trabalho *React Native*;
- Como aplicar o conceito de bancos não relacionais com a utilização do framework *Firebase Firestore*;

- Comunicação com *Api* 's do Google.

1.4 Organização Textual

A organização dos capítulos seguintes foi assim realizada: fundamentação teórica, materiais e métodos, desenvolvimento, resultados e considerações finais.

No próximo capítulo, será apresentado um contexto teórico dos conceitos aplicados neste trabalho, capítulo que se faz necessário para melhor compreensão acerca do projeto.

No capítulo subsequente, materiais e métodos, apresenta-se aos leitores quais ferramentas foram utilizadas para o desenvolvimento da aplicação, citação de autores, suas contribuições e, por fim, o ambiente de desenvolvimento.

Posteriormente, no capítulo de desenvolvimento, é exibido como a aplicação foi desenvolvida e demonstra o código em *javascript* unido ao *framework React Native* e as comunicações com as *Application Programming Interface (API): Map SDK for Android, Maps Javascript API, Places API, Directions API*.

No capítulo Resultados, são apresentadas imagens ilustrando cada tela da aplicação e suas funções.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, será abordada a fundamentação teórica necessária para melhor compreensão do projeto, tendo sido dividido nos seguintes tópicos: criação de aplicações móveis, aplicativo nativo, aplicativo híbrido, linguagem *web* no desenvolvimento *mobile*, comparação com *apps* existentes e *Application Programming Interface* (API).

2.1 Criação de aplicações móveis

Os dispositivos móveis crescem cada dia mais e já representam a maioria dos sistemas computadorizados atualmente. Por isso, torna-se cada vez mais importante criarmos aplicativos voltados para o segmento do qual os ciclistas fazem parte (DEVMEDIA, 2014).

As aplicações móveis têm se tornado uma febre atualmente e é cada vez mais comum ver pessoas utilizando seus *smartphones* para além de apenas ligações como era comum durante os anos 90 e início dos anos 2000, isso motivado pelas inúmeras evoluções de *hardwares* que os *smartphones* vêm recebendo ao longo dos últimos anos. Os celulares parecem não ter limites quando se fala em evolução. Cada vez mais novos recursos aparecem, melhorias são adicionadas e tudo continua ocupando o mesmo espaço (FÁBIO JORDÃO,2009).

Devido a tantas evoluções, tem sido muito comum a preferência na criação de aplicações móveis em relação às aplicações *desktop*, que são aquelas instaladas em computadores físicos como computadores de mesa e *notebooks*. Porém existem aplicações que ainda são melhores desempenhadas em *desktop* por sua grande potência em *hardware*. Conforme Rene Ribeiro (2018) cita a Apple, atualmente a maior empresa do segmento diz que seus novos iPad Pros são mais rápidos do que 92% de todos os laptops, tablets e PCs conversíveis comercializados. E incluindo PCs com processadores Intel Core i7 que atualmente ainda estão presentes em grande parte dos computadores de grande porte do mercado.

Uma característica importante ao projetar e criar aplicativos para dispositivos móveis, é que “estes possuem um propósito bem definido. Ao contrário dos aplicativos

para *desktop*, em que tínhamos aplicativos de propósitos mais gerais, nos aplicativos móveis tem-se mais oportunidades para aplicativos que resolvem problemas mais específicos” (DEV MEDIA, 2014). Um exemplo que pode ser citado é o pacote *Adobe*, que tem inúmeros propósitos no contexto de *desktop* e pode ser acessado de forma única nos computadores de mesa. Para a criação e utilização desse mesmo pacote em um *smartphone*, é necessário fragmentá-lo em várias aplicações com seus objetivos específicos, fazendo do *smartphone* a melhor opção no desenvolvimento de uma aplicação que tem o objetivo específico de ser um centro de interação entre usuários de mesmo perfil.

Em relação à criação de aplicações móveis, em termos de linguagens e ferramentas de trabalho, é visível uma maior distribuição de recursos para o seu desenvolvimento. Atualmente os dois maiores sistemas operacionais ligados às plataformas móveis são *Android* e *iOS*. As duas plataformas são totalmente opostas em termos de arquitetura, o que nos anos iniciais de produção das plataformas gerava uma grande dificuldade no desenvolvimento de aplicações. No caso de uma aplicação que desse suporte para os dois sistemas operacionais, era necessário que fosse desenvolvido duas aplicações diferentes, com dois códigos diferentes, uma para cada contexto, os chamados aplicativos nativos. Nesse período, as linguagens que se destacaram no desenvolvimento de aplicações móveis eram o *Java* para *android* e *Swift* para o desenvolvimento *iOS*.

O Desenvolvimento *Android* e *iOS* gira em torno de pacotes chamados *software development kit* (SDK). “Estes pacotes reúnem bibliotecas, exemplos de códigos, notas técnicas, depuradores, compiladores, *APIs*, entre outros utilitários que permitem manipular arquivos necessários durante o desenvolvimento das aplicações” (BOCARD,2020). Para cada SDK existe uma linguagem específica de programação e plataformas específicas. Logo, ao desenvolver para *Android* e *iOS* de forma nativa, como citado anteriormente, era necessário criar dois códigos distintos gerando um *rework* excessivo. “O desenvolvimento de aplicações *Android* é praticamente impossível sem o *Android SDK*, material produzido pelo Google. Afinal, a empresa é a proprietária da plataforma” (BOCARD,2020).

Seguindo a lógica do desenvolvimento *Android*, faz-se necessário a utilização de *SDK* no desenvolvimento *iOS* e diferente do *Android SDK* o *iOS SDK* é um produto fornecido pela própria *Apple*.

Segundo Bocard (2020), “além dos pacotes SDK nativos das plataformas, existem inúmeros outros pacotes, e qualquer ferramenta digital está sujeita a ter seu próprio SDK “ (BOCARD, 2020). Algumas delas são: *Facebook SDK, Mobile Ads SDK, Admob, Mapbox, Google Maps, Pagar.me* etc.

O projeto abordado no trabalho faz uso dos pacotes SDK 's, *Android SDK* e *Google Maps*, porém em uma plataforma híbrida, possibilitando a integração entre *Android* e *iOS* a partir do mesmo código fonte por meio da linguagem *web Javascript*.

2.2 Aplicativo Nativo

Com o advento da evolução dos *smartphones*, fez-se necessário a criação de lojas virtuais que permitissem o *download* e acesso de novos aplicativos de forma rápida e prática. “O aplicativo nativo é o tipo de aplicativo comumente encontrado nas lojas de aplicativos e construído em uma linguagem exclusiva para um determinado sistema operacional” (MARKTEAM, 2020).

Em teoria, características de um aplicativo nativo é que ele é mais rápido e confiável que os demais, permitindo a utilização de recursos nativos do telefone como câmera, GPS e Notificações *Push*. Entretanto, já existem ferramentas de desenvolvimento que otimizam a performance de aplicativos híbridos, tornando-os muito próximos aos nativos.

2.3 Aplicativo Híbrido

Segundo a Redação Cronapp (2018), os aplicativos híbridos surgem como uma solução eficiente para atender às necessidades de usuários de diferentes sistemas operacionais. Afinal, como citado no capítulo anterior, para se criar um aplicativo que rodasse tanto em *Android* quanto *iOS*, seria necessário programá-lo duas vezes em linguagens distintas. Entretanto, os aplicativos híbridos, por meio de uma ferramenta específica, permitem que o programador crie um modelo único de código que é convertido para as plataformas nas quais o aplicativo deve rodar.

Na prática, tem-se um código diferente que atende cada uma das plataformas escolhidas, *Android* ou *iOS*. Portanto a utilização de aplicativos híbridos torna-se interessante a partir da diminuição de custos mediante a utilização da metade dos

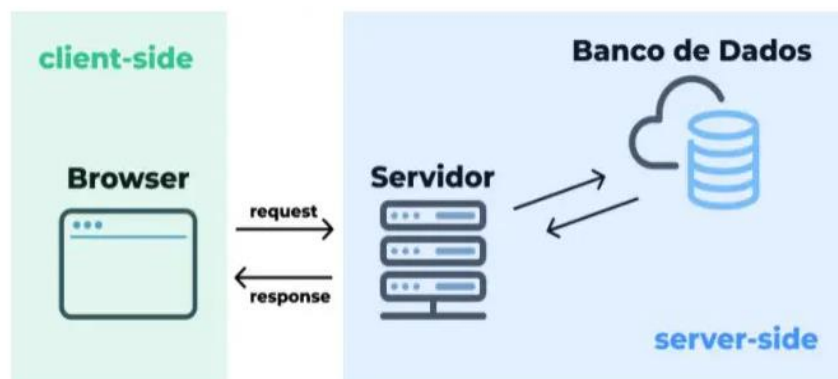
recursos que as aplicações nativas necessitam, também possui uma curva de aprendizagem bem inferior pois o desenvolvedor necessita do domínio de apenas uma linguagem para desenvolver a aplicação e, por fim, a manutenção também é um fator reduzido, pois o esforço para correção de problemas é relativo a apenas um código. Esses inúmeros fatores fazem com que os aplicativos híbridos aos poucos tenham domínio dentro do mercado Mobile.

2.4. Linguagem *Web* no Desenvolvimento *Mobile*

O *Javascript* é uma linguagem de programação que faz parte da tríade do desenvolvimento *web* que é formada por HTML, *Javascript* e CSS. O *Javascript* trabalha na parte *front-end* da aplicação que pode ser definida como a camada visual da aplicação, aquilo que é visto aos olhos humanos. Entretanto com a evolução corriqueira dos *frameworks*, hoje já é possível utilizar o *Javascript* para desenvolver o *back-end*, aquilo que não é visto pelo usuário e que normalmente se comunica com o banco de dados.

Além de *back-end*, é possível também utilizar o *Javascript* para o desenvolvimento de aplicações móveis com o auxílio, por exemplo, do *framework React Native*, que será melhor abordado nos capítulos seguintes deste trabalho. De acordo com a figura 1 o *Javascript* funciona como a linguagem responsável pela lógica do lado do cliente, que é a parte que é visível e acessada pelo próprio navegador e é por meio dele que é possível a comunicação com o lado do servidor, onde normalmente ficam as lógicas de estrutura de dados.

Figura 1 - Comunicação client-side com server-side



Fonte: Cairo Noletto, 2020.

Existem inúmeras vantagens de se utilizar *Javascript* no desenvolvimento de uma aplicação como, por exemplo, pelo fato dela ser uma linguagem muito leve, o interpretador nativo da linguagem interpreta o código *Javascript* muito rápido. Além disso, o *Javascript* é uma linguagem bastante versátil e flexível, por ser uma linguagem interpretada, pode ser utilizada em vários navegadores diferentes. Também é possível usar o *Javascript* do lado servidor, tornando possível criar até mesmo jogos e aplicações móveis.

Um dos *frameworks* mais conceituados do momento é o *React Native*, uma das inúmeras ferramentas que possibilitam o *Javascript* ser utilizado para atividades que não fazem parte do seu princípio básico. “O *React Native* é um framework baseado no já aclamado *React*, desenvolvido pela equipe do *Facebook*, que possibilita o desenvolvimento de aplicações *mobile*, tanto para *Android*, como para *iOS*, utilizando apenas *Javascript*” (BECKER, 2021).

De inúmeras funções interessantes que o *React Native* possui, uma delas é o *Hot Reload*, que permite que a cada nova atualização no código, no momento que o arquivo que está sendo editado é salvo, a própria aplicação entende que houve uma alteração e atualiza automaticamente para a alteração ser refletida para o desenvolvedor, por utilizar *javascript* que em sua essência é uma linguagem de navegador.

O *React Native* trabalha com o conceito de estados de componente. “Quando trabalhamos com programação nós definimos estado o tempo todo. O estado de uma aplicação nada mais é que as informações armazenadas no nosso programa em um determinado tempo”(SOUZA, 2018). Logo a partir dos estados de aplicação é possível manipular componentes de forma que estes possam apresentar novos valores de acordo com o desejo e objetivo do desenvolvedor. Outro conceito bastante importante no desenvolvimento utilizando o *React Native* são os *Lifecycles*. Um componente em *ReactJs* e *React Native* tem contabilizado o momento que ele foi criado, o momento que ele foi atualizado e o momento que este foi destruído. Por meio de algumas chamadas de funções próprias do *React Native* é possível manipular estes ciclos de vida. O ciclo de vida mais comumente utilizado para manipular um estado de componente é o momento que o componente é criado, no contexto do *React Native + Hook's* a função responsável é chamada de “*useEffect*”. Um exemplo para esta utilização é quando um usuário deseja que um componente de carregamento de tela

tenha seu estado definido como falso no momento em que um componente de tela seja construído. Portanto assim que o componente for construído aquele componente de carregamento deixará de ser visível na tela.

No projeto do Viver Bike no momento em que a tela inicial é criada é utilizado o *“useEffect”* para o consumo de dados do banco de dados e a construção da lista de salas no momento exato que a tela é criada.

Baseado nos conceitos fundamentais do *React Native* é interessante a sua utilização pois ele permite que seja construída toda uma aplicação *mobile* utilizando linguagem web e ainda sim não torna aquela aplicação uma aplicação web pois não é construído um *HTML5* e sim uma aplicação robusta que não se distingue das feitas de forma nativa utilizando *JAVA*. Logo programadores *web* possuem uma grande facilidade de migrar de aplicações web para aplicações *mobile* a partir do *React Native*.

O *React Native* foi lançado em 2015 e seu código foi colocado no *Github*, que é uma plataforma colaborativa de gerenciamento de versões e compartilhamento de código. A partir disso, o seu código tem sido melhorado ano após anos e, mediante isso, inúmeras aplicações de grande conceito no mercado têm utilizado dos seus recursos. Uber, AirBnB e Instagram estão dentro desse grupo.

O projeto realizado neste trabalho utiliza da ferramenta *React Native* unida ao *Javascript* como principais ferramentas para o seu desenvolvimento. Ferramentas como *React Native* fazem do projeto uma aplicação atual, facilita sua manutenção, diminui custos e promove inovação.

2.5. Comparação com apps existentes

O Viver *Bike*, projeto realizado para este trabalho, possui algumas referências de aplicações para auxiliar e dispor de ideias para sua implementação. A principal aplicação ligada ao Viver *Bike* é o Strava. O Strava é uma aplicação que funciona como rede social para corredores e ciclistas, sendo hoje a maior aplicação do nicho. A aplicação conta com versões para *Android* e *iOS* e permite registrar os dados dos usuários e de suas atividades físicas. O Strava é uma aplicação que está há bastante tempo no mercado dos aplicativos móveis tendo seu lançamento datado no ano de

2009, recebendo inúmeras atualizações desde então e com poucos concorrentes à sua altura.

O Strava mede e cria um histórico das pedaladas do usuário, coletando informações como: distância, altitude acumulada, velocidade máxima e média, batimentos cardíacos, o traçado do percurso no mapa, entre outras. Para isso, o usuário precisa ter um ciclocomputador com GPS ou um celular com sistema *Android* ou *iOS* (PEDRO CURY, 2015).

O Viver *Bike* em comparação ao Strava em sua versão inicial, não conta com rastreamento da localização em tempo real, mas permite que o usuário crie salas com rotas pré-definidas e dados importantes como distância e tempo de percurso, o que o torna do mesmo nicho do concorrente, porém com uma proposta um pouco diferente.

O Strava busca apoiar cada atleta individualmente, já o Viver *Bike* busca apoiar grupos de ciclismo, possibilitando, em um futuro próximo, que os usuários possam criar perfis de grupos e gerar uma coletividade entre esses grupos. A aplicação do Strava busca motivar seus usuários a pedalar por meio da gamificação apresentada no aplicativo, onde os players, ao pedalar, são recompensados com medalhas e pontos em um ranqueamento global. O Viver *Bike* busca motivar os seus usuários a partir de um pressuposto próximo ao do Strava, mas apelando para a coletividade dos usuários, onde existirá um sistema de ranqueamento, porém a evolução será dada aos grupos e às atividades que os grupos realizam para evolução no *ranking*. O crescimento do grupo está ligado aos dados de cada pedal que o grupo se esforça para fazer e o apoio a novos membros que buscam uma equipe para pedalar.

Existem outras aplicações que trazem boas referências para do desenvolvimento do Viver *Bike*, como por exemplo o Wikiloc, uma aplicação criada na Espanha que permite a criação de rotas e compartilhamento com outros usuários, e o Komoot, uma aplicação que tem seu atrativo em torno do cicloturismo, trazendo informações como: onde dormir, onde comer, pontos de interesse em uma rota ou cultural. Porém a aplicação de maior valor na criação do Viver *Bike* e que inspira na sua construção é o Strava.

2.6. Application Programming Interface (API)

Este módulo tenta trazer à compreensão dos leitores o que é uma *Application Programming Interface* (API).

“Uma API é um conjunto de aplicações que permitem a construção de uma interface de comunicação inteligente e, quando configurado, permite que dois sistemas se comuniquem” (BRANDÃO, 2019). Uma API é criada quando existe a intenção de uma empresa que outros desenvolvedores possam criar aplicações relacionadas com seus serviços. Atualmente existem inúmeras API 's no mercado e muitas delas permitem que outros sistemas possam ter acesso a recursos de grande complexidade, que gastariam muito de seus recursos e tempo dos desenvolvedores para recriar.

Existem API 's gratuitas, onde os donos do serviço aproveitam para divulgar seu serviço, e API' s pagas, com custo determinado pelo número de serviços consumidos pelo desenvolvedor. O Google é uma das empresas que disponibiliza o maior número de APIs para consumo, unida ao *Facebook* e à *Amazon*. Um exemplo de consumo de API, segundo a Redação do CanalTech (2020), é quando uma pessoa acessa uma página de um hotel e é possível visualizar dentro do próprio *site* o mapa do *Google Maps* ou quando um usuário tem a opção de fazer o acesso a uma aplicação a partir da sua conta do *Google* ou *Facebook*.

As API 's permitem que aplicativos de funções distintas possam se comunicar entre si com ou sem a intervenção do usuário, elas definem o comportamento de determinado objeto em uma interface. O uso de API 's tem se tornado muito comum, estando cada vez mais presente nas aplicações, e todo dia nasce uma nova API permitindo que desenvolvedores tenham mais praticidade na hora de resolver problemas em suas aplicações, além do aumento na integração entre organizações por meio do compartilhamento de API's. O *Viver Bike* faz uso de API 's, facilitando processos como por exemplo o uso de mapas e rotas a partir do *Google Maps*.

A API do *Google Maps* é uma interface que permite a utilização do mapa do *Google Maps* e todas as propriedades que estão ligadas a ela como por exemplo a opção de criação de rotas. Durante o processo de criação do *Viver Bike* esta API foi bastante utilizada para criação das rotas em cada sala de forma rápida e prática. Para

criar uma rota basta passar como *props* os valores de latitude e longitude da origem e do destino e a partir disso uma rota em preto será gerado. É possível também a partir disto buscar dados de duração de rota e distância de um ponto a outro. É importante ressaltar que há a possibilidade de definir o tipo de veículo a ser utilizado. Além disto o Google *Maps* faz parte de um pacote com outras *API's* do Google localizado no Google *Cloud*, plataforma do Google que disponibiliza funções para trabalhar com sistemas em nuvem. Existem outras *API's* para utilização de mapas porém a preferência pelo Google *Maps* deve-se a uma assistência técnica maior que o Google oferece além de um maior número de tutoriais que explicam como implementá-lo em uma aplicação. Uma solução de *API* no caso de que não haja o desejo de utilizar a *API* do Google *Maps* é a utilização da *API MapBox* que tem o custo reduzido porém sua implementação tem maior complexidade e ela disponibiliza de poucos recursos de assistência de implementação.

3. MATERIAIS E MÉTODOS

Nesta etapa do trabalho, diferente das anteriores, o objetivo é demonstrar como o projeto foi construído, os materiais que foram utilizados e os métodos utilizados.

Para o desenvolvimento da aplicação foi utilizado uma máquina com alto potencial por conta do uso de emuladores android que demandam bastante memória RAM e processamento. O sistema operacional utilizado foi o *Windows 10* unido ao emulador SDK do *Android Studio*. Na tabela 1 é demonstrado os detalhes de configuração do sistema.

Tabela 1 - Configuração da máquina utilizada

Memória Ram	16 Gb de Ram a 2666Mhz
Processador	Amd Ryzen 5 3600 x
Placa de Vídeo	Nvidia Gtx 1060
Disco Rígido	1 HDD de 1Tb e 1 SSD de 240Gb

Fonte: Próprio Autor

A partir do sistema citado, foi necessária a configuração do ambiente para desenvolvimento, utilizando-se, como citado antes, da linguagem *Javascript* e o *framework React Native*. A plataforma utilizada para escrever o código da aplicação foi o VSCode.

O *Visual Studio Code*, ou VSCode, é uma plataforma de desenvolvimento *Open Source* da *Microsoft*, cujo lançamento ocorreu em 2015 e foi destinada ao desenvolvimento de aplicações *web*. O VSCode é uma ferramenta leve e multiplataforma que está disponível para *Windows*, *Mac OS* e *Linux*. A aplicação dá suporte para linguagens como *Javascript*, *Python* e *C++*, além de atender uma gama de projetos como *ASP.NET* e *Node.js*. A figura 2 demonstra como o *Visual Studio* se comporta visualmente.

Figura 2 - Visual Studio Code

```

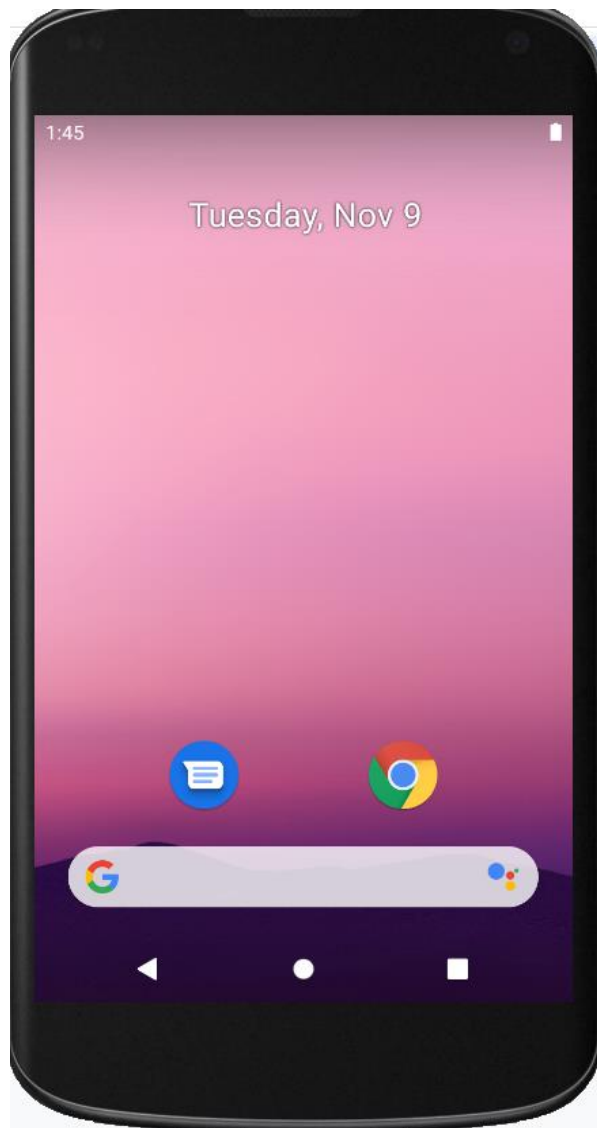
src > screens > JS RoomScreen.js > RoomScreen > popover
12  const [Duration, setDuration] = useState("");
13  const dispatch = useDispatch();
14
15  let {width, height} = Dimensions.get("window");
16  const ASPECT_RATIO = width / height;
17  const LATITUDE_DELTA = 0.0922;
18  const LONGITUDE_DELTA = LATITUDE_DELTA * ASPECT_RATIO;
19
20  console.tron.log('localizacao:', localizacao);
21
22  const origin = {...localizacao.localizacaoOrigem};
23  const destination = {...localizacao.localizacaoDestino};
24
25  const GOOGLE_MAPS_APIKEY = "AIzaSyC0b1180WfW5ZaMGMUZFU4csC06-RuDU";
26
27  function DurationFormat(Duration) {
28    return Duration / 0.6;
29  }
30
31  function popover() {
32    return (
33      <Popover
34        triggers={triggerProps} > {
35        return (
36          <Fab
37            {...triggerProps}
38            position="absolute"
39            borderRadius="full"
40            colorScheme="indigo"
41            right={3}
42            bottom={60}
43            Label="Detalhes Rota"
44          />
45        );
46      });
47    <Popover.Content accessibilityLabel="Delete Customer" w="50">
48    <Popover.Arrow />
49    <Popover.Header>Detalhes da Rota</Popover.Header>
50    <Popover.Body>Distancia: ${Distance}km
51    Duração: ${DurationFormat(Duration).toFixed(2)} Minutos</Popover.Body>
52    </Popover.Content>
53    </Popover>
54  );
55  }
56
57  return (
58    <View style={styles.container}>
59      {popover()}
60      <MapView
61        style={styles.maps}

```

Fonte: Próprio Autor

O Emulador *Android* do *Android Studio* é uma aplicação que simula um celular *android* em seu computador como se fosse um *smartphone* físico. É um sistema muito inteligente que auxilia na visualização e depuração do que está sendo feito no código. Segundo Fillipe Cordeiro (2019), dentro do emulador *android* existem cópias prontas do sistema operacional, que inclui uma série de aplicações pré-instaladas. A figura 3 ilustra como é exibido o emulador utilizado para testes da aplicação.

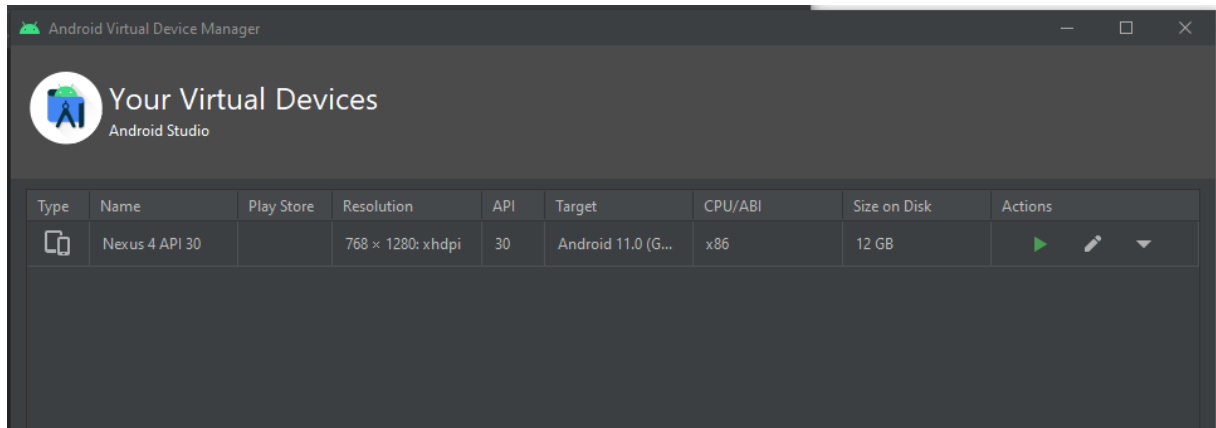
Figura 3 - Emulador Android



Fonte: Próprio Autor

A Figura 4 é uma ilustração de como é criado um dispositivo virtual no *Android Studio*.

Figura 4 - Seleção de emulador

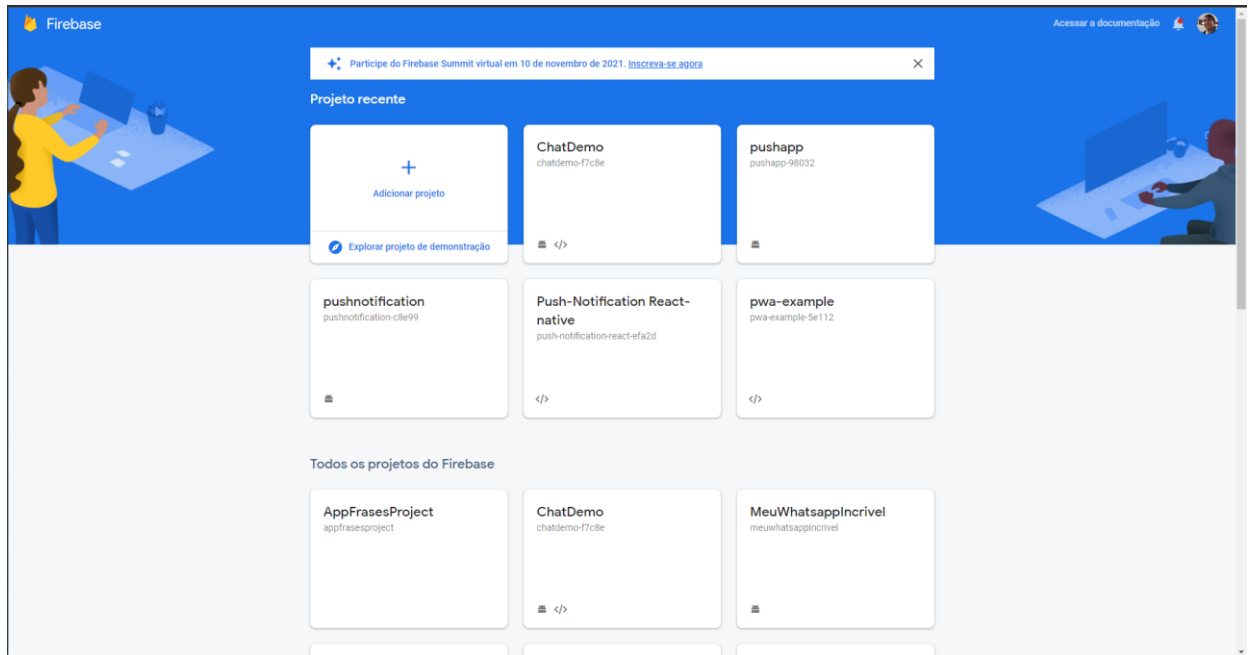


Fonte: Próprio Autor

Como banco de dados para armazenamento de algumas informações do usuário e da aplicação, foi utilizado o *Firebase*, um banco de dados NoSql disponibilizado pelo Google. NoSql são bancos não relacionais, onde os dados ficam em um mesmo registro e não há relação entre os registros. Comumente bancos não relacionais são usados em *Data Science*, porém é válida para aplicações que não demandam muita complexidade em seus bancos. O *Firebase* também dispõe de algumas *API's* para facilitar o desenvolvimento da aplicação como por exemplo: a *API* de Autenticação que permite o usuário cadastrar e verificar a existência de novos usuários diminuindo a necessidade de aplicações de comunicação com banco de dados. Inúmeras aplicações atualmente utilizam de recursos disponibilizados pelo *Firebase* para diminuir o trabalho na hora de desenvolver uma aplicação do zero. O *Viver Bike* é composto em grande parte por consumo de *API 's* utilizando o banco apenas para gerar e armazenar as salas e as rotas pré-definidas por quem criou as salas.

A Figura 5 demonstra a tela inicial de acesso a página do *Firebase* e os projetos do administrador da conta.

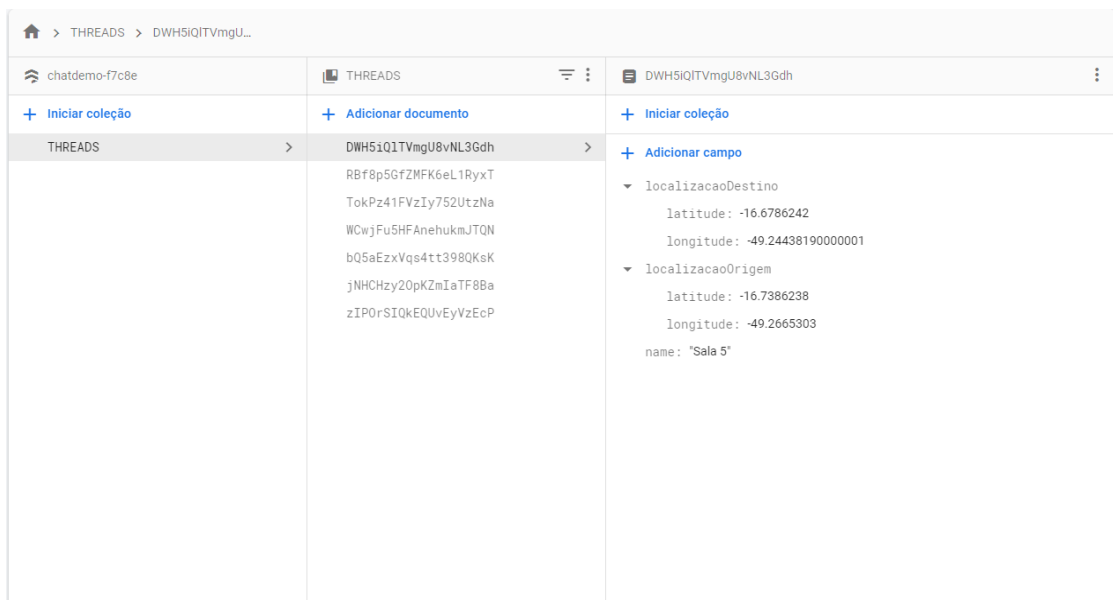
Figura 5 - Firebase



Fonte: Próprio Autor

A Figura 6 demonstra de forma simplificada como é apresentada a pagina do *Firestore* onde é encontrado as *collections* do banco de dados.

Figura 6 - Cloud Firestore

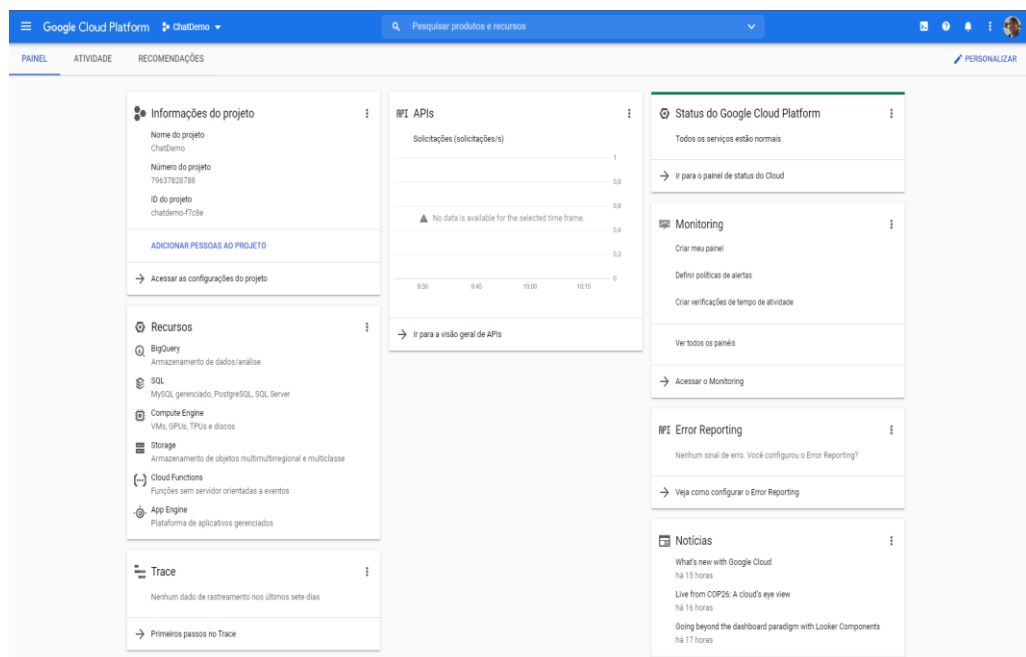


Fonte: Próprio Autor

Por fim, durante o processo de desenvolvimento da aplicação foi utilizado um método essencial, a API do Google *Maps* a partir do Google *Clouds*. O Google *Clouds* é a plataforma em nuvem e de suporte a desenvolvedores do Google, a partir dessa plataforma é possível acessar API 's, Máquinas Virtuais, Kubernetes, *Cloud Storage* entre outros. O Google *Maps* é a API de suporte ao mapa mais famosa do mundo, a maior parte das aplicações grandes do mercado utilizam de suas *features* no desenvolvimento de suas aplicações. Porém, o Google *Maps* não é uma ferramenta gratuita. O Google utiliza um sistema de faturamento individual para cada serviço do Google *Clouds*, o valor aumenta de acordo com a quantidade de serviços e a qualidade dos recursos contratados e, no caso das APIs, o gasto é calculado a partir do número de requisições feitas às APIs. A API *Directions* API tem um consumo de R\$ 57,52 a cada 1000 requisições, já a Google *Places* é gratuita.

Existem outras APIs que dão suporte a mapas, porém tais serviços não são tão completos como o do Google, possuem maior complexidade no seu desenvolvimento e o custo é aproximado ao do Google. Portanto, neste projeto foram utilizadas as APIs do Google. A Figura 7 demonstra a página inicial do Google *Cloud* após o usuário ter logado.

Figura 7 - Tela inicial Google Cloud



Fonte: Próprio Autor

A Figura 8 exibe a tela de ativação de API's no Google *Cloud*. Nesta página o autor da aplicação pode escolher quais API's seriam interessantes no desenvolvimento da aplicação, sendo a API de rotas a principal entre elas.

Figura 8 - API's do Google

The screenshot displays the Google Cloud Platform API Library interface. At the top, there is a blue navigation bar with the Google Cloud Platform logo and a user profile icon. Below the navigation bar, the page title is "Biblioteca de APIs". A central heading reads "Olá! Esta é a Biblioteca de APIs" followed by a sub-heading "A Biblioteca de APIs contém documentos, links e uma experiência de pesquisa inteligente." and a search input field with the placeholder text "Pesquisar APIs e serviços".

The main content area is divided into two sections: "Mapas" and "Aprendizado de máquina".

Mapas (VER TUDO (15))

- Maps SDK for Android** (Google): Maps for your native Android app.
- Maps SDK for iOS** (Google): Maps for your native iOS app.
- Maps JavaScript API** (Google): Maps for your website.
- Places API** (Google Enterprise API): Get detailed information about 100 million places.
- Roads API** (Google Enterprise API): Snap-to-road functionality to accurately trace GPS breadcrumbs.
- Directions API** (Google Enterprise API): Directions between multiple locations.

Aprendizado de máquina (VER TUDO (9))

- Dialogflow API** (Google Enterprise API): Builds conversational interfaces.
- Cloud Vision API** (Google Enterprise API): Image Content Analysis.
- Cloud Natural Language API** (Google Enterprise API): Provides natural language understanding technologies, such as sentiment analysis, entity...
- Cloud Speech-to-Text API** (Google Enterprise API): Speech recognition.
- Cloud Translation API** (Google Enterprise API): Integrates text translation into your website or application.
- AI Platform Training & Prediction API** (Google Enterprise API): An API to enable creating and using machine learning models.

Filter by

VISIBILIDADE

- Público (355)
- Particular (2)

CATEGORIA

- Publicidade (14)
- Análise (5)
- Big Data (15)
- Blog e CMS (1)
- Computação (8)
- CRM (1)
- Bancos de dados (6)
- Pilhas do desenvolvedor (2)
- Ferramentas para desenvolve... (18)
- E-mail (1)
- APIs do Google Enterprise (182)
- Serviços financeiros (1)
- Firebase (5)
- Google Workspace (18)
- Healthcare (4)
- Aprendizado de máquina (9)

Fonte: Próprio Autor

4. DESENVOLVIMENTO

Este módulo é o principal módulo do trabalho, diferente dos módulos anteriores que buscavam justificar o motivo da aplicação, o módulo de desenvolvimento tem como objetivo discorrer sobre como foi desenvolvido o *Viver Bike*. A forma como serão apresentadas as partes da aplicação será de acordo com cada tela desenvolvida, a partir da tela de *login* até as telas de sala.

4.1. Tela de *Login*

O Sistema de autenticação é uma das *features* mais indispensáveis em toda aplicação bem construída. Quase todas aplicações atualmente possuem dados sensíveis onde apenas o usuário quer ter acesso a esses dados. No caso do *Viver Bike* não é diferente: o usuário terá um perfil com dados pessoais do qual ele deseja esconder informações como, por exemplo, endereço residencial ou mesmo número de telefone. O *Login* tem como objetivo gerenciar a permissividade de acesso do usuário a aplicação, sendo necessário a inserção de um username (nome de usuário) e uma senha para acesso. O username no *Viver Bike* é um endereço de e-mail válido que tenha sido registrado na tela de cadastro de usuário e a senha também possui o mesmo processo de cadastro.

A tela de login foi desenvolvida utilizando a linguagem Javascript, como já citado anteriormente, unido ao *framework React Native*. Os componentes apresentados na tela de login são: Título “Bem-vindo ao *Viver Bike*”, Campo de entrada do e-mail, Campo de entrada do *Password* (Senha) e botão de ação “*Login*”. Os campos de entrada de dados são os responsáveis pela autenticação do usuário no *Firebase* e o botão de *Login* é responsável por disparar a ação que vai fazer essa comunicação. O título tem uma função simples de dar boas vindas aos usuários da aplicação. O Código da figura 9 refere-se a tela de login apresentada no capítulo de resultados posteriormente.

Figura 9 - Código Visual Tela Login

```

import React, {useContext, useState} from 'react';
import {View, StyleSheet} from 'react-native';
import {Title} from 'react-native-paper';
import FormInput from '../components/FormInput';
import FormButton from '../components/FormButton';
import {AuthContext} from '../navigation/AuthProvider';

export default function LoginScreen({navigation}) {
  const {login} = useContext(AuthContext);

  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');

  return (
    <View style={styles.container}>
      <Title style={styles.titleText}>Bem vindo ao Viver Bike</Title>
      <FormInput
        LabelName="Email"
        value={email}
        autoCapitalize="none"
        onChangeText={userEmail => setEmail(userEmail)}
      />
      <FormInput
        LabelName="Password"
        value={password}
        secureTextEntry={true}
        onChangeText={userPassword => setPassword(userPassword)}
      />
      <FormButton
        title="Login"
        modeValue="contained"
        LabelStyle={styles.loginButtonLabel}
        onPress={() => login(email, password)}
      />
    </View>
  );
}

```

Fonte: Próprio Autor

O *Javascript* é uma linguagem orientada a funções, e as primeiras versões do *React Native* o *framework* tentava unir o mundo da orientação objeto com classes e as funções da orientação a funções criando, assim, as classes funcionais, porém essa união não foi tão efetiva e trouxe mais complexidade para o *ReactJs* e *React Native*. Contudo, no ano de 2019, com toda complexidade das classes funcionais e inúmeros *feedbacks* dos usuários da plataforma, a equipe do *React Native* lançou outro *framework* que possibilita a utilização de apenas funções básicas para desenvolvimento com *React Native*, nasce então o *framework* “Hooks”. Contextualizado sobre o nascimento e objetivo do *framework* “Hooks”, na figura anterior, primeiramente é demonstrado as importações de bibliotecas para a comunicação com outros arquivos *javascript* e componentes já pré-desenvolvidos, facilitando e acelerando na criação da estrutura da tela de *Login*.

Logo abaixo das importações de bibliotecas, é demonstrado a criação da função principal da tela que exportará toda a camada visual desta tela para a camada de renderização principal da aplicação encontrada no arquivo “app.js”, que será melhor discutido posteriormente. Essa sintaxe funcional só é possível pela utilização da *Framework “Hooks”*, já citado anteriormente. O *Framework “Hooks”* não é apresentado nas importações, pois ele já vem embutido no *Framework “React Native”*.

Posteriormente, é possível visualizar 3 chamadas de funções que possuem o mesmo padrão de prefixo “use” e a atribuição do seu valor retorno a um objeto ou um “array”. Esse padrão de prefixo é o utilizado para todas as funções ligadas ao *Framework “Hooks”*. A função “*useState()*”, por exemplo, tem o objetivo de criar estados manipuláveis de componentes. Uma das principais, senão a principal feature que o *React Native* possui é a de manipulação de estados. Um estado de componente, segundo William Oliveira Souza (2018), é onde os dados de componente são armazenados e, além disso, esses dados podem ser manipulados de forma dinâmica. Um relógio do qual os segundos são demonstrados em tempo real na tela é um exemplo clássico de manipulação de estado, onde o desenvolvedor inicializa o estado de segundos como zero e determina que esse estado vai incrementar em +1 a cada novo segundo, quando esse estado de segundo dentro da lógica das horas for envolto em um componente visual como, por exemplo, “<text>”, ele irá ser demonstrado em tempo real na exibição da tela.

Os “*useState()*” encontrado na figura 9, é utilizado para definir um estado para e-mail e senha, quando um novo caractere for adicionado nos respectivos campos, esse caractere já será refletido dentro da variável de estado em tempo real e essas variáveis podem ser enviados para função que comunica com o *Firebase* através de um botão por exemplo.

O “*useContext()*” tem como objetivo o acesso ao contexto de um *Provider*, no caso apresentado na figura 10, o *AuthContext* é uma referência ao provedor de autenticação onde fica a lógica de comunicação entre a aplicação e o *Firebase*. O Código referente ao *AuthContext.Provider* é exibido na figura 10.

Figura 10 - AuthContext.Provider

```

1  import React, {createContext, useState} from 'react';
2  import auth from '@react-native-firebase/auth';
3
4  export const AuthContext = createContext({});
5
6  export const AuthProvider = ({children}) => {
7    const [user, setUser] = useState(null);
8
9    return (
10     <AuthContext.Provider
11       value={{
12         user,
13         setUser,
14         login: async (email, password) => {
15           try {
16             await auth().signInWithEmailAndPassword(email, password);
17           } catch (e) {
18             console.log(e);
19           }
20         },
21         register: async (email, password) => {
22           try {
23             await auth().createUserWithEmailAndPassword(email, password);
24           } catch (e) {
25             console.log(e);
26           }
27         },
28         logout: async () => {
29           try {
30             await auth().signOut();
31           } catch (e) {
32             console.error(e);
33           }
34         },

```

Fonte: Autor

Anteriormente, ao final da explicação da tela de *Login*, é necessário conceituar e explicar o que ocorre no ambiente do *AuthContext.Provider*. O “*AuthContext.Provider*” tem como objetivo então identificar se esse usuário está logado ou acessar a conta, cadastrado realmente ou realizar o cadastro, ou deseja sair da sua conta, além de persistir a sessão desse usuário passando todos os dados dele para dentro da aplicação, a partir do momento que ele a acessa corretamente. As funções aqui utilizadas são “*createContext()*”, “*useState()*”, “*auth().signInWithEmailAndPassword(Email,Senha)*”, “*auth().createUserWithEmailAndPassword(Email, Senha)*” e “*auth().signOut()*”.

Na linha 4 é possível visualizar a criação do contexto do autenticador. Já na linha 7 é criado um estado de usuário que identifica este usuário ou adiciona um novo usuário. A partir da linha 10, a API de autenticação do *Firebase*, no caso do *login*, verifica se o usuário já existe e faz o *login* e, se não existir, quando solicitado, a API

de cadastro do *Firebase* manipula esse estado a partir de uma inserção de dados pelo usuário e persiste esses dados, além de realizar o acesso automaticamente à aplicação logo após o cadastro. E, por fim, da linha 28 a 34, está refletida a chamada da função “*signOut()*” referente a deslogar que, quando chamado, permite o usuário sair de sua conta.

Retornando à figura 9, que demonstra a invocação de componentes referentes à tela de *Login*, é possível visualizar que o *React Native* utiliza de uma sintaxe de componentes visuais muito semelhante às *tags* do HTML5, conhecido como JSX, porém os tais não funcionam da mesma forma que na linguagem referida. O *ReactJs* e o *React Native*, segundo a documentação oficial encontrada no portal do *framework*, diz que o JSX produz “elementos” *react* e que “a lógica de renderização é inerentemente acoplada com outras lógicas de UI: como eventos são manipulados, como o *state* muda com o tempo e como os dados são preparados para exibição”. Como também pode-se perceber esses componentes JSX são englobadas em uma propriedade *return*, isso deve-se ao fato de que esses componentes vão ser retornados na chamada da função no momento que todas as telas forem renderizadas no arquivo *app.js*.

O componente “*View*” é responsável pela criação do container visual principal da tela, naturalmente o componente “*view*” no conceito de herança é o componente pai de outros componentes filhos. O componente “*Title*” é responsável por determinar o título da aplicação na tela de login por default, esse componente já possui estilização de tamanho de fonte e tipo de fonte, porém pode ser estilizada de outras formas a partir da propriedade “*Style*”. O componente posterior ao componente “*Title*” é o componente “*FormInput*”, este componente diferente dos anteriores é um componente personalizado, o que quer dizer que ele não é diretamente consumido de uma biblioteca.

Um recurso muito interessante do *React Native* é que ele permite a criação de componentes de acordo com a necessidade do desenvolvedor. No caso do “*FormInput*”, fez-se necessário um componente que cria-se um campo de inserção de dados com uma estilização genérica para a aplicação toda, logo não há mais a necessidade da criação de um novo campo para cada caso de uso.

Em *React Native* existe um conceito muito importante em torno dos componentes que são as “*props*”. Segundo a documentação do *React Native*, “quando

o *React* vê um elemento representando um componente definido pelo usuário, ele passa atributos JSX e componentes filhos para esse componente como um único objeto. Nós chamamos esse objeto de “props”. Resumidamente a forma do qual um componente pai pode passar atributos JSX para seus componentes filhos é por meio de “props”. No caso do “*FormInput*” e “*FormButton*”, é possível visualizar as props passadas em cada caso, o que permite um componente ser genérico para vários contextos, como um mesmo “*FormInput*” ser utilizado para inserção de “e-mail” e “password”.

4.2. Tela de cadastro

Diferente da tela de *login*, a tela de cadastro tem o objetivo de criar uma credencial válida para a autenticação do usuário. O *Firebase* do Google facilita a criação de novos usuários no *Viver Bike* a partir de sua API de autenticação, além de possuir outros recursos de criação de cadastro a partir de uma conta Google ou mesmo uma conta *Facebook*.

Quatro componentes estão presentes na tela de cadastro de usuário, são eles: Dois “*FormInput*” de entrada de dados, um botão de cadastro e um botão de retorno para a tela de *login*. Da mesma forma da tela de *Login*, a tela de cadastro se comunica com o *Firebase* a partir do *AuthContext.Provider*, ele envia um sinal com os dados de entrada do usuário para o *Firebase* e, a partir disso, o *Firebase* registra essas credenciais enviadas no banco de dados. As Figuras a seguir refletem as informações e regras das credenciais registradas no *Firebase*.

As figuras de 11 a 15 referem-se às tabelas de regra de autenticação. A Tabela exibida na figura 11 é a mais importante em relação à autenticação do usuário no *Firebase*, a partir dela é possível visualizar dados como identificador, data de criação, último *login* e UID do usuário, além de permitir a exclusão de um usuário diretamente no *Firebase*.

Figura 11 - Regra de autenticação I

Identificador	Provedores	Data de criação	Último login	UID do usuário
dkzeiraagames@gmail.com		21 de set. d...	4 de nov. de...	6yOQXPV0IAXfw7QgEM6bvzUDfF...

Fonte: Próprio Autor

Na figura 12 são exibidos os provedores de login. No caso do Viver *Bike*, o provedor escolhido para autenticação foi e-mail e senha, porém o *Firebase* dispõe de outros mecanismos de autenticação como, por exemplo, autenticar a partir de uma conta Google ou mesmo uma conta *Facebook*.

Figura 12 - Regra de autenticação II

Provedor	Status
E-mail/senha	ativado

Fonte: Próprio Autor

Já na figura 13, é possível adicionar domínios autorizados de acesso a API de autenticação. Por *default*, os domínios *localhost*, “nomedaaplicacao.firebaseio.com” e “nomedaaplicacao.web.app” já estão ativados, permitindo um acesso remoto à aplicação sem necessidade de configuração de servidor e tudo mais.

Figura 13 - Regra de autenticação III

Domínios autorizados ⓘ

[Adicionar domínio](#)

Domínio autorizado	Tipo
localhost	Default
chatdemo-f7c8e.firebaseio.com	Default
chatdemo-f7c8e.web.app	Default

Fonte: Próprio Autor

A opção da figura 14, quando selecionada, inibe o usuário de criar várias contas com o mesmo e-mail e como isso é uma regra genérica em boa parte das aplicações, ela já vem marcada como verdadeira por padrão.

Figura 14 - Regra de autenticação IV

Avançado

Uma conta por endereço de e-mail

Evita que os usuários criem várias contas usando o mesmo endereço de e-mail com provedores diferentes de autenticação. [Saiba mais](#) ⓘ

[Alterar](#)

Fonte: Próprio Autor

Por fim, esta opção, mostrada na figura 15, consta como mais uma forma de segurança que protege projetos contra abusos, limitando o número de acessos ao aplicativo com o mesmo IP da rede. Todas essas formas de segurança trazem um interesse maior na utilização do Firebase e maior confiabilidade no uso e compartilhamento de dados com a aplicação.

Figura 15 - Regra de autenticação V

Gerenciar cota de inscrição

Para proteger seu projeto contra abusos, limitamos o número de novas inscrições anônimas e com e-mail/senha feitas no seu app usando o mesmo endereço IP. É possível solicitar e programar alterações temporárias nesta cota.
Cota atual por hora: 100

[Alterar](#)

Fonte: Próprio Autor

A Figura 16 refere-se ao código principal da tela de cadastro, onde estão presentes os componentes da tela e possuem uma formação parecida com a da tela de *login*, pois as duas telas possuem lógicas parecidas.

Figura 16 - Código Componentes Tela de Cadastro

```
return (  
  <View style={styles.container}>  
    <Title style={styles.titleText}>Registro Viver Bike</Title>  
    <FormInput  
      LabelName="Email"  
      value={email}  
      autoCapitalize="none"  
      onChangeText={userEmail => setEmail(userEmail)}  
    />  
    <FormInput  
      LabelName="Senha"  
      value={password}  
      secureTextEntry={true}  
      onChangeText={userPassword => setPassword(userPassword)}  
    />  
    <FormButton  
      title="Cadastrar"  
      modeValue="contained"  
      LabelStyle={styles.loginButtonLabel}  
      onPress={() => register(email, password)}  
    />  
  </View>  
)
```

Fonte: Próprio Autor

De modo semelhante à tela de *Login*, já comentada anteriormente, todos os componentes aqui utilizados são os mesmos da tela de *login*, pois estes são componentes genéricos de inserção de dados, portanto possuem a mesma função de inserir um e-mail, senha e acionar a ação de cadastro a partir de um botão. A Figura 17 demonstra o botão de voltar a página anterior na tela de cadastro.

Figura 17 - Código Componentes Tela de Cadastro II

```
37     <TouchableOpacity
38       size={30}
39       style={styles.navButton}
40       color="#6646ee"
41       onPress={() => navigation.goBack()}>
42       <FontAwesomeIcon size={20} icon={faArrowLeft} />
43     </TouchableOpacity>
44   </View>
45 );
46 }
```

Fonte: Próprio Autor

Um detalhe que não está presente na tela de login é o componente *TouchableOpacity*. Este componente é um botão nativo do *React Native* que não possui interface visual, o que nos permite utilizar um ícone para clique. A função deste componente na tela de cadastro é ao clicado retornar para tela anterior que na ocasião é a própria tela de *login*. A expressão *navigation.goBack()* refere-se a função da biblioteca *React Navigation* que faz o retorno sempre à tela anterior.

Para a demonstração do ícone de retorno, foi utilizado o componente *Font Awesome Icon*, que possui um número ilimitado de ícones para desenvolvimento.

4.3. Tela inicial

A tela inicial da aplicação tem como objetivo direto demonstrar uma lista de salas públicas, criadas pelos próprios usuários para organização de pedaladas. A lista de salas é um dos sistemas mais importantes da aplicação e a linha principal para o onde o projeto deve crescer mais. Cada sala possui nome e ponto de partida da rota criada. No momento não há tanta complexidade na listagem das salas. A sala é criada

no banco de dados e esses dados da sala são consumidos por um “*FlatList*”, componente nativo do *React Native* que cria uma lista de dados a partir dos dados passados. A Figura 18 exibe a função para mapear dados do banco de dados do *Firebase*.

Figura 18 - Código Tela Inicial

```

1  import React, {useState, useEffect} from 'react';
2  import {View, StyleSheet, FlatList, TouchableOpacity} from 'react-native';
3  import {List, Divider} from 'react-native-paper';
4  import firestore from '@react-native-firebase/firestore';
5  import Loading from '../components/Loading';
6
7  export default function HomeScreen({navigation}) {
8    const [threads, setThreads] = useState([]);
9    const [loading, setLoading] = useState(true);
10
11    /**
12     * Fetch threads from Firestore
13     */
14    useEffect(() => {
15      const unsubscribe = firestore()
16        .collection('THREADS')
17        .orderBy('LatestMessage.createdAt', 'desc')
18        .onSnapshot(querySnapshot => {
19          const threads = querySnapshot.docs.map(documentSnapshot => {
20            return {
21              _id: documentSnapshot.id,
22              // give defaults
23              name: '',
24              ...documentSnapshot.data(),
25            };
26          });
27          setThreads(threads);
28
29          if (loading) {
30            setLoading(false);
31          }
32        });
33    });
34
35    /**
36     * unsubscribe listener
37     */
38    return () => unsubscribe();
39  }, []);

```

Fonte: Próprio Autor

O cabeçalho da função principal do componente segue o padrão das outras telas anteriores, por conta da utilização do *Framework Hook's*, como já citado anteriormente, passando à prop “*navigation*” como parâmetro.

Nas linhas 8 e 9 são declarados dois “*useStates*”, um referente à propriedade “*Thread*” e outro referente à “*Loading*”. O “*useState*” referente à “*Threads*” tem a finalidade de manipular dados vindos da *collection* “*Thread*” encontrada no *Firebase*

e o outro `useState` tem a finalidade de manipular um componente de carregamento, determinando quando este será renderizado ou não na tela.

Após a declaração destas funções, pode-se visualizar a declaração de outra função inerente ao *Framework "Hook's"*, o `useEffect`. Esta função tem como objetivo gerenciar os ciclos de vida de cada componente. Cada componente dentro do contexto do *React Native* possui um ciclo de vida, determinando em que momento aquele componente passa a ser renderizado, ou deixa de ser renderizado na tela. O `useEffect`, no contexto da tela inicial da aplicação, determina que o trecho de código envolto por ele será disparado toda vez que o componente for renderizado, como o `useEffect` está presente no componente pai do qual representa toda a tela inicial isso quer dizer que todas as vezes que a tela for iniciada ou reiniciada, aquele trecho de código irá acontecer novamente. O Código envolto pela função `useEffect`, no contexto da tela inicial, tem o objetivo de percorrer todos os itens dentro da collection `Thread` no Firebase, retornar esses itens criando um objeto com `id` e `nome`. Um objeto no contexto do *Javascript* é uma coleção de dados ou funcionalidades relacionadas que normalmente consistem em variáveis e funções.

Por fim, este objeto é atribuído ao `useState`, `Threads` podendo ser manipulado ou mesmo atribuído a outro componente, exibindo os valores de `nome` e `id`, além disso, da linha 30 a 32 existe uma condição para que, se a propriedade `loading` for verdadeira, torna-se então falsa, pois o processo do qual necessitava de carregamento não necessita mais.

A propriedade retorno do componente principal da tela inicial reflete a lista de salas a partir do componente nativo do *React Native* chamado `FlatList`. O Objetivo do `FlatList` é gerar uma lista vertical pré-estilizada a partir de uma estrutura de dados `Array`. No Caso do *Viver Bike*, o `Array` está atribuído ao `state Threads` manipulado no `useEffect` do componente. O `FlatList`, então, a partir de sua props `data`, recebe esse `Array` e desestrutura o objeto de cada elemento do `Array`. Com o objeto desestruturado, o acesso aos itens do array de objetos torna-se direto e, com isso, as outras props do `FlatList` podem consumir esses dados a partir de um `callback` simples. O `keyExtractor`, por exemplo, acessando `item.id` pode definir para cada elemento da lista o seu respectivo `id`. Já o `ItemSeparatorComponent` cria linhas divisoras entre os elementos da lista. O `renderItem` é o responsável por renderizar os itens na lista. Para acessar as salas, cada uma delas necessita ter um botão invisível. Baseado

neste escopo, o “*TouchableOpacity*” permite que toda a área de cada item da lista seja clicável. No momento que o botão for clicado, a aplicação irá navegar para a tela principal da sala que ele escolheu e irá enviar o *array* “*Thread*” do índice selecionado como parâmetro para dentro da sala, podendo consumir esses dados dentro da sala. Por fim, as outras props ligadas apenas à lista são demonstradas dentro do “*List.Item*” e a estilização dos componentes segue a mesma ideia das outras telas.

4.4. Modal para criação de salas

Além da listagem de salas na tela inicial, o Viver *Bike* permite a criação de novas salas públicas com novas rotas para pedalar em grupo. No *Header* da tela inicial é possível localizar um ícone de criação de novas salas. Ao clicar nesse ícone o usuário é levado à tela de criação de sala.

Os componentes presentes na tela de criação de salas são: *Title*, campo para entrada do nome da sala, campo para pesquisa de ponto de origem e campo para pesquisa de ponto de destino. O Componente *Title* tem a função de definir o título da tela, já o campo de entrada do nome da sala definirá o nome da sala criada. Os campos de pesquisa de ponto de origem e ponto de destino têm a função de determinar o ponto de origem e destino da rota que será realizada pelos ciclistas e, por fim, o botão criar, que ao ser acionado dispara a ação para criar a sala desejada.

Ainda sobre os campos de origem e destino para criação de rotas, é importante ressaltar que os mesmos são os primeiros componentes a utilizarem uma das APIs do *GoogleMaps*, o *GooglePlacesAutoComplete*. O *GooglePlacesAutoComplete* proporciona o acesso a uma *bigdata* global de localizações no mundo todo e, no momento que o usuário começa a digitar o ponto de origem ou destino que deseja procurar, a API gera uma lista de sugestões com nomes parecidos ao que o usuário deseja procurar, facilitando a localização ao usuário e facilitando a criação de rotas.

Por fim, igualmente à tela de cadastro, a tela de criação de sala possui um botão no canto superior direito que permite a navegação para a tela anterior.

A respeito do código da tela, é possível visualizar alguns trechos nas figuras 19 e 20.

Figura 19 - Código Tela de Criação de Salas I

```

1  import React, {useEffect, useState} from 'react';
2  import {
3    View,
4    StyleSheet,
5    Dimensions,
6    Text,
7    TouchableOpacity,
8  } from 'react-native';
9  import {Title} from 'react-native-paper';
10 import FormInput from '../components/FormInput';
11 import FormButton from '../components/FormButton';
12 import firestore from '@react-native-firebase/firestore';
13 import {GooglePlacesAutocomplete} from 'react-native-google-places-autocomplete';
14 import {FontAwesomeIcon} from '@fortawesome/react-native-fontawesome';
15 import {faArrowLeft} from '@fortawesome/free-solid-svg-icons';

```

Fonte: Próprio Autor

Este trecho de código semelhante ao das telas anteriores demonstra as importações de bibliotecas para sua utilização no escopo desta tela específica. Neste contexto, existem duas bibliotecas de suma importância, são elas: *GooglePlacesAutoComplete* e *Firestore*. O *GooglePlacesAutoComplete*, como já citado anteriormente, tem o objetivo de sugerir localizações a partir de uma pesquisa, e o *Firestore* é a biblioteca do serviço de *database* do *Firebase*, um banco de dados NoSql, que significa que este banco é um banco não relacional onde os dados não estão presentes em tabelas e sim em coleções, trabalhando em paralelo e tempo real. Isso possibilita o acesso simultâneo de usuários aos mesmos dados presentes na sala.

Figura 20 - Código Tela de Criação de Sala II

```

17  const {width, height} = Dimensions.get('screen');
18
19  export default function AddRoomScreen({navigation}) {
20    const GOOGLE_MAPS_API = 'AIzaSyC0bIi8KWIfm5ZaMIGmU2fU4csCd6-RuDU';
21    const [roomName, setRoomName] = useState('');
22    const [coordinateOrigem, setcoordinateOrigem] = useState({
23      lat: -16.829168407265488,
24      lng: -49.536632461742435,
25    });
26    const [coordinateDestination, setcoordinateDestination] = useState({
27      lat: -16.829168407265488,
28      lng: -49.536632461742435,
29    });

```

Fonte: Próprio Autor

Na linha 17 da figura 20, pode-se visualizar uma função diferente das outras telas, a função `Dimensions.get('screen')`. Esta função retorna os valores de altura e largura da tela e é muito utilizada quando é necessário determinar o tamanho de algum componente baseado na resolução do celular do usuário para que aquele componente seja responsivo e seja apresentado da mesma forma em qualquer resolução. O cabeçalho do componente principal continua seguindo o padrão das outras telas.

Na linha número 20, existe uma atribuição diferente das outras telas, uma constante chamada "GOOGLE_MAPS_API" e uma chave atribuída a essa constante. Esta chave é a `API_KEY` do *Google Maps*, é por meio dela que se pode acessar aos serviços do Google e consumi-los. Cada usuário ao ativar um serviço gera uma *key* diferente.

Por fim, a função *GooglePlacesInputOrigin* demonstra o componente de pesquisa de localização citado anteriormente. O *GooglePlacesInputDestino* tem o mesmo código de origem, porém com os dados de latitude e longitude, de acordo com o local de destino. O retorno dos componentes principais da tela segue o mesmo padrão das outras telas.

4.5. Telas dentro da sala escolhida

Esta seção demonstra como funcionam as três telas presentes nas salas criadas pelos usuários. Como as três salas estão ligadas pelo mesmo modo de navegação, logo as três telas serão apresentadas no mesmo módulo. As telas aqui presentes são: Tela do mapa com rota, tela de chat e tela de apoio.

A primeira tela, ao acessar uma sala, é a tela de mapa com a rota que os usuários irão fazer. Um ponto é de entrada e outro ponto de destino. Este trajeto é determinado no momento de criação da tela e é refletido no mapa no momento de acesso.

No canto inferior direito é possível visualizar os detalhes da rota, com distância e tempo de trajeto.

A distância entre um ponto e outro da rota é dada em Quilômetros e a duração é dada em minutos.

A API utilizada para a demonstração da rota é a Directions API do Google. Essa API permite a criação de rotas a partir de um ponto de origem e um ponto de destino passados como props para o componente `MapViewDirections`.

No contexto do Viver Bike, os usuários terão acesso a essa rota, irão se encontrar no ponto de origem e, a partir de lá, irão ter acesso a toda rota até chegarem no ponto de destino.

Em relação ao código da tela do mapa é importante se atentar ao fato de que os dados de latitude, longitude e nome da sala são consumidos a partir do banco de dados, possibilitando que cada sala tenha seus dados únicos.

“`LocalizacaoDestino`” e “`LocalizacaoOrigem`” são os objetos ligados à latitude e à longitude do ponto de origem e destino, determinados na tela de criação de sala. Estas duas propriedades são geradas a partir do `GooglePlacesAutoComplete`. No momento que é selecionado o local de origem e destino, os dados de latitude e longitude são gerados. Com isso basta apenas manipular esses dados, tornando-os objetos e persistindo-os no banco de dados.

O trecho de código apresentado na figura 21 demonstra a lógica em torno do componente principal do mapa, o `MapView`.

Figura 21 - Código Tela de Criação de Sala III

```

51 function GooglePlacesInputOrigin() {
52   return (
53     <GooglePlacesAutocomplete
54       styles={searchInputStyle}
55       minLength={2}
56       autoFocus={false}
57       renderDescription={row => row.description}
58       GooglePlacesDetailsQuery={{fields: 'geometry'}}
59       fetchDetails={true}
60       placeholder="Search"
61       query={{
62         key: GOOGLE_MAPS_API,
63         language: 'en', // language of the results
64       }}
65       onPress={(data: any, details: any = null) => {
66         console.tron.log('data', data);
67         console.tron.log('details', details);
68         console.tron.log(JSON.stringify(details?.geometry?.location));
69         setcoordinateOrigem(
70           details?.geometry?.location
71             ? details?.geometry?.location
72             : {
73               latitude: -16.829168407265488,
74               longitude: -49.536632461742435,
75             },
76         );
77       }}
78       onFail={error => console.error(error)}
79     />
80   );
81 }

```

Fonte: Próprio Autor

As props mais importantes dentro do *MapViewDirections* são as props de *origin* e *destination*, pois é a partir delas que o componente consegue comparar as latitudes e longitudes e determinar uma rota entre dois pontos.

Os componentes *Markers* são os componentes responsáveis por gerarem os marcadores nos respectivos pontos de origem e destino e por fim a props *onReady* quando o componente é criado e está pronto para ser utilizado os valores de distância e duração são retornados.

E, por fim, para que o componente seja refletido na tela há a necessidade de que a chave da API seja passada na props “apikey”.

A segunda tela que o usuário pode acessar a partir das salas é a tela do *chat*. A partir desta tela, os usuários podem se comunicar com os outros ciclistas que irão

participar do Pedal, tirar dúvidas e interagir, possibilitando o início de novos círculos sociais.

Os itens presentes na tela do *chat* são: a área de inserção de mensagens, a mensagem enviada com o remetente da mensagem junto a uma imagem do usuário se o mesmo possuir imagem registrada.

O componente presente na tela de *chat* é o componente “*GiftedChat*”. Este componente foi previamente construído e consumido por meio da biblioteca “*react-native-gifted-chat*”.

Por fim, a última tela presente na aplicação diz respeito ao suporte ao usuário. Por meio dessa aba o usuário tem acesso aos serviços básicos mais rápido e em caso de problemas com a aplicação o e-mail do suporte técnico da aplicação também está presente.

Caso haja algum problema durante o trajeto, acidente ou erro na aplicação o usuário tem acesso rápido ao contato dos serviços de urgência, trazendo uma maior segurança para os ciclistas com a utilização do *app*.

O código que demonstra esta tela utiliza apenas de componentes básicos de escrita sem nenhuma complexidade no seu desenvolvimento.

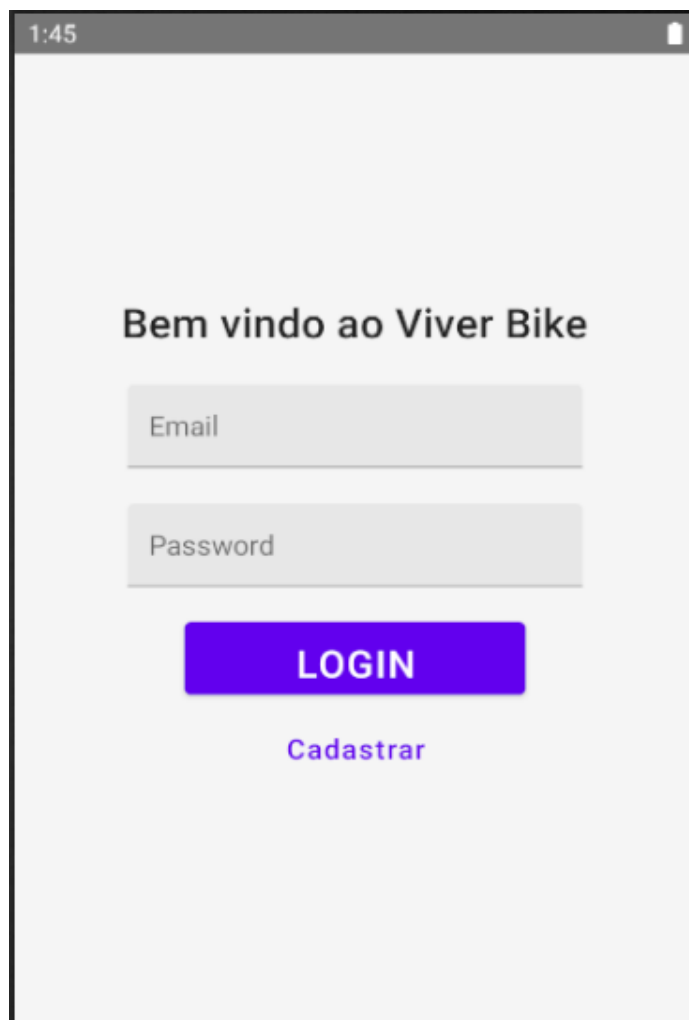
Os componentes apresentados na tela são: *Title*, *Text* e *FontAwesomeIcon*. O Componente “*Title*”, como já dito anteriormente, tem a função de definir o título da tela, neste caso, “Contatos - Emergência” e “Contato - Suporte Técnico”, já os componentes “*Text*”, definem o texto encontrado na tela, no caso, os números de telefone de cada serviço e o e-mail do suporte técnico. Por fim, o *FontAwesomeIcon* é um componente consumido a partir de uma biblioteca que permite a geração de ícones para casos específicos.

5. RESULTADOS

Este módulo tem como objetivo demonstrar os resultados do trabalho, particularmente, no caso deste trabalho, será demonstrado os resultados da aplicação e o impacto da aplicação na rotina dos ciclistas. A forma de demonstração será a partir de figuras que expressam o estado que a aplicação se encontra e as funcionalidades já implementadas.

A figura 22 demonstra a tela de *Login*, para autenticação do usuário.

Figura 22 - Tela de Login



1:45

Bem vindo ao Viver Bike

Email

Password

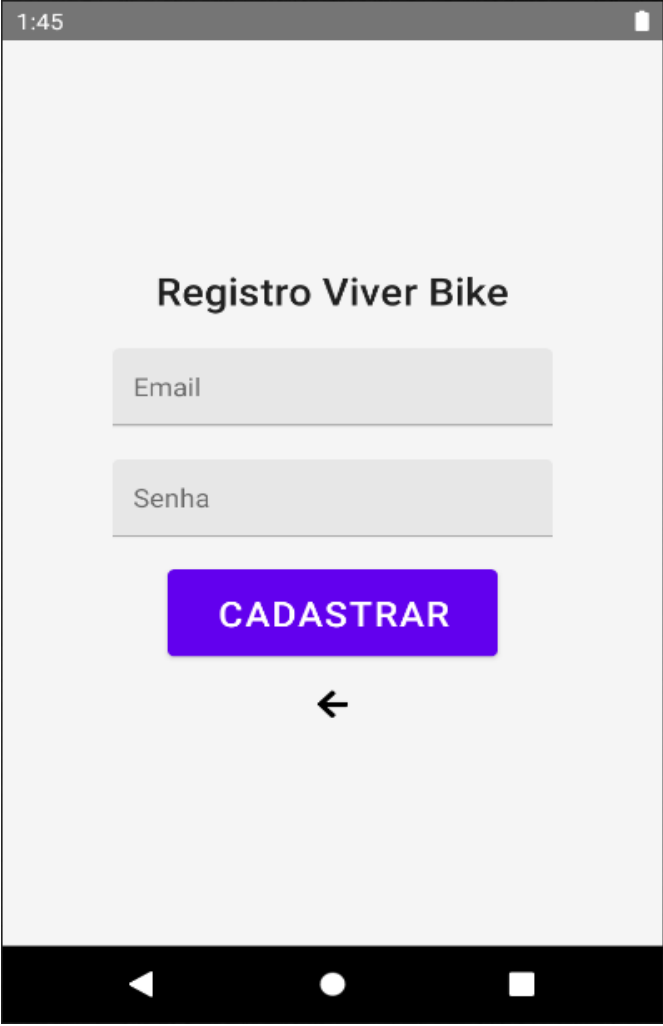
LOGIN

Cadastrar

Fonte: Próprio Autor

Já a figura 23 demonstra os componentes funcionais da tela de cadastro. Por meio desta tela o usuário tem permissão de acesso a aplicação.

Figura 23 - Cadastro Viver Bike

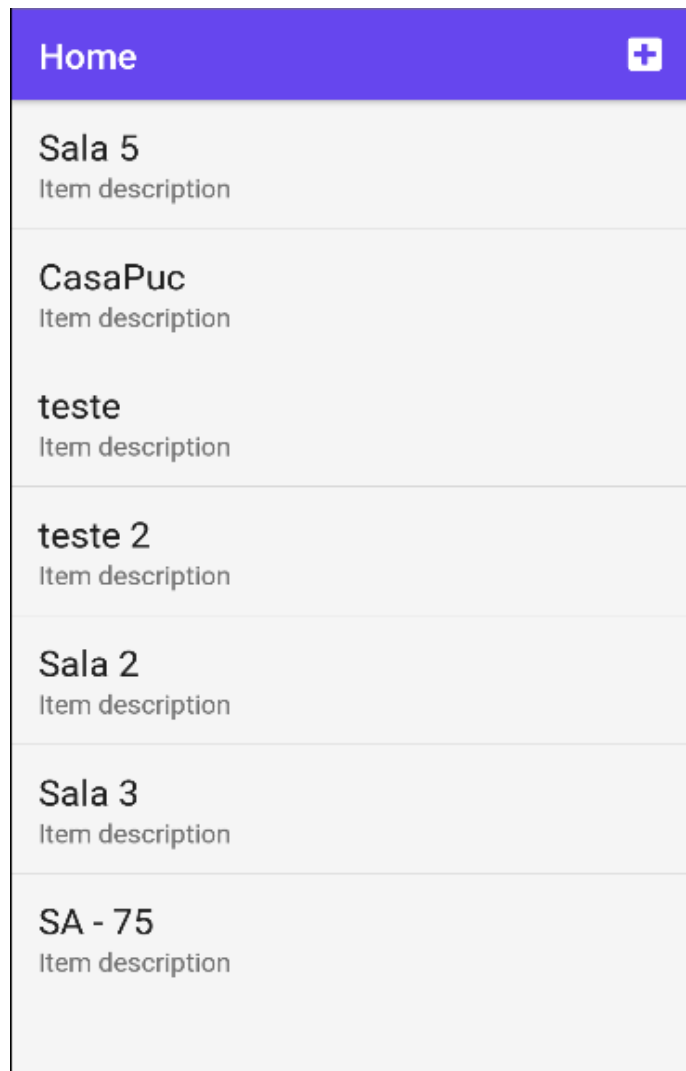


The image shows a mobile application registration screen. At the top, the status bar displays the time 1:45 and a battery icon. The main content area has a light gray background. The title 'Registro Viver Bike' is centered. Below the title are two input fields: 'Email' and 'Senha'. A prominent purple button with the text 'CADASTRAR' is centered below the input fields. Below the button is a black left-pointing arrow. At the bottom of the screen, the Android navigation bar is visible with its standard icons.

Fonte: Próprio Autor

A Figura 24 demonstra a tela inicial da aplicação, onde é exibida a lista de salas acessíveis ao usuário. No canto superior direito é demonstrado o botão de acesso à tela de criação de novas salas.

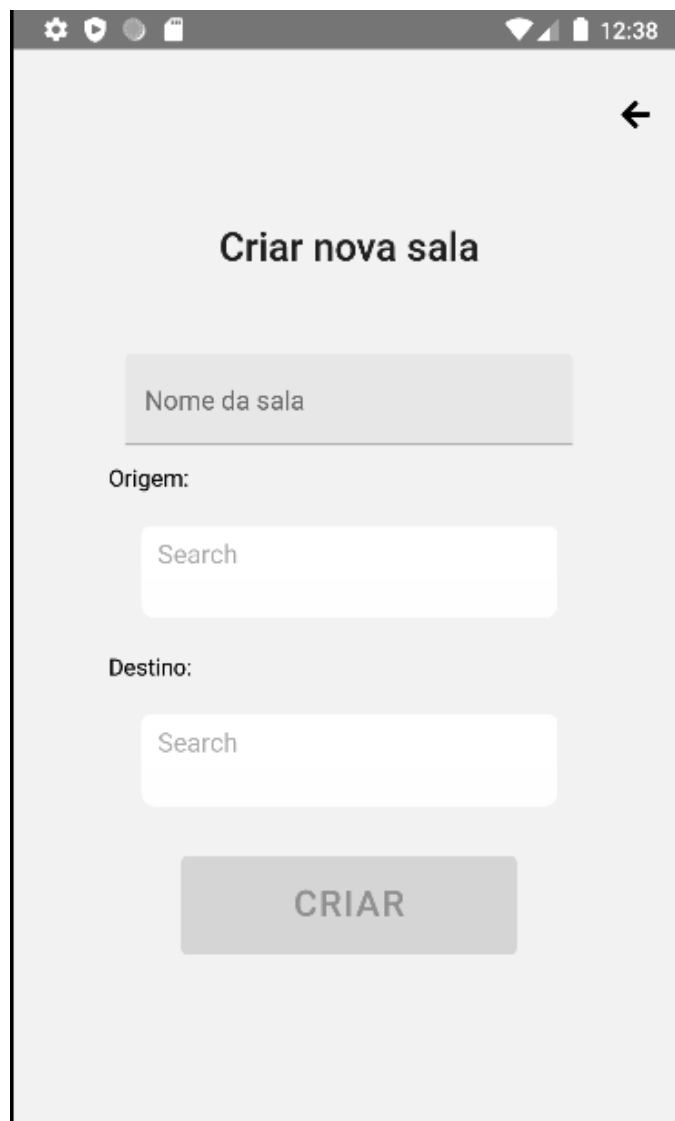
Figura 24 - Tela Inicial



Fonte: Próprio Autor

A tela demonstrada na figura 25 possibilita a criação das salas as quais os usuários irão acessar. A partir de um ponto de origem e um ponto de destino definidos pelo administrador da sala, esta rota será desenhada na primeira tela a acessar a sala, permitindo que os usuários tenham uma orientação em relação à rota.

Figura 25 - Tela de Criação de Sala



Nome da sala

Origem:

Search

Destino:

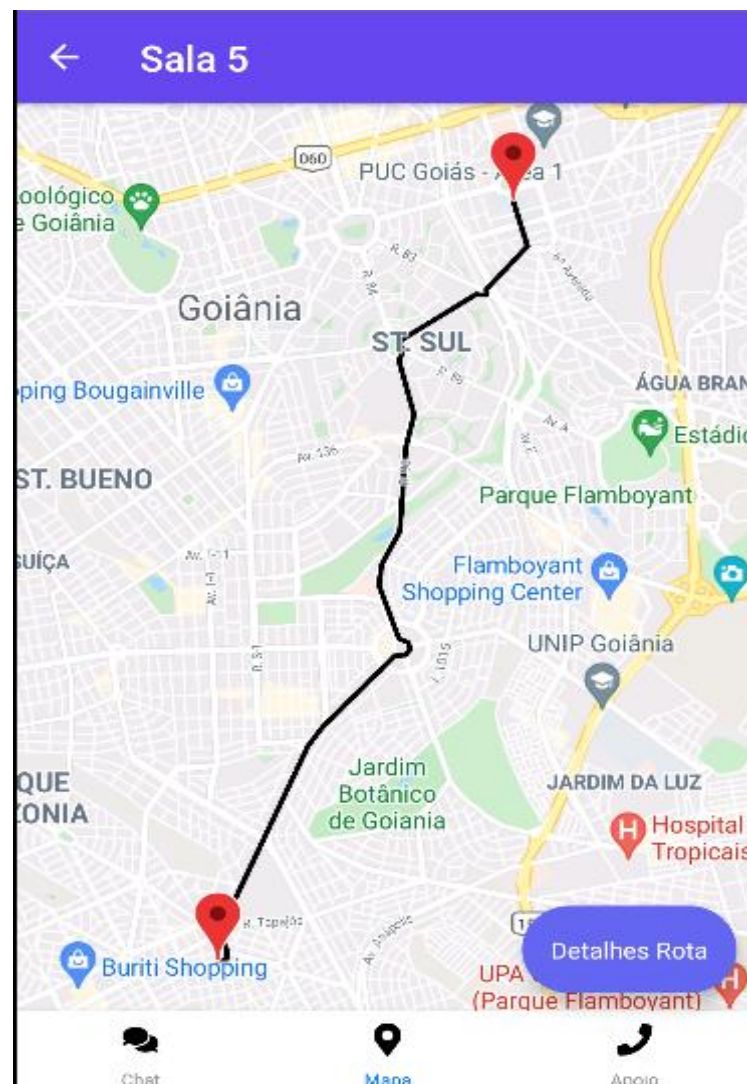
Search

CRIAR

Fonte: Próprio Autor

A figura 26 demonstra que o usuário, ao acessar a sala, pode se orientar a partir de uma rota pré-determinada, com o ponto de origem e de destino pelo criador da sala. O usuário também pode acessar os detalhes da rota que o grupo irá fazer, clicando no botão de Detalhes da rota no canto inferior direito. Ao acessar o *pop-up* de detalhamento de rotas, o usuário terá acesso a tempo de duração da rota e distância média.

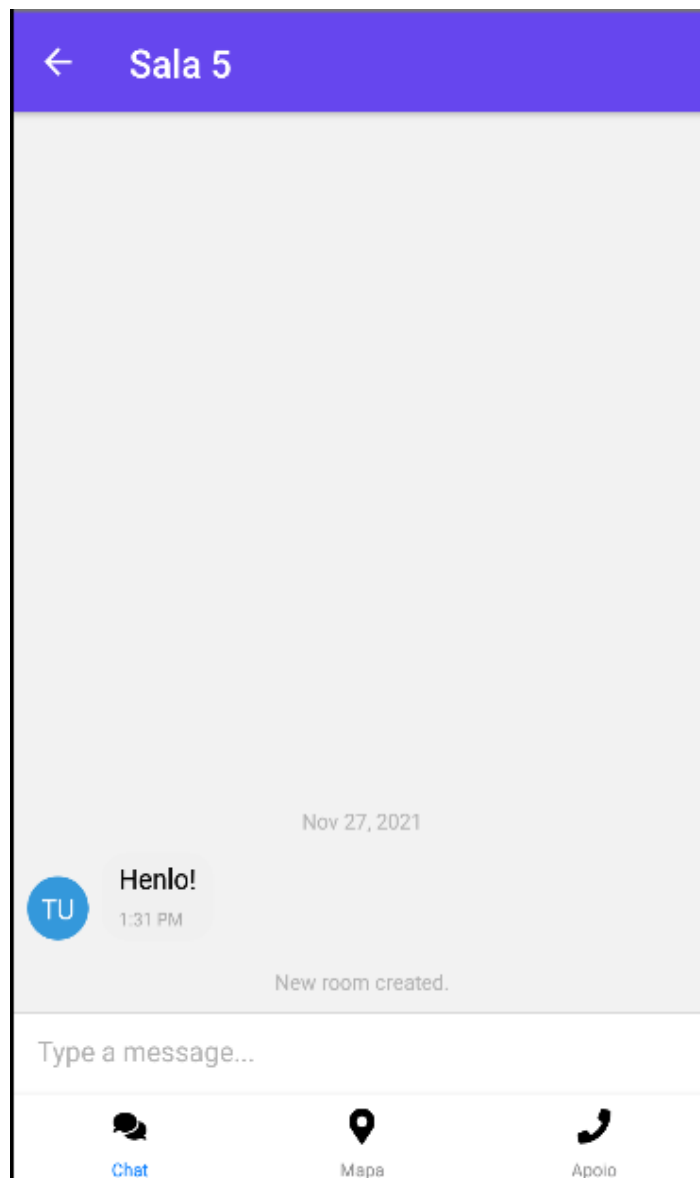
Figura 26 - Tela do Mapa com Rota



Fonte: Próprio Autor

A tela de chat na figura 27 permite a interação entre os usuários, ela possui uma interface de fácil utilização e aumenta a proximidade entre os usuários do aplicativo.

Figura 27 - Tela do Chat



Fonte: Próprio Autor

O Resultado encontrado na tela 28 são informações de apoio ao usuário e acesso rápido a informações importantes ao usuário.

Figura 28 - Tela de Apoio



Fonte: Próprio Autor

6. CONSIDERAÇÕES FINAIS

O Projeto apresentado conseguiu concluir com êxito o objetivo proposto. A ideia da aplicação partiu do autor do trabalho, mediante o levantamento de requisitos com ciclistas que sentem a necessidade de mais inclusão virtual e meios sociais para movimentar novos grupos de ciclistas. Em meio a seu desenvolvimento, inúmeros desafios foram encontrados, que precisavam ser vencidos para seu desenvolvimento inicial.

Alguns dos desafios foram durante o desenvolvimento da aplicação em relação à criação do mapa com rotas. Houve desafios também na parte visual do projeto por conta da priorização no funcionamento das features do sistema. Em relação à parte de levantamento de requisitos, também foram apresentadas algumas dificuldades, pois os usuários entrevistados pelo autor já se utilizavam de outras ferramentas de suporte para suas pedaladas, porém uma melhor opção para trabalhar com a comunidade e a parte social ainda era uma dor do grupo. Felizmente todos os contratempos foram solucionados, permitindo a finalização do projeto.

Após a apresentação de todo o projeto, pode-se determinar vantagens e desvantagens na sua utilização. A primeira vantagem na sua utilização, que deve ser levantada, é o fácil acesso à aplicação e sua usabilidade simples, pois com poucos cliques o usuário consegue criar uma sala e começar a interagir com novos usuários. Uma segunda vantagem deve-se ao fato, que já foi discutido anteriormente no trabalho, onde sabe-se que a aplicação possibilita a interação dos membros por meio de chat, então questões que trazem dúvidas a um usuário podem ser resolvidas de forma simples, e os administradores da aplicação estarão atentos a qualquer problema na utilização da aplicação, a partir do e-mail de suporte técnico. Por fim, uma última vantagem que merece atenção é que a aplicação utiliza dos melhores recursos para desenvolvimento mobile possíveis atualmente, possibilitando uma fácil manutenção e construção de novos recursos para aplicação.

Como desvantagens, é válido citar que o projeto está em versão alpha, não possuindo todos os recursos que serão disponibilizados na versão final da aplicação.

Uma segunda desvantagem, e recurso que não foi implementado nesta versão inicial, foi a Interface de usuário proposto pelo autor, adotando uma interface mais simples e direta ao usuário. Outra desvantagem que o projeto possui em relação aos concorrentes é o fato de ser um projeto desenvolvido por apenas um programador, o que eleva o tempo da implementação de novos recursos.

Como citado anteriormente, o projeto está em fase inicial e, neste trabalho, foi apresentado apenas o núcleo central da aplicação. Portanto, existem inúmeras funções que podem ser pesquisadas e implementadas em trabalhos futuros. Dois casos que devem ser explorados em trabalhos futuros são: uma opção para deletar salas, porém o único perfil com essa capacidade deve ser o perfil responsável pela criação da mesma e, como pré-requisito para a implementação desta funcionalidade, é interessante a criação de perfis de usuários mais complexos com a opção de inserção de fotos e outros dados do usuário, assim como a melhor estilização da interface de usuário, tornando-o mais atraente ao usuário.

Todas as desvantagens apresentadas serão adotadas como projetos futuros para este trabalho.

REFERÊNCIAS

REACT NATIVE. **Learn once, write anywhere.** Disponível em <<https://reactnative.dev/docs/getting-started>>. Acesso em 22/03/2021.

STRAVA. Monitoramento de corridas e ciclismo. Disponível em <<https://www.strava.com/?hl=pt-BR>>. Acesso em 22/03/2021.

EDOOLS, Rafael Carvalho. **O que é gamificação e como ela funciona?**. Disponível em <<https://www.edools.com/o-que-e-gamificacao/#:~:text=sendo%20%E2%80%9Cgamificado%E2%80%9D.-,Gamifica%C3%A7%C3%A3o%20%C3%A9%20o%20uso%20de%20mec%C3%A2nicas%20e%20din%C3%A2micas%20de%20jogos,fora%20do%20contexto%20de%20jogos>>. Acesso em 22/03/2021

REDHAT, Interface de programação de aplicações. **O que é API?**. Disponível em <<https://www.redhat.com/pt-br/topics/api/what-are-application-programming-interfaces>>. Acesso em 23/03/2021.

UOL PEDALA, Diego Salgado. **Pesquisa: Ciclismo é apontado como esporte mais seguro durante a pandemia.** Disponível em: <<https://www.uol.com.br/carros/colunas/pedala/2020/11/06/pesquisa-ciclismo-e-apontado-como-esporte-mais-seguro-durante-a-pandemia.htm>>. Acesso em 13/04/2021.

BRCICLISMO, **Strava: O aplicativo mais querido dos ciclistas.** Disponível em: <<https://brcyclismo.com.br/aplicativo-strava-o-que-e-como-funciona/#:~:text=Strava%3A%20o%20aplicativo%20mais%20querido%20dos%20ciclistas&text=O%20app%20%C3%A9%20sem%20d%C3%BAvida,percorrido%20e%20muitas%20outras%20coisas>>. Acesso em 13/04/2021.

JORNAL DA USP, Vitória Pierre. **Ciclismo ganha destaque durante pandemia com aumento de adeptos e recorde de vendas.** Disponível em: <<https://jornal.usp.br/campus-ribeirao-preto/ciclismo-ganha-destaque-durante-pandemia-com-aumento-de-adeptos-e-recorde-de-vendas/>>. Acesso em 25/05/2021.

REVISTA BICICLETA, Escola de Bicicleta. **Pedalar pode combater a obesidade?**. Disponível em: <https://revistabicicleta.com/destaque/pedalar-pode-combater-a-obesidade/>. Acesso em 01/06/2021.

DEV MEDIA, Higor. **Projetando e criando aplicativos para dispositivos móveis.** Disponível em: <<https://www.devmedia.com.br/projetando-e-criando-aplicativos-para-dispositivos-moveis/30671/>>. Acesso em 04/12/2021.

TECMUNDO, Fábio Jordão. **História: a evolução do celular.** Disponível em: <<https://www.tecmundo.com.br/celular/2140-historia-a-evolucao-do-celular.htm/>>. Acesso em 04/12/2021

OLHAR DIGITAL, Rene Ribeiro. **Processadores mobile ja são tão rápidos quanto a maioria dos chips de PC.** Disponível em: < <https://olhardigital.com.br/2018/11/01/noticias/processadores-mobile-ja-sao-tao-rapidos-quanto-a-maioria-dos-chips-de-pc/>>. Acesso em 04/12/2021.

USE MOBILE, Taysa Bocard. **O que é SDK? Saiba sua utilidade no desenvolvimento mobile.** Disponível em: < <https://usemobile.com.br/sdk/>>. Acesso em 04/12/2021.

USE MOBILE, Marketeam. **Aplicativo nativo, web app ou aplicativo híbrido?** Disponível em: < <https://usemobile.com.br/aplicativo-nativo-web-hibrido/>>. Acesso em 04/12/2021.

CRONAPP, Redação Cronapp. **Compreenda as diferenças entre o desenvolvimento nativo e híbrido.** Disponível em: < <https://blog.cronapp.io/compreenda-as-diferencas-entre-o-desenvolvimento-nativo-e-hibrido/>>. Acesso em 04/12/2021.

ORGÂNICA NATURAL MARKETING, Lauro Becker. **O que é React Native?** Disponível em: < <https://www.organicadigital.com/blog/o-que-e-react-native/>>. Acesso em 04/12/2021.

PEDAL.COM.BR, Pedro Cury. **Strava – O que é e como usar essa rede social de ciclistas.** Disponível em: < https://www.pedal.com.br/strava-o-que-e-e-como-usar-essa-rede-social-de-ciclistas_texto8842.html/>. Acesso em 04/12/2021.

MAPLINK, Bruna Brandão. **Você sabe o que é API de integração? Entenda de uma vez por todas!** Disponível em: < <https://maplink.global/blog/o-que-e-api/>>. Acesso em 04/12/2021.

CANALTECH, Redação. **O que é API?** Disponível em: < <https://canaltech.com.br/software/o-que-e-api/>>. Acesso em 04/12/2021.

HEARTBEAT, Aman Mittal, **Chat app with React Native: Build reusable UI form elements using react-native-paper.** Disponível em < <https://heartbeat.comet.ml/chat-app-with-react-native-part-1-build-reusable-ui-form-elements-using-react-native-paper-75d82e2ca94f>>. Acesso em 04/12/2021

IMASTERS, William Souza, **Entendendo estado de componentes com React na prática.** Disponível em < <https://imasters.com.br/front-end/entendendo-estado-de-componentes-com-react-na-pratica/>>. Acesso em 15/12/2021