PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS
ESCOLA POLITÉCNICA
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

SOFTWARE DE SIMULAÇÃO DE ESTRATÉGIAS *TRADING* NA BOLSA DE VALORES

HÉRMESON BARROSO FREITAS

GOIÂNIA
2022/2

HÉRMESON BARROSO FREITAS

SOFTWARE DE SIMULAÇÃO DE ESTRATÉGIAS *TRADING* NA BOLSA DE VALORES

Trabalho de conclusão de curso apresentado a Escola de Politécnica, da Pontifícia Universidade Católica de Goiás, como parte dos requisitos para obtenção do título de Bacharel em Ciência da Computação.

Orientadora:
Profa. Ma. Angélica da Silva Nunes

Banca examinadora:
Prof. Me. Fernando Gonçalves Abadia
Prof. Dra. Solange da Silva

GOIÂNIA
2022/2

HÉRMESON BARROSO FREITAS

SOFTWARE DE SIMULAÇÃO DE ESTRATÉGIAS *TRADING* NA BOLSA DE VALORES

Este Trabalho de conclusão de Curso julgado adequado para obtenção do título de Bacharel em Ciência da Computação, e aprovado em sua forma final pela Escola Politécnica, da Pontifícia Universidade Católica de Goiás, em ____/____/_____.

_____.

Profa. Ma. Ludmilla Reis Pinheiro dos Santos Coordenadora de Trabalho de Conclusão de Curso.

Banca examinadora:

_____.

Orientadora: Profa. Ma. Angélica Silva Nunes.

_____.

Prof. Me. Fernando Gonçalves Abadia.

_____.

Prof. Dra. Solange da Silva.

GOIÂNIA

2022/2

RESUMO

Este trabalho teve como objetivo desenvolver um *software* criação de estratégias de investimento na bolsa de valores, são detalhados todos os aspectos do *software*, desde a motivação por sua construção, como pode ser desenvolvido e como pode ser utilizado pelo investidor para tomar decisões ao criar estratégias de investimento na bolsa de valores. Foi realizada também uma descrição sobre os princípios da análise técnica e quais tecnologias são ideais para a construção do *software*. Como resultado do desenvolvimento do *software*, obtém-se uma ferramenta útil para investidores iniciantes ou experientes que buscam realizar operações na bolsa de valores por meio da criação e testes de estratégias baseadas na análise técnica em dados históricos de negociação de ativos.

Palavras-chaves: Análise técnica, Bolsa de valores, *Trading*.

ABSTRACT

This work aims to develop a software for creating investment strategies in the stock market, all aspects of the software are detailed, from the motivation for its construction, how it can be developed and how it can be used by the investor to make decisions when creating investment strategies in the stock market. A description was also given about the principles of technical analysis and which technologies are ideal for building the software. As a result of the development of the software, a useful tool is obtained for beginners or experienced investors who seek to carry out operations on the stock exchange through the creation and testing of strategies based on technical analysis on historical asset trading data.

Keywords: Technical analysis, Stock Exchange, Trading.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS

API *Application Programming Interface*

CSV *Comma Separated Values*

IDE *Integrated Development Environment*

SUMÁRIO

1 INTRODUÇÃO

A bolsa de valores tem o propósito de gerar mais capital nas empresas, fornecendo parcelas da empresa a acionistas, que ao adquiri-los, tornam-se sócios da empresa, dividindo lucros e prejuízos, a primeira bolsa de valores oficial se originou em 1487, na Bélgica (NOVA ESCOLA, 2009).

Antigamente as ordens de compra ou venda de papéis na bolsa de valores eram executadas por meio do pregão viva-voz, com milhares de operadores reunidos em um salão para executarem as ordens dos acionistas, no entanto, esse sistema era sujeito a falhas, como erros realizados pelos operários (REIS, 2019).

Com o aumento de investidores na bolsa de valores, o pregão viva-voz se tornou impraticável, sendo necessário mover as atividades para um sistema *online*, pregão viva-voz da bolsa *Bovespa* teve fim em 2005 (REIS, 2019).

Com o sistema de operações na bolsa de valores funcionando de forma *online*, o número de investidores na bolsa de valores cresceu significativamente, chegando a dobrar no primeiro semestre de 2020, também se tornou possível a criação de robôs de investimento para realização de operações de compra e venda de forma automática para pessoas físicas (EINVESTIDOR, 2021).

Uma parcela dos investidores na bolsa de valores opta pela modalidade *day trade*, que consiste em operações de curto prazo, sendo finalizadas no mesmo dia, no entanto mais de 90% das pessoas físicas que tentaram viver de operações a curto prazo na bolsa de valores, obtiveram prejuízo (EINVESTIDOR, 2021).

Riscos de obter prejuízos na bolsa de valores podem ser minimizados realizando operações de teste em um simulador antes de realizar operações reais, apesar de haver ferramentas especializadas em desenvolvimento e testes de estratégias de *day trade*, como o *Metatrader 4*, essas ferramentas não são disponibilizadas em certas plataformas, como *smartphones*, e necessitam de conhecimento de programação, o que diminui a acessibilidade e facilidade de uso da ferramenta.

Diante do contexto apresentado, este trabalho pretende responder a seguinte questão:

- Como criar um *software* multiplataforma de desenvolvimento de estratégias de *trading*?

1.1 Objetivo Geral

- Desenvolver um *software* multiplataforma de desenvolvimento, otimização e teste de estratégias de *trading*.

1.2 Objetivos Específicos

- Aprofundar conhecimentos sobre estratégias de análise técnica;
- Aprofundar conhecimentos sobre automação de investimentos (robôs);
- Apresentar tecnologias de desenvolvimento multiplataforma;
- Apresentar as funcionalidades da Google Cloud Platform;
- Desenvolver a *interface* da aplicação;
- Realizar testes de estratégias.

1.3 Metodologia

Esta pesquisa quanto a natureza é um resumo de assunto, pois busca fazer uma análise de ferramentas da análise técnica e *kits* de desenvolvimento multiplataforma e de como utilizar essas ferramentas para o desenvolvimento do *software* de criação de estratégias de *trading* (WASLAWICK, 2014).

Quantos aos objetivos é uma pesquisa descritiva, pois busca-se coletar dados a respeito da realidade a ser estudada e analisar os dados por meio de experimentos sem que haja tentativa de explicá-los por meio de criação de teorias (WASLAWICK, 2014).

Quanto aos procedimentos técnicos, é uma pesquisa experimental, pois manipula variáveis a fim de obter melhores resultados quanto a utilidade do *software* (WASLAWICK, 2014).

1.4 Resultados Esperados

Esta pesquisa tem como resultado esperado apresentar o processo de desenvolvimento de um *software* que auxilie *day traders* a desenvolver, testar e otimizar estratégias de operações em diversos tipos de mercados da bolsa de valores.

1.5 Estrutura da Monografia

No Capítulo 2, é feita uma fundamentação teórica sobre os princípios da análise técnica.

No Capítulo 3, é feita uma fundamentação teórica sobre as tecnologias a serem empregadas no *software*.

No Capítulo 4, são descritos os passos para a implementação das tecnologias apresentadas na construção do *software*.

No Capítulo 5, são feitos testes do *software* e avaliação de seus resultados em ativos de *trading*.

2 ANÁLISE TÉCNICA DE INVESTIMENTOS

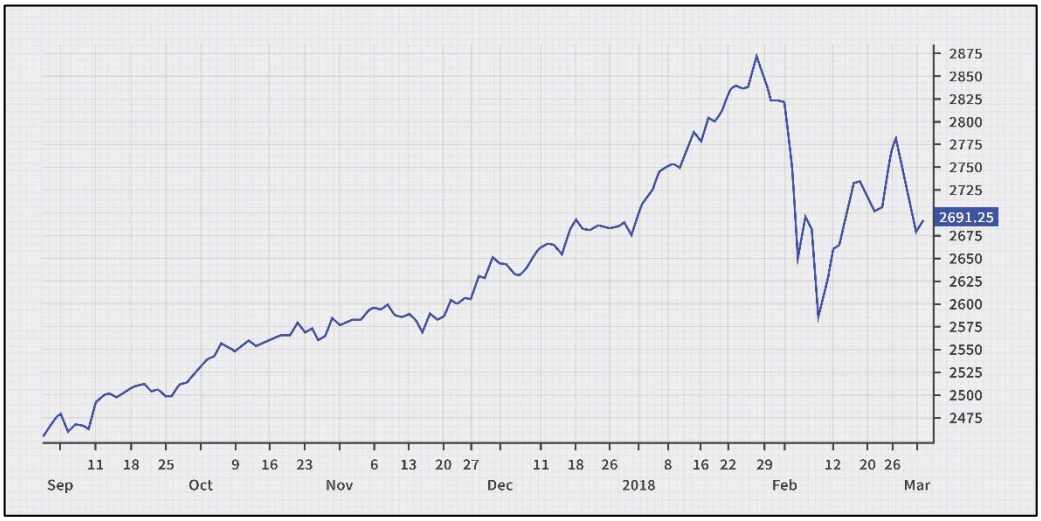
A análise técnica consiste em um conjunto de ferramentas utilizadas por *traders* na tentativa de prever prováveis movimentações de um ativo baseado em seus dados históricos, tornando assim possível ganho de capital por meio de operações de compra e venda (INVESTOPEDIA, 2022).

2.1 Gráficos

A principal ferramenta da análise técnica é o gráfico, nele é possível visualizar a variação do preço do ativo ao longo de uma escala temporal para buscar padrões que permitem a estimativa de uma movimentação (INVESTOPEDIA, 2022).

Existem vários tipos de gráficos conhecidos, o mais simples é conhecido como gráfico de linhas, que exibe apenas a variação do preço de fechamento do ativo relativo a uma escala de tempo, como horas, minutos ou segundos, conforme exibido na Figura 1 (KOTAK SECURITIES, 2005).

Figura 1 - Gráfico de Linhas



Fonte: INVESTOPEDIA, 2021.

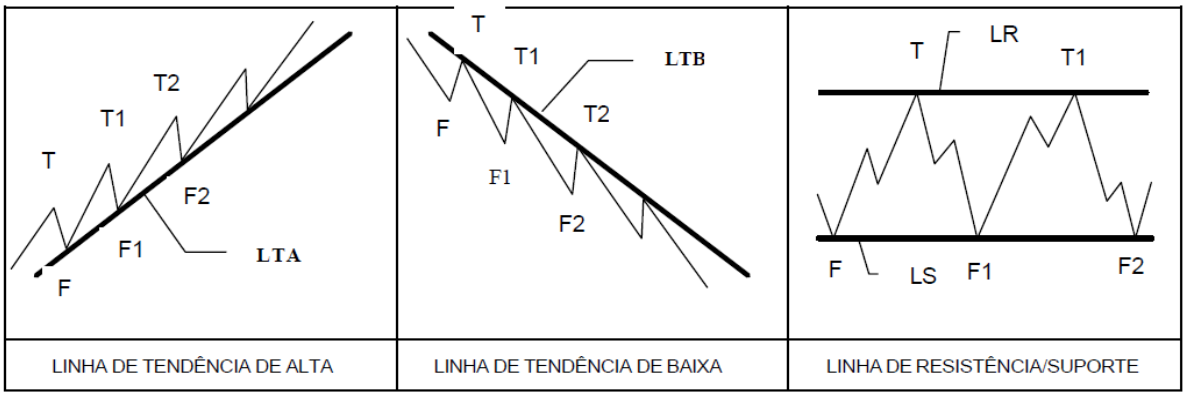
Outro tipo de gráfico bastante é o gráfico de *Candlestickets*. Ele é similar ao gráfico de linhas, mas também contém informações extras como preço de abertura / fechamento, e o preço máxima / mínimo de cada *ticket*, conforme exibido na Figura 2 (KOTAK SECURITIES, 2005).

Figura 2 - Gráfico de *Candlestickets*

Fonte: SWING *TRADING* WITH *CANDLESTICKET* PATTERNS, 2017.

Ao analisar qualquer tipo de gráfico que represente a variação do valor de um ativo, é possível identificar uma de três tipos de tendências: alta, baixa ou lateralizada. Cada tipo de tendência pode formar linhas de suporte, quando o valor se encontra consistentemente acima de uma linha ou resistência, quando o valor se encontra consistentemente abaixo de uma linha ou ainda em gráficos lateralizados, quando as linhas de suporte e resistência representam uma região consolidada também chamada de caixote, conforme exibido na Figura 3 (MORAES, 2016).

Figura 3 - Tipos de Tendência



Fonte: UNIVERSO DO FOREX, 2010.

2.2 Indicadores

Os indicadores são funções criadas com o propósito de fornecer algum tipo de percepção de um determinado ativo. Essa percepção é utilizada para tomar alguma decisão, como a compra ou venda do ativo. Portanto os indicadores são funções incompletas em relação a decisão de

compra ou venda, visto que é necessário que haja outra função que mapeia os indicadores para a ação final a ser realizada (IG, 2022).

2.2.1 Média Móvel Aritmética

Na média móvel aritmética cada ponto T no tempo é a média aritmética dos últimos N preços em relação ao ponto T, o indicador é calculado pela Equação 1 e exibido na Figura 4 (IG, 2022).

$$MMA = \frac{P1+P2+P3+\cdots+PN}{N} \quad (1)$$

Sendo que:

P = preço;

N = quantidade de períodos.

A Figura 4 exibe o comportamento da média móvel aritmética.

Figura 4 - Gráfico da média móvel aritmética

Fonte: TRADINGVIEW, 2011.

2.2.2 Média Móvel Exponencial

A média móvel exponencial é semelhante à média móvel aritmética, com a diferença que a média móvel exponencial reage mais rapidamente as mudanças recentes do preço. O indicador é calculado pela seguinte Equação 2 e exibido na Figura 5 (IG, 2022).

$$MME = PN * K + MME(PN - 1) * (1 - K) \quad (2)$$

Sendo que:

P = preço;

N = quantidade de períodos;

K = constante de suavização.

A Figura 5 exibe o comportamento da média móvel exponencial.

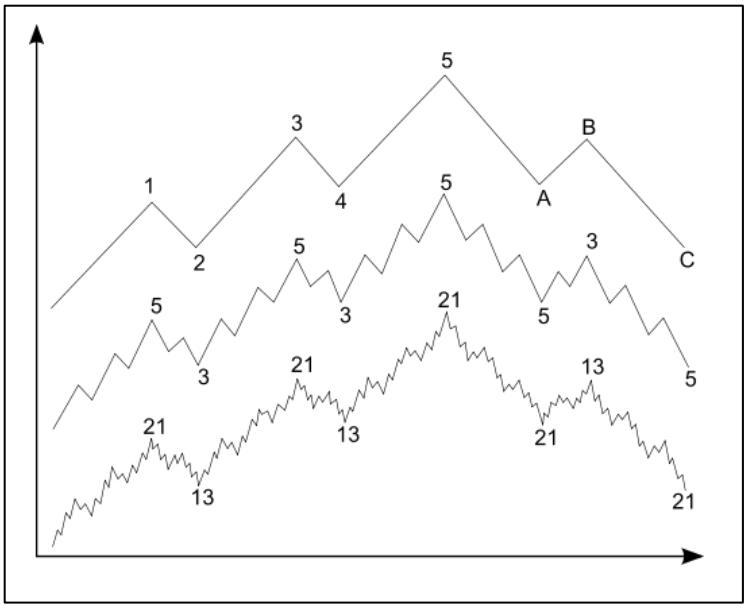
Figura 5 - Gráfico da média móvel exponencial

Fonte: TRADINGVIEW, 2011.

2.2.3 Ondas de Elliot

A teoria das ondas de Elliot pode ser utilizada em operações de reversão de tendência. Essa teoria afirma que o mercado se movimenta em ciclos que refletem a psicologia humana, com uma variação previsível de otimismo e pessimismo, a Figura 6 exibe o comportamento das ondas de Elliot (MORAES, 2016).

Figura 6 - Ondas de Elliot



Fonte: MASUR, 2007.

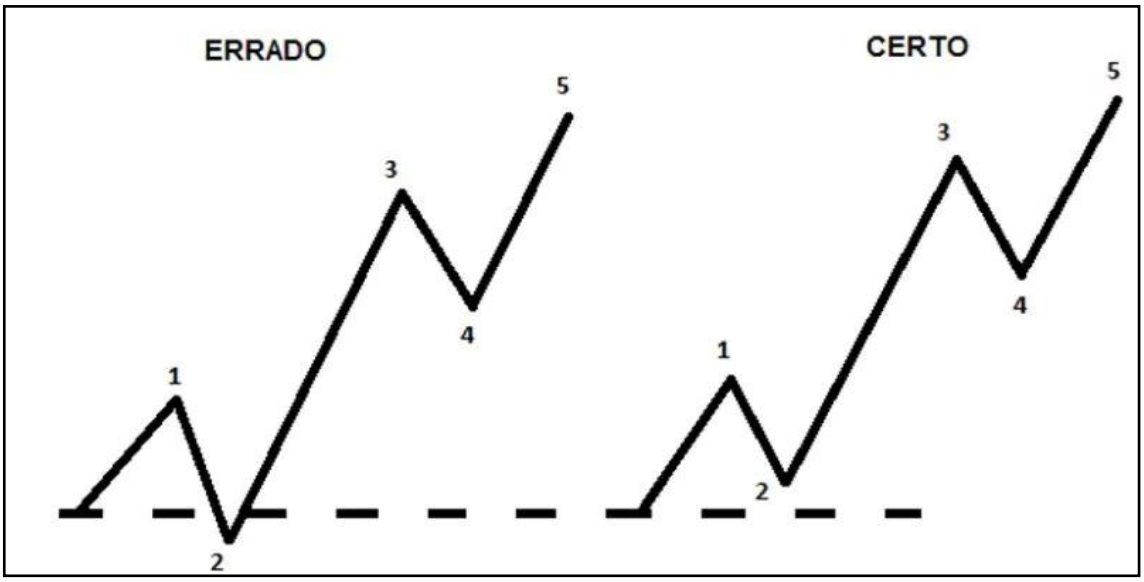
As ondas de Elliot são formadas por cinco ondas motivas (1, 2, 3, 4, 5) e três ondas corretivas (A, B, C):

- Onda 1: Caracterizada por uma possível reversão de tendência de baixa, nela os *traders* institucionais começam a se posicionar;
- Onda 2: Caracterizada por um comportamento de início de reversão de posições do mercado;
- Onda 3: Caracterizada por movimentos rápidos de reversão;
- Onda 4: Caracterizada por um mercado em alta, mas com volume baixo;
- Onda 5: Caracterizada por um mercado em alta, mas com volume consideravelmente baixo;
- Onda A: Caracterizada por uma forte queda;
- Onda B: Caracterizada por uma alta com volume baixo;
- Onda C: Caracterizada por uma nova queda.

As ondas de Elliot também tem regras que devem se manter verdadeiras para que o padrão realmente indique ondas de Elliot, que são descritas a seguir.

A regra 1 afirma que a onda 2 não deve ultrapassar o início da onda 1. Como exibido na Figura 7.

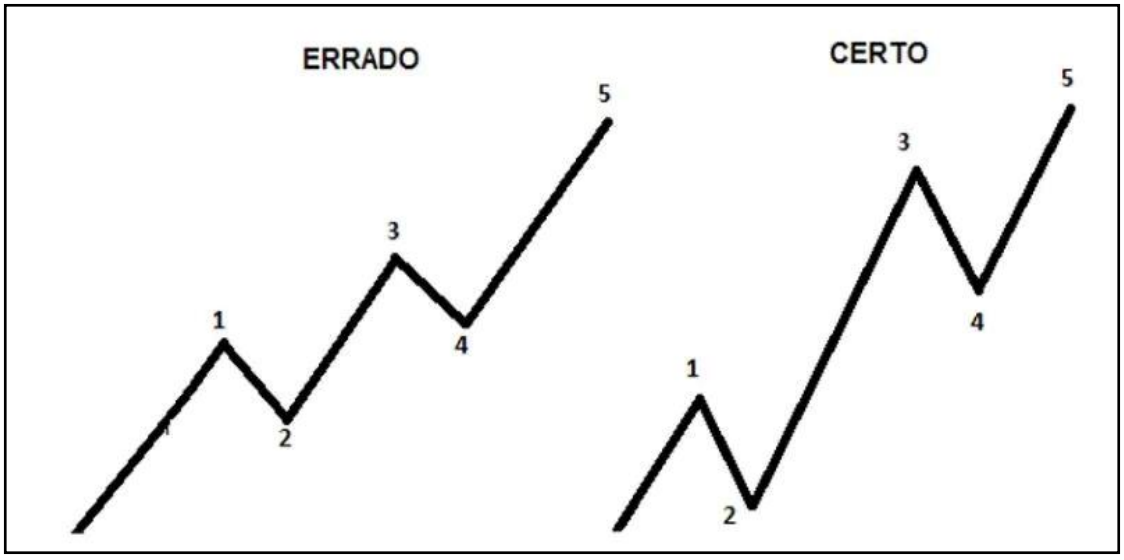
Figura 7 - Regra 1 das Ondas de Elliot



Fonte: PORTAL DO *TRADE*R, 2021.

A regra 2 afirma que a onda 3 não deve ser menor que as ondas 1, 3 e 5. Como exibido na Figura 8.

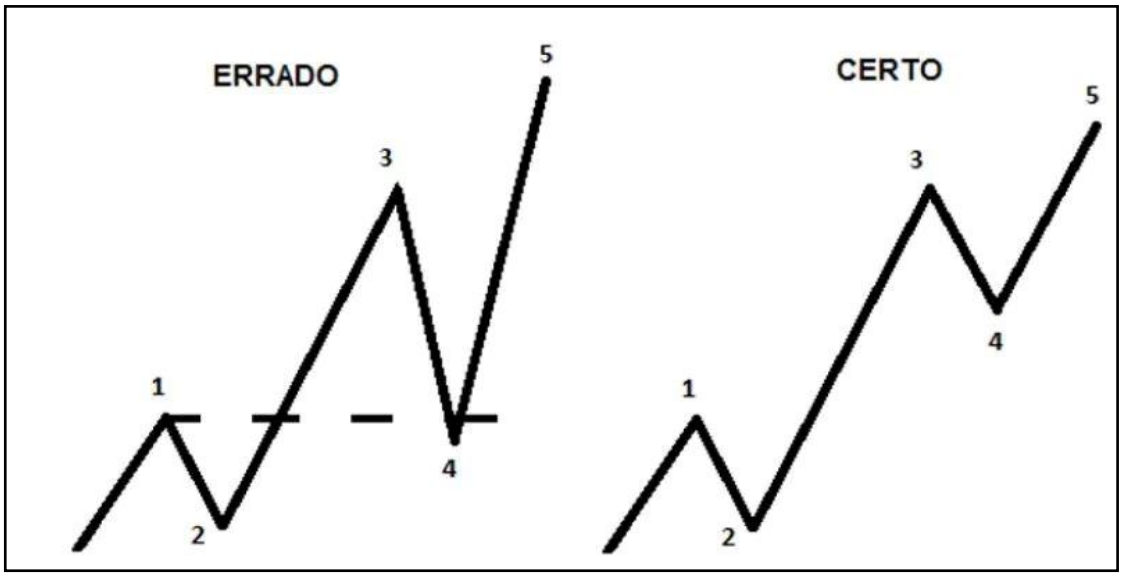
Figura 8 - Regra 2 das Ondas de Elliot



Fonte: PORTAL DO TRADER, 2021.

Na regra 3 deve-se observar se a onda 4 não está no nível da onda 1. Como exibido na Figura 9.

Figura 9 - Regra 3 das Ondas de Elliot



Fonte: PORTAL DO *TRADE*R, 2021.

2.2.4 Bandas de Bollinger

As bandas de Bollinger tem como objetivo indicar a volatilidade de um ativo em relação ao tempo. Esse indicador é composto por três curvas, uma superior, uma central e uma inferior, conforme exibido na Figura 10 (MORAES, 2016). O indicador é calculado da seguinte forma:

- Banda superior: média móvel aritmética (N períodos) + (2x desvio padrão de N períodos);
- Banda central: média móvel aritmética (N períodos);
- Banda inferior: média móvel aritmética (N períodos) – (2x desvio padrão de N períodos);

Figura 10 - Gráfico das Bandas de Bollinger

Fonte: SUNO, 2018.

2.3 Estratégias de *trading*

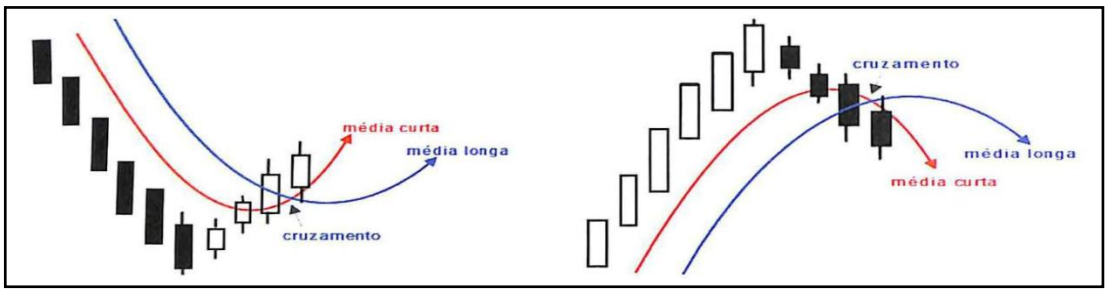
As estratégias de *trade* podem ser definidas como uma função que converte um conjunto de dados em uma decisão final que, geralmente, visa maximizar o lucro financeiro de uma operação. O conjunto de dados em um contexto de análise técnica, pode conter, por exemplo, um conjunto de gráficos e um conjunto de indicadores (MORAES, 2016).

2.3.1 Cruzamento da Média Móveis

A estratégia de cruzamento da média móveis é uma estratégia que utiliza como sinal de compra ou venda o evento de uma média móvel curta (como 5 ou 9 períodos) cruzar uma longa (como 21 períodos), conforme exibido na Figura 11. A escolha da quantidade ideal de períodos é um hiperparâmetro que deve ser avaliado de acordo com o ativo escolhido (FERNANDES, 2014).

Exemplo de sinal de compra ou venda: Quando a média móvel mais curta cruzar a média móvel mais longa para cima ou para baixo, a operação é realizada no fechamento do *candle* que provocou o cruzamento ou na abertura do *candle* seguinte (FERNANDES, 2014).

Figura 11 - Exemplo de estratégia que utiliza cruzamento de médias móveis



Fonte: FERNANDES, 2014.

2.3.2 Fechou Fora, Fechou Dentro

A estratégia “fechou fora fechou dentro”, é uma operação que utiliza as bandas de Bollinger para realizar uma operação de contra tendência (FERNANDES, 2014).

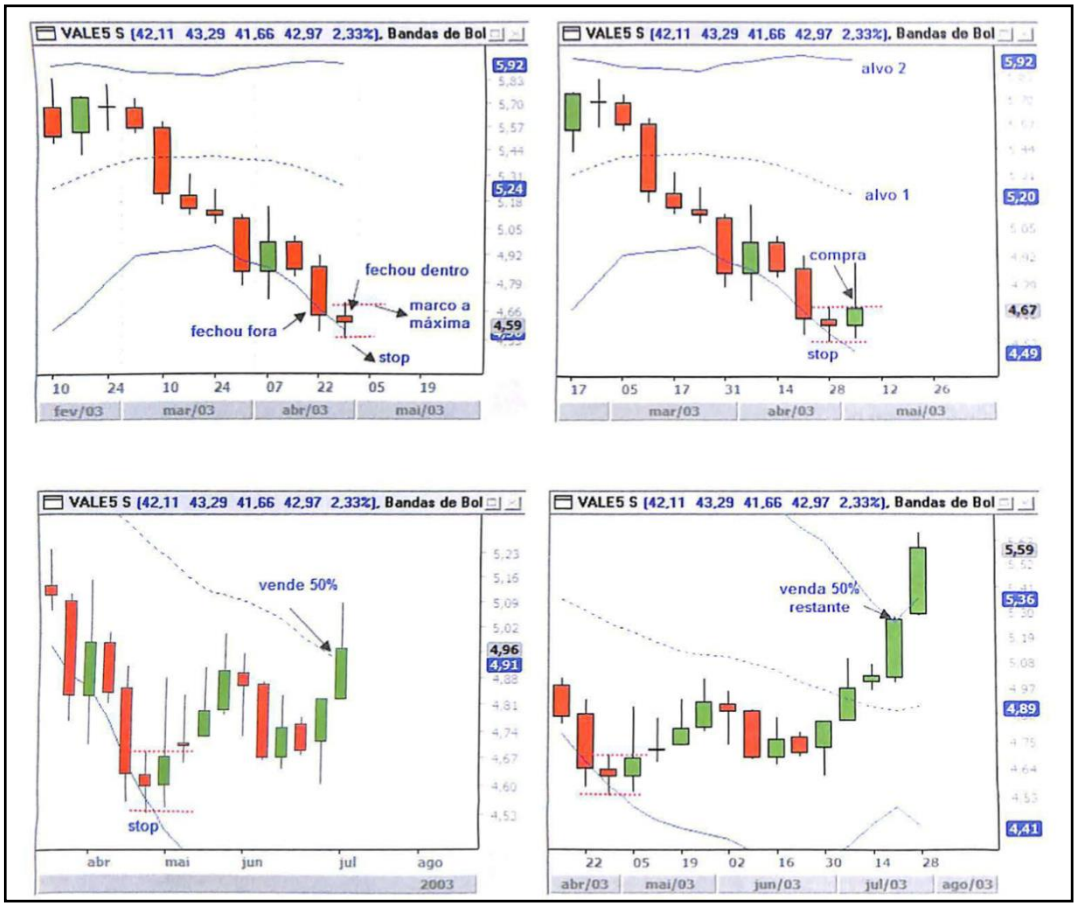
Essa estratégia possui algumas regras que são descritas a seguir:

- Esperar fechar um *candle* 10% fora de uma banda de Bollinger (superior ou inferior);
- Esperar o próximo *candle* fechar dentro da banda (banda superior - marca-se a mínimo do *candle*, banda inferior - marca-se a máxima do *candle*);
- Esperar, no máximo 2 *candle*s, o rompimento da mínima (ou máxima);

- Entrar (banda superior - vendido, banda inferior - comprado);
- Sair 50% na banda central e os outros 50% na banda oposta, se o preço chegar em qualquer banda sem passar pela banda central antes sair 100% (*stop loss*).

Um exemplo de aplicação da estratégia é exibido na Figura 12.

Figura 12 - Exemplo de operações de compra



Fonte: FERNANDES, 2014.

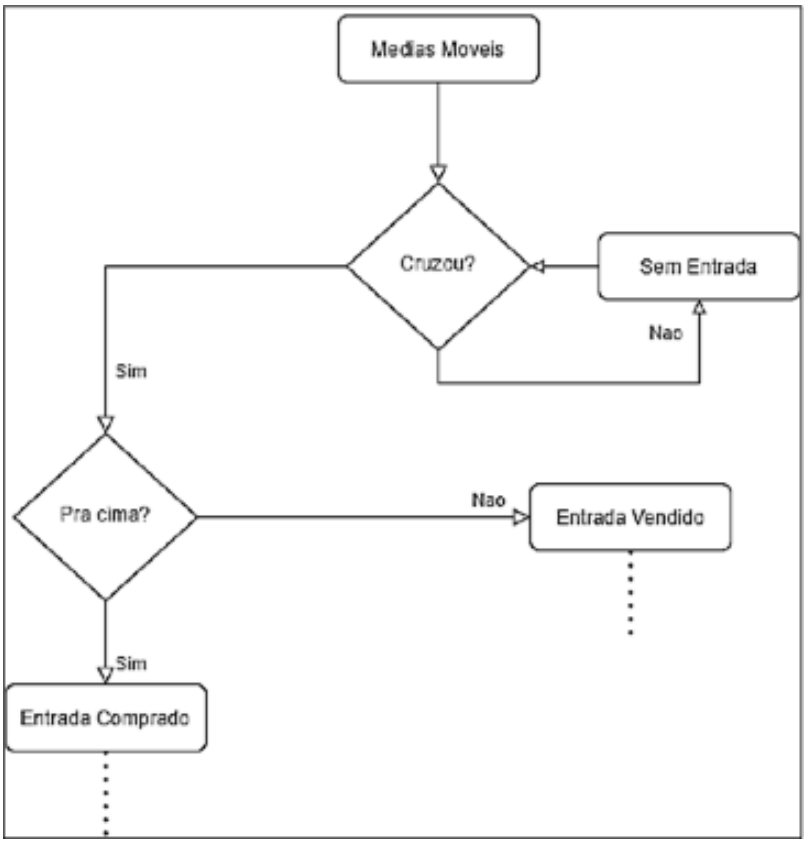
3 PROJETO DE UM S*OFTWARE* DE ESTRATÉGIAS DE *TRADING*

Neste capítulo apresentam-se conceitos pivotais do *software* de desenvolvimento de estratégias de *trading*, tal como o conceito de fluxograma e variáveis.

3.1 Fluxograma

O fluxograma é uma forma de visualizar processos, uma descrição visual de como realizar um processo etapa por etapa. É composto por blocos e conexões entre blocos, em um contexto de programação visual no qual cada bloco representa operações lógicas, como ações e decisões, conforme mostra a Figura 13.

Figura 13 - Exemplo de fluxograma



Fonte: MARCOS, 2021.

Fluxogramas como esse serão utilizados para fornecer ao usuário uma *interface* mais atrativa de desenvolvimento de estratégias de *trading*, especialmente aos usuários sem familiaridade com linguagens de programação.

A *interface* de edição de fluxograma permite um conjunto de tipos blocos. Cada bloco pode conter um ou mais parâmetros de entrada e zero ou um parâmetro de saída. Por exemplo, o bloco que calcula a média móvel pode ter o número de períodos como parâmetro de entrada.

Em outro exemplo, o bloco que calcula a média móvel pode retornar a média móvel como parâmetro de saída.

Todos os blocos do *software* de estratégias de *trading* precisam estar conectados ao fluxograma. Os tipos de blocos do *software* são descritos a seguir:

- Bloco de obter preço:

  Descrição: Obtém o preço do ativo em um instante do passado;

  Entrada: “Instante”: Instante para obter o preço;

  Saída: “Resultado”: Preço do ativo naquele instante;

- Bloco de obter indicador:

  Descrição: Obtém um indicador do ativo em um instante do passado;

  Esse bloco pode conter entradas extras dependendo do indicador a ser obtido (por exemplo, média móvel também contém o número de períodos como parâmetro de entrada);

  Entrada: “Instante”: Instante para obter o indicador;

  Saída: “Resultado”: Valor do indicador naquele instante;

- Bloco aritmético:

  Descrição: Realiza uma operação aritmética;

  Existem 4 operações nesse bloco: SOMA, SUBTRAÇÃO, MULTIPLICAÇÃO, DIVISÃO;

  Entradas: “Valor A”, “Valor B”.

  Saída: “Resultado”: Resultado da operação.

- Bloco lógico:

  Descrição: Realiza uma operação lógica.

  Existem 4 operações nesse bloco: MAIOR QUE, MENOR QUE, IGUAL, DIFERENTE, OU, E;

  Entradas: “Valor A”, “Valor B”.

  Saída: “Resultado”: Resultado da operação.

- Bloco condicional:

  Descrição: Divide o fluxograma em dois caminhos: um será executado se a condição for verdadeira, o outro será executado se a condição for falsa;

  Entrada: “Condição”.

- Bloco de atribuir variável:

  Descrição: Atribui um valor a uma variável;

Entradas: “Nome”: Nome da variável a ser atribuída, Valor”: Valor a ser atribuído a variável;

- Bloco de obter variável:

  Descrição: Obtém uma variável;

  Entrada: “Nome”: Nome da variável a ser obtido;

  Saída: “Valor”: Valor da variável.

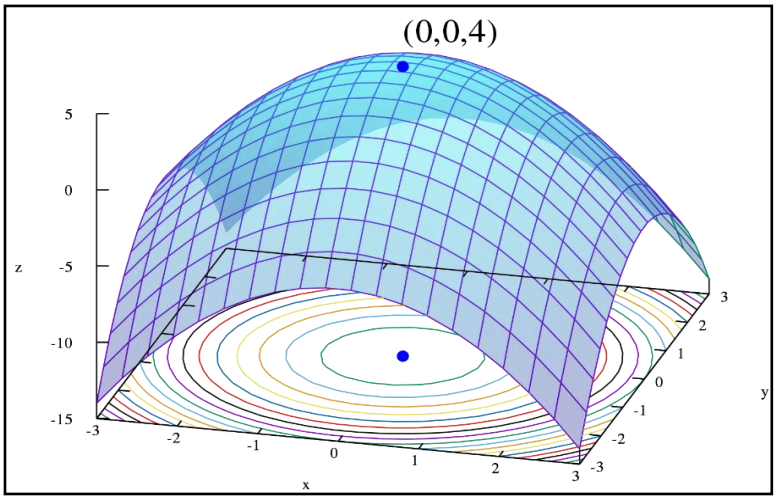
3.2 Otimização

Uma das funcionalidades principais do *software* é a otimização. A otimização em um contexto matemático é definida como: “a seleção de um melhor elemento, em relação a algum critério, de algum conjunto de alternativas disponíveis”. No contexto do *software*, a otimização é uma ferramenta que ajuda o *trade*r a encontrar, automaticamente, os melhores parâmetros para os indicadores do fluxograma (YERRIC, 2002).

Em um exemplo de uso da otimização, um fluxograma que utilize a média móvel aritmética, requer que o usuário insira o número de períodos que deseja para o cálculo (parâmetro de entrada). Com a ferramenta de otimização o programa pode executar muitas simulações para encontrar automaticamente o melhor número de períodos com o objetivo maximizar os ganhos no ativo selecionado.

Ao desenvolver o fluxograma, o usuário está desenvolvendo um programa que recebe dados e retorna uma ação (comprar, vender ou esperar). O ato de otimizar as variáveis do fluxograma para maximizar o lucro financeiro é implicitamente equivalente a encontrar o ponto máximo de uma função matemática como a exibida na Figura 14.

Figura 14 - Exemplo do ponto máximo de uma função



Fonte: IKAMUSUMEFAN, 2015.

Nos itens a seguir são apresentadas formas de se encontrar o ponto máximo da função que o fluxograma gera ao ser otimizado para maximizar o lucro financeiro.

3.2.1 Algoritmo Random Search

O algoritmo de *Random Search* (busca aleatória) buscando a melhor solução testando soluções aleatórias no espaço de busca, como exibido na Figura 15 (BROWNLEE, 2020).

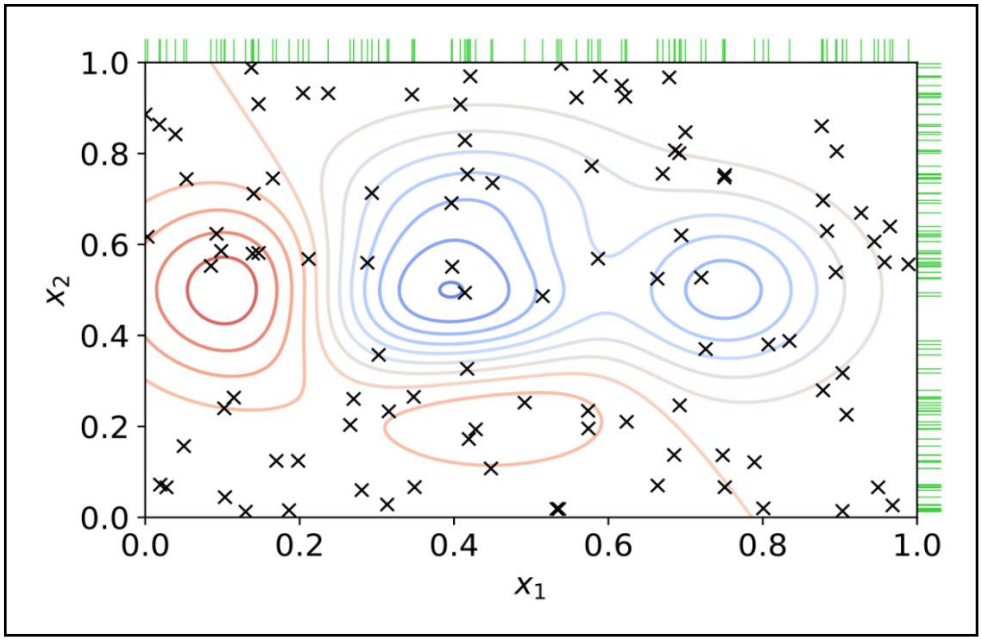
Segue o pseudocódigo do algoritmo *Random Search*:

```
A: todas as variáveis com valores aleatórios dentro de um conjunto de
valores possíveis;
Até o critério de término ser alcançado:
      B: todas as variáveis com valores aleatórios dentro de um conjunto
      de valores possíveis;
      Se a função com o conjunto de variáveis B gerar um resultado
      melhor que a função com o conjunto de variáveis A:
            A = B;
```

O critério de término pode ser definido, por exemplo, como a quantidade de iterações executadas.

O algoritmo de *Random Search* pode ser exemplificado fazendo analogia a uma pessoa que tenta o valor de 15 para o número de períodos de uma média móvel aritmética e descobre que o lucro financeiro foi de 20% ao final do dia, então a pessoa testa os valores 3 e 56 também para verificar se são melhores que o valor de 15.

Figura 15 - Visualização do algoritmo *Random Search*



Fonte: ELVERS, 2019.

3.3 Algoritmo *Hill Climbing*

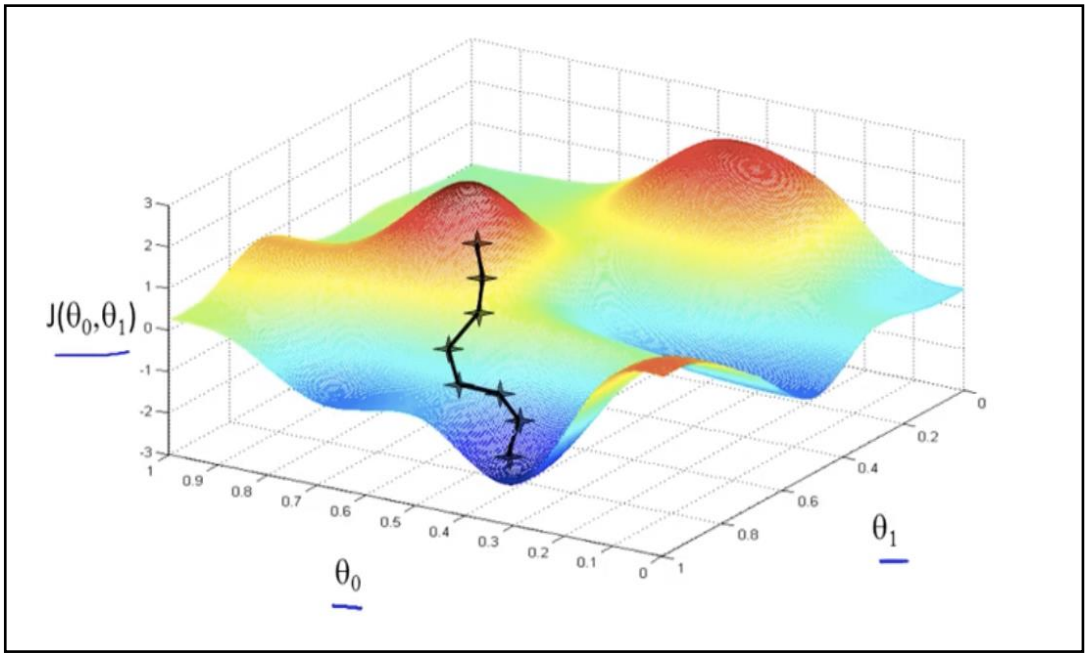
O algoritmo de *Hill Climbing* (subida de montanha) busca soluções melhores do que a solução ao redor da solução atual como exibido na Figura 16, ao invés de procurar soluções aleatórias no espaço de busca como o algoritmo *Random Search* (CERIGO, 2018).

Segue o pseudocódigo do algoritmo *Hill Climbing*:

```
A = todas as variáveis com valores aleatórios dentro de um conjunto de
valores possíveis;
Executar em loop:
      B = vizinhos de A;
      Se a função com pelo menos um dos conjuntos de variáveis em B
      gerar um resultado melhor que a função com o conjunto de variáveis
      A:
            A = melhor conjunto de variáveis de B;
      Senão:
Parar
```

O algoritmo de *Hill Climbing* pode ser exemplificado fazendo analogia a uma pessoa que tenta o valor de 15 para o número de períodos de uma média móvel aritmética e descobre que o lucro financeiro foi de 20% ao final do dia, então a pessoa testa os valores 14 e 16 também para verificar se são melhores que o valor de 15.

Figura 16 - Exemplo de funcionamento do algoritmo *Hill Climbing*



Fonte: CERIGO, 2018.

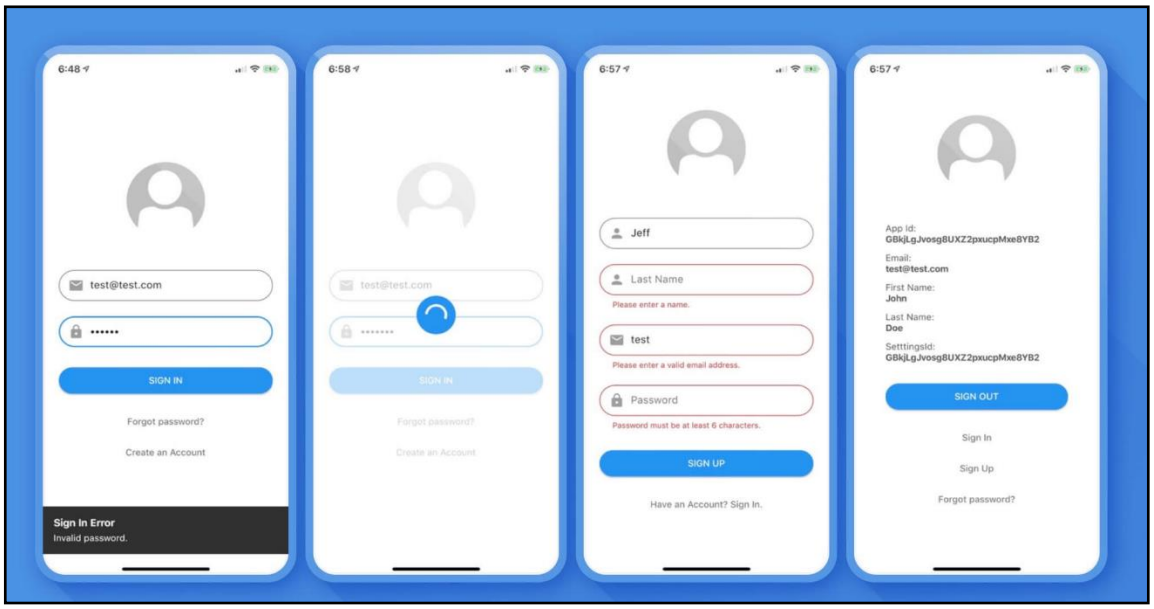
3.4 Plataforma de desenvolvimento

3.4.1 Flutter

O Flutter é um *kit* de desenvolvimento que permite que com uma única base de código, seja possível implementar um *software* em diversos sistemas operacionais, como Android, iOS, Windows, Linux e Mac OS (BRIAN, 2020).

O Flutter foi apresentado pela Google em 2017 para resolver um problema encontrado por muitos desenvolvedores, que ao desenvolver o aplicativo era necessário reimplementá-lo múltiplas vezes pois com as abordagens nativas não era necessário com a mesma base de código publicar para múltiplas plataformas como Android e iOS por exemplo, o que elevava os custos de desenvolvimento (BRIAN, 2020), a Figura 17 exibe um exemplo de aplicação desenvolvida com o Flutter.

Figura 17 - Exemplo de aplicação desenvolvida com o Flutter



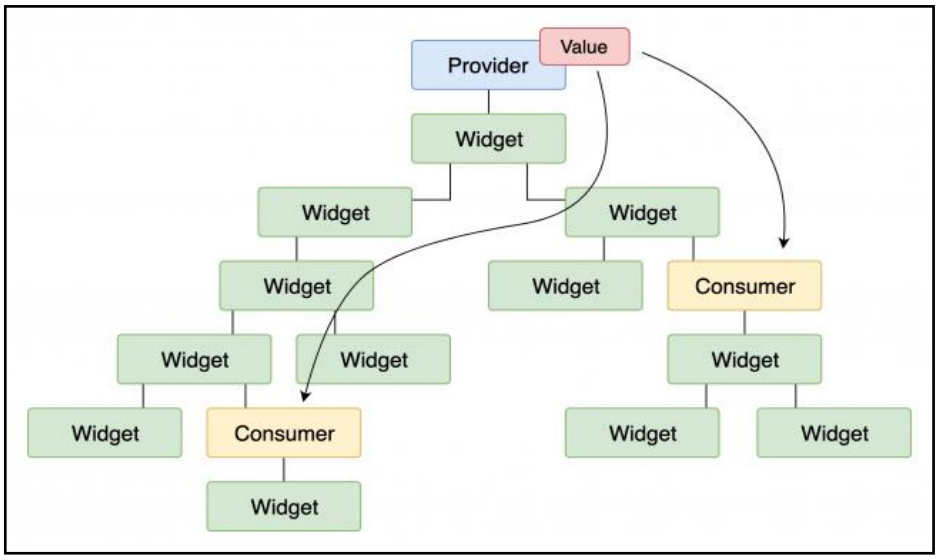
Fonte: MCMORRIS, 2019.

3.4.2 Widgets

Para desenvolver *interface*s de usuário, o Flutter utiliza como bloco de construção principal os *widgets*, que são componentes reutilizáveis que descrevem como a *interface* de usuário deve ser exibida a partir do estado atual da aplicação (FLUTTER, sem data).

No Flutter, cada widget contém outros *widgets*, de forma que juntos eles formam uma árvore, como exibido na Figura 18.

Figura 18 - Árvore de *widgets* do Flutter



Fonte: SANDE, 2020.

Alguns exemplos de *widgets* são:

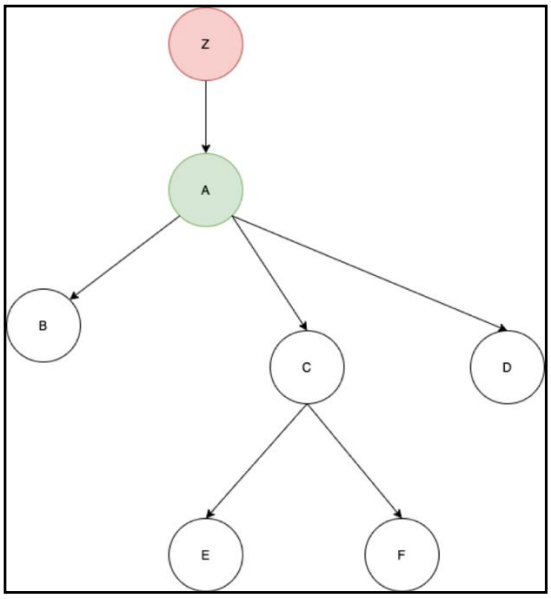
- *Text*: exibe um texto;
- *Image*: exibe uma imagem;
- *Button*: exibe um botão/
- *SafeArea*: impede os *widgets* de serem desenhados fora da área visível da tela (por exemplo, abaixo de um *notch*);

Quando um *widget* é alterado (por exemplo, um botão é alterado), o Flutter analisa a árvore e reconstrói apenas a parte que foi alterada da *interface*, assim melhorando o desempenho do aplicativo (JAIN, 2020).

O Flutter foi desenvolvido para criar aplicativos em um estilo chamado de declarativo. Esse estilo difere do estilo imperativo que é muito comum em programas que utilizam orientação a objetos. O estilo declarativo é utilizado porque ele tende a criar aplicativos com *interface* gráfica mais fácil de manter (SANDE, 2020).

No estilo declarativo, o *framework* reconstrói a *interface* quando uma mudança acontece. Por outro lado, no estilo imperativo, o próprio elemento realiza a mudança e se reconstrói. Como no Flutter os *widgets* são organizados em uma árvore, um *widget* de botão pode precisar de alguma forma de enviar uma mensagem para um *widget* mais acima da árvore quando o botão é pressionado. Existem vários padrões de projeto para organizar esses eventos. Um padrão popular é chamado de *Provider*, a árvore de *widgets* formada com o padrão *Provider* é exibida na Figura 19 (JAIN, 2020).

Figura 19 - Padrão *Provider*



Fonte: JAIN, 2020.

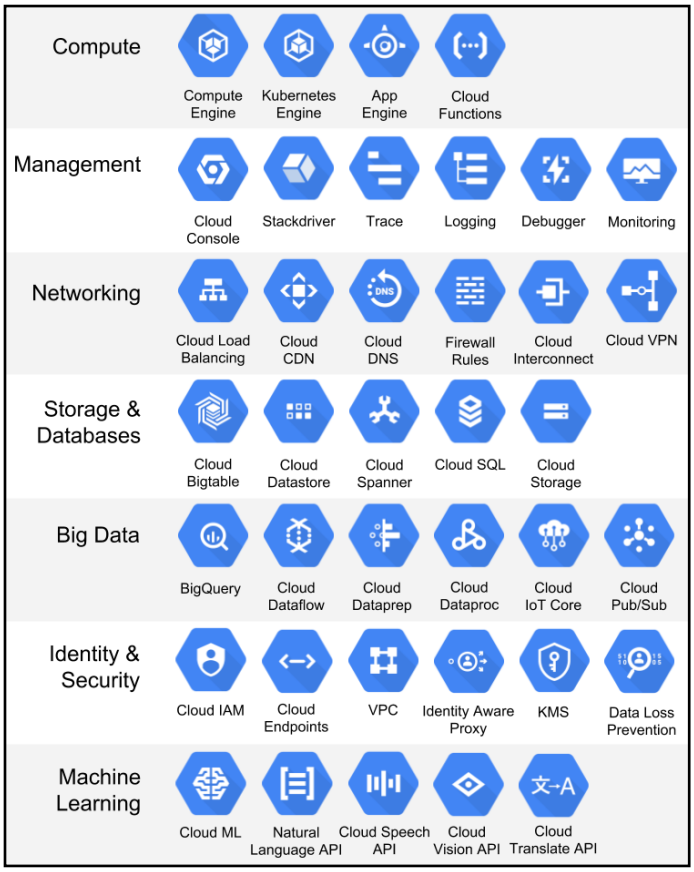
No padrão *Provider*, um nó é adicionado na árvore com o intuito apenas de comunicar com os nós abaixo (JAIN, 2020).

3.5 Banco de Dados

Como o *software* de desenvolvimento de estratégias de *trading* contém funcionalidades como otimização e *backtesting*, é necessário que haja um banco de dados históricos de ativos para que o usuário possa utilizar. Esse banco de dados pode ser fornecido manualmente pelo usuário, como em um arquivo externo ou por meio de *download* diretamente pela aplicação.

O Google Cloud Platform é um serviço que fornece uma infraestrutura completa para construir aplicações na nuvem. A Google utiliza o Google Cloud Platform para serviços como o YouTube e o buscador do Google, a Figura 20 exibe diversos serviços fornecidos pelo Google Cloud Platform (PERFORCE, 2021).

Figura 20 - Serviços do Google Cloud Platform



Fonte: HUNTER et al, 2018.

Para a criação do banco de dados será utilizado nesse projeto o serviço Cloud Firestore *Application Programming Interface* (API), que possibilita a criação e manutenção de um servidor de arquivos na nuvem do Google Cloud Platform.

4 DESENVOLVIMENTO DO S*OFTWARE* DE ESTRATÉGIAS DE *TRADING*

Neste capítulo apresentam-se as etapas no desenvolvimento do *software* de desenvolvimento de estratégias de *trading*, tal como o ambiente de desenvolvimento e a funcionalidade de cada tela do *software*.

4.1 Ambiente de desenvolvimento

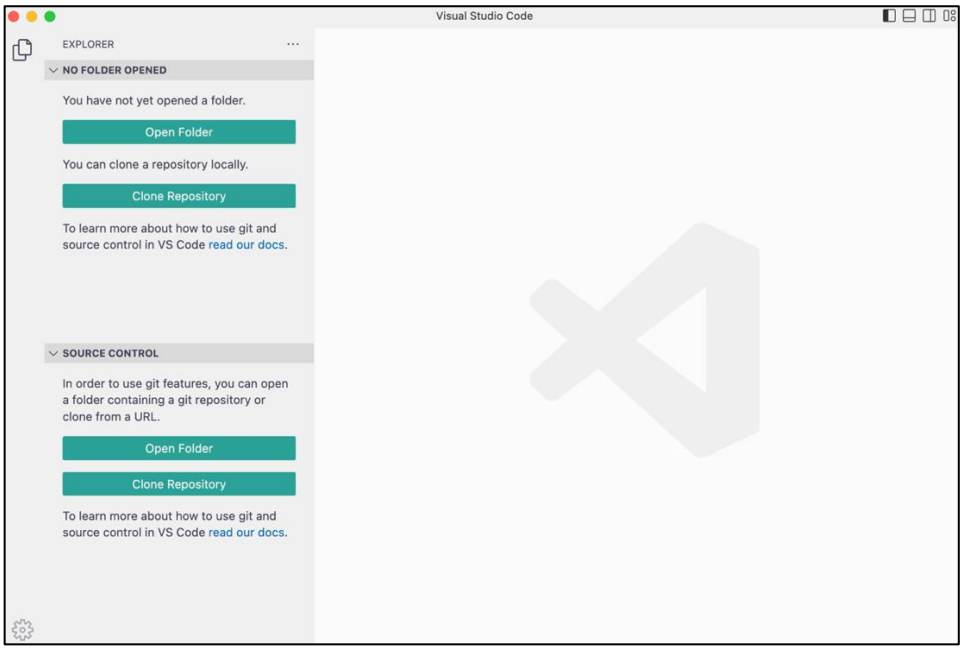
Nesta seção apresentam-se as ferramentas que compõe o ambiente de desenvolvimento do *software*, tais ferramentas incluem o *Integrated Development Environment* (IDE) e a ferramenta de design da interface gráfica.

4.1.1 *Software*

O *Integrated Development Environment*, Ambiente de Desenvolvimento Integrado (IDE) é uma aplicação de *software* que provê ferramentas que agilizam no desenvolvimento de *softwares*. Tais ferramentas podem incluir a construção automática de *interfaces* gráficas, sistema de controle de versão e hierarquia de classes (CODE ACADEMY, sem data).

A IDE escolhida para o desenvolvimento do *software* é o *Visual Studio Code*, que proporciona ferramentas para o desenvolvimento de aplicações multiplataforma por meio do *React Native*. A tela inicial do *Visual Studio Code* é exibida na Figura 21.

Figura 21 – Tela inicial do *Visual Studio Code*.



Fonte: Capturado pelo autor de *Visual Studio Code*.

Para construir o diagrama inicial da *interface* gráfica do *software*, foi utilizada a ferramenta *Figma*, que consiste em um editor visual no qual é possível construir modelos configurando elementos visuais, como imagens e textos, em posições específicas. O diagrama construído na ferramenta pode ser visualizado na Figura 22.

Figura 22 – Modelo da *interface* gráfica construída no *Figma*.

Fonte: Capturado pelo autor de *Figma*.

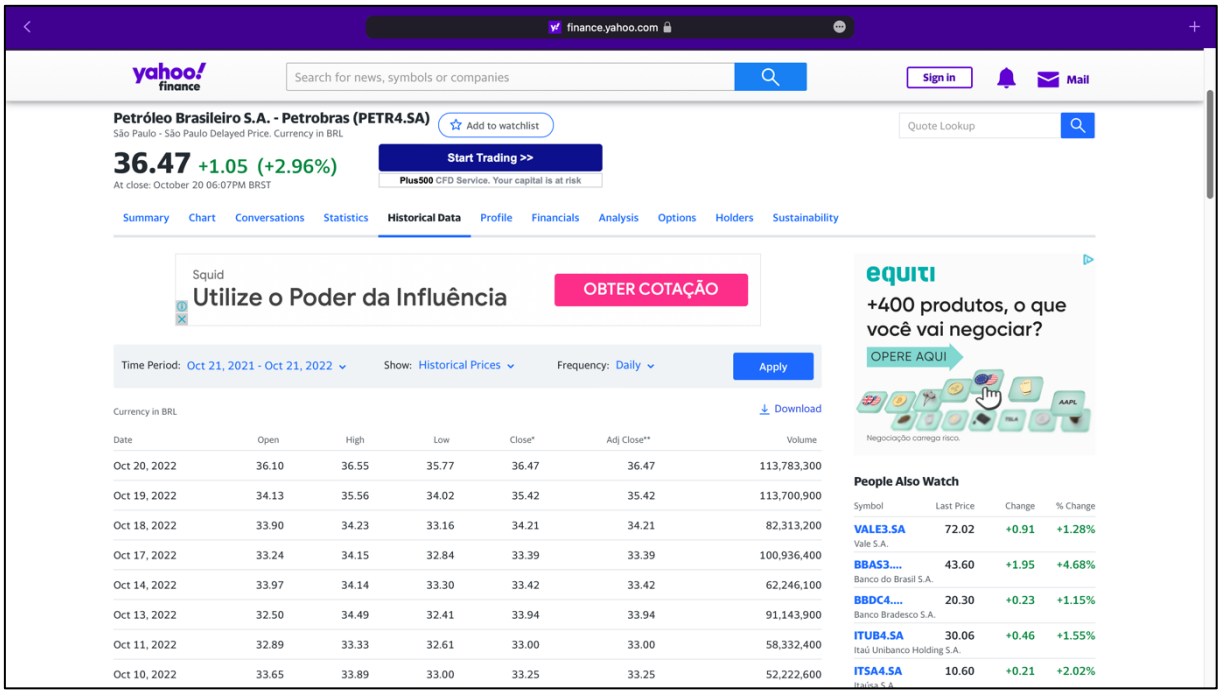
Pode-se observar na Figura 22, que o *software* possui 4 telas:

- Tela inicial: permite navegar para as outras telas;
- Lista de ativos: permite baixar e selecionar um ativo para simular no simulador;
- Fluxograma: permite a construção da estratégia por meio do fluxograma;
- Simulador: realiza a simulação da estratégia e exibe os resultados.

4.1.2 Banco de dados

Para a criação do banco de dados foram utilizados os dados históricos disponíveis no *Yahoo! Finance*, um *website* que oferece notícias, dados e comentários relacionados a eventos do mercado financeiro. A Figura 23 mostra um exemplo de tela do *Yahoo! Finance* que exibe os dados históricos das ações de Petrobrás (PETR4) (YAHOO! FINANCE, 2022).

Figura 23 – Dados históricos da Petrobrás no *Yahoo! Finance*.



Fonte: YAHOO! FINANCE, 2022.

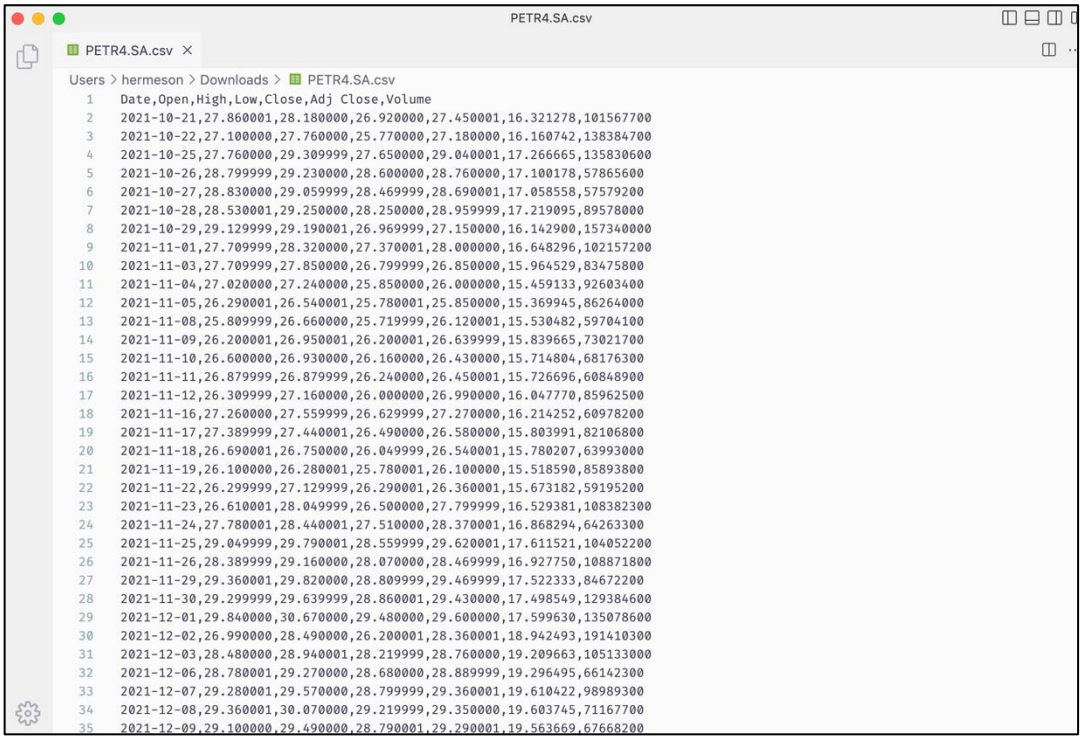
Os dados históricos de alguns ativos disponíveis em *Yahoo! Finance* foram exportados para posteriormente abastecer o Servidor de Banco de Dados. Para isso, foi feita a busca de um determinado ativo (exemplo: PETR4) e, ao clicar na seção *Show Historical Data*, é disponibilizado o *link* para *Download* no formato *Comma Separated Values*, Valores Separados por Vírgulas (CSV). Os dados históricos de um dos ativos exportados são exibidos na Figura 24.

Conforme pode ser observado na Figura 24, a primeira linha do arquivo é composta pelo rótulo de cada campo. No arquivo é possível visualizar os campos:

- *Date*: Data;
- *Open*: Preço de abertura do dia;
- *High*: Preço máximo do dia;
- *Low*: Preço mínimo do dia;

- *Close*: Preço de fechamento do dia;
- *Adj Close*: Preço de fechamento do dia ajustado;
- *Volume*: Volume financeiro de negócios.

Figura 24 – Dados históricos da Petrobrás salvos localmente.



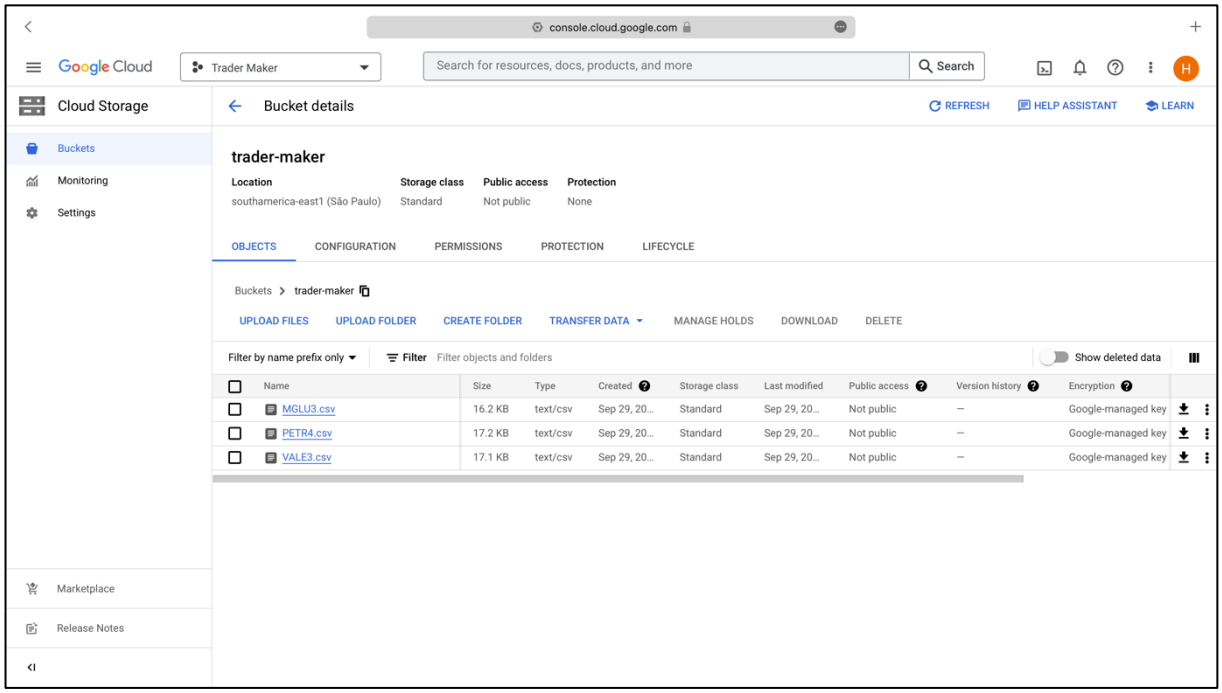
Fonte: Autoria própria.

Para criação da tabela no banco de dados, foram utilizados apenas os campos *Date* e *Close*, pois são eles que contém as informações necessárias para a criação do fluxograma, que opera considerando os valores do preço de fechamento em cada dia.

O envio dos dados históricos para o *Google Cloud Platform* foi feito por meio do painel *Cloud Storage Buckets*, que contém um *link* para *upload* em *Upload Files*. A tela de envio de arquivos é exibida na Figura 25.

Uma vez que os dados históricos foram enviados ao *Google Cloud Storage*, um *link* público é criado para cada arquivo importado, que possibilita acessá-lo via rede, para que a aplicação possa efetuar o *download* localmente em cada instância executável do *software*.

Figura 25 – Tela de envio de arquivos do *Google Cloud Storage*.



Fonte: Google Cloud Platform, [s. d.].

4.2 *React Native*

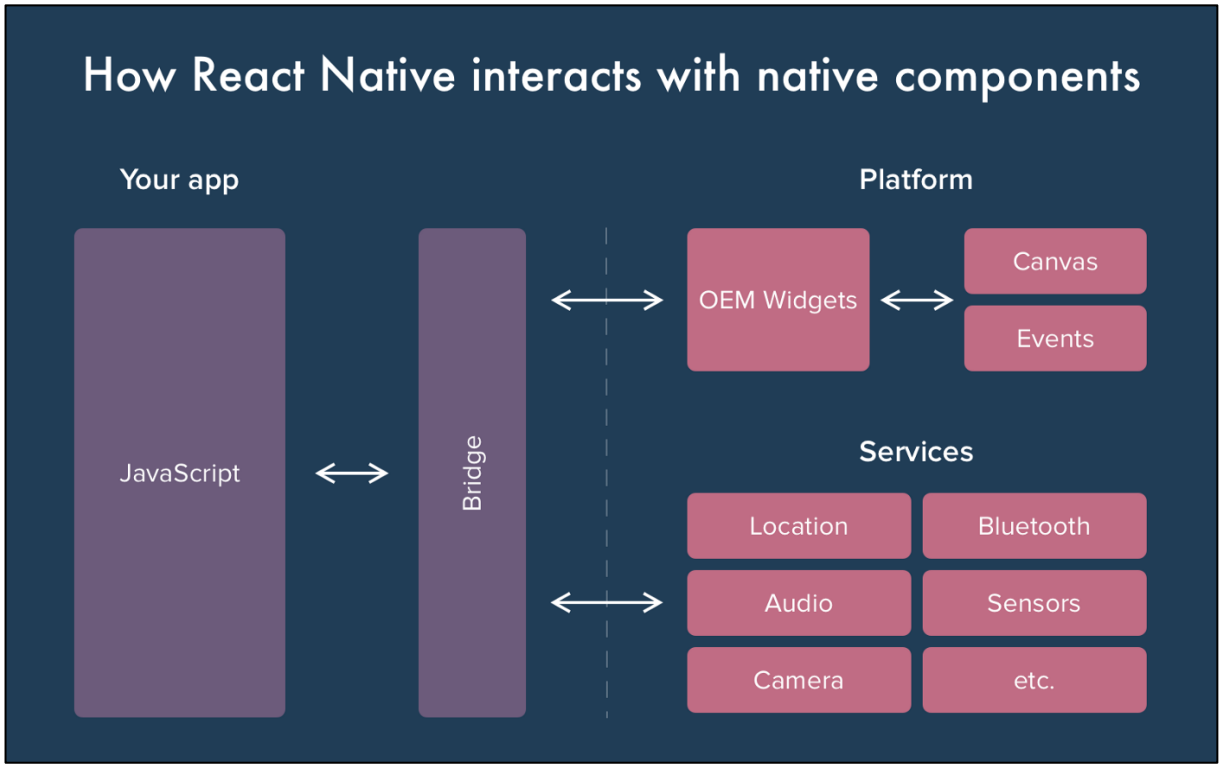
O *React Native* é um *kit* de desenvolvimento de aplicações para múltiplas plataformas similar ao *Flutter* e foi lançado pela primeira vez pela *Meta Platforms, Inc* em 2015 (REACT NATIVE, 2022).

A principal diferença entre o *Flutter* e o *React Native* é que, aplicações desenvolvidas com o *Flutter* tem exatamente o mesmo *design* independentemente do sistema operacional que está sendo executada. Já uma aplicação desenvolvida com o *React Native* tem um *design* que se adapta ao *design* padrão adotado no sistema operacional que está sendo executado (CEDRIC, 2022).

Baseado nessa observação, notou-se que o fato do *React Native* se adaptar ao sistema operacional do qual o *software* está sendo executado forneceria uma interface gráfica mais familiar ao usuário, melhorando a inteligibilidade do *software*, assim foi feita a migração do código fonte do *Flutter* para o *React Native*.

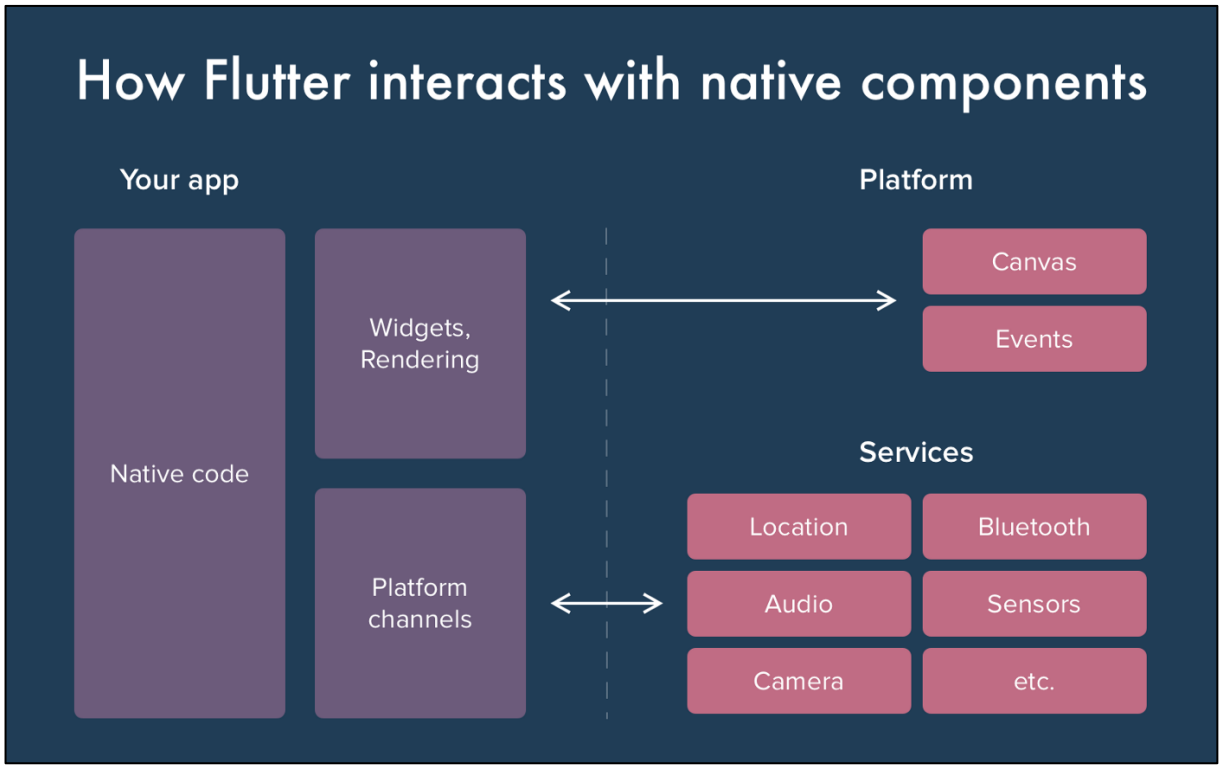
Essa diferença se deve ao fato de que o *Flutter* desenha elementos de forma customizada em um *Canvas*, já o *React Native* possui uma arquitetura em que o código da aplicação interage com o processo em plano de fundo que se comunica diretamente com o sistema operacional, e envia comandos para adicionar elementos nativos na tela. A arquitetura do *React Native* e do *Flutter* são exibidos na Figura 26 e na Figura 27 (CEDRIC, 2022).

Figura 26 – Arquitetura do *React Native*.



Fonte: KUPRENKO, 2019.

Figura 27 – Arquitetura do *Flutter*.



Fonte: KUPRENKO, 2019.

4.3 Tela inicial

Nessa etapa foi utilizado o *React Native* para construção da *interface* gráfica, seguindo o modelo criado na ferramenta *Figma*. A primeira tela a ser desenvolvida foi a tela inicial, na qual possui o título do aplicativo posicionado na parte superior da tela, o ativo selecionado, e dois botões para acessar as telas “Editor” e “Simulador”. A imagem da tela inicial pode ser visualizada na Figura 28.

Figura 28 – Tela inicial.

Fonte: Autoria própria.

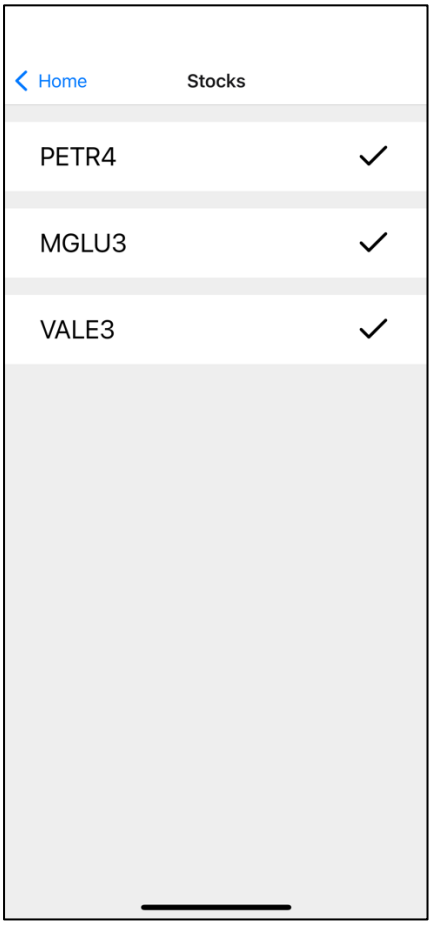
A tela inicial é a primeira tela a ser visualizada pelo usuário ao abrir o *software*, ao clicar nos botões o usuário pode navegar para as outras telas, e é possível retornar para a tela inicial clicando no botão de voltar de cada tela, seguindo o fluxo exibido na Figura 22.

4.4 *Download* dos ativos

Para selecionar o ativo a ser utilizado na simulação, é utilizado a tela de *download* de seleção de ativos. Nela é possível visualizar uma lista de ativos que estão disponíveis no banco de dados criado no *Google Cloud Platform*.

Cada ativo pode ser baixado clicando pela primeira vez no botão que representa aquele ativo. Durante o *download*, os dados históricos do ativo são baixados por meio do *link* do *Google Cloud Platform* e salvo localmente no disco para acesso. Após o *download*, o ativo permanece selecionado como ativo principal na tela inicial e na tela de simulação, conforme é exibido na Figura 29.

Figura 29 – Tela de seleção de ativos.



Fonte: Autoria própria.

4.5 Fluxograma

O fluxograma é a ferramenta utilizada pelo usuário para desenvolver suas estratégias, todo fluxograma é iniciado por um *node* de início e terminado por um *node* de compra, *node* de venda ou *node* de nenhuma ação.

Cada *node* é conectado a um *node* anterior ou a um *node* próximo, o que representa o fluxo lógico da estratégia, quando um node é modificado o fluxograma é salvo automaticamente no disco para que a estratégia permaneça salva ao voltar para a tela inicial. A tela de criação do fluxograma é exibida na Figura 30.

Figura 30 – Tela de criação do fluxograma.

Fonte: Autoria própria.

No *React Native*, a implementação do fluxograma é composta por uma iteração na lista de estruturas de *nodes* adicionados no grafo e conversão para *nodes* visualizáveis na tela. Cada *node* possui uma largura, altura, posição horizontal e posição vertical para ser desenhado na tela. O código de representação dos *nodes* para serem apresentados na tela é exibido na Figura 31.

Figura 31 – Código de criação de *nodes* para apresentação.

```
props.nodes.forEach(node => {
    nodes.push(
        <Node node={node} updateNode={props.updateNode} selectedNode={selectedNode} setSelectedNode={(id: number | null) => {
            if (selectedNode == null) {
                setSelectedNode(id)
            }

            else {
                const nodeInterface = props.nodes.get(selectedNode)!
                nodeInterface.nextId = id
                props.updateNode(selectedNode, nodeInterface)
                setSelectedNode(null)
            }
        }} onLayout={(layoutChangeEvent) => {
            const nodeInterface = props.nodes.get(node.id)!
            nodeInterface.bounds = {width: layoutChangeEvent.nativeEvent.layout.width, height: layoutChangeEvent.nativeEvent.layout.height}
            props.updateNode(node.id, nodeInterface)
        }}/>
    )

    const nextPosition = props.nodes.get(node.nextId!)

    if (nextPosition != null) {
        const currentBounds = node.bounds
        const nextBounds = props.nodes.get(node.nextId!)!

        lines.push(
            <Line
                x1={node.position.x + currentBounds.width / 2}
                y1={node.position.y + currentBounds.height}
                x2={nextPosition.position.x + nextBounds.bounds.width / 2}
                y2={nextPosition.position.y} stroke='black' strokeWidth='2'/>
        )
    }
})
```

Fonte: Autoria própria.

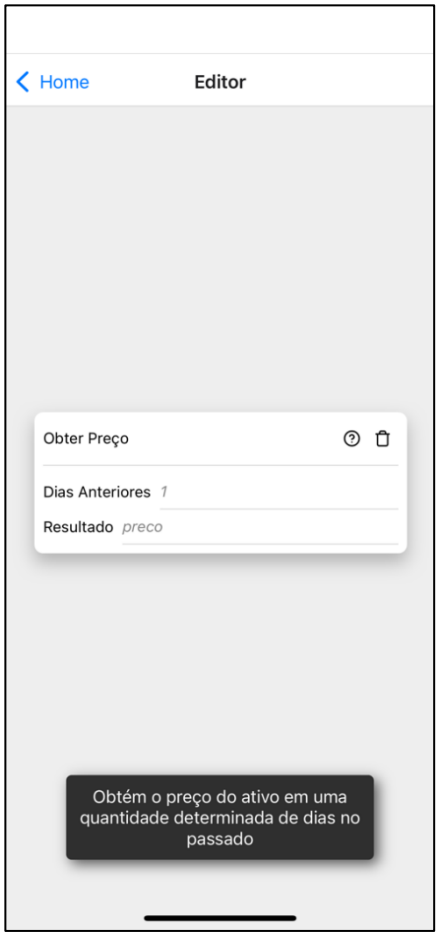
Para cada estrutura de *node* é criado um componente visualizável e adicionado na lista de *nodes* para ser apresentado na tela. Cada vez que a lista de *nodes* é modificada por uma interação do usuário, o *React Native* realiza uma nova atualização para exibir os componentes atualizados na tela de forma dinâmica.

Cada *node* possui uma estrutura composta pelos elementos:

- Título: Exibe o título do *node*;
- Botão de ajuda: Exibe uma descrição da funcionalidade do *node* (exibido na Figura 32*)*;
- Botão de exclusão: Exclui o *node* do fluxograma;
- Lista de variáveis: Exibe uma lista de variáveis necessárias para a operação realizada pelo *node*.

Cada variável da lista de variáveis é composta por uma descrição da variável e um nome de variável. Essa estrutura é mantida e salva em disco por meio de uma estrutura chave-valor, como um *Hash Map*.

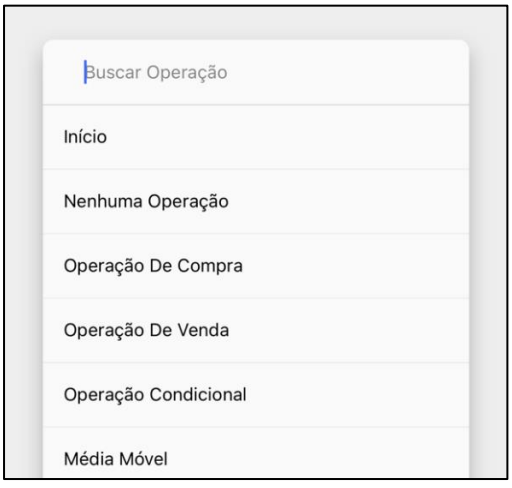
Figura 32 – Funcionalidade de ajuda.



Fonte: Autoria própria.

Ao tocar duas vezes consecutivas em uma área vazia do editor é aberto o menu de adição de *nodes* que permite ao usuário adicionar novos *nodes* no fluxograma, conforme exibido na Figura 33.

Figura 33 – Menu de adição de *nodes*.



Fonte: Autoria própria.

O menu de adição de *nodes* é composto por uma caixa de pesquisa e uma lista de botões para cada operação suportada pelo *software*. Ao clicar em um botão, o menu é fechado e o *node* selecionado é adicionado no fluxograma.

Para implementar o menu de adição de *nodes*, a cada vez que o usuário toca em uma área vazia do editor, é registrado o momento em que o toque foi realizado. Se o último toque ocorreu há menos de meio segundo, o menu é aberto. A implementação dessa funcionalidade é exibida na Figura 34.

Figura 34 – Implementação da abertura do menu de adição de *nodes*.

```
<View onResponderGrant={({nativeEvent}) => {
    const pressTime = new Date().getTime()
    const deltaPressTime = pressTime - lastPressTime.current
    lastPressLocation.current = {x: nativeEvent.locationX, y: nativeEvent.locationY}
    lastPressTime.current = pressTime

    if (deltaPressTime < 500) {
        setMenuList(OPERATORS)
        setMenuInput('')
        setMenuVisible(true)
    }
}}
```

Fonte: Autoria própria.

A caixa de pesquisa é utilizada para buscar uma operação específica. Ao escrever um texto na caixa de pesquisa, por exemplo, “Média”, todas as operações que começam com o texto “Média” são filtradas e exibidas na lista de operações, o que facilita para o usuário a busca de operações de Média Móvel Aritmética ou Média Móvel Exponencial. A implementação da caixa de pesquisa é exibida na Figura 35.

Figura 35 – Implementação da caixa de pesquisa.

```
<TextInput
    autoCapitalize='none'
    autoComplete='off'
    autoCorrect={false}
    autoFocus={true}
    onChangeText={(text) => {
        text = validateString(text)
        setMenuList(OPERATORS.filter((operator) => validateString(operator.name).startsWith(text)))
        setMenuInput(text)
    }}
    placeholder={toCamelCase(DICT.text.search_operation)}
    placeholderTextColor='#888888'
    returnKeyType='done'
    style={{borderRadius: 8, minHeight: 50, paddingHorizontal: 32}}
    value={menuInput}/>
```

Fonte: Autoria própria.

Cada *node* possui um identificador que permite que a tela de simulação identifique cada *node* e realize de forma correta a simulação da operação, conforme exibido na implementação representada na Figura 36.

Figura 36 – Implementação da execução lógica de vários *nodes*.

```
else if (currentNode!.operator.name == DICT.text.max_indicator) {
    SIMULATION_MAP.set(
        currentNode.variables.get(DICT.text.result)!.toString(),
        Math.max(SIMULATION_MAP.get(currentNode.variables.get(DICT.text.indicator_1)!.toString())!,
        SIMULATION_MAP.get(currentNode.variables.get(DICT.text.indicator_2)!.toString())!))
}

else if (currentNode!.operator.name == DICT.text.min_indicator) {
    SIMULATION_MAP.set(
        currentNode.variables.get(DICT.text.result)!.toString(),
        Math.min(SIMULATION_MAP.get(currentNode.variables.get(DICT.text.indicator_1)!.toString())!,
        SIMULATION_MAP.get(currentNode.variables.get(DICT.text.indicator_2)!.toString())!
    ))
}

else if (currentNode!.operator.name == DICT.text.sum_indicators) {
    SIMULATION_MAP.set(
        currentNode.variables.get(DICT.text.result)!.toString(),
        SIMULATION_MAP.get(currentNode.variables.get(DICT.text.indicator_1)!.toString())!
        + SIMULATION_MAP.get(currentNode.variables.get(DICT.text.indicator_2)!.toString())!
    )
}

else if (currentNode!.operator.name == DICT.text.subtract_indicators) {
    SIMULATION_MAP.set(
        currentNode.variables.get(DICT.text.result)!.toString(),
        SIMULATION_MAP.get(currentNode.variables.get(DICT.text.indicator_1)!.toString())!
        - SIMULATION_MAP.get(currentNode.variables.get(DICT.text.indicator_2)!.toString())!
    )
}

else if (currentNode!.operator.name == DICT.text.multiply_indicators) {
    SIMULATION_MAP.set(
        currentNode.variables.get(DICT.text.result)!.toString(),
        SIMULATION_MAP.get(currentNode.variables.get(DICT.text.indicator_1)!.toString())!
        * SIMULATION_MAP.get(currentNode.variables.get(DICT.text.indicator_2)!.toString())!
    )
}

else if (currentNode!.operator.name == DICT.text.divide_indicators) {
    SIMULATION_MAP.set(
        currentNode.variables.get(DICT.text.result)!.toString(),
        SIMULATION_MAP.get(currentNode.variables.get(DICT.text.indicator_1)!.toString())!
        / SIMULATION_MAP.get(currentNode.variables.get(DICT.text.indicator_2)!.toString())!
    )
}
```

Fonte: Autoria própria.

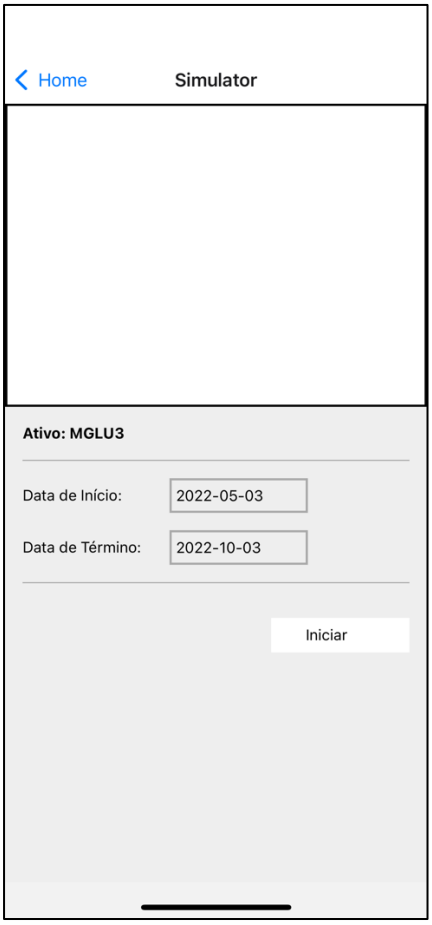
Durante todo o período da simulação, é mantido um *Hash Map* com as variáveis do fluxograma. Tais variáveis são acessadas e atualizadas dinamicamente pela lógica formada no grafo do fluxograma, de forma similar a uma linguagem de programação, que em uma linha

pode configurar uma variável resultado, como a soma de duas outras variáveis e usar essa variável para realizar outras operações em linhas posteriores.

4.6 Simulação

A tela de simulação é o local em que a estratégia do usuário é testada para verificar se ela gera retorno financeiro. Ao abrir a tela de simulação, o fluxograma criado pelo usuário é carregado para ser executado nos dias definidos pelas datas de início e de término da simulação, conforme é exibido na Figura 37.

Figura 37 – Tela de simulação.



Fonte: Autoria própria.

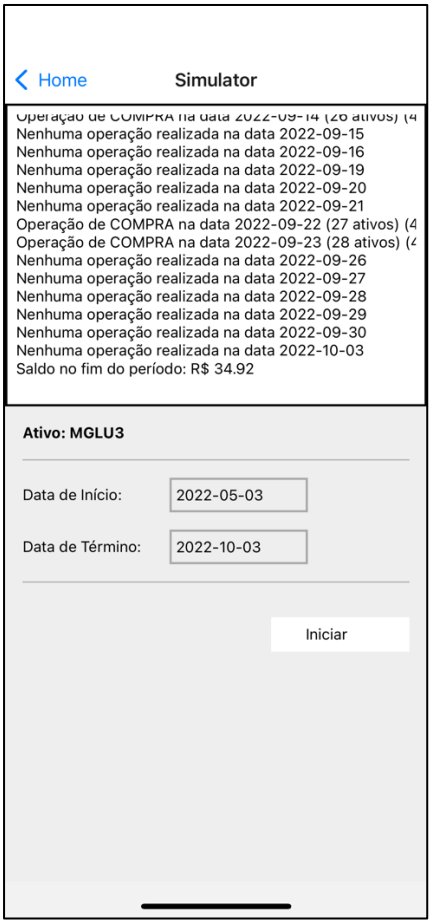
As funcionalidades dos componentes disponíveis na tela de simulação são descritas a seguir:

- Painel de resultados: Exibe os resultados da simulação;
- Ativo selecionado: Exibe o ativo na qual a estratégia será simulada;
- Data de início: Data de início da simulação;

- Data de término: Data de término da simulação.

Ao clicar no botão iniciar, o *software* realiza uma iteração de cada dia desde a data inicial até a data final, simulando a estratégia criada pelo usuário no fluxograma e salvando as operações realizadas na simulação e, por fim, o lucro ou prejuízos acumulados no período. Todos os dados são exibidos em um campo de texto na parte superior da tela, conforme é exibido na Figura 38.

Figura 38 – Resultados da simulação.



Fonte: Autoria própria.

Durante a simulação da estratégia em cada dia no período definido, o *software* pode realizar uma das três operações:

- Operação de compra: Simula a compra de um ativo;
- Operação de venda: Simula a venda total dos ativos acumulados até então;
- Nenhuma operação: Não realiza operação no dia.

5 TESTES E RESULTADOS

Neste capítulo são apresentados os testes realizados no *software* para analisar sua eficácia no auxílio da tomada de decisões do investidor. Os testes apresentam estratégias baseadas em análise gráfica e outras que envolvem a combinação de análise gráfica e análise fundamentalista da bolsa de valores.

5.1 Estratégia simples

Para criar uma estratégia simples de simulação de compra e venda no *software*, pode-se criar uma estrutura no fluxograma que opera baseada nas condições:

Se (Preço do ativo > Valor superior)
Então
    Faça (Operação de compra)
Senão Se (Preço do ativo < Valor inferior)
        Então
            Faça (Operação de venda)
        Senão
            Faça (Nenhuma operação)

A estrutura condicional descrita pode ser implementada no fluxograma conforme exibido na Figura 39, e o resultado da simulação é exibido na Figura 40.

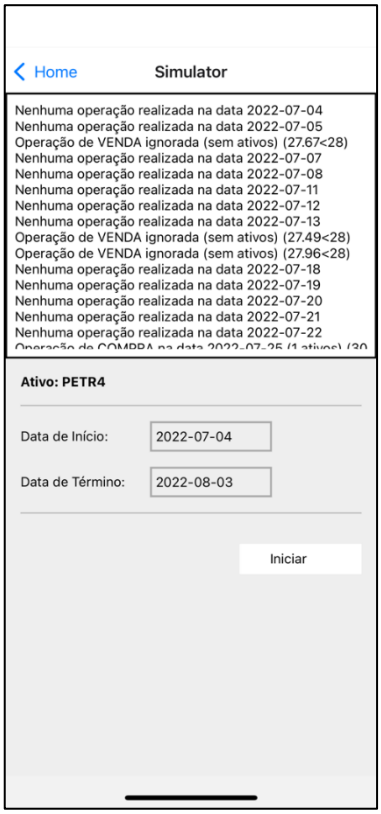
O fluxograma exibido na Figura 39 é composto por 4 *nodes*:

- Início: O início do fluxograma;
- Obter preço: Obtém o preço naquele dia (0 dias anteriores em relação ao dia atual), e armazena o resultado em uma variável chamada “preco”;
- Operação condicional: Realiza uma operação de compra ou venda baseada em condições definidas (nesse caso, realiza compra se o preço for maior que 30 e realiza venda se o preço for menor que 28), caso nenhuma das condições seja satisfeita, a execução do fluxograma continua;
- Nenhuma operação: Sinaliza que não deve realizar nenhuma operação naquele dia, já que nenhuma condição de compra ou venda foi satisfeita.

Figura 39 – Estratégia condicional simples implementada no fluxograma.

Fonte: Autoria própria.

Figura 40 – Simulação da estratégia condicional simples.



Fonte: Autoria própria.

Como é possível observar na Figura 40, a simulação foi executada no período entre 4 de julho de 2022 e 3 de agosto de 2022. No campo de resultados são exibidas as operações para cada dia do período definido, e o saldo no final da simulação foi de R$ 9,94. Para efeito de comparação, o gráfico desse ativo no *Yahoo! Finance* no período simulado pode ser visualizado na Figura 41, que confirma que esse ativo teve alta de 0,30%, o que justifica o ganho da simulação.

Figura 41 – Dados históricos da Petrobrás no *Yahoo! Finance*.

Fonte: *YAHOO! FINANCE*, 2022.

5.2 Estratégia de cruzamento de médias móveis

A estratégia de cruzamento de médias testada foi baseada nas seguintes regras, o fluxograma é exibido na Figura 42 e o resultado da simulação é exibido na Figura 43.

Se (Média móvel de 3 períodos > Preço do ativo && Média móvel de 5 períodos > Preço do ativo)

Então

Faça (Operação de compra)

Senão Se (Média móvel de 3 períodos < Preço do ativo && Média móvel de 5 períodos < Preço do ativo)

Então

Faça (Operação de venda)

Senão

Faça (Nenhuma operação)

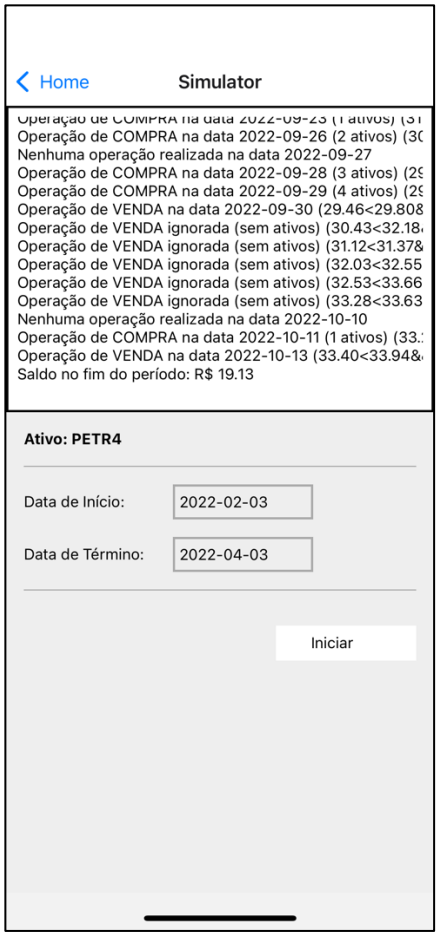
Figura 42 – Estratégia de cruzamento de médias móveis implementada no fluxograma.

Fonte: Autoria própria.

O fluxograma exibido na Figura 42 é composto por 6 *nodes*:

- Início: O início do fluxograma;
- Média móvel: Obtém a média móvel aritmética de 3 períodos e armazena o resultado em uma variável chamada “mm3”;
- Média móvel: Obtém a média móvel aritmética de 5 períodos e armazena o resultado em uma variável chamada “mm5”;
- Obter preço: Obtém o preço naquele dia (0 dias anteriores em relação ao dia atual), e armazena o resultado em uma variável chamada “preco”;
- Operação condicional: Realiza uma operação de compra ou venda baseada nas condições definidas nas regras acima, caso nenhuma das condições seja satisfeita, a execução do fluxograma continua;
- Nenhuma operação: Sinaliza que não deve realizar nenhuma operação naquele dia, já que nenhuma condição de compra ou venda foi satisfeita

Figura 43 – Simulação da estratégia de cruzamento de médias móveis.



Fonte: Autoria própria.

Como é possível observar na Figura 43, a simulação foi executada no período entre 3 de fevereiro de 2022 e 3 de abril de 2022. No campo de resultados são exibidas as operações para cada dia do período definido, e o saldo no final da simulação foi de R$ 19,13.

5.3 Combinando análise gráfica com análise fundamentalista

Também é possível utilizar o *software* para criar uma estratégia que combina a análise gráfica com análise fundamentalista (análise de variáveis socioeconômicas que podem influenciar a bolsa de valores).

Apesar do *software* não conter um componente capaz de analisar notícias e outras variáveis da bolsa de valores que não estão disponíveis no preço e nos indicadores, o investidor pode manualmente selecionar períodos chave em que ocorreu uma determinada variável socioeconômica e criar uma estratégia focada apenas naquelas janelas de tempo em que aquela variável aparece.

Essa variável que pode causar um movimento na bolsa de valores pode ser, por exemplo, notícias importantes, como fortalecimento ou enfraquecimento de relações entre países, aumento ou redução da inflação, eleição de um determinado partido nas eleições, entre outros.

Supondo que a variável socioeconômica escolhida seja a eleição de um presidente do Brasil (que acontece a cada 4 anos), o investidor pode criar uma estratégia e aplicá-la apenas no período das eleições, de forma que possa se encontrar algum padrão que possa ser explorado nestes períodos.

6 CONSIDERAÇÕES FINAIS

Com a facilitação da possibilidade de investir na bolsa de valores, existe um número cada vez maior de pessoas físicas atraídas pela chance de se obter rentabilidade *online* com *trading*. Porém as estatísticas mostram que mais de 90% das pessoas que tentam operar *day trade* perdem mais dinheiro do que ganham, o investidor iniciante acredita que, ao comprar um curso de um investidor de sucesso, tem sucesso garantido, seguindo cegamente os conselhos do orientador com investimentos reais (EINVESTIDOR, 2021).

Este trabalho fornece uma ferramenta de simulação, na qual o investidor que pretende operar com análise técnica pode testar estratégias na ferramenta utilizando dados do passado, antes de tentar qualquer operação com investimentos reais, mitigando assim, o risco de obter prejuízo na bolsa de valores.

O *software* desenvolvido para auxiliar o investidor possui, como componentes principais, o fluxograma, na qual o desenvolvedor pode implementar sua estratégia de análise técnica com uma *interface* visual de fácil uso, sem necessidade de conhecimentos de programação para sua utilização, e o simulador que executa as operações diárias em conformidade com o fluxograma criado e, ao final exibe o resultado contendo o valor do lucro ou prejuízo, conforme o caso, para que o investidor possa entender a eficácia de sua estratégia antes de fazer investimentos reais.

Apesar de já existirem *softwares* similares que permitem ao investidor implementar suas estratégias e testá-las antes de realizar investimentos reais, como o *Metatrader 4*, que é disponível para as plataformas *Mac*, *Windows* e *Linux*, estes requerem conhecimentos de programação para implementar as estratégias, o que pode ser uma barreira decisiva para que o investidor escolha tentar sem o auxílio de um *software*. Além disso, os *softwares* mais populares de teste de estratégias de investimento, como o *Metatrader 4*, não estão disponíveis para plataformas móveis, com a vantagem extra do *software* desenvolvido neste trabalho ser multiplataforma, podendo ser executado tanto em *desktops* como em *smartphones* Android e iPhone.

Desta forma, o *software* resultado deste trabalho é útil para investidores de qualquer perfil, conservador (que busca investimentos de baixo risco, porém de menor rentabilidade), moderado (que busca lucros maiores, em investimentos de risco mais vultosos) ou arrojado (que busca lucros consideravelmente maiores, mas em investimentos de risco muito elevados).

Para o desenvolvimento do *software*, foi inicialmente escolhido o *Flutter* como a plataforma de desenvolvimento multiplataforma. No entanto, durante o processo foi verificado

que o *React Native* é uma melhor escolha, por ter a capacidade de exibir componentes na tela com estilo específico do sistema operacional em que é executado, diferentemente do *Flutter* que exibe componentes com estilo genérico, resultando em uma experiência não familiar para o usuário. A partir dessa constatação foi realizada a migração do código fonte.

A funcionalidade do *software* de simular a estratégia do usuário requer que haja um banco de dados salvo para que o algoritmo possa iterar a execução do fluxograma dia a dia no período definido pelo usuário. Para a criação de tal banco de dados foi utilizado os dados contidos no *website Yahoo! Finance*, que dispõe de uma opção de *download* dos dados históricos em formato *CSV*.

Conclui-se, portanto, que a contribuição do *software* descrito neste trabalho é o auxílio do investidor, seja seu nível de experiência iniciante ou experiente, na tomada de decisões no mercado financeiro, sempre fundamentadas nos resultados das simulações, e com a possibilidade de criar estratégias em uma ferramenta que funciona tanto em dispositivos móveis quanto em *Desktops*.

6.1 Sugestões de trabalhos futuros

Como sugestão para trabalhos futuros, podem ser feitas melhorias no *software* para aumentar seu potencial de utilidade para o investidor, como nos itens a seguir:

- Adicionar a função de otimização automática de parâmetros;
- Integrar um banco de dados de indicadores baseados em notícias para tomada decisões no fluxograma;
- Implementar mais indicadores para tomada de decisões no fluxograma (Ondas de Elliot, Bandas de Bollinger, Análise de Caixote).

REFERÊNCIAS

BRIAN, Joe, 2020. **13 Reasons why Flutter is the Future of Mobile App Development.** Disponível em: <https://dev.to/joebrain032/13-reasons-why-flutter-is-the-future-of-mobile-app-development-240n/>. Acesso em: maio 2022.

BROWNLEE, Jason, 2020. **Hyperparameter Optimization With Random Search and Grid Search.** Disponível em: <https://machinelearningmastery.com/hyperparameter-optimization-with-random-search-and-grid-search/>. Acesso em: dezembro 2022.

CEDRIC, Putten, 2022. **React Native.** Disponível em: <https://github.com/facebook/react-native#readme/>. Acesso em: outubro 2022.

CERIGO, Daniel, 2018. **On Why Gradient Descent is Even Needed.** Disponível em: <https://medium.com/@DBCerigo/on-why-gradient-descent-is-even-needed-25160197a635/>. Acesso em: maio 2022.

CODE ACADEMY, sem data. **What Is an IDE?** Disponível em: <https://www.codecademy.com/article/what-is-an-ide/>. Acesso em: outubro 2022.

EINVESTIDOR, 2021. **Em cinco anos, mais da metade dos *day traders* perdeu dinheiro na Bolsa.** Disponível em: <https://einvestidor.estadao.com.br/comportamento/day-*traders*-perdem-dinheiro-na-bolsa/>. Acesso em: maio 2022.

ELVERS, Alexander, 2019. **Hyperparameter Optimization using Random Search.** Disponível em: <https://commons.wikimedia.org/wiki/File:Hyperparameter_Optimization_using_Random_Search.svg/>. Acesso em: maio 2022.

FANTASTICO, 2021. ***Day trade*: prática explode no Brasil; veja alertas e cuidados para não perder dinheiro.** Disponível em: <https://globoplay.globo.com/v/9227160/>. Acesso em: maio 2022.

FERNANDES, Alexandre. **Estratégias Operacionais De Análise Técnica De Ações**. [*S. l.*]: Palex, 2014. 351 p. ISBN 859170911X.

FIGMA INC, 2016. **FIGMA**. Versão 2022. Disponível em: <https://www.figma.com/>. Acesso em: outubro 2022.

FLUTTER, sem data. **Introduction to widgets.** Disponível em: <https://docs.flutter.dev/development/ui/widgets-intro/>. Acesso em: maio 2022.

GOOGLE CLOUD PLATFORM, sem data. **Cloud Storage.** Disponível em: <https://console.cloud.google.com/storage/>. Acesso em: outubro 2022.

HUNTER, Ted; PORTER, Steven, 2018. **Google Cloud Platform for Developers.** Disponível em: <https://packtpub.com/product/google-cloud-platform-for-developers/9781788837675/>. Acesso em: maio 2022.

IKAMUSUMEFAN, 2015. **Mathematical optimization.** Disponível em: <https://en.wikipedia.org/wiki/Mathematical_optimization/>. Acesso em: maio 2022.

IG, 2022. **10 trading indicators every trader should know.** Disponível em: <https://www.ig.com/en/trading-strategies/10-trading-indicators-every-trader-should-know-190604/>. Acesso em: dezembro 2022.

INVESTOPEDIA, 2021. **What Is a Line Chart?** Disponível em: <https://investopedia.com/terms/l/linechart.asp/>. Acesso em: maio 2022.

INVESTOPEDIA, 2022. **Technical Analysis: What It Is and How to Use It in Investing.** Disponível em: <https://www.investopedia.com/terms/t/technicalanalysis.asp/>. Acesso em: dezembro 2022.

JAIN, Ayusch, 2020. **How to Use the Provider Pattern in Flutter.** Disponível em: <https://freecodecamp.org/news/provider-pattern-in-flutter/>. Acesso em: maio 2022.

KOTAK SECURITIES, 2005. **Understanding Technical Stock Charts & their Types.** Disponível em: <https://www.kotaksecurities.com/ksweb/share-market/what-are-stock-charts/>. Acesso em: dezembro 2022.

KUPRENKO, Vitaly, 2019. **Flutter vs React Native — Comparing the Features of Each Framework**. Disponível em: <https://levelup.gitconnected.com/flutter-vs-react-native-comparing-the-features-of-each-framework-f61bfd146a90/>. Acesso em: outubro 2022.

MASUR, 2007. **Elliott wave principle.** Disponível em: <https://en.wikipedia.org/wiki/Elliott_wave_principle/>. Acesso em: maio 2022.

MCMORRIS, Jeff, 2019. **Flutter Auth with Firebase Example.** Disponível em: <https://jeffmcmorris.medium.com/flutter-auth-with-firebase-example-96b64375fff3/>. Acesso em: maio 2022.

METAQUOTES SOFTWARE, 2009. **Metatrader**. Versão 4. Disponível em: <https://www.metatrader4.com/>. Acesso em: novembro 2022.

MICROSOFT, 2015. **Visual Studio Code**. Versão 1.72. Disponível em: <https://code.visualstudio.com/>. Acesso em: outubro 2022.

MORAES, André. **Se Afastando Da Manada**: Estratégias Para Vencer No Mercado De Ações. [*S. l.*]: Ipsis, 2016. 151 p. ISBN 8598741930.

NOVA ESCOLA, 2009. **O que faz e como surgiu a bolsa de valores?** Disponível em: <https://novaescola.org.br/conteudo/2388/o-que-faz-e-como-surgiu-a-bolsa-de-valores/>. Acesso em: maio 2022.

PERFORCE, 2021. **What Is Google Cloud Platform?** Disponível em: <https://www.perforce.com/blog/vcs/what-google-cloud-platform/>. Acesso em: maio 2022.

PORTAL DO *TRADE*R, 2021. **Ondas de Elliott: Entenda O que são e Como Utilizar na Análise Técnica.** Disponível em: <https://portaldo*trade*r.com.br/blog/analise-tecnica/ondas-de-elliott-como-identificar/>. Acesso em: maio 2022.

REACT NATIVE, 2022. **React Native.** Disponível em: <https://reactnative.dev/>. Acesso em: dezembro 2022.

REIS, Tiago, 2019. **Pregão viva voz: conheça como era o funcionamento da bolsa no passado.** Disponível em: <https://suno.com.br/artigos/pregao-viva-voz/>. Acesso em: maio 2022.

SANDE, Jonahan, 2020. **State Management With Provider.** Disponível em: <https://raywenderlich.com/6373413-state-management-with-provider/>. Acesso em: maio 2022.

SUNO, 2018. **Bandas de Bollinger: saiba como funciona essa ferramenta de análise.** Disponível em: <https://suno.com.br/artigos/bandas-de-bollinger/>. Acesso em: maio 2022.

SWING *TRADING* WITH *CANDLESTICKET* PATTERNS, 2017. **Are All *Candlesticket* Patterns Worthy to *Trade*?** Disponível em: <https://swing*trading*with*candlesticket*patterns.com/2017/01/18/*candlesticket*-patterns-worthy-to-*trade*/>. Acesso em: maio 2022.

TRADINGVIEW INC. **TradingView**. [*S. l.*], 11 set. 2011. Disponível em: https://tradingview.com. Acesso em: 1 maio 2022.

UNIVERSO DO FOREX, 2010. **Linha de Tendência.** Disponível em: <https://universodoforex.webnode.com.br/linha%20de%20tendência/>. Acesso em: maio 2022.

WASLAWICK, Raul. **Metodologia de Pesquisa para Ciência da Computação**: Campus, 2014. 21 p. ISBN 978-85-352-7783-8.

YAHOO! FINANCE, 2022. **Petróleo Brasileiro S.A.** Disponível em: <https://finance.yahoo.com/quote/PETR4.SA/history?p=PETR4.SA/>. Acesso em: outubro 2022.

YERRIC, Damian, 2002. **Mathematical optimization.** Disponível em: <https://en.wikipedia.org/wiki/Mathematical_optimization/>. Acesso em: maio 2022.

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 Hérmeson Barroso Freitas do Curso de Ciência da Computação, matrícula 20181002802270, telefone: (62) 99163-8279 e-mail hermeson.freitas@yahoo.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 Ciência da Computação, 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, 14 de Agosto de 2022.

Assinatura do(s) autor(es): Hérmeson Barroso Freitas

Nome completo do autor: Hérmeson Barroso Freitas

Assinatura do professor-orientador:

Nome completo do professor-orientador: Angélica da Silva Nunes