



**Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis
Campus "José Santilli Sobrinho"**

CRISTHIAN NUNES DIAS

APLICATIVO PARA CONTROLE FINANCEIRO UTILIZANDO FLUTTER

**Assis/SP
Ano 2020**



**Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis
Campus "José Santilli Sobrinho"**

CRISTHIAN NUNES DIAS

APLICATIVO PARA CONTROLE FINANCEIRO UTILIZANDO FLUTTER

Exame de Qualificação como requisito para o Trabalho de Conclusão de Curso apresentado ao curso de Análise e Desenvolvimento de Sistemas do Instituto Municipal de Ensino Superior de Assis – IMESA e a Fundação Educacional do Município de Assis – FEMA, como requisito parcial à obtenção do Certificado de Conclusão.

Orientando (a): Cristhian Nunes Dias
Orientador (a): Dr. Almir Rogério Camolesi

Assis/SP
Ano 2020

APLICATIVO PARA CONTROLE FINANCEIRO UTILIZANDO FLUTTER

CRISTHIAN NUNES DIAS

Exame de Qualificação como requisito para o Trabalho de Conclusão de Curso apresentado ao Instituto Municipal de Ensino Superior de Assis, como requisito do Curso de Graduação, avaliado pela seguinte comissão examinadora:

Orientador: _____
Dr. Almir Rogério Camolesi

Examinador: _____
Dr. Osmar Aparecido Machado

Assis/SP
Ano 2020

DEDICATÓRIA

Dedico este trabalho primeiramente a Deus que está sempre me dando forças para continuar apesar de todas as dificuldades, a minha mãe Vera Lúcia Nunes Dias e meu pai Aparecido Ferreira Dias que sempre me cobraram e me ajudaram para estudar e ter uma vida melhor. Da mesma forma dedico-o para meu amigo, professor e orientador Almir Rogério Camolesi por sempre confiar no meu potencial, me ensinar tudo que sei hoje e me dar forças para continuar.

AGRADECIMENTOS

Agradeço a Deus por sempre estar do meu lado, me ajudando e dando forças para continuar e por me dar condições físicas e psicológicas para trabalhar e continuar estudando. Por toda a experiência vivida e a sabedoria que me foi dada.

Agradeço aos meus pais Aparecido Ferreira Dias e Vera Lúcia Nunes Dias por sempre me ajudar a conquistar tudo que tenho hoje, e por me dar privilégios que foram cruciais para o término desse trabalho. Pelo apoio incondicional que me proporcionaram desde sempre.

Agradeço aos meus amigos de trabalho e de faculdade que me ajudaram nesse caminho, pela paciência e pela confiança.

Agradeço principalmente ao meu amigo, professor e orientador Dr. Almir Rogério Camolesi, por sempre me ajudar a crescer profissionalmente e individualmente, por estar comigo nessa caminhada estressante, e toda vez utilizando seu vasto conhecimento para nos ensinar e motivar para melhorar. Pois trata-se de uma pessoa que quer o bem de seus alunos acima de tudo.

Agradeço a você que está lendo meu trabalho.

“Não espere o futuro mudar tua vida, porque o futuro é a consequência do presente. ”

Racionais Mc's

RESUMO

As empresas hoje precisam de desenvolvimento produtivo e com uma ótima qualidade de software, e isso não é nenhuma novidade. Dito isso, foram lançadas várias tecnologias para auxiliar, porém a que será falada nesse trabalho é o Flutter, que através dele será desenvolvido um aplicativo de controle financeiro.

O objetivo do presente trabalho é o controle financeiro de forma simples e objetiva com ênfase no desenvolvimento com a ferramenta Flutter, de forma que, os leitores inspirem-se por essa nova tecnologia da Google.

Palavras-chave: Flutter, Dart, MongoDB, Desenvolvimento Híbrido, UML, Astah Community, Visual Studio Code, IntelliJ IDEA, Java

ABSTRACT

Companies today need productive development and excellent software quality, and this is nothing new. That said, several technologies were launched to assist, but the one that will be talked about in this work is the Flutter, which through it will be developed a financial control application.

The objective of the present work is the financial control in a simple and objective way with an emphasis on development with the Flutter tool, so that, be inspired by this new technology from Google.

LISTA DE ILUSTRAÇÃO

Figura 1: Árvore de Widgets	17
Figura 2: Diagrama da visão geral do sistema Flutter	18
Figura 3: Interface gráfica do container vermelho.....	20
Figura 4: Interface gráfica com textos em linha	21
Figura 5: Imagem ilustrativa de hierarquia de Widgets	22
Figura 6: Interface gráfica dos textos em cada linha	22
Figura 7: Código inicial	23
Figura 8: Interface gráfica do aplicativo	23
Figura 9: Código de inserção da AppBar	24
Figura 10: Interface gráfica da AppBar	24
Figura 11: Código de inserção do ícone menu	25
Figura 12: Interface gráfica da inserção do ícone	25
Figura 13: Código de alteração de cor do ícone menu	26
Figura 14: Interface gráfica da alteração de cor do ícone menu	26
Figura 15: Código de alteração do ícone	27
Figura 16: Interface gráfica da alteração do ícone.....	27
Figura 17: Código da chamada de função de campo de texto.....	28
Figura 18: Interface gráfica da inserção do campo de texto	28
Figura 19: Código da função para inserir campo de texto	28
Figura 20: Código da inserção do botão flutuante	29
Figura 21: Interface gráfica da inserção do botão flutuante	29
Figura 22: Código da inserção da barra inferior de navegação	30
Figura 23: Interface gráfica da inserção da barra inferior de navegação	30
Figura 24: Código da ancoragem do botão com a barra de navegação	31

Figura 25: Interface gráfica da ancoragem do botão com a barra de navegação	31
Figura 26: Exemplo da interface gráfica da lista de pagamentos	37
Figura 27: Diagrama de Caso de Uso – Geral.....	39
Figura 28: Diagrama de Caso de Uso – Fazer login.....	40
Figura 29: Diagrama de Caso de Uso – Inserir receita ou despesa	42
Figura 30: Diagrama de Caso de Uso – Exibir transações	42
Figura 31: Diagrama de Caso de Uso – Notifica pagamento.....	43
Figura 32: Diagrama de Caso de Uso – Notifica vencimento	44
Figura 33: Diagrama de classe	45
Figura 34: Diagrama de Atividade para inserir despesa ou receita	47
Figura 35: Arquitetura do projeto	48
Figura 36: Fluxograma da estrutura da API Rest.....	50
Figura 37: Método da classe de serviço de contas.....	51
Figura 38: Interface gráfica de acesso.....	52
Figura 39: Interface gráfica da página inicial do aplicativo.....	52
Figura 40: Interface gráfica do menu do aplicativo	52
Figura 41: Interface gráfica da inserção de pagamento.....	53
Figura 42: Interface gráfica da inserção de recebimento	53
Figura 43: Interface gráfica da lista de pagamentos	53
Figura 44: Interface gráfica da lista de recebimentos	54

SUMÁRIO

1. INTRODUÇÃO	12
1.1. OBJETIVOS	13
1.2. JUSTIFICATIVA	13
1.3. MOTIVAÇÃO.....	14
1.4. PERSPECTIVA DE CONTRIBUIÇÃO.....	14
1.5. METODOLOGIA.....	14
1.6. PÚBLICO ALVO	15
1.7. ESTRUTURA DO TRABALHO.....	15
2. TECNOLOGIA FLUTTER.....	17
2.1. ARQUITETURA FLUTTER	18
2.2. WIDGETS.....	19
2.3. DESENVOLVIMENTO COM FLUTTER.....	23
3. CONCEITOS E TECNOLOGIAS COMPLEMENTARES UTILIZADAS NO TRABALHO	32
3.1. UML.....	32
3.2. ASTAH COMMUNITY	32
3.3. DART.....	33
3.4. VISUAL STUDIO CODE.....	33
3.5. INTELLIJ IDEA	34
3.6. JAVA.....	34
3.7. SPRING BOOT	34
3.8. API REST	35
3.9. MONGODB	35
3.10. AWS AMAZON EC2.....	36
4. PROPOSTA DE MODELAGEM DO ESTUDO DE CASO.....	37
4.1. MAPA MENTAL.....	38
4.2. DIAGRAMA DE CASO DE USO	39
4.3. DIAGRAMA DE CLASSES.....	44
4.4. DIAGRAMA DE ATIVIDADE	46
5. DESENVOLVIMENTO	48

5.1.	DESENVOLVIMENTO BACK-END.....	49
5.2.	DESENVOLVIMENTO FRONT-END	50
5.3.	INTERFACE GRÁFICA	52
6.	CONSIDERAÇÕES FINAIS	55
7.	REFERÊNCIAS	56

1. INTRODUÇÃO

A tecnologia aprimora-se cada vez mais com o passar dos anos. O que antes era feito de forma manuscrita, hoje temos ferramentas na palma da mão que suprem nossas necessidades de maneira mais rápida e eficiente. Com essa grande evolução, conforme Paula (2017) tanto pessoas quanto empresas estão investindo em aplicações móveis para facilitarem o seu dia a dia.

Dito isso, a procura de aplicações fluídas para controle de gastos e planejamento financeiro está cada vez maior. Com base nessas observações, é proposto o desenvolvimento de uma aplicação para gerenciar tais transações.

No desenvolvimento, a administração do tempo é muito importante, pois se feito de maneira correta, a produtividade aumenta significativamente. Hoje no mercado há várias ferramentas para nos auxiliarem, e uma delas é o Flutter¹, que traz uma facilidade enorme em desenvolver aplicativos móveis de uma forma híbrida sem precisar retrabalhar no código.

Segundo Magalhães (2019) o Flutter tem seu próprio framework de processamento, o que quer dizer que ele não dependerá de nada específico de cada plataforma. Todos os efeitos que há no sistema operacional IOS² e Android³ estão incluídos na ferramenta. Nele também contém uma opção de visualização em segundos de todas alterações feitas sem precisar recompilar o aplicativo no celular, isso tudo de forma rápida e simples para o desenvolvimento da aplicação móvel de finanças.

Os aplicativos que se sobressaem no mercado hoje, são os com design mais trabalhados, com animações que chamam atenção e de uma usabilidade rápida. A Tecnologia Flutter contém tudo isso, sendo escalável, pois permite uma fácil manutenção de código fonte.

¹ <https://flutter.dev/> acessado em 19 nov. 2019.

² É um sistema operacional de dispositivos móveis da Apple.

³ É um sistema operacional baseado no núcleo Linux, sendo seu maior colaborador o Google.

1.1. OBJETIVOS

Um dos objetivos com o desenvolvimento e implementação desse trabalho é produzir por meio de um aplicativo híbrido, informações que auxiliará o usuário a fazer o controle monetário, pois servirá como suporte financeiro pessoal de diversas despesas pessoais, nele será contido as funcionalidades de inserção, remoção, exclusão e edição de dados de uma maneira simplificada para que não seja difícil de utilizar e muito menos demorado. Conterá também uma interface fluída, elegante, sempre priorizando o desempenho.

Além da aplicação móvel, também terá como um dos focos expor mais sobre a ferramenta Flutter, pois trata-se de uma tecnologia de fácil aprendizagem e com um ótimo desempenho por se utilizar comunicação nativa com o dispositivo.

1.2. JUSTIFICATIVA

Este trabalho se justifica pela necessidade de aprimorar a produtividade em programação móvel híbrida, utilizando todo o conhecimento adquirido pelo curso de Análise e Desenvolvimento de Sistema e juntamente com a indispensabilidade da aplicação de controle monetário pessoal. Como mencionado anteriormente, a aplicação será para o controle e planejamento financeiro, auxiliando no desenvolvimento rápido e fluído com um melhor aproveitamento de códigos.

Outra defesa para esse trabalho, dá-se pela ausência de informações para essa nova tecnologia, que está crescendo pelo decorrer dos anos. Essa ferramenta mencionada tem como finalidade principal o desenvolvimento híbrido de aplicações, que só necessita de uma linguagem de programação para ser feita, dito isso, com apenas um código, é possível instalar e executar em qualquer dispositivo móvel, que consequentemente acaba sendo obtido um aumento significativo na produtividade e ao mesmo tempo expondo seus pontos positivos e negativos para futuros desenvolvedores. Essa ferramenta será o Flutter que utiliza a linguagem de programação Dart⁴ para fins de obter ótimos resultados de desempenho na execução da aplicação e velocidade no seu desenvolvimento.

⁴ <https://dart.dev/> acessado em 07 abr. 2020.

1.3. MOTIVAÇÃO

A motivação para a produção desse trabalho surgiu da necessidade de gerar aplicações rápidas, com design elegante, com uma altíssima fluidez de animações, com código híbrido, executando tanto em IOS quanto em Android e para conclusão do curso de Análise e Desenvolvimento de Sistemas.

Além disso, toda a experiência que foi aprendida em engenharia de software teve um papel primordial para definição de todo o projeto, por se tratar de tecnologias que não usamos no curso, consegue-se desenvolver a aplicação, entretanto, o conceito utilizado foi o mesmo. E por fim, a maior motivação é aprender uma nova tecnologia e desenvolver uma aplicação que me ajudará a crescer profissionalmente, com a finalidade de não ser um trabalho em vão.

1.4. PERSPECTIVA DE CONTRIBUIÇÃO

Com esse trabalho, pretende-se mostrar como é rápido gerar interfaces gráficas profissionais, animações nativas da própria ferramenta, sem precisar digitar muitos códigos comparando com ferramentas concorrentes para geração do aplicativo financeiro.

Será contribuído para comunidade também as vantagens e desvantagens de utilizar a ferramenta Flutter para desenvolver aplicações.

Como a ferramenta é nova, a agregação para a área de estudo é alta, pois não há muito material disponível (artigos, livros, tutoriais, cursos, etc.) sobre o Flutter.

1.5. METODOLOGIA

O projeto será desenvolvido com uma análise realizada por diversas pessoas que tem interesse de fazer o controle financeiro. A metodologia da análise e a própria implementação será feita orientada a objetos. Será feito todo em cima de uma engenharia de software. Este estudo de caso será feito com base em pessoas que precisam se planejar

financeiramente, com agregação de um levantamento de requisitos, ou seja, as funcionalidades que necessitam em uma aplicação para os ajudar.

Para fazer os casos de usos, diagrama de atividades, diagrama de classes será usado a linguagem UML com base no programa Astah⁵, nele consegue-se realizar trabalhos com uma agilidade maior por ser simples e objetivo. Para o desenvolvimento do aplicativo em geral será utilizado Flutter que tem Dart como a linguagem principal. O banco de dados será feito no MongoDB⁶, que foi hospedado em nuvem utilizando a tecnologia de serviço computacional AWS Amazon⁷. Para acessar os dados do banco e enviar para o aplicativo será utilizado Spring Boot⁸ que também estará em nuvem. E para escrever os códigos será utilizado o programa Visual Studio Code⁹.

1.6. PÚBLICO ALVO

O aplicativo que foi desenvolvido nesse trabalho tem como público principal qualquer pessoa que deseja manter seu planejamento financeiro de uma forma eficiente e prática. Visa também alcançar novos profissionais na área de desenvolvimento híbrido que pretendem-se iniciar na programação em Flutter ou em tecnologias similares. Dito isso, servirá tanto para pessoas que não sabem programação e querem o aplicativo para seu controle pessoal, quanto para desenvolvedores que querem aprender mais sobre essa ferramenta recente ou aprender mais sobre desenvolvimento híbrido.

1.7. ESTRUTURA DO TRABALHO

Este trabalho está dividido em capítulos. No primeiro capítulo é definido por introdução, objetivos, justificativas, motivação, perspectiva de contribuição, metodologia, público alvo e estrutura do trabalho.

⁵ <https://astah.net/> acessado em 07 abr. 2020.

⁶ <https://www.mongodb.com/> acessado em 27 jul. 2020.

⁷ <https://aws.amazon.com/pt/> acessado em 27 jul. 2020.

⁸ <https://spring.io/projects/spring-boot> acessado em 27 jul. 2020.

⁹ <https://code.visualstudio.com/> acessado em 07 abr. 2020.

Segundo capítulo explica sobre a tecnologia principal desse trabalho, o Flutter. Também com ênfase em explicar como funciona a arquitetura do mesmo, que foco na profundidade de como funciona o desenvolvimento.

O terceiro capítulo explana sobre os conceitos e tecnologias que foram utilizadas para o desenvolvimento desse trabalho.

O quarto capítulo é apresentado as modelagens do estudo de caso do projeto, para que o leitor tenha um maior entendimento referente ao foco do projeto.

Desenvolvimento do aplicativo é o quinto capítulo, onde é apresentado as camadas do front-end¹⁰, back-end¹¹ e as interfaces gráficas do aplicativo com suas respectivas explicações de funcionamento.

Capítulo seis é apresentado as considerações finais do trabalho.

Por fim, o capítulo sétimo consta as referências usadas nesse trabalho.

¹⁰ Front-end é a parte visual da aplicação

¹¹ Back-end é tudo que vai por trás de uma aplicação, como por exemplo regras de negócios

2. TECNOLOGIA FLUTTER

Flutter é um framework da Google que inicialmente foi criado para desenvolvimento de aplicativos móveis, porém, atualmente se expande para desenvolvimentos web e aplicações para sistemas operacionais. Sua produtividade é excelente, pois com um único código é possível executar em todas as plataformas mencionadas anteriormente.

Segundo Santana (2019) o Flutter funciona na maior parte utilizando *Widgets*, que são componentes que se montam em uma árvore e que pode conter outros *Widgets* como filhos, dito isso, acaba contendo uma hierarquia. A renderização desses *Widgets* acontecem conforme a árvore é montada. Para melhorar o entendimento do item mencionado nesse parágrafo, veja a imagem a seguir:

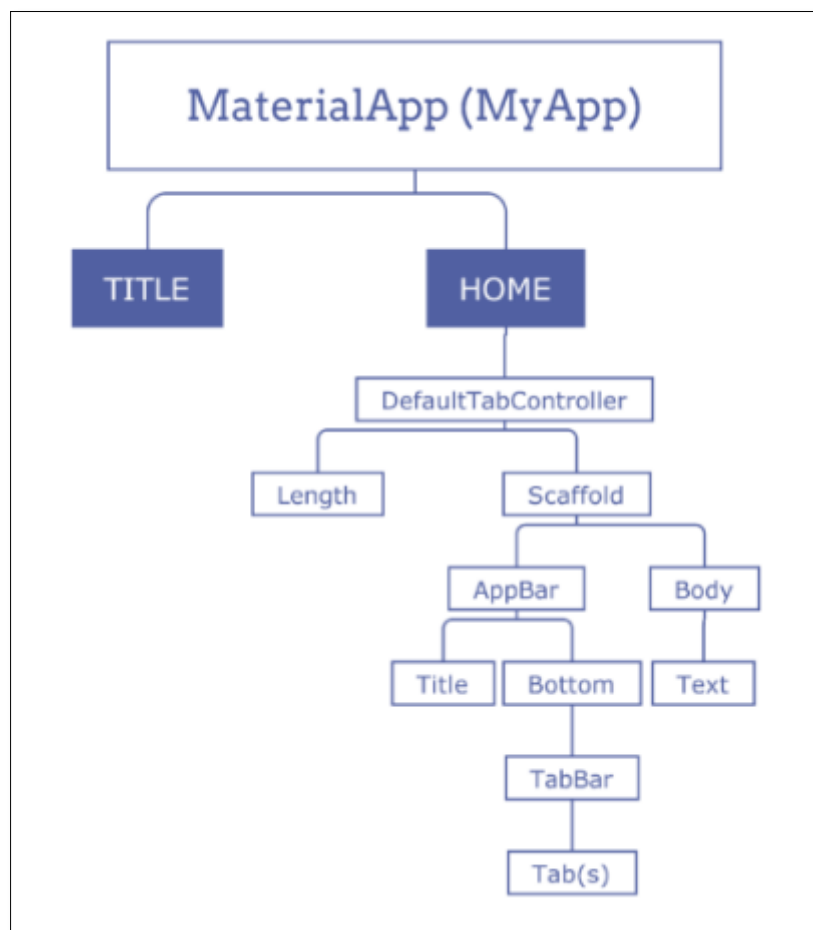


Figura 1: Árvore de Widgets

Todos esses itens da árvore são *widgets* que dependem de outros *widgets* para funcionarem e criarem uma interface para o usuário.

2.1. ARQUITETURA FLUTTER

Para o maior entendimento da tecnologia abordada nesse trabalho, nessa seção será exibida a arquitetura do Flutter. A figura 4 abaixo mostra a visão geral do sistema.

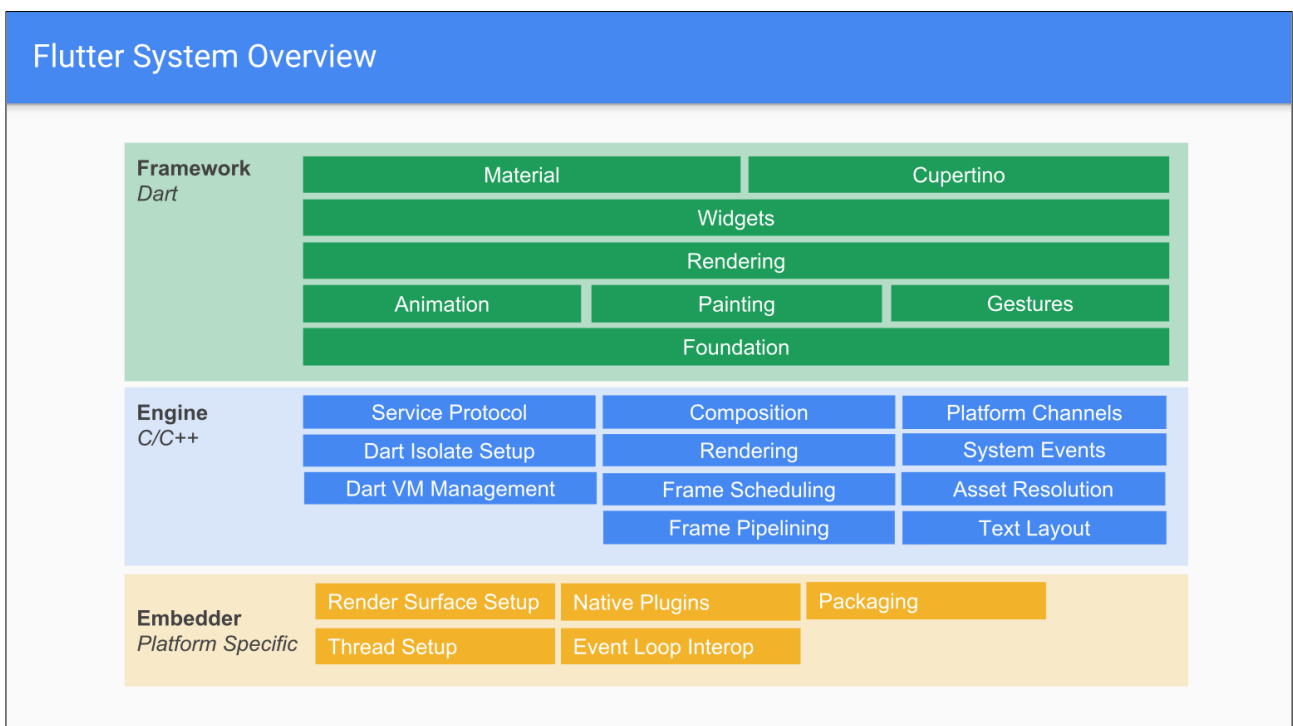


Figura 2: Diagrama da visão geral do sistema Flutter

Após a leitura da figura acima, tira-se a conclusão que o Flutter tem 3 camadas para seu funcionamento. A primeira camada em verde é toda a estrutura do Flutter escrita em Dart, onde será executado o trabalho do desenvolvedor, essa parte é muito importante que seja bem entendida para quem for desenvolver. A segunda camada em azul, é onde o desenvolvedor não precisa se preocupar, pois é o núcleo da ferramenta, onde fica o cérebro do mesmo. Nessa camada é onde fica designada para resolver toda parte gráfica para cada sistema, que é o ponto diferencial do Flutter comparada com outras tecnologias concorrentes. Na terceira camada em amarelo é onde fica uma ponte dinâmica entre a

ferramenta e o sistema do dispositivo, sem a necessidade de ter uma ponte específica para cada sistema.

2.2. WIDGETS

Tudo em Flutter é lido como um *Widget*, como se fosse um recipiente dentro de outros recipientes. A finalidade dessa estratégia segundo Magalhães (2019) é para não haver herança e sim composições a fins de facilitar reaproveitamento de componentes já existentes.

Vamos imaginar que temos que criar um aplicativo que cada linha deve haver um texto, que seria “Texto 1”, “Texto 2” e “Texto 3”. De uma forma simples, o primeiro passo do aplicativo será iniciar com um Container com fundo vermelho, pois nele que seria armazenado as informações de texto. Obrigatoriamente precisa haver um *Widget* para iniciar o aplicativo. Na figura abaixo será a simulação de forma didática de como ficaria esse primeiro processo.



Figura 3: Interface gráfica do container vermelho

O Container pode conter qualquer altura, largura, imagem de fundo e cor, até mesmo gradiente de cores. Nele há uma propriedade que se chama *child*, onde é inserido outro *Widget* para fazer parte do aplicativo (inserção na árvore de *Widgets*), que no nosso exemplo seria os textos, porém eu não posso inserir os 3 textos em *child*, porque essa propriedade aceita somente um *Widget* e para cada texto precisamos de um *Widget*. Para solucionar esse problema poderíamos inserir um *Widget* chamado *Row* no *child* de Container onde nele é aceito mais de um *Widget*, nele há a propriedade chamada *children*, que resolveria nosso problema inicial, conforme figura a seguir:



Figura 4: Interface gráfica com textos em linha

Percebe-se pela explicação e imagem acima, que tudo que tiver *child* pode haver somente um *Widget* e o que tiver *children* pode ser inserir mais de um, porém no nosso exemplo os textos deveriam ficar um em cada linha e para isso, ao invés de chamar de primeiro momento o *Widget Row* que trabalha na horizontal, podemos chamar o *Widget Column* que trabalha na vertical e também tem a propriedade *children* e para cada uma pode ser inserido uma *Row* e dentro de cada *Row* inserir um ou mais Textos. Segue figura abaixo para entendimento onde laranja claro é o Container, azul claro é a *Column* e as *Row* são os verdes:

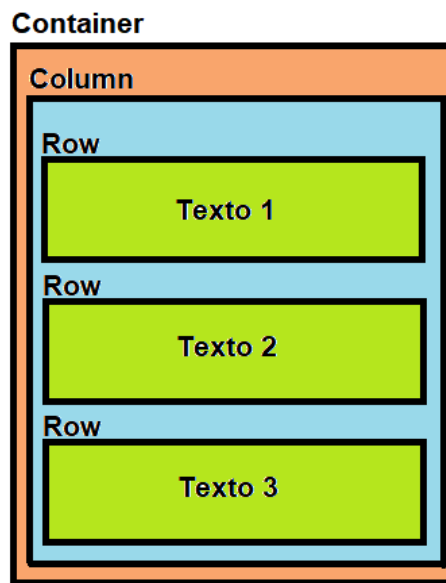


Figura 5: Imagem ilustrativa de hierarquia de Widets

E com isso finalizaríamos nosso exemplo, e ficaria como a figura a seguir:

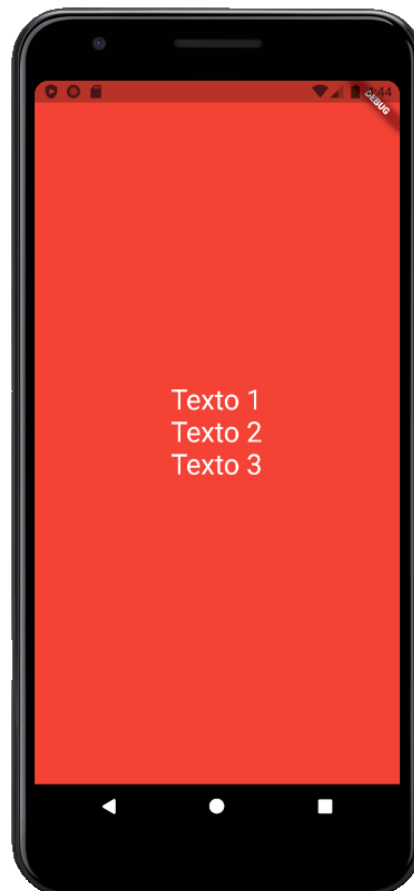


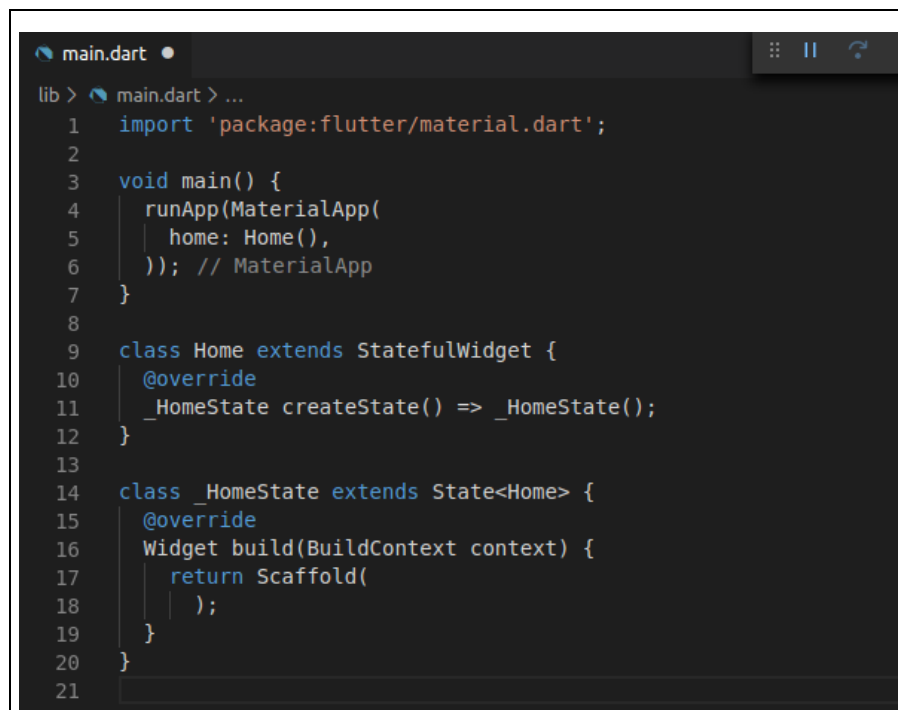
Figura 6: Interface gráfica dos textos em cada linha

Conclui-se com esse exemplo que tudo que há na interface gráfica do aplicativo deve ser um *Widget* e para cada *Widget* há suas propriedades específicas, sendo assim, o desenvolvedor terá que analisar qual melhor *Widget* para o caso atual dele.

2.3. DESENVOLVIMENTO COM FLUTTER

Nessa seção será apresentada facilidade de desenvolvimento utilizando Flutter, passo a passo como fazer uma interface gráfica de inserção de nome, sobrenome e idade com botão de adicionar, porém sem funcionalidades.

Do lado esquerdo ficará o código da aplicação e do lado direito a interface gráfica que é referente ao código lateral, isso será padrão para todas imagens abaixo dentro dessa seção.



```
main.dart
lib > main.dart > ...
1  import 'package:flutter/material.dart';
2
3  void main() {
4    runApp(MaterialApp(
5      home: Home(),
6    )); // MaterialApp
7  }
8
9  class Home extends StatefulWidget {
10   @override
11   _HomeState createState() => _HomeState();
12 }
13
14 class _HomeState extends State<Home> {
15   @override
16   Widget build(BuildContext context) {
17     return Scaffold(
18       );
19   }
20 }
21
```

Figura 7: Código inicial

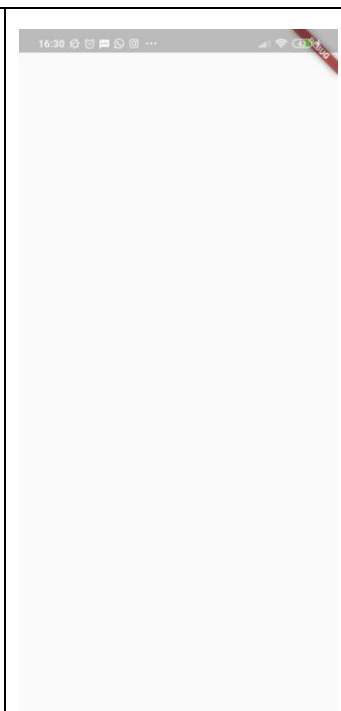


Figura 8: Interface gráfica do aplicativo

Como mostrada na figura acima (Código inicial), percebe-se que temos um método **main** na aplicação onde chama outro método **Home** que cria um estado para o método **_HomeState**. Esse método que é criado o estado retorna os *Widgets* que desenharam o aplicativo. Como não há nada retornando para desenhar no aplicativo, o próprio fica em branco.

```
class _HomeState extends State<Home> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("Test app"),
      ), // AppBar
    ); // Scaffold
  }
}
```

Figura 9: Código de inserção da AppBar

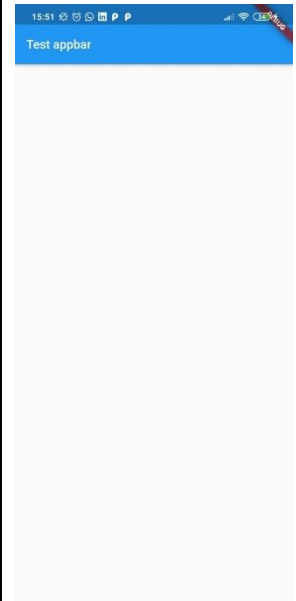


Figura 10: Interface gráfica da AppBar

Para criação de uma AppBar¹² é de extrema obrigatoriedade ter como pai um *Scaffold*, que é um *Widget*, tudo em Flutter é feito com *Widget* dentro de outro *Widget*. Passamos como parâmetro dentro do *AppBar* um texto que será mostrado na barra. Para ficar claro o entendimento da velocidade do desenvolvimento, essa *AppBar* foi criada em 26 segundo, rodando instantaneamente no aplicativo em depuração.

¹² AppBar: Sua tradução é Barra de Aplicativos.

```
class _HomeState extends State<Home> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("Test appbar"),
        backgroundColor: Colors.green,
        leading: IconButton(
          onPressed: (){},
          icon: Icon(Icons.menu),
        ), // IconButton
      ), // AppBar
    ); // Scaffold
  }
}
```

Figura 11: Código de inserção do ícone menu

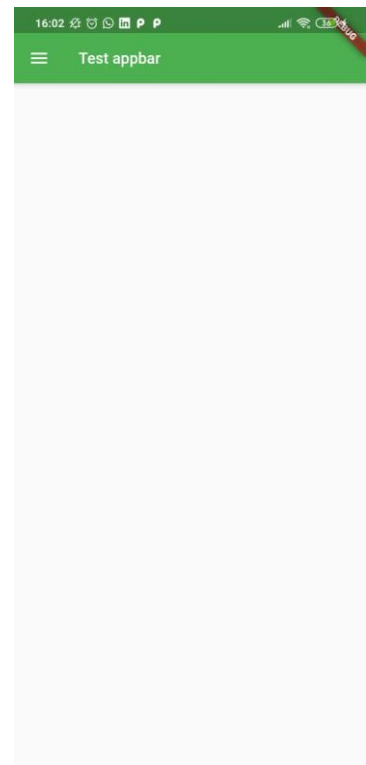


Figura 12: Interface gráfica da inserção do ícone

Para deixar um aplicativo mais atraente, foi inserido um ícone de menu na barra do aplicativo. Chamamos um *leading* que é propriedade da própria *AppBar* e colocamos o ícone de botão com um método vazio e chamando a imagem do menu.

```
@override
_HomeState createState() => _HomeState();
}

class _HomeState extends State<Home> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("Test appbar"),
        backgroundColor: Colors.green,
        leading: IconButton(
          onPressed: (){},
          icon: Icon(Icons.menu),
          color: Colors.orange,
        ) // IconButton
      ), // AppBar
    ); // Scaffold
  }
}
```

Figura 13: Código de alteração de cor do ícone menu

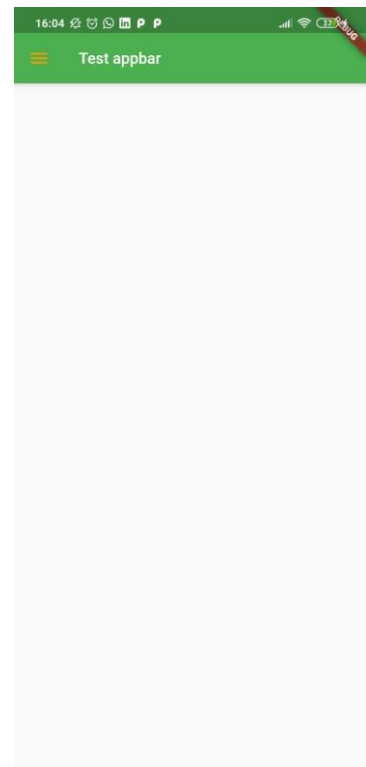


Figura 14: Interface gráfica da alteração de cor do ícone menu

Na figura de código acima, mostra-se o código de troca da cor do ícone. No quesito cor, o Flutter contém várias cores definidas, porém se não achar a ideal, tem a opção de inserir o código rgb ou hexadecimal da cor requerida.

```
class _HomeState extends State<Home> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("Test appbar"),
        backgroundColor: Colors.green,
        leading: IconButton(
          onPressed: (){},
          icon: Icon(Icons.save),
        ) // IconButton
      ), // AppBar
    ); // Scaffold
  }
}
```

Figura 15: Código de alteração do ícone

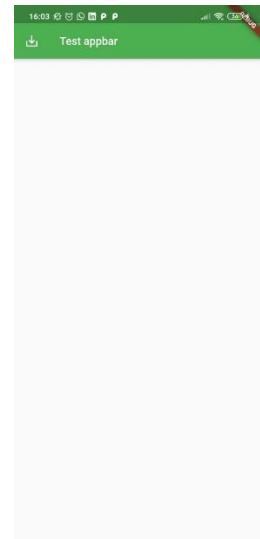


Figura 16: Interface gráfica da alteração do ícone

Na figura acima é exibida algumas das opções de ícones que começam com a letra “s”. No próprio site da ferramenta há todas as imagens dos ícones e seus nomes para poder ser inserido em aplicações.

```

appBar: AppBar(
  title: Text("Test appbar"),
  backgroundColor: Colors.black,
  leading: IconButton(
    onPressed: (){},
    icon: Icon(Icons.menu),
    color: Colors.white,
  ) // IconButton
), // AppBar
body: Container(
  padding: EdgeInsets.only(left: 40, right: 40, top: 10),
  child: Column(
    children: <Widget>[
      retornaTextField("Nome"),
    ], // <Widget>[]
  ), // Column
), // Container
); // Scaffold
}

```

Figura 17: Código da chamada de função de campo de texto

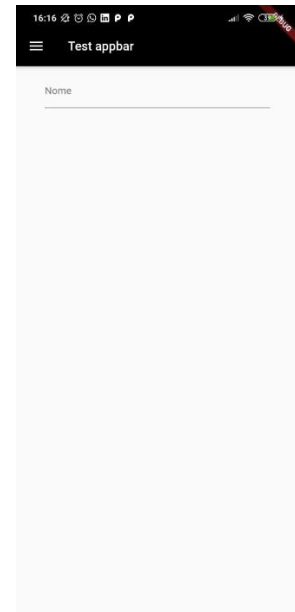


Figura 18: Interface gráfica da inserção do campo de texto

Para inserir uma caixa de texto, foi chamado uma função com o parâmetro do tipo texto e que retorna um *Widget TextField* que é o que foi mostrado nas figuras acima.

```

Widget retornaTextField(String descricao) {
  return TextField(
    decoration: InputDecoration(
      hintText: descricao,
    )); // InputDecoration // TextField
}

```

Figura 19: Código da função para inserir campo de texto

Na função **retornaTextField** é retornado um *Widget TextField*. Dentro desse *Widget* é inserido uma decoração de dica de texto, com finalidade de auxiliar o usuário a saber o que inserir no campo.

```

        retornaTextField("Nome"),
        SizedBox(height: 20.0,),
        retornaTextField("Sobrenome"),
        SizedBox(height: 20.0,),
        retornaTextField("Idade"),
        SizedBox(height: 20.0,),
      ], // <Widget>[]
    ), // Column
  ), // Container
  floatingActionButton: FloatingActionButton(
    child: Icon(Icons.add),
    onPressed: (){},
    backgroundColor: Colors.black,
  ), // FloatingActionButton

```

Figura 20: Código da inserção do botão flutuante

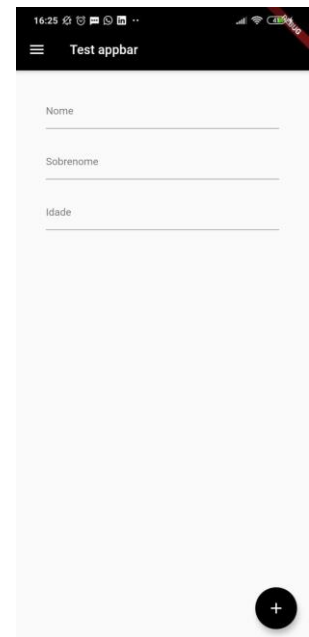


Figura 21: Interface gráfica da inserção do botão flutuante

Com o intuito de deixar um botão padronizado na tela, foi criado um botão flutuante que independentemente do tamanho da tela e de informações que nela contém ele sempre ficará fixo. O botão poderá ficar no lado direito, esquerdo ou centro, com qualquer ícone de qualquer cor.

```

        retornaTextField("Nome"),
        SizedBox(height: 20.0,),
        retornaTextField("Sobrenome"),
        SizedBox(height: 20.0,),
        retornaTextField("Idade"),
        SizedBox(height: 20.0,),
      ], // <Widget>[]
    ), // Column
  ), // Container
  floatingActionButton: FloatingActionButton(
    child: Icon(Icons.add),
    onPressed: (){},
    backgroundColor: Colors.black,
  ), // FloatingActionButton
  bottomNavigationBar: BottomAppBar(
    color: Colors.black,
    child: Container(
      height: 50.0,
    ), // Container
  ), // BottomAppBar
]; // Scaffold
}
}

```

Figura 22: Código da inserção da barra inferior de navegação

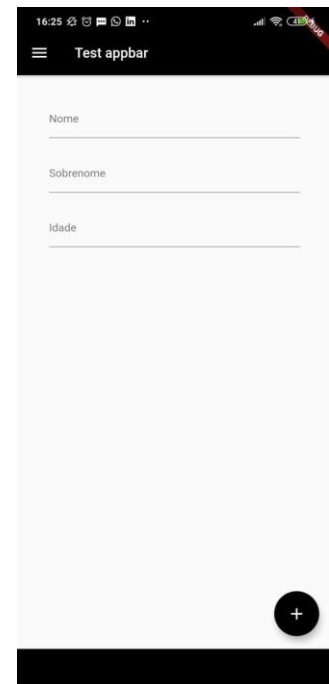


Figura 23: Interface gráfica da inserção da barra inferior de navegação

Para deixar o aplicativo de exibição mais elegante, foi inserido como mostra nas figuras acima uma barra de navegação inferior, nela poderá também existir ícones e funcionalidades, porém com a inserção dessa barra o botão acabou subindo e ficando um pouco deslocado, isso tem resolução e será exibido na próxima imagem.

```

return TextField("Sobrenome"),
      SizedBox(height: 20.0),
      return TextField("Idade"),
      SizedBox(height: 20.0),
    ], // <Widget>[]
  ), // Column
), // Container
floatingActionButton: FloatingActionButton(
  child: Icon(Icons.add),
  onPressed: () {},
  backgroundColor: Colors.black,
), // FloatingActionButton
bottomNavigationBar: BottomAppBar([
  color: Colors.blueGrey,
  child: Container(
    height: 50.0,
  ), // Container
]), // BottomAppBar
floatingActionButtonLocation: FloatingActionButtonLocation.centerDocked,
); // Scaffold
}
}

```

Figura 24: Código da ancoragem do botão com a barra de navegação

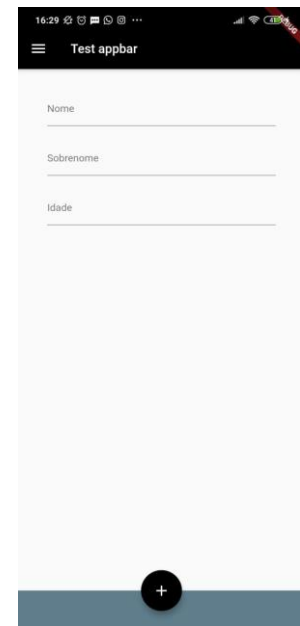


Figura 25: Interface gráfica da ancoragem do botão com a barra de navegação

Para finalizar o aplicativo de exibição, foi resolvido o problema da barra inferior com o botão flutuante fazendo a chamada do *FloatingActionByttonLocation* e chamando o **centerDocked** para se misturar com a barra inferior.

O desenvolvimento dessa simples tela foi feito em 5 minutos e 49 segundos. Percebe-se a alta velocidade de desenvolvimento da ferramenta Flutter para uma tela simples de inserção de dados. Acredita-se que com pouco aprendizado qualquer pessoa que tem o mínimo de conceito poderá fazer o aplicativo sem nenhum problema.

3. CONCEITOS E TECNOLOGIAS COMPLEMENTARES UTILIZADAS NO TRABALHO

Nessa seção será explanada sobre as tecnologias que foram utilizadas para desenvolver esse trabalho por completo.

A principal tecnologia estudada é o Flutter, que contém a linguagem de programação Dart, pois trata-se de uma linguagem de compilação eficaz. Firebase terá como responsabilidade ser o banco de dados do aplicativo, pois é um banco em nuvem que promete ser rápido. Por fim, para entendimento melhor da aplicação, será utilizado a linguagem UML feita no programa Astah Community.

Para a compreensão mais aprofundada das tecnologias citadas anteriormente, será explicada detalhadamente cada uma abaixo:

3.1. UML

Durante o trabalho de modelagem da aplicação serão desenvolvidos os diagramas conforme a linguagem UML. Para Portalgsi (2011?) UML é uma linguagem de modelagem que é utilizada para fazer as modelagens de objetos do mundo real. Essa linguagem é para auxiliar no desenvolvimento de todos tipos de sistemas para facilitar o entendimento do mesmo em forma de “desenhos”.

3.2. ASTAH COMMUNITY

Por fim, para auxiliar na produção de diagramas da linguagem UML será utilizada a ferramenta Astah Community. E conforme Lima (2016):

Astah Community é um software para modelagem UML (*Unified Modeling Language* – Linguagem de Modelagem Unificada) com suporte a UML 2, desenvolvido pela Change Vision, Inc e disponível para sistemas

operacionais Windows 64 bits. Anteriormente conhecido por JUDE, um acrônimo de Java and UML Developers Environment (Ambiente para Desenvolvedores UML e Java).

3.3. DART

Será necessária uma linguagem de programação para o desenvolvimento do aplicativo, e será utilizada a linguagem Dart que segundo Guedes (2019), é uma linguagem de programação que foi criada pela empresa Google que é fortemente tipada, que significa que todas as características de um objeto devem ser informadas em sua declaração. O objetivo principal dessa linguagem foi para substituir o JavaScript em criações de aplicações web. Mas, sua evolução foi mais do que esperado pelos desenvolvedores, fez com que ela se tornasse uma linguagem multi-paradigma.

A linguagem não obteve sucesso em seu objetivo inicial que era para substituir o JavaScript, mas com a grande evolução do Flutter, o Dart voltou a ganhar o público novamente.

Essa tecnologia foi inspirada na linguagem C, sendo assim, facilitando migrações de desenvolvedores que já trabalham com Java, C# e outras linguagens similares.

3.4. VISUAL STUDIO CODE

Para codificar o projeto é essencial usar um editor de texto para auxiliar no desenvolvimento do mesmo, e nesse trabalho será utilizado o Visual Studio Code, que conforme Dionisio (2016) é um editor leve, gratuito e de multi-plataforma que foi desenvolvido pela empresa Microsoft. Sua ideia inicial era para desenvolvimento web, porém como se tornou uma ferramenta de código aberto, a comunidade acabou estudando e criando novas funcionalidades para o editor.

Para esse projeto em Flutter o Visual Studio Code auxiliou muito, pois nele é possível baixar plug-ins que facilita o desenvolvimento de código, como por exemplo a funcionalidade de comunicar com o celular e rodar a aplicação em tempo real e auto completar do código.

3.5. INTELLIJ IDEA

Segundo JetBrains (2020?), IntelliJ IDEA é uma plataforma de código aberto com objetivo de oferecer ferramentas para desenvolver. Essa IDE (*Integrated Development Environment*) foi desenvolvida na linguagem de programação Java e oferece um comportamento de plataforma cruzada com o intuito de desenvolver ferramentas para quaisquer tipos de linguagens. Todo seu código encontra-se no GitHub, sendo assim, possibilitando criação de plug-ins para auxílios de desenvolvimento pela comunidade interessada.

3.6. JAVA

Java é uma linguagem de programação que foi criada pela empresa Microsystems no ano de 1995 conforme JAVA (2010?). Esta linguagem é orientada a objetos e seu maior objetivo é construir apenas um código e seja utilizado por diversos tipos de dispositivos, pois há uma *Java Virtual Machine* que traduz o código para todos dispositivos, porém para que isso seja realmente possível, o requisito mínimo é a instalação do próprio Java. Em algumas aplicações é necessária a utilização do Java em seu computador ou até mesmo em seu dispositivo móvel.

3.7. SPRING BOOT

Segundo Afonso (2017), Spring Boot é uma ferramenta para agilizar configurações iniciais e publicações de aplicações no ecossistema Spring, dito isso, havendo uma rapidez considerável em executar o projeto que é trabalhado. No exemplo desse documento é a API Rest feita com a linguagem de programação Java, que tem como objetivo devolver informações para o aplicativo através de requisições HTTP¹³.

¹³HTTP é um protocolo de transferência de informações

3.8. API REST

API (*Application Programming Interface*) de acordo com Pires (2017) é um grupo de regras e padrões definidos e documentada por uma aplicação para que essa aplicação seja utilizada por outras. O intuito dessa tecnologia é comunicações entre aplicações e com os usuários que as utilizam.

O Rest (*Representational State Transfer*) é uma abstração da arquitetura Web. Nele é possível criar projetos com uma boa definição de interfaces, porém para que isso ocorra, há regras a serem seguidas. Se essas características forem feitas, é possível uma comunicação entre aplicações.

3.9. MONGODB

Conforme Soares (2016) MongoDB é um banco de dados orientado a documentos (NoSQL) que é de código aberto e que foi desenvolvido em c++. Por ser orientado a documentos, não há a obrigatoriedade de se preocupar com a estrutura de dados, como colunas e tipos de valores. Esse banco é muito semelhante a estrutura JSON, que acaba facilitando a leitura e escrita dos dados. Diferente de um banco de dados estruturado, o MongoDB armazena documentos e coleções, onde documentos seriam as tabelas e as coleções seriam as bases de dados.

MongoDB vem com o objetivo de ter uma melhor performance para uma gama gigante de dados em comparação a banco de dados convencionais, porém tudo isso é possível por causa da desnormalização (dados redundantes).

3.10. AWS AMAZON EC2

De acordo com Amazon (2020) o Amazon EC2 (*Amazon Elastic Compute Cloud*) é um serviço computacional dimensionável em nuvem, que oferece um serviço que dispensa o investimento em máquinas físicas e que oferece uma função mais prática, podendo pagar somente pelo que for usar e cancelar o serviço quando quiser. Também há serviços como abrir mais de um servidor virtual, configurar a segurança e gerenciar o armazenamento da máquina. Em conclusão, é a disponibilização de soluções de TI¹⁴ sob demanda pela internet.

¹⁴ TI é abreviação para Tecnologia da Informação

4. PROPOSTA DE MODELAGEM DO ESTUDO DE CASO

Pensando na facilitação do desenvolvimento do aplicativo, nessa seção será exibida ferramentas que também fazem parte da disciplina de engenharia de software que auxiliou no entendimento geral do aplicativo. Para todos exemplos abaixo, foi utilizado o Astah Community que foi citado anteriormente.

Para melhorar a compreensão, será explicado mais sobre o objetivo da aplicação mencionada nesse trabalho. Podemos usar o seguinte exemplo, o usuário tem suas contas do mês corrente e dos meses seguintes, mas o mesmo não tem a informação de seu saldo para cada mês, dito isso, o aplicativo armazena todas informações de contas atuais e futuras e seus recebimentos (salário, empréstimos, etc), com a possibilidade de inserir contas parceladas, que serão geradas automaticamente pelo aplicativo, sendo assim, inserindo somente uma vez. Com as informações inseridas de pagamentos e recebimentos, o aplicativo informa seu saldo para cada mês. Confira a ilustração a seguir de um exemplo:

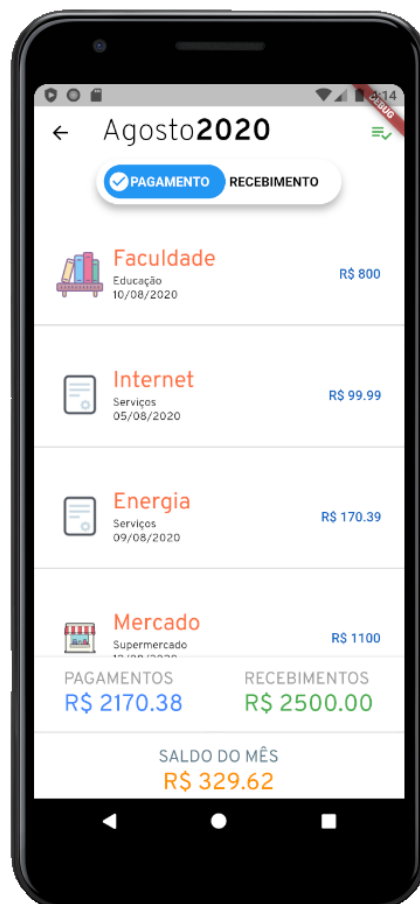
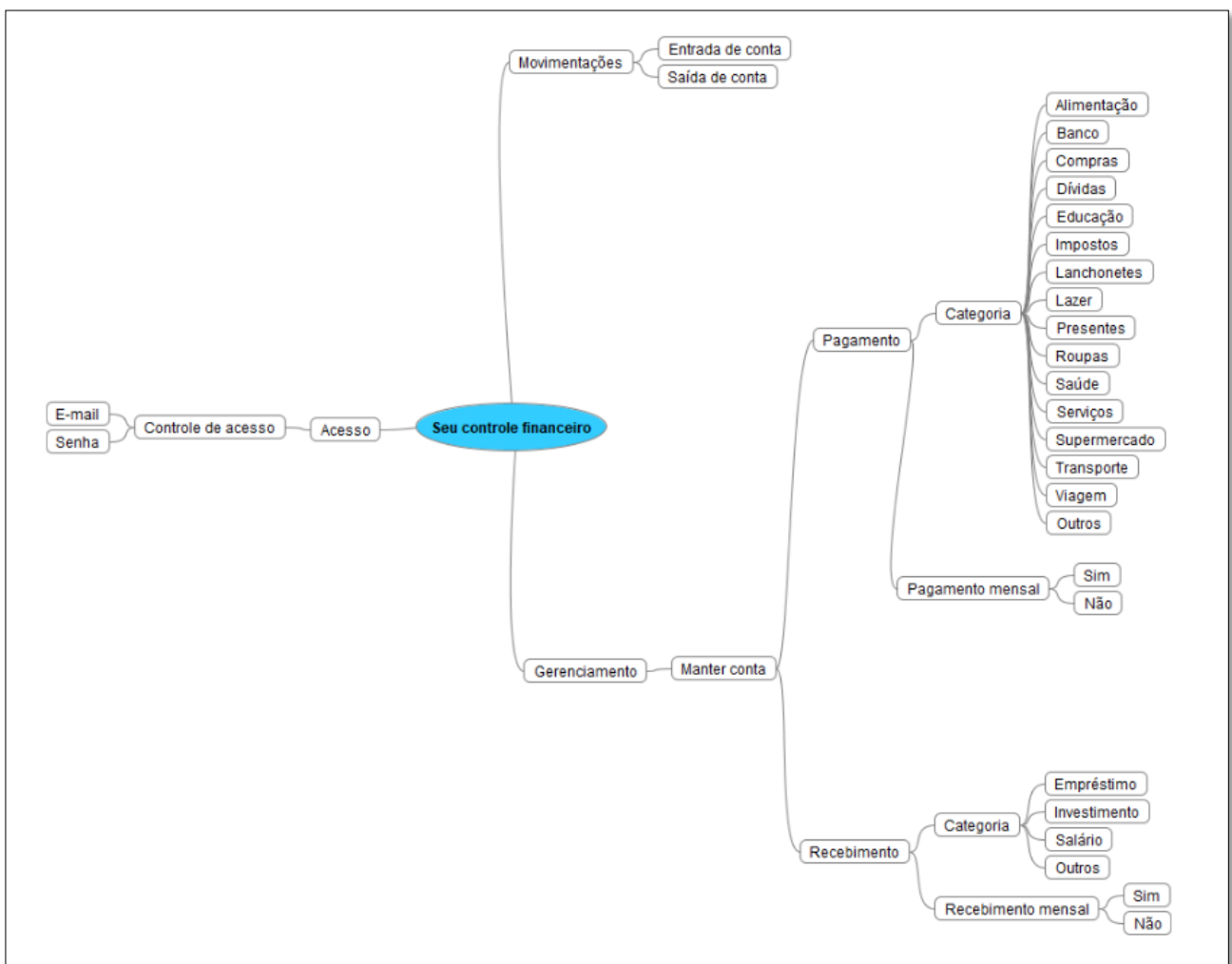


Figura 26: Exemplo da interface gráfica da lista de pagamentos

Para o exemplo da imagem acima, temos um total para pagar em agosto de 2020 de 2170,38 reais e 2500,00 reais para receber, dito isso, havendo um saldo positivo de 329,62 reais nesse mês.

4.1. MAPA MENTAL

Com intuito de entender mais um pouco sobre o que foi pensado para a solução desse projeto, foi construída o mapa mental do mesmo, que está sendo ilustrada na figura abaixo:



4.2. DIAGRAMA DE CASO DE USO

O diagrama de casos de uso normalmente é concluído no início do projeto, na parte de análise e os levantamentos de requisitos da aplicação. Isso não quer dizer que será somente usado no início, mas sim no desenvolvimento inteiro do aplicativo. A finalidade desse diagrama é mostrar de uma forma simples para o desenvolvedor, usuário e até mesmo leitor desse trabalho o que o aplicativo irá fazer, mesmo a pessoa não sabendo nada da área de desenvolvimento, porque tem uma leitura de fácil compreensão.

A figura abaixo apresenta o Diagrama de Caso de Uso geral.

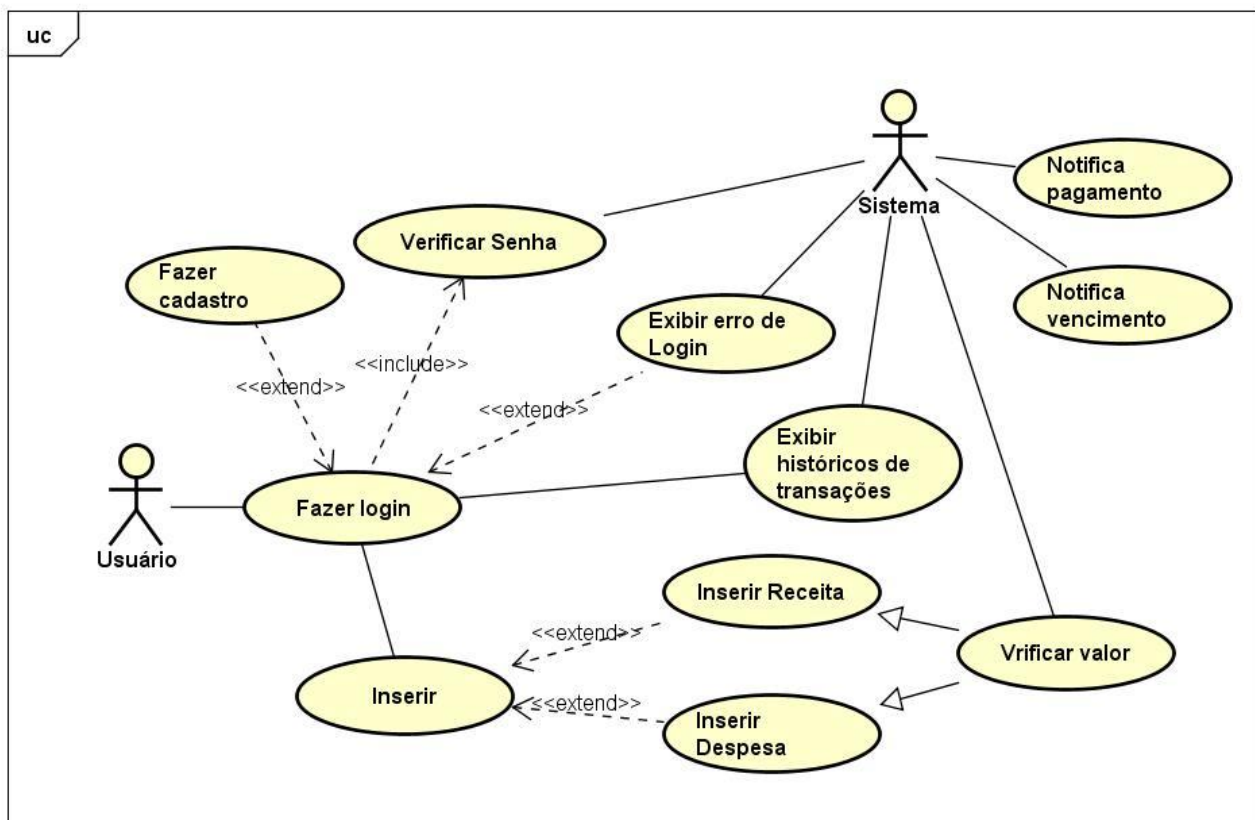
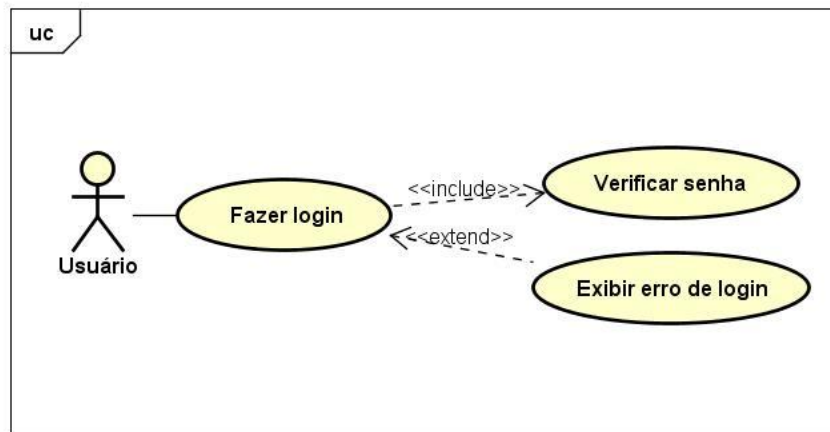


Figura 27: Diagrama de Caso de Uso – Geral

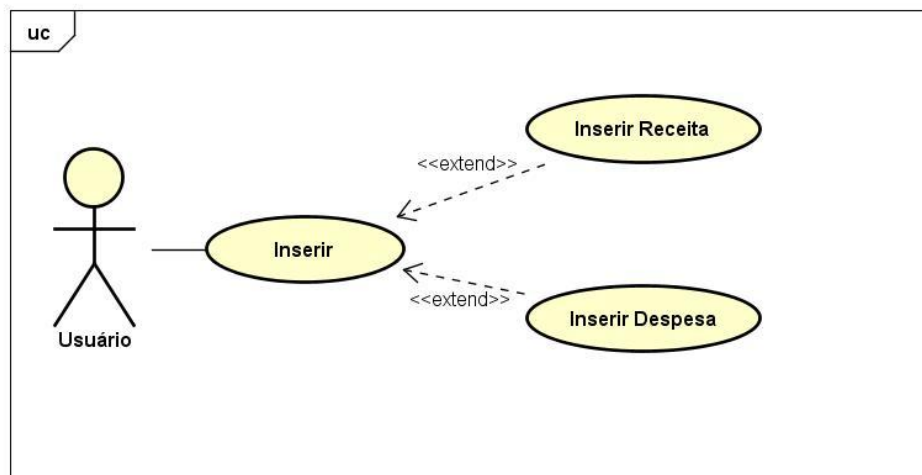


1. **Finalidade / Objetivo:** O usuário realiza o acesso com o aplicativo.
2. **Ator:** Usuário.
3. **Evento inicial:** O ator insere as suas informações de acesso para entrar no aplicativo.
4. **Fluxo principal:**
 - a. O sistema oferece interface gráfica para o usuário inserir suas informações de acesso;
 - b. O usuário informa seus dados para acessar;
 - c. O usuário confirma suas informações e seleciona a opção Logar (A1).
5. **Fluxo Alternativo:**

A1 – O usuário não tem cadastro:

- a. O sistema exibe que não existe cadastro para as informações inseridas;
- b. O usuário clica em cadastrar;
- c. O sistema exibe uma interface gráfica para o usuário inserir suas informações;
- d. O usuário insere suas informações de usuário e senha;
- e. O usuário clica em cadastra;
- f. O sistema salva os dados no banco;
- g. O sistema exibe a interface gráfica de login.

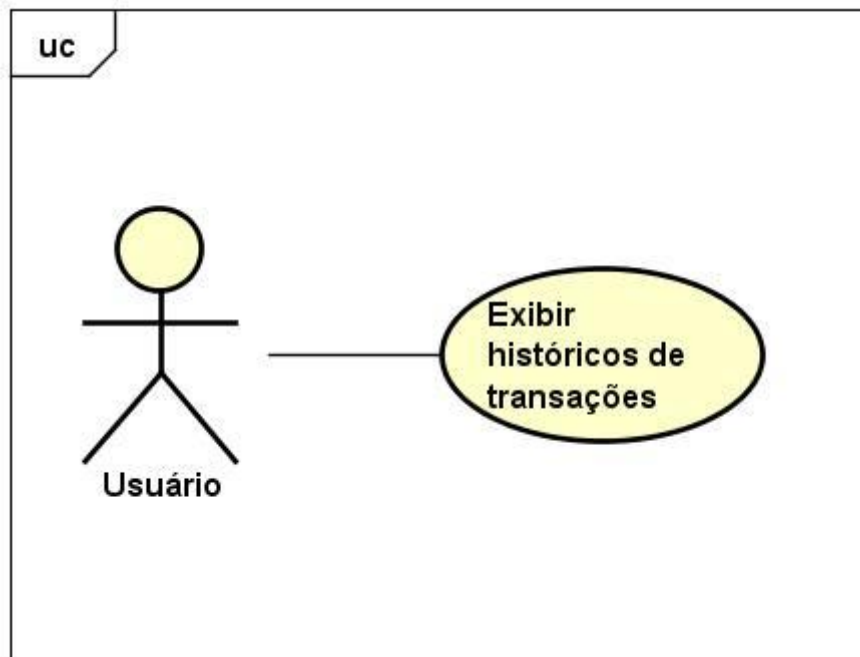
Figura 28: Diagrama de Caso de Uso – Fazer login



1. **Finalidade / Objetivo:** O usuário insere despesa ou receita.
2. **Ator:** Usuário.
3. **Evento inicial:** O usuário começa o caso de uso clicando em inserir *despesa/receita*.
4. **Fluxo principal:**
 - a. O sistema oferece uma interface gráfica para iniciar a inserção (A1);
 - b. O usuário seleciona se é *despesa* ou *receita*;
 - c. O usuário insere o valor;
 - d. O usuário seleciona a *categoria* da despesa ou receita;
 - e. O usuário insere uma *descrição* da despesa ou receita;
 - f. O usuário tem a opção de ativar *receita* ou *despesa mensal*;
 - g. O usuário insere a data de pagamento ou cobrança;
 - h. O usuário clica no botão de *salvar*;
 - i. O sistema valida o valor (A2);
 - j. O sistema salva no banco as informações;
 - k. O sistema exibe a interface gráfica principal do aplicativo.
5. **Fluxo alternativo:**
 - A1 – O usuário cancela a operação:**
 - a. O usuário cancela a inserção;
 - b. O sistema exibe a tela de início.
 - A2 – O usuário digita informações inválidas:**

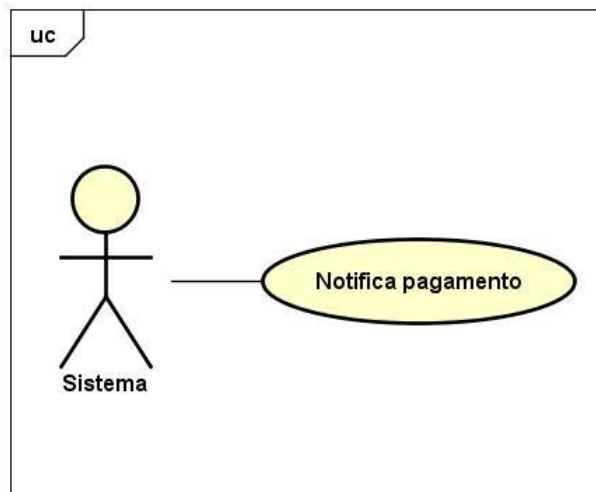
- a. O sistema informa para o usuário por meio de uma interface gráfica as informações inválidas.

Figura 29: Diagrama de Caso de Uso – Inserir receita ou despesa



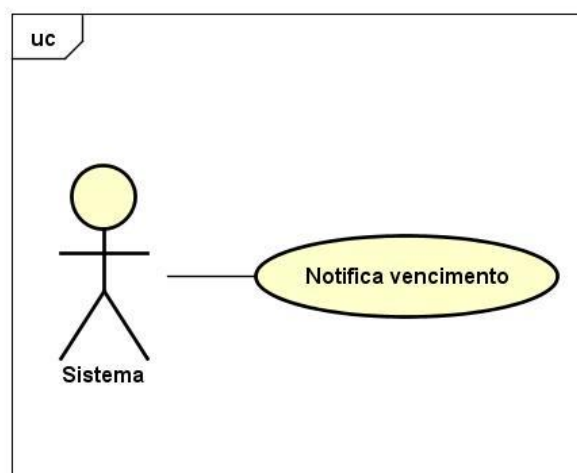
1. **Finalidade / Objetivo:** O usuário consegue ver todas as transações feitas.
2. **Ator:** Usuário.
3. **Evento inicial:** O ator clica no saldo atual que está na tela inicial do aplicativo.
4. **Fluxo principal:**
 - a. O sistema oferece uma interface gráfica de todas as transações do usuário;
 - b. O usuário informa o período que deseja para analisar;
 - c. O sistema busca no banco as informações;
 - d. O sistema exibe todas as transações desejadas;

Figura 30: Diagrama de Caso de Uso – Exibir transações



1. **Finalidade / Objetivo:** Exibir notificação de pagamento a receber.
2. **Ator:** Sistema.
3. **Evento inicial:** O sistema faz uma varredura de contas a receber do dia atual;
4. **Fluxo principal:**
 - a. O sistema colhe informações das contas a serem recebidas;
 - b. O sistema exibe na barra de notificação do celular do usuário que o mesmo tem um pagamento a receber.

Figura 31: Diagrama de Caso de Uso – Notifica pagamento



1. **Finalidade / Objetivo:** Exibir notificação de contas a pagar.
2. **Ator:** Sistema.

3. **Evento inicial:** O sistema faz uma varredura de contas a pagar do dia atual;
4. **Fluxo principal:**
 - a. O sistema colhe informações das contas a serem pagas;
 - b. O sistema exibe na barra de notificação do celular do usuário que o mesmo tem uma conta a receber.

Figura 32: Diagrama de Caso de Uso – Notifica vencimento

4.3. DIAGRAMA DE CLASSES

O diagrama de classe é um diagrama que tem como público alvo pessoas que estão desenvolvendo o projeto, mais específico para o desenvolvedor, pois de acordo com Tybel (2016), o Diagrama de Classe é exibição de uma estrutura do banco de dados, ou seja, estrutura das classes interligadas que são como se fossem modelos para a aplicação. Dito isso, facilita para o programador a estruturação do banco para o próximo passo, que seria o desenvolvimento do aplicativo. Na imagem a seguir, será exibido o diagrama de classe que contém os principais atributos e métodos que será necessário em cada objeto para a aplicação móvel.

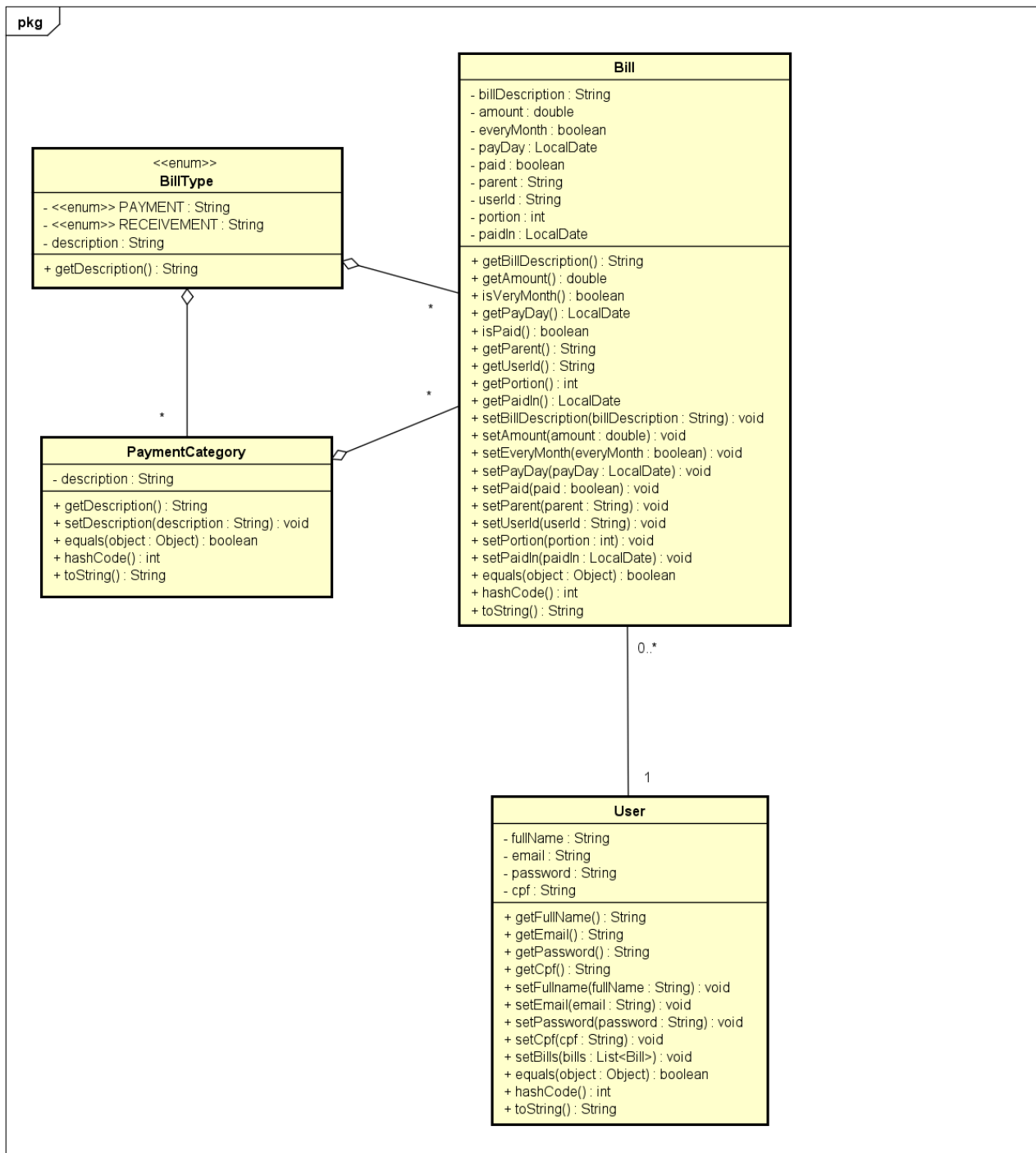


Figura 33: Diagrama de classe

A leitura da figura exibida acima é da seguinte forma:

- Usuário pode zero ou muitas contas;
- Uma conta pode ter apenas um usuário;
- Uma conta precisa depende de um tipo de conta e uma categoria;
- Uma categoria de pagamento depende de um tipo de conta;
- Um tipo de conta pode ter muitas contas;
- Um tipo de conta pode ter muitas categorias de pagamentos;

4.4. DIAGRAMA DE ATIVIDADE

Para se comunicar dentro de uma organização sobre um projeto e mostrar de uma forma mais clara de como ele funciona, usam-se diagrama de atividades que segundo Lucidchart (2018?) possui como finalidade principal unir desenvolvedores e pessoas da área de negócio a entender um determinado processo, ou seja, uma atividade.

Nesse diagrama mostra-se as lógicas do algoritmo, etapas das atividades, processo de negócio entre usuários e sistema e também o esclarecimento do projeto. Na figura * é exibido o diagrama de atividade do projeto, nele consegue-se entender como funciona uma inclusão de despesa ou receita no aplicativo.

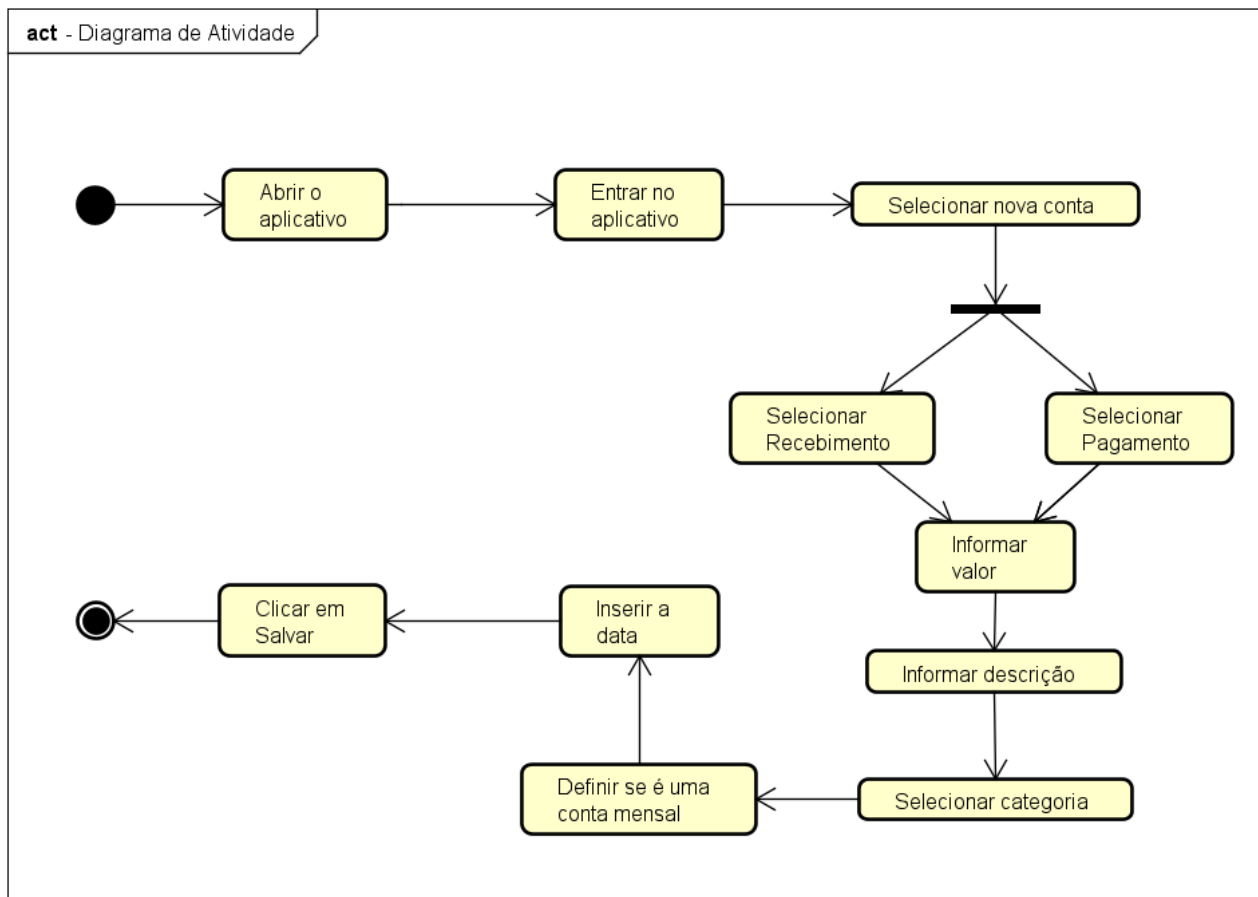


Figura 34: Diagrama de Atividade para inserir despesa ou receita

5. DESENVOLVIMENTO

Nessa seção será esclarecido o desenvolvimento do trabalho. E para maior didática, a imagem abaixo ilustra a arquitetura que foi utilizada no aplicativo SEU CONTROLE FINANCEIRO:

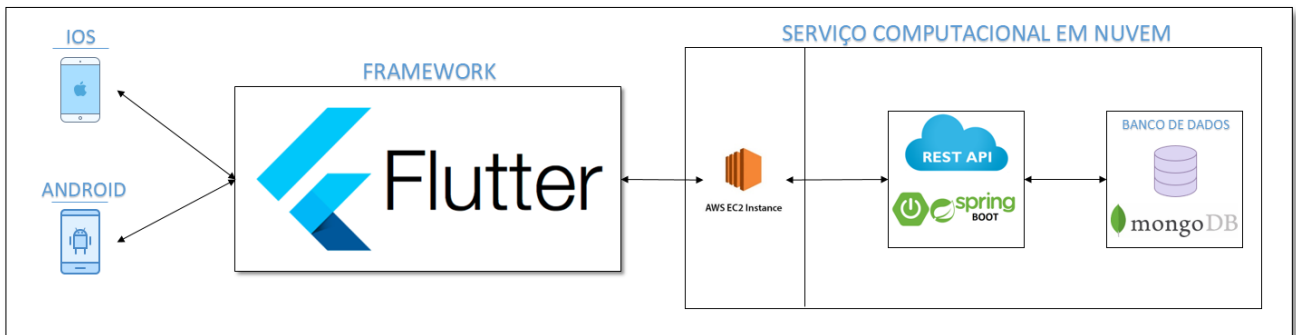


Figura 35: Arquitetura do projeto

A imagem acima exibe as principais tecnologias utilizadas para a implementação do aplicativo desse trabalho, tendo foco no desenvolvimento híbrido utilizando Flutter, dito isso, sendo executado em dispositivos com sistema operacional IOS e ANDROID.

Em serviço computacional em nuvem (máquina virtual) fica instalado o banco de dados e a aplicação API Rest. O serviço utilizado para esse armazenamento é o AWS EC2 Instance da Amazon. A parte de Framework (Flutter) fica a codificação do front-end, onde no mesmo é feito codificação das interfaces do aplicativo e codificação para acessar o serviço da Amazon.

O fluxo que acontece a partir do dispositivo, que envia solicitações através de uma interface que o Flutter fornece juntamente com os dados. Para ter esses dados, o próprio Flutter faz uma comunicação (via HTTP) com o serviço da Amazon, e que em seguida recebe uma resposta, podendo ser resposta de erro do serviço computacional ou resposta da API Rest. A API se comunica com o banco (MongoDB) para colher informações e levar através dessas pontes.

5.1. DESENVOLVIMENTO BACK-END

O desenvolvimento do back-end foi feito em Spring Boot com a utilização da linguagem de programação Java. A finalidade é que toda regra de negócio fique nessa parte, ficando fracamente acoplada, dito isso, quando precisar migrar ou inserir outra tecnologia tem uma facilidade maior, pois não haverá a necessidade de escrever o código novamente. Para melhorar o aproveitamento de código e fácil manutenção, o back-end foi separado nas camadas de *config*, *domain*, *dto*, *form*, *resources*, *repository* e *services*.

Config: Nessa camada fica toda a configuração da aplicação, ou seja, toda vez que a aplicação for executada essa camada será a primeira a ser chamada. Nela pode conter inserção de dados padrões e configurações de segurança de acesso.

Domain: Onde ficará todas as classes modelos ou classes de domínios juntamente com seus métodos.

DTO: Essa camada é a abreviação de *Data Access Object* que significa em português objeto de acesso a dados. Ela tem como responsabilidade limitar os dados que serão retornados para o usuário. Podemos usar um exemplo uma classe de modelo Pessoa que tem os atributos email, nome, cpf e senha, nessa camada (DTO) podemos criar uma classe que tenha todos atributos com exceção da senha, sendo assim, quando o usuário solicitar uma consulta a dados de Pessoa, será retornado o objeto da classe DTO que não contém o atributo senha para que não haja vazamento de informações privadas.

Form: Já nessa camada terá todos os atributos que é necessário ser inserido através de um usuário, se utilizarmos o exemplo que foi explicado na camada DTO, podemos ter o atributo senha, pois esse será o objeto que vem no corpo de uma requisição. Nesse caso seria uma inserção de dados e o DTO uma consulta a dados.

Resources: Nessa camada é onde a requisição via HTTP chega, nela será redirecionada para o método correto, sendo eles: GET, POST, PUT, DELETE. No método GET tem finalidade de retornar para o usuário que solicitou as informações desejadas. No POST é o método responsável para fazer a chamada de inserção de dados. PUT é o método responsável para fazer a chamada de edição dos dados que o usuário enviou. Por fim o DELETE tem como finalidade fazer chamada a funções que deletam o objeto que o usuário deseja.

Services: A camada *services* tem como objetivo tratar toda regra das chamadas de

visualização, edição, criação e deleção. Podemos usar um exemplo de um objeto que o usuário solicitou exclusão e o mesmo não há na base de dados, essa camada irá analisar se existe o objeto e se não houver retorna à informação para camada de *resources*, caso contrário faz uma solicitação para camada *repository* que será explicado em seguida.

Repository: Essa camada é responsável por se comunicar com o banco de dados.

Para entender melhor o fluxo da requisição do usuário para o back-end, veja a imagem exibida abaixo:

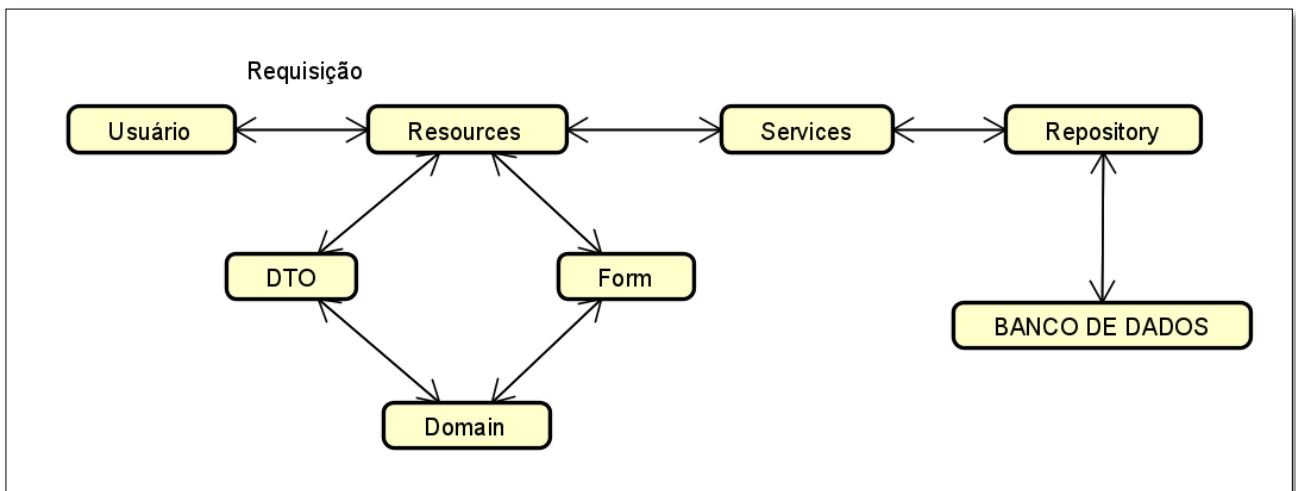


Figura 36: Fluxograma da estrutura da API Rest

5.2. DESENVOLVIMENTO FRONT-END

Para o desenvolvimento do front-end foi utilizado a ferramenta Flutter com a utilização da linguagem de programação Dart. O objetivo é se comunicar com o back-end para colher informações para retornar juntamente com a interface gráfica que a própria ferramenta fornece através de programação.

Nele a separação de camada foi mais simples, contendo as pastas das interfaces do aplicativo e os arquivos de serviço.

O arquivo de serviço tem como responsabilidade comunicar com a API para obter informações para devolver para o usuário através de métodos HTTP.

```

9  class BillService {
10     static Future<List<BillModel>> getBills() async {
11         var prefs = await SharedPreferences.getInstance();
12         String token = prefs.getString('token') ?? '';
13         String tokenType = prefs.getString('type') ?? '';
14         String user = prefs.getString('user') ?? '';
15
16         var header = {
17             "Content-Type": "application/json",
18             "Authorization": "$tokenType $token"
19         };
20
21         var url = UriGlobal.url() + '/scf-service/users/$user/bills';
22
23         Response response =
24             await Dio().request(url.toString(), options: Options(headers: header));
25
26         List<BillModel> bills = List<BillModel>();
27
28         for (Map<String, dynamic> item in response.data) {
29             bills.add(BillModel.fromJson(item));
30         }
31         return bills;
32     }

```

Figura 37: Método da classe de serviço de contas

A classe da imagem acima é de um serviço do aplicativo. Esse método tem como finalidade retornar uma lista de contas referente ao usuário que está conectado.

Dentro dele há informações que são colhidas da memória do aplicativo, como o *Token*, tipo do *Token* e o usuário. Após colher essas informações a URL é construída para passar futuramente numa requisição HTTP. Em seguida é feito uma requisição passando os parâmetros de *header* para conseguir autenticar na API. Quando a informação é retornada acaba sendo convertida para o objeto referente da requisição e por fim retorna a lista de contas para quem chamou o método.

5.3. INTERFACE GRÁFICA

A seguir será exibido imagens das interfaces gráficas do aplicativo sendo executada plataforma Android.



Figura 38: Interface gráfica de acesso

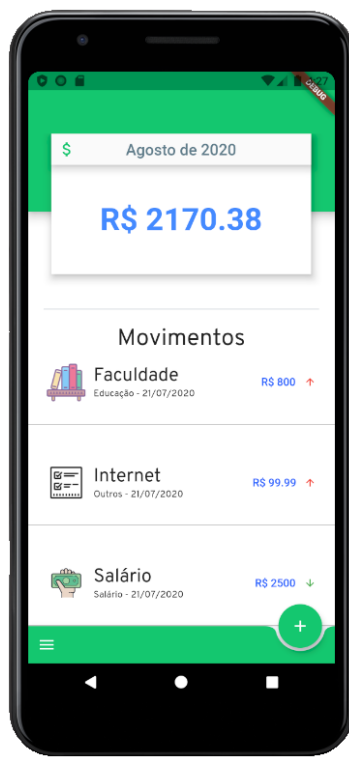


Figura 39: Interface gráfica da página inicial do aplicativo

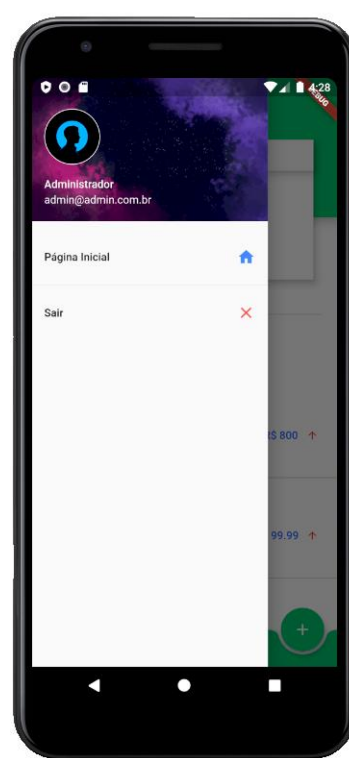


Figura 40: Interface gráfica do menu do aplicativo

Na **interface gráfica de acesso** é onde o usuário precisa inserir os dados que foram cadastrados pelo administrador para acessar ao aplicativo.

Em **interface gráfica da página inicial do aplicativo** é onde fica todos movimentos de contas do usuário conectado, contendo cartões para cada mês com recebimentos ou pagamentos e uma linha do tempo do que foi movimentado em todo período de utilização.

Já em **interface gráfica do menu do aplicativo** é somente um menu que exhibe os dados do usuário conectado, um botão para redirecionar para página inicial e outro botão para desconectar do aplicativo.

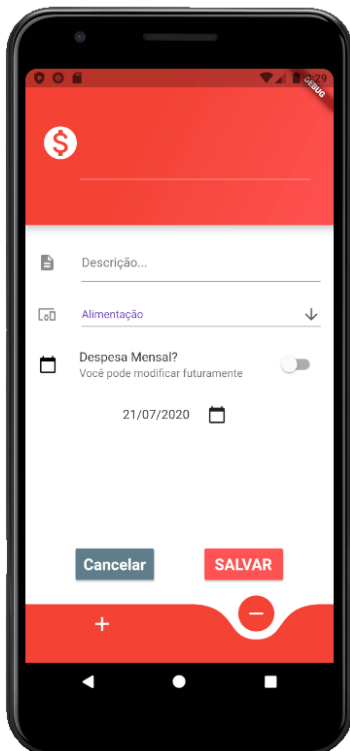


Figura 41: Interface gráfica da inserção de pagamento

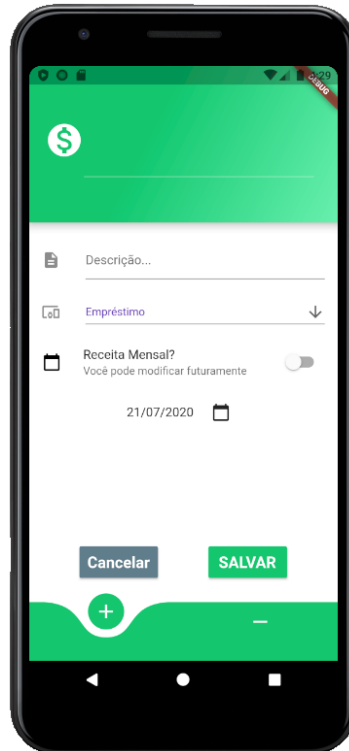


Figura 42: Interface gráfica da inserção de recebimento

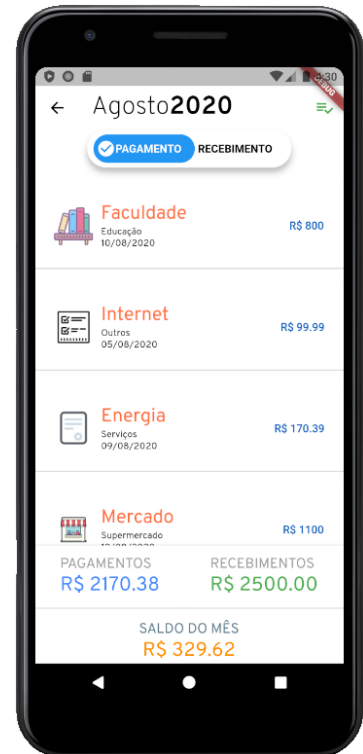


Figura 43: Interface gráfica da lista de pagamentos

Em **interface gráfica da inserção de pagamento** tem como finalidade inserir qualquer pagamento que o usuário desejar, também há a possibilidade de inserir um valor, descrição, categoria, data do pagamento e possibilitando notificar que é uma despesa mensal.

Na **interface gráfica da inserção de recebimento** é onde se insere os recebimentos que o usuário desejar, nele também há as mesmas informações de pagamentos, como valor, descrição, categoria, data do recebimento e receita mensal.

Interface gráfica da lista de pagamentos tem como objetivo informar os detalhes de todas as informações inseridas referente a contas do aplicativo, separados por aba, uma contendo os pagamentos e outra os recebimentos. Nessa interface é mostrado informações do mês que o usuário quiser, somatório dos pagamentos e recebimentos do mês selecionado e para cada item contém as seguintes informações: descrição da conta, categoria da conta, data de pagamento da conta e valor da conta.

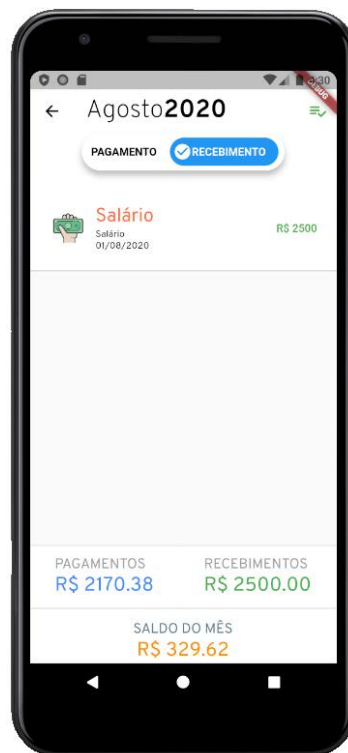


Figura 44: Interface gráfica da lista de recebimentos

Em **interface gráfica da lista de recebimentos** mostra todos as informações que a de pagamentos tem, somente alterando para o próprio recebimento.

6. CONSIDERAÇÕES FINAIS

Houve diversas adversidades no processo de finalização desse trabalho, pois trata-se de uma tecnologia nova no mercado e consequentemente não obtendo muitos exemplos em livros e artigos que pudessem agregar na construção do mesmo, entretanto, a maioria dos objetivos foram alcançados graças a pesquisas em inglês, proporcionou-se o sucesso desse projeto e com uma excelente experiência em aprendizado. A maior dificuldade mesmo que tive foi achar *Widgets* personalizados, como um botão que troca de posição dependendo da opção selecionada com a finalidade de deixar a aplicação mais atraente ao usuário. Obtive outras dificuldades também como conectar meu aplicativo com o banco em nuvem, pois alguns tutoriais ou sites não estavam dando certo para minha versão, somente uma vídeo aula que me ajudou a ter sucesso na conexão. Obtive problemas também para resolver problemas eventuais que aconteceram no aplicativo, como funcionalidades que pararam de funcionar após atualização de código.

Vale enfatizar a facilidade e rapidez do desenvolvimento do aplicativo quando se estuda mais a fundo a ferramenta e suas funcionalidades.

O projeto inteiro ficará disponível no GitHub, com isso, auxiliará novos desenvolvedores, pessoas interessadas nessa tecnologia maravilhosa e principalmente nos meus trabalhos futuros como desenvolvedor. E com base na finalização desse trabalho, chegou-se à conclusão que a aplicação mencionada terá um valor enorme para comunidade.

7. REFERÊNCIAS

AFONSO, Alexandre. **O que é Spring Boot?** Disponível em <<https://blog.algaworks.com/spring-boot/>>. Acesso em: 20 jul. 2020.

AMAZON. **O que é o Amazon EC2?** Disponível em <https://docs.aws.amazon.com/pt_br/AWSEC2/latest/UserGuide/concepts.html>. Acesso em: 21 jul. 2020.

DIONISIO, Edson. **Introdução ao Visual Studio Code.** Disponível em <<https://www.devmedia.com.br/introducao-ao-visual-studio-code/34418>>. Acesso em: 02 mar. 2020.

GUEDES, Marylene. **O que é Dart?** Disponível em <<https://www.treinaweb.com.br/blog/o-que-e-dart/>>. Acesso em: 11 fev. 2020.

JETBRAINS. **IntelliJ Platform.** Disponível em <<https://www.jetbrains.com/opensource/idea/>>. Acesso em: 21 jul. 2020.

LIMA, Davi. **Modelos softwares com Astah Community.** Disponível em <<https://www.techtudo.com.br/tudo-sobre/astah-community.html>>. Acesso em: 19 nov. 2019.

LUCIDCHART. **O que é um diagrama de atividades.** Disponível em <<https://www.lucidchart.com/pages/pt/o-que-e-diagrama-de-atividades-uml>>. Acesso em: 02 mar. 2020.

MAES, Jefferson. **Firebase o que é e para que serve?** Disponível em <<http://digitalprimews.com/google-firebase/>>. Acesso em: 25 fev. 2020.

MAGALHÃES, Túlio. **Flutter: tudo sobre o queridinho do google.** Disponível em <<https://www.zup.com.br/blog/flutter>>. Acesso em: 21 out. 2019.

PAULA, Welington. **Rumo do Desenvolvimento Mobile.** Disponível em <<https://www.devmedia.com.br/rumo-do-desenvolvimento-mobile/24129>>. Acesso em: 21 nov. 2019.

PIRES. **O que é API? REST e RESTful? Conheça as definições e diferenças!** Disponível em <<https://becode.com.br/o-que-e-api-rest-e-restful/>>. Acesso em: 20 jul. 2020.

PORTAGSI. **O que é UML?** Disponível em <<https://www.portalgsti.com.br/uml/sobre/>>. Acesso em: 21 nov. 2019.

SANTANA, Fabiano. **Flutter: porque você deveria apostar nesta tecnologia.** Disponível em <<https://tableless.com.br/flutter-porque-investir-nessa-tecnologia/>>. Acesso em: 10 jul. 2020.

SOARES, Jhonathan. **O que é MongoDB e porque usá-lo?** Disponível em <https://codigosimples.net/2016/03/01/o-que-e-mongodb-e-porque-usa-lo/>>. Acesso em: 21 jul. 2020.

TYBEL, Douglas. **Orientações básicas na elaboração de um diagrama de classes.** Disponível em <<https://www.devmedia.com.br/orientacoes-basicas-na-elaboracao-de-um-diagrama-de-classes/37224>>. Acesso em: 02 mar. 2020.

VIANA, Daniel. **Firebase: descubra no que esta plataforma pode te ajudar.** Disponível em <<https://www.treinaweb.com.br/blog/firebase-descubra-no-que-esta-plataforma-pode-te-ajudar>>. Acesso em: 03 out. 2019.