CENTRO PAULA SOUZA

FACULDADE DE TECNOLOGIA DE FRANCA

“Dr. THOMAZ NOVELINO”

TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

LERRY AUGUSTO PIRANI

GABRIEL DE CASTRO BASILIO SOUZA

WALLETDONE

Trabalho de Graduação apresentado à Faculdade de Tecnologia de Franca - “Dr. Thomaz Novelino”, como parte dos requisitos obrigatórios para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Orientadora: Profª. Dra. Jaqueline Brigladori Pugliesi

FRANCA/SP

2022

WALLETDONE

Lerry Augusto Pirani[1]
Gabriel de Castro Basilio Souza[2]

Resumo

Tendo em vista a dificuldade em realizar um gerenciamento financeiro, algumas pessoas se perdem nos valores, não sabendo ao certo como controlar suas contas, não se recordam dos débitos realizados e perdem a noção da quantidade de dinheiro que estão utilizando, causando um acúmulo de dívidas desnecessárias. O projeto tem o intuito de desenvolver uma aplicação mobile que auxiliará no controle das finanças. Os usuários terão a oportunidade de registrar suas contas mensais, incluindo as do cartão bancário, boletos e registrar manualmente o valor incluído em sua carteira física. O aplicativo fará com que seja mais fácil o gerenciamento das contas e a ter um bom entendimento dos seus limites financeiros.

Palavras-chave: Aplicativo mobile. Controle de contas. Gestão financeira.

Abstract

Having been difficult in carrying out financial management, some people get lost in values, do not know for sure how to control their accounts, do not remember the debts made and lose track of the amount of money they are using, causing an accumulation of unnecessary debts. The project aimed to develop a mobile application that will assist in the control of finances; users will have the opportunity to register their monthly accounts, including bank card accounts, slips and manually register the amount included in their physical wallet. The app will make it easier to manage accounts and have a good understanding of your financial limits.

Keywords: *Mobile app. Account control. Financial management.*

1 Introdução

Nos dias atuais, verificamos que algumas pessoas estão buscando um maior autocontrole financeiro sendo um dos motivos o constante aumento da inflação e da desvalorização salarial, outro é a falta de conhecimento já que e o nosso ensino público a educação financeira não faz parte do curriculo escolar, causando uma falsa sensação de perda de controle e de ações imprudentes caindo fazendo o individuo se endividar de forma crescente. Devido a isso, desenvolvemos um aplicativo *mobile* que fornece aos seus usuários a experiência satisfatória de poder ter de maneira fácil e rápida seus gastos financeiros detalhados em suas mãos, permitindo ter o conhecimento contínuo de seus gastos diários, mensais e anuais.

[1] Graduando em Análise e Desenvolvimento de Sistemas pela Fatec Dr Thomaz Novelino – Franca/SP. Endereço eletrônico: lerry.pirani@gmail.com.
[2] Graduando em Análise e Desenvolvimento de Sistemas pela Fatec Dr Thomaz Novelino – Franca/SP. Endereço eletrônico: gabre_346@hotmail.com.

Percebemos que muitas pessoas e famílias estão sofrendo por não conseguir se estabilizar ,de forma que seus ganhos sirvam para pagar contas basicas não tendo oportunidade a nenhum lazer se desgastando fisicamente e mentalmente todos os dias, trazendo para si problemas emocionais e psicológicos, muitas vezes fazendo gastos absurdos, e não buscando ajuda de profissionais financeiros.

Devido a esses problemas sociais de nossa sociedade capitalista, pensamos no desenvolvimento de um aplicativo *mobile* para possuir um controle financeiro na palma da mão no qual forneceremos mão de obra para empresas ou governos,recebendo por nossos serviços uma moeda de troca válida e legal.

O aplicativo disponibilizará de forma simples os valores reais de suas contas, dinheiro disponível para gastos e pendências a serem pagas, buscando organizar e inserir de forma manual os dados financeiros através de funcionalidades criadas no mesmo, ampliando a visão do usuário com suas despesas e estado atual do próprio orçamento, disponibilizando dicas financeiras e alertando os usuários quando deve ser realizada alguma ação para manter o saldo controlado.

2 Viabilidade do projeto

As atividades chave são a parte mais essencial do projeto; o conhecimento de manusear e extrair todas as capacidades possíveis da ferramenta, para que possamos entregar um *app* que seja de uso semelhante a outros aplicativos bancários, e de confiança.

Noções financeiras é outro ponto importante já que os dados processados devem ter uma base firme e correta.

Esse projeto entrega ao cliente um software voltado a oferecer facilidade e conforto a mais na vida dos usuários, possibilitando um melhor controle do gasto financeiro, podendo gerar economia financeira ou uma melhor eficiência em seus gastos. A capacidade de verificar as movimentações em tempo real e efetuar comparações entre períodos proporciona ao usuário uma noção maior de suas capacidades financeiras. A Figura 1 apresenta o Canvas do projeto.

Figura 1 - Canvas

Fonte: Autores

Colocando o projeto em plataformas *mobile* permitirá uma divulgação e acesso mais fácil a ferramenta, isso também nos permite a inclusão de anúncio no *app* o que geraria um lucro, para continuar a manutenção e *upgrades* no aplicativo e futuramente criando uma versão paga e melhorada da aplicação (CANVAS, 2022).

3 Levantamento de Requisitos

3.1 Elicitação e especificação dos Requisitos

Atualmente, pesquisa feita pelo Serviço de Proteção ao Crédito (SPC) Brasil aponta que cerca de 58% da população brasileira admitem que nunca, ou poucas vezes, controlam a sua vida financeira. Isso equivale a 6 em cada 10 brasileiros que necessitam utilizar cartão de crédito, cheque especial ou até mesmo pedir dinheiro emprestado para conseguir pagar as contas do mês. Uma das dificuldades que os brasileiros enfrentam é pensar que tem o controle financeiro gastando o que ainda não tem, o famoso ‘Cartão de crédito’, cobrindo o que precisa no momento e se deparando com a dívida muito além do que é esperado, isso fazendo a recorrer a empréstimos e outros serviços, tornando uma ‘bola de neve’ no final, e não conseguindo sair da situação. Interpretar números é importante para tomar boas

decisões financeiras, porém nem todos têm algum tipo de familiaridade com a matemática e conhecimento sobre números. (AGENCIABRASIL, 2018)

Com o intuito de ajudar as pessoas, sabendo das dificuldades com a matemática financeira e detalhar os gastos pessoais, o aplicativo foi idealizado pra que solucionar os problemas apresentados. Visto que a tecnologia faz parte da vida cotidiana e que a boa parte da população possui um smartphone, a solução seria utilizar a tecnologia digital para disponibilizar de forma rápida e acessível a todos os dados de finanças pessoais, trazendo as despesas e créditos que a pessoa possui e apresentando uma tabela de controle de compreensão e edição intuitivas, dando a opção da pessoa ativar um assistente que se definido uma meta ou plano de economia, este alertará o usuário quando estiver chegando perto do limite para não ultrapassá-lo e poderá dar dicas garantindo que além do controle de gastos, poderá separar uma reserva. A empresa responsável pelo desenvolvimento da OLIVIA AI (2022), uma inteligência artificial que auxilia no controle financeiro de empresas e pessoas, mostra que é possível fazer a busca automática dos dados de seus usuários. Com isso em mente pretendemos fazer o processo de busca os dados.

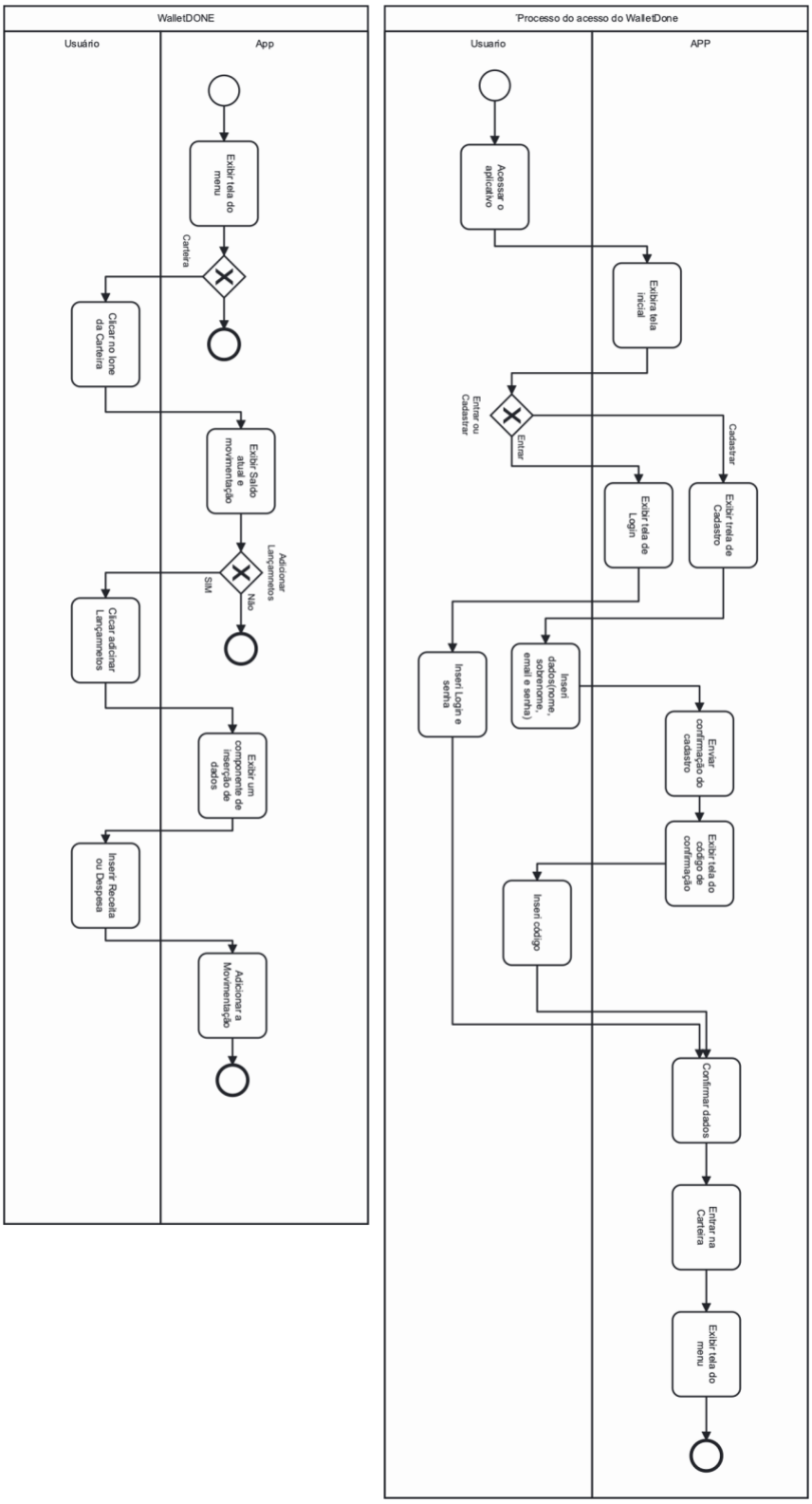
O uso deste recurso pode garantir o equilibro e a autossufiencia finaceira, afinal a organização e controle oferecida pelo *app* tomará as redeas da parte econômica oportunizando o controle de entrada e saída de recursos mostrando ao cliente que é possivel manusear os gastos de forma positiva dado a ele satisfação por usar nossos serviços e tranquilidade por estar no controle de seu dinheiro.

3.2 BPMN

O BPMN (*Business Process Model and Notation*) é um método de fluxograma que define as etapas, do início ao fim, de um processo de negócio. A notação BPMN é formada por ícones que servem para desenhar o fluxo do processo. A partir do mapeamento, é possível identificar os papéis de cada um, os eventos e todos os demais componentes de um processo. Dentre as principais vantagens, o BPMN serve como uma ferramenta de comunicação universal, já que utiliza uma linguagem que pode ser entendida por todos envolvidos. Além disso, também é muito versátil pois pode ser aplicado a diversos tipos de processos (REIS, 2008).

Na Figura 2 pode-se visualizar o BPMN do projeto.

Figura 2 - BPMN da aplicação



Fonte: Autores

3.3 Requisitos Funcionais

Os requisitos funcionais servem para descrever os comportamentos do sistema, a funcionalidade de cada item, sendo o que o sistema deve possuir para sua conclusão (PRESSMAN, 2011).

O Quadro 1 apresenta os requisitos funcionais do sistema.

Quadro 1 – Requisitos Funcionais do sistema

RF001-Cadastrar Usuário	Categoria: () Oculto (X)Evidente	Prioridade: (X) Altíssima () Alta () Média () Baixa
Descrição: O sistema deve exigir o cadastro do usuário, por meio de email e senha, permitindo o registro individual dos dados pessoais com validação por email		
RF002- Autenticar usuário/ Logar usuário	Categoria: () Oculto (X) Evidente	Prioridade: () Altíssima (X) Alta () Média () Baixa
Descrição: Além da tela de Cadastro o sistema deve ter uma tela de Login, para acesso de sua carteira quando possuir já cadastro		
RF003- Adicionar categoria de lançamento	Categoria: () Oculto (X) Evidente	Prioridade: () Altíssima (X) Alta () Média () Baixa
Descrição: O sistema deve permitir a inserção de filtros de que tipo de lançamento será inserido (ex: roupas, combustível e etc)		
RF004- Inserir lançamento	Categoria: () Oculto (X) Evidente	Prioridade: () Altíssima (X) Alta () Média () Baixa
Descrição: O sistema deve permitir a inserção de valores monetários na carteira virtual que será feita de forma manual pelo usuário		
RF005- Listar lançamentos / Filtrar lançamentos	Categoria: () Oculto (X) Evidente	Prioridade: () Altíssima () Alta (X) Média () Baixa
Descrição: O sistema deve ter uma lista com os lançamentos inseridos pelo usuário, e a opção dessa lista ser filtrada por receitas e despesas		
RF006- Ícone de ajuda	Categoria: () Oculto (X) Evidente	Prioridade: () Altíssima () Alta () Média (X) Baixa
Descrição: O Aplicativo deve possuir um ícone de ajuda, com possíveis dúvidas		

3.4 Requisitos Não Funcionais

Os requisitos não funcionais são relacionados a parte técnica da aplicação como desempenho, segurança, manutenção, suas tecnologias e usabilidades (PRESSMAN, 2011). O Quadro 2 apresenta os requisitos não funcionais do sistema.

Quadro 2 – Requisitos Não Funcionais do sistema

RNF001- Aplicativo	O sistema deve ser desenvolvido especificamente para o formato de aplicativo mobile	Tipo Plataforma	( ) Desejável (X) Obrigatório	(X) Permanente () Transitório
RNF002- Usabilidade	O sistema deve ser intuitivo e semelhante aos aplicativos de banco	Tipo Usabilidade	(X) Desejável () Obrigatório	(X) Permanente () Transitório
RNF003- Conexão com internet	O sistema precisa estar conectado a rede móvel ou wi fi, para suas funcionalidades	Tipo Conectividade	( ) Desejável (X) Obrigatório	(X) Permanente () Transitório
RNF004- Plataforma	O aplicativo funcionará tanto na plataforma Android e IOS	Tipo Plataforma	(X) Desejável () Obrigatório	(X) Permanente () Transitório
RNF005- Linguagem	O Sistema será desenvolvido em Dart com framework Flutter	Tipo Linguagem	( ) Desejável (X) Obrigatório	(X) Permanente () Transitório

Matriz de Rastreabilidade RF x RNF

Tabela 1 - Matriz de Rastreabilidade entre os requisitos do sistema

	RF001	RF002	RF003	RF004	RF005	RF006
RNF001	X	X		X		
RNF002			X	X		X
RNF003	X	X	X	X	X	
RNF004	X	X	X	X	X	X
RNF005	X	X	X	X	X	X

3.5 Regras de Negócio

As regras de negócios são as políticas da empresa como os processos tem de ser feitos, sendo a visão da empresa ou ideia como visto em (PRESSMAN, 2011). Deste sistema, as regras de negócio são apresentadas no Quadro 3.

Quadro 3 – Regras de Negócio do sistema.

RN001 – Inserir informações constantemente
Descrição: O usuário deve inserir todas as suas informações financeiras com frequência
RN002 – Monitorar e inserir informações de transações
Descrição: O usuário deve especificar as transações ocorridas em cada inserção financeira, para monitoração e controle

3.6 Casos de Uso

Como pode ser visto na Figura 3, o diagrama de caso de uso utiliza do aspecto visual para descrever as funcionalidades do projeto, e deve possuir as descrições de cada caso especificando os casos numerados (SOMMERVILLE, 2011).

Especificação de cada um dos casos de uso:

UC001 - Fazer Login

UC002 - Erro de login

UC003 - Verificação de Acesso

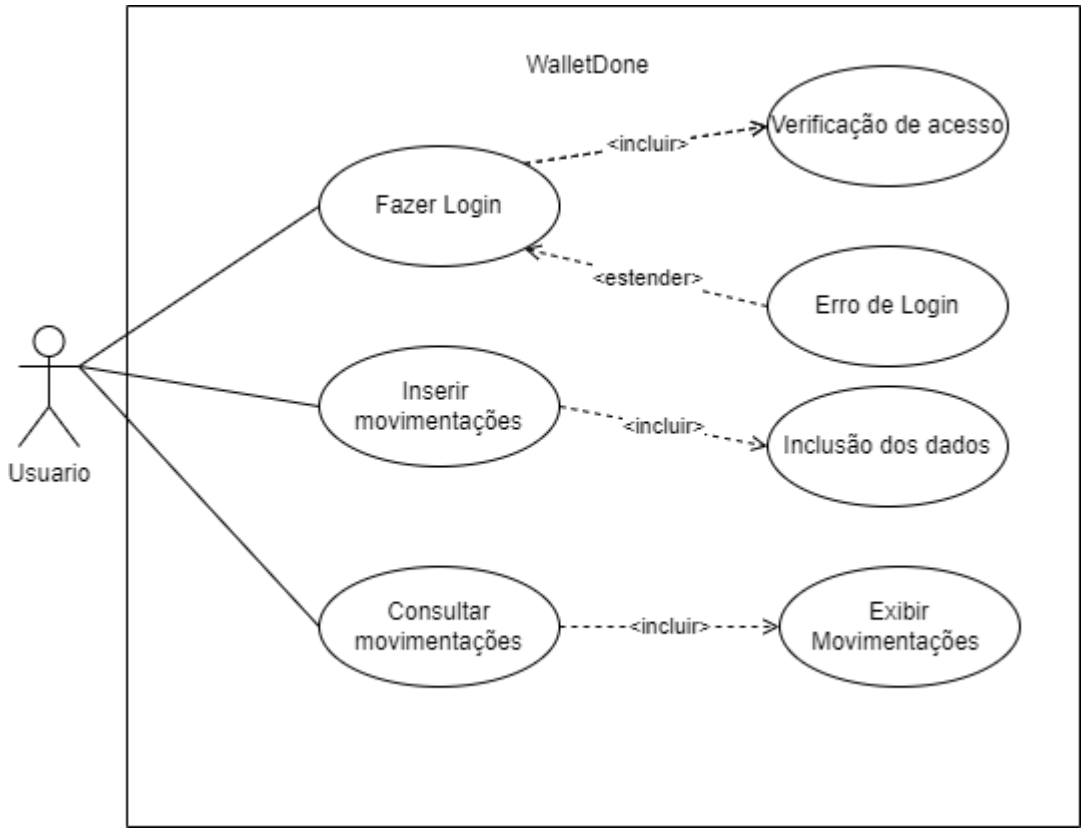
UC004 - Inserir Movimentação

UC005 - Inclusão dos Dados

UC006 - Consultar Movimentação

UC007 - Exibir Movimentação

Figura 3 - Diagrama de Casos de Uso do sistema



Fonte: Autores

Os casos de uso são descritos nos Quadros 4 a 11.

Quadro 4 – Use Case Login Usuário

Caso de Uso – Fazer Login	
ID	UC 001
Descrição	Este caso de uso tem por objetivo logar
Ator Primário	Usuário do sistema
Pré-condição	Nenhuma
Cenário Principal	1. O use case inicia quando o usuário seleciona a opção Logar 2. O sistema carrega o formulário de usuário e senha 3. O usuário insere o seu usuário e senha 4. O sistema loga 5. O use case encerra
Pós-condição	Nenhuma
Cenário Alternativo	3a - O usuário preenche os dados incorretos 3a- O sistema informa que os dados estão incorretos

Quadro 5 – Use Case erro no login

Caso de Uso – Erro de Login	
ID	UC 002
Descrição	Este caso de uso e no caso de que aja uma falha no *login*
Ator Primário	Usuário do sistema
Pré-condição	Caso 01
Cenário Principal	1. O use case inicia depois usuário seleciona a opção *logar* 2. O sistema carrega o formulário de usuário e senha 3. E pode acontecer um erro de login incorreto
Pós-condição	Dados incorretos inseridos
Cenário Alternativo	3a - O usuário preenche os dados corretos 3a- O sistema *loga*

Quadro 6 – Use Case verificação de acesso

Caso de Uso – Inserir Movimentação	
ID	UC 003
Descrição	Este caso de uso tem por objetivo de verificar os dados que serão passados ao usuário
Ator Primário	Sistema
Pré-condição	Ter Logado
Cenário Principal	1. O use case inicia quando o login foi validado 2. Fazendo a verificação e a disponibilização dos dados
Pós-condição	Login validado
Cenário Alternativo	

Quadro 7 – Use Case Inserir movimentação

Caso de Uso – Inserir Movimentação	
ID	UC 004
Descrição	Este caso de uso tem por objetivo inserir a movimentação
Ator Primário	Usuário do sistema
Pré-condição	Ter Logado
Cenário Principal	1. O use case inicia quando o usuário seleciona inserir movimentação 2. O sistema exibe o componente de inserção 3. O usuário inseri ou sua receita ou suas despesas 4. O sistema incluir os dados na tabela de movimentação 5. O use case encerra
Pós-condição	Nenhuma
Cenário Alternativo	

Quadro 8 – Use Case inclusão de dados

Caso de Uso – Consultar Movimentação	
ID	UC 005
Descrição	Este caso de uso tem por objetivo que o sistema integre os dados
Ator Primário	Sistema
Pré-condição	Ter Logado
Cenário Principal	1. O use case inicia quando o usuário incluo a movimentações 2. O sistema salva os dados no banco 3. O use case encerra
Pós-condição	Nenhuma
Cenário Alternativo	

Quadro 9 – Use Case consultar informações

Caso de Uso – Consultar Movimentação	
ID	UC 006
Descrição	Este caso de uso tem por objetivo que o sistema exibir as informações já anexada
Ator Primário	Usuário do Sistema
Pré-condição	Ter inserido as movimentações
Cenário Principal	1. O use case inicia quando o usuário consulta movimentação 2. O sistema exibe os dados salvos 3. O use case encerra
Pós-condição	Nenhuma
Cenário Alternativo	

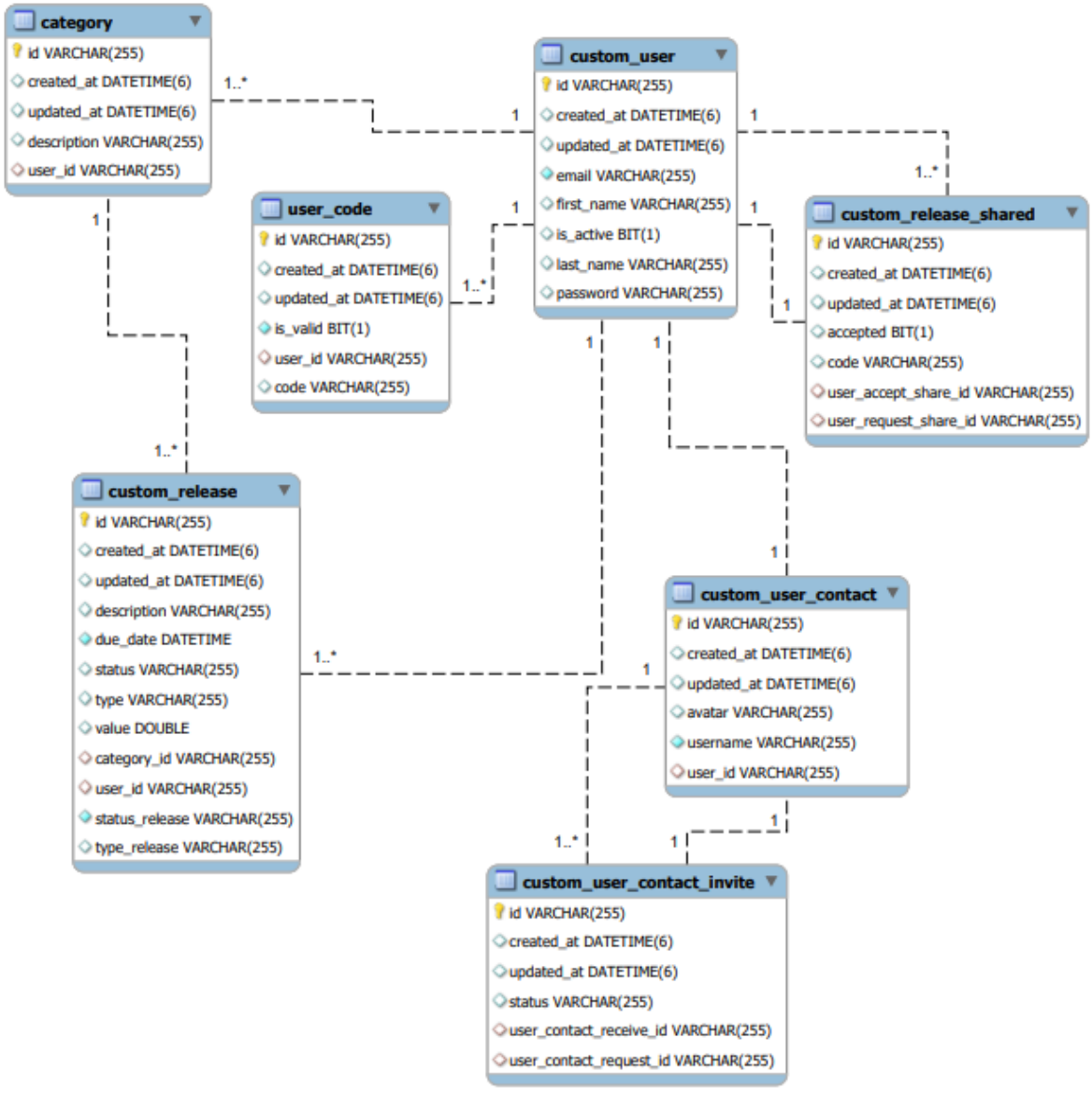
Quadro 10 – Use Case exibir informações

Caso de Uso – Consultar Movimentação	
ID	UC 007
Descrição	Este caso de uso tem por objetivo que o sistema exibir as informações já anexada no banco
Ator Primário	Sistema
Pré-condição	Ter inserido as movimentações e executado a consulta
Cenário Principal	1. O use case inicia quando o usuário consulta movimentação 2. O sistema exibe os dados salvos em seu banco de dados 3. O use case encerra
Pós-condição	Nenhuma
Cenário Alternativo	

3.7 Modelagem do banco de dados

O Diagrama de Modelo Logico (DML) é um fluxograma que ilustra como as entidades se conectam uma na outra, pelo seu banco de dados (SOMMERVILLE, 2011). O DML do sistema pode ser visto na Figura 4.

Figura 4 - Modelo lógico do banco de dados do projeto



Fonte: Autores

4. Ferramentas e Métodos ou Desenvolvimento

4.1 Ferramentas

Na programação as ferramentas são softwares que permitem que o programador manipule os códigos ao seu modo. A seguir listamos as que foram utilizadas no sistema.

4.1.1 Banco de dados

Para a construção do Banco de Dados, foi utilizada a linguagem SQL e o Banco de Dados MySQL.

O SQL (*Structured Query Language*), é uma linguagem para trabalhar com banco de dados relacional, essencialmente declarativa. O SQL foi desenvolvido pela IBM na década 70 e a partir da década de 80 passou a se tornar a linguagem padrão para gerenciamento de informações em um banco de dados relacional (TECMUNDO, 2019).

O MySQL, que é um sistema de gerenciamento de banco de dados, e armazena os dados divididos em tabelas. Criado em 1995, passou por evoluções com o tempo e hoje em dia é uma das plataformas mais utilizadas no mundo, por se tratar de um serviço estável, seguro e confiável (WEBLINK, 2022).

Alguma das GUIs (*Graphical User Interface*) de MySQL mais usadas são o MySQL Workbench, DBVisualizer, SequelPro e Naticat DB Admin Tool e é necessário que cada usuário escolha a mais adequada para o negócio.

A popularidade do MySQL se dá pelas suas principais características, que são: alto desempenho, pois independente da quantidade armazenada de dados ou de atividades de inteligência de negócio, o MySQL faz isso com velocidade otimizada; flexível e fácil de usar, levando em conta que o seu código fonte pode ser alterado de acordo com as suas necessidades; segurança, devido ao seu Sistema de Acesso Privilegiado e Gerenciamento de Contas de Usuários; e um padrão da indústria, já que é uma plataforma muito confiável e que existe muitos recursos para os desenvolvedores.

Tais vantagens justificam a escolha dessas tecnologias, além de oferecerem ótimas soluções de forma gratuita.

4.1.2 Backend

Para o gerenciamento das informações, foi desenvolvida uma API (*Application Programming Interface*) com a linguagem Java, utilizando o *framework* Spring Boot.

Java é uma linguagem de programação e ambiente computacional criada pela empresa Sun Microsystems no início da década de 90, posteriormente foi adquirida pela Oracle. O código em Java é baseado em classes e orientado a objetos (TECMUNDO, 2009).

O software desenvolvido em Java não é compilado de forma nativa, como em outras linguagens, mas através de um código intermediário chamado bytecode, interpretado e executado pela JVM (*Java Virtual Machine*), que é uma máquina virtual do Java.

Além disso, o Java possui arquitetura neutra e portável, podendo ser utilizada em diversos sistemas operacionais e/ou dispositivos, provendo segurança, alta performance e solidez.

O *Spring Boot* é um *framework* utilizado para facilitar o processo de configuração de aplicações desenvolvidas em Java com o ecossistema *Spring*, fornecendo os componentes necessários para este processo.

Dentre as principais funcionalidades do *Spring Boot,* é possível citar:

- Servidores web como o *Tomcat*, *Jetty* e *Undertow* embutidos.
- A criação de aplicações *Spring* autossuficientes.
- Configuração automática de bibliotecas *Spring* e de terceiros sempre que possível.
- Dispensar a necessidade de configuração XML e geração de código.
- Funcionalidades para ambiente de produção como métricas, *health checks* e configurações externalizadas.

Os grandes diferenciais do *Spring Boot* são a Inversão de Controle e Injeção de Dependências. A Inversão de Controle permite delegar a um elemento o controle sobre um terceiro, definindo como e quando um objeto deve ser criado ou quando um método deve ser executado. (GEEKHUNTER, 2019)

Já a Injeção de Dependências é uma das maneiras de implementar a Inversão de Controle, pois com sua utilização cada classe pode focar exclusivamente na utilização dos recursos das dependências para realizar as tarefas necessárias.

Além da configuração inicial, o *Spring* oferece outras dependências que auxiliam no desenvolvimento do projeto, como: *Spring Data* (utilizada para facilitar a comunicação com o banco de dados); *Spring Web* (oferece todo o suporte para o desenvolvimento *web*); *Spring Security* (fornece recursos necessários para que a aplicação seja segura); entre outras (DEVMEDIA, *sd*).

4.1.3 Mobile

Para o desenvolvimento da aplicação mobile, foi escolhido o *framework* Flutter, que utiliza a linguagem Dart.

O Dart é uma linguagem de programação, inicialmente criada pela Google, em 2011, fortemente tipada com o intuito de substituir o JavaScript para desenvolvimento de *scripts* em páginas *web*, mas hoje com a evolução dessa linguagem, ela pode ser considerada uma linguagem multiplataforma, atendendo aplicações *mobile*, *desktop* e *web*, sendo vista como uma excelente opção para desenvolvedores que buscam uma opção multiplataforma robusta e eficiente. (TREINAWEB A, s.d)

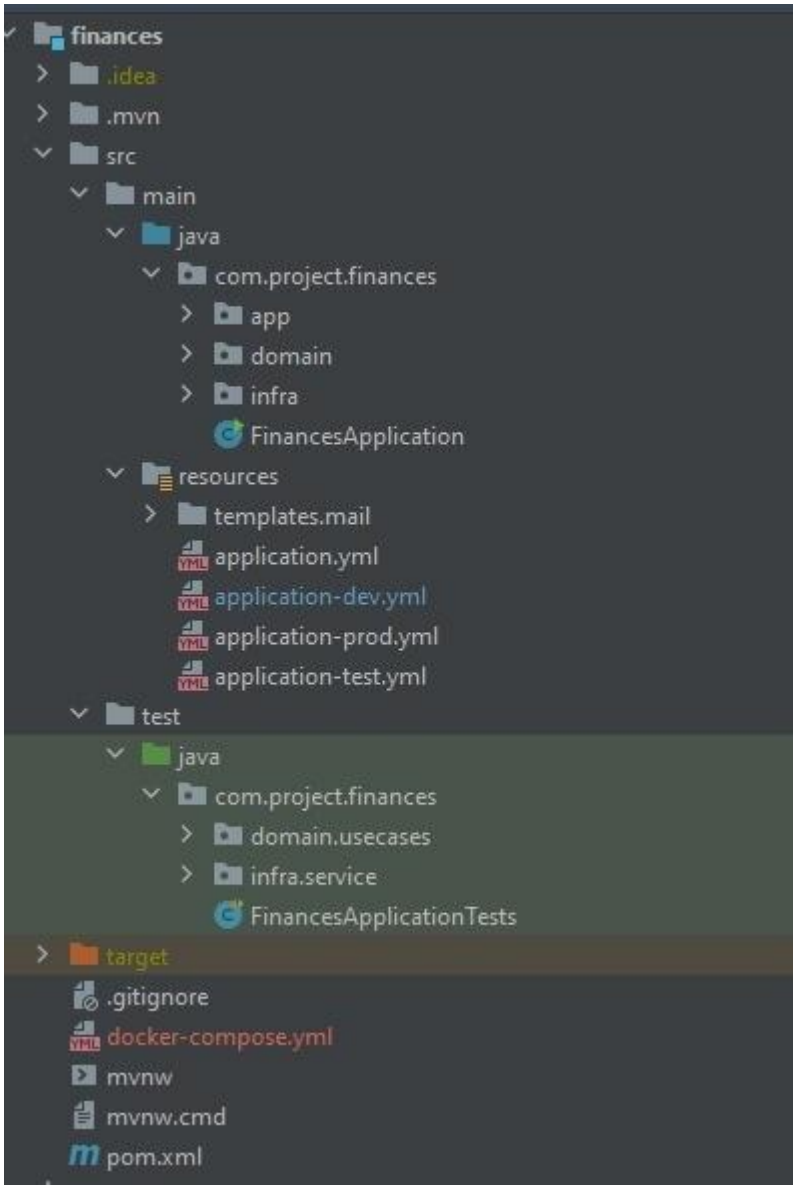
Já por sua vez, o Flutter é um *framework* facilitador no desenvolvimento de diversas aplicações. Sendo multiplataforma e sua linguagem base utilizada ser o Dart, com ele é possível desenvolver aplicações em qualquer sistema operacional, o que permite também a criação de apps nativos, auxiliando nos acessos aos recursos nativos do dispositivo, tanto Android ou IOS a partir de um único código-base por sua compilação ser feita para a linguagem base do dispositivo, e isso tudo gerando um maior desempenho nas aplicações criadas com ele. Na sua construção é possível utilizar uma série de *widgets* customizáveis, já desenvolvidos de forma reativa para a construção de uma interface flexível para o desenvolvedor e atrativa para o usuário.

Tendo isso em vista, a escolha dessas tecnologias é de grande benefício para o desenvolvedor e para o cliente, por sua versatilidade, baixa curva de aprendizado, e praticidade, tudo isso por sua enorme biblioteca de *widgets* disponíveis (DIGITALHOUSE, 2020).

4.2 Métodos ou Desenvolvimento

Para o desenvolvimento do *backend*, foi criado um projeto em Java utilizando a arquitetura *onion*, desacoplando assim as classes de domínio e de dados das camadas externas. A Figura 5 apresenta uma visão macro da aplicação

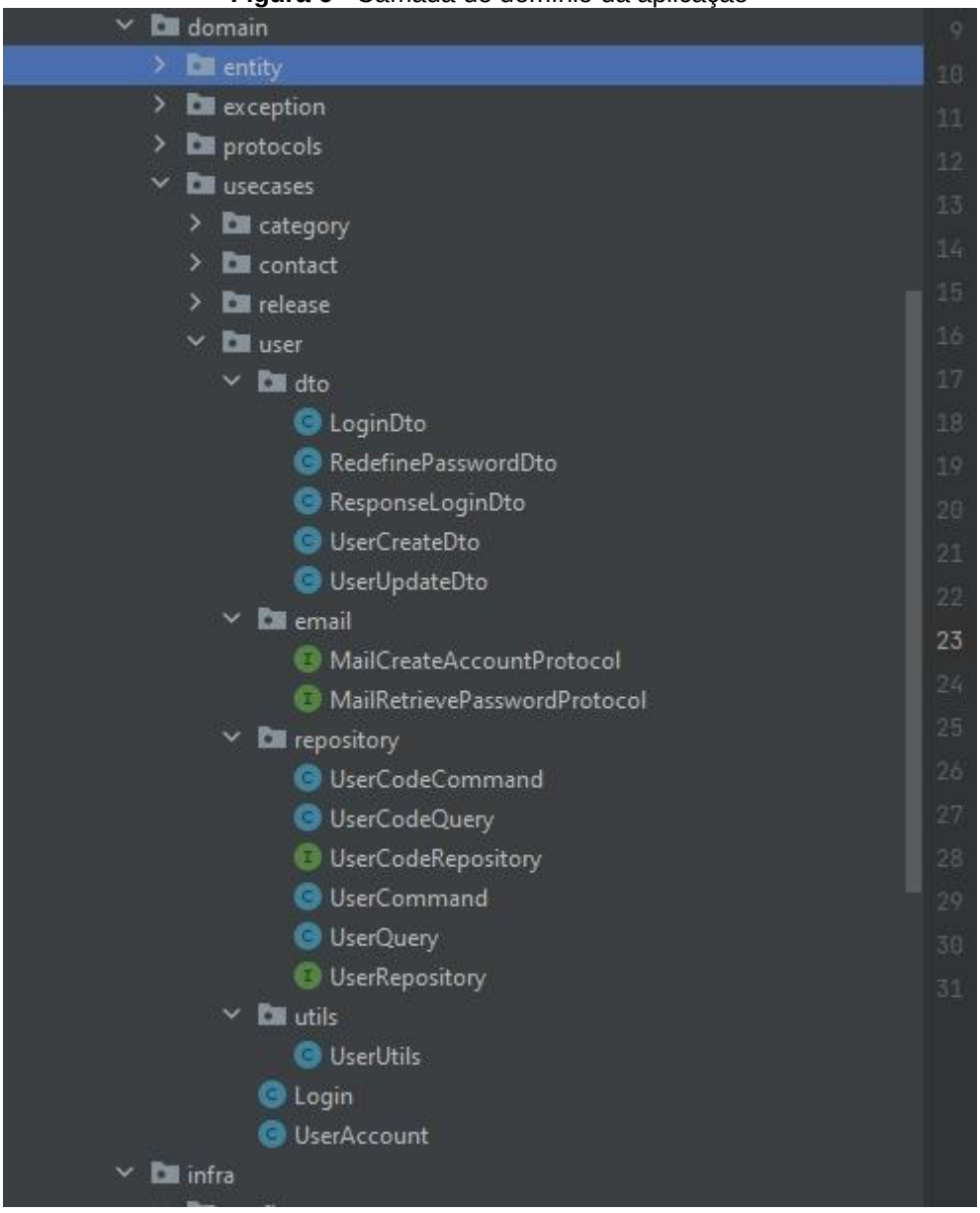
Figura 5 - Estrutura do projeto Java



Fonte: Autores

Na camada de domínio, foram criadas as classes de entidade, *repository* e *adapters*, mantendo assim o *core* da aplicação isolado da camada de entrada e de ambientes externos. Esta camada é responsável por toda a lógica do projeto e pela comunicação com o banco de dados. A Figura 6 mostra de forma mais detalhada a estruturação desta camada.

Figura 6 - Camada de domínio da aplicação

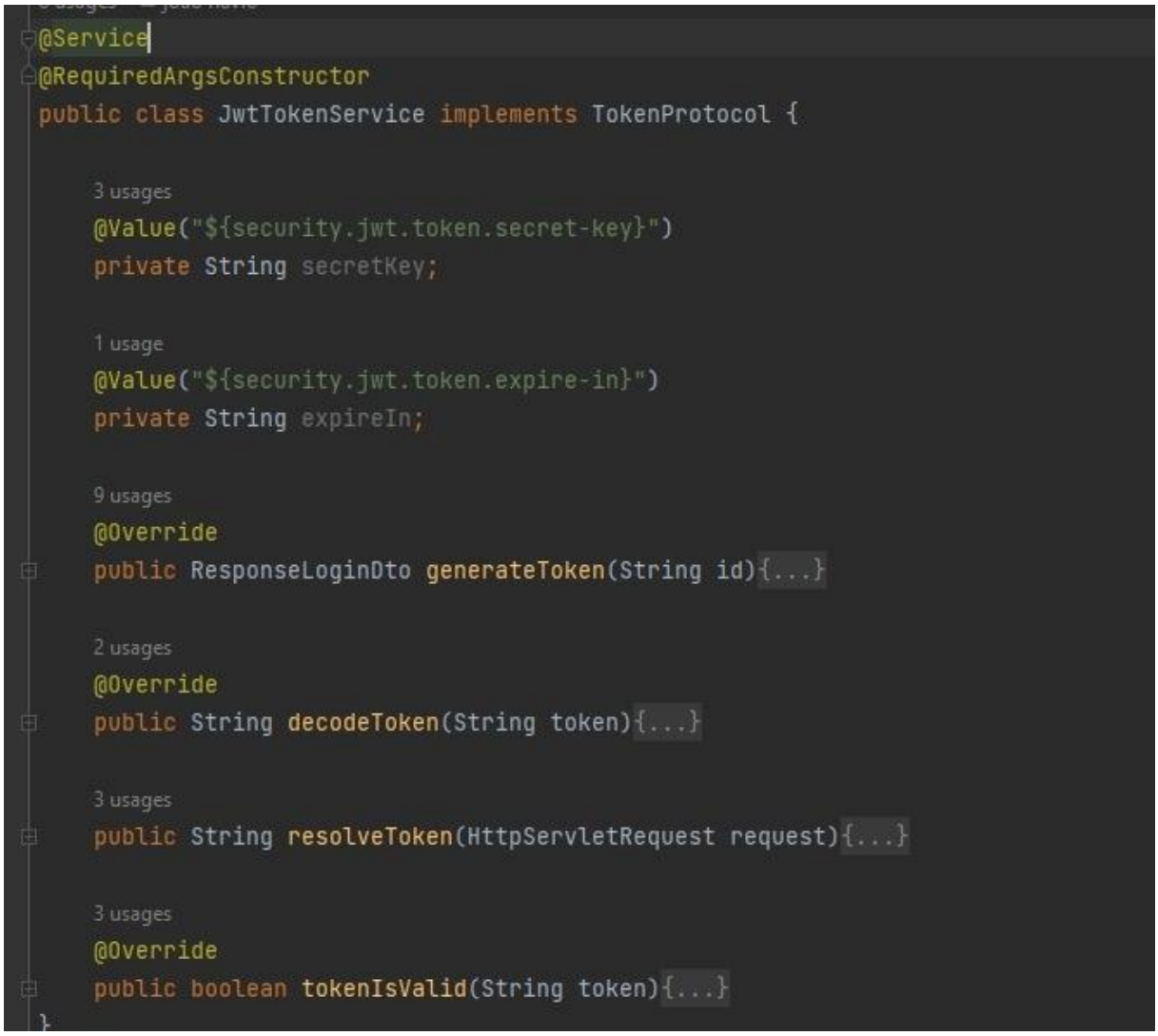


Fonte: Autores

Ainda sobre a camada de domínio, o *repository* foi separado em serviços de comandos e de *query*. Além destes, tem-se classes utilitárias e algumas interfaces de *e-mail*, pois será feito o envio de *e-mail* para ativação do usuário e recuperação de senha.

Para a autenticação do usuário, é gerado um *token jwt*, que é passado no *header* de cada requisição. A Figura 7 apresenta a classe responsável pela geração deste *token*.

Figura 7 - Classe responsável pela geração e validação do token

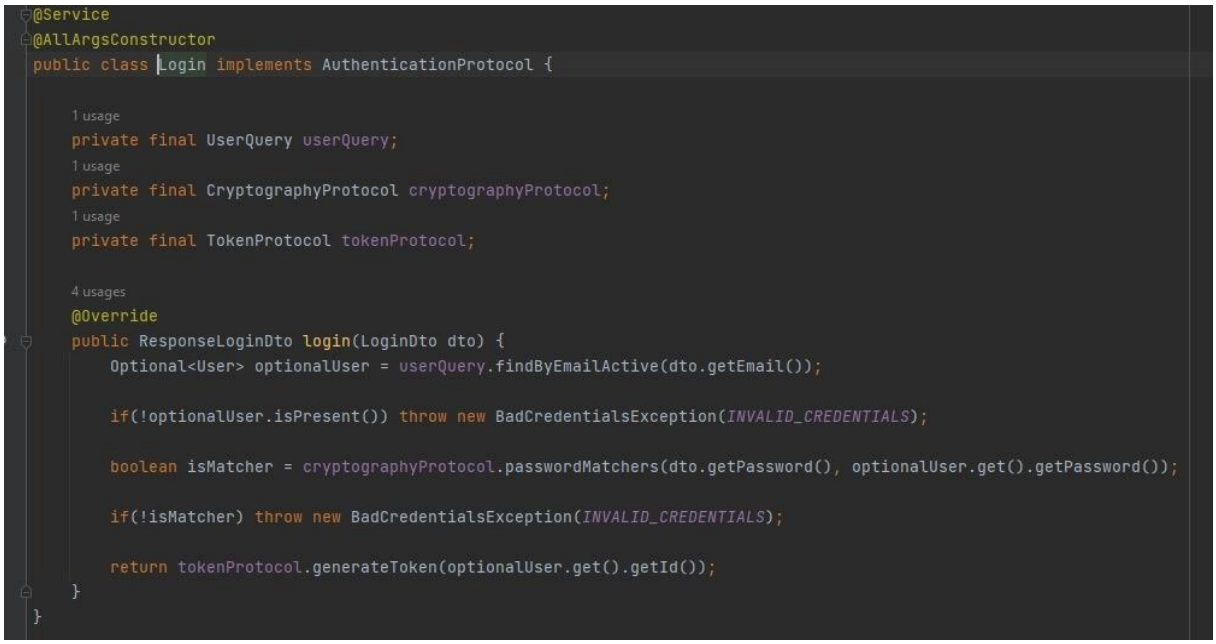


```
@Service
@RequiredArgsConstructor
public class JwtTokenService implements TokenProtocol {

    3 usages
    @Value("${security.jwt.token.secret-key}")
    private String secretKey;

    1 usage
    @Value("${security.jwt.token.expire-in}")
    private String expireIn;

    9 usages
    @Override
    public ResponseLoginDto generateToken(String id){...}

    2 usages
    @Override
    public String decodeToken(String token){...}

    3 usages
    public String resolveToken(HttpServletRequest request){...}

    3 usages
    @Override
    public boolean tokenIsValid(String token){...}
}
```

Fonte: Autores

A classe é referenciada na classe de *Login*, através da injeção de dependências. Desta forma, esta classe, que estende de *AuthenticationProtocol*, pode utilizar a geração de *token* em seus métodos de autenticação. Ainda sobre a classe *Login*, há um método que busca no banco de dados um registro ativo com o *e-mail* informado pelo usuário, em caso positivo é feita a comparação da senha informada com a senha do registro encontrado no banco (esta que se encontra criptografada) e, se ambas forem idênticas, o token é gerado e retornado no *response* da requisição. Em caso negativo para alguma etapa, é retornada uma exceção informando que as credenciais inseridas são inválidas. A Figura 8 mostra a implementação da classe de *Login*.

Figura 8 - Implementação da classe Login

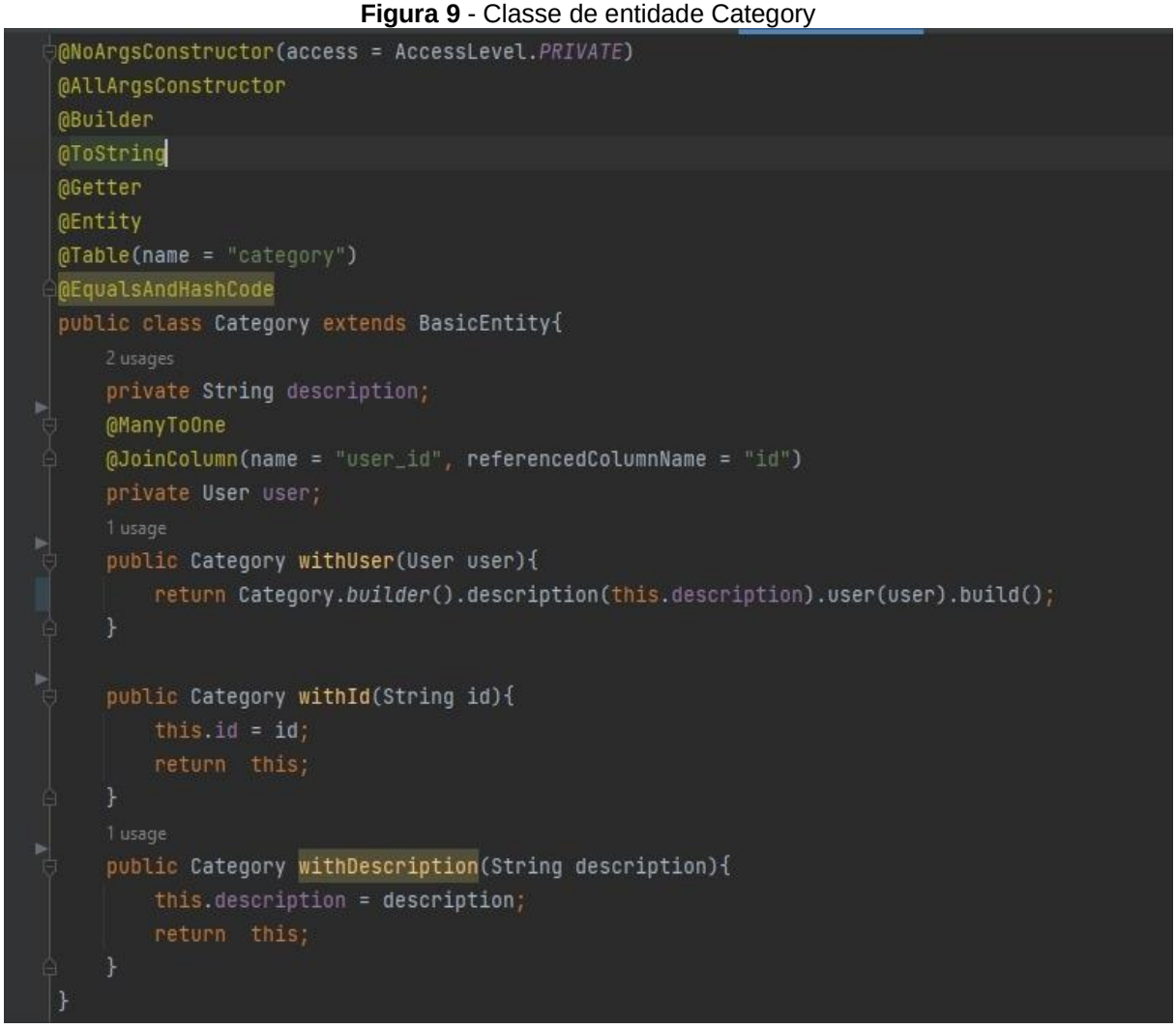


```java
@Service
@AllArgsConstructor
public class Login implements AuthenticationProtocol {

    1 usage
    private final UserQuery userQuery;
    1 usage
    private final CryptographyProtocol cryptographyProtocol;
    1 usage
    private final TokenProtocol tokenProtocol;

    4 usages
    @Override
    public ResponseLoginDto login(LoginDto dto) {
        Optional<User> optionalUser = userQuery.findByEmailActive(dto.getEmail());

        if(!optionalUser.isPresent()) throw new BadCredentialsException(INVALID_CREDENTIALS);

        boolean isMatcher = cryptographyProtocol.passwordMatchers(dto.getPassword(), optionalUser.get().getPassword());

        if(!isMatcher) throw new BadCredentialsException(INVALID_CREDENTIALS);

        return tokenProtocol.generateToken(optionalUser.get().getId());
    }
}
```

Fonte: Autores

Na Figura 9 tem-se a classe de entidade responsável pelas categorias. Tal classe possui anotações da ferramenta *Lombok* que geram construtores, *getters*, *setters* e outros métodos de maneira automática; além de anotações que referenciam o banco de dados, como *Table* (que possui o atributo nome da tabela), *ManyToOne* (indica o relacionamento entre tabelas; neste caso, n para 1) e *JoinColumn* (cria uma chave estrangeira de uma tabela em outra). Por último, esta classe estende de *BasicEntity*, que é uma classe de entidade com atributos genéricos utilizados no banco de dados como *id*, *createdAt* e *updatedAt*.

Figura 9 - Classe de entidade Category

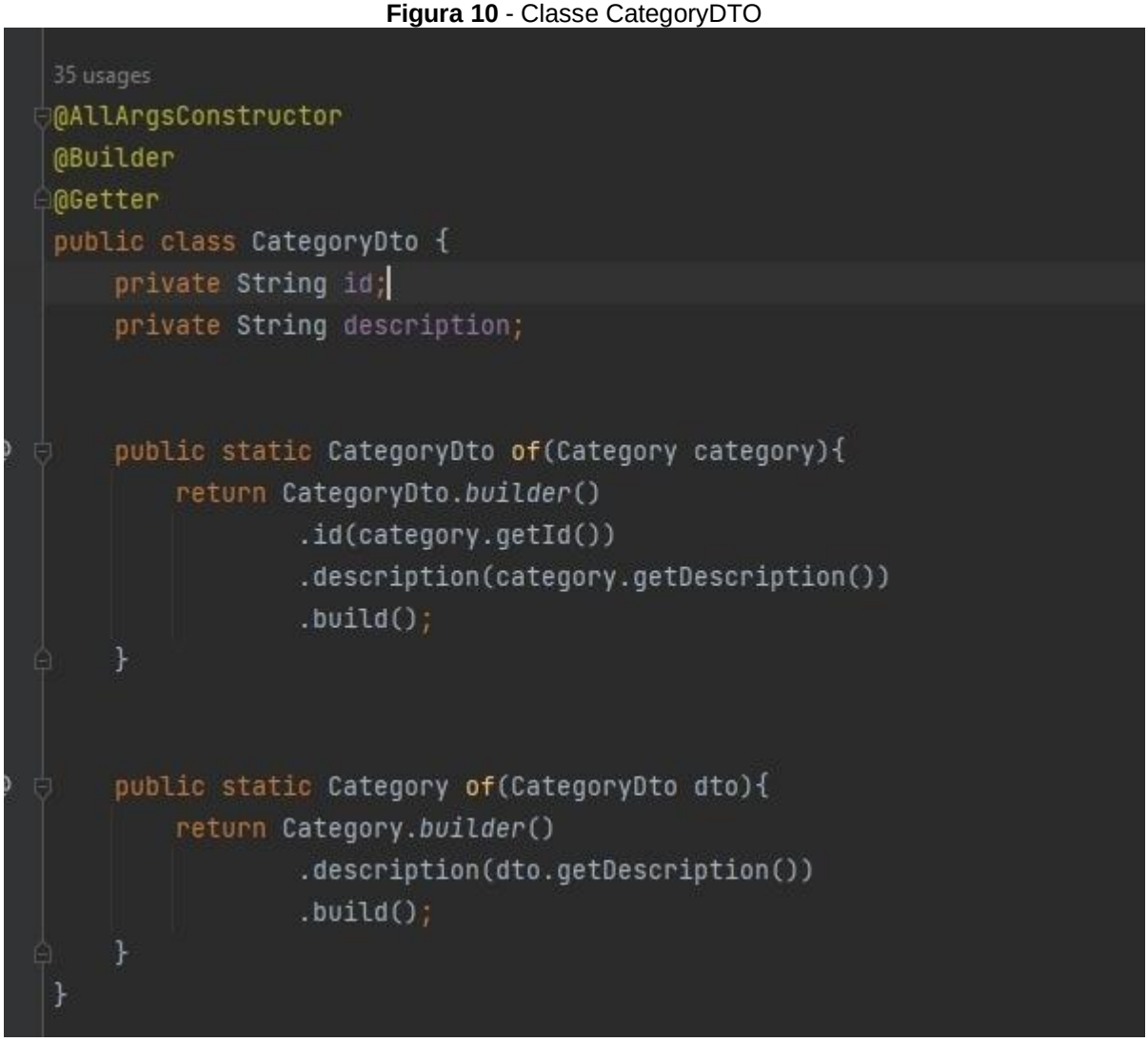


```java
@NoArgsConstructor(access = AccessLevel.PRIVATE)
@AllArgsConstructor
@Builder
@ToString
@Getter
@Entity
@Table(name = "category")
@EqualsAndHashCode
public class Category extends BasicEntity{
    2 usages
    private String description;
    @ManyToOne
    @JoinColumn(name = "user_id", referencedColumnName = "id")
    private User user;
    1 usage
    public Category withUser(User user){
        return Category.builder().description(this.description).user(user).build();
    }

    public Category withId(String id){
        this.id = id;
        return  this;
    }
    1 usage
    public Category withDescription(String description){
        this.description = description;
        return  this;
    }
}
```

Fonte: Autores

Além das classes de entidade, têm-se as classes de transferência de objeto (DTO). Este tipo de classe é utilizado na entrega do objeto para o cliente (camada de aplicação, controllers, entre outros). Os atributos necessários são convertidos da entidade mapeada do banco para a classe DTO e retornados no formato desejado, neste caso é retornado no formato json. Desta forma, a requisição se torna mais leve e performática. A Figura 10 apresenta a classe de transferência para as categorias.

Figura 10 - Classe CategoryDTO

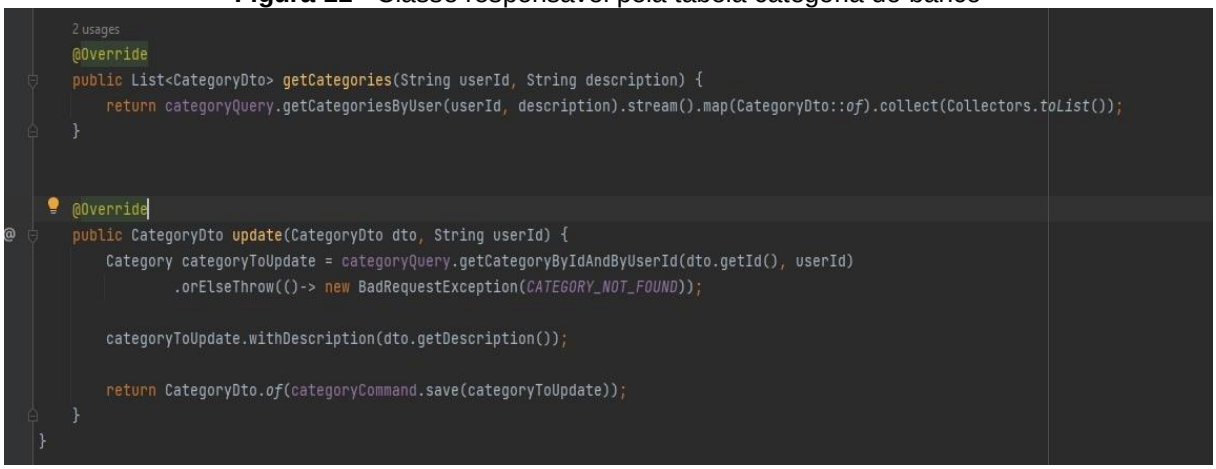


```
35 usages
@AllArgsConstructor
@Builder
@Getter
public class CategoryDto {
    private String id;
    private String description;


    public static CategoryDto of(Category category){
        return CategoryDto.builder()
                .id(category.getId())
                .description(category.getDescription())
                .build();
    }


    public static Category of(CategoryDto dto){
        return Category.builder()
                .description(dto.getDescription())
                .build();
    }
}
```

Fonte: Autores

Por último, tem-se um exemplo de como é feita a implementação da listagem e alteração de categorias na classe de serviço (Figura 11). Nessa listagem, é possível perceber que é chamada a classe responsável pela consulta de categorias no banco de dados e o retorno desta consulta é convertido em um objeto do tipo CategoryDto, utilizando uma expressão lambda. Já para a alteração de uma categoria, a mesma é consultada no banco de dados através de seu id e do userId e, caso não seja encontrado algum registro, é retornada uma exceção; em caso positivo, a descrição da categoria é atualizada e as novas informações são salvas no banco de dados.

Figura 11 - Classe responsável pela tabela categoria do banco



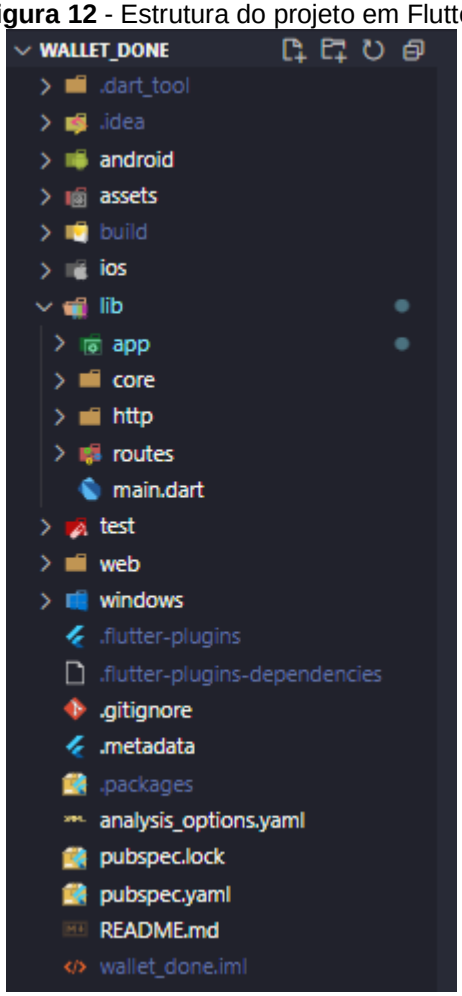
```
2 usages
@Override
public List<CategoryDto> getCategories(String userId, String description) {
    return categoryQuery.getCategoriesByUser(userId, description).stream().map(CategoryDto::of).collect(Collectors.toList());
}


@Override
public CategoryDto update(CategoryDto dto, String userId) {
    Category categoryToUpdate = categoryQuery.getCategoryByIdAndByUserId(dto.getId(), userId)
            .orElseThrow(()-> new BadRequestException(CATEGORY_NOT_FOUND));

    categoryToUpdate.withDescription(dto.getDescription());

    return CategoryDto.of(categoryCommand.save(categoryToUpdate));
}
}
```

Fonte: Autores

Após todo o processo e validação da API, foi iniciado o desenvolvimento do aplicativo *mobile.* O aplicativo foi desenvolvido com o *framework Flutter* na versão estável 2.10.5 e a linguagem *Dart* na sua versão estável 2.16.2, este *framework* com seu objetivo voltado a aplicações híbrida. Consta na estrutura em seu diretório (Figura 12) configurações para a compilação do código para *iOS* e *Android*, estendendo a compatibilidade para aplicação *Web* e Desktop. A estrutura escolhida foi baseada na estrutura *getx_pattern*, usando a base de organização em módulos, na estrutura consta a pasta chamada ‘assets’ que é direcionada para arquivos de imagens e semelhantes para a utilização dela na aplicação, a pasta ‘lib’, onde este é o local que todo o desenvolvimento será feito, e por fim ao gerar uma aplicação *Flutter* é gerado um arquivo chamado ‘pubspec.yaml’ (Figura 13) que nele são inseridas as configurações e dependências utilizadas no projeto. As principais dependências utilizadas foram:

- *get* – utilizado no gerenciamento de estado, gestão de dependências, gerenciamento de rotas e *widgets* adicionais.
- *get_storage* – utilizada para o armazenamento tipo chave-valor de informações localmente, como token e o id do usuário logado;
- *dio* – possibilita as chamadas requisições para serviços externos (neste caso, as requisições à API desenvolvida em Java);
- *intl* – permite a captação de transformação de dados no formato de horas e datas.

Figura 12 - Estrutura do projeto em Flutter



Fonte: Autores

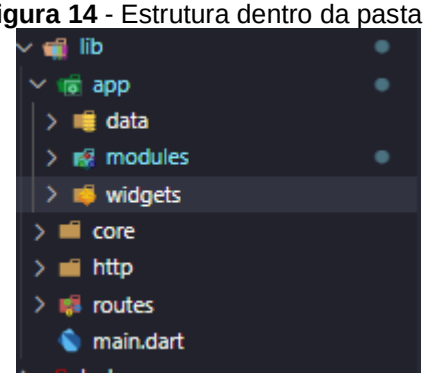
Figura 13 - Trecho do código no arquivo Pubspec.yaml

```yaml
name: wallet_done
description: A new Flutter project.

publish_to: 'none' # Remove this line if you wish to publish to pub.dev

version: 1.0.0+1

environment:
  sdk: ">=2.16.1 <3.0.0"

dependencies:
  flutter:
    sdk: flutter
  flutter_localizations:
    sdk: flutter

  get: ^4.6.1
  dio: ^4.0.4
  get_storage: ^2.0.3
  random_avatar: ^0.0.6
  shimmer: ^2.0.0
  cupertino_icons: ^1.0.2
  intl: ^0.17.0
  brasil_fields: ^1.4.2
```

Fonte: Autores

No diretório lib, há três divisões baseadas na estrutura escolhida do *getx pattern* (Figura 14), onde primeiramente temos a pasta app que contém a maior parte do código de desenvolvimento voltado ao aplicativo, a pasta core que é voltada para utilidades e serviços, a pasta http que nela é definido o local de requisição externa e por fim routes que é o diretório responsável por conter os nossos arquivos que gerencia nossas rotas. Na pasta app encontra-se subdivisões que são: a pasta *modules* que é responsável por armazenar todos arquivos das telas do aplicativo subdivididos em pastas, logo depois a *data* que é apenas um diretório responsável por guardar tudo relacionado aos seus dados, ou seja, seu *repository*, suas classes e *providers* e, por fim, a *widgets* que armazena todos os componentes que podem ser reaproveitados em qualquer parte do aplicativo.

Figura 14 - Estrutura dentro da pasta lib



Fonte: Autores

O arquivo *main.dart* (Figura 15) contém a configuração principal que é a primeira configuração que inicia no carregamento do aplicativo. Nele há o direcionamento da primeira rota a ser apontada e onde é encaminhado ao arquivos com todas rotas definidas, também responsável por definir a localização do dispositivo e o tema definido no aplicativo.

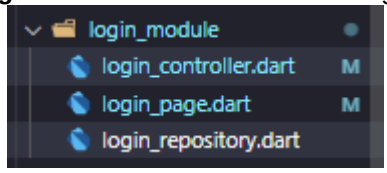
Figura 15 - Arquivo principal main.dart

```dart
void main() async {
  await GetStorage.init();
  runApp(
    GetMaterialApp(
        localizationsDelegates: const [
          GlobalMaterialLocalizations.delegate,
          GlobalCupertinoLocalizations.delegate,
          GlobalWidgetsLocalizations.delegate
        ],
        supportedLocales: const [
          Locale('pt', 'BR')
        ],
        title: 'WalletDone',
        debugShowCheckedModeBanner: false,
        defaultTransition: Transition.fade,
        getPages: AppPages.routes,
        initialRoute: Routes.INITIAL,
        theme: appThemeData), // GetMaterialApp
  );
}
```

Fonte: Autores

Na organização da pasta *modules* há três arquivos distintos e cada um com sua função, levando como exemplo o módulo de *login* (Figura 16), tem-se nele o arquivo *page, repository*, e *controller,* sendo o primeiro com o objetivo de ser a parte visual do aplicativo, o segundo de ser o responsável pela chamada da api e devolver a resposta e o terceiro de conter a parte lógica do aplicativo e suas funções e fazer ponte entre o primeiro e segundo.

Figura 16 - Pasta do módulo de login

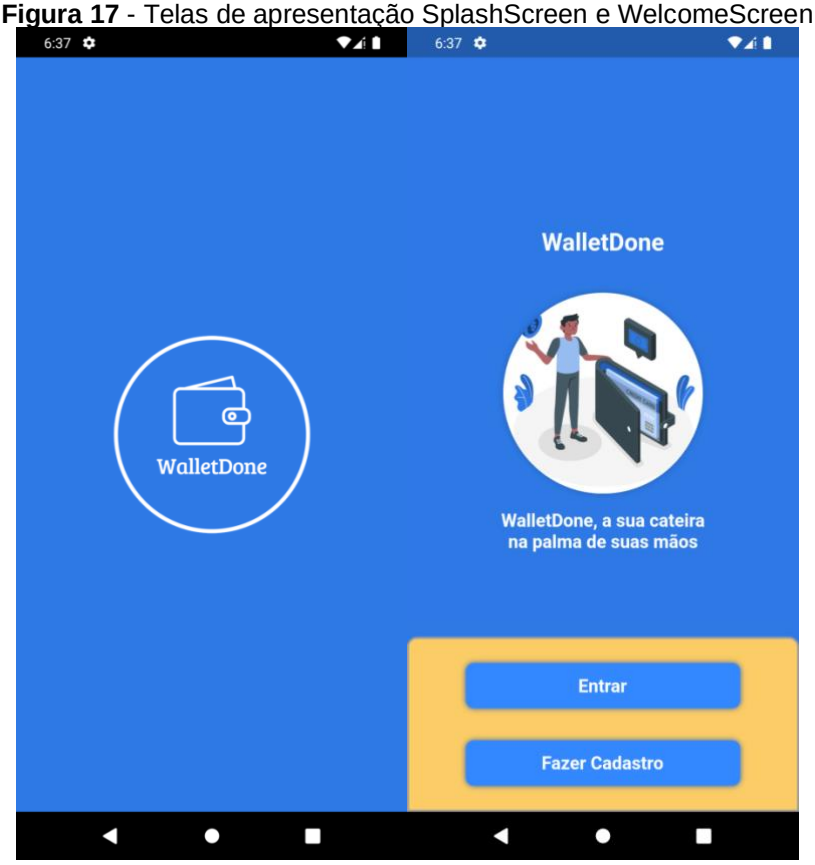


Fonte: Autores

5 Resultados e Discussão

O aplicativo, denominado *WalletDone,* com a proposta apresentada no projeto, possui as seguintes telas. A Figura 17 mostra a Tela de *SplashScreen* e *WelcomeScreen* que segue com a logo do aplicativo e dá a primeira escolha ao usuário.

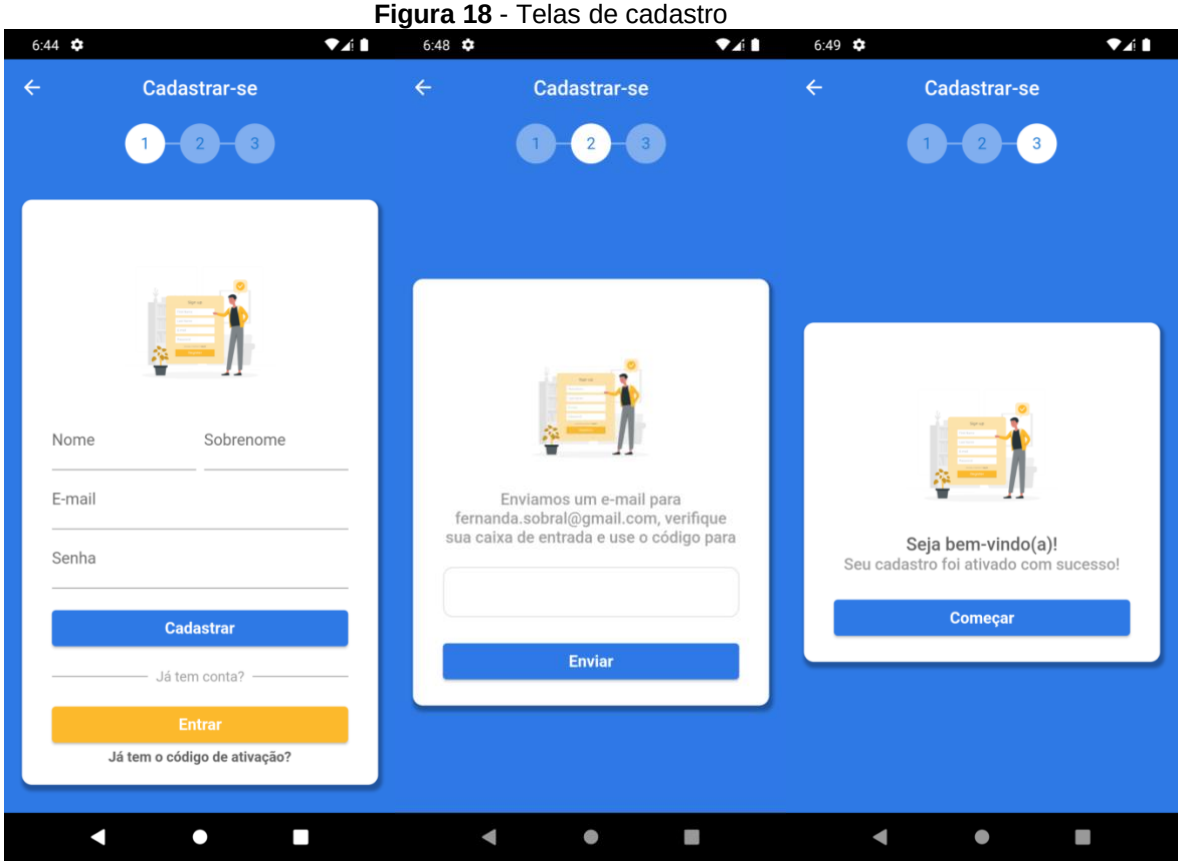
Figura 17 - Telas de apresentação SplashScreen e WelcomeScreen



Fonte: Produzido pelos autores

Na Figura 18, tem-se a primeira funcionalidade caso o usuário escolha a opção de fazer cadastro. A tela de cadastro é realizada em 3 etapas: o preenchimento dos dados, a validação do email com código gerado automático para o usuário e, por fim, a confirmação e avanço de tela.

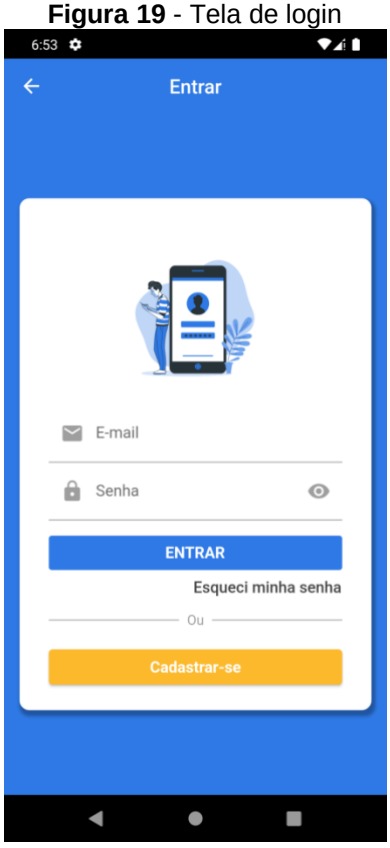
Figura 18 - Telas de cadastro



Fonte: Autores

Caso o usuário queira fazer login em sua conta já cadastrada, a Figura 19 apresenta a tela em que o usuário insere seus dados já cadastrados para acessar sua conta.

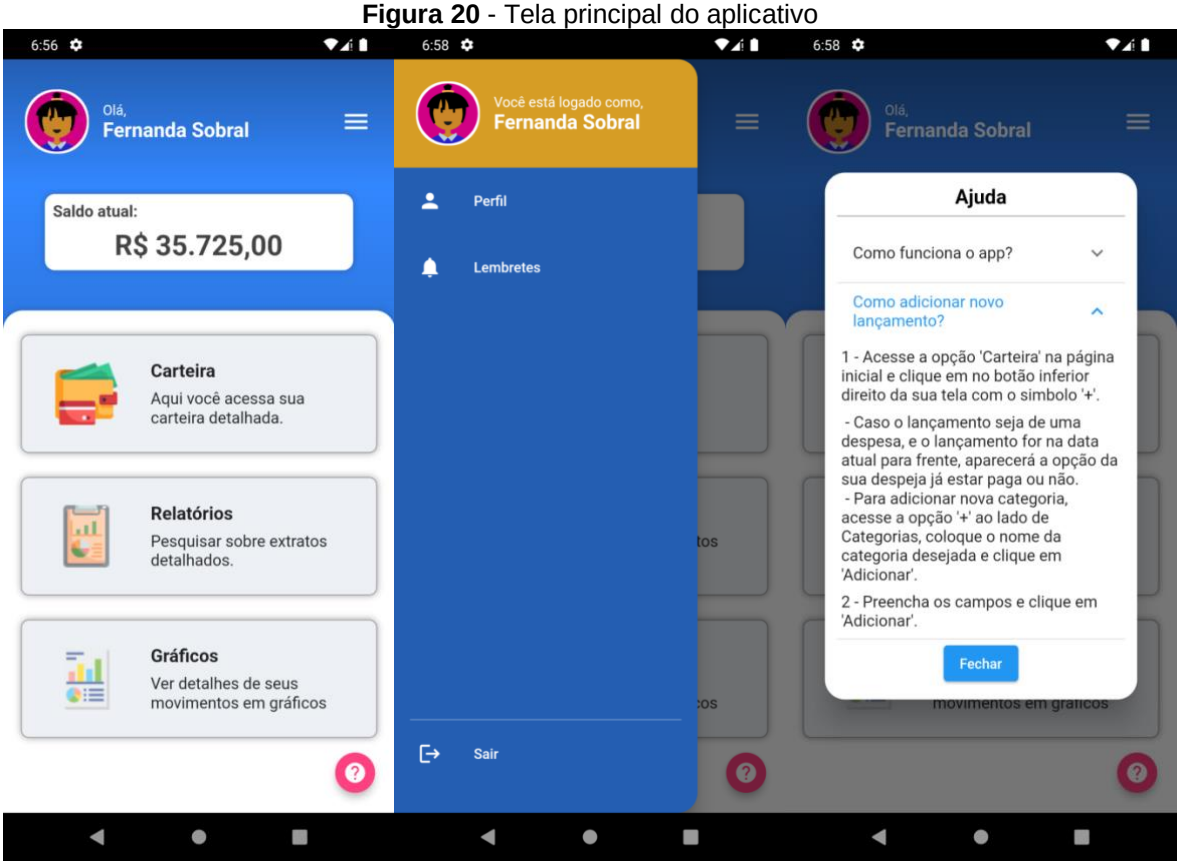
Figura 19 - Tela de login



Fonte: Autores

Ao realizar o login, será redirecionado para a página principal *HomeScreen* (Figura 20), onde é apresentado o nome do usuário logado na parte superior da tela, com a opção de abrir o *drawer,* que é um menu que fornece a opção de sair da conta, e mais duas opções que serão implementadas no futuro: “Perfil” e “Lembretes”. As opções na tela principal em funcionamento há a Carteira que foca no objetivo principal do aplicativo; a opção Relatórios que dará a oportunidade do usuário pesquisar sobre o extrato detalhado e filtrar de acordo com o interesse do usuário e a opção Gráficos, que por sua vez dará a possibilidade do usuário visualizar os detalhes de suas transações em gráficos. Essas duas serão implementadas futuramente. Para dar acessibilidade, temos um botão flutuante no canto inferior direito que há dicas de como usar o aplicativo.

Figura 20 - Tela principal do aplicativo

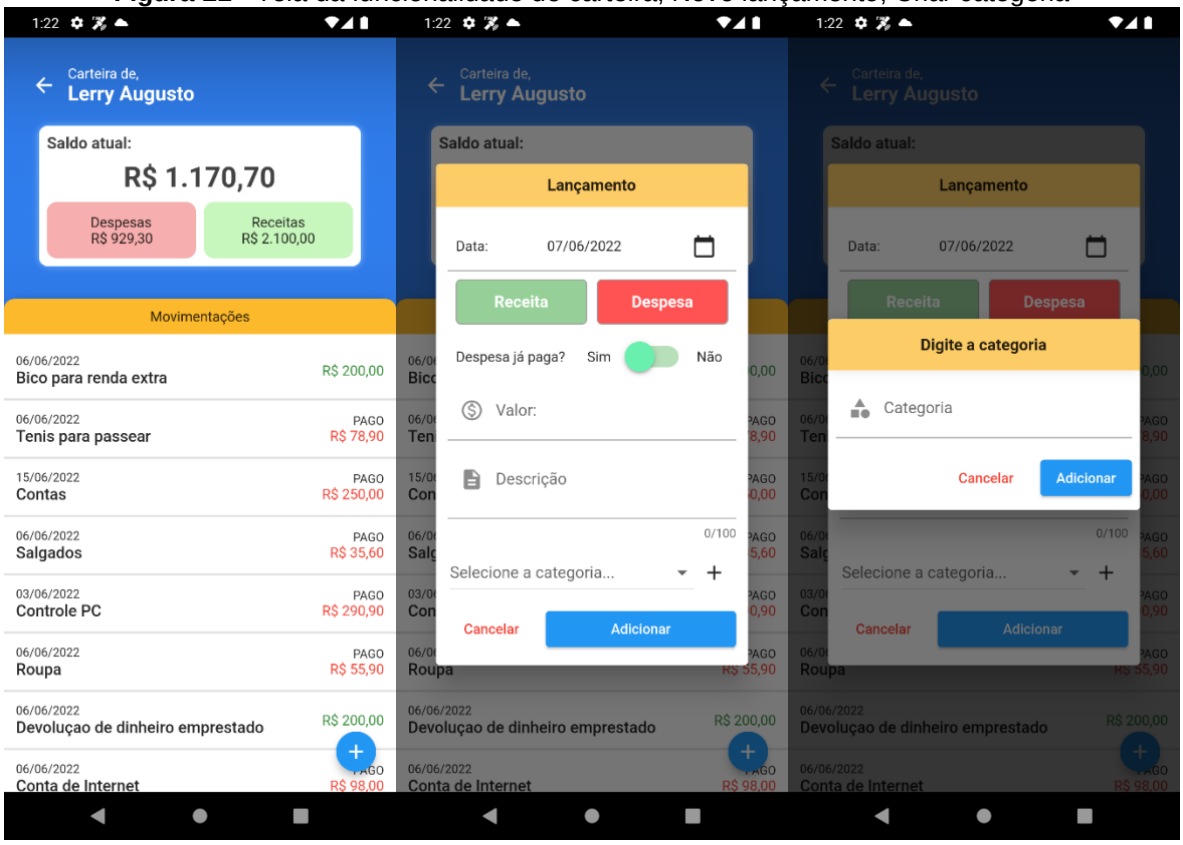


Fonte: Autores

Na página da Carteira (Figura 21), existe na parte superior da tela o saldo atual do usuário sendo a soma das despesas e receitas, e logo abaixo o total das somas das despesas e o total de somas das receitas.

Depois é exibido para o usuário uma lista de movimentações com todos os lançamentos feitos pelo usuário. No canto inferior direito da tela há um botão que direciona o usuário para um novo lançamento, abrindo uma caixa para ser inserido todos os dados sobre o lançamento que será criado, dando a oportunidade do usuário escolher a data do lançamento, podendo ser dias anteriores ou posteriores ao atual. Caso o usuário escolha do dia atual para frente, irá habilitar um botão para escolher se caso o tipo do lançamento for uma despesa. O usuário escolherá se ela já foi paga ou não, deverá inserir o valor, a descrição e a categoria. Caso o usuário não tenha a categoria desejada, poderá criar uma nova, clicando no botão com o símbolo de ‘+’, que abrirá outra tela para inserir a categoria.

Figura 21 - Tela da funcionalidade de carteira, Novo lançamento, Criar categoria



Fonte: Autores

Por fim, quando o usuário adicionar novo lançamento, será incluído e atualizado na lista de movimentações da carteira, e também na tela inicial do aplicativo, deixando-o informado de todos seus gastos e somas dele.

Considerações finais

Neste trabalho foi apresentado a estrutura de desenvolvimento de uma carteira pessoal digital ao usuário. O propósito do aplicativo é facilitar o controle de suas finanças de uma maneira simples e prática, retornando ao usuário seu saldo e o lembrando de suas contas a vencer. A origem da ideia do aplicativo é própria dos autores por ter dificuldades de colocar suas contas no papel e querer ter de forma digital e prática no próprio celular pessoal.

Com toda a ideia projetada e a documentação estruturada, o desenvolvimento do sistema exclusivamente no formato de aplicativo *mobile*, por meio da linguagem

Dart, com *framework Flutter* consumindo uma API desenvolvida em Java e com todos os dados armazenados no banco de dados MySQL.

A partir do cadastro pessoal, a pessoa tem acesso a carteira digital, podendo incluir e monitorar seus lançamentos. Vale ressaltar que a interface gráfica é um dos destaques do aplicativo, por ser intuitiva e amigável, com opções claras e de fácil acesso.

Durante o longo período de desenvolvimento do aplicativo, um dos principais desafios enfrentados foi o aprendizado de uma nova ferramenta de programação e de um novo *framework*, tendo por si só diferentes meios de desenvolvimento, estrutura e gerenciamento de estado. Em contrapartida, a documentação do *Flutter* é completa, auxiliando nos estudos e implementações do aplicativo.

Com objetivos futuros, pretende-se incrementar funções de extrato, exportando para pdf, função de carteira conjunta onde com convites, poderá acessar uma nova página com extrato e detalhes das contas nela incluída, mas restringindo a edição apenas a conta própria do usuário, também opção de incluir movimentações de cartão de crédito, adicionando prazos e parcelas. Por fim, pretendemos buscar formas automatizar a aplicação como citado no aplicativo Olivia Ia para sincronizar as movimentações do bancarias do usuário ao aplicativo.

Concluímos que o projeto foi um sucesso, baseado no objetivo principal de ser uma carteira digital e que futuramente poderá se expandir e auxiliar muitas pessoas na parte de controlar as finanças do usuário.

Referências

AGENCIABRASIL, **Pesquisa revela que 58% dos brasileiros não se dedicam às próprias finanças.** 28/03/2018. Disponível em < https://agenciabrasil.ebc.com.br/economia/noticia/2018-03/pesquisa-revela-que-58-dos-brasileiros-nao-se-dedicam-proprias-financas#:~:text=Seis%20em%20cada%2010%20brasileiros,pagar%20as%20contas%20do%20m%C3%AAs.>. Acesso em: 25/11/2021

CANVAS. **Coloque as suas ideias em prática**. 05/05/2022. Disponível em: <https://sebraecanvas.com/?checkedSAS=false#/>. Acesso em: 05.mai.2022.

DEVMEDIA. **O que é Spring?** [s.d]. Disponível em: <https://www.devmedia.com.br/exemplo/como-comecar-com-spring/73>. Acesso em: 07.mai.2022.

DIGITALHOUSE. **Introdução ao Flutter: como funciona o framework e sua linguagem Dart,** 13/07/2020. Disponível em: <https://www.digitalhouse.com/br/blog/o-que-e-flutter-e-como-funciona/>. Acesso em: 06.mai.2022.

GEEKHUNTER. **Spring Boot: Tudo que você precisa saber!** 08/10/2019. Disponível em: <https://blog.geekhunter.com.br/tudo-o-que-voce-precisa-saber-sobre-o-spring-boot/>. Acesso em: 07.mai.2022.

OLIVIA AI. **Serviços financeiros inteligentes**. Disponível em: <https://www.olivia.ai/>. Acesso em: 07.mai.2022.

PRESSMAN, Roger S. **Engenharia de Software: Uma Abordagem Profissional**. 7ª Ed. São Paulo: Editora Sênior Arysinha Jacques Affonso, 2011.

REIS, Glauco. S. **Modelagem de Processos de Negócios com BPMN - Curso Completo**. 1ª Ed. São Paulo: Editora PortalBPM, 2008.

SOMMERVILLE, Ian. **Engenharia de Software**. 9ª Ed. São Paulo: Editora Pearson Addison - Wesley, 2011.

TECMUNDO. **O que é Java?** 09/09/2009. Disponível em <https://www.tecmundo.com.br/programacao/2710-o-que-e-java-.htm>. Acesso em: 07.mai.2022.

TECMUNDO. **O que é SQL e para que ele serve?** 03/10/2019. Disponível em <https://www.tecmundo.com.br/software/146482-sql-que-ele-serve.htm>. Acesso em: 09.mai.2022.

TECNOBLOG. **O que é Java? [Guia para iniciantes]**. *sd*. Disponível em <https://tecnoblog.net/responde/o-que-e-java-guia-para-iniciantes/>. Acesso em: 07.mai.2022.

TREINAWEB A. **O que é Dart?** *sd*. Disponível em: <https://www.treinaweb.com.br/blog/o-que-e-dart>. Acesso em: 06.mai.2022.

TREINAWEB B. **O que é o Spring Boot?** *sd*. Disponível em: <https://www.treinaweb.com.br/blog/o-que-e-o-spring-boot>. Acesso em: 07.mai.2022.

WEBLINK. **O que é MySQL: Guia para Iniciantes.** 25/09/2019. Disponível em: <https://www.weblink.com.br/blog/o-que-e-mysql/>. Acesso em: 09.mai.2022.