



igti

RELATÓRIO

PROJETO APLICADO

Instituto de Gestão e Tecnologia da Informação
Relatório do Projeto Aplicado

Aplicativo para controle financeiro

Guilherme Boschetti

Orientador(a): Ricardo Brito Alves

Julho/2022





GUILHERME BOSCHETTI

INSTITUTO DE GESTÃO E TECNOLOGIA DA INFORMAÇÃO

RELATÓRIO DO PROJETO APLICADO

APLICATIVO PARA CONTROLE FINANCEIRO

Relatório de Projeto Aplicado
desenvolvido para fins de conclusão do
curso MBA em Arquitetura de Software e
Soluções.

Orientador (a): Ricardo Brito Alves

Caxias do Sul - RS

Julho/2022

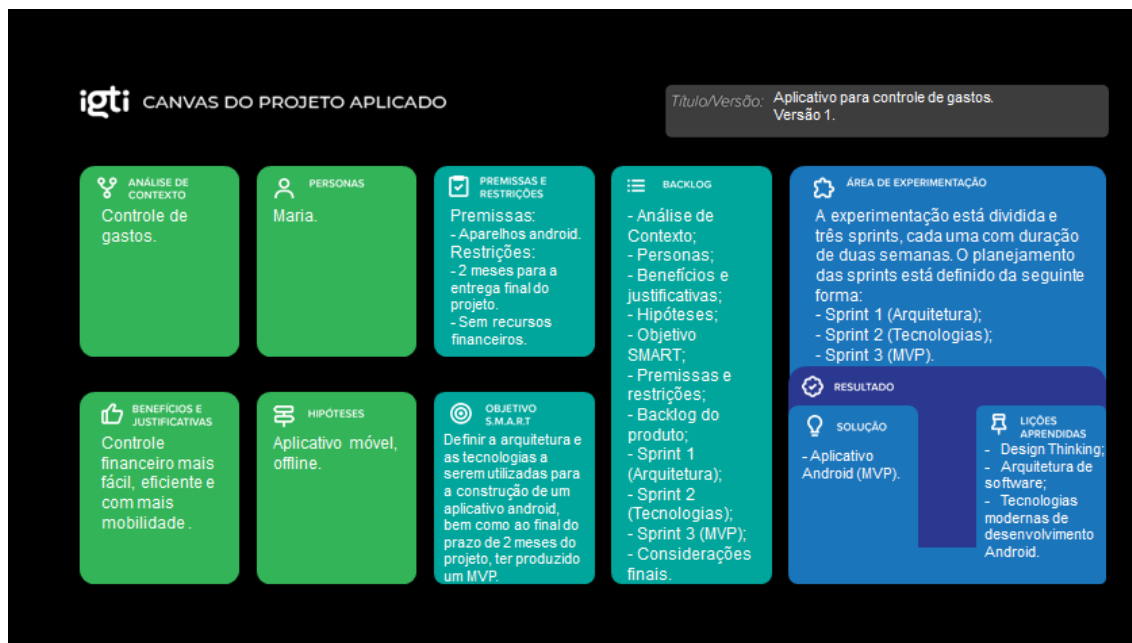
Sumário

1. CANVAS do Projeto Aplicado	4
1.1 Desafio	5
1.1.1 Análise de Contexto	5
1.1.2 Personas	7
1.1.3 Benefícios e Justificativas	9
1.1.4 Hipóteses	11
1.2 Solução	13
1.2.1 Objetivo SMART	13
1.2.2 Premissas e Restrições	14
1.2.3 Backlog de Produto	15
2. Área de Experimentação	16
2.1 Sprint 1	17
2.1.1 Solução	17
• Evidência do planejamento:	17
• Evidência da execução de cada requisito:	18
• Evidência dos resultados:	26
2.1.2 Experiências vivenciadas	27
2.2 Sprint 2	28
2.2.1 Solução	28
• Evidência do planejamento:	28
• Evidência da execução de cada requisito:	29
• Evidência dos resultados:	37
2.2.2 Experiências vivenciadas	39
2.3 Sprint 3	40
2.3.1 Solução	40
• Evidência do planejamento:	40
• Evidência da execução de cada requisito:	41
• Evidência dos resultados:	47
2.3.2 Experiências vivenciadas	48
3. Considerações Finais	49
3.1 Resultados	49
3.2 Contribuições	50
3.3 Próximos passos	51
4. Referências	52

1. CANVAS do Projeto Aplicado

A Figura 1 apresenta uma representação conceitual de todas as etapas do Projeto Aplicado.

Figura 1 - Canvas do projeto.



Fonte: Autoria própria.

1.1 Desafio

1.1.1 Análise de Contexto

Controlar os gastos do mês pode ser uma tarefa difícil para diversas pessoas. Muitas delas, não mantêm um controle sobre seus gastos, e algumas vezes acabam ficando negativadas por gastarem mais do que poderiam. Outras pessoas mantêm um controle de seus gastos anotando, seja em algum caderno, agenda, diário ou até mesmo um bloco de notas, e utilizam o auxílio de uma calculadora para calcular seus gastos e/ou seu saldo do mês.

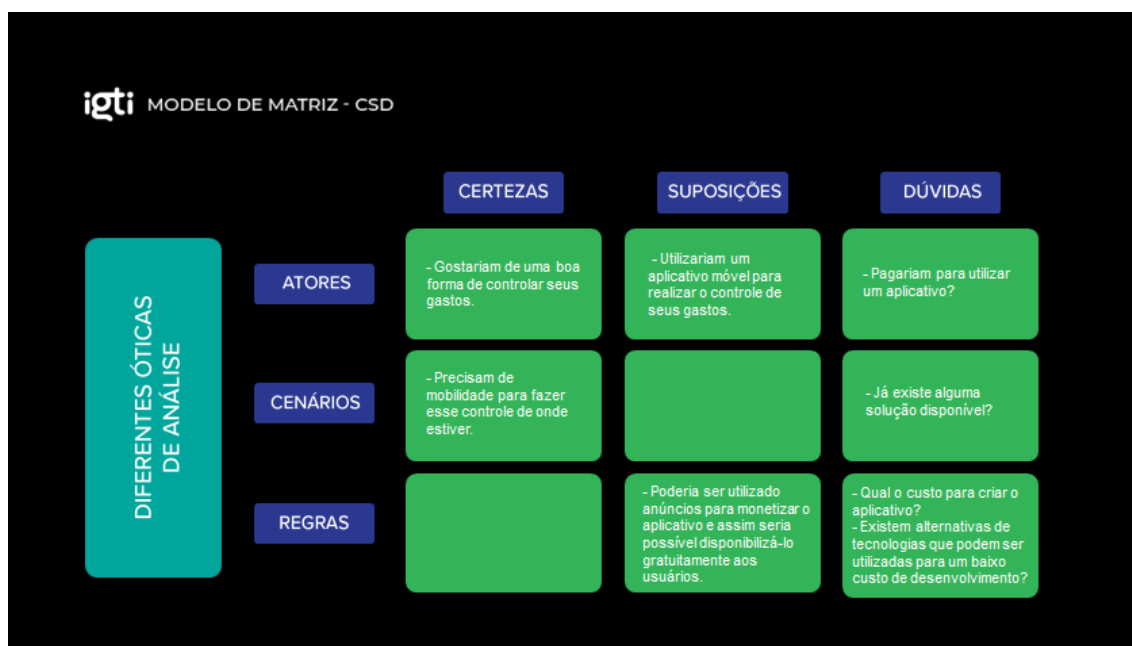
Esses gastos podem ser dos mais variados tipos, como por exemplo, compras no mercado, contas da casa, como água, energia elétrica, telefone, gás, prestações de compras de roupas ou utensílios domésticos, gastos com saúde, medicamentos, gastos com estudo, material escolar, despesas com veículos, enfim, existe uma infinidade de tipos possíveis de gastos que as pessoas têm todos os meses, e que muitas dessas pessoas têm dificuldade para manter um controle de seus gastos.

Um dos problemas sofridos pelas pessoas que anotam seus gastos em papel é que nem sempre que realizam uma compra, ou que pagam por alguma conta ou despesa, elas têm seu papel e caneta à mão para fazer suas anotações, e posteriormente acabam esquecendo de anotar, e com isso acabam por não ter um controle eficiente de seus gastos.

Neste contexto pretende-se com este trabalho desenvolver um MVP de um aplicativo móvel para auxiliar as pessoas a fazer seu controle de gastos. Será realizado um estudo dos principais padrões de arquitetura de *software* utilizados em desenvolvimento *mobile* para definir qual será o padrão arquitetural a ser utilizado no desenvolvimento do aplicativo, bem como serão avaliadas algumas tecnologias disponíveis para o desenvolvimento.

A seguir, na Figura 2, apresenta-se a Matriz CSD (Certezas, Suposições e Dúvidas) montada a partir da contextualização do problema.

Figura 2 - Matiz CSD.



Fonte: Autoria própria.

Com base na contextualização do problema também foi montado um quadro de observações POEMS (acrônimo do inglês People, Objects, Environment, Messages and Services, que significa Pessoas, Objetos, Ambiente, Mensagens e Serviços), apresentado a seguir, na Figura 3.

Figura 3 - Quadro POEMS.



Fonte: Autoria própria.

1.1.2 Personas

A seguir apresenta-se a descrição de uma persona envolvida diretamente no problema apresentado, bem como um mapa de empatia na Figura 5.

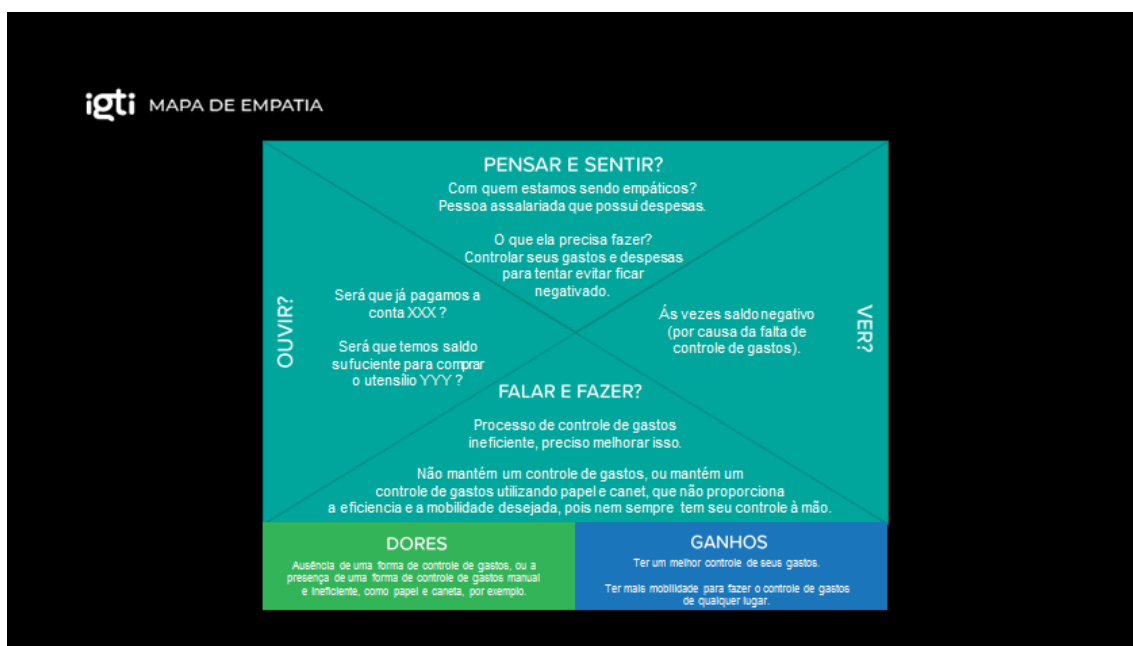
Figura 4 - Maria (Avatar Persona).



Fonte: <https://pt.xavatar.io/> (acessado: 05.06.2022).

Maria, 36 anos, casada, 2 filhos, de 10 e 6 anos, trabalha como atendente de loja. Maria e seu esposo trabalham e moram de aluguel. Maria se encarrega de pagar as despesas da casa, utilizando o salário dela e de seu esposo. Dentre as contas que Maria paga, estão: aluguel, água, energia elétrica, gás, super mercado, roupas e materiais escolares para as crianças. Maria sente um pouco de dificuldade de controlar tantos gastos mensais fixos e mais os gastos eventuais do dia a dia, por isso gostaria de ter à mão uma ferramenta que a ajudasse a controlar esses gastos, saber quais contas já pagou e quais ainda não pagou, além de saber quanto dinheiro gastou e quanto ainda tem.

Figura 5 - Mapa de Empatia.



Fonte: Autoria própria.

1.1.3 Benefícios e Justificativas

Facilitar o controle financeiro das pessoas, deixando-o mais eficiente e com mais mobilidade são os principais benefícios que justificam a motivação para o desenvolvimento deste projeto.

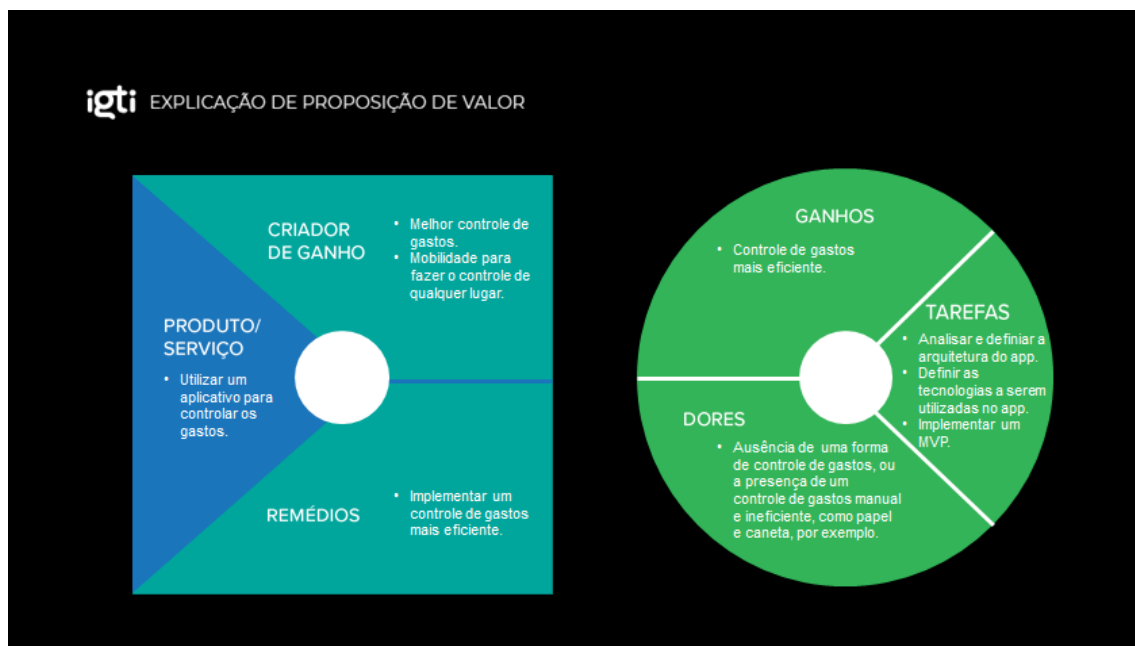
Para auxiliar a análise de benefícios e justificativas foram utilizadas as ferramentas de Blueprint (Figura 6) e de Proposição de Valor (Figura 7), apresentadas a seguir.

Figura 6 - Blueprint.

Itens	Detalhamento
Objetivos	Controlar os gastos.
Atividades	Anotar o que gasta e ir somando.
Questões	Esqueci de anotar algum gasto? Será que já paguei a conta XXX? Será que tenho saldo suficiente para comprar YYY?
Barreiras	Nem sempre se tem papel e caneta à mão para anotar os gastos.
Funcionalidades	Anotar gastos de qualquer lugar e fazer somatório de gastos automático. Possibilitar informar contas com data de vencimento. Notificar quando estiver próximo da data de vencimento.
Interação	Compras e Pagamentos.
Mensagem	Melhore sua gestão financeira.
Onde ocorre	Qualquer lugar.
Tarefas aparentes	Somatório de gastos. Notificações.
Tarefas escondidas	Monitoramento de 'crashes' no aplicativo.
Processos de suporte	Avaliações e comentários na loja de aplicativos.
Saída desejável	Controle de gastos mais eficiente.

Fonte: Autoria própria.

Figura 7 - Proposição de Valor.



Fonte: Autoria própria.

1.1.4 Hipóteses

Com as informações apresentadas até o momento, foi possível realizar algumas observações, levantar algumas hipóteses e coletar algumas ideias. As observações e hipóteses irão auxiliar no direcionamento do desenvolvimento do projeto e estão estruturadas na tabela a seguir (Figura 8).

Figura 8 - Observações e Hipóteses.

Observações	Hipóteses
Algumas pessoas sentem dificuldade em controlar seus gastos.	Poderiam utilizar alguma ferramenta para auxiliar no controle dos gastos.
Precisam utilizar a ferramenta de qualquer lugar.	Poderiam utilizar uma aplicativo móvel.
Nem sempre tem internet disponível.	O aplicativo poderia funcionar offline.

Fonte: Autoria própria.

Para a priorização de ideias foi utilizada a matriz BASICO, acrônimo dos seguintes critérios de avaliação: Benefícios, Abrangência, Satisfação, Investimento, Cliente e Operacionalidade. A matriz de priorização de ideias pode ser observada na Figura 9.

Figura 9 - Priorização de ideias com matriz BASICO.

Priorização de ideias:

Ideias	Critérios de comparação						Somatório	Priorização
	B	A	S	I	C	O		
Utilizar uma arquitetura mobile robusta e escalável para o desenvolvimento do aplicativo.	5	5	4	4	5	5	28	1
Funcionalidade de login, para proteger com senha a entrada no aplicativo.	5	5	5	4	4	4	27	3
Possibilitar inserir e deletar registros de gastos, informando os valores gastos, bem como somar os gastos do mês apresentando o resultado.	5	5	5	4	5	4	28	2
Mostrar apenas os gastos do mês corrente, mas possibilitar a visualização dos gastos dos meses anteriores.	4	4	4	4	4	4	24	6
Implementar tema noturno (dark).	2	3	3	4	3	5	20	8
Possibilitar informar contas com data de vencimento. Notificar quando estiver próximo da data de vencimento.	5	4	4	4	5	4	26	4
Possibilitar o funcionamento offline, e também possibilitar sincronização das informações com uma nuvem, para que o usuário não perca seus dados, ao trocar de aparelho, por exemplo.	4	4	4	4	5	4	25	5
Anúncios no aplicativo, para a monetização do mesmo.	5	5	2	5	2	5	24	7

Fonte: Autoria própria.

Para aplicar as notas da priorização de ideias foram utilizados os balizadores apresentados a seguir, na Figura 10.

Figura 10 - Balizadores para notas matriz BASICO.

Balizadores para notas:

Escala	B - Benefícios	A - Abrangência	S - Satisfação	I - Investimento	C - Cliente	O - Operacionalidade
5	Muito significativo	Muito grande	Muito grande	Muito pouco	Nenhum impacto	Muito fácil
4	Significativo	Grande	Grande	Pouco	Pequeno impacto	Fácil
3	Razoável	Média	Média	Médio	Médio impacto	Média dificuldade
2	Pouco significativo	Pequena	Pequena	Alto	Grande impacto	Difícil
1	Quase não notado	Muito pequena	Muito pequena	Muito alto	Impacto muito grande	Muito difícil

Fonte: Autoria própria.

1.2 Solução

1.2.1 Objetivo SMART

Pretende-se com realização deste projeto, definir o padrão arquitetural e as tecnologias a serem utilizadas para a construção de um aplicativo para dispositivos móveis, mais especificamente para a plataforma Android, bem como ao final do prazo de 2 meses de duração do projeto, ter produzido um MVP (do inglês Minimum Viable Product que significa Produto Mínimo Viável), que contemple (pelo menos) os dois itens com mais prioridade especificados na matriz de priorização de ideias (Figura 9).

1.2.2 Premissas e Restrições

As premissas para a execução do projeto são:

- É possível melhorar o controle de gastos de pessoas com dificuldade em controlar seus gastos.
- Muitas das pessoas com dificuldade em controlar seus gastos possuem um aparelho Android (O MVP pode ser desenvolvido para a plataforma Android).

As restrições para a execução do projeto são:

- Período de 2 meses para a entrega final do projeto.
- Custo; não terão recursos financeiros para o desenvolvimento do projeto, portanto deve-se utilizar ferramentas que não gerem custo.
- Avaliar a LGPD (Lei Geral de Proteção de Dados pessoais) a fim de verificar se existem restrições sobre os dados que o aplicativo deve armazenar.

Riscos do projeto:

A partir das premissas e restrições, levantou-se uma lista de itens que podem representar um risco para o andamento do projeto:

- Risco 001: Não ser possível a conclusão do MVP no período estipulado;
- Risco 002: Necessidade de utilização de alguma ferramenta com custo para o desenvolvimento do projeto;
- Risco 003: Baixa aderência de usuários ao aplicativo.

Para a avaliação dos riscos foi utilizada a matriz de riscos demonstrada na Figura 11.

Figura 11 - Matriz de Riscos.

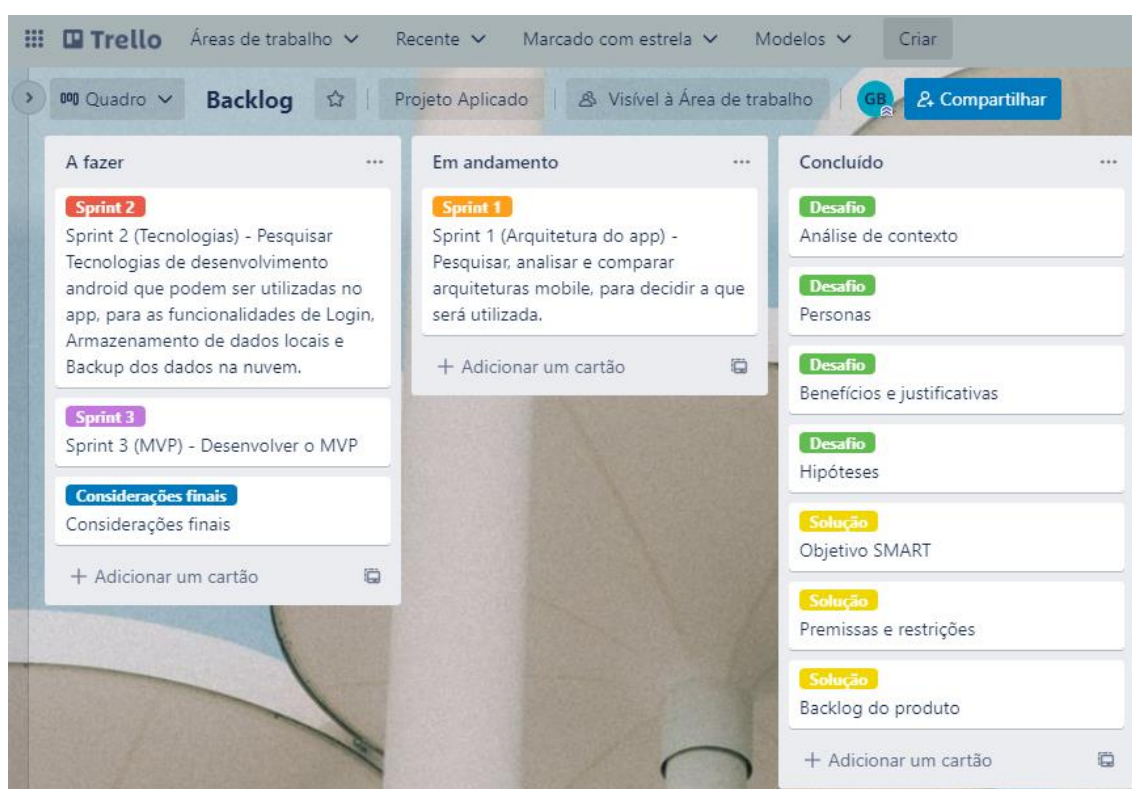
Risco	Probabilidade	Impacto	Ações preventivas	Ações corretivas
Risco 001	Média	Médio	Dedicar tempo para trabalhar no MVP.	Entregar o projeto sem a apresentação de um MVP, somente entregar o relatório.
Risco 002	Muito baixa	Muito alto	Pesquisar outras ferramentas.	Não usar a ferramenta.
Risco 003	Muito alta	Muito baixo	Pesquisar formas de divulgação.	Divulgar.

Fonte: Autoria própria.

1.2.3 Backlog de Produto

Foi escolhido o kanban de atividades para uma visualização do progresso do projeto de maneira clara e objetiva. Esse método é bastante utilizado, pois proporciona transparência das tarefas do projeto. Para isso está sendo utilizada a ferramenta Trello¹. A Figura 12 apresenta o quadro com as atividades do *backlog* do projeto, na data da entrega parcial do relatório contendo os capítulos de Desafio e Solução. No quadro podem ser observadas três colunas: ‘A fazer’, que representa as tarefas a serem realizadas; ‘Em andamento’, que representa as tarefas que estão sendo executadas; e ‘Concluído’ que representa as tarefas que já foram executadas.

Figura 12 - *Backlog* do projeto.



Fonte: Autoria própria.

¹ <https://trello.com/> (acessado: 08.06.2022).

2. Área de Experimentação

A experimentação está dividida em três *sprints*, cada uma com duração de duas semanas. O planejamento das *sprints* está definido da seguinte forma:

Sprint 1 - Realização de um estudo dos principais padrões de arquitetura de *software* utilizados em desenvolvimento *mobile* e definição do padrão arquitetural de *software* a ser utilizado no desenvolvimento do aplicativo.

Sprint 2 - Verificação de algumas tecnologias disponíveis para o desenvolvimento do aplicativo, por exemplo, verificar quais tecnologias poderiam ser utilizadas para as funcionalidades de Login (autenticação), armazenamento de dados e backup dos dados (na nuvem, por exemplo); para definir quais tecnologias serão utilizadas.

Sprint 3 - Desenvolvimento de um MVP que contemple (pelo menos) os dois itens com mais prioridade especificados na matriz de priorização de ideias (Figura 9), que são:

- 1) Utilizar uma arquitetura robusta e escalável (arquitetura definida na *Sprint 1*); e
- 2) Possibilitar inserir e deletar registros de gastos, informando os valores gastos, bem como somar os gastos do mês apresentando o resultado. O aplicativo deve ser desenvolvido para a plataforma Android.

2.1 Sprint 1

2.1.1 Solução

FORD et al. (2021) afirmam que o verdadeiro trabalho de um arquiteto está em sua capacidade de determinar e avaliar objetivamente os ‘*trade-offs*’ de situações, para realizar uma tomada de decisão e resolver a situação da melhor maneira possível.

A *Sprint 1* contém a realização de um estudo dos principais padrões de arquitetura de *software* utilizados em desenvolvimento *mobile*, e a escolha do padrão a ser utilizado no desenvolvimento do aplicativo com base no estudo realizado.

Os padrões arquiteturais de *software* estudados neste trabalho são: MVC, MVP, MVVM e VIPER. Segundo PANDEY (2018) e KAZAP (2020), estes são os principais e mais utilizados padrões de arquitetura *mobile*.

- **Evidência do planejamento:**

A Figura 13 apresenta o *Backlog* da *Sprint 1* como evidência do planejamento.

Figura 13 - *Backlog* da *Sprint 1*.



Fonte: Autoria própria.

- Evidência da execução de cada requisito:

MVC (Model-View-Controller)

O padrão Model-View-Controller (MVC) é um padrão de arquitetura de *software* amplamente utilizado, pode ser usado para diversos tipos de aplicativos, e é usado principalmente em aplicações Web. Este padrão tem uma estrutura pertinente para construir interfaces de usuário e fornece uma separação das diferentes responsabilidades envolvidas (INGENO, 2018).

O padrão MVC consiste nas camadas Model, View e Controller, que possuem responsabilidades distintas (INGENO, 2018). As responsabilidades de cada camada são descritas a seguir.

- **Model**

Camada que gerencia os dados e o estado da aplicação, além de conter as regras de negócio. Entre suas responsabilidades está o processamento de dados e o acesso às fontes de armazenamento de dados, como um banco de dados por exemplo. A camada *Model* é independente das outras camadas (*View* e *Controller*), isso permite que um *Model* seja reutilizado com diferentes interfaces de usuário; e permite também que seja testado de forma independente (INGENO, 2018; KAZAP, 2020).

Os *Models* recebem comandos dos *Controllers* para obter e atualizar dados. Os *Models* também fornecem atualizações de estado da aplicação. Em algumas variações do MVC, o *Model* é passivo e deve receber uma solicitação para enviar uma atualização de estado da aplicação. Em outras variações, pode estar ativo e enviar notificações de alterações de estado para uma *View* (INGENO, 2018).

- **View**

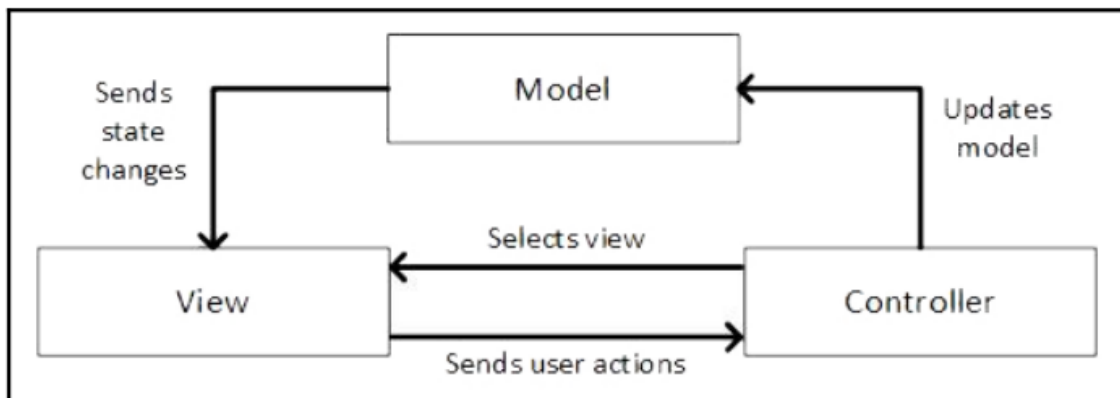
Camada responsável pela apresentação da aplicação e pelas interações com o usuário. Com base nas informações recebidas do *Controller*, a *View* exibe a interface de usuário (UI do inglês User Interface) apropriada, e envia as informações resultantes da interação com o usuário para o *Controller*. Se o *Model* estiver fornecendo atualizações de estado da aplicação diretamente para *Views*, elas também poderão ser atualizadas com base nessas notificações (INGENO, 2018).

- **Controller**

Essa camada atua como intermediária entre as outras camadas (*View* e *Model*). Um *Controller* executa a lógica para selecionar a *View* apropriada e envia as informações necessárias para renderizar a UI. As *Views* notificam os *Controllers* sobre as interações do usuário para que o *Controller* possa responder a elas, e atualizar o *Model* (INGENO, 2018).

A Figura 14 apresenta a estrutura do padrão arquitetural MVC.

Figura 14 - Padrão Arquitetural MVC.



Fonte: INGENO (2018).

Vantagens do padrão MVC:

- Permite uma separação de responsabilidades (INGENO, 2018).
- Mais fácil de fazer manutenção. Ao separar a apresentação dos dados, fica mais fácil alterar um deles sem afetar o outro (INGENO, 2018).
- Mais fácil de testar. Ao separar a apresentação dos dados, fica mais fácil de testar cada parte da aplicação (INGENO, 2018).
- Reutilização de código. Separar a interface de usuário dos dados permite que *Views* sejam reutilizadas, assim como um *Model* pode ser reutilizado com mais de uma *View* (INGENO, 2018).
- A separação entre as camadas de apresentação e da regra de negócios e dados permite, no caso de desenvolvimento web, acelerar o processo de desenvolvimento, pois algumas tarefas podem ocorrer em paralelo, por exemplo, um desenvolvedor pode trabalhar na interface de usuário (*front-end*) enquanto outro trabalha na regra de negócios (*back-end*) (INGENO, 2018).

Desvantagens do padrão MVC:

- Difícil conseguir uma separação (independência) completa das camadas, por exemplo, adicionar um novo campo na aplicação exigirá uma alteração nos dados (*Model*) e na apresentação (*View*) (INGENO, 2018).
- No caso de desenvolvimento web, exige desenvolvedores qualificados em tecnologias distintas, *front-end* e *back-end* (INGENO, 2018).

MVP (Model-View-Presenter)

O padrão Model-View-Presenter (MVP) é uma variação do padrão MVC. Assim como no padrão MVC, existe uma separação entre a lógica da UI e a lógica de negócios. O padrão MVP consiste nas camadas *Model*, *View* e *Presenter*, neste padrão a camada *Presenter* toma o lugar da camada *Controller* do padrão MVC (INGENO, 2018; KAZAP, 2020).

Neste padrão cada UI (*View*) normalmente está acoplada diretamente ao *Presenter* correspondente. Diferentemente do que pode acontecer no padrão MVC, no padrão MVP a *View* e o *Model* não interagem diretamente entre si (INGENO, 2018).

A seguir são descritas as responsabilidades de cada camada do padrão MVP.

- **Model**

Assim como no padrão MVC, a camada *Model* gerencia os dados e contém as regras de negócio. Os *Models* recebem comandos dos *Presenters* para obter e atualizar dados e fornecem atualizações de estado da aplicação aos *Presenters* (INGENO, 2018).

- **View**

A camada *View* é responsável por exibir a UI e apresentar os dados. Cada *View* no padrão MVP implementa uma interface. A *View* se comunica com o *Presenter* sobre as interações do usuário, e dependem do *Presenter* para fornecer os dados a exibir. O *Presenter* é acoplado à *View* por meio da interface implementada (INGENO, 2018).

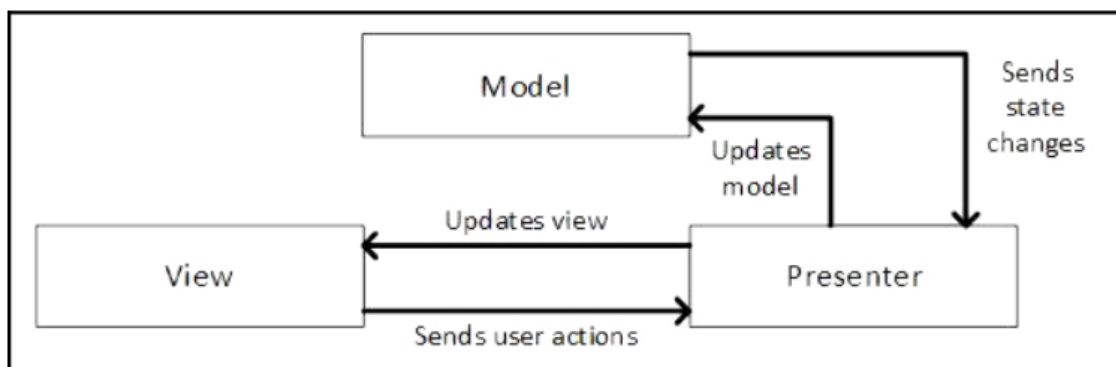
- **Presenter**

A camada *Presenter* é intermediária entre a *Model* e a *View*, e interage com as duas. Cada *View* tem um *Presenter*; a *View* notifica o *Presenter* sobre as interações do usuário; o *Presenter* atualiza o *Model* e recebe as alterações de estado do *Model*, além de receber dados do *Model* e os formatar para a *View*, desempenhando um papel ativo na lógica de apresentação (INGENO, 2018).

Ao contrário do padrão MVC, em que um *Controller* pode interagir com várias *Views*, no padrão MVP, cada *Presenter* normalmente manipula apenas uma *View* (INGENO, 2018).

A Figura 15 apresenta a estrutura do padrão arquitetural MVP.

Figura 15 - Padrão Arquitetural MVP.



Fonte: INGENO (2018).

Pela semelhança com o padrão MVC, o padrão MVP possui as mesmas vantagens e desvantagens do padrão MVC, e possui também as seguintes vantagens:

- É um bom padrão a utilizar quando a quantidade de interações dos usuários é alta e também a lógica associada à camada de apresentação (*View*) é complexa, ou ainda quando é necessário realizar mudanças frequentes na camada *View*; uma vez que as alterações da lógica de apresentação são concentradas em uma única camada, *Presenter*, o que simplifica a manutenção do sistema (KAZAP, 2020).
- Facilita a reutilização das lógicas de apresentação (KAZAP, 2020).

MVVM (Model-View-ViewModel)

O padrão Model-View-ViewModel (MVVM) compartilha semelhanças com MVC e MVP, pois todos eles fornecem uma separação de responsabilidades, o que torna uma aplicação mais fácil de manter, entender e testar. O padrão MVVM consiste nas camadas *Model*, *View* e *ViewModel* (INGENO, 2018).

O padrão MVVM surgiu após o MVC e o MVP, com o objetivo de atualizá-los, melhorando alguns aspectos. Esse padrão tem o objetivo de manter o código da UI simples, facilitando o gerenciamento (KAZAP, 2020).

A seguir são descritas as responsabilidades de cada camada do padrão MVVM.

- **Model**

Assim como nos padrões MVC e MVP, a camada *Model* gerencia os dados e contém as regras de negócio. Em aplicações MVVM os *Models* podem gerar notificações de alteração de propriedade (INGENO, 2018).

- **View**

A camada *View* é responsável por exibir a UI e apresentar os dados. No padrão MVVM, a *View* é ativa. Ao contrário de uma função passiva em que a visualização é completamente manipulada por um *Controller* ou *Presenter*, no MVVM as *Views* manipulam seus próprios eventos, porém elas não mantêm o estado (INGENO, 2018).

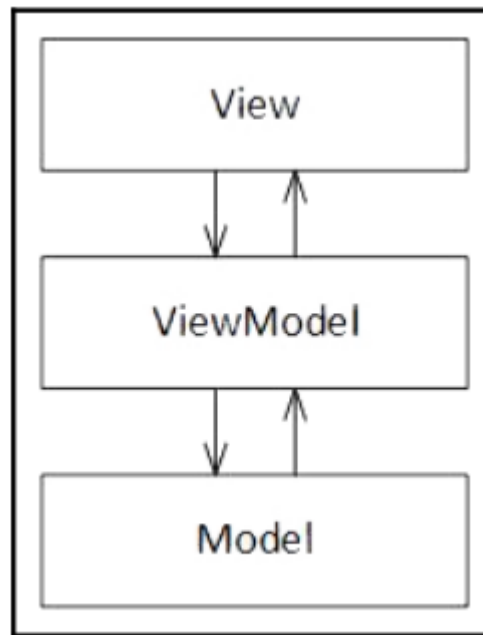
As *Views* comunicam as interações do usuário para o *ViewModel*, o que pode ser feito por meio de um mecanismo como vinculação de dados (*data binding*) ou comandos (*commands*). O padrão MVVM promove uma minimização da quantidade de código nas *Views* (INGENO, 2018).

- **ViewModel**

A camada *ViewModel* no padrão MVVM é semelhante às camadas *Controller* e *Presenter* dos padrões MVC e MVP, pois são intermediários entre *View* e *Model*. Os *ViewModels* fornecem dados às *Views* para exibição e também contêm lógica de interação para se comunicar com *Views* e *Models*. Os *ViewModels* tratam as ações do usuário e a entrada de dados enviados das *Views*, além de conter a lógica de navegação entre *Views* diferentes (INGENO, 2018).

A Figura 16 apresenta a estrutura do padrão arquitetural MVVM.

Figura 16 - Padrão Arquitetural MVVM.



Fonte: INGENO (2018).

Pela semelhança com o padrão MVC, o padrão MVVM possui as mesmas vantagens e desvantagens do padrão MVC, e possui também as seguintes vantagens:

- Minimização da quantidade de código nas *Views* (INGENO, 2018).
- A lógica de apresentação fica no *ViewModel*, o que faz com que os testes possam ser realizados nos *ViewModels* (KAZAP, 2020).

VIPER (View-Interactor-Presenter-Entity-Router)

Diferentemente dos padrões estudados anteriormente (MVC, MVP e MVVM), que possuem 3 camadas, este padrão (VIPER) possui 5 camadas. O VIPER começou sendo utilizado principalmente em aplicações para sistemas IOS, e hoje também é utilizado em aplicações para sistemas Android. Este padrão tem o objetivo de manter o código organizado dividindo-o em módulos. O VIPER é especialmente indicado para projetos que estão em constante evolução, pois é capaz de se adaptar às alterações e de manter o código e a arquitetura do projeto limpa e organizada (KAZAP, 2020).

Este padrão arquitetural é baseado no Princípio de Responsabilidade Única (Single Responsibility Principle) e no paradigma de Arquitetura Limpa (Clean Architecture) (PANDEY, 2018; BOROVSKOY, 2019; KATZ, 2020).

A seguir são descritas as responsabilidades de cada camada do padrão VIPER.

- **View**

É a camada de UI. A responsabilidade da *View* é notificar o *Presenter* sobre as interações do usuário, fazer a troca de dados com o *Presenter* e mostrar na UI o que o *Presenter* solicitar. Conceitualmente uma *View* deve interagir com um único *Presenter* (PANDEY, 2018; CABRAL, 2019).

- **Interactor**

É a camada que contém as regras de negócios. Tem a responsabilidade de fazer o acesso às fontes de dados, como banco de dados ou API externa. Essa camada possui conhecimento da camada *Entity* e faz uso da mesma. (PANDEY, 2018; CABRAL, 2019).

- **Presenter**

Esta camada obtém os dados do *Interactor* envia para a *View* apresentá-los. O *Presenter* também se comunica com o *Router* para navegação de telas (UIs). Essa camada tem conhecimento das camadas *View*, *Router* e *Interactor*. (PANDEY, 2018; CABRAL, 2019).

- **Entity**

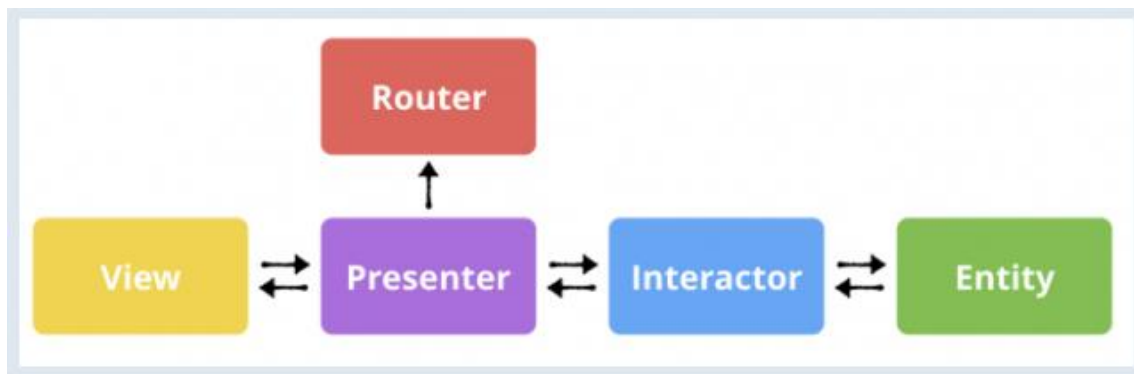
Camada que contém objetos de modelo, ou seja, a estrutura dos dados. Os objetos dessa camada são usados pela camada *Interactor* (PANDEY, 2018; CABRAL, 2019).

- **Router**

Camada que possui a lógica de navegação, ou seja, define quais telas devem ser mostradas e quando devem ser mostradas. Além de controlar as trocas de telas, controla também a troca de dados entre as telas (PANDEY, 2018; CABRAL, 2019).

A Figura 17 apresenta a estrutura do padrão arquitetural VIPER.

Figura 17 - Padrão Arquitetural VIPER.



Fonte: KATZ (2020).

Vantagens do padrão VIPER:

- Possibilidade de reutilização de módulos (CABRAL, 2019).
- Menor impacto em mudanças no código (CABRAL, 2019).
- Maior divisão de responsabilidades entre as camadas (CABRAL, 2019).
- Mais fácil de testar (CABRAL, 2019).

Desvantagens do padrão VIPER:

- Mais escrita de código, uma vez que tem mais camadas (CABRAL, 2019).
- Curva de aprendizado maior (CABRAL, 2019).

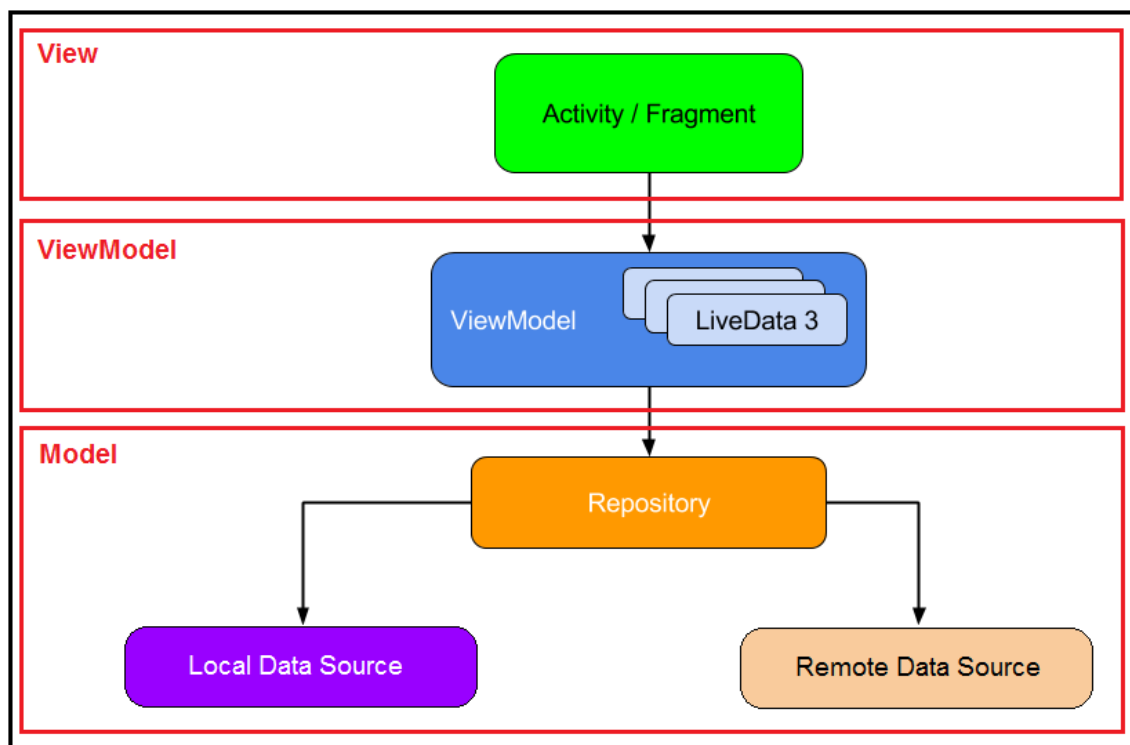
- Evidência dos resultados:

Com base no estudo realizado definiu-se que o padrão arquitetural a ser utilizado no desenvolvimento do aplicativo é o padrão arquitetural MVVM.

A escolha do padrão MVVM foi dada pelo fato deste padrão ter surgido como uma evolução dos padrões MVC e MVP; ele separa a interface de usuário do restante do aplicativo, particionando as responsabilidades, tornando o código mais fácil de manter, entender e testar. Além disso este padrão se encaixa bem neste projeto, por ser um projeto de pequeno escopo e não estará em constante evolução, ao contrário do padrão VIPER, que é mais indicado para projetos maiores, com constante evolução (INGENO, 2018; KAZAP, 2020).

A Figura 18 apresenta a estrutura do padrão arquitetural MVVM a ser utilizado no projeto. Esta estrutura será atualizada na próxima seção (*Sprint 2*) com algumas das tecnologias que serão definidas para a utilização no projeto.

Figura 18 - Padrão Arquitetural do Projeto.



Fonte: Adaptado de AHMED (2017).

2.1.2 Experiências vivenciadas

A escolha da arquitetura do *software* pode influenciar em diversas características da aplicação, como por exemplo, manutenção, testabilidade, escalabilidade, performance, disponibilidade, entre outras (BASS et al., 2012; INGENO, 2018; KAZAP, 2020).

Durante o decorrer da *Sprint* 1 foi possível compreender melhor as características de alguns dos padrões arquiteturais de *software* mais utilizados, com foco nos padrões utilizados para desenvolvimento *mobile*. Os padrões estudados possuem semelhanças, mas cada um tem suas particularidades, vantagens e desvantagens. Conhecer bem os padrões arquiteturais de *software* é importante e auxilia na escolha do padrão ideal para cada tipo de projeto.

2.2 Sprint 2

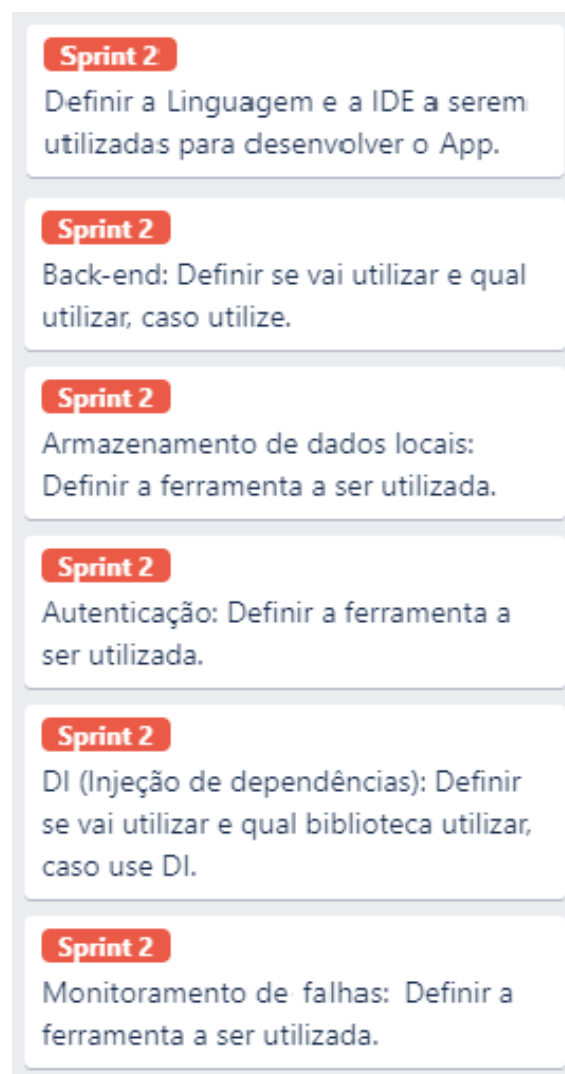
2.2.1 Solução

A *Sprint 2* contém a definição de algumas tecnologias a serem utilizadas no desenvolvimento do aplicativo.

- Evidência do planejamento:

A Figura 19 apresenta o *Backlog* da *Sprint 2* como evidência do planejamento.

Figura 19 - *Backlog* da *Sprint 2*.



Fonte: Autoria própria.

- Evidência da execução de cada requisito:

Linguagem e IDE

O Ambiente de Desenvolvimento Integrado ou IDE, do inglês Integrated Development Environment, oficial para o desenvolvimento de aplicações Android é o Android Studio². O Android Studio é baseado no IntelliJ IDEA³; além do editor de código e das ferramentas de desenvolvedor avançadas do IntelliJ, o Android Studio oferece mais recursos para aumentar a produtividade na criação de aplicações Android, como um emulador por exemplo, dentre outros recursos.

Existem 2 linguagens de programação que podem ser utilizadas para desenvolvimento Android, Java e Kotlin. Inicialmente Java era a única linguagem disponível para desenvolvimento Android, mas em 2019 o Google anunciou que o desenvolvimento do Android será cada vez mais Kotlin, e que as atualizações no Android serão focadas primeiro em Kotlin. Isso significa que as novas ferramentas e conteúdo de desenvolvimento do Android serão projetadas primeiro para Kotlin e continua-se fornecendo suporte para o uso das APIs em linguagem Java. O Google recomenda o uso da linguagem Kotlin. A Figura 20 apresenta um “recorte” da página oficial do Android, onde é recomendada a utilização da linguagem Kotlin.

Figura 20 - Recomendação de uso Kotlin, página oficial Android.

No Google I/O 2019, anunciamos que o desenvolvimento do Android será cada vez mais Kotlin, e mantivemos esse compromisso. O Kotlin é uma linguagem de programação expressiva e concisa que reduz erros comuns de código e se integra facilmente a aplicativos existentes. Se você quer criar um app Android, recomendamos começar com o Kotlin para aproveitar os melhores recursos do setor.

Fonte: <https://developer.android.com/kotlin/first> (acessado: 28.06.2022).

Dessa forma, para o desenvolvimento do aplicativo, será utilizada a IDE Android Studio, e será utilizada a linguagem de programação Kotlin.

² <https://developer.android.com/studio/intro> (acessado: 28.06.2022).

³ <https://www.jetbrains.com/idea/> (acessado: 28.06.2022).

Back-end

As funcionalidades do aplicativo a ser desenvolvido neste projeto permitem que o aplicativo funcione por si só e ‘offline’, ou seja, sem a necessidade de utilização de um *back-end* ou de estar conectado com a internet. Porém, dessa forma, caso o usuário do aplicativo necessite trocar o seu dispositivo, ele acabaria perdendo seus dados ao instalar novamente o aplicativo no novo dispositivo. Por isso, é interessante a utilização de algum *back-end* ou nuvem para guardar um *backup* dos dados. Uma solução muito interessante para o aplicativo deste projeto é a utilização de um *Back-end* como um serviço do inglês Back-end As A Service (BAAS).

BAAS é um serviço baseado em nuvem onde os provedores fornecem as funcionalidades de *back-end* para as aplicações, tanto para aplicativos *Web* quanto para aplicativos móveis. Os provedores fornecem SDKs (Kits de Desenvolvimento de Software do inglês Software Development Kits) e/ou APIs (Interface de Programação de Aplicação do inglês Application Programming Interface) para os desenvolvedores utilizarem os serviços fornecidos. BAAS é uma excelente opção para projetos de pequeno e médio porte, pois dessa forma elimina-se a necessidade de desenvolver um *back-end*, isso economiza tempo e aumenta a produtividade, além de economizar o custo geral de desenvolvimento (CLARK, 2021).

Atualmente existem diversos provedores de BAAS. Para este projeto, avaliou-se 3 possíveis provedores, são eles: Back4app⁴, AWS Amplify⁵ e Firebase⁶. A Figura 21 apresenta uma matriz de decisão utilizada para auxiliar na escolha do provedor.

Figura 21 - Matriz de Decisão BAAS.

Recursos	Peso	Provedores		
		Back4app	AWS Amplify	Firebase
Plano grátis	2	Sim = 2	Sim = 2	Sim = 2
Armazenamento dos dados (<i>data storage</i>)	2	Sim = 2	Sim = 2	Sim = 2
Armazenamento de arquivos (<i>file storage</i>)	1	Sim = 1	Sim = 1	Sim = 1
Notificações (<i>notifications</i>)	1	Sim = 1	Sim = 1	Sim = 1
Autenticação (<i>authentication</i>)	2	Sim = 2	Sim = 2	Sim = 2
Análise (<i>analytics</i>)	1	Não = 0	Sim = 1	Sim = 1
Monitoramento de falhas (<i>crash reporting</i>)	1	Não = 0	Não = 0	Sim = 1
Total		8	9	10

Fonte: Autoria própria.

⁴ <https://www.back4app.com/> (acessado: 02.07.2022).

⁵ <https://aws.amazon.com/pt/amplify/> (acessado: 02.07.2022).

⁶ <https://firebase.google.com/> (acessado: 02.07.2022).

Todos os 3 provedores avaliados possuem planos grátis, e possuem recursos que são úteis para o aplicativo, ou que podem ser úteis futuramente para uma possível evolução do aplicativo. Além do armazenamento dos dados (*data storage*), os provedores tem disponíveis os recursos de: armazenamento de arquivos (*file storage*), notificações (*notifications*) e autenticação (*authentication*). Porém o Firebase tem uma ferramenta de monitoramento de falhas (*crash reporting*) que os outros provedores avaliados não têm, o Crashlytics. O Firebase e o AWS Amplify possuem ferramenta de análise (*analytics*), porém o Back4app não possui. Com base na análise realizada, o provedor de BAAS escolhido para utilizar no aplicativo deste projeto é o Firebase.

Dentre as opções de armazenamento de dados disponíveis no Firebase estão ‘Realtime Database’ e ‘Cloud Firestore’. A página oficial do Firebase apresenta um comparativo entre essas 2 opções, com as principais considerações para que possa ser feita a escolha da opção mais adequada para cada projeto. Após analisar as informações fornecidas na página (<https://firebase.google.com/docs/database/rtdb-vs-firestore>), decidiu-se que a opção que melhor se encaixa no projeto é a utilização do ‘Realtime Database’.

Armazenamento de dados de forma local

O sistema Android oferece opções para salvar arquivos e dados do aplicativo (app). Dentre as opções disponíveis estão:

Armazenamento específico do app: Opção para armazenamento de arquivos. Nesta opção os arquivos são usados apenas pelo app, seja em diretórios dedicados no armazenamento interno ou em outros diretórios dedicados no armazenamento externo.

Armazenamento compartilhado: Opção para armazenamento de arquivos. Nesta opção os arquivos podem ser compartilhados com outros aplicativos.

Preferências: Opção para armazenamento de dados. Nesta opção os dados são usados apenas pelo app e só é possível armazenar dados primitivos em pares de chave-valor.

Bancos de dados: Opção para armazenamento de dados. Nesta opção os dados são usados apenas pelo app e são armazenados de forma estruturada em um banco de dados.

Uma vez que o armazenamento de dados estruturados é mais conveniente no desenvolvimento do app deste projeto, optou-se pela utilização de banco de dados para o armazenamento dos dados de forma local. O Android fornece o banco de dados SQLite e disponibiliza as APIs necessárias para usá-lo. Além disso, é fornecida a biblioteca Room, que é uma biblioteca de persistência que oferece uma camada de abstração sobre o SQLite; por isso é recomendado o uso da biblioteca Room em vez de usar diretamente as APIs SQLite. As Figuras 22 e 23 apresentam “recortes” da página oficial do Android, onde é recomendada a utilização da biblioteca Room.

Figura 22 - Recomendação 1 de uso Room, página oficial Android.

- **Bancos de dados:** armazene dados estruturados em um banco de dados particular usando a biblioteca de persistência do Room.

Fonte: <https://developer.android.com/training/data-storage> (acessado: 28.06.2022).

Figura 23 - Recomendação 2 de uso Room, página oficial Android.

A biblioteca de persistência Room oferece uma camada de abstração sobre o SQLite para permitir um acesso fluente ao banco de dados, aproveitando toda a capacidade do SQLite. O Room oferece principalmente estes benefícios:

- Verificação de consultas SQL durante a compilação.
- Anotações de conveniência que minimizam o código boilerplate repetitivo e propenso a erros.
- Caminhos de migração de banco de dados simplificados.

Dessa forma, recomendamos que você use o Room em vez de [usar diretamente as APIs SQLite](#).

Fonte: <https://developer.android.com/training/data-storage/room>
(acessado: 28.06.2022).

Dessa forma, para o armazenamento de dados de forma local no aplicativo será utilizado banco de dados SQLite, e para o acesso ao banco será utilizada a biblioteca Room.

Autenticação

Uma vez que foi decidido utilizar o Firebase como BAAS do projeto, a escolha do serviço de autenticação do aplicativo será o Firebase Authentication. São fornecidos diversos serviços de autenticação de forma gratuita, como por exemplo: Autenticação com e-mail e senha, Login do Google, Login do facebook, entre outros. O aplicativo pode fornecer mais de uma opção de autenticação aos usuários, porém para um MVP decidiu-se implementar apenas uma forma de autenticação, a autenticação com e-mail e senha.

Ao utilizar o serviço de autenticação do Firebase também tem-se o benefício de manter o gerenciamento dos serviços na mesma plataforma de BAAS, isso facilita o gerenciamento.

Injeção de Dependências (DI)

Injeção de Dependências do inglês Dependency Injection (DI) é uma das maneiras de implementar Inversão de Controle do inglês Inversion of Control (IoC). Inversão de Controle é um princípio de *design* de *software*, onde a responsabilidade de informar a implementação (dependência) a ser utilizada deixa de ser da classe, e passa a ser do consumidor da classe. Isso ajuda a seguir o primeiro e o último dos cinco princípios SOLID, princípio de responsabilidade única (Single-responsibility principle) e princípio de inversão de dependência (Dependency Inversion Principle) (INGENO, 2018; LANFREDI, 2017).

DI é uma técnica que fornece dependências para uma classe, alcançando assim a inversão de dependência. As dependências são passadas (injetadas) para uma classe que precisa delas. Utilizar DI contribui para a redução da complexidade do código, e com isso um aumento na facilidade de manutenção e implementação de novas funcionalidades, além de aumentar a testabilidade, uma vez que o código fica fracamente acoplado e pode ser testado mais facilmente. (INGENO, 2018; LANFREDI, 2017).

Existem diversas bibliotecas disponíveis para a realização de injeção de dependências. Para Android destacam-se Dagger⁷ e Hilt⁸ disponíveis para uso nas linguagens Java e Kotlin, e Koin⁹ disponível para uso na linguagem Kotlin. O Koin se destacou por sua simplicidade e facilidade. O Dagger foi criado em 2010, pela Square¹⁰, e em 2015, a Square passou o Dagger para a Google, que implementou novidades e melhorias. Recentemente, a Google construiu o Hilt, com base no Dagger (RABELLO, 2020).

Para o desenvolvimento do aplicativo deste projeto optou-se pela utilização da biblioteca Hilt, pois o uso da mesma é recomendado pela Google. A Figura 24 apresenta um “recorte” da página oficial do Android, onde é recomendada a utilização da biblioteca Hilt.

Figura 24 - Recomendação de uso Hilt, página oficial Android.

Recomendamos seguir os padrões de injeção de dependência e usar a biblioteca Hilt em apps Android. A biblioteca Hilt constrói os objetos automaticamente percorrendo a árvore de dependências, além de oferecer garantias de tempo de compilação nas dependências e criar contêineres de dependência para classes do framework do Android.

Fonte: <https://developer.android.com/topic/architecture> (acessado: 28.06.2022).

⁷ <https://dagger.dev/> (acessado: 28.06.2022).

⁸ <https://dagger.dev/hilt/> (acessado: 28.06.2022).

⁹ <https://insert-koin.io/> (acessado: 28.06.2022).

¹⁰ <https://square.github.io/> (acessado: 28.06.2022).

Monitoramento de falhas

Para o monitoramento de falhas ou *'bugs'* no aplicativo existem algumas opções disponíveis, como por exemplo o Splunk MINT Express¹¹, o Sentry¹² e o Firebase Crashlytics¹³. O Splunk MINT Express não tem opção sem custo. O Sentry tem opção de uso sem custo para desenvolvedor. Já o Firebase Crashlytics é completamente gratuito.

O serviço de monitoramento de falhas escolhido para usar no aplicativo é o Firebase Crashlytics, uma vez que é gratuito, e além disso, mantém-se o gerenciamento deste serviço na mesma plataforma de BAAS escolhida, o Firebase, o que facilita o gerenciamento.

¹¹ https://www.splunk.com/en_us/blog/tips-and-tricks/bugsense-is-now-splunk-mint-express-why-the-change.html (acessado: 02.07.2022).

¹² <https://sentry.io/welcome/> (acessado: 02.07.2022).

¹³ <https://firebase.google.com/products/crashlytics> (acessado: 02.07.2022).

- Evidência dos resultados:

Como resultado das pesquisas e definições realizadas na *Sprint 2* tem-se:

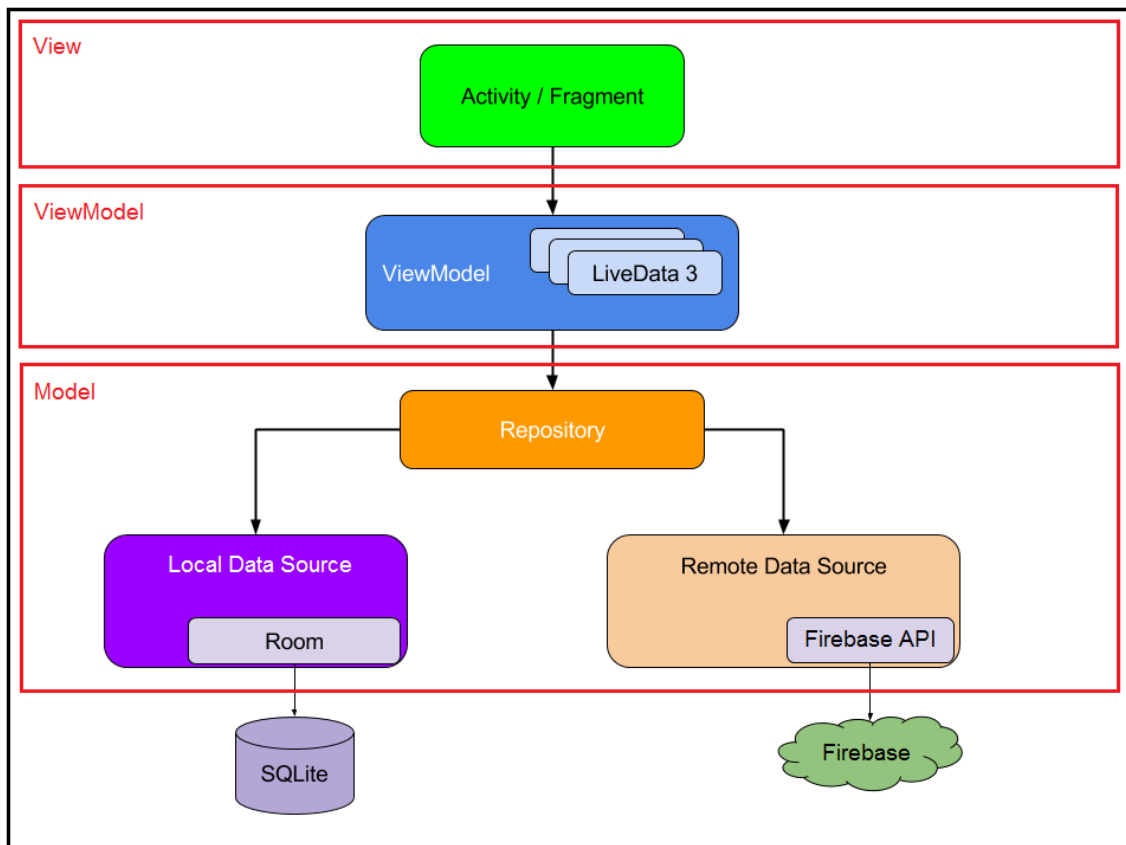
- A IDE a ser utilizada será Android Studio;
- A linguagem de programação a ser utilizada será Kotlin;
- Não será desenvolvido um *back-end* para o aplicativo, mas será utilizada uma tecnologia BAAS (*Back-end as a Service*) para ter um *backup* dos dados em nuvem e um sistema de autenticação. O provedor BAAS utilizado será o Firebase, que oferece diversos serviços. Será utilizado o serviço ‘Realtime Database’ para o *backup* dos dados em nuvem, e o serviço ‘Authentication’ para a autenticação dos usuários no aplicativo; também será utilizado o serviço ‘Crashlytics’ para o monitoramento de falhas no aplicativo.

Manter os diversos serviços utilizados no aplicativo (*backup* dos dados em nuvem, autenticação e monitoramento de falhas) na mesma plataforma, o Firebase, facilita o gerenciamento.

- Para o armazenamento de dados local será utilizado banco de dados SQLite, e para o acesso ao banco será utilizado o Room, que é uma biblioteca de persistência que oferece uma camada de abstração sobre o SQLite.
- Para a Injeção de Dependências será utilizada a biblioteca Hilt, que é recomendada na página oficial do Android.

A Figura 25 apresenta a estrutura do padrão arquitetural MVVM a ser utilizado no projeto, atualizada com as tecnologias de persistência de dados definidas para serem utilizadas.

Figura 25 - Padrão Arquitetural do Projeto atualizado.



Fonte: Adaptado de AHMED (2017).

2.2.2 Experiências vivenciadas

Durante o decorrer da *Sprint 2* foi possível compreender melhor a importância de pesquisar os serviços e ferramentas disponíveis além de entender suas características, para assim fazer uma escolha mais acertiva dos serviços e das ferramentas que melhor se adequam aos projetos.

Assim como a arquitetura, os serviços e as ferramentas utilizados nos projetos podem influenciar em características como manutenção, escalabilidade, performance, disponibilidade, custo, entre outras.

2.3 Sprint 3

2.3.1 Solução

A *Sprint 3* contém o desenvolvimento de um MVP do aplicativo para a plataforma Android, que contempla alguns dos itens especificados na matriz de priorização de ideias (Figura 9). Este aplicativo utiliza a arquitetura e as tecnologias definidas nas *Sprints 1* e *2*.

- Evidência do planejamento:

A Figura 26 apresenta o *Backlog* da *Sprint 3* como evidência do planejamento.

Figura 26 - *Backlog* da *Sprint 3*.




Fonte: Autoria própria.

- Evidência da execução de cada requisito:

Telas relacionadas a login

As telas de Login, Criação de conta e de Recuperação de senha são telas relacionadas a autenticação do usuário no aplicativo. A Figura 27 apresenta as telas de Login, Criação de conta e de Recuperação de senha como evidência da execução.

Figura 27 - Telas relacionadas a login.

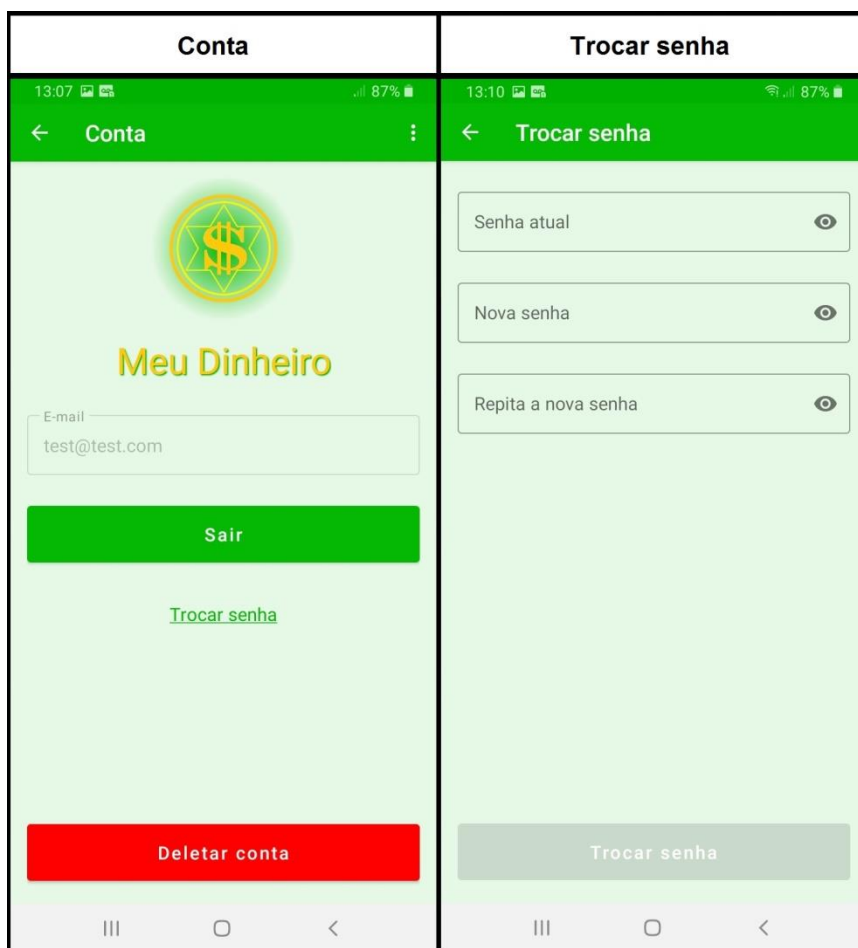
Login	Criar conta	Esqueceu a senha
 <p>Meu Dinheiro</p> <p>E-mail</p> <p>Senha</p> <p>Login</p> <p>Esqueceu a senha?</p> <p>Não tem uma conta?</p> <p>Criar conta</p>	<p>← Criar conta</p> <p>E-mail</p> <p>Repita o e-mail</p> <p>Senha</p> <p>Repita a senha</p> <p><input type="checkbox"/> Eu aceito os termos e condições</p> <p>Criar conta</p>	<p>← Recuperar senha</p> <p>Informe seu e-mail para recuperar sua senha</p> <p>E-mail</p> <p>Recuperar senha</p>

Fonte: Autoria própria.

Telas relacionadas a conta do usuário

As telas de Conta e de Troca de senha são telas relacionadas ao usuário autenticado no aplicativo. A Figura 28 apresenta as telas de Conta e de Troca de senha como evidência da execução.

Figura 28 - Telas relacionadas a conta.

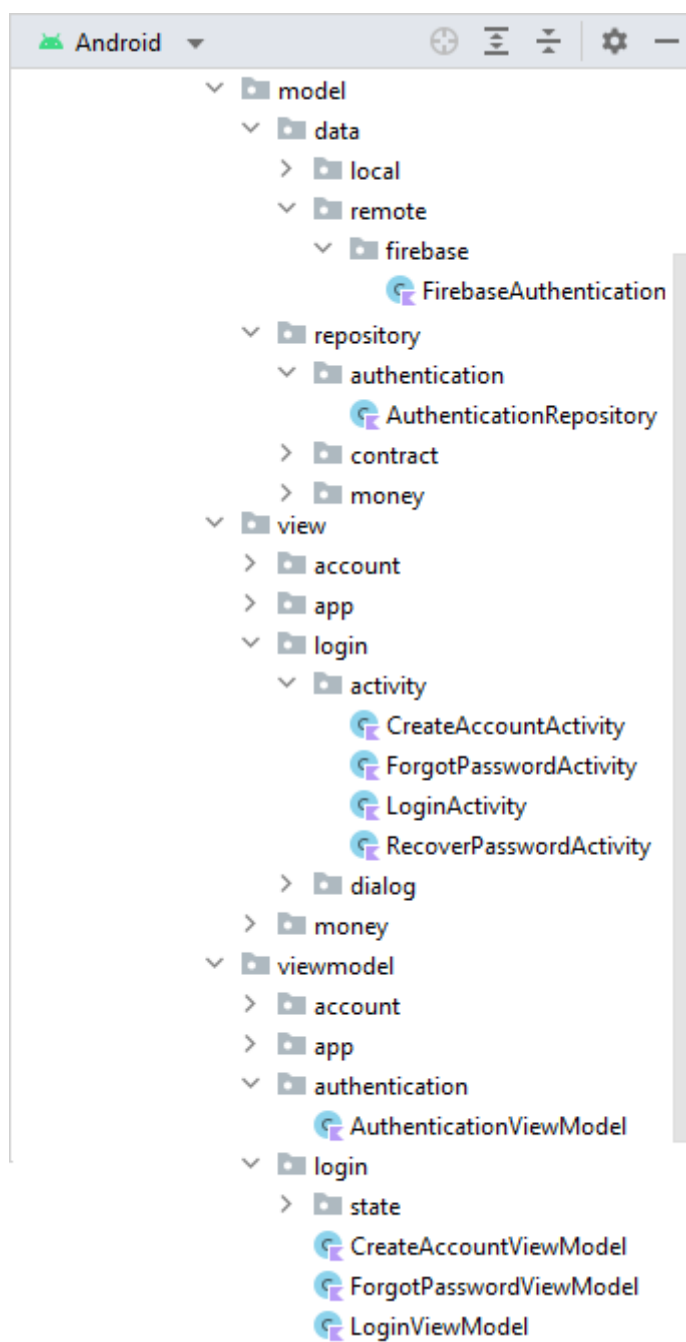


Fonte: Autoria própria.

Autenticação

O código que trata a autenticação do usuário no aplicativo, bem como a integração com o Firebase, foi desenvolvido com base na arquitetura MVVM definida para a construção do aplicativo. A Figura 29 apresenta as classes utilizadas na autenticação e na integração com o Firebase como evidência da execução.

Figura 29 - Classes utilizadas na autenticação.



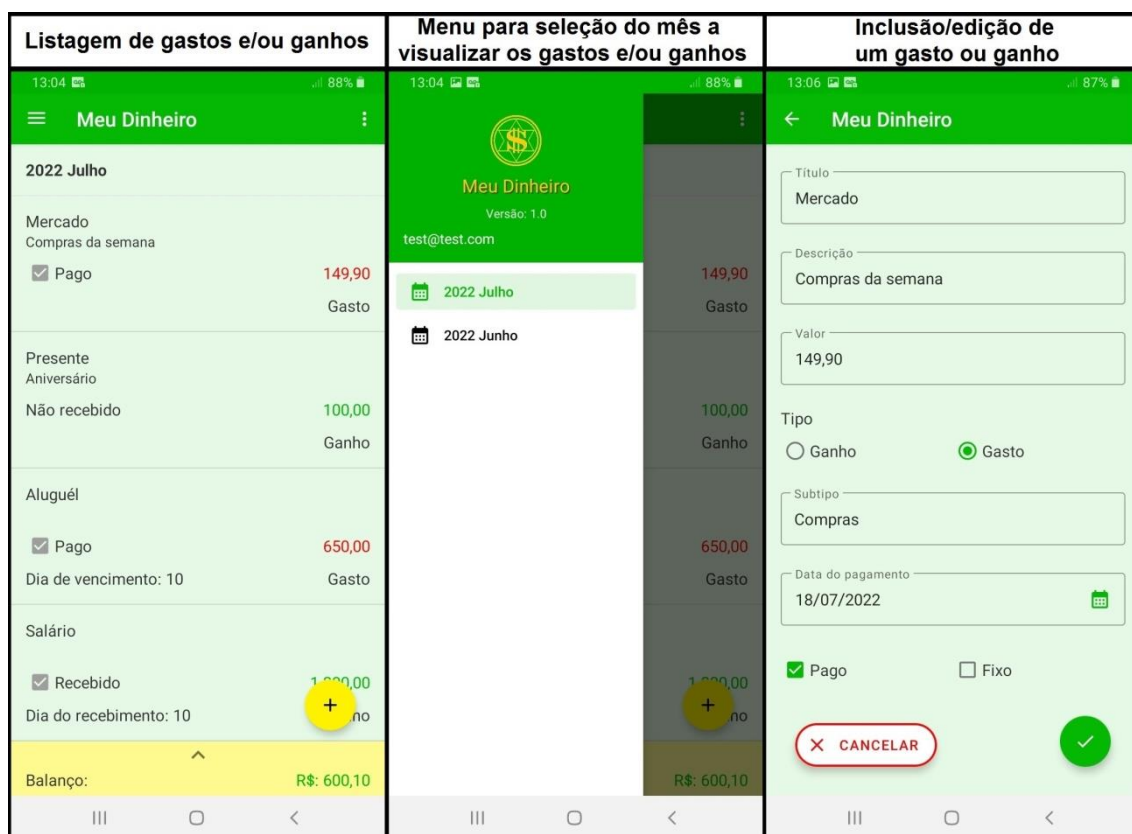
Fonte: Autoria própria.

Telas relacionadas ao controle financeiro

As telas relacionadas ao controle financeiro são as telas em que o usuário pode informar seus gastos e/ou ganhos, para poder visualizá-los, bem como visualizar o balanço entre seus ganhos e gastos, para dessa forma fazer seu gerenciamento financeiro.

A Figura 30 apresenta, como evidência da execução, a tela de Listagem de gastos e/ou ganhos e a tela de Inclusão/edição de um gasto ou ganho, bem como o menu presente na tela de Listagem de gastos e/ou ganhos que contém os meses em que é possível selecionar para visualizar os gastos e/ou ganhos do mês selecionado.

Figura 30 - Telas relacionadas ao controle de gastos.



Fonte: Autoria própria.

Armazenamento de dados de forma local

O armazenamento de dados de forma local é realizado através da utilização da biblioteca Room. Apenas uma tabela foi definida para armazenar as informações de cada gasto ou ganho no banco de dados, a tabela MONEY. A Figura 31 apresenta o diagrama da entidade MONEY que representa a tabela do banco de dados.

Figura 31 - Diagrama da entidade.

MONEY				
ID	INTEGER	NOT NULL	PK	UNIQUE
USER_EMAIL	TEXT	NOT NULL		UNIQUE
TITLE	TEXT	NOT NULL		
DESCRIPTION	TEXT			
VALUE	REAL	NOT NULL		
TYPE	TEXT	NOT NULL		
SUBTYPE	TEXT			
PAY_DATE	TEXT			
PAID	INTEGER	NOT NULL		
FIXED	INTEGER	NOT NULL		
DUE_DAY	INTEGER			
CREATION_DATE	TEXT	NOT NULL		

Fonte: Autoria própria.

A descrição das colunas da tabela MONEY é apresentada a seguir:

ID: Identificador, um número inteiro de auto incremento, essa coluna é a Primary Key (PK) da tabela.

USER_EMAIL: E-mail do usuário, as colunas ID e USER_EMAIL formam uma Unique Key (UK).

TITLE: Título do gasto ou ganho.

DESCRIPTION: Descrição do gasto ou ganho (opcional).

VALUE: Valor do gasto ou ganho.

TYPE: Tipo (Ganho ou Gasto).

SUBTYPE: Subtipo para categorizar um gasto ou ganho (opcional).

PAY_DATE: Data de pagamento, data que foi pago o gasto ou recebido o ganho.

PAID: Pago, número inteiro, 0 ou 1 (zero ou um), indica se o gasto foi pago ou se o ganho foi recebido.

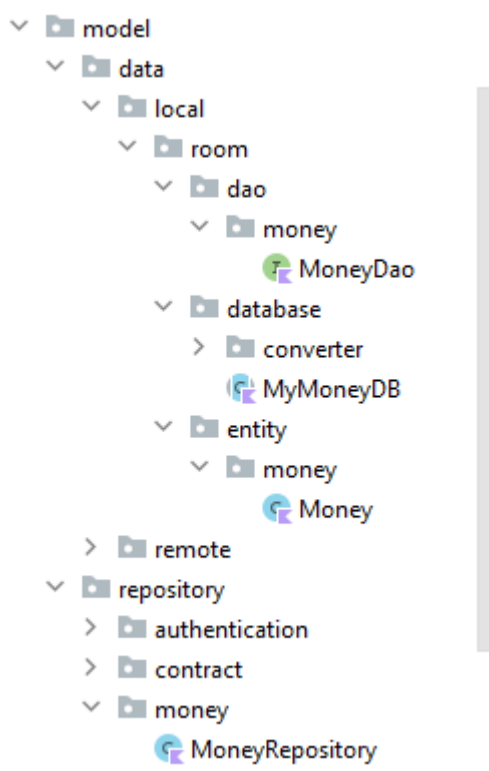
FIXED: Fixo, número inteiro, 0 ou 1 (zero ou um), indica se é um gasto ou ganho fixo, ou seja, que é recorrente, tem todos os meses.

DUE_DAY: Dia de vencimento do ganho ou do gasto fixo.

CREATION_DATE: Data de criação, data em que o gasto ou o ganho foi criado no aplicativo.

Todo o aplicativo foi desenvolvido com base na arquitetura MVVM definida para sua construção, portanto o código que trata o armazenamento de dados também está no padrão MVVM. A Figura 32 apresenta as classes utilizadas no acesso aos dados locais e na integração com o Room como evidência da execução.

Figura 32 - Classes utilizadas no acesso a dados locais.



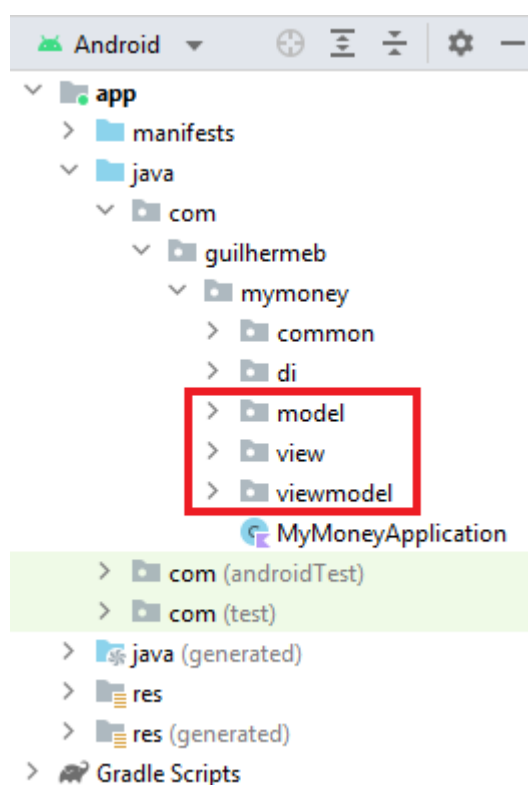
Fonte: Autoria própria.

- Evidência dos resultados:

Como resultado da *Sprint 3* tem-se um aplicativo Android MVP, chamado de “Meu Dinheiro”, que possui uma arquitetura moderna e bem definida e também um código limpo e bem estruturado, que utiliza algumas das tecnologias mais modernas disponíveis para desenvolvimento Android. O código fonte do aplicativo desenvolvido está disponível em: <https://github.com/guilherme-boschetti/MyMoney>.

A Figura 33 apresenta a divisão das camadas da arquitetura MVVM no código fonte desenvolvido para o aplicativo.

Figura 33 - Divisão das camadas da arquitetura MVVM no aplicativo.



Fonte: Autoria própria.

2.3.2 Experiências vivenciadas

Durante o decorrer da Sprint 3 foi possível colocar em prática a teoria, compreender melhor a importância de planejar um projeto e também a importância de utilizar uma arquitetura moderna e bem definida e também a importância de escrever código limpo, bem estruturado e reutilizável, além de aprender a utilizar algumas tecnologias modernas de desenvolvimento Android.

No MVP foi implementado algumas das principais funcionalidades previstas e foi utilizado algumas das principais tecnologias previstas. As funcionalidades e tecnologias que não foram possíveis implementar durante a *Sprint* 3 serão implementadas futuramente; pois o período de desenvolvimento do aplicativo foi muito curto; somente sendo um MVP com poucas funcionalidades, como foi desenvolvido neste caso, para se encaixar no prazo de 2 semanas.

3. Considerações Finais

3.1 Resultados

Como resultado final do Projeto Aplicado tem-se muito aprendizado. Durante o decorrer do Projeto Aplicado foi possível aprender: 1) Conceitos, vistos em aula, de DesignThinking, que é uma abordagem para resolver problemas e fomentar a criatividade e a inovação. 2) Características de alguns dos padrões arquiteturais de *software* mais utilizados, com foco nos padrões utilizados para desenvolvimento *mobile*. 3) A importância de pesquisar os serviços e ferramentas disponíveis e de entender suas características, para assim fazer uma escolha mais acertiva dos serviços e das ferramentas a serem utilizados nos projetos. 4) A importância de planejar um projeto e também a importância de utilizar uma arquitetura adequada e bem definida e também a importância de escrever código limpo, bem estruturado e reutilizável, além de aprender a utilizar algumas tecnologias modernas de desenvolvimento Android.

Todo esse aprendizado gerou um produto MVP, um aplicativo Android que foi construído com base em toda a teoria estudada, aplicando-se a arquitetura de *software* e as tecnologias definidas, no qual foi possível colocar em prática a teoria. O código fonte do aplicativo desenvolvido está disponível em: <https://github.com/guilherme-boschetti/MyMoney>.

3.2 Contribuições

Além do aprendizado, este Projeto Aplicado contribuiu para a comunidade de desenvolvimento de *software*, uma vez que o projeto está disponibilizado em um repositório público no GitHub¹⁴, onde qualquer desenvolvedor pode ter acesso, usufruir e aprender com o projeto disponível.

Futuramente, quando o aplicativo for disponibilizado na loja (Play Store) poderá contribuir com o gerenciamento financeiro das pessoas interessadas em instalar o aplicativo em seus dispositivos Android e usá-lo.

¹⁴ <https://github.com/>

3.3 Próximos passos

Como próximos passos têm-se a implementação das funcionalidades e tecnologias que não foram possíveis implementar no desenvolvimento do MVP na *Sprint 3* e a publicação do aplicativo na loja (Play Store). Os próximos passos estão descritos a seguir:

- Implementação do *backup* dos dados no Firebase Realtime Database;
- Implementação do monitoramento de falhas com o Firebase Crashlytics;
- Implementação de testes de unidade no aplicativo;
- Implementação de notificações de gastos fixos não pagos próximo do vencimento;
- Implementar a adição do saldo do mês anterior no balanço do mês atual;
- Implementação de anúncios no aplicativo, para a monetização do mesmo;
- Publicação do aplicativo na loja (Play Store).

Durante o desenvolvimento do projeto surgiram novas ideias para o aplicativo, que também podem ser implementadas futuramente, como:

- Criar uma tela que mostra um gráfico dos gastos do mês;
- Exportar os gastos do mês para um arquivo de Excel.

4. Referências

AHMED, Shahbaz. **Getting started with android architecture components and MVVM Part-1**. 2017. (acessado: 18.06.2022). Disponível em: <<https://medium.com/android-news/getting-started-with-android-architecture-components-and-mvvm-156a96a1bd05>>.

BASS, Len. *et al.* **Software architecture in practice**. 3. ed. Addison-Wesley Professional, 2012. 624 p.

BOROVSKOY, Alex. **VIPER in Android: The Practical Guide or How to Catch a Snake?**. 2019. (acessado: 09.06.2022). Disponível em: <<https://medium.com/omisoft/https-medium-com-omisoft-viper-in-android-the-practical-guide-or-how-to-catch-a-snake-78cc17e96d63>>.

CABRAL, Diogo. **Arquitetura limpa nas apps: utilizando VIPER no Android**. 2019. (acessado: 09.06.2022). Disponível em: <<https://medium.com/android-dev-br/arquitetura-limpa-nas-apps-utilizando-viper-no-android-f39e51b44723>>.

CLARK, Mariana. **Os 10 melhores servidores de backend para um APP Android**. 2021. (acessado: 02.07.2022). Disponível em: <<https://blog.back4app.com/pt/os-10-melhores-servidores-de-backend-para-um-app-android/>>.

FORD, Neal. *et al.* **Software Architecture: The Hard Parts: Modern Trade-Off Analyses for Distributed Architectures**. 1. ed. O'Reilly Media, 2021. 464 p.

INGENO, Joseph. **Software Architect's Handbook: Become a successful software architect by implementing effective architecture concepts**. 1. ed. Packt Publishing, 2018. 594 p.

KATZ, Michael. **Getting Started with the VIPER Architecture Pattern**. 2020. (acessado: 09.06.2022). Disponível em: <<https://www.raywenderlich.com/8440907-getting-started-with-the-viper-architecture-pattern>>.

KAZAP. **Como escolher a arquitetura mobile da melhor forma?**. 2020. (acessado: 09.06.2022). Disponível em: <<https://blog.kazap.com.br/arquitetura-mobile/>>.

LANFREDI, Eduardo. **Injeção de Dependência**. 2017. (acessado: 28.06.2022). Disponível em: <<https://medium.com/@eduardolanfredi/inje%C3%A7%C3%A3o-de-depend%C3%Aancia-ff0372a1672>>.

PANDEY, Bipin. **VIPER-Architecture for iOS project with simple demo example.** 2018. (acessado: 09.06.2022). Disponível em: <<https://medium.com/cr8resume/viper-architecture-for-ios-project-with-simple-demo-example-7a07321dbd29>>.

RABELLO, Ramon Ribeiro. **Hilt Series: Introdução ao Dagger Hilt.** 2020. (acessado: 28.06.2022). Disponível em: <<https://medium.com/android-dev-br/hilt-series-introdu%C3%A7%C3%A3o-ao-dagger-hilt-50faa1b3a194>>.