**Relatório Técnico:**  
**CRUD – Lista de Despesas**

Bredley Bauer

Engenharia de Software - Centro Universitário - Católica de Santa Catarina

**1 INTRODUÇÃO**

Este aplicativo tem como objetivo auxiliar os usuários no controle de despesas mensais, permitindo registrar, visualizar, editar e excluir despesas. Desenvolvido em Flutter com integração ao Firebase, o projeto oferece uma interface intuitiva e funcionalidades de CRUD (Create, Read, Update, Delete) para armazenamento e manipulação dos dados financeiros. A aplicação classifica as despesas por categorias, exibe um resumo mensal e possibilita uma visão detalhada das despesas recentes.

**2.1 REQUISITOS FUNCIONAIS**

* **RF001 – Adicionar Despesa:** permitir que o usuário adicione uma nova despesa, selecionando categoria, valor, data e opcionalmente marcando a despesa para renovação mensal automática;
* **RF002 – Exibir Despesas recentes: e**xibir as 5 despesas mais recentes na tela inicial para fácil acesso e monitoramento dos gastos;
* **RF003 – Exibir Valor total do mês: e**xibir a soma do valor de todas as despesas adicionadas para o mês corrente;
* **RF004 – Exibir Despesas do mês atual:** listar todas as despesas do mês corrente na tela de relatório, em ordem de adição;
* **RF005 – Editar Despesa:** possibilitar a edição dos detalhes de uma despesa existente, incluindo nome, categoria e valor;
* **RF006 – Excluir Despesa:** permitir que o usuário exclua uma despesa específica, com uma confirmação prévia para evitar exclusões acidentais;
* **RF007 – Iniciar Tela de login:** iniciar o aplicativo na tela de login, que servirá de entrada enquanto o sistema de autenticação não é implementado;
* **RF008 – Possibilitar Logout:** permitir que o usuário desconecte sua conta através de um botão de logout na página principal.

**2.2 REQUISITOS NÃO FUNCIONAIS**

* **RNF001 – Usabilidade:** a interface do usuário deve ser intuitiva e de fácil navegação, permitindo que qualquer usuário, independentemente do nível de experiência, consiga utilizar o aplicativo sem necessidade de instruções complexas;
* **RNF002 – Desempenho:** as operações de adição, edição, e exclusão de despesas devem ser rápidas e não demorar mais do que poucos segundos para serem processadas e refletidas na interface do usuário;
* **RNF003 – Manutenibilidade:** o código deve ser estruturado de forma modular, facilitando futuras atualizações e manutenções no projeto - inclui a utilização de um padrão de arquitetura (MVVM) para separar a lógica de negócio da interface;

**3 ARQUITETURA DO SISTEMA**

A arquitetura do sistema segue o padrão MVVM (Model-View-ViewModel), promovendo separação entre lógica, interface de usuário e manipulação de dados:

* **Model:** classe *Despesa* representa o modelo de dados da despesa, com os atributos de título, valor, categoria, data e renovação;
* **ViewModel:** classe *DespesaViewModel* lida com a comunicação entre a *View* e o Firebase, gerenciando operações de adicionar, atualizar, excluir e recuperar dados;
* **View:** telas com elementos visuais Flutter para adicionar, editar e visualizar despesas.

lib/

├── main.dart

├── **models**/

│ └── despesa.dart

├── **viewmodels**/

│ └── despesa\_viewmodel.dart

└── **views**/

├── adicionar.dart

├── home.dart

├── login.dart

└── relatorio.dart

**4 EXPLICAÇÃO DO CÓDIGO**

* **Tela de adição/atualização de item:** o código da tela de adição permite que o usuário insira detalhes sobre uma nova despesa e envie os dados para o Firebase. A função *\_**addExpense()* chama o *ViewModel* para validar e salvar a despesa. Caso o usuário esteja editando uma despesa existente, os campos são preenchidos com os dados da despesa e uma função de atualização no *ViewModel* é chamada;
* **Leitura dos itens:** a tela inicial utiliza o método *despesasRecentesStream*, do *DespesaViewModel*, para obter as cinco despesas mais recentes do mês. A leitura ocorre em tempo real utilizando o *StreamBuilder*, que escuta alterações no Firebase e atualiza a interface automaticamente;
* **Atualização e exclusão:** o modal de edição permite ao usuário alterar os detalhes da despesa e salvar as mudanças no Firebase através do método *atualizarDespesa* no *ViewModel*. A exclusão é confirmada através de um diálogo, e, em caso positivo, a despesa é removida do Firebase;
* **Atualização e exclusão:** o modal de edição permite ao usuário alterar os detalhes da despesa e salvar as mudanças no Firebase através do método *atualizarDespesa* no *ViewModel*. A exclusão é confirmada através de um diálogo, e, em caso positivo, a despesa é removida do Firebase;
* **Validação e tratamento de erros:** a aplicação valida o título e o valor das despesas antes de enviá-los ao Firebase, garantindo que os dados sejam completos e válidos. Em caso de falhas de conexão ou erros durante operações com o banco de dados, o sistema exibe mensagens de erro ao usuário, orientando-o a tentar novamente;
* **Configuração e estrutura:** no Firebase, foi criada a coleção despesas, onde cada documento representa uma despesa com os campos *titulo*, *categoria*, *valor*, *data* e *renovacao*:

Coleção: **despesas**

└── Documento: ID gerado aleatoriamente

├── Campo: **titulo** (string)

├── Campo: **valor** (number)

├── Campo: **categoria** (string)

├── Campo: **renovacao** (boolean)

└── Campo: **data** (timestamp)

**5 REGRAS DE NEGÓCIO**

* **Validação de título e valor:** uma despesa só é salva se tiver um título não vazio e um valor maior que zero, garantindo consistência dos dados;
* **Limite de exibição:** na tela inicial, apenas as 5 despesas mais recentes são exibidas, para facilitar o monitoramento rápido dos gastos mais recentes;
* **Renovação automática:** despesas podem ser marcadas para renovação mensal, o que ajuda no gerenciamento de despesas recorrentes;
* **Categorias fixas:** as categorias de despesas são predefinidas e exibidas em uma lista suspensa, impedindo que o usuário crie novas categorias. Cada categoria é associada a um ícone personalizado, proporcionando melhor interface visual;
* **Paleta de cores:** a paleta do aplicativo utiliza tons escuros para reduzir o cansaço visual, especialmente em ambientes de baixa iluminação; as cores foram escolhidas para atender o padrão visual de aplicativos financeiros.

**6 DESAFIOS E SOLUÇÕES**

* **Configuração do Firebase e filtragem mensal:** configurar o Firebase para filtrar as despesas do mês atual em tempo real, mantendo a eficiência na consulta – a solução foi utilizar *where('data', isGreaterThanOrEqualTo: Timestamp.fromDate(startOfMonth))* no *despesasDoMesStream* em *DespesaViewModel*, garantindo que apenas as despesas do mês vigente fossem exibidas;
* **Implementação de validação de campos:** garantir validações no formulário de adição e edição – no método *\_addExpense* de *AdicionarState*, a validação envolve *if* *(!\_isValid(despesa))*, para verificar que o título não está vazio e que o valor é numérico e positivo. O erro é exibido com *\_showError*, validando *title* e *value*;
* **Manutenção da estrutura MVVM:** implementar a camada *ViewModel* para gerenciar a comunicação entre views e o Firebase – cada ação de CRUD, como *adicionarDespesa* e *excluirDespesa*, foi isolada no *DespesaViewModel*. Nos modais de edição, a chamada *\_viewModel.atualizarDespesa()* manteve o código da interface desacoplado da lógica;
* **Sincronização em tempo real:** integrar atualizações em tempo real nas views – utilizando *StreamBuilder* com o *despesasStream* do *DespesaViewModel*, o código atualiza automaticamente as despesas mais recentes no app e permite a organização das listas por ordem de adição.

**7 CONCLUSÃO**

Este projeto apresentou uma implementação completa e funcional de um aplicativo de controle financeiro pessoal, onde a estrutura MVVM e a integração com o Firebase construíram um código modular, reutilizável e conciso. Além de funcionalidades básicas, como adicionar, editar e excluir despesas, o projeto possui atualização em tempo real das informações, permitindo ao usuário acompanhar os gastos mensais com praticidade.

A experiência de desenvolvimento exigiu a solução de desafios em filtragem, otimização de consultas e comunicação entre as camadas, resultando em um projeto que cumpre os requisitos na apresentação das funcionalidades. O processo ofereceu aprendizado sobre boas práticas de desenvolvimento mobile, manipulação de dados em tempo real e a importância da coesão na arquitetura de software, com foco em integração a banco de dados.

**REFERÊNCIAS**

FLUTTER, Google. **Flutter documentation**. Disponível em: <https://docs.flutter.dev>. Acesso em: out. 2024.

FIREBASE, Google. **Documentação do Firebase**. Disponível em: <https://firebase.google.com/docs>. Acesso em: out. 2024.

FIREBASE, Google. **Adicionar o Firebase ao seu app Flutter**. Disponível em: <https://firebase.google.com/docs/flutter/setup >. Acesso em: out. 2024.