A B A K Ó S
PUC Minas Instituto de Ciências Exatas e Informática

Desenvolvimento de um sistema para criação de planos alimentares baseado em orçamento e preferência de refeições*

João Pedro Teixeira Dias[1]
Magali Rezende Gouvêa Meireles[2]
Octavio Augusto Pereira Martins[3]

Resumo

O presente trabalho descreve o desenvolvimento de um aplicativo móvel dedicado ao âmbito nutricional, oferecendo a capacidade de personalizar um plano de alimentação para qualquer pessoa. O objetivo é simplificar o processo de planejamento, economia e aderência a uma dieta. Com base nas informações fornecidas pelo usuário, o aplicativo permite a substituição de refeições e alimentos, respeitando as diretrizes da dieta. O aplicativo disponibiliza uma lista abrangente de alimentos e refeições, permitindo ao usuário realizar substituições. Quanto aos custos, cada item na lista é acompanhado pelo seu preço, conforme fornecido pelas lojas ou supermercados, ajudando o usuário a tomar decisões financeiramente conscientes.

Palavras-chave: Nutricional, Planejamento, Praticidade, Plano Alimentar, Gastos, Orçamento, Preferência.

*Artigo apresentado à Revista Abakos
[1]Bacharelando em Sistemas de Informação pela PUC Minas, Brasil, 1272244@sga.pucminas.br
[2]Doutora em Ciência da Informação pela UFMG, Brasil, magali@pucminas.br
[3]Bacharelando em Sistemas de Informação pela PUC Minas, Brasil, oapmartins@sga.pucminas.br

Abstract

This work describes the development of a mobile application dedicated to the nutritional field, offering the ability to customize a meal plan for anyone. The goal is to simplify the process of planning, saving and adhering to a diet. Based on the information provided by the user, the application allows you to replace meals and foods, respecting dietary guidelines. The application provides a comprehensive list of foods and meals, allowing the user to make substitutions. As for costs, each item on the list is accompanied by its price, as provided by stores or supermarkets, helping the user to make financially conscious decisions.

Keywords: Nutritional, Planning, Practicality, Diet Plan, Expenses, Budget, Preference.

1 INTRODUÇÃO

Um plano alimentar é um guia personalizado que delineia escolhas nutricionais e hábitos alimentares específicos para indivíduos. Adaptado às necessidades e metas de saúde, o plano alimentar serve como um roteiro detalhado que abrange desde a seleção dos alimentos até a organização das refeições. Ao criar um plano alimentar, consideram-se fatores como necessidades calóricas, preferências alimentares e objetivos de bem-estar. Além disso, outros elementos são levados em conta, incluindo a ingestão de nutrientes essenciais, variedade de alimentos para maior adesão ao plano e proporções equilibradas de macro e micronutrientes.

Na prática nutricional, a adesão dos pacientes aos planos alimentares propostos enfrenta barreiras e desafios significativos. Esses desafios se apresentam em diversos pontos, desde a resistência do paciente em mudar hábitos alimentares, até questões socioeconômicas, como a disponibilidade de certos alimentos no ambiente cotidiano ou restrições financeiras que limitam o acesso a determinados alimentos recomendados. Essas barreiras tornam-se ainda mais desafiadoras quando se considera a necessidade de manter a adesão a longo prazo, demandando estratégias personalizadas e um acompanhamento constante para promover uma mudança efetiva de hábitos alimentares.

A fim de auxiliar pacientes que buscam atingir seus objetivos e necessidades de saúde, aplicações estão sendo cada vez mais populares e reconhecidas pelos usuários e também pelos profissionais da saúde. Dentre esses aplicativos, os voltados à nutrição são reconhecidos por auxiliar usuários a manterem um estilo de vida saudável (WANGLER; JANSKY, 2021). Este trabalho visa unir os conhecimentos de desenvolvimento de aplicações, e os métodos nutricionais de construção de plano alimentar, para auxiliar o usuário final a adequar o plano alimentar ao seu orçamento disponível.

Diante disso, abre-se uma oportunidade de inovar e trazer ao mercado, uma aplicação que permite o usuário remanejar seu plano alimentar mantendo as mesmas propriedades calóricas e distribuição de macronutrientes. Com esta aplicação, o usuário pode adaptar suas refeições às necessidades individuais, substituindo alimentos por opções mais econômicas, práticas ou até mesmo disponíveis.

O objetivo geral deste trabalho é desenvolver um aplicativo que busca facilitar a vida de pessoas que pretendem fazer dieta, dando maior controle sobre os gastos com cada alimento e refeição, oferecendo opções alimentares que possam ser modificadas com base nas preferências e condição financeira. Para atingir esse objetivo, estabelecemos os seguintes objetivos específicos:

1. Criação de um sistema onde o usuário consegue informar uma dieta.
2. Criação de um sistema de trocas de alimentos respeitando as características e macronutrientes da dieta.
3. Criação de um sistema de cálculo de preço, onde o usuário poderá ver o valor da sua dieta.

4. Projeto de interface de usuário simples, intuitiva e amigável que é de fácil navegação e registro de informações.

Estes objetivos específicos foram definidos para garantir que o aplicativo atenda às necessidades dos usuários de maneira eficaz e eficiente, fornecendo controle financeiro e planos alimentares personalizados para facilitar a jornada de dieta de cada indivíduo.

As próximas seções estão organizadas da seguinte forma. A Seção 2 apresenta o referencial teórico, contendo informações sobre dieta e seus componentes, arquitetura, tecnologias adotadas no sistema, e artigos que foram relacionados com o presente trabalho. A Seção 3 apresenta a metodologia aplicada, o planejamento, a definição das etapas a forma de validação. A seção 4 apresenta o tópico de desenvolvimento da aplicação, onde é informado tudo que foi aplicado para construção da aplicação. Por fim, a seção 5, apresenta o tópico das considerações finais, onde é feito um resumo geral do trabalho e possíveis próximos passos para o sistema.

2 REFERENCIAL TEÓRICO

As próximas subseções apresentam conceitos relacionados a dieta e seus componentes, arquitetura e desenvolvimento de sistemas, e tecnologias e ferramentas para desenvolvimento do aplicativo.

2.1 Dieta, seus componentes e tipos

A dieta é um termo que se refere aos hábitos alimentares de um indivíduo, grupo ou comunidade. Ela engloba não apenas os alimentos consumidos, mas também a frequência, a quantidade e a qualidade das refeições. Esses hábitos alimentares são influenciados por diversos fatores como, cultura, disponibilidade de alimentos, preferências pessoais, necessidades nutricionais, restrições médicas e objetivos específicos, como perda de peso, ganho de massa muscular, controle de condições de saúde, entre outros.

As dietas são geralmente medidas em termos de ingestão calórica diária, comumente expressa em calorias(kcal) ou joules(J). Essa medida reflete a quantidade de energia obtida a partir dos alimentos consumidos ao longo do dia.

2.1.1 Componentes de uma dieta

Os carboidratos, proteínas e lipídios, por serem os nutrientes de maior contribuição energética para uma dieta, se enquadram na classe dos macronutrientes. Uma distribuição adequada desses nutrientes é de extrema importância para a nutrição de um paciente. Os macronutrientes têm valores energéticos distintos que influenciam diretamente na composição calórica

de uma dieta. Os carboidratos e as proteínas fornecem aproximadamente 4 kcal por grama, enquanto os lipídios contribuem com cerca de 9 kcal por grama. ((FORC), 2023) Esses valores são fundamentais para calcular a distribuição calórica dos macronutrientes em um plano alimentar. Essa distribuição é feita em relação ao percentual de calorias que o indivíduo necessita. Portanto, indivíduos diferentes, com necessidades e objetivos diferentes, terão diferentes distribuições de macronutrientes em seu plano alimentar.

As vitaminas e minerais são considerados os micronutrientes, sendo encontrados em alimentos como frutas, legumes e verduras. São considerados micronutrientes pois esses compostos não apresentam relevante potencial calórico e energético para uma dieta. No entanto são necessários para diversas reações químicas que ocorrem no corpo, tais como, correto funcionamento do sistema imunológico, visão, crescimento celular, entre outros (MAGNO et al., 2018).

2.1.2 Tipos de dieta e Abordagens

Existem diferentes abordagens quando se trata de seguir um plano alimentar. Dentre elas, três padrões principais são: Restrição rígida, Restrição flexível e Alimentação intuitiva (LINARDON et al., 2020).

A restrição rígida é tipo de dieta que se caracteriza por ser rígida e específica quanto aos tipos e quantidades de alimentos consumidos. Geralmente, é baseada em regras fixas e pode levar a uma abordagem mais inflexível em relação à alimentação.

Ao contrário da restrição rígida, a restrição flexível permite mais flexibilidade dentro de certos limites estabelecidos. Aqui, há uma conscientização das escolhas alimentares, mas ainda com uma estrutura definida para guiar a dieta. Essa estrutura e os limites são definidos de acordo com os macronutrientes que o indivíduo precisa para atingir o objetivo.

Os objetivos do presente trabalho estão alinhados com o modelo de restrição flexível. Uma vez que o usuário terá um acompanhamento de um profissional para a elaboração do plano alimentar, no qual serão estabelecidas a estrutura e os limites da dieta, e a aplicação terá total capacidade para oferecer ao usuário alternativas de alimentos diferentes do plano original, mantendo, no entanto, todas as propriedades calóricas e a distribuição de macronutrientes necessárias.

2.2 Arquitetura de Sistemas

A arquitetura adotada durante o desenvolvimento foi o MVC (*Model-View-Controller*). Este padrão de arquitetura foi criado por Trygve Reenskaug, em 1979. Com ele, é possível realizar a divisão do projeto em camadas, viabilizando a independência e reutilização de componentes, e o desenvolvimento de um código organizado e enxuto (SILVA, 2012).

A composição do MVC é feita pelas camadas de Modelo, Visão e Controle. O Modelo é responsável pela coleta, tratamento, acesso e manipulação dos dados dentro da aplicação. Ela se comunica diretamente com o controlador. A camada de Visão é a parte responsável pela interface do usuário. Nessa camada, os dados são requisitados ao Controlador que por sua vez requisitada ao modelo. O Controlador é a camada responsável por intermediar a conexão entre o Modelo e a Visão.

A escolha do modelo de *web services*(NOTE, 2004) para a comunicação entre as camadas do modelo MVC foi fundamental para garantir uma integração fluida e eficiente entre os componentes da aplicação. Os *web services* permitem a interação e troca de informações entre sistemas distintos de forma padronizada.

Para o *web service* em questão foi utilizado o modelo de arquitetura *Representational State Transfer* (REST). De acordo com a pesquisa de [FIELDING, 2002, p. 1]: "O objetivo do REST é atender às necessidades de uma rede hipermídia distribuída em escala de Internet, tentando minimizar a latência e a comunicação de rede enquanto ao mesmo tempo, maximizando a independência e escalabilidade das implementações de componentes."

2.3 Scrum e Metodologias Ágeis

As Metodologias Ágeis revolucionaram o desenvolvimento de software, priorizando a adaptabilidade, colaboração e entrega incremental. Dentro desse contexto, o Scrum se destaca como um dos *frameworks* mais adotados, oferecendo uma abordagem flexível e iterativa para gerenciar projetos complexos. Em termos de vantagens, o Scrum proporciona uma melhor visibilidade do progresso do projeto, permitindo ajustes contínuos, maior engajamento das partes interessadas e entrega de valor de forma incremental, o que possibilita a adaptação às mudanças de requisitos com mais facilidade, baseado nos três pilares, Transparência, Fiscalização e Addaptação (SCHWABER; SUTHERLAND, 2020).

O conceito do Scrum se encaixa de forma parcial ao projeto, de modo que técnicas como *Sprint Planning*, *Sprint Review*, *Sprint Backlog* e *Increment* fazem total sentido para o desenvolvimento da aplicação, mas devido ao tamanho limitado do time, a existência de um *Product Owner*(PO) ou um *Scrum Master*(SM) não se faz necessária. Além disso, o conceito de entregas menores e contínuas faz-se de extrema importância, uma vez que existem duas frentes de desenvolvimento que precisam correr em paralelo. Desse modo, com as entregas alinhadas de acordo com as necessidade de cada uma das frentes, o desenvolvimento da aplicação como um todo fica dinâmico.

2.4 Tecnologias e Ferramentas

As próximas subseções apresentam conceitos relacionados às tecnologias e às plataformas utilizadas para desenvolvimento do sistema.

2.4.1 Dart

Dart é uma linguagem desenvolvida pela Google, para criação de *softwares* multiplataforma. Seu lançamento ocorreu na GOTO Conference 2011, Dinamarca (RISSI; DALLILO, 2022).

Sua missão inicial era substituir o JavaScript como principal tecnologia presente nos navegadores, a ideia era oferecer uma alternativa mais moderna e robusta para o desenvolvimento *Web*. Um dos pontos que dificultaram sua adoção, no início, foi o JavaScript já ser uma linguagem consolidada no mercado, e também a má fama do Google em descontinuar projetos.

O Dart é uma linguagem orientada a objetos, mas pode ser considerada flexível, que permite que os programadores definam variáveis que aceitam todos os tipos de valores, chamadas de variáveis dinâmicas. A linguagem possui sintaxe com estilo baseado no C. Isso faz com que seja muito similar à linguagens como Java e C#.

2.4.2 Flutter

Flutter foi criada pelo Google como uma plataforma de desenvolvimento móvel de código aberto, também destinada a auxiliar os desenvolvedores na criação de interfaces para múltiplas plataformas. É construída na linguagem Dart, que é uma linguagem orientada a objetos que se compila em código nativo. É de fácil aprendizado para desenvolvedores provenientes de outras linguagens como Java e JavaScript, e oferece recursos como forte tipagem, genéricos e programação assíncrona (WU, 2018).

O Flutter é baseado no conceito de *widgets*, que são basicamente componentes *UI* que podem ser usados para criar interfaces de usuário complexas com apenas algumas linhas de código. Esse *framework* fornece um conjunto de *widgets* pré-construídos, mas também possibilita que os desenvolvedores façam seus próprios. Os *widgets* são classificados como *Stateless* e *Stateful widgets*, dependendo se eles gerenciam ou não algum estado. *Widgets Stateless* são aqueles que simplesmente exibem informações e não exigem nenhum dado mutável. Os *Stateful widgets*, por outro lado, gerenciam o estado. Este estado pode ser interno ao próprio *widget*, ou pode vir de uma fonte externa. Para que um *Stateful Widget* possa mudar sua aparência em resposta a eventos, ele precisa ter acesso a um objeto imutável conhecido como uma instância *State* (WU, 2018; FENTAW, 2020).

Várias camadas compõem a estrutura Flutter (SHAH et al., 2019). A renderização da interface do usuário é de responsabilidade da camada *Widgets*. Ela usa uma abordagem declar-

ativa para definir a interface de usuário. Isto significa que não é necessário escrever código imperativo para atualizar a interface de usuário. A camada *Engine*, por sua vez, está encarregada de coordenar a comunicação entre o *framework* e o código nativo da plataforma. Por fim, a camada de fundação fornece serviços fundamentais, incluindo rede, E/S e suporte de acessibilidade.

2.4.3 Python

Python é uma linguagem de programação de alto nível que se destaca por sua simplicidade e legibilidade. Criada pelo matemático Guido van Rossum, teve sua primeira versão lançada em 1990. Desde então, a linguagem tem evoluído com a colaboração de uma comunidade global de desenvolvedores e tornou-se uma das linguagens de programação mais populares do mundo (COUTO; SILVA, 2021).

O Python se caracteriza pela ênfase na legibilidade do código, o que o torna uma escolha ideal para programadores, desde iniciantes até profissionais experientes. A linguagem utiliza indentação para definir blocos de código, tornando o código mais claro e fácil de entender.

O Python se tornou uma parte importante da ciência de dados e da inteligência artificial. A ampla variedade de bibliotecas e *frameworks*, a facilidade na integração com outras tecnologias, a comunidade ativa, e o aprendizado de máquina e processamento de linguagem natural, são pontos que ajudam na adoção quando o assunto é inteligência artificial. Os cientistas e analistas utilizam códigos Python para analisar *big datas*, conjuntos de dados e projetar algoritmos de aprendizado de máquina (FOUNDATION, 2023).

2.4.4 Firebase

Firebase é uma plataforma de gerenciamento e desenvolvimento de dispositivos móveis, desenvolvida pela Firebase, Inc. em 2011 e adquirida pelo Google em 2014. A plataforma surgiu de uma necessidade dos desenvolvedores em gerenciar várias infraestruturas que faziam parte de um mesmo aplicativo. Com ela, os desenvolvedores podem focar na criação de sua própria aplicação, poupando tempo e recursos (LI et al., 2018).

Atualmente o Firebase é dividido em 3 partes nomeadas de : *Analytics*; *Develop* e *Grow*. O *Analytics* é a seção onde são disponibilizadas informações sobre o aplicativo em geral. É possível obter informações como: localidade, idade e sexo dos usuários, eventos que acontecem dentro do aplicativos e erros de sintaxe. O firebase oferece uma função de exportar todos os dados coletados para um *data warehouse* hospedado em nuvem, permitindo a criação de relatórios detalhados de dados de forma personalizada.

A seção de *Develop* é onde são localizadas as ferramentas de desenvolvimento que auxiliam o desenvolvedor. As informações são armazenadas no *Realtime Database*, que é uma base

de dados hospedada em nuvens que utiliza a linguagem *NoSql*. Para manter a lógica e regra de negócio do *backend* centralizada e segura, foi criado o *Cloud Functions*, que exclui a necessidade de gerenciar o próprio servidor. Para a criação e gerencimento de domínios, existe o *Firebase Hosting*. Para *download* e *upload* de arquivos é utilizado o *Firebase Storage*. Os erros que acontecem nos aplicativos podem ser capturados pelo *Crash Reporting*, onde o desenvolvedor consegue a diagnosticar a causa das possíveis falhas. E por fim, opções de gerencimanento de desempenho e performance também são oferecidas pelo *Performance Monitoring* (CHATTERJEE et al., 2018).

A seções de *Grow* é a parte relacionada ao crescimento e engajamento. São fornecidas funções que ajudam na realização testes dentro do aplicativo, enviar mensagens para os usuário e visualizar padrões de comportamento dentro do aplicativo.

2.5 Trabalhos Relacionados

O primeiro artigo é analisado é o PRUS: Sistema de recomendação de produtos baseado em especificações de usuários e avaliações de clientes. A recomendação dos produtos é realizada com base nos pontos acumulativos que o usuário recebe ao avaliar cada produto. Os resultados sugerem que o PRUS confere independência ao usuário, permitindo-lhe selecionar a lista recomendada com base em características específicas relacionadas aos aspectos positivos ou negativos dos produtos. (HUSSAIN et al., 2016)

O segundo artigo apresenta o CardNutri: Um *software* de planejamento de cardápios nutricionais semanais para alimentação escolar aplicando inteligência artificial (MOREIRA et al., 2017). A técnica escolhida para realizar a recomendação automática dos cardápios foi a utilização de Algoritmos Genéticos, uma abordagem que se baseia no processo de evolução biológica. À medida que o processo evolui, soluções mais eficientes são obtidas. Dessa forma, a recomendação é personalizada levando em consideração faixa etária e a quantidade necessária de micronutrientes para uma alimentação adequada. Essa estratégia visa proporcionar cardápios adaptados às necessidades específicas de cada usuário.

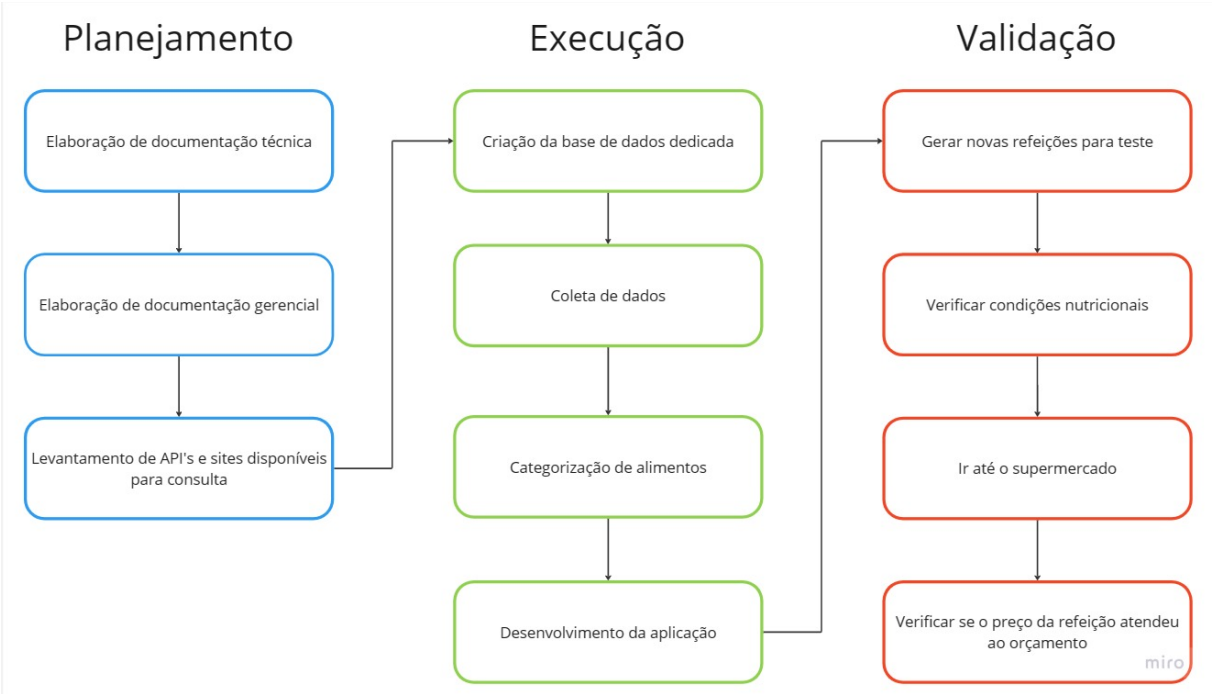
Como terceira análise realizada, foi utilizado um trabalho de conclusão de curso, que apresenta recomendação de presentes em dispositivos móveis. A análise do perfil de cada usuário é feita a partir de compras que foram realizadas, ou seja, conforme o histórico de produtos adquiridos é feito o mapeamento de interesses. Para a recomendação, os interesses são utilizados para recomendar produtos semelhantes aos que o usuário já adquiriu (PEREIRA; COSTA, 2016).

Este trabalho tem análise dos produtos baseada nas comparações das tabelas nutricionais, localidade do usuário, e o preço dos produtos, usando a localidade para aferir de forma mais precisa os preços. Além disso, a recomendação de alimentos é feita com base no plano alimentar, na localidade, no orçamento do paciente e nas preferências do usuário, que pode selecionar quais tipos de alimentos deseja consumir em quais faixas de horários do dia.

3 METODOLOGIA

Para a metodologia, foi adotado um modelo sequencial baseado em três etapas: Planejamento, Execução e Validação, as quais são detalhadas na Figura 1. No planejamento, as terefas executadas, se dão pela elaboração de documentos e levantamento de *API's*. Na execução foram os processos técnicos, como criação de banco de dados, coleta de dados, categorização de alimentos e o desenvolvimento da aplicação. Para a validação, foram geradas novas refeições para teste, verificar condições nutricionais, ir até o supermercado e verificar se o preço da refeição atender ao orçamento.

Figura 1 – Etapas metodológicas



Fonte: elaborada pelos autores.

3.1 Planejamento

Para o planejamento, foram definidos documentos técnicos em auxílio do processo de construção do *software*. Dentre eles estão:

- Levantamento de requisitos funcionais e não funcionais. São os nossos guias para a elaboração das funcionalidades do sistema. Tudo o que estiver contido nos documentos possibilita que o *software* seja atualizado e reparado sempre que necessário de acordo com o que foi inicialmente estipulado.

- Casos de uso baseados em histórias de usuários. É importante trazer situações reais para a aplicação, portanto a utilização de historias de usuários é indispensável.
- Restrição e mecanismos arquiteturais. Foi feito um levantamento das tecnologias a serem utilizadas para atender a todas as demandas funcionais e não funcionais do sistema.

3.2 Execução

A etapa de execução foi subdividida entre: Elaboração de diagramas, Fonte de Dados e Desenvolvimento da aplicação.

3.2.1 Elaboração de diagramas

Ainda na área da documentação técnica, a elaboração de alguns diagramas é a primeira etapa da execução do projeto. Esses diagramas são facilitadores e fontes de consulta para o desenvolvimento da aplicação. Os documentos definidos para esta etapa foram:

- Documento arquitetural do sistema.
- Diagrama de classes.
- Diagrama de casos de uso.
- Requisitos funcionais e não-funcionais.

3.2.2 Fontes de dados

Este projeto utiliza duas fontes de dados principais para coletar informações nutricionais. Ambas as fontes são acessadas por meio de um *Web Crawler*, que é um programa automatizado que navega pela *web* e coleta informações. Essas duas fontes de dados, quando combinadas, fornecem uma visão completa e detalhada das informações nutricionais disponíveis, permitindo análises aprofundadas e precisas.

3.2.3 Desenvolvimento da aplicação

A etapa de desenvolvimento da aplicação foi subdividida entre: *back end* e *front end*.

3.2.3.1 Desenvolvimento do *back end*

O *back end* do projeto consiste nos componentes Modelo e Controle da arquitetura MVC. O componente Modelo foi desenvolvido em Python, aproveitando as vantagens dessa linguagem de programação de alto nível, com uma comunidade robusta e vasta biblioteca de recursos para *Web Scraping* e *NoSQL*. Para a coleta de dados, foram utilizadas as bibliotecas *BeautifulSoup* e *Selenium*, enquanto para a comunicação com a camada de persistência dos dados *Firebase* foi utilizada a biblioteca *Firebase-admin* disponibilizada pela própria *Google.*

O componente Modelo também foi desenvolvido em Python utilizando a biblioteca *Flask* para a criação um *web service RESTful*. A implementação do *web service* seguiu a arquitetura *RESTful*, promovendo a comunicação padronizada entre sistemas. O código foi organizado de maneira modular, simplificando tanto a manutenção quanto a expansão do sistema.

3.2.3.2 Desenvolvimento do *front end*

No desenvolvimento do *front end*, foi utilizado o *framework* Flutter em conjunto com a linguagem Dart. No ambiente Flutter, foram empregadas diversas bibliotecas com o intuito de facilitar o desenvolvimento da aplicação, entre suas funcionalidades estão: estabelecer conexão com o backend, conectar-se ao Firebase, ampliar a variedade de ícones disponíveis, além de bibliotecas auxiliares para a criação de componentes personalizados, como caixas de seleção e mensagens de notificação customizadas.

3.3 Validação

Para a validação do aplicativo, foram utilizados planos alimentares reais que já continham metas de calorias, ou macronutrientes previstas. Esses planos foram reproduzidos no aplicativo, buscando uma variedade maior de alimentos em cada um, e após isso os resultados do aplicativo foram comparados com a previsão dos nutricionistas para aquela alimentação. Essa comparação usou como base maior o valor de calorias, e a partir da diferença entre o plano alimentar real e o resultado do aplicativo, foi calculada a margem de erro que cada plano gerou.

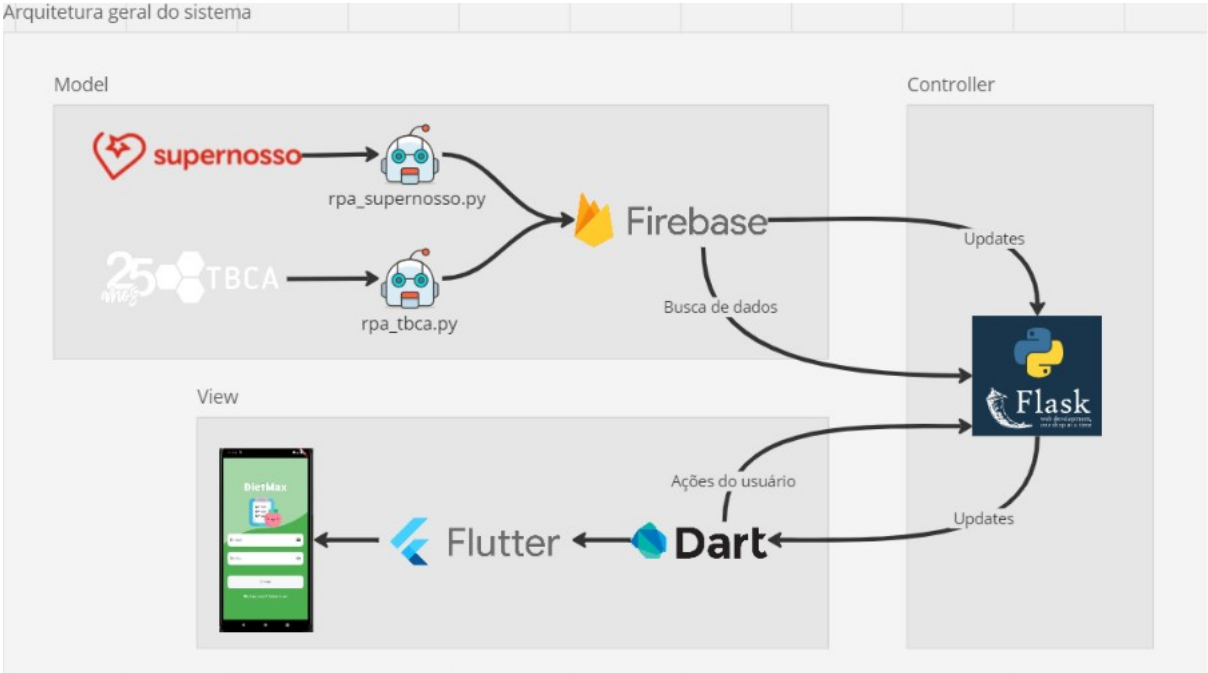
4 DESENVOLVIMENTO

Nesta seção, são apresentadas as três etapas do desenvolvimento da aplicação, das quais são: Planejamento, Execução e Validação.

4.1 Planejamento

Nessa seção, é mostrado como foi desenvolvido a aplicação. A Figura 2 apresenta a arquitetura geral do sistema, demonstrando o modelo MVC e as tecnologias utilizadas:

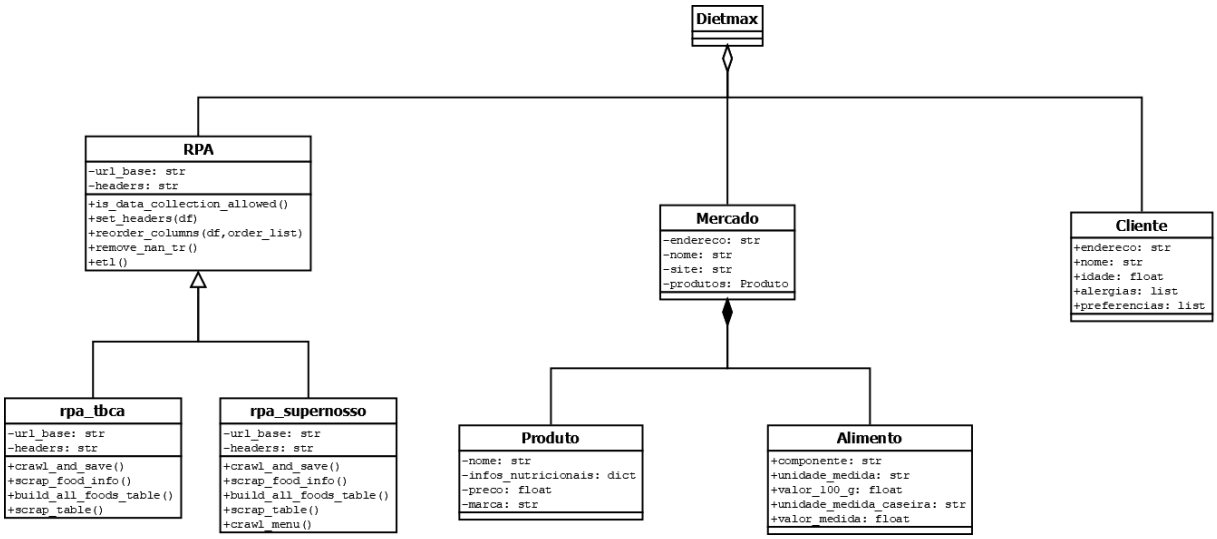
Figura 2 – Arquitetura geral do sistema



Fonte: elaborada pelos autores.

A Figura 3 representa o diagrama de classes do projeto, que é composto por 8 classes. Essas classes representam os *Web Crawlers*, os usuários do aplicativo "Clientes" e os alimentos que são sugeridos.

Figura 3 – Diagrama de classes



Fonte: elaborada pelos autores.

As Figuras 4 e 5, representam respectivamente, os requisitos funcionais e os requisitos não funcionais obtidos. Na primeira coluna da tabela são apresentados os requisitos, na segunda coluna, a descrição do requisito e, na terceira coluna, a classificação de prioridade.

Figura 4 – Requisitos Funcionais

Requisitos	Descrição	Prioridade
RF1: Inserir dieta	O aplicativo deve permitir que os usuários insiram informações sobre sua dieta e defina o horário de cada refeição.	Alta
RF2: Visualização da dieta	O aplicativo deve permitir a visualização simplificada da dieta inserida pelo usuário.	Alta
RF3: Visualizar valor nutricional	O aplicativo deve permitir que os usuários visualizem o valor nutricional dos alimentos.	Alta
RF4: Seleção de alimentos variados	O aplicativo deve fornecer ao usuário uma gama de alimentos diversificados para montar a sua dieta.	Média
RF5: Recomendação de alimentos	O aplicativo deve fornecer recomendações personalizadas com base nas informações inseridas pelo usuário	Alta
RF6: Edição dos dados do usuário	O aplicativo deve permitir o usuário alterar dados pessoais.	Baixa

Fonte: elaborada pelos autores.

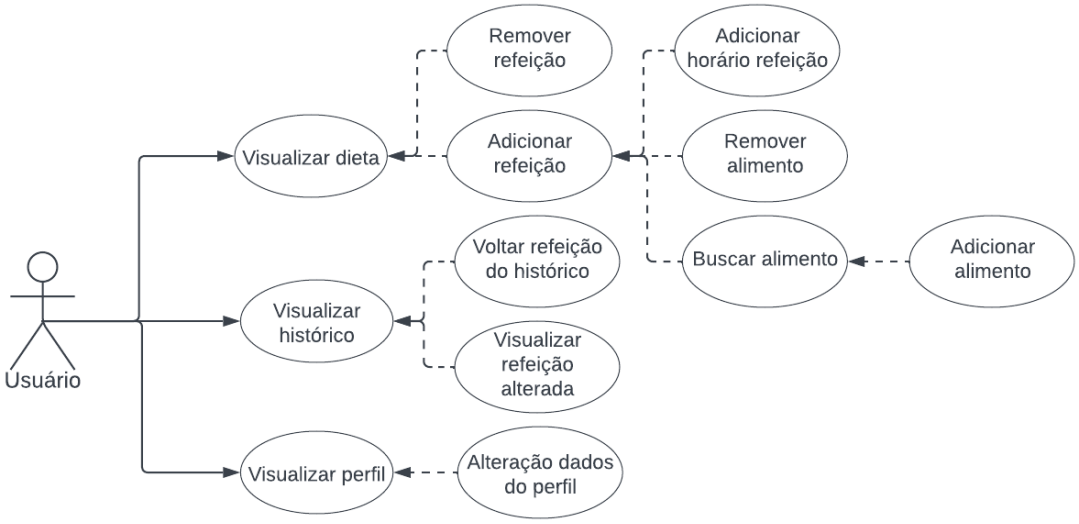
Figura 5 – Requisitos Não-Funcionais

Requisitos	Descrição	Prioridade
RNF1: Usabilidade	O aplicativo deve ser de fácil utilização e ter uma interface amigável. Onde o usuário consiga se localizar sem muito esforço.	Média
RNF2: Desempenho	O aplicativo deve ser capaz de lidar com grandes quantidades de dados sem diminuir o desempenho.	Alta
RNF3: Experiência do Usuário	O aplicativo deve ser capaz de fornecer recomendações precisas e confiáveis com base nas informações inseridas pelo usuário.	Alta
RNF4: Segurança	O aplicativo deve ser seguro e proteger as informações do usuário.	Alta
RNF5: Portabilidade	O aplicativo deve ser compatível com diferentes dispositivos móveis. Sendo possível a utilização em Android e IOS.	Média

Fonte: elaborada pelos autores.

A Figura 6 representa as principais funcionalidades do sistema, fornecendo aos usuários a capacidade de criar planos alimentares personalizados, substituir alimentos, calcular custos, e receber recomendações de refeições adaptadas às suas preferências.

Figura 6 – Casos de uso



Fonte: elaborada pelos autores.

4.2 Execução

As telas do aplicativo, apresentada nesta subseção, foram projetadas para manter o fácil entendimento dos fluxos por qualquer faixa etária. Esta subseção se divide em: Fonte de dados, Desenvolvimento da aplicação e Validação.

4.2.1 Fontes de dados

Este trabalho utiliza duas fontes de dados principais para coletar informações nutricionais. Ambas as fontes são acessadas por meio de um *Web Crawler*, que é um programa automatizado que navega pela *web* e coleta informações. Essas duas fontes de dados, quando combinadas, fornecem uma visão completa e detalhada das informações nutricionais disponíveis, permitindo análises aprofundadas e precisas.

4.2.1.1 Tabela Brasileira de Composição de Alimentos

A Tabela Brasileira de Composição de Alimentos (TBCA) é a primeira fonte de dados

do projeto. A TBCA é uma compilação extensa e minuciosa de informações nutricionais relacionadas a uma vasta variedade de alimentos amplamente consumidos no Brasil. Essa valiosa fonte de dados é resultado de um estudo cuidadoso e abrangente que visa fornecer informações detalhadas sobre calorias, proteínas, carboidratos, gorduras, fibras, vitaminas e minerais presentes em cada alimento ((FORC), 2023).

4.2.1.2 *Site* do SuperNosso

O SuperNosso é um supermercado online que oferece uma variedade de produtos alimentícios, desde frutas e verduras até carnes e laticínios. Para obter as informações nutricionais desses produtos, o projeto utiliza um *Web Crawler*, que é um programa que navega pelo site do SuperNosso e extrai os dados relevantes. Dessa maneira, o projeto pode acessar uma grande quantidade de produtos e suas respectivas informações nutricionais, como calorias, proteínas, gorduras, vitaminas e minerais. Esses dados são essenciais para a análise e a recomendação de alimentos adequados ao orçamento e às preferências do usuário.

4.2.2 Desenvolvimento da aplicação

A etapa de desenvolvimento da aplicação foi subdividida entre: *back end* e *front end*.

4.2.2.1 Desenvolvimento do *back end*

O desenvolvimento do *back end* contém os componentes de *Model* e *Controller* baseado na arquitetura MVC, ambos feitos em Python. Sendo o *Model* de persistência composto por dois *Web Crawlers* e a camada de inserção dos dados coletados para o banco com o Firebase. Já o *Controller* é composto por um *Web Service* que comunica com Firebase e formata os dados para entregar ao componente *View*.

O primeiro *Web Crawler* do componente *Model* é o TBCA, que foi organizado em 3 diferentes arquivos, e seguem as etapas ETL.

- O arquivo de extração de dados é chamado "rpa_tbca.py", ele utiliza as bibliotecas *BeautifulSoup e Requests* para percorrer todas as páginas de "composição de alimentos em medidas caseiras" do site da TBCA, além disso, ele coleta todas as informações a respeito daquele alimento como, os componentes de nutrição medidos, as informações dos macronutrientes e as quantidades, essas informações todas vem em formato de tabela.
- O arquivo de tratamento dos dados é chamado "etl_tbca.py" utiliza a biblioteca *Pandas* para fazer as tratativas necessárias, como transposição de colunas, remoção de caracteres e colunas e ordenação.

- Por fim, o arquivo de inserção de dados é chamado "db_tbca.py" e utiliza a classe *Db_firebase* que foi criada pelos autores para fazer toda a comunicação com o Firebase. Esse componente é responsável por formatar os dados no padrão criado para armazenamento e de fato inserir.

O segundo *Web Crawler* do componente *Model* é o SUPERNOSSO, que foi organizado também em 3 diferentes arquivos seguindo a mesma estrutura do TBCA.

- O arquivo de extração de dados é chamado "rpa_supernosso.py", ele utiliza as bibliotecas *Selenium, Requests BeautifulSoup e Regex* para percorrer algumas das categorias de alimentos comercializadas no Supernosso. Essa seleção de categorias levou em consideração apenas as categorias que contém alimentos, portanto, categorias de bebidas e de outros itens não foram selecionadas. A utilização da biblioteca *Selenium* foi necessária pois o conteúdo das páginas do site do Supernosso é dinâmico, então buscamos o recurso de simular a utilização humana para carregar e ser possível acessar os conteúdos via *Web Scrapping*. Além disso, devido a necessidade de recalcular a dieta do usuário baseado nos macros, somente foram coletados os produtos que contém informação nutricional em sua descrição.

- O arquivo de tratamento dos dados é chamado "etl_supernosso.py" utiliza a biblioteca *Pandas* para fazer as tratativas necessárias. Boa parte das tratativas é semelhante ao "etl_tbca.py", porém, pela diferença na forma de receber os dados que são coletados como texto pelo RPA, foi necessária uma tratativa mais específica.

- Por fim, o arquivo de inserção de dados é chamado "db_tbca.py" e utiliza a classe *Db_firebase* que foi criada pelos autores para fazer toda a comunicação com o Firebase. Esse componente é responsável por formatar os dados no padrão criado para armazenamento e de fato inserir.

A camada de persistência do componente *Model* é composta somente pelo Firebase. Os dados foram divididos em 3 diferentes coleções, "SUPERNOSSO", "TBCA" e "USUARIOS". Os dados das coleções "SUPERNOSSO" e "TBCA" são armazenados em documentos com estruturas semelhantes, porém cada um com suas especifidades. Além disso, cada alimento das coleções tem também um padrão a ser seguido, porém, esse padrão não é engessado pois cada alimento tem diferentes componentes e informações, e para o funcionamento da aplicação é necessária essa versatilidade que um banco de dados *NoSQL* pode fornecer. Já a coleção "USUARIOS" contém um documento com as informações do usuário, mas também é composta por duas outras coleções, "dieta_base" e "Historico". A coleção "dieta_base" contém todas as refeições que stão na dieta do usuário atualmente. A coleção "Historico" contém todas as refeições que já estiveram um dia na dieta do usuário. Portanto, uma vez que um usuário faz uma troca de alimentos em uma refeição, uma nova refeição é criada, adicionada à "dieta_base" do usuário e a refeição que foi modificada fica armazenada no "Historico" para futuros acessos.

O componente *Controller* foi feito utilizando a biblioteca *Flask* e foi criada uma *API* para comunicação com os componentes *View* e *Model*. Essa *API* foi hospedada em uma plataforma

de desenvolvimento como serviço chamada *Railway*. A *Railway* é uma *Paas, Platform as a Service* que permite que usuários tenham U$5 dólares de crédito ao se registrarem na plataforma, e esses U$5 dólares são contabilizados por hora/uso da plataforma.

A *API* contém ao todo 7 rotas, sendo elas 5 para busca de informações e 2 para inserção de informações na cama de persistência, sendo elas:

- "get_all_foods" que não recebe nenhum parâmetro nem *body* e retorna todos os alimentos disponíveis no Firebase, porém essa rota não retorna todas as informações dos alimentos, retorna as informações básicas.
- "get_product" que recebe como parâmetro o *id* de um produto específico e retorna todas as informações desse produto.
- "get_user_diet" que recebe como parâmetro o *user_id* do usuário e retorna todas as informações da dieta do usuário.
- "get_similar_macro_predominante" que recebe como parâmetro um macronutriente predominante e retorna todos os alimentos que tem aquele macronutriente como predominante também. Essa rota é utilizada no momento da troca de alimentos do usuário, pois caso ele esteja trocando um alimento que tem como macro predominante a proteína por exemplo, ele receberá como sugestão outros alimentos que também são mais ricos em proteínas.
- "get_history" que recebe como que recebe como parâmetro o *user_id* do usuário e retorna todas as refeições que estão armazenadas no histórico de refeições do usuário.
- "set_user" que recebe como parâmetro o *user_id* do usuário e também recebe como *body* as informações do usuário a serem inseridas no Firebase.
- "insert_diet" que recebe como parâmetro o *user_id* do usuário e também recebe como *body* toda a informação da dieta do usuário. Essa rota é utilizada também para as trocas de alimentos. Sempre recebendo toda a dieta do usuário, essa rota é responsável por identificar se houve qualquer alteração na dieta, seja de alimentos, seja exclusão de refeição, seja adição de refeição.

4.2.2.2 Desenvolvimento do *front end*

Para a criação da interface, foi utilizado a linguagem Dart e seu *framework* Flutter, dando assim a liberdade do sistema operar em multiplataformas. Para o desenvolvimento das telas, não foram encontrados grandes problemas. Para o *design* das telas foram realizadas buscas de inspirações em sistemas já existentes.

A estrutura das pastas do projeto foi organizada de acordo com o seguinte modelo: a pasta *config* abriga as configurações de variáveis da *Api*; a pasta *pages* contém os módulos de

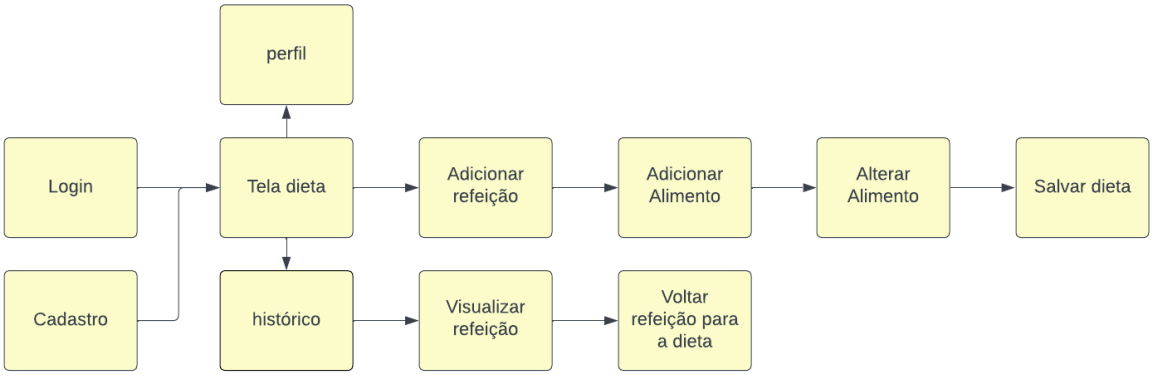
telas da aplicação; a pasta *service* é responsável pela conexão com o *back end*; e a pasta *util* inclui todos os *widgets* que podem ser reutilizados em outras telas. Cada uma dessas pastas tem um propósito específico, garantindo uma organização eficiente do projeto.

O funcionamento de todas as telas são mantidos com os controladores, que são criados a partir da inicialização das telas. Cada uma tem o seu controlador específico, e nele são armazenados as regras de negócio para cada componente. Dentro dos controladores também existem métodos que fazem conexão com os arquivos de serviço, esses que se conectam a *Api*.

A principal biblioteca da aplicação é o *GetX*, com ele foi possível manter o desacoplamento total da *View* com o *Controller*, realizar a injeção de dependência e navegação sem a necessidade do contexto. Bibliotecas para seleção de imagens na galeria, adicionar fontes do Google ao projeto e adicionar componentes visuais, também foram utilizadas. Todas as rotas da aplicação, foram mapeadas e nomeadas utilizando também o utilizando.

A tela de perfil foi construída de maneira diferente das outras. Para ela, não foi necessário a criação de serviços de consulta nem alteração. Foi utilizado a biblioteca própria do Firebase para buscar e realizar alterações nos dados do usuário.

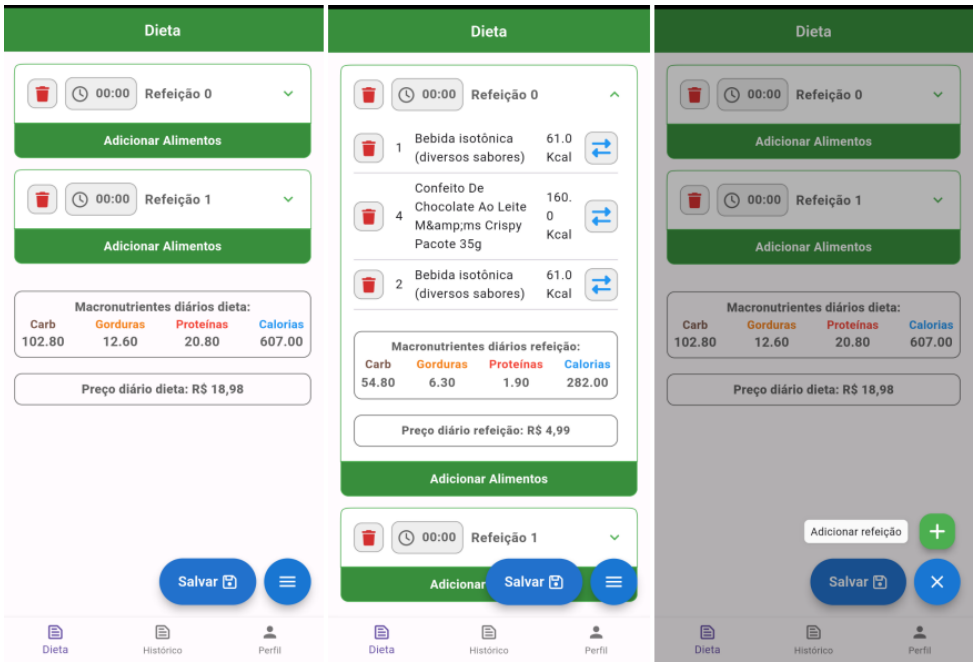
Figura 7 – Fluxo de Utilização



Fonte: elaborada pelos autores.

A figura 7 representa o fluxo de interação com o aplicativo, descrevendo o modelo sequencial que o usuário terá na utilização do aplicativo. Iniciando pelo login ou cadastro, seguindo para a tela principal de dieta, podendo ir para o histórico ou para o perfil. Dentro da tela da dieta existem as telas de consulta dos alimentos e tela de adição de alimentos.

Figura 8 – Telas de Dieta



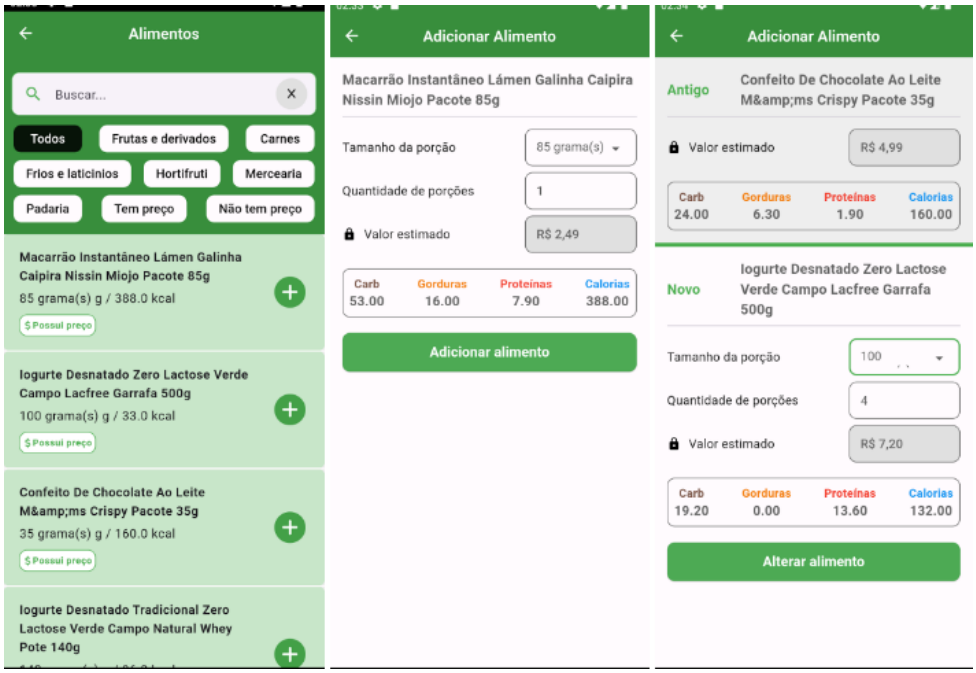
Fonte: elaborada pelos autores.

Para iniciar o fluxo do aplicativo, é exibido a tela de objetivos, apresentada na Figura 8. É a tela principal do aplicativo, e onde se encontra a dieta do usuário. Ela é composta por uma lista de refeições, em que é possível realizar diversas ações. Cada refeição oferece opções como excluir (representada por uma lixeira vermelha) e adicionar hora (representada por um relógio). Dentro de cada refeição, o usuário pode localizar os alimentos adicionados, tendo a capacidade de excluir (ícone de lixeira vermelha) ou alterar (ícone de setas azuis) cada item. As informações apresentadas para cada alimento incluem a quantidade selecionada, o nome e as calorias.

Ao final de cada *card* de refeição é possível visualizar um resumo dos macronutrientes, tais como carboidratos, gorduras, proteínas e calorias. O valor total diário da refeição também é exibido ao final do componente. Cada *card* de refeições possui um botão "Adicionar Alimentos", que direciona para a tela de seleção dos alimentos.

Ao final da página é possível visualizar uma tabela contendo os macronutrientes totais da dieta, e o valor total diário. É possível encontrar botões flutuantes para salvar a dieta do usuário e acessar um menu, que apresenta a opção para adicionar novas refeições.

Figura 9 – Tela de Busca e Adição de alimento



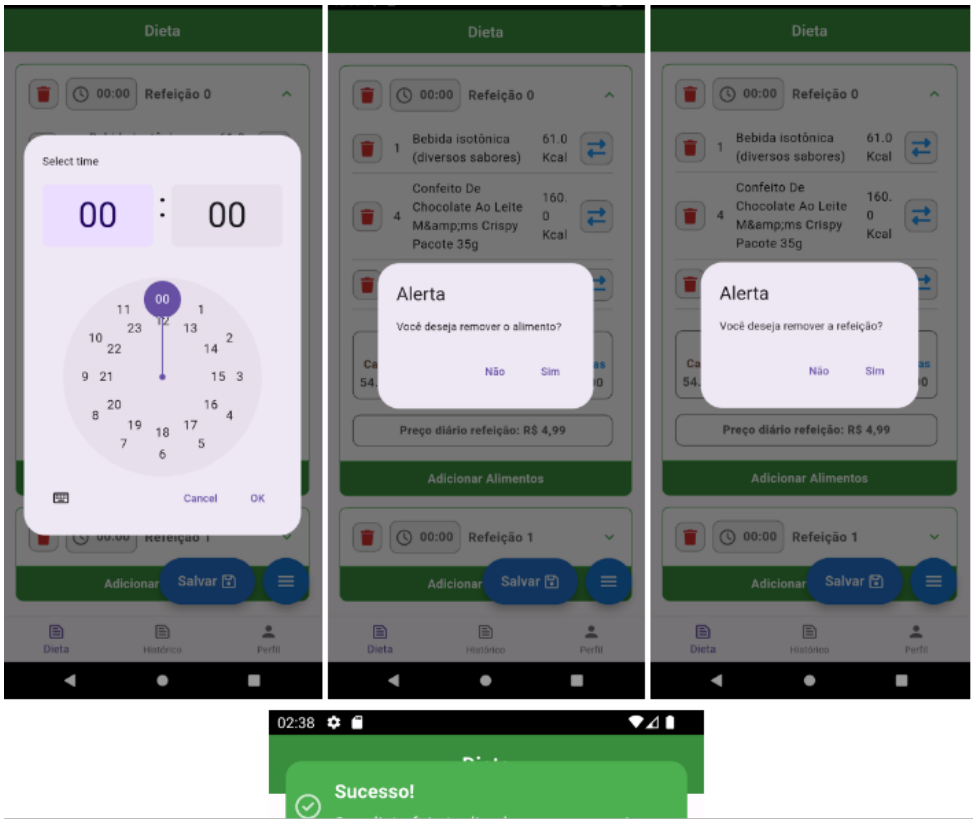
Fonte: elaborada pelos autores.

A tela de listagem dos alimentos, que é representada pela 9, é exibida após a solicitação ser feita pelo botão "Adicionar Alimentos" representado pela imagem 8. Para localizar os alimentos, é apresentada uma caixa de pesquisa por nome, juntamente com um grupo de opções selecionáveis para aplicação de filtros. Os alimentos são exibidos em formato de lista, e aqueles que possuem valor exibem uma *tag* "Possui Valor", enquanto os que não têm não apresentam nenhuma informação adicional.

Após a escolha do alimento, o usuário é direcionado para a tela de adição, na qual são exibidas informações cruciais, tais como o tamanho da porção a ser escolhida, a quantidade de porções, o valor estimado, e um quadro informativo sobre os macronutrientes, como carboidratos, gorduras, proteínas e calorias. O campo referente ao tamanho da porção é selecionável, oferecendo diversas opções. A quantidade deve ser informada por um valor inteiro, e o valor estimado permanece inalterável, bloqueado contra alterações por parte do usuário.

A terceira tela também é uma tela de adição de alimentos, porém, é exibida somente quando o usuário deseja realizar alguma troca de alimentos dentro de sua dieta. As informações apresentadas são as mesmas da tela de adição normal, contudo, também são exibidas as informações do alimento que está sendo substituído.

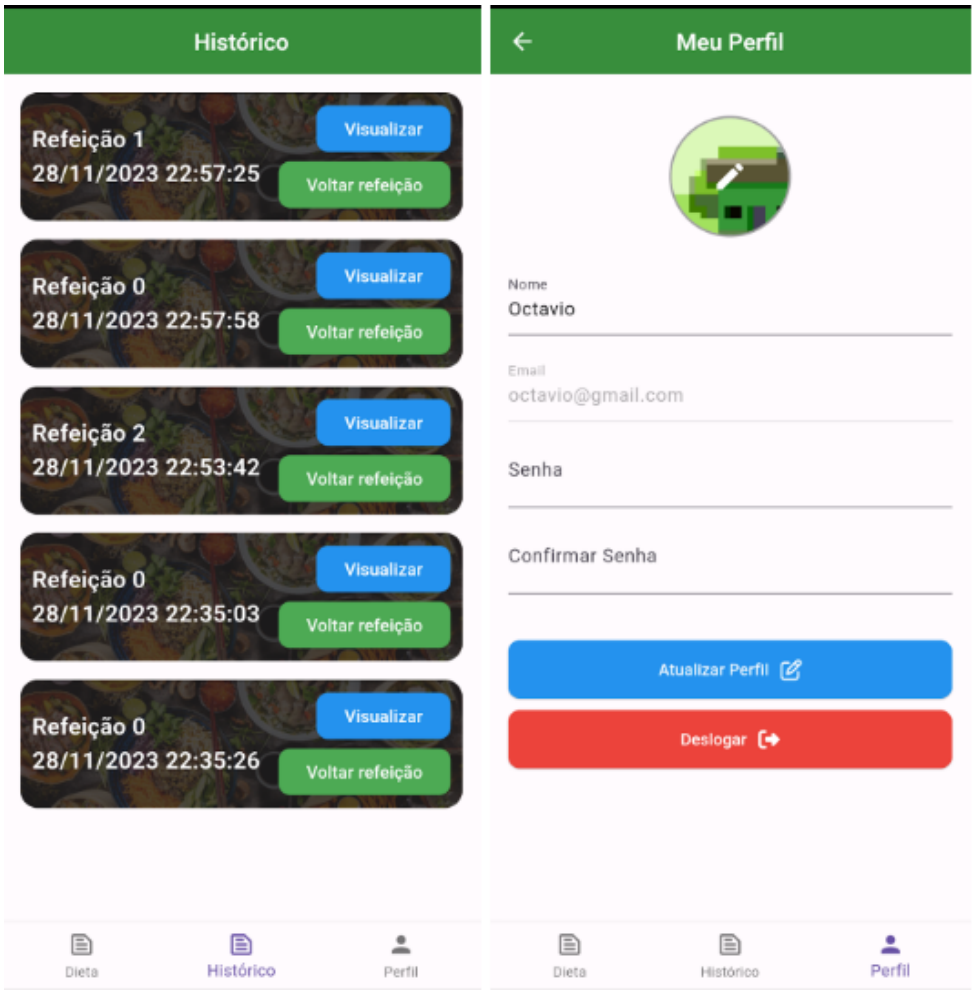
Figura 10 – Pop-ups



Fonte: elaborada pelos autores.

A primeira imagem é um *pop-up* de seleção de horário, que surge assim que o usuário seleciona o ícone de relógio dentro da refeição na tela inicial. Já a segunda e terceira imagem representam *pop-ups* que aparecem quando o usuário solicita a exclusão de um alimento ou de uma refeição, proporcionando uma experiência visual clara e intuitiva. A quarta imagem retrata um *snack bar* que aparece na tela quando o usuário salva sua dieta.

Figura 11 – Histórico e Perfil



Fonte: elaborada pelos autores.

A imagem ilustrada na Figura 11, representa a tela de histórico, onde são registradas alterações feitas pelo usuário em sua dieta. Em cada *card* do histórico, é possível visualizar o nome da refeição, data e hora da modificação, um botões de visualização dos alimentos que fazem parte da dieta, e também um botão de voltar a refeição. O botão "Voltar Refeição" que, quando acionado, permite substituir a refeição selecionada pela de mesma posição na dieta atual. Essa funcionalidade proporciona ao usuário uma maneira eficaz de revisitar e ajustar as mudanças realizadas em sua alimentação.

4.3 Validação

Como objetivo de validação das funcionalidades do aplicativo, foi feita uma visita técnica ao SuperNosso. Após prescrever uma dieta utilizando o aplicativo, com os alimentos presentes no estabelecimento, foi realizado um comparativo de macronutrientes totais, e de preço total. Para uma validação de usabilidade, foram utilizados três planos alimentares reais. Esses planos foram reproduzidos no aplicativo, e foram feitos os comparativos de quantidade calórica

e macronutrientes.

Para o primeiro teste de plano alimentar, foi usada a dieta de uma mulher de cinquenta e cinco anos, que faz uma dieta que prevê a ingestão de 1700 kcal. Esse plano alimentar não tinha especificações a respeito dos macronutrientes, apenas do valor calórico. Essa dieta continha cinco refeições e na aplicação totalizou em 1729.74 kcal, contendo 193.21g de carboidratos, 49.36g de gordura e 138g de proteínas. Para este caso, a margem de erro no valor total de calorias foi de 1.75%.

Para o segundo teste, foi utilizado um plano alimentar de um jovem de vinte dois anos que previa a ingestão de 264g de carboidratos, 200g de proteínas, 73.6g de gorduras e por fim 2018.4 kcal. O resultado obtido com o uso do aplicativo, foi de 187.20g de carboidratos, 190.87g de proteínas, e por fim 1926.60kcal. Para este caso, a margem de erro no valor total de calorias foi de 4.55%.

O terceiro teste utilizou o plano alimentar de um jovem de vite e um anos que previa a ingestão de 296g de carboidratos, 156g de proteínas, 43g de gorduras e 2101 kcal. O resultado obtido com o aplicativo foi de 286.46g de carboidratos, 167.70g de proteínas, 46.40g de gorduras, totalizando 2186.20 kcal. Para esse caso, a margem de erro no valor total de calorias foi de 4.06%.

O aplicativo teve uma margem de erro variando entre 1.75 - 4.55% considerando o valor total de calorias da dieta do indivíduo.

5 CONSIDERAÇÕES FINAIS

Neste trabalho foi desenvolvido um aplicativo para dispositivos *mobile* com o objetivo de simplificar o processo de planejamento, economia e aderência a uma dieta. Com isso, percebeu-se a necessidade de dar maior autonomia ao usuário na hora de seguir um plano alimentar. Para a criação da aplicação, buscou-se um conjunto de tecnologias que atendessem as necessidades de desenvolvimento, como o *framework* de desenvolvimento *mobile* Flutter, o ambiente de execução Pyhton, e o Firebase como plataforma de gerenciamento do aplicativo, responsável pelo armazenamento dos dados. Ao fim, foram validadas as funcionalidades e a interface e usabilidade dos usuários. Para a validação das funcionalidades técnicas do aplicativo, uma visita ao SuperNosso foi realizada, onde foram realizados testes com o aplicativo. Para a validação de interface e usabilidade de usuários, foi pedido que os usuários utilizassem a aplicação, e ao final, foram realizadas perguntas sobre toda a jornada dentro do aplicativo.

Desafios foram enfrentados durante o desenvolvimento do projeto, especialmente na busca por atender de maneira efetiva às necessidades reais dos usuários, garantindo uma interface intuitiva. Enfrentamos, também, obstáculos significativos durante a coleta de dados do site do SuperNosso. No entanto, por meio do aprimoramento contínuo do conhecimento e das técnicas de desenvolvimento, conseguimos superar esses desafios de maneira eficaz.

O desenvolvimento constante é sempre importante para o âmbito da tecnologia. Visando

isto, como sugestão de próximas contribuições para o trabalho, pode ser realizado a implementação de recomendação de alimentos mais similares ao que o usuário deseja realizar a troca, a expansão da base de dados com a adição de novos supermercados, implementação de aviso do horário das refeições por meio de notificações, recomendação de receitas prontas e gerar uma lista de compras de forma semanal ou mensal.

Referências

CHATTERJEE, Nilanjan et al. Real-time communication application based on android using google firebase. **IJARCSMS**, v. 6, 04 2018.

COUTO, Flávio M. do; SILVA, Julio F. da. Uso do módulo python uncertainties no cálculo de incertezas experimentais da diferença de potencial e corrente elétrica de um protótipo experimental. Campus de Alegre, Alegre, ES, BR, 2021. Disponvel em: https://www.scielo.br/j/rbef/a/zPhrQhWFVzGfcmVwmbkv8bs. Acesso em: 16 de out. 2023.

FENTAW, Awel Eshetu. Cross platform mobile application development: a comparison study of react native vs flutter. 2020.

(FORC), Universidade de São Paulo (USP). Food Research Center. **Tabela Brasileira de Composição de Alimentos (TBCA)**. 2023. <http://www.fcf.usp.br/tbca>. Acessado em: 28/09/2023.

FOUNDATION, Python Software. **Documentação Python**. [S.l.], 2023. Acesso em: 18 de out. 2023. Disponível em: <https://docs.python.org/pt-br/3/>.

HUSSAIN, Naveed et al. Prus: Sistema de recomendação de produtos baseado em especificações de usuários e avaliações de clientes. 2016. Disponvel em: https://ieeexplore.ieee.org/document/10196427. Acesso em: 28 de nov. 2023.

LI, Wu-Jeng et al. Justiot internet of things based on the firebase real-time database. 2018. Disponível em: https://ieeexplore.ieee.org/abstract/document/8353979. Acesso em: 04 de set. 2023.

LINARDON, Jake et al. Interactions between different eating patterns on recurrent binge eating behavior: A machine learning approach. 2020.

MAGNO, Fernanda Cristina Carvalho Mattos et al. **Macro e micronutrientes na orientação nutricional para obesidade**. 44. ed. Juiz de Fora, MG, BR: HU Revista, 2018. 251-259 p.

MOREIRA, Rafaela Priscila Cruz; MARTINS, Flávio Vinícius Cruzeiro; WANNER, Elizabeth Fialho. Cardnutri: Um software de planejamento de cardápios nutricionais semanais para alimentação escolar aplicando inteligência artificial. Rio de Janeiro, RJ, BR, 2017. Disponvel em: https://www.reciis.icict.fiocruz.br/index.php/reciis/article/view/1272. Acesso em: 24 de nov. 2022.

NOTE, W3C Working Group. **Web Services Architecture**. [S.l.], 2004. Acesso em: 10 nov. 2023. Disponível em: <https://www.w3.org/TR/ws-arch/>.

PEREIRA, Caíque de Paula; COSTA, Ruyther Parente da. **Algoritmo de Recomendação de Presentes em Dispositivos Móveis**. 2016 — Universidade de Brasília - UnB, acesso em: 18 de out. 2023. Disponível em: <https://fga.unb.br/articles/0001/6858/tcc-gifter-caique-ruyther.pdf>.

RISSI, Matheus; DALLILO, Felipe Diniz. Flutter um framework para desenvolvimento mobile. 2022. Disponvel em: https://recima21.com.br/index.php/recima21/article/view/2230. Acesso em: 04 de set. 2023.

SCHWABER, Ken; SUTHERLAND, Jeff. **The 2020 Scrum Guide**. [S.l.], 2020. Acesso em: 10 de nov. 2023. Disponível em: <https://scrumguides.org/scrum-guide.html\#acknowledgements>.

SHAH, Kewal; SINHA, Harsh; MISHRA, Payal. Analysis of cross-platform mobile app development tools. In: IEEE. **2019 IEEE 5th International Conference for Convergence in Technology (I2CT)**. [S.l.], 2019. p. 1–7.

SILVA, Valéria Martins da. Revisão sistemática da evolução mvc na base acm. 2012. Disponível em: https://41jaiio.sadio.org.ar/sites/default/files/31$_E ST_2 012.pdf. Acessoem : 04deset.2023.$

WANGLER, Julian; JANSKY, Michael. The use of health apps in primary care–results from a survey amongst general practitioners in germany. **Wien Med Wochenschr**, v. 171, n. 7-8, p. 148–156, May 2021. Epub 2021 Feb 11.

WU, Wenhao. React native vs flutter, cross-platforms mobile application frameworks. Metropolia Ammattikorkeakoulu, 2018.