UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

DEPARTAMENTO DE COMPUTAÇÃO

LariFood - Uma rede social móvel para recomendação de receitas

Trabalho de Conclusão de Curso

Renato Vinícius Gomes Campos

São Cristóvão – Sergipe

2022

UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

DEPARTAMENTO DE COMPUTAÇÃO

Renato Vinícius Gomes Campos

LariFood - Uma rede social móvel para recomendação de receitas

Trabalho de Conclusão de Curso submetido ao Departamento de Computação da Universidade Federal de Sergipe como requisito parcial para a obtenção do título de Bacharel em Engenharia da Computação.

Orientador(a): Thiago Dias Bispo
Coorientador(a): Leonardo Nogueira Matos

São Cristóvão – Sergipe

2022

Agradecimentos

Agradeço a todos os meus familiares, em especial ao meu pai e a minha mãe que sempre me deram todo o suporte e nunca mediram esforços para eu ter a melhor educação possível.

Quero agradecer também aos meus amigos que me ajudaram a pensar em vários temas para o presente trabalho e também à minha namorada por todo o suporte durante essa trajetória.

Por fim agradeço também ao meu orientador Thiago e ao meu coorientador Leonardo pelas orientações, conhecimentos repassados e pelas sugestões dadas ao longo da construção deste trabalho.

Toda a vida
O dia inteiro
Não seria exagero
Se depender de mim
Eu vou até o fim
(Humberto Gessinger)

Resumo

As redes sociais são bastante utilizadas no cotidiano e entregam uma grande quantidade de informação para o usuário. Entretanto, nem sempre o que é entregue ao usuário é relevante para ele. Dessa forma, o LariFood é um aplicativo para dispositivos Android, criado para servir como ponto de interação entre usuários que possuem interesse comum por receitas. O aplicativo tem como função armazenar receitas inseridas pelos usuários. Servindo assim como um livro de receitas virtual. Também pode-se publicar receitas para que os usuários possam interagir, seja comentando ou curtindo as mesmas. Uma pesquisa de mercado foi realizada para que fosse possível conhecer as soluções existentes e realizar o levantamento de requisitos. Ao fim do processo de desenvolvimento foi possível conceber uma aplicação que além de permitir o cadastro de receitas e interações entre usuários, também empregasse técnicas de Inteligência Artificial e possui diferencias inovadores que o distingue dos presentes no estado da arte e estado da técnica. O presente trabalho foi testado com a ferramenta Insomnia e com uso da metodologia TDD.

Palavras-chave: Aplicativo. Rede Social. Receitas. Inteligência Artificial.

Abstract

Social networks are widely used in everyday life and deliver a large amount of information to the user. However, not always what is delivered to the user is relevant to him. Thus, LariFood is an application for Android devices, created to serve as a point of interaction between users who have a common interest in recipes. The application's function is to store recipes entered by users. Thus serving as a virtual cookbook. You can also publish recipes so that users can interact, either by commenting or liking them. A market research was carried out so that it was possible to know the existing solutions and carry out the requirements survey. At the end of the development process, it was possible to achieve an application that, in addition to allowing the registration of recipes and completely between users, also employed Artificial Intelligence techniques and has innovative differentials that distinguish it from those present in the state of the art and state of the art. This work was tested with the Insomnia tool and using the TDD methodology.

Keywords: Application. Social Media. Recipe. Artificial Intelligence.

Lista de ilustrações

Lista de quadros

Lista de tabelas

Lista de abreviaturas e siglas

MVC	*Model-View-Controller*
MVP	*Minimum Viable Product*
UFS	Universidade Federal de Sergipe
API	*Application Programming Interface*
BoW	*Bag of Words*
ORM	*Object Relational Mapping*
TDD	*Test Driven Development*
JSON	*JavaScript Object Notation*
UML	*Unified Modeling Language*
KNN	*K-nearest neighbors*

Sumário

1

Introdução

Não há dúvidas sobre o fenômeno que são as redes sociais nos dias de hoje. Elas não somente são um meio de comunicação entre as pessoas, mas também se tornaram importantes ferramentas de negócio no qual empresas se comunicam com seus clientes e fazem diversos anúncios nessas redes (SIQUEIRA, 2021). Vários são os temas discutidos pelos usuários nessas redes e o tipo do assunto depende muito de qual categoria a rede social se encaixa. Os temas, tais como notícias, músicas, futebol e até mesmo receitas se fazem presentes nessas redes. Foi pensando no último tema listado, o de receitas, que esse presente trabalho se debruça.

Antes da pandemia da Covid-19, o tema das receitas já era bastante famoso na internet, principalmente em plataformas como Youtube (COSTA, 2021). Entretanto, após o começo da pandemia da Covid-19, ganhou ampla popularidade o fenômeno de vídeos curtos de até um minuto. Também se tornou mais comum que as pessoas façam mais receitas caseiras já que as mesmas tinham que estar em casa por conta das restrições de circulação da Covid-19. Na concomitância desses dois fatores várias redes sociais implementaram funcionalidades voltadas para esse público em específico. Entretanto, como mostra Siqueira (2021), *Instagram* e *TikTok* são, por exemplo, redes de propósito geral e, portanto, sem foco um público específico.

Além disso, com o advento das redes sociais um fenômeno que se tornou bastante presente o é da sobrecarga de informação que representa o comportamento do consumidor sob a influência de um número de informações maior do que este pode processar (JACOBY DONALD E. SPELLER, 1974). Dessa forma, para resolver tal problemática, surgiram os sistemas de recomendação que podem ser definidos como sistemas que procuram auxiliar indivíduos a identificarem conteúdos de interesse em um conjunto de opções que poderiam caracterizar uma sobrecarga (CAZELLA, 2022). Esses sistemas fazem parte do que é conhecido como Inteligência Artificial (IA). IA refere-se a sistemas ou máquinas que mimetizam a inteligência humana para executar tarefas e podem se aprimorar iterativamente com base nas informações que eles coletam (ORACLE, 2022).

Dessa forma, firma-se a necessidade e utilidade de ferramentas voltadas para o público interessado em fazer ou ter acesso a receitas caseiras e que gostariam de compartilhá-las com outras pessoas. Tais ferramentas também seriam úteis mesmo para pessoas que desejam ter seu próprio livro de receitas virtual, mesmo sem intersse de compartilhamento. É neste contexto que surge o **LariFood**.

Um dos diferenciais do LariFood é o uso de inteligência artificial como forma de recomendar receitas para os usuários facilitando assim, a descoberta de novas receitas. Também há a possibilidade de buscar receitas por ingredientes, definindo quais ingredientes o usuário quer que a receita contenha.

1.1 Objetivos

1.1.1 Objetivo geral

Desenvolver um aplicativo móvel de rede social voltado para o registro, compartilhamento e indicação de receitas virtuais.

1.1.2 Objetivos específicos

- Desenvolver a API modelada;
- Testar a aplicação com a metodologia *Test Driven Development* (TDD);
- Publicar a API na internet tornando-a acessível pela interface *mobile*;
- Desenvolver a interface mobile para consumir a API modelada.

1.2 Metodologia

Para alcançar os objetivos acima estimados, foi necessário que uma sequência de etapas fosse seguida. Estas etapas têm seu conteúdo detalhado nas subseções abaixo:

- Revisão das ferramentas e técnicas necessárias para o desenvolvimento da API;
- Levantamento de requisitos da API;
- Implementação do projeto.

1.3 Estrutura do documento

Este documento está estruturado nos seguintes capítulos, são eles:

- Capítulo 1 - Introdução: Corresponde ao capítulo atual, que apresenta os objetivos deste trabalho;
- Capítulo 2 - Fundamentação Teórica: Apresenta conceitos importantes para o entendimento do trabalho;
- Capítulo 3 - Metodologia: Explica a metodologia utilizada e quais foram os métodos e materiais empregados;
- Capítulo 4 - Trabalhos Relacionados: Cita os trabalhos, principalmente aplicativos, que possuem objetivos semelhantes ao deste documento, explicando seu funcionamento e suas características;
- Capítulo 5 - Requisitos: Disserta sobre os requisitos funcionais e não-funcionais da aplicação;
- Capítulo 6 - Arquitetura: Disserta sobre o conceito de arquitetura e qual arquitetura foi escolhida para o desenvolvimento do projeto;
- Capítulo 7 - Considerações Finais: É a conclusão da monografia, retomando alguns tópicos das seções anteriores e apresenta sugestões para trabalhos futuros.

2
Fundamentação Téorica

Este capítulo apresenta a definição de conceitos básicos necessários para o entendimento deste trabalho. Apresenta também técnicas que serão utilizadas para a implementação de alguns algoritmos. Além disso, também aborda algumas ferramentas que serão estudadas, seja para o desenvolvimento dos algoritmos, como também para o desenvolvimento da aplicação em si.

2.1 Rede Social

Como define Siqueira (2021), redes sociais são estruturas formadas dentro ou fora da internet, por pessoas e organizações que se conectam a partir de interesses ou valores comuns. Além de haver interação entre os usuários, outro característica central em uma rede social é compartilhamento de infomações. Uma rede social que se destaca neste cenário é o Linkedin onde um usuário pode se conectar com outro e mandar mensagens, mas também pode compartilhar conquistas como a conclusão de um curso ou outros tipos de realização. No caso do presente trabalho, as duas características apresentadas anteriormente, se fazem presentes: a interação entre os usuários e o compartilhamento de informações.

Por fim, como define Siqueira (2021), existem diversos tipos de redes sociais. A Tabela 1 mostra os tipos e suas respectivas definições. Como pode-se observar, o aplicativo desenvolvido se enquadra no tipo de rede social de nicho já o que é direcionado a um público específico.

2.2 Sistemas de Recomendação

Um fenômeno recente é o denominado como Sobrecarga de Informação, que se caracteriza pela explosão informacional ao trazer a disposição das pessoas, por meio de vários canais, um fluxo de mensagens e informações de forma rápida, instantânea e em grande quantidade (SIQUEIRA, 2006). Em outras palavras, trata-se do fenômeno no qual um indivíduo ou grupo é exposto a

Tabela 1 – Tipos de redes socais.

Tipos	**Definição**
Rede social de relacionamento	Foco total no quesito. Ex: *Facebook* e *Instagram*.
Rede social de entretenimento	Foco no consumo de conteúdo e não no relacionamento direto com as pessoas. Ex: *Youtube* e *Pinterest*.
Rede social profissional	Foco no relacionamento profissional, ou seja, divulgação de conquistas profissionais, apresentação currículo ou indicação de vagas. Ex: *Linkedin*.
Rede social de nicho	São redes voltadas a um público específico ou que possuem algum interesse em comum . Ex: *TripAdvisor*.

Fonte: Elaborado pelo autor.

uma quantidade de informações maior do que consegue assimilar. Ou seja, uma quantidade de informações que simplesmente não pode ser processada completamente por uma ou mais pessoas (KAPTIVA, 2022). Com isso, surgiu uma dificuldade de consumo de conteúdos relevantes, a menos que a abordagem *word of mouth* (SHARDANAND, 1995) fosse usada. Com ela, uma pessoa recebe a recomendação de um conteúdo diretamente de outra, supostamente com gostos parecidos com o seu.

A *word of mouth* pode se mostrar um pouco falha no contexto mais virtual, por conta disso se fez necessário algumas ferramentas que automatizassem essas recomendações. Surgiram, então, os sistemas de recomendação, que de acordo com Takahashi (2015), podem ser definidos como técnicas de aprendizado de máquina que filtram um grande conjunto de dados, tendo como base informações dos usuários e itens. Dessa forma, o sistema consegue compreender o comportamento do usuário e realizar recomendações relevantes de conteúdos novos.

Do ponto de vista de um usuário de rede social é fácil perceber o uso de sistemas de recomendação no dia a dia, como por exemplo no caso da *Amazon* que é referência em se tratando de serviços de internet e possui um sistema de recomendação de produtos muito poderoso, considerado um dos mais efetivos (TAKAHASHI, 2015). Já na Netflix cada filme visto e avaliado pelo usuário é utilizado como base para as próximas recomendações e isso faz com que 75% a 80% do que é assistido no site provenha das recomendações ao invés da busca (TAKAHASHI, 2015). Por esses motivos, percebe-se que, seja no site da *Amazon* ou da *Netflix*, há um grande volume de informações e os sistemas de recomendação das duas plataformas atuam entregando para o usuário um conteúdo relevante, reduzindo assim, a sobrecarga de informação.

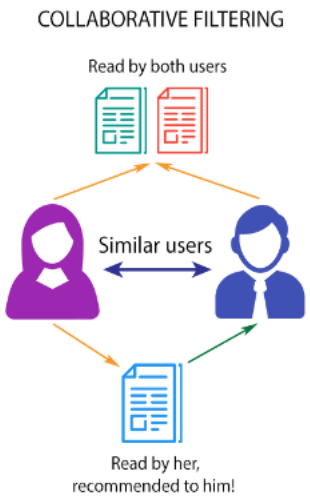
Como vimos, há varios exemplos de sistemas de recomendação: em um site de compra quando ao clicar em um determinado produto ele te recomenda produtos parecidos com o escolhido, em um site de *streaming* onde ao terminar de assistir a um filme o site recomenda ao usuário outros filmes para assistir, um site de viagens que recomenda pacotes de viagens com maior chance do usuário comprar, entre outros. De acordo com Takahashi (2015) em um problema de recomendação, as principais entidades são o usuário e o item e o objetivo é recomendar os itens mais provavelmente ao usuário. Ainda conforme Takahashi (2015) os

sistemas de recomendação são um tipo de Sistemas de Aprendizagem Computacional e fazem uso de algoritmos que são divididos em 3 grupos: filtragem colaborativa, filtragem baseada em conteúdo e filtragem híbrida. A seguir, descreveremos cada uma dessas abordagens.

2.2.1 Filtragem Colaborativa

Nessa abordagem, no caso de um *e-commerce* por meio dos itens já comprados ou interagidos pelo usuário A, procura por usuários semelhantes que tiveram um comportamento igual ou similar (interagiram com os mesmos items) e seleciona itens com os quais o usuário A ainda não interagiu (TAKAHASHI, 2015). Uma representação dessa situação pode ser vista na Figura 1 que mostra que dois usuários, um homem e uma mulher, são similares pois ambos leram os mesmos livros e, por serem similares, o sistema recomenda para o homem, um livro que a mulher leu.

Figura 1 – Filtragem baseada em usuários.



Fonte: (SANTANA, 2018)

Portanto, nessa abordagem, o foco é determinar a similaridade entre usuários. Para isso, uma abordagem bastante utilizada consiste na construção de uma matriz conforme a representada na Figura 2 na qual as linhas representam os usuários e as colunas os itens com os quais eles interagiram. Os itens são jogos e onde há o número um significa que o usuário daquela linha gostou do jogo representado na coluna em questão. E onde não há número algum, significa que o usuário não gostou.

Entretanto, Matos (2021) cita algumas desvantagens desta abordagem que precisam ser levadas em conta:

- **Esparsidade dos dados**: Acontece quando a maioria dos itens nunca foram avaliados ou

Figura 2 – Usuários x Itens.

	Jogo A	Jogo B	Jogo C	Jogo D	Jogo E
Usuário 1		1	1		
Usuário 2	1	1		1	
Usuário 3		1		1	
Usuário 4		1	1	1	1
Usuário 5	1	1		1	
Usuário 6			1		1

Fonte: (SANTANA, 2018)

com os quais poucos usuários interagiram, fazendo com que a linha ou coluna da matriz tenha muitos valores zerados;

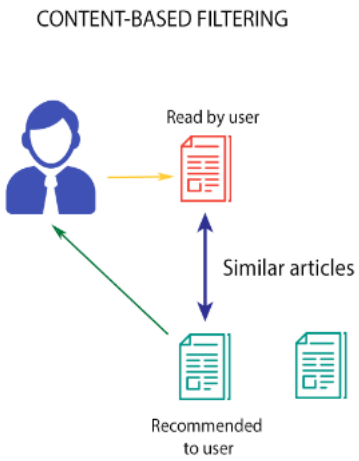
- **Escalabilidade**: À medida que o número de usuários e itens cresce, maior será o esforço computacional necessário para processar essa quantidade de dados;
- **Partida a frio**: Uma vez que estes algoritmos dependem de informações de avaliação, é impossível recomendar itens que ainda não foram avaliados pelos usuários, ou dar recomendações a usuários que não forneceram qualquer informação de preferência usando a abordagem de filtragem colaborativa.

2.2.2 Filtragem baseada em Conteúdo

Ao contrário da filtragem colaborativa, a filtragem baseada em conteúdo tem como partida a utilização exclusiva das informações dos itens como por exemplo, no caso de filmes: o diretor, os atores, o ano de lançamento, gênero e até, possivelmente, imagens do filme. A abordagem parte do princípio de que: se um usuário possuiu uma opinião sobre um item com certas características no passado, então ele terá uma opinião similar sobre um outro item com características similares no futuro (MATOS, 2021). O comportamento anterior está ilustrado na Figura 3 onde um livro foi lido por determinado usuário e o sistema recomendou para ele um outro livro similar ao livro já lido.

Dessa maneira, fica evidente que o foco dessa abordagem não é mais a similaridade entre usuários como na abordagem anterior, mas sim a similaridade entre itens, o que se torna um desafio ainda maior, pois antes, para determinar a similaridade entre usuários poderia-se usar as avaliações feitas pelos usuários, sejam elas, avaliações explícitas (uma nota atribuída) ou implícitas (tempo visto em um filme ou cliques em uma notícia). Já na abordagem atual o cálculo da similaridade entre itens será feito com base em atributos desses registros, o que faz com que a escolha desses atributos seja muito importante e muitas vezes bastante difícil de ser feita.

Figura 3 – Filtragem baseada em conteúdo.



Fonte: (SANTANA, 2018)

Assim como na abordagem anterior, Matos (2021) também cita algumas desvantagens que têm que ser levadas em conta:

- **Redundância**: Como o modelo só faz recomendações com base nos interesses de um único usuário, ele acaba tendo uma capacidade limitada de expandir os seus interesses já existentes, podendo sofrer de recomendações demasiadamente tendenciosas;
- **Escolha de atributos**: Torna a implementação do modelo mais difícil de ser feita, pois há muitos contextos onde os atributos do itens não são suficientes para definir uma similaridade de fato. Por exemplo: se a entidade a ser analisada for um filme, atributos como categoria, duração, atores e atrizes, ano de lançamento são muito fracos e não definem, com uma boa precisão, tal entidade.

2.2.3 Filtragem Híbrida

Como há vantangens e desvantagens nas duas abordagens uma solução muito utilizada atualmente é a chamada Filtragem Hibrída que tem o intuito de complementar as limitações de cada uma destas abordagens, evitando os problemas inerentes de cada sistema (MATOS, 2021). Existem várias abordagens para deixar o sistema híbrido. A mais simples é unificar as recomendações geradas por sistemas separados em uma única lista (SANTANA, 2018).

2.3 Medidas de similaridade

Como visto na seção anterior indepedente da abordagem escolhida sempre há a necessidade de saber a similiridade seja entre itens ou entre usuários. Como mostra Sousa (2018) há algumas

técnicas de utilizadas pela comunidade, algumas das quais são citadas a seguir.

2.3.1 Distância Manhattan

É a métrica de similaridade mais fácil e mais rápida para ser calculada. A Distância Manhattan entre dois vetores, **p** e **q** , é a diferença absoluta do comprimento dos segmentos que ligam esses vetores a suas origens, respectivamente.

$$D = \sum_{n}^{i=1} \left|(p_i - p_j)\right| \tag{2.1}$$

2.3.2 Distância Euclidiana

Seja dois vetores **p** e **q** , quanto mais a distância entre o comprimento do segmento que liga pq se aproxima de 0, mais similares são os vetores.

$$D = \sqrt{\sum_{n}^{i=1} (p_i - p_j)^2} \tag{2.2}$$

2.3.3 Distância de Minkowski

É uma generalização das distâncias de Manhattan e a Euclidiana.

$$D = (\sum_{n}^{i=1} \left|p_i - p_j\right|^r)^{1/r} \tag{2.3}$$

Onde se r = 1, a fórmula fica igual a distância de Manhattan e se r = 2, fica igual a distância Euclidiana.

2.3.4 Coeficiente de Correlação de Pearson

É uma medida usada para corrigir um problema que ocorre quando os usuários avaliam itens de maneiras muito diferentes. Essa métrica varia de -1 até 1, onde valores maiores que zero indicam correlação positiva entre os usuários, valores negativos indicam uma correlação negativa e valores nulos indicam ausência de correlação.

$$r = \frac{\sum(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{(\sum(x_i - \overline{x})^2)(\sum(y_i - \overline{y})^2)}} \tag{2.4}$$

2.3.5 Distância do cosseno

Técnica empregada para medir o ângulo entre dois vetores. Quando o ângulo se aproxima de 0, o cosseno se aproxima de 1 e assim podemos dizer que os dois vetores são similares. É uma técnica muita boa de ser usada para o caso em que há dados esparsos (SALAZAR, 2012).

$$D = cos(\theta) = \frac{< X, A >}{||X||.||A||} \tag{2.5}$$

2.3.6 Critérios para a escolha da medida de similaridade

De forma resumida pode-se dizer que se houver o problema chamado de grau de inflação, onde usuários diferentes podem estar usando escalas diferentes, então usa-se *Pearson*. Podemos visualizar esse problema, por exemplo, na avaliação de um filme, na qual a nota varia de 1 a 10, e um usuário A dá nota 10 para um filme considerado bom e 8 para um considerado ruim, enquanto que outro usuário B dá nota 8 para um filme que ele curtiu e 2 para um que não curtiu. Sendo assim, para o primeiro usuário as notas variam entre 8 e 10, enquanto que para o segundo, variam entre 2 e 8, ou seja, estão usando escaladas diferentes para o mesmo tipo de avaliação.

Caso os dados sejam esparsos como por exemplo, na abordagem de filtragem colaborativa, então usa-se *Similaridade do Cosseno*. Agora se os dados forem densos como será por exemplo, o caso de similaridades entre receitas descrito mais a frente, então usar a distância Euclidiana ou de Manhattan teria uma execução mais rápida.

2.4 Processamento de Linguagem Natural

De acordo com Pereira (2011) o Processamento de Linguagem Natural (PLN) consiste no desenvolvimento de modelos computacionais para a realização de tarefas que dependem de informações expressas em uma língua natural. Uma linguagem natural seria por exemplo: português, espanhol, inglês, etc. A área de PLN é muito importante nos dias atuais pela sua alta aplicabilidade: tradução e interpretação de textos, busca de informações em documentos, interface homem-máquina (chatterbots), etc.

Antes mesmo da aplicação de técnicas de PLN, uma das etapas é o pré-processamento do texto. Essa etapa é importante pois consiste em limpar ruídos presentes no texto como por exemplo: pontuações, texto maiúsculo ou minúsculo, *stop-words*, etc (AGRAWAL, 2021). Tal etapa de pré-processamento é importante, pois quando a similaridade das receitas for calculada haverá comparações entre ingredientes e como os nomes dos ingredientes serão inseridos pelos usuários, faz-se necessário haver uma etapa de padronização para que por exemplo 'Açúcar' seja igual a 'acucar' no momento da comparação. As etapas de pré-processamento aplicadas no presente trabalho foram escolhidas de acordo com Agrawal (2021) e são exibidas abaixo:

- ***Lowercase***: Essa etapa consiste em deixar toda a sentença com letras minúscula. Exemplo: 'ACUCAR' para 'acucar';
- **Remover pontuações**: Essa etapa consiste em retirar todas as pontuações existentes na língua portuguesa. Para isso foi criado um vetor com as pontuações mais comumente usadas e assim esse vetor foi percorrido verificando para cada pontuação a sua ocorrência na sentença. Exemplo: 'ACUCAR.' para 'ACUCAR';
- ***Stop Word***: *Stop words* são palavras comuns de aparecerem em uma sentença, mas que normalmente não apresentam nenhuma informação semântica relevante. Um exemplo seria a palavra 'de' em 'Farinha de trigo', que se reduziria a somente 'Farinha trigo'.
- **Acentos**: Essa etapa consiste em retirar todos os acentos em caracteres. Exemplo: 'AÇÚCAR' para 'ACUCAR';
- **Stemming**: *Stemming* é o processo de reduzir a palavra a sua raiz, retirando o sufixo ou prefixo. Exemplo: ' manteiga ' para 'manteig';
- **Trim**: *Trim* consiste em retirar os espaços em branco mais a direita ou a esquerda da palavra é. Exemplo: 'manteiga ' para 'manteiga'.

3
Metodologia

Este capítulo apresenta a definição de conceitos básicos necessários para o entendimento deste trabalho. Apresenta também técnicas que serão utilizadas para a implementação de alguns algoritmos. Além disso, também aborda algumas ferramentas que serão estudadas, seja para o desenvolvimento dos algoritmos, como também para o desenvolvimento da aplicação em si.

3.1 Métodos

Nessa seção será explicado quais foram os métodos utilizados para o desenvolvimento do projeto.

3.1.1 Revisão das ferramentas e técnicas necessárias para o desenvolvimento da API

Algumas técnicas envolvendo a área de Inteligência Artificial (IA) tiveram que ser estudas, como por exemplo: similaridade entre objetos, algoritmos de recomendação, classificação de imagens, etc. Também foi feita uma pesquisa de mercado para entender quais recursos que os aplicativos concorrentes utilizam. Além disso houve a necessidade de ser estudado o *framework* AdonisJS que será utilizado para fazer o *back-end*[1] da aplicação. Por fim também foi estudado o *framework* Flutter para fazer a interface *mobile*. Ambos os *frameworks* serão apresentados na Seção 2.5.

3.1.2 Levantamento de requisitos da API

Uma vez obtido o conhecimento técnico das ferramentas necessárias para a realização do projeto, foi iniciada a etapa que envolve a elicitação de requisitos. Essa parte é muito importante

[1] Parte da aplicação responsável pela regra de negócio e por interagir com o banco de dados.

pois define quais as principais funcionalidades aplicação terá e assim serve como guia para todo o desenvolvimento do projeto.

3.1.3 Implementação do projeto

A última etapa é o desenvolvimento em si do projeto, começando pelo *back-end*, que de acordo com, Machado (2021), envolve servidor, banco de dados e aplicação. Começar pelo *back-end* é importante pois ele será o responsável pela lógica de negócio de toda a aplicação, desde a inserção das receitas no sistema, até mesmo por executar o algoritmo de recomendação. Uma vez o *back-end* concluído, é o momento do *front-end* começar a ser construído, também conhecido como "o lado do cliente", o *front-end* é o responsável por toda a estrutura, design, conteúdo, comportamento, desempenho e capacidade de resposta de um site ou aplicação, ou seja, tudo o que é apresentado aos usuários para interação (MACHADO, 2021).

3.2 Ferramentas e técnicas relacionadas

Nesta seção serão explicadas quais foram as ferramentas utilizadas como por exemplo: *frameworks* e metodologias. Também abordará alguns conceitos importantes, como o da API.

3.2.1 API

Uma Application Programming Interface (API), conforme Hat (2020), é um conjunto de definições e protocolos usado no desenvolvimento e na integração de aplicações. Às vezes, as APIs são descritas como um contrato entre um provedor e um usuário de informações, estabelecendo o conteúdo exigido pelo consumidor (a chamada) e o conteúdo exigido pelo produtor (a resposta).

Em termos práticos uma API é muito utilizada nos dias atuais para realizar a integração entre dois sistemas. Um exemplo é quando *e-commerce* é acessado e no momento do cadastro do endereço, ao inserir o CEP, todos os outros campos relacionados a endereço são automaticamente preenchidos. Isso acontece pois o sistema do *e-commerce* acessa a API dos Correios realizando uma requisição Hypertext Transfer Protocol (HTTP) e recebe como resposta os dados do endereço.

Outro exemplo é no desenvolvimento de aplicativos móveis, onde geralmente há a divisão entre duas partes. Uma delas, chamada *front-end* responsável por ser a interface gráfica vista pelo usuário, ou seja, as telas, os botões, os textos, etc. Já a outra parte, chamada de *back-end* é resposável por interagir diretamente com o banco de dados e é no desenvolvimento dessa parte que é comum de se fazer uma API, pois assim, o *front-end* da aplicação poderá interagir com o *back-end* por meio de uma API.

A Representational State Transfer (REST) é uma arquitetura de software que impõe condições sobre como uma API deve funcionar. A REST foi criada inicialmente como uma

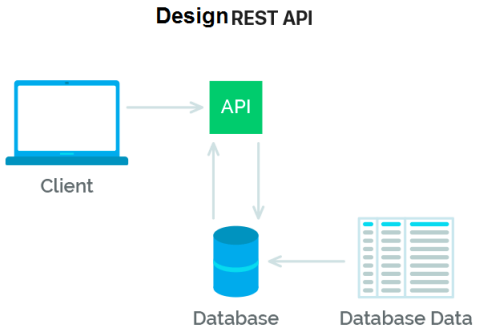
diretriz para gerenciar a comunicação em uma rede complexa como a internet (AWS, 2022). Dentre essas condições, de acordo com Neto (2022), podemos citar:

- **Cliente-Servidor**: Trata a respeito da separação de responsabilidades, ou seja, separar as preocupações de interface do usuário (User Interface) do banco de dados, abstraindo a dependência entre os lados clientes/servidor e permitindo a evolução desses componentes sem impacto e quebra de contrato.
- **Interface Uniforme**: É a interoperabilidade entre os componentes cliente e servidor. Para isso, há quatro princípios a serem seguidos: Identificação dos recursos, Representação dos recursos, Mensagens auto-descritivas e Componente HATEOAS.
- **Stateless**: Cada requisição acionada entre a comunicação cliente-servidor deve possuir toda a informação necessária e compreensível para realizar a origem da requisição, não sendo de responsabilidade do servidor armazenar qualquer tipo de contexto.
- **Cache**: É utilizado para melhorar a performance de comunicação entre aplicações, otimizando o tempo de resposta na comunicação entre cliente-servidor.
- **Camadas**: A separação de responsabilidades é importante nesse modelo de arquitetura. Os princípios e as boas práticas na arquitetura e design de um projeto, sugerem a construção de camadas independentes e auto gerenciadas, em que cada camada não pode conhecer as demais camadas. Caso ocorra mudanças em uma delas, as demais não serão impactadas.

Uma API que implementa a arquitetura REST é conhecida por ser RESTFful. Conforme AWS (2022) para serviços REST, o servidor normalmente realiza a identificação de recursos usando um uniform resource locator (URL). O URL especifica o caminho para o recurso. Além disso no estilo arquitetural REST, a manipulação dos recursos disponibilizados para o cliente é realizada através de métodos do protocolo HTTP (NETO, 2022). Os métodos HTTP indicam os diferentes tipos de operações que o cliente pode realizar para manipular os dados através de requisições para cada contrato oferecido. Os métodos mais utilizados estão descritos abaixo:

- **GET**: Os clientes usam GET para acessar recursos localizados no URL especificado no servidor;
- **POST**: Os clientes usam POST para enviar dados ao servidor. Eles incluem a representação de dados com a solicitação;
- **PUT**: Os clientes usam PUT para atualizar recursos existentes no servidor;
- **DELETE**: Os clientes usam a solicitação DELETE para remover o recurso. Uma solicitação DELETE pode alterar o estado do servidor. No entanto, se o usuário não tiver a autenticação apropriada, a solicitação falhará.

Figura 4 – Representação de uma API REST.



Fonte: (COSTA, 2022)

Como mostra a Figura 4 uma API é utilizada para fazer a interligação entre dois sistemas, no caso da Figura, está ligando um cliente a um banco de dados, sendo possível que mais clientes acessem a mesma API para se comunicar com o banco.

3.2.2 AdonisJS

Na programação, um *framework* é um conjunto de códigos genéricos capaz de unir trechos de um projeto de desenvolvimento (ZUCHER, 2022). O *framework* a ser usado para fazer a API do projeto será o *AdonisJS* construído tendo como base o NodeJS. Há alguns motivos relevantes para o *AdonisJS* ter sido escolhido para o presente trabalho:

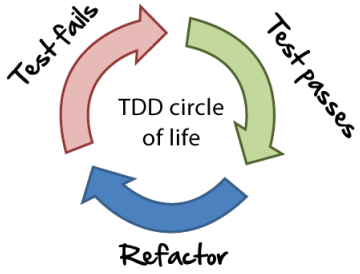
- Usa o padrão MVC (Model-View-Controller);
- Possui sistema de autenticação já pronto para uso;
- Possui um ORM (Object Relational Mapping) chamado Lucid. Ter um ORM é importante pois facilita a interação do desenvolvedor com o banco de dados, sem ter que escrever consultas Structured Query Language (SQL) puras;
- Possui uma boa documentação;
- Empiricamente percebeu-se que ele é de fácil aprendizado.

3.2.3 TDD

Como define Techlise (2022), TDD é a sigla para Test Driven Development, que em português significa Desenvolvimento Orientado por Testes. Esse é um método de desenvolvimento muito comum atualmente. Como mostra Alexandre (2022), resumidamente o TDD se baseia

em pequenos ciclos de repetições, e para cada funcionalidade do sistema um teste é criado antes. Este novo teste criado inicialmente falha, já que ainda não temos a implementação da funcionalidade em questão e, em seguida, implementamos a funcionalidade para fazer o teste passar. Uma imagem que representa bem a metodologia é a Figura 5.

Figura 5 – Metodologia TDD.



Fonte: (COSTA, 2022)

3.2.4 Flutter

Flutter é um *framework* utilizado para desenvolver aplicativos multiplataforma *mobile*, ou seja, ele é responsável por definir toda a interface *mobile* de um aplicativo e o código feito gera uma aplicação tanto para Android como para Ios, diferente de soluções nativas que geram um aplicativo para suas respectivas plataformas. É um *framework* relativamente novo, mas já bastante adotado pela comunidade e até mesmo por empresas como *Nubank* e *Ifood*. Algumas das características do *framework* que o fizeram ganhar destaque são:

- É de fácil aprendizado quando comparado com soluções nativas;
- É multiplataforma, ou seja, o mesmo código executa tanto no ambiente Android, quanto IOS;
- Possui uma excelente perfomance (FEDYK, 2022);
- É bastante estável nas suas atualizações (FEDYK, 2022).

3.2.5 Sendiblue

A Sendinblue é uma ferramenta para a gestão de estratégias de email marketing. Com diversas funcionalidades e planos, a solução surge como uma opção valiosa para quem deseja criar uma relação ainda mais próxima com o seu público (CAMARGO, 2020). Por conta disso, o Sendiblue foi escolhido para servir como servidor *Simple Mail Transfer Protocol* (STMP) da aplicação.

4

Trabalhos Relacionados

O objetivo deste capítulo é apresentar aplicações que possuem algumas semelhanças com o objetivo do trabalho proposto neste documento, não somente no quesito de cadastro receitas, mas também pela existência de características de rede social. Tal busca é importante para descobrir o que já existe no mercado, visando não somente entender os pontos fortes e fracos das aplicações, mas também o que pode ser adicionado ao produto viável mínimo (MVP) que o faça se destacar entre os demais.

4.1 Metologia de busca

A busca foi feita na loja de aplicativos da Google Play por dois motivos: O primeiro é que ela possui uma grande quantidade de aplicações para o sistema Android, e o segundo seria a facilidade de acesso a essa base, já que um dispositivo Android é suficiente. Também buscou-se registros em base de patentes e artigos científicos relacionados a certos temas convenientes ao aplicativo proposto.

4.1.1 Critérios de seleção

Com a finalidade de filtrar os resultados obtidos nas bases de pesquisa, foram formulados alguns critérios de triagem, levando em conta a proposta deste trabalho. Os critérios adotados foram:

- **Critério 01** - A aplicação deve possuir uma plataforma mobile;
- **Critério 02** - A aplicação também funciona no território brasileiro;
- **Critério 03** - A aplicação foi publicada nos últimos 10 anos.

As strings de busca foram aplicadas na base de patentes WIPO e na Play Store constam na Quadro 1.

Quadro 1 – Bases de dados e strings de busca aplicados

Base de dados	**Strings de busca**
WIPO	recipe AND social media; algorithms AND suggestion; algorithms AND recommendation AND social media; algorithms AND recommendation AND recipe
Scopus	recipe AND social media; algorithms AND suggestion; algorithms AND recommendation AND social media; algorithms AND recommendation AND recipe
Google Play Store	rede social receita

Fonte: Elaborado pelo autor

É bom ressaltar que em algumas strings de busca foi colocada a palavra "algortithms" pois o presente trabalho pretende fazer uso de certas classes de algoritmos como os de recomendação e algoritmos também de classificação de imagem.

4.1.2 Critérios de análise

Fizemos a busca nas bases de dados e os resultados das bases de patentes tiveram seus resumos e títulos lidos, enquanto que os resultados da Google Play Store tiveram as descrições lidas e as imagens visualizadas. Nessa fase de análise dos resultados, levamos em conta os critérios de análise a seguir:

- **Critério 1** - A aplicação deve permitir o cadastro de receitas;
- **Critério 2** - A aplicação também deve ter funcionalidades relacionadas a uma rede social como monstra a Seção 2.1.

Sobre o critério 1, quando é dito que a aplicação deve permitir o cadastro de receitas entende-se que a aplicação sirva também como um de livro de receitas virtual. Já o critério 2 diz respeito ao fato de que a aplicação também deve ter algumas características de rede social, como por exemplo:

- Usuário seguir e ser seguido por outro usuário;
- Usuário poder visualizar receitas postadas por outros usuários;
- Usuário poder curtir ou comentar outras receitas.

Com o uso dos critérios de análise reduziu-se a quantidade de resultados finais, conforme apresentado na Tabela 2. Com isso, os resultados das bases de patentes não foram relevantes, restando apenas aqueles originados da Google Play Store.

Tabela 2 – Quantidade de resultados da busca antes e depois da filtragem

Base de dados	**Quantidade de resultados na busca**	**Quantidade de resultados após filtragem**
WIPO	64	0
Scopus	45	5
Google Play Store	30	4

Fonte: Autor.

Como podemos ver no Quadro Tabela 2 acima, os resultados na base de patentes WIPO não foram muito relevantes, pois os remanescentes após o filtro dissertam sobre um "sistema de gerenciamento de receitas" o que não tem relação com o objetivo proposto desse trabalho. Quanto na base de artigos científicos Scopus, selecionamos alguns artigos pois tinham relação com algumas classes de algoritmos que foram utilizadas no trabalho.

4.2 Aplicativos relacionados

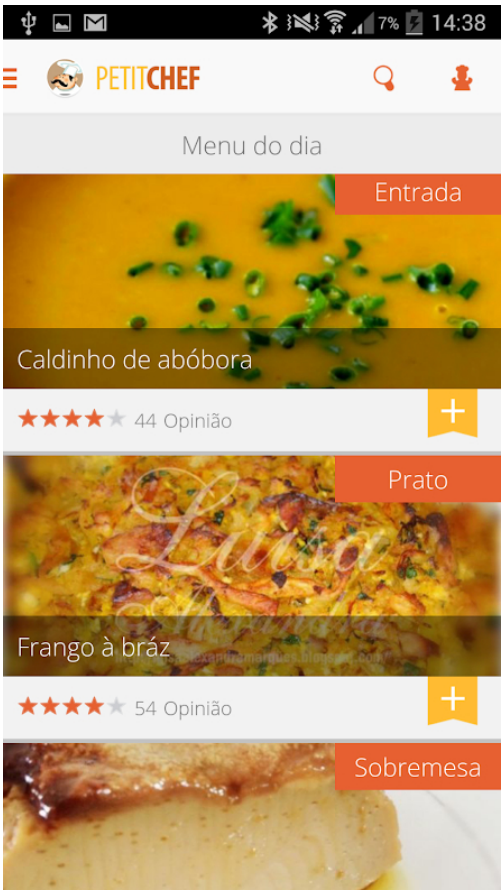
Cada aplicativo selecionado após filtragem foi instalado e testado para verificação direta cada uma das funcionalidades. Criamos uma conta e fizemos o cadastro de receitas. Além disso também foram realizadas várias interações com os outros usuários afim de descobrir recursos relacionados ao conceito de rede social.

4.2.1 Petit Chef

O aplicativo possui uma interface não tão amigável se assemelhando na verdade a interface de um site só que adaptado para celular. Apesar disso permite o cadastro de receitas e a interação com receitas de outros usuários como por exemplo: Votar, comentar e favoritar.

Outro ponto positivo é que ele possui uma filtro de pesquisa muito avançado permitindo assim que a pesquisa seja feita fazendo o uso de vários filtros como por exemplo: grau de de dificuldade da receita, tempo de preparado, etc. Por fim, um ponto negativo é a quantidade exagerada de anúncios ao fazer uso do aplicativo diminuindo, assim, a qualidade da experiência do usuário.

Figura 6 – PetitChef.



Fonte: (PLAY, 2022)

4.2.2 Tudo Gostoso

Sem sombras de dúvidas é um dos sites de receitas mais famosos no Brasil (CASAREDO, 2022). A sua versão *mobile* é um pouco mais limitada que o site, mas ainda assim fornece uma gama de funcionalidades interessantes: cadastro de receitas, interação com receitas de outros usuários e busca por receitas. Entretanto assim como o Petit Chef também há um grande volume de anúncios.

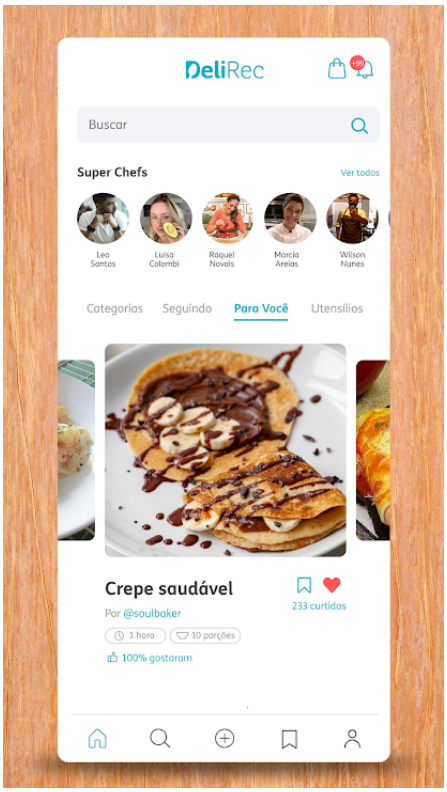
Figura 7 – Tudo Gostoso.

Fonte: (PLAY, 2022)

4.2.3 DeliRec

Enquanto os aplicativos anteriores versavam mais sobre o aspecto de cadastro de receitas e visualização de receitas de outros usuários, o DeliRec já possui funcionalidades mais relacionadas como uma rede social em si. Por exemplo, no aplicativo o usuário consegue ver o perfil de um usuário, seguir outro usuário. Consegue também visualizar as receitas que deu *like*, visualizar receitas recomendadas, pesquisar por usuários. E possui uma loja própria para a venda de utensílios usados nas receitas. Como ponto negativo pode-se apontar o fato da Tela Inicial ser um pouco complexa prejudicando um pouco a experiência do usuário.

Figura 8 – DeliRec.

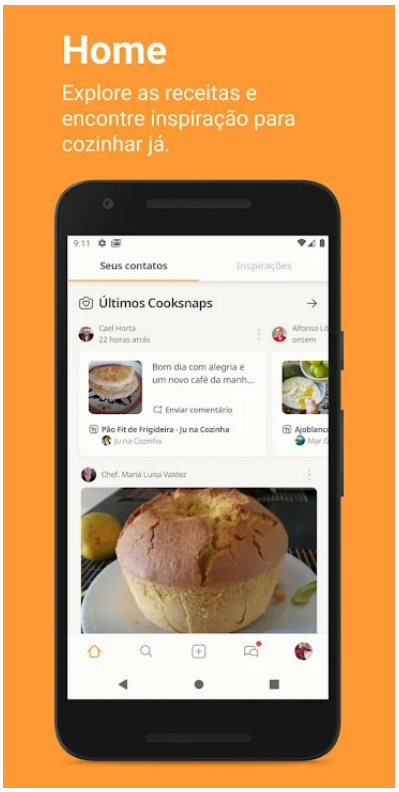


Fonte: (PLAY, 2022)

4.2.4 Cookpad

O Cookpad é muito parecido com o DeliRec em termos de funcionalidades, mas possui alguns diferenciais. Um deles é a seção de dicas, permitindo assim, que o usuário cadastre pequenos trechos de texto dando dicas como: fazer desmoldante caseiro, deixar o bolo mais fofo, etc. Outro ponto é que um usuário pode fazer a receita de outra pessoa e postar o resultado da receita na mesma postagem, servindo assim como forma de motivar outros usuários a também fazer a receita. Faz o uso de tags como forma de classificar as receitas de uma maneira mais versátil.

Figura 9 – Cookpad.



Fonte: (PLAY, 2022)

4.3 Conclusão sobre os aplicativos encontrados

Dessa forma, pode-se perceber que dentre todos os aplicativos analisados e testados, os que mais se aproximaram da proposta do LariFood foram o DeliRec e o Cookpad. A razão para tal reside no fato de ambos aplicativos terem características mais pertinentes ao de uma rede social como por exemplo: interação com outros usuários e interação nas receitas.

O resumo das principais funcionalidades encontradas está presente no Quadro 2 e nas subseções seguintes as funcionalidades são detalhadas para cada aplicativo selecionado.

No Quadro 3 podemos ver o significado cada uma das funcionalidades expostas no Quadro acima

Quadro 2 – Principais funcionalidades encontradas nos produtos

Aplicativo	**Cadastrar receitas**	**Interação nas receitas**	**Busca avançada**	**Recomendação de receita**	**Interação com outros usuários**
Petit Chef	X	X	X	X	-
Tudo Gostoso	X	X	-	-	-
DeliRec	X	X	-	X	X
Cookpad	X	X	-	-	X
LariFood	X	X	X	X	X

Fonte: Autor.

Quadro 3 – Funcionalidades e suas definições

Funcionalidade	**Definição**
Cadastrar Receitas	Permitir o cadastro de receitas por parte do usuário
Interação nas receitas	Permtir visualizar, comentar e curtir outra receita
Busca avançada	Pertmir buscas mais com mais filtros e não apenas pelo nome da receitas
Recomendação de receita	O sistema recomendar receitar com base em dados do usuários
Interação com outros usuários	Seguir e visualizar as receitas de outros usuários

Fonte: Autor.

5

Desenvolvimento

Este capítulo tem o propósito de apresentar o processo de construção da API, passando pelo levantamento de requisitos, pela descrição das funcionalidades no *back-end* e *front-end* e pelos testes realizados.

5.1 Levantamento de requisitos

Conforme definido em Veríssimo (2022), o levantamento de requisitos é umas das partes mais importantes do processo que resultará no desenvolvimento de um sistema. Entender aquilo que o cliente deseja ou o que o cliente acredita que precisa e as regras do negócio ou processos do negócio. Isso é o cerne que move essa importante função que faz parte da engenharia de requisitos. Os subtópicos a seguir apresentam os requisitos funcionais e não funcionais.

5.1.1 Requisitos Funcionais

Os requisitos funcionais representam as funcionalidades da aplicação, aquilo que ela se propõe a solucionar, serviços que oferece ou funções que o software deve desempenhar. Os requisitos presentes no Quadro 4 foram agrupados em funcionalidades, dispostas da seguinte forma:

- **Funcionalidades de autenticação**: RF1 até o RF5;
- **Funcionalidades de receitas**: RF6 até o RF19;
- **Funcionalidades de outros recursos**: RF20 até o RF23.

Quadro 4 – Requisitos funcionais do sistema

Identificação	**Título**	**Descrição**
RF1	Cadastro de usuário	O usuário não autenticado cadastra conta.
RF2	Login	O usuário não autenticado entra com uma conta.
RF3	Redefinição de senha	O usuário não autenticado redefine a senha de sua conta.
RF4	Alteração de senha	O usuário altera a senha de sua conta.
RF5	Logout	O usuário sai da sua conta.
RF6	Visualização de receitas	O usuário visualiza receitas.
RF7	Pesquisa de receitas	O usuário pesquisa receitas.
RF8	Favoritar receita	O usuário pode favoritar uma receita.
RF9	Curtir receita	O sistema permite o usuário curtir uma receita
RF10	Detalhes de receita	O sistema exibe detalhes de uma receita, como ingredientes, modo de prepararo e imagens.
RF11	Comentar receita	O usuário pode comentar uma receita.
RF12	Cadastrar receita	O usuário pode cadastrar uma receita.
RF13	Atualizar receita	O usuário pode atualizar uma receita.
RF14	Pesquisa de contas	O usuário pesquisa outros usuários.
RF15	Receitas Semelhantes	Ao visualizar determinada receita, o sistema exibe receitas semelhantes.
RF16	Busca com filtros	O sistema permite realizar pesquisa usando o nome dos ingredients como filtro.
RF17	Receitas favoritdas	O sistema exibe as receitas favoritadas.
RF18	Modo de visualização da receita	O sistema permite cadastrar uma receita pública ou privada.
RF19	Recomendação de receitas	O sistema possui algoritmo para recomendação de receitas.
RF20	Seguir	O sistema permite um usuário seguir outro usuário.
RF21	Feed	O sistema possui um *feed* para o usuário visualizar atualizações de outros usuários.
RF22	Visualização de perfil	O sistema permite visualizar o próprio perfil e de outros usuários.
RF23	Atualização de perfil	O sistema permite o usuário atualizar o próprio perfil.

Fonte: Elaborado pelo autor.

5.1.2 Requisitos Não Funcionais

Os requisitos não funcionais representam restrições acerca das funcionalidades do sistema, expressando condições ou especificidades que o software deve atender, como os do Quadro 5.

Quadro 5 – Requisitos não funcionais do sistema

Identificação	**Descrição**
RFN1	O sistema deve executar em dispositivos móveis com sistema operacional Android.
RFN2	O sistema utilizará o SGBD MySQL.
RFN3	Toda operação de consulta ou atualização deve ser realizada em 5 segundos para 90% dos casos.
RFN4	A interface mobile do sistema utilizará o framework Flutter.
RFN5	O servidor do sistema utilizará o framework Adonis.js.

Fonte: Elaborado pelo autor.

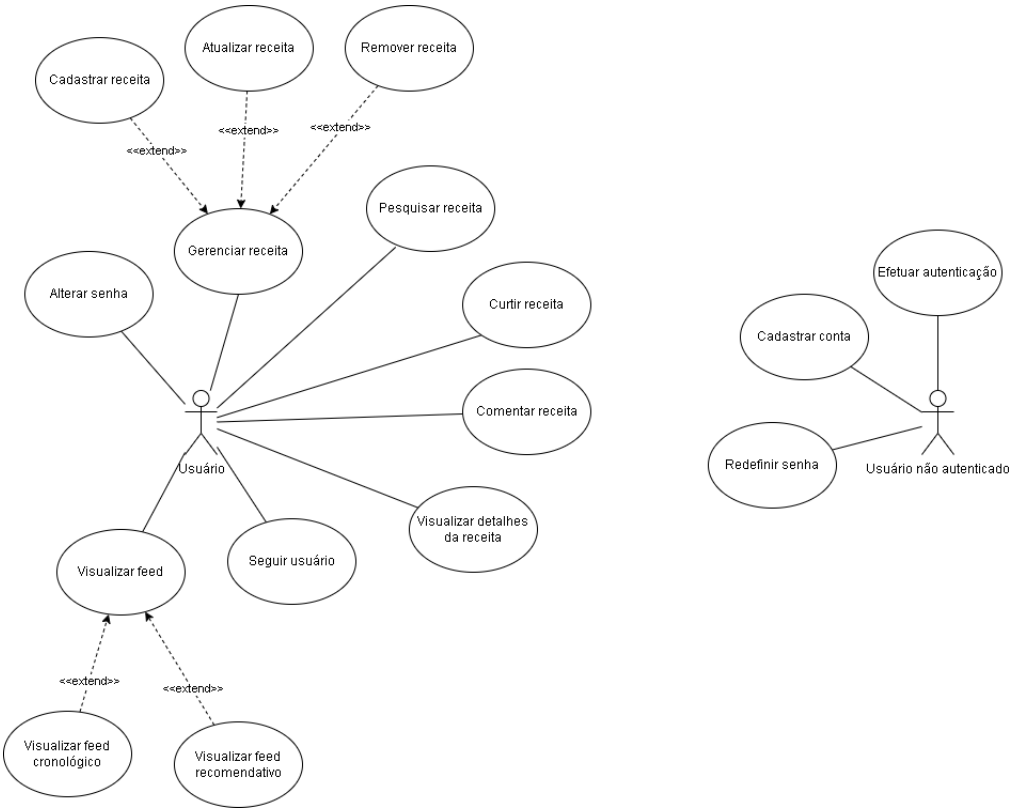
5.2 Casos de uso

Com base nos requisitos funcionais levantados na Seção 4.1 foi possível modelar o diagrama de casos de uso como mostra a Figura 10. Esse diagrama é importante, pois é um tipo de diagrama UML comportamental e frequentemente usado para analisar vários sistemas. Eles permitem que você visualize os diferentes tipos de papéis em um sistema e como essas funções interagem com o sistema (CREATELY, 2021).

5.3 Diagrama Entidade Relacionamento

A partir dos requisitos funcionais descritos nas seções anteriores foi possível traçar as principais entidades e relacionamentos necessários para o desenvolvimento do projeto. Além disso também foram descritos os atributos das entidades como mostra a Figura 11. É bom destacar que algumas das tabelas (adonis_schema, adonis_schema_versions) são tabelas específicas do *framework* utilizado.

Figura 10 – Diagrama de casos de uso do LariFood



Fonte: Elaborado pelo autor.

5.4 Funcionalidades da aplicação

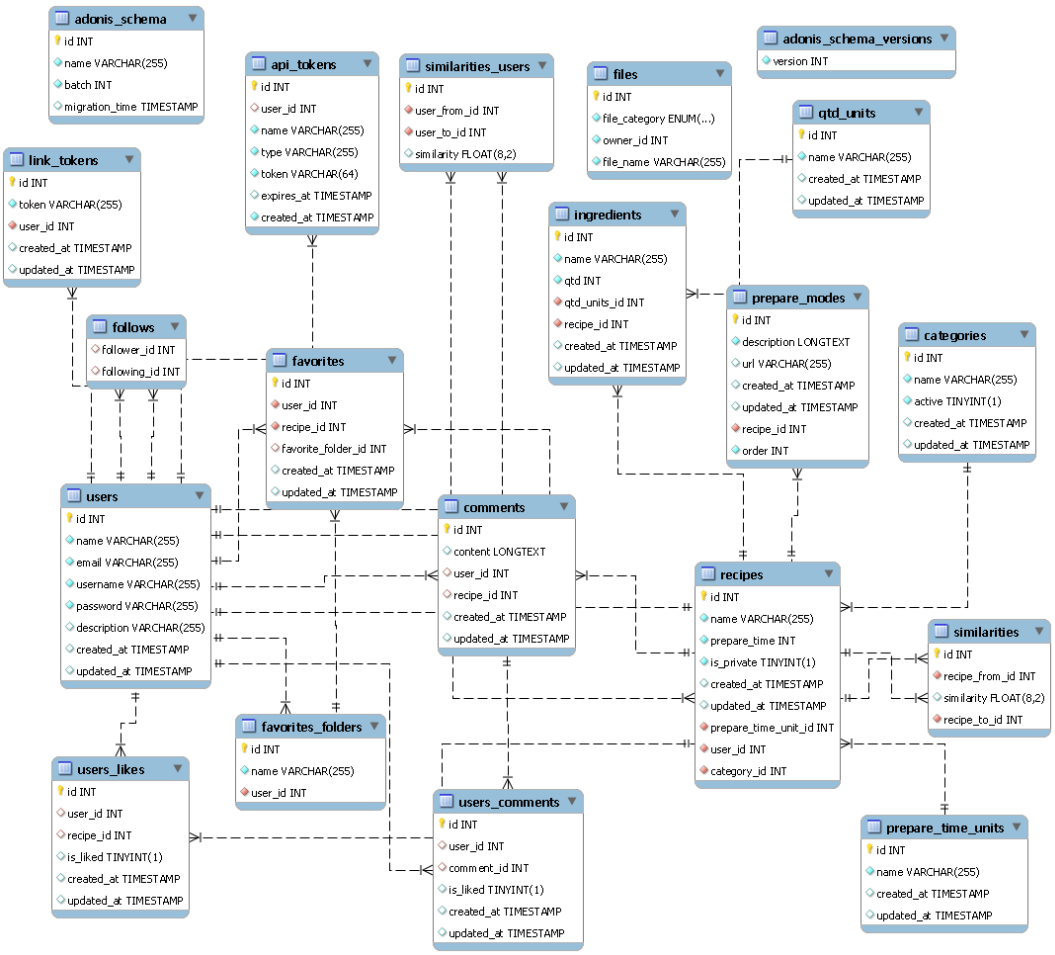
Esta seção trata das principais funcionalides desenvolvidas na aplicação. Cada seção consiste em uma funcionalidade específica onde será explicado: definição, rota, conteúdo das requisições HTTP, *print* das telas e por fim, explicação de cada uma das telas exibidas.

5.4.1 Cadastro e Autenticação

Antes de tudo, o usuário deve criar uma conta na aplicação para que possa usufruir de todas as suas funcionalidades. Após a criação da conta, deve-ser realizar o processo de *login* para que no corpo de resposta seja recebido e guardado um *token* de autenticação. Esse *token* serve para identificar o usuário e validar se determinado usuário tem acesso a determinado recurso da API.

Primeiramente o usuário deve criar uma conta na API (por meio da interface gráfica) realizando uma requisição HTTP POST, informando no corpo da requisição os atributos descritos no Quadro 6. O corpo das requisições ou das respostas serão sempre no formato JSON (JavaScript Object Notation), pois é um formato compacto e simples de troca de dados entre sistemas amplamente utilizado em diversas linguagens e tecnologias (PALERMO, 2022). Como mostra a

Figura 11 – Diagrama entidade relacionamento da API do LariFood



Fonte: Elaborado pelo autor.

Figura 12, o arquivo JSON possui uma estrutura chave-valor, onde a chave é sempre uma *string* e o valor pode ser de vários tipos: *string*, numero, vetor, outro JSON.

Figura 12 – Exemplo de um arquivo JSON.

```json
{
    "nome": "João da Silva",
    "idade": 20,
    "matricula": "2018123490",
    "curso": "Sistemas de Informação",
    "cadeiras": [
        "Estrutura de Dados",
        "Organização de Computadores",
        "Matemática Discreta"
    ]
}
```

Fonte: (PALERMO, 2022)

A partir daí, o serviço recebe os dados e faz algumas validações, como por exemplo: tamanho mínimo de seis caracteres para a senha, verifica se o usuário já existe no banco de dados, etc. Em caso de sucesso, o usuário é criado no banco de dados podendo assim, partir para o processo de *login*.

Quadro 6 – Dados da requisição para cadastro de usuário

URI	**/users**
Método	POST
Corpo da Requisição	nome, email, senha, *username*.

Fonte: Elaborado pelo autor.

No processo de *login* ele deve realizar uma requisição HTTP POST informando no corpo da requisição o *email* e senha como monstra o Quadro 7. Em caso de sucesso, o usuário recebe no corpo da resposta os dados do usuário e um *token* de autenticação. Como dito anteriormente, o *token* deve ser armazenado pelo cliente, ou seja pela aplicação *frontend*, para que nas futuras requisições possa ser enviado no cabeçalho no campo *Authorization*.

Quadro 7 – Dados da requisição para autenticação de usuário

URI	**/sessions**
Método	POST
Corpo da Requisição	email, senha.

Fonte: Elaborado pelo autor.

Como podemos ver na Figura 13 a tela de *login* consiste em um formulário com os campos de e-mail, senha e com o botão Logar. O botão só ficará ativo caso os campos acima sejam preenchidos. Nessa mesma tela também é possível clicar em dois links: um para redefinir a senha e outro para criar conta. O processo de login consiste em verificar no armazenamento local do dispositivo se existe algum *token* guardado e, se existir, é feita a checagem da sua validade, sendo válido, o usuário entra automaticamente na tela inicial da aplicação, caso não seja, ele é redirecionado para a tela de *login*.

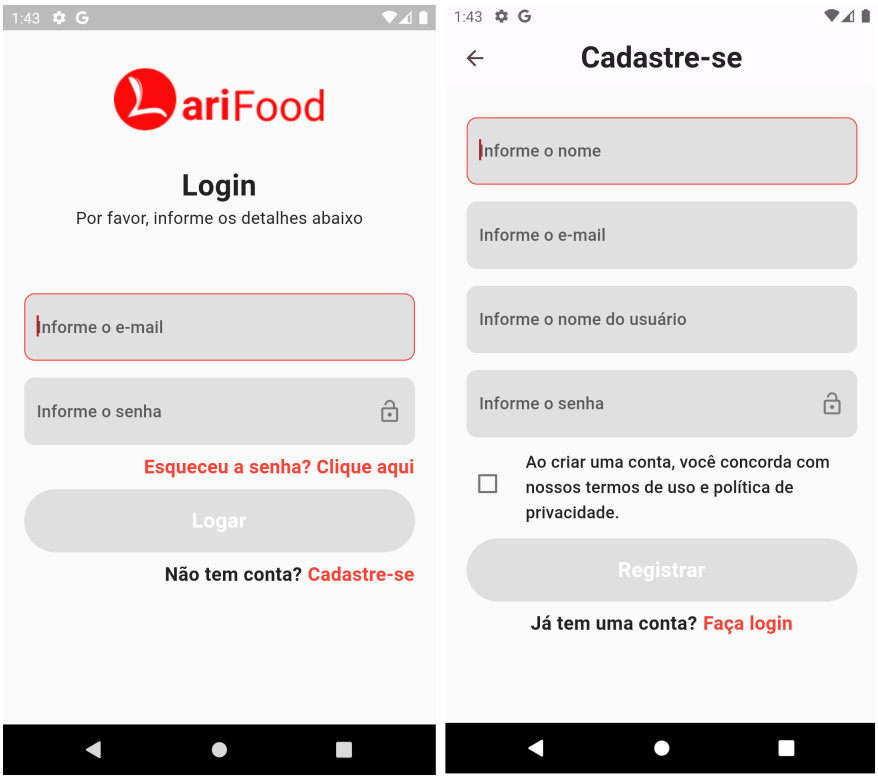
Já na tela de cadastro, o usuário deve informar o nome, e-mail, nome de usuário e senha para que o botão Registrar seja ativado.

5.4.2 Redefinição de Senhas

A operação de redefinição de senha acontece quando o usuário esquece a sua senha e tem que criar uma nova senha para poder se autenticar na aplicação novamente. Para isso o processo consite em receber um email com um *link*, clicar no link e o aplicativo do LariFood será aberto em uma tela específica que permitirá ao usuário alterar sua senha.

O usuário deve então enviar uma requisição HTTP POST, informando em JSON os campos descritos Quadro 8. O campo e-mail se refere ao e-mail do usuário que receberá os

Figura 13 – Tela para login(à esquerda) e criaçaõ de conta (à direita).



Fonte: Elaborado pelo autor.

dados de recuperação. Enquanto que o campo URL consiste de uma URL padrão pré definida no *back-end*. Ao receber esses dados também ocorre validações sendo uma delas a verificação se o campo e-mail de fato foi recebido em um formato válido. Uma vez feitas as validaçõe, um token aleatório com validade de 2 horas é gerado e um e-mail é enviado para o usuário contendo como link na URL o parâmetro token gerado.

O e-mail é enviado por um servidor SMTP chamando Sendiblue bastando para isso definir no serviço o *host*, porta e dados de autenticação. Esse serviço foi escolhido pois no plano gratuito oferece a quantidade de 300 e-mails por dia, o que é mais que suficiente para os propósitos desse MVP. Há outros serviços gratuitos também como o do *Gmail*, mas também possui limitação diária de até 500 e-mails/dia. Como 300 e-mails são suficientes, por isso o *Sendinblue* foi escolhido.

Quadro 8 – Dados da requisição para envio de e-mail de recuperação de senha

URI	**/forgot-password**
Método	POST
Corpo da Requisição	e-mail, url

Fonte: Elaborado pelo autor.

Por fim, ao tentar redefinir a senha, como mostra o Quadro 9 o usuário informará a nova

senha no corpo da requisição. Tendo tambem que enviar o próprio token recebido no e-mail. O envio do token é importante, pois é ele quem identifica o usuário ao qual foi associado o token gerado previamente. Uma vez recebido o corpo da requisição, o serviço faz validações, identifica o usuário associado ao token, verifica se o token está valido e caso esteja atualiza a senha do usuário e deleta o token já usado.

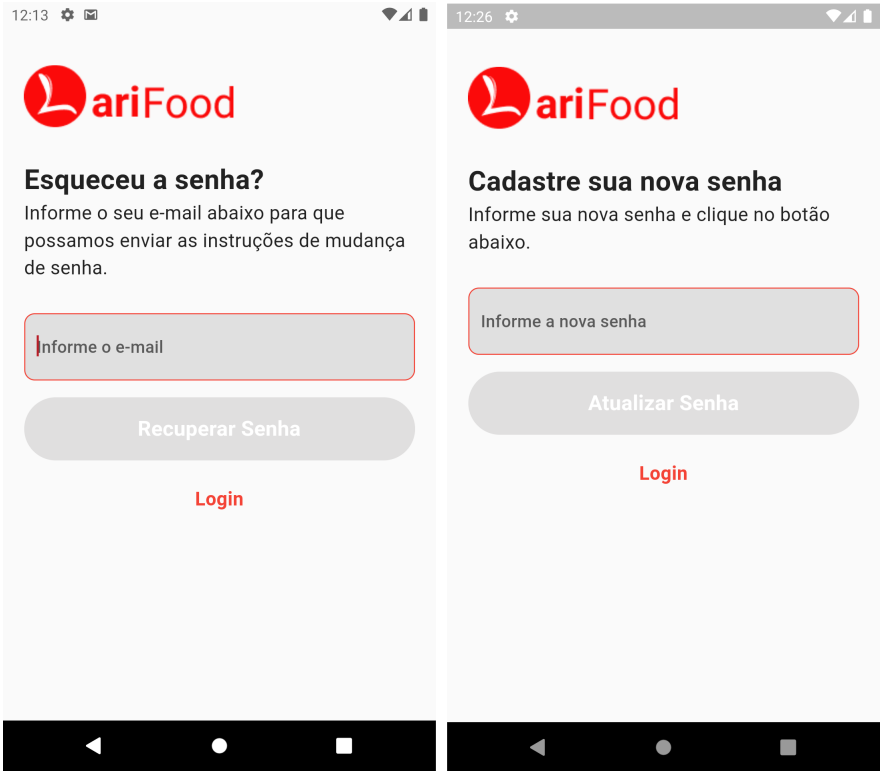
Quadro 9 – Dados da requisição para envio de e-mail de alteração de senha

URI	**/reset-password**
Método	POST
Corpo da Requisição	token, password

Fonte: Elaborado pelo autor.

Como mostra a Figura 14 as telas para as operações acima descritas são bastante simples bastando que o usuário informe o seu e-mail. Ao receber o e-mail, é possível clicar no link 'Reset sua senha' como pode ser visto na Figura 15. Dessa forma o aplicativo abrirá a tela na qual será possível cadastrar uma nova senha.

Figura 14 – Tela para envio de e-mail(à esquerda) e redefinição de senha(à direita).



Fonte: Elaborado pelo autor.

Figura 15 – Exemplo de e-mail recebido pelo usuário.

Fonte: Elaborado pelo autor.

5.4.3 Cadastro de receitas

A operação de cadastrar uma receita é uma das mais importantes de todo o *back-end* pois as receitas são o cerne da aplicação.

Para realizar tal operação, como mostra o Quadro 10, vários campos precisam ser recebidos pelo serviço.

- ***name***: Nome da receita.
- ***isPrivate***: É um campo booleano para determinar se a receita será pública ou privada.
- ***prepareTime***: Campo númerico que define o tempo da receita.
- ***prepareTimeUnitId***: É o *id* referente a qual unidade de medida de tempo será escolhida (min ou h).

- ***categoryId***: É o *id* referente a qual categoria da receita pertence.
- ***prepareModes***: Vetor que irá conter todos os ingredientes da receitas.
- ***ingredients***: Vetor com os diferentes modos de preparo da receita.

Todos esses campos serão validados: o id do usuário tem que de fato existir no backend, a lista de ingiredientes tem que ter no mínimo um elemento, etc. Caso tudo esteja correto na validação os dados receita são retornados em JSON no corpo da resposta.

Quadro 10 – Dados da requisição para cadastro de receita

URI	**/recipes**
Método	POST
Corpo da Requisição	*name, isPrivate, prepareTime, userId, prepareTimeUnitId, categoryId, ingredients, prepareModes*

Fonte: Elaborado pelo autor

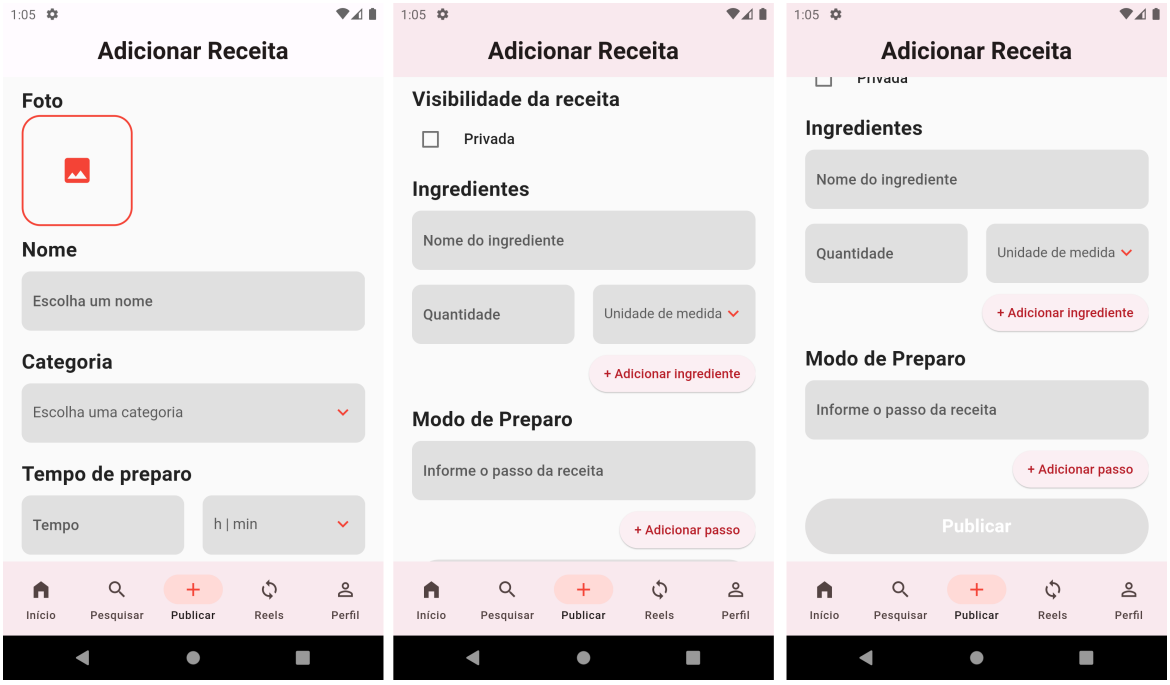
Na aplicação podemos ver que a tela de cadastro de uma receita é bastante completa. Primeiramente o usuário pode anexar uma foto bastando para isso clicar no ícone de foto como mostra a Figura 16. Depois, ele pode cadastrar um nome, categoria, tempo de preparo, escolha a visibilidade da receita, tornando-a pública ou privada. Também é possível cadastrar ingredientes, onde cada um é definido pelo nome, quantidade e unidade de medida. Para adicionar mais ingredientes basta clicar no botão '+Adicionar ingrediente'. Uma vez cadastro todos os dados basta clicar no botão 'Publicar'.

5.4.4 Feed cronológico de receitas

O conceito de Feed consiste em ser uma lista de receitas as quais o usuário pode percorrer e escolher qual delas desejaria visualizar com mais detalhes. Na aplicação há dois tipos de feed: cronológico e recomendativo. O feed da presente subseção é responsável por listas as receitas enviadas por usuários, os quais o usuário logado segue. Assim, essas receitas aparecerão para o usuário logado em ordem cronológica, das mais recentes para as mais antigas.

Para a recuperação do *feed* cronológico não é necessário enviar nenhum dado no corpo da requisição, porém no cabeçalho será enviado o token de autenticação previamente guardado ao realizar o login. Uma vez feita a indentificação pelo serviço de qual usuário fez a requisição, é obtida uma lista com os *ids* dos usuários que ele segue. Por fim basta realizar uma *query* filtrando as receitas as quais o usuário proprietário está presente na lista anteriormente obtida. Também é bom ressaltar que serão obtidas apenas as receitas que foram marcadas como públicas. Por fim, para cada receita também é informado se o usuário já marcou essa receita como favorita ou se já curtiu essa receita.

Figura 16 – Exemplo de cadastro de receita. O fluxo do cadastro começa da esquerda para a direita.



Fonte: Elaborado pelo autor.

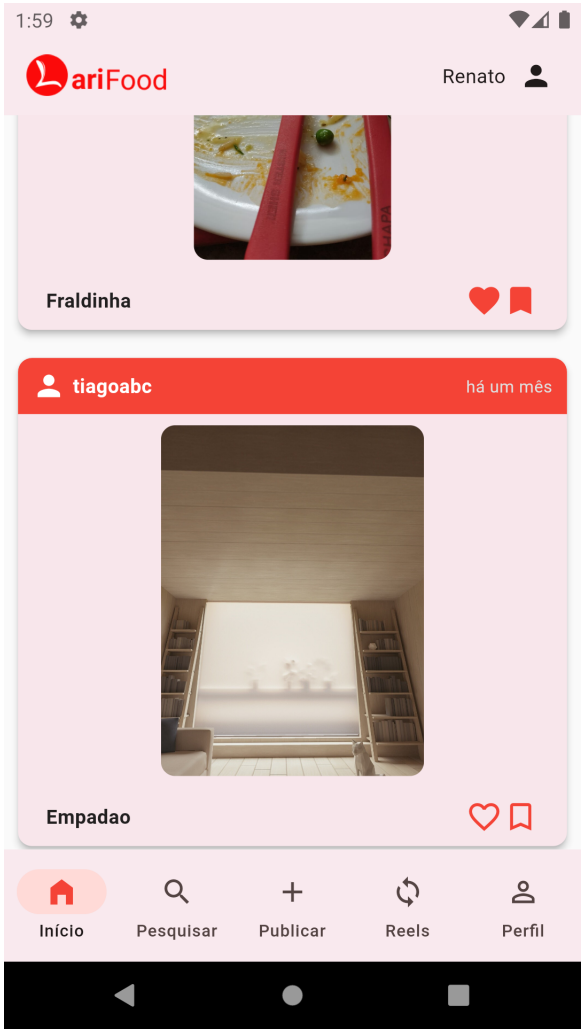
Quadro 11 – Dados da requisição para obter o feed cronológico

URI	**/chronological-feed**
Método	POST
Corpo da Requisição	vazio

Fonte: Elaborado pelo autor

Na tela do aplicativo, pode-se observar que na tela 'Início' há uma lista de receitas publicadas pelos usuários os quais o usuário logado segue. Para cada receita é informado, quem postou, o tempo da postagem, a foto, o titulo e se o usuário logado ja curtiu ou marcou como favorita. Por fim, para cada uma das receitas exibidas o usuário também pode clicar na receita para ver mais detalhes dela.

Figura 17 – Exemplo de feed cronológico



Fonte: Elaborado pelo autor.

5.4.5 Pesquisa

Outra funcionalidade muito importante da aplicação é de busca, que permite buscar por usuários, por receitas (por nome ou por ingredientes). Quando a busca é executada, uma lista de receitas ou usuários é gerada e exibida na tela.

No serviço foram criados 3 *endpoints* sendo o primeiro o que permite buscar por receitas tendo como base o campo nome. Como mostra o Quadro 12 a requisição é do tipo HTTP GET e recebe como parâmetro na URL a *string* de busca. A consulta é feita no banco de dados e retorna as receitas que contenham no campo nome a *string* de busca.

O outro *endpoint* é muito parecido com o primeiro diferenciando apenas pelo fato de que em vez de buscar receitas pelo nome, irá buscar usuários pelo nome como mostra o Quadro 13.

Por fim, há também um recurso que permite que sejam buscadas receitas a partir dos seus

Quadro 12 – Dados da requisição para pesquisar por receita (nome)

URI	**/search-recipe/:searchString**
Método	GET
Corpo da Requisição	vazio

Fonte: Elaborado pelo autor

Quadro 13 – Dados da requisição para pesquisar por usuário

URI	**/search-user/:searchString**
Método	GET
Corpo da Requisição	vazio

Fonte: Elaborado pelo autor

ingredientes. Por exemplo, o usuário pode enviar uma lista de ingredientes ([’ovo’, ’banana’]) e assim uma consulta é feita para trazer receitas que contenham na sua lista de ingredientes os ingredientes obtidos do corpo da requisição.

Quadro 14 – Dados da requisição para pesquisar por receitas (ingredientes)

URI	**/search-recipe**
Método	GET
Corpo da Requisição	*ingredients*

Fonte: Elaborado pelo autor

Na Figura 18 há dois campos, um onde deve ser digitado a *string* de busca e outro que na verdade é um *dropdown* onde deve ser escolhida qual tipo de busca será feita: Usuários, Receitas pelo nome ou Receitas por ingredientes. Como dito anteriormente, uma lista é exibida na tela possibilitando ao usuário escolher qual elemento da lista quer visualizar em detalhes.

5.4.6 Guardar nos favoritos

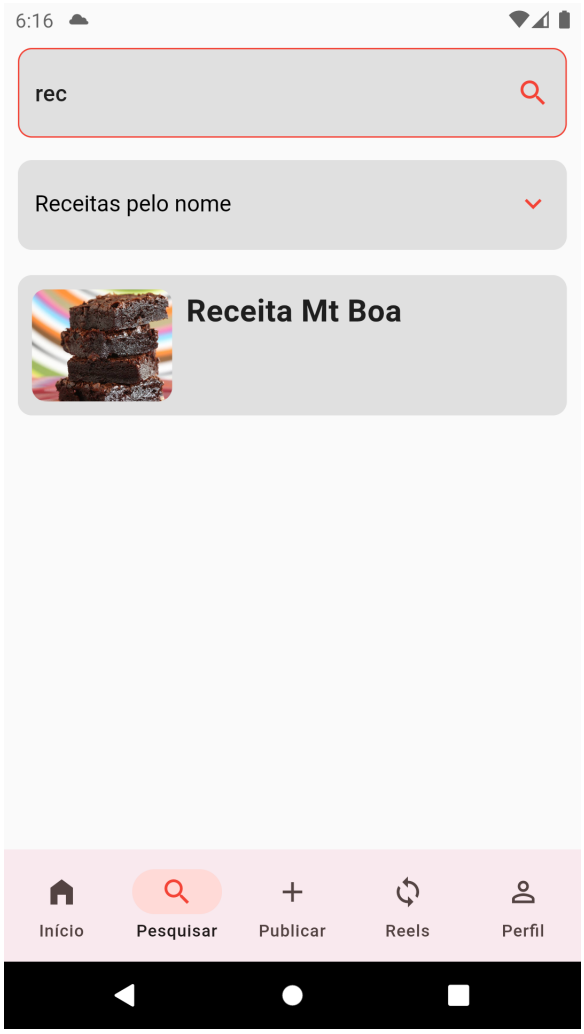
Um recurso muito famoso em várias redes sociais é o que possibilita guardar postagens como favoritas para que possam ser consumidas pelos usuários posteriormente.

O método para guardar uma receita nos favoritos é bastante simples, bastando para isso que o usuário envie como parâmetro na URL o *id* da receita. Com o *id* da receita, basta criar um registro na tabela *favorites* associando o usuário logado com o a receita.

Por fim, outro recurso é o que possibilita o usuário visualizar as receitas que ele guardou nos favoritos, sendo suficiente realizar uma requisição HTTP GET e o método retornará a lista de receitas.

A Figura 19 mostra a tela inicial, onde para cada receita exibida há um ícone que quando clicado permite o usuário pode guardar a receita nos favoritos. Também na Figura 19 mostra a

Figura 18 – Tela de busca



Fonte: Elaborado pelo autor.

Quadro 15 – Dados da requisição para guardar uma receita nos favoritos

URI	**/favorite/:recipeId**
Método	POST
Corpo da Requisição	

Fonte: Elaborado pelo autor

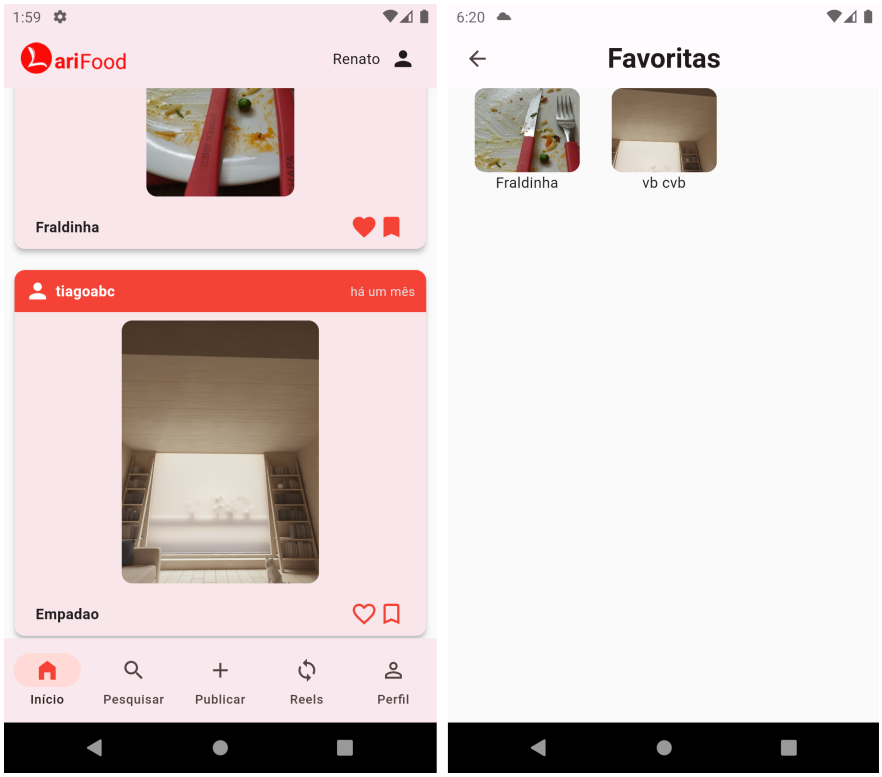
Quadro 16 – Dados da requisição para exibir as receitas guardadas nos favoritos de um usuário

URI	**/favorites**
Método	GET
Corpo da Requisição	

Fonte: Elaborado pelo autor

tela que exibe todos os favoritos que o usuário guardou.

Figura 19 – Tela do feed cronológico exibindo o botão para guardar nos favoritos (à esquerda) e tela mostrando os favoritos (à direita)



Fonte: Elaborado pelo autor.

5.5 Funcionalidades da aplicação - IA

Nesta seção ainda serão apresentadas funcionalidades da aplicação, mas serão funcionalidades relacionadas ao sistema de recomendação desenvolvido. Dessa forma, separamos em duas seções para que seja possível dar maior destaque às funcionalidades da presente seção.

5.5.1 Receitas similares

Essa funcionalidade é utilizada no momento em que uma determinada receita é visualizada e na tela de detalhes há uma seção mostrando para o usuário as 6 receitas mais similares a que está sendo exibida.

5.5.1.1 Similaridade

Para definir a similaridade entre as receitas temos que entender primeiro qual técnica será utilizada. Pode-se definir a similaridade entre duas receitas a partir da semelhança entre os títulos

dessas. Entretanto apesar de ser uma boa técnica, ela falha pra alguns casos, como por exemplo quando uma determinada receita, leva o título de **Empadão de Frango** enquanto outra receita que também é um empadão de frango, mas recebe o título de **Lanche**. Nesse caso o algoritmo não iria identificar similaridade entre as receitas apesar delas serem basicamente "iguais". Portanto, faz-se necessário o uso de outra abordagem, que é a de determinar similaridade entre receitas com base na lista de ingredientes das duas receitas. Dessa forma, a precisão da comparação aumenta bastante dando maior confiabialidade para a medida de similaridade.

A técnica usada aqui consiste em fazer o uso da distância do cosseno como grau de similaridade entre duas receitas, medida bastante usada em abordagens tradicionais de Natural Language Processing (NLP) (LANE COLE HOWARD, 2019). Para isso, como definido em Siqueira (2022) temos a seguinte expressão:

$$Similaridade = cos(\theta) = \frac{< X, A >}{||X||.||A||} \tag{5.1}$$

Como descrito na Seção 2 ideia é que podemos comparar o quão similares dois vetores são com base no ângulo entre eles. Sendo assim, se eles forem iguais o ângulo será 0 e assim o cosseno será igual 1. Já quando forem diferentes, o ângulo será diferente de zero e o cosseno será menor que 1.

Dito isso, precisamos representar as receitas, que são informações textuais, em vetores numéricos, tipo de dado que a distância do cosseno de fato aceita. Existem diversas técnicas para o mapeamento de textos em números. Uma das mais tradicionais e porquanto utilizada neste trabalho é a denominada Bag of Words (BoW). Como explica Fonseca (2022), BoW é uma forma de representar o texto de acordo com a ocorrência das palavras nele. Traduzindo para o português, o “saco de palavras” recebe esse nome porque não leva em conta a ordem ou a estrutura das palavras no texto, apenas se ela aparece ou a frequência com que aparece nele, dependendo da abordagem.

5.5.1.2 Exemplo Prático

Para testar as duas técnicas acima explicadas foi feito um código em Python, onde nele temos duas receitas de *brownie* cujos ingredientes são os exibidos abaixo e separados por vírgula:

$$brownie\_2 = [\text{“}manteiga\text{”}, \text{“}ovos\text{”}, \text{“}achocolatado\text{”}, \text{“}acúcar\text{”}, \text{“}farinhadetrigo\text{”}]$$

$$brownie\_3 = [\text{“}manteiga\text{”}, \text{“}ovos\text{”}, \text{“}chocolat\text{”}, \text{“}acúcar\text{”}, \text{“}farinhadetrigo\text{”}]$$

Podemos ver que são duas receitas bem semelhantes. Primeiramente o algoritmo percorre as duas listas de ingredientes e assim constrói uma outra lista contendo todos ingredientes das duas receitas, mas sem repetição. Ou seja, nesse caso a lista seria:

$$[\text{'}chocolate\text{'}, \text{'}acúcar\text{'}, \text{'}achocolatado\text{'}, \text{'}manteiga\text{'}, \text{'}ovos\text{'}, \text{'}farinhadetrigo\text{'}]$$

Após isso, o algoritmo monta outras duas listas que irão conter a quantidade de ocorrências de cada uma das palavras da lista acima em cada uma das receitas. Por exemplo, para a primeira receita, *brownie_2*, temos o seguinte vetor:

$$\langle 0, 1, 1, 1, 1, 1 \rangle$$

Observa-se que o primeiro elemento é 0 pois de fato, a palavra 'chocolate' não está presente na lista de ingredientes da primeira receita, entretanto o segundo elemento é 1 pois a palavra 'acúcar' está presente e assim por diante. Desse modo, foram construídos dois vetores e comparados usando a similaridade do cosseno vista acima. Sendo assim o resultado obtido foi de aproximadamente 0.79 indicando que as duas receitas são muito similares.

De fato, conforme discutido em (LANE COLE HOWARD, 2019), em vetores de documentos que possuem uma similaridade próxima a 1, podemos concluir que eles estão usando palavras similares em proporções similares.

Antes mesmo de ser retornado pelo *back-end* as receitas mais similares, a primeira operação a ser realizada é o cálculo das similaridades. Para isso como mostra o Quadro 17 o corpo da requisição é vazio e deve ser realizada uma requisição HTTP GET. É preciso entender que que a tabela que armazena os dados se chama *similarities* e armazena o valor da similaridade entre as receitas e os *ids* dessas receitas. Assim, o método começa executando a deleção desses registros. Isso foi feito para manter a consistência dos dados.

Prosseguindo, para cada par entre receitas é executado um método chamado **similaridade**, esse método recebe como parâmetro dois vetores, onde cada vetor é um lista de ingredientes representando cada receita. Para cada palavra de cada um dos vetores ocorre um processo de pré processamento textual como explicando do Capitulo 2. Após essa etapa, ocorre como descrito no Capitulo 2 um processo conhecido como Bag of Words onde são criados dois vetores números que contém a frequência de cada palavra em cada vetor. Por fim, esses dois vetores números são enviados para outro método chamado **calculaSimilaridade** que executa o cálculo da similaridade do cosseno e retorna o valor calculado. Com a similaridade calculada um registro é feito na tabela de similaridades.

Quadro 17 – Dados da requisição para calcular as similaridades entre receitas

URI	**/similarity-recipes**
Método	GET
Corpo da Requisição	vazio

Fonte: Elaborado pelo autor

Agora com a similaridade calculada, a rota para calcular as receitas mais similares como mostra o Quadro 18 precisa receber como parametro de rota apenas o id da receita. Com o id, é

feita uma consulta na tabela de similaridades buscando as 6 receitas mais similares em ordem decrescente.

Quadro 18 – Dados da requisição para obter as receitas mais similares a uma determinada receita

URI	**/similarity/:recipeId**
Método	GET
Corpo da Requisição	vazio

Fonte: Elaborado pelo autor

Na tela do aplicativo basta o usuário entrar em alguma receita e ao rolar para baixo irá ver uma seção chamada 'Receitas semelhantes'. Essa seção irá exibir as 6 receitas mais similares à receita visualizada, podendo inclusive clicar em uma das receitas e visualizar os detalhes dela.

Figura 20 – Exemplo de receitas mais similares

Fonte: Elaborado pelo autor.

5.5.2 Feed recomendativo

Essa é uma das operações mais importantes da aplicação e consiste em mostrar para o usuário uma lista de receitas que provavelmente não foram visualizadas por ele e que têm grande possibilidade de interessá-lo. Para poder montar essa lista foram usadas as técnicas de filtragem explicadas no Capítulo 2, a filtragem colaboritiva e a baseada em conteúdo.

5.5.2.1 Recomendação

Há várias técnicas para se fazer um sistema de recomendação. Entretanto, duas características são importante, como mostra Nara (2022). Uma delas é que é necessário saber algum dado de entrada do usuário, ou seja, no caso do presente trabalho, saber quais tipos de receita ele mais gosta. A outra seria ter alguma espécie de medida de similaridade entre as receitas para que a indicação tenha mais valor ao usuário. Um exemplo seria para o usuário que gosta de ver muitas receitas doces, mais especificamente envolvendo o ingrediente 'chocolate'. Nesse caso faz sentido recomendar receitas doces e que contenham chocolate.

Uma vez que já se tem uma medida de similaride entre as receitas, basta agora estabelecer qual técnica será utilizada para com bases nos gostos do usuário poder fazer uma boa recomendação.

Para o sistema montar a lista baseada em filtragem colaborativa primeiramente tem que calcular as similaridades entre os usuários, o que é feito a partir de uma requisição HTTP GET como monstra o Quadro 19. É preciso entender que que a tabela que armazena os dados se chama *similarities_users* e armazena o valor da similaridade entre os usários e os *ids* desses usuários. Assim, o método começa executando a deleção desses registros. Isso foi feito para manter a consistência dos dados.

Prosseguindo, para cada par entre usuários é executado um método chamado **similaridade**, esse método recebe como parâmetro dois vetores, onde cada vetor é um lista de números onde representando quais receitas determinado usuário gostou ou não. Por fim, esses dois vetores númericos são enviados para outro método chamado **calculaSimilaridade** que executa o cálculo da similaridade do cosseno e retorna o valor calculado. Com a similaridade calculada um registro é feito na tabela de similaridades entre usuários.

Quadro 19 – Dados da requisição para calcular as similaridades entre usuários

URI	**/similarity-users**
Método	GET
Corpo da Requisição	vazio

Fonte: Elaborado pelo autor

Com as similaridades entre usuários calculadas, pode-se agora gerar a lista de recomendação para o usuário. Começa-se então pela abordagem da filtragem baseada em conteúdo onde

é obtido do banco de dados 50 - valor escolhido arbitrariamente - receitas as quais o usuário logado gostou, a lista é embaralhada e o primeiro elemento da lista é armazenado em um variável. Com esse elemento armazenado é feita uma consulta no banco de dados recuperando as 20 receitas mais similares a essa receita armazenada. Já para gerar a lista da filtragem colaborativa recupera-se do banco de dados o usuário mais similar ao usuário logado e por fim obtemos as receitas que esse usuário similar gostou. Com as duas listas geradas basta retorná-las no corpo da resposta.

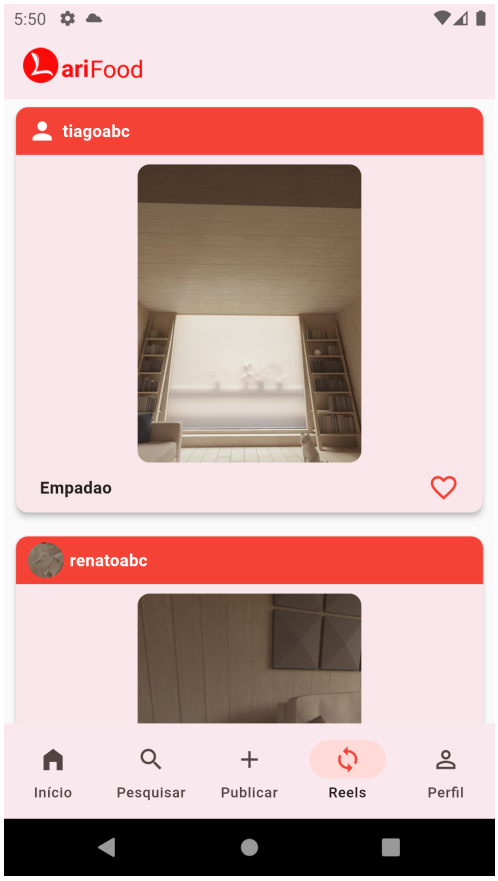
Quadro 20 – Dados da requisição para gerar o *feed* recomendativo

URI	**/recomendative-feed**
Método	GET
Corpo da Requisição	vazio

Fonte: Elaborado pelo autor

A Figura 21 mostra que a tela de visulização das receitas no feed recomendativo é muito parecida com a tela do feed cronológico, tendo como diferença: a fonte de dados utilizada, a não exibição da data da postagem e não possibilidade de guardar uma receita nos favoritos.

Figura 21 – Exemplo de feed recomendativo



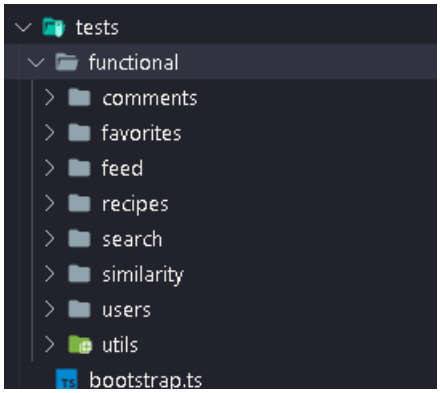
Fonte: Elaborado pelo autor.

5.6 Testes

Como descrito na Seção 2, a metodologia TDD foi aplicada no desenvolvimento do presente trabalho, em especial no desenvolvimento do *back-end*. Existem dois tipos de testes: testes unitários e de integração (SOMMERVILLE, 2011). No projeto foi aplicado os testes unitários que procuram aferir a corretude do código, em sua menor fração. Em linguagens orientadas a objetos, essa menor parte do código pode ser um método de uma classe. Sendo assim, os testes unitários são aplicados a esses métodos, a partir da criação de classes de testes (DEVMEDIA, 2022).

A Figura 22 mostra que os testes foram divididos em pastas, sendo cada que cada pasta representa algum recurso da API. Já na Figura 23 pode-se observar um exemplo de teste criado para testar a implementação do método 'store' responsável por criar um novo usuário no banco de dados. O teste começa com a criação da variável **userPayload** que simula um JSON a ser enviado para a requisição. Depois é feita uma requisção para a rota **/users** com a variável **userPayload** enviada no corpo da requisição. Por fim, são feitos vários **asserts** que nada mais são que validações para saber se a resposta recebida pelo servidor é igual ao que realmente seria esperado. Portanto foi feito um **assert** para checar o status da resposta, outro **assert** para checar se o corpo possui o campo **user** e por fim outro **assert** foi realizado como medida de segurança para checar se a senha não foi enviada no corpo da resposta.

Figura 22 – Estrutura de pasta para os testes.



Fonte: Elaborado pelo autor.

Para implementar a metodologia TDD foram criados diagramas como o da Figura 24 onde primeiro é definido qual é o recurso, no caso, **Users**, depois são definidas as operações a serem realizadas: **Create** e **Update**. Por fim são definidos os cenários de erro e de o cenário de sucesso.

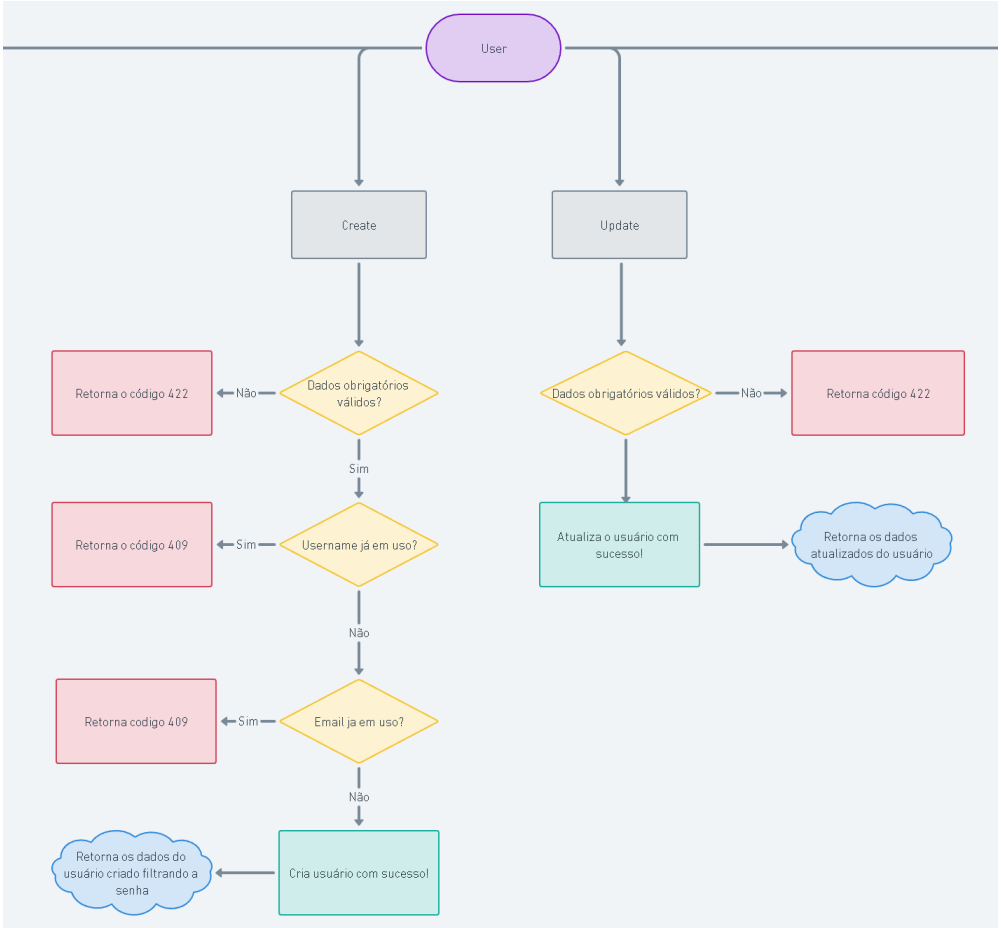
O que foi visto na Figura 24 em diagrama pode ser visualizado no código na Figura 25.

Figura 23 – Exemplo do teste para criação de um usuário.

```
// UsersController.store
test('it should create an user', async ({ assert, client }) => {
  const userPayload = {
    email: 'test@test.com',
    username: 'test',
    password: 'testee',
  }

  const response = await client.post('/users').json(userPayload)

  const { password, ...expected } = userPayload

  response.assertStatus(201)
  response.assertBodyContains({ user: expected })

  assert.exists(response.body().user, 'User undefined')
  assert.notExists(response.body().user.password, 'Password defined')
})
```

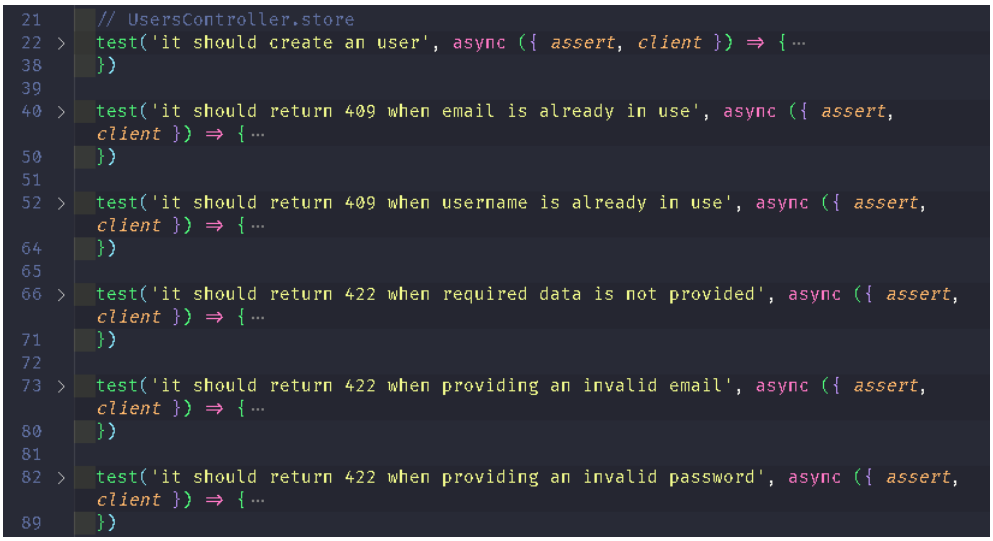
Fonte: Elaborado pelo autor.

Figura 24 – Exemplo de testes unitários.



Fonte: Elaborado pelo autor.

Figura 25 – Cenários de testes e sucesso para criar um usuário.



Fonte: Elaborado pelo autor.

5.7 *Deploy* da API

Com o objetivo da interface *mobile* ter acesso a API foi preciso disponibilizá-la em um servidor. O escolhido foi o Heroku é uma PaaS (Plataforma como um Serviço) que permite hospedagem, configuração, testagem e publicação de projetos virtuais na nuvem. Entre outras funções, ele facilita o trabalho dos desenvolvedores na configuração da infraestrutura para o deploy, ou seja, a implantação das aplicações (BARROS, 2021).

Com o *Heroku* conseguimos criar uma aplicação e conectá-la ao projeto da API que está hospedado no *Github*. Assim, sempre o código for atualizado no *Github*, o *Heroku* irá automaticamente identificar essa mudança e realizará novamente o deploy. Entre as vantagens do Heroku citadas por Barros (2021), estão:

- Deploy automático;
- Correção de problemas de implantação através do recebimento de e-mails;
- Praticidade e rapidez pelo fato de não ter que configurar servidores do zero;
- Desenvolvimento escalável com a possibilidade de aumentar a capacidade do servidor à medida que o projeto cresce.

Para a disponibilização da API desenvolvida neste projeto foi o escolhido o plano gratuito por ser suficiente para o propósito deste trabalho. O banco de dados escolhido foi o *PostgreSQL*. Entretanto é bom destacar que a gratuidade do Heroku está perto de acabar e que por causa disso será explicado em anexo, como realizar o deploy em outra plataforma chamada Render.

5.8 Resumo

Dessa forma, foi possível elicitar os requisitos funcionais necessários para o desenvolvimento do projeto. A partir dos requisitos funcionais o diagrama de casos de uso foi elaborado. Com isso, a aplicação foi desenvolvida e vimos as suas principais funcionalidades na Seção 4.4, com explicações sobre o que foi feito do lado do *back-end* e como interagir no *front-end*. Também vimos as funcionalidades relacionadas a inteligência artificial em uma seção à parte. Por fim, foi explicado como foram realizados os testes na aplicação e o *deploy* da API.

6
Arquitetura

É importante que todo projeto antes de ser executado tenha toda sua estrutura definida previamente, o que também ocorre em projetos de engenharia de software que necessitam da aplicação da arquitetura de software. De acordo com Perry (1992) a arquitetura de software é um conjunto de elementos arquiteturais (de dados, de processamento, de conexão) que possuem alguma organização. Os elementos e sua organização são definidos por decisões tomadas para satisfazer objetivos e restrições.

Uma das características mais importantes de uma arquitetura é o seu estilo, também conhecido como estilo arquitetural. Para Vergilio (2022) um estilo arquitetural pode ser definido por um conjunto de padrões, ou pela escolha de componentes ou conectores específicos que funcionarão como os tijolos básicos da construção. O mesmo autor também cita alguns exemplos de estilos arquiteturais.

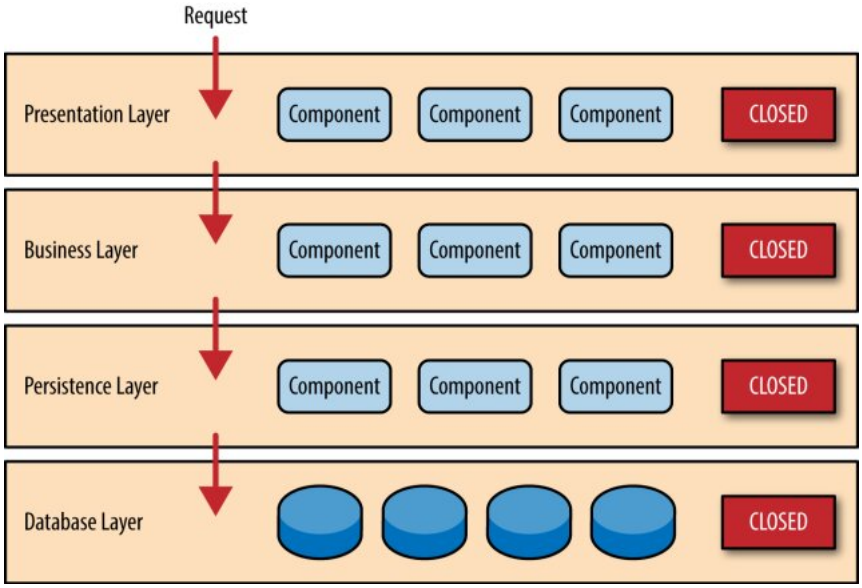
- Cliente-Servidor
- Camadas
- Filtros e dutos
- Repositório
- Orientado a eventos

O estilo escolhido para o presente projeto foi de camadas pois é um dos padrões arquiteturais mais usados, desde que os primeiros sistemas de software de maior porte foram construídos nas décadas de 60 e 70 (VALENTE, 2022). Também de acordo com Valente (2022) Em sistemas que seguem esse padrão, as classes são organizadas em módulos de maior tamanho, chamados de camadas, que por sua vez estão dispostas de forma hierárquica. Assim, uma camada

somente pode usar serviços — isto é, chamar métodos, instanciar objetos, estender classes, declarar parâmetros, lançar exceções, etc. — da camada imediatamente inferior.

A Figura 26 mostra um exemplo de arquitetura em camadas, onde uma requesição é recebida pela camada superior, sendo essa a camada de apresentação, que envia para a camada de negócios. Uma vez que a lógica de negócios foi aplicada, os dados têm que ser persistidos no banco de dados, o que leva a requisição para a camada de persistência e por fim a camada de banco de dados.

Figura 26 – Estilo arquitetural em camadas



Fonte: (SZALAI, 2022)

A arquitetura em camadas tem como vantagens as seguintes características:

- Escalabilidade da aplicação
- Aumenta a manutenibilidade da aplicação
- Facilita a organização dos componentes

6.1 Arquitetura da API

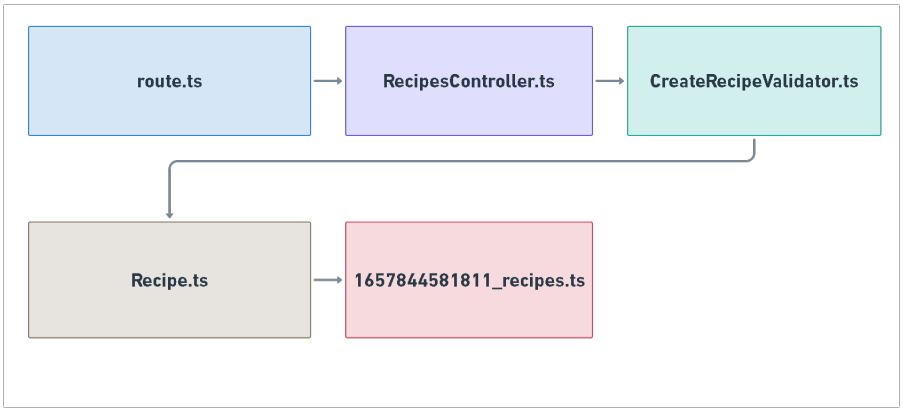
A aplicação da arquitetura em camadas no contexto da API desenvolvida pode ser observada a partir de 2 principais camadas. Na Figura 27 podemos ver qual o caminho que uma requisição para cadastrar uma receita faz ao ser recebida pelo serviço.

Primeiramente há o arquivo de rotas que identifica qual rota está sendo executada e qual o método da classe *controller* (seguindo o denominado padrão MVC) será chamado. Dentro

desse método, os dados recebidos no corpo da requisição serão validados. Por fim, os dados serão inseridos no banco de dados por meio do *model* de receitas. As duas principais camadas da aplicação são as seguintes:

- Camada de negócio: É onde está armazenada toda a lógica de negócio da aplicação, ou seja, armazena validações, lança exceções e por fim, interage com a camada abaixo.
- Camada de dados: É a camada que de fato interage com o banco de dados seja inserindo, recuperando, editando ou deletando.

Figura 27 – Representação da arquitetura da API



Fonte: Elaborado pelo autor.

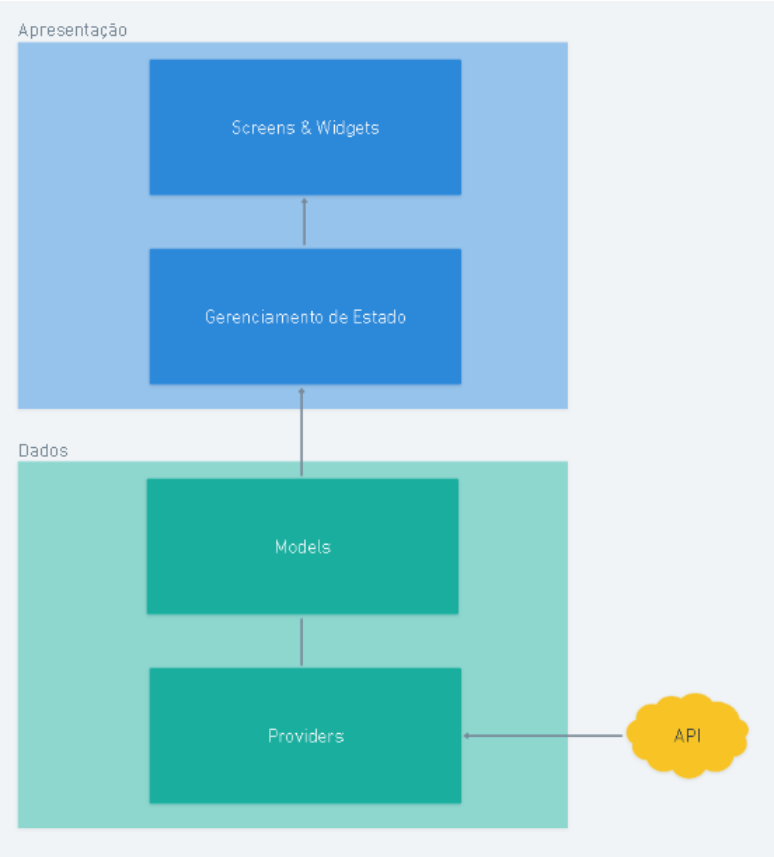
6.2 Arquitetura da Aplicação

A Figura 28 mostra que a divisão baseada no estilo arquitetural escolhido resultou em duas camadas: apresentação e dados. E para uma dessas camadas há algumas subcamadas.

Um ponto importante é a forma como os arquivos foram divididos em pastas. A divisão se deu basicamente em módulos, onde cada módulo representa uma parte da aplicação *front-end*, ou seja, uma tela, como mostra a Figura 29.

Por fim, a Figura 30 mostra que temos a pasta *core* que é resposável por conter algumas configurações gerais, como por exemplo, o tema da aplicação. A pasta *data* contém os *models* e *providers*. Enquanto que a pasta *modules* como explicado anteriormente contém os módulos do aplicativo. *routes* possui a configuração de todas as rotas e suas respectivas telas. Por fim, *widgets* é reposável por ter os componentes ou *widgets* globais, ou seja, que são usadas em várias telas do aplicativo.

Figura 28 – Fluxo entre as camadas da aplicação



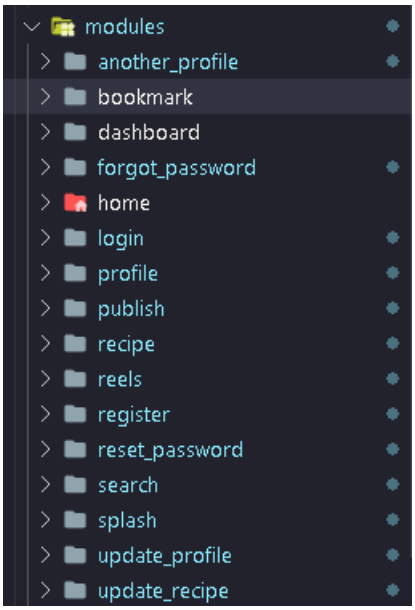
Fonte: Elaborado pelo autor.

6.2.1 Camada de apresentação

A camada de apresentação é responsável por todos os elementos visuais do aplicativo, contendo também os arquivos chamados de *controllers*, responsáveis pela gerência de estado das telas. Cada módulo da aplicação pode ser divido ainda em subcamadas: *page*, *controller*, *widgets*, *models*.

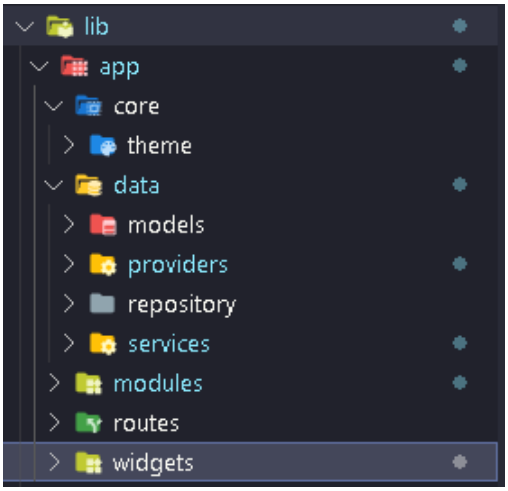
A camada *page* contém ó código utilizado para exibir a interface do aplicativo na tela. Como essa tela, que a que exibe detalhes de uma receita é um pouco complexa, pode-se subdividr os seus componentes ou *widgets* em *widgets* locais, é para isso que foi criada uma pasta de mesmo nome dentro do módulo *recipe*. Como para cada módulo, muitas das vezes precisamos renderizar informações diferentes do que geralmente é renderizado em outro módulo, foi criada a pasta *models* em alguns desses módulos para que seja possível atender tais especificidades. Por fim temos a camada *controller* responsável por toda a lógica de negócios e também pela gerência de estado da tela. Para realizar a gerência de estado foi utilizado um pacote chamado Getx, pois combina gerenciamento de estado de alto desempenho, injeção de dependência inteligente e gerenciamento de rotas de forma rápida e prática (GET, 2022).

Figura 29 – Módulos da aplicação



Fonte: Elaborado pelo autor.

Figura 30 – Divisão em pastas da aplicação geral



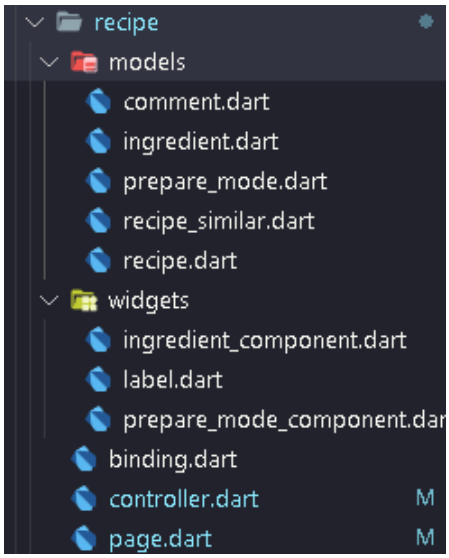
Fonte: Elaborado pelo autor.

6.2.2 Camada de dados

A camada de dados é resposável por executar operações referente a persistência dos dados, sejam eles presentes em fontes externas (API) ou internas (armazenamento local).

A primeira subcamada é a de *models* que nada mais é do que a representação de uma entidade do DER (Figura 11) e também é utilizada para realizar a serialização e desserialização de dados. O processo de desserialização converte dados externos em dados estruturados para a

Figura 31 – Divisão do módulo *recipe*



Fonte: Elaborado pelo autor.

aplicação e o processo de serialização converte dados estruturados em algum formato externo utilizado por uma API, por um armazenamento interno ou por outra fonte que recebe os dados (ISOCPP, 2022).

Já subcamada *providers* realiza a comunicação diretamente com o *back-end*, enviando ou recebendos dados no corpo das requisições. Na Figura 32 vemos um exemplo onde é enviado um JSON com o e-mail e a senha (*password*) para a rota
textitsessions.

Figura 32 – Requisição para a rota de *login*

```
Future loginUser(Map<String, dynamic> map) async {
  var response = errorHandler(await post('/sessions', {
    "email": map['email'],
    "password": map['password'],
  }));

  return response.body;
}
```

Fonte: Elaborado pelo autor.

7

Considerações Finais

As redes sociais são muito importantes nos dias atuais, seja porque facilitam a comunicação entre pessoas, mas também pois se tornaram ferramentas relavantes para as relações entre empresas e seus clientes. Além disso, devido ao grande volume de conteúdo, as pessoas acabam não conseguindo consumir os melhores conteúdos. Nesse sentido, este trabalho teve como objetivo construir uma aplicação denominada LariFood que visa ser uma rede social focada no conteúdo relacionado a receitas e que também faz uso de algoritmos de IA para construir um sistema de recomendação. O *download* da aplicação pode ser realizado clicando aqui [1].

Foram analisados os algoritmos de sistema de recomendação usados, as técnicas de similaridade e os vários tipos de filtragem existentes. Também foi feito um levantamento de aplicativos semelhantes existentes no mercado com o objetivo de indentificar os recursos utilizados e possíveis melhorias a implementar. Após, foi realizado o levantamento de requisitos junto com alguns outros artefatos de engenharia de software.

A partir disso, o desenvolvimento pode ser iniciando começando pelo *back-end* onde foi utilizado o *framework* AdonisJS e a metodologia TDD para auxiliar na implementação dos testes. Uma vez feito o *back-end* partiu para o *front-end* onde foi utilizado o *framework* Flutter. As funcionalidades desenvolvidas foram: cadastro de receitas, feed cronológico, feed recomendativo, interação entre usuários, etc.

7.1 Trabalhos Futuros

Há algumas sugestões que visam melhorar ainda mais a qualidade da aplicação:

- **Classificação de imagens**: O sistema irá classificar as imagens enviadas pelo usário no momento de cadastro de receita, impedindo assim, que sejam enviadas imagens inválidas;

[1] https://github.com/rvgcampos/larifood-app

- **Notificações**: O aplicativo irá notificar o usuário em determinadas situações: ao receber uma curtida, ao ser seguido por outro usuário, etc;
- **Organização dos favoritos**: O sistema permitiria adicionar as receitas guardadas com favoritas, em pastas, aumentando a organização;
- **Criação do modelo**: Uma outra técnica para definir similaridades entre entidade é o uso de K-nearest neighbors (KNN);
- **Anúncios**: Criar conta no serviço do Google chamado Admob e configurar o aplicativo para exibir anúncio em determinadas telas.
- **Integração com o LudiiPrice**: Integrar com o aplicativo, LudiiPrice (ARAUJO, 2021), desenvolvido em outro trabalho, para que seja possível obter o preço de cada um dos ingredientes e obter o valor total de determinada receita.
- **Testar a aplicação**: Realizar testes exploratórios com usuários reais com o intuito de entender possíveis erros ou melhorias na aplicação. As etapas podem consistir em: Desenvolver um plano, definir o escopo e as metas de teste do usuário, escolher um método de teste do usuário, indicar a localização, horário e equipamento, criar cenários de tarefas, identificar as principais métricas, estabelecer critérios de seleção dos usuários que irão testar, observar o processo, fazer um relatório e analisar os resultados (APPS, 2020).

Referências

AGRAWAL, R. *Must Known Techniques for text preprocessing in NLP*. 2021. Disponível em: <https://www.analyticsvidhya.com/blog/2021/06/must-known-techniques-for-text-preprocessing-in-nlp/>. Acesso em: 14 oct 2022. Citado na página 23.

ALEXANDRE. *Test Driven Development: TDD Simples e Prático*. 2022. Disponível em: <https://www.devmedia.com.br/test-driven-development-tdd-simples-e-pratico/18533>. Acesso em: 07 mai 2022. Citado na página 28.

APPS, R. *Etapas para realização de testes de usuários*. 2020. Disponível em: <https://rockapps.com.br/insights/etapas-para-realizacao-de-testes-de-usuarios/#Escolha_um_metodo_de_teste_do_usuario>. Acesso em: 25 nov 2022. Citado na página 69.

ARAUJO, A. C. *Ludiiprice: API para comparação de preços através de notas fiscais de produtos emitidas por estabelecimentos comerciais*. 2021. Acesso em: 07 mai 2022. Citado na página 69.

AWS. *O que é API RESTful?* 2022. Disponível em: <https://aws.amazon.com/pt/what-is/restful-api/>. Acesso em: 25 oct 2022. Citado na página 27.

BARROS, L. *O que é o Heroku e como a ferramenta revolucionou o desenvolvimento escalável?* 2021. Disponível em: <https://imaginedone.com.br/blog/o-que-e-o-heroku/>. Acesso em: 02 nov 2022. Citado na página 60.

CAMARGO, G. *O que é Sendinblue, como funciona e quais as vantagens e desvantagens da ferramenta?* 2020. Disponível em: <https://rockcontent.com/br/blog/sendinblue/>. Acesso em: 22 nov 2022. Citado na página 29.

CASAREDO. *5 apps que mostram receitas com o que tem em casa*. 2022. Disponível em: <https://www.casaredo.com/blog/2020/04/28/apps-mostram-receitas-com-o-que-tem-em-casa/>. Acesso em: 01 nov 2022. Citado na página 33.

CAZELLA, S. C. *SISTEMAS DE RECOMENDAÇÃO*. 2022. Disponível em: <http://www.nuted.ufrgs.br/compoa_2013_1/IntroducaoSR%20%281%29.pdf>. Acesso em: 18 nov 2022. Citado na página 14.

COSTA, H. *O que Test-Driven-Development não é*. 2022. Disponível em: <https://medium.com/creditas-tech/o-que-test-driven-development-n%C3%A3o-%C3%A9-584af2d4c65a>. Acesso em: 13 mai 2022. Citado 2 vezes nas páginas 28 e 29.

COSTA, M. M. A. A. Ana Carolina Silva da. *A percepção de internautas sobre as receitas mais acessadas em mídia digital*. 2021. Disponível em: <https://rsdjournal.org/index.php/rsd/article/download/20461/18437/250611>. Acesso em: 18 nov 2022. Citado na página 14.

CREATELY. *Tutorial do diagrama de caso de uso (guia com exemplos)*. 2021. Disponível em: <https://creately.com/blog/pt/diagrama/tutorial-de-diagrama-de-caso-de-uso/>. Acesso em: 24 oct 2022. Citado na página 40.

DEVMEDIA. *E aí? Como você testa seus códigos?* 2022. Disponível em: <https://www.devmedia.com.br/e-ai-como-voce-testa-seus-codigos/39478>. Acesso em: 25 oct 2022. Citado na página 58.

FEDYK, Y. *7 Benefits of Flutter: Why Use Flutter for Your Next App*. 2022. Disponível em: <https://inveritasoft.com/blog/7-flutter-advantages-a-perfect-platform-for-your-next-app>. Acesso em: 01 nov 2022. Citado na página 29.

FONSECA, C. *Introdução a Bag of Words e TF-IDF*. 2022. Disponível em: <https://medium.com/turing-talks/introduç~ao-a-bag-of-words-e-tf-idf-43a128151ce9>. Acesso em: 07 mai 2022. Citado na página 53.

GET. *Get*. 2022. Disponível em: <https://pub.dev/packages/get>. Acesso em: 20 oct 2022. Citado na página 65.

HAT, R. *API REST*. 2020. Disponível em: <https://www.redhat.com/pt-br/topics/api/what-is-a-rest-api>. Acesso em: 25 oct 2022. Citado na página 26.

ISOCPP. *Serialization and Unserialization*. 2022. Disponível em: <https://isocpp.org/wiki/faq/serialization>. Acesso em: 07 mai 2022. Citado na página 67.

JACOBY DONALD E. SPELLER, C. A. K. J. *Brand Choice Behavior as a Function of Information Load*. 1974. Disponível em: <https://www.jstor.org/stable/3150994>. Acesso em: 18 nov 2022. Citado na página 14.

KAPTIVA. *Information Overload: tudo o que você precisa saber*. 2022. Disponível em: <https://www.kaptiva.com.br/2019/11/29/information-overload-tudo-o-que-voce-precisa-saber/>. Acesso em: 25 oct 2022. Citado na página 18.

LANE COLE HOWARD, H. M. H. H. *Natural Language Processing in Action*. Rio de Janeiro: Manning, 2019. Citado 2 vezes nas páginas 53 e 54.

MACHADO, A. *Qual a diferença entre front-end e back-end?* 2021. Disponível em: <https://tecnoblog.net/responde/qual-a-diferenca-entre-front-end-e-back-end/>. Acesso em: 01 nov 2022. Citado na página 26.

MATOS, P. C. *Estudo comparativo de algoritmos de sistemas de recomendação de filmes*. 2021. Disponível em: <https://www.maxwell.vrac.puc-rio.br/57546/57546.PDF>. Citado 3 vezes nas páginas 19, 20 e 21.

NARA, R. *Algoritmos de recomendação: o que são e como implementá-los?* 2022. Disponível em: <https://blog.geekhunter.com.br/algoritmos-de-recomendacao-o-que-sao-e-como-implementa-los/>. Acesso em: 07 mai 2022. Citado na página 56.

NETO, C. *API REST: o que é e como montar uma API sem complicação?* 2022. Disponível em: <https://blog.betrybe.com/desenvolvimento-web/api-rest-tudo-sobre/>. Acesso em: 25 oct 2022. Citado na página 27.

ORACLE. *O que é IA? Saiba mais sobre Inteligência Artificial*. 2022. Disponível em: <https://www.oracle.com/br/artificial-intelligence/what-is-ai/>. Acesso em: 18 nov 2022. Citado na página 14.

PALERMO, F. *PostgresSQL JSON types*. 2022. Disponível em: <https://www.ufsm.br/pet/sistemas-de-informacao/2021/08/09/postgressql-json-types/>. Acesso em: 24 oct 2022. Citado 2 vezes nas páginas 41 e 42.

PEREIRA, S. do L. *Processamento de Linguagem Natural*. 2011. Disponível em: <http://walderson.com/2011-2/IA/07-processamentolinguagemnatural.pdf>. Acesso em: 14 oct 2022. Citado na página 23.

PERRY, W. *Foundations for the study of software architecture*. 1992. Citado na página 62.

PLAY, G. *Google Play*. 2022. Disponível em: <https://play.google.com/store/games?hl=pt_BR&gl=US>. Acesso em: 25 oct 2022. Citado 4 vezes nas páginas 33, 34, 35 e 36.

SALAZAR, F. A. S. R. *Um estudo sobre o papel de medidas de similaridade em visualização de coleções de documentos*. 2012. Disponível em: <https://teses.usp.br/teses/disponiveis/55/55134/tde-24012013-155903/publico/DissertacaoFrizziSRS.pdf>. Acesso em: 01 nov 2022. Citado na página 23.

SANTANA, M. *Deep Learning para Sistemas de Recomendação (Parte 2) — Filtragem Colaborativa com AutoEncoders*. 2018. Disponível em: <https://medium.com/data-hackers/deep-learning-para-sistemas-de-recomenda%C3%A7%C3%A3o-parte-2-filtragem-colaborativa-com-autoencoders-347ba7d53bae>. Acesso em: 12 oct 2022. Citado 3 vezes nas páginas 19, 20 e 21.

SHARDANAND, P. M. U. *Social Informating Filtering: Algorithms for Automating 'Word of Mouth'*. 1995. Disponível em: <https://dl.acm.org/doi/10.1145/223904.223931>. Citado na página 18.

SIQUEIRA, A. *Redes Sociais*. 2021. Disponível em: <https://resultadosdigitais.com.br/tudo-sobre-redes-sociais/>. Acesso em: 07 mai 2022. Citado 2 vezes nas páginas 14 e 17.

SIQUEIRA, J. K. *Similaridade – O que é ser igual ou similar na matemática?* 2022. Disponível em: <https://www.deviante.com.br/noticias/ciencia/similaridade-o-que-e-ser-igual-ou-similar-na-matematica/>. Acesso em: 07 mai 2022. Citado na página 53.

SIQUEIRA, M. M. S. *Efeitos da sobrecarga da Informação no cotidiano de jornalistas em Campo Grande MS*. 2006. Disponível em: <https://repositorio.unb.br/bitstream/10482/5520/1/2006_Marina%20Medina%20Saber.pdf>. Citado na página 17.

SOMMERVILLE, I. *Engenharia de Software*. 2011. Citado na página 58.

SOUSA, P. S. de. *Estimando Similaridade Entre Entidades Quando Apenas Seus Nomes Estão Disponíveis*. 2018. Disponível em: <https://www.repositorio.ufop.br/bitstream/123456789/10677/1/DISSERTA%C3%87%C3%83O_EstimandoSimilaridadeEntidades.pdf>. Citado na página 21.

SZALAI, A. *Padrões De Projeto – Arquitetura Em Camadas*. 2022. Disponível em: <http://blog.askm.com.br/2019/02/01/padroes-de-projeto-arquitetura-em-camadas/>. Acesso em: 20 oct 2022. Citado na página 63.

TAKAHASHI, M. *Estudo comparativo de Algoritmos de Recomendação*. 2015. Disponível em: <https://bcc.ime.usp.br/tccs/2014/marcost/monografia_final.pdf>. Citado 2 vezes nas páginas 18 e 19.

TECHLISE. *TUDO O QUE VOCÊ PRECISA SABER SOBRE TDD*. 2022. Disponível em: <https://www.techlise.com.br/blog/tudo-o-que-voce-precisa-saber-sobre-tdd>. Acesso em: 07 mai 2022. Citado na página 28.

VALENTE, M. T. *Engenharia de Software Moderna*. 2022. Disponível em: <https://engsoftmoderna.info/cap7.html>. Acesso em: 20 oct 2022. Citado na página 62.

VERGILIO, S. R. *Arquitetura de Software*. 2022. Disponível em: <https://www.inf.ufpr.br/andrey/ci163/IntroduzArquiteturaAl.pdf>. Acesso em: 20 oct 2022. Citado na página 62.

VERíSSIMO, R. *Levantamento de Requisitos e Mapeamento de Processos*. 2022. Disponível em: <http://www.linhadecodigo.com.br/artigo/1564/levantamento-de-requisitos-e-mapeamento-de-processos.aspx>. Acesso em: 07 mai 2022. Citado na página 38.

ZUCHER, V. *O que é um framework? Pra que serve e por que você deveria saber?* 2022. Disponível em: <https://www.lewagon.com/pt-BR/blog/o-que-e-framework>. Acesso em: 25 oct 2022. Citado na página 28.

Apêndices

APÊNDICE A – Versões das tecnologias utilizadas - *Front-end*

A seguir encontram-se as versões dos *plugins* utilizados no código da aplicação, a versão 3.3.3 do Flutter foi utilizada durante o desenvolvimento.

A.1 Dependências do aplicativo

- **get**: 4.6.3
- **get_storage**: 2.0.3
- **intl**: 0.17.0
- **transparent_image**: 2.0.0
- **image_picker**: 0.8.5+3
- **jiffy**: 5.0.0
- **form_validator**: 1.0.2
- **http**: 0.13.5
- **uni_links**: 0.5.1

A.2 Dependências de desenvolvimento

- **flutter_test**

APÊNDICE B – Versões das tecnologias utilizadas - *Back-end*

A seguir encontram-se as versões dos pacotes utilizados no código da API, a versão 5 do AdonisJS foi utilizada durante o desenvolvimento.

B.1 Dependências da API

- **@adonisjs/auth**: 8.2.1
- **@adonisjs/bouncer**: 2.3.0
- **@adonisjs/core**: 5.8.0
- **@adonisjs/lucid**: 18.1.0
- **@adonisjs/mail**: 8.1.2
- **@adonisjs/repl**: 3.1.0
- **@adonisjs/view**: 6.1.6
- **luxon**: 2.4.0
- **mysql**: 2.18.1
- **node-snowball**: 0.6.0
- **phc-argon2**: 1.1.3
- **proxy-addr**: 2.0.7
- **reflect-metadata**: 0.1.13
- **source-map-support**: 0.5.21
- **util**: 0.12.4

B.2 Dependências de desenvolvimento

- **@adonisjs/assembler**: 5.8.0
- **@japa/preset-adonis**: 1.1.0
- **@japa/runner**: 2.0.9
- **eslint**: 8.19.0
- **eslint-config-prettier**: 8.5.0
- **eslint-plugin-adonis**: 2.1.0
- **eslint-plugin-prettier**: 4.2.1
- **pino-pretty**: 8.1.0
- **prettier**: 2.7.1
- **sqlite3**: 5.0.8
- **typescript**: 4.6
- **youch**: 3.2.0
- **youch-terminal**: 2.1.4

Anexos

ANEXO A – Manual do usuário da aplicação

A.1 Introdução

O LariFood, na sua versão atual, é um aplicativo cujo objetivo é servir como uma rede social para usuários interessados em receitas. É possível cadastrar receitas e interagir com outros usuários. Também faz uso de Inteligência Artificial para prover o seu sistema de recomendação.

A.2 Como utilizar o aplicativo

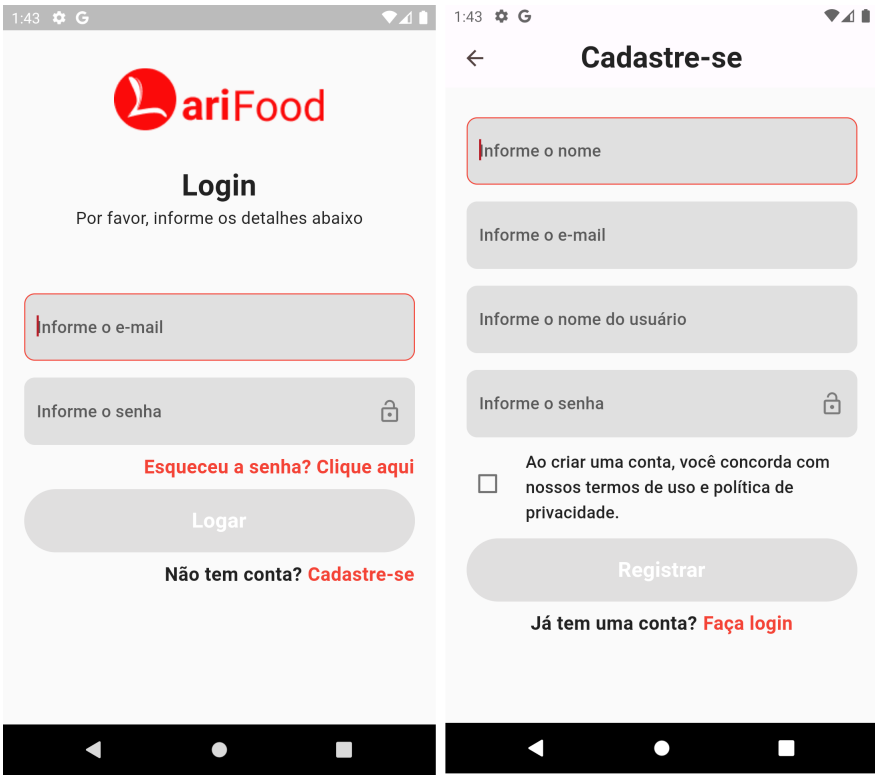
Esta seção descreve como utilizar todas as funcionalidades do aplicativo.

A.2.1 Criar conta e realizar login

Ao iniciar o aplicativo, o usuário irá visualizar um texto em vermelho escrito "Cadastre-se". Ao selecionar esse texto, o usuário será redirecionado para a tela à direita onde poderá criar uma conta inserindo as informações exigidas nos campos exibidos. Para tal basta inserir os dados e apertar no botão "Registrar".

Uma vez cadastrada uma conta, basta inserir o e-mail e a senha e clicar no botão "Logar"para entrar na tela inicial do aplicativo.

Figura 33 – Tela de cadastro

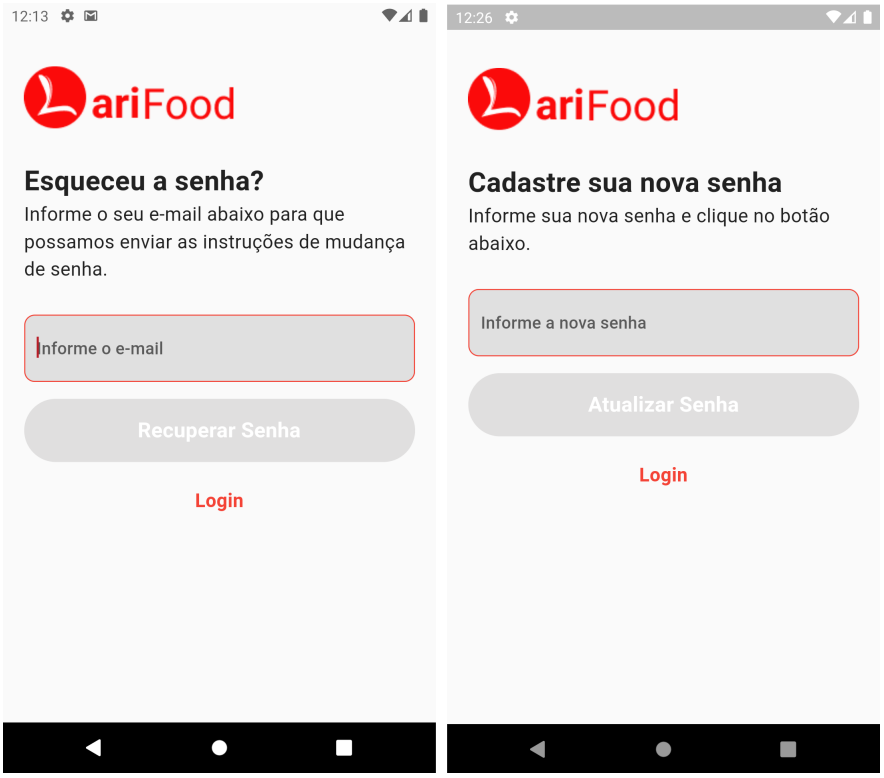


Fonte: Elaborado pelo autor.

A.2.2 Redefinir senha

Caso o usuário tenha esquecido sua senha, basta na tela inicial acessar a opção "Esqueceu a senha? Clique aqui". Ele será redirecionado para a tela abaixo onde será necessário informar seu e-mail cadastrado e apertar no botão "Recuperar Senha". Após, um e-mail será recebido pelo usuário onde haverá um *link* que deverá ser clicado. Ao clicar, será redirecionado para a tela à direita da 34 onde deverá inserir uma nova senha.

Figura 34 – Tela de redefinição

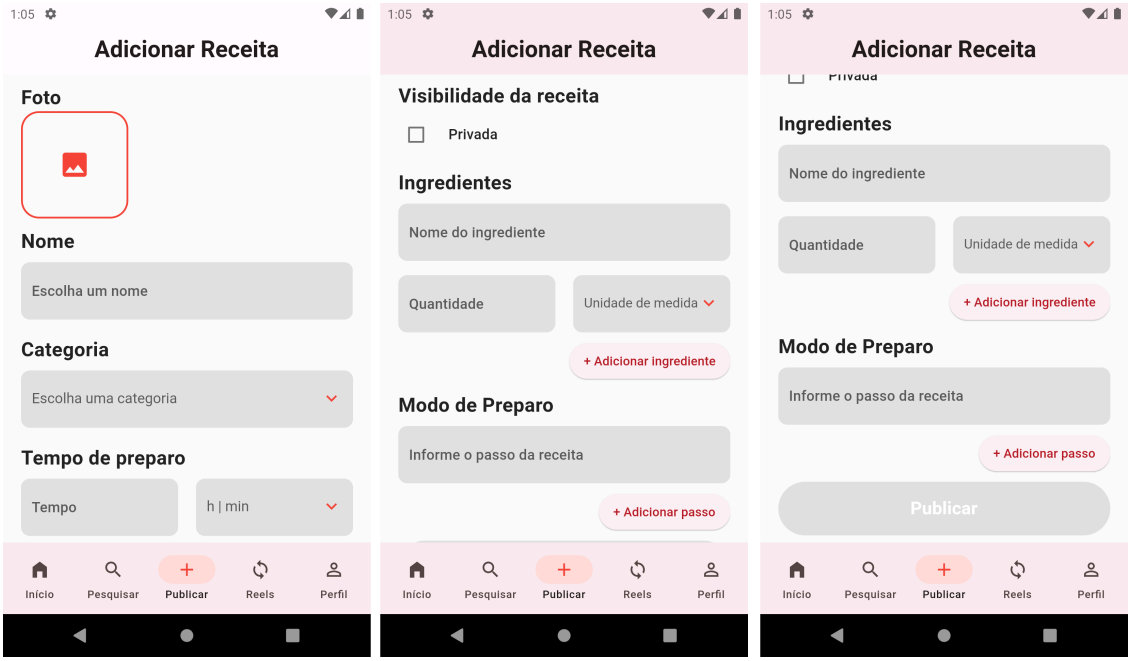


Fonte: Elaborado pelo autor.

A.2.3 Cadastrar receita

Para cadastrar uma receita o usuário deve apertar o botão "Publicar"presente no meio da barra da tela inicial. Na tela de cadastro ele deve inserir as informações necessárias, como ingredientes, modos de preparo, etc. Por fim, basta apertar no botão "Publicar"para finalizar a operação.

Figura 35 – Tela de cadastro de receita

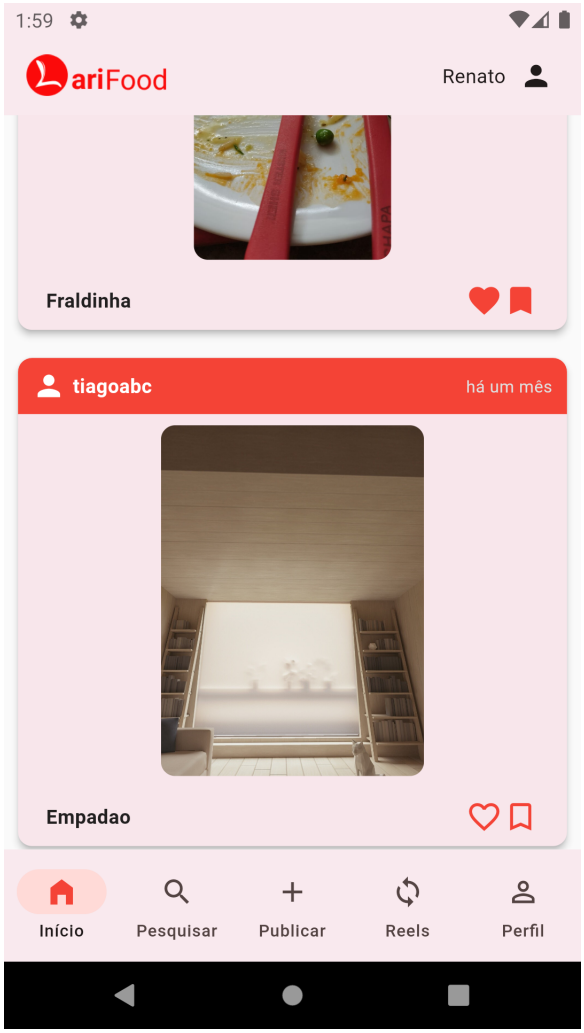


Fonte: Elaborado pelo autor.

A.2.4 Feed cronológico

Para visualizar a tela de feed cronológico, o usuário deve apertar o botão 'Início'. Esse feed exibe as postagens realizadas pelos seguidores do usuário. Nessa tela ele pode acessar a receita selecionada-a. Pode também curtir ou favorita na receita, tocando nos seus respectivos ícones.

Figura 36 – Feed cronológico



Fonte: Elaborado pelo autor.

A.2.5 Ver receitas similares

Para visualizar as receitas mais similares a uma determinada receita, basta que ele acesse uma receita e na seção ’Receitas semelhantes’ haverá as 6 receitas mais similares.

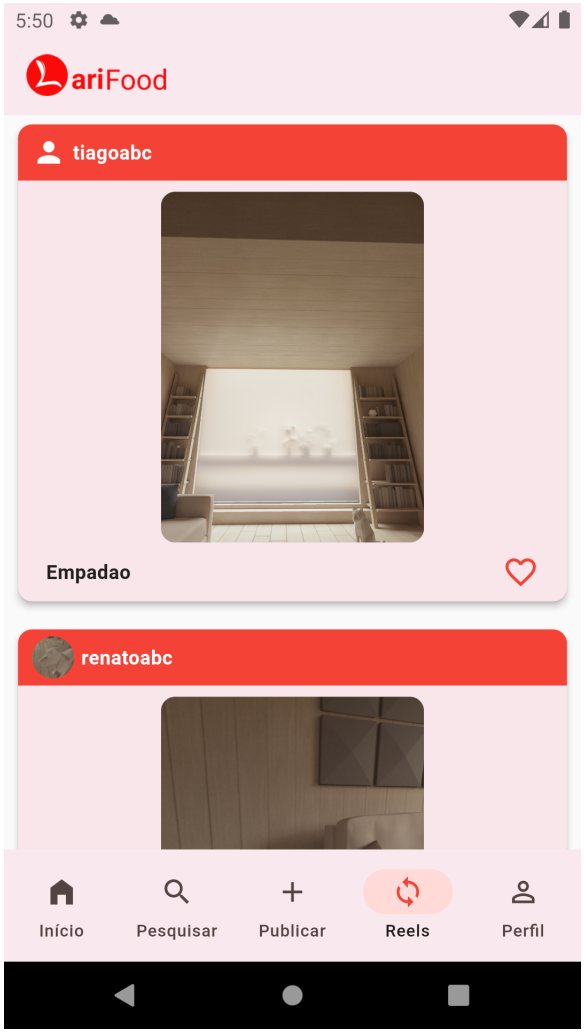
Figura 37 – Receitas mais similares

Fonte: Elaborado pelo autor.

A.2.6 Feed recomendativo

Para visualizar a tela de feed recomendativo, o usuário deve apertar o botão 'Reels' na barra que fica abaixo da tela inicial. Esse feed exibe postagens de interesse ao usuário, baseadas no cálculos realizados pelo sistema de recomendação. Nessa tela o usuário pode curtir a receita e acessá-la.

Figura 38 – Feed recomendativo

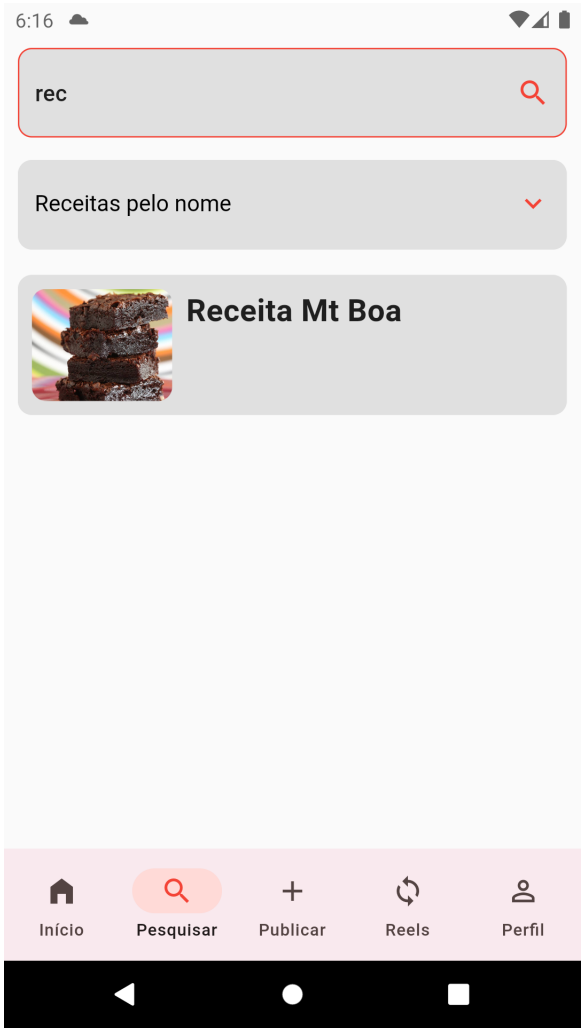


Fonte: Elaborado pelo autor.

A.2.7 Pesquisa

Para acessar a funcionalidade de pesquisa, o usuário deve apertar o botão "Pesquisa"na barra de botões. No rimeiro campo ele deve inserir o texto que será pesquisado. Já no segundo, ele deve escolher se quer pesquisar por receitas ou usuários.

Figura 39 – Pesquisa



Fonte: Elaborado pelo autor.

ANEXO B – Manual de deploy da aplicação

B.1 Heroku

Primeiro deve-se realizar o comando *git init* na pasta local para iniciar um repositório git. Depois deve-se conectar esse repositório ao repositório remoto.

Para conectar o repositório remoto ao Heroku deve-se criar uma aplicação no Heroku e adicionar um add-on do banco de dados *PostgreSQL*. Depois basta ir na aba **Deploy** e clicar no botão **Deploy Branch**.

Porém antes disso, é importante configurar as variáveis de ambiente. São elas: variáveis do servidor STMP, variáveis do banco de dados criado no Heroku e algumas variáveis do próprio framework.

B.2 Render

Já nessa outra solução, tem que se criar um **Web Service** e um banco de dados **PostgreSQL**. Também deve-se conectar o **Web Service** ao seu repositório no Github e também configurar as variáveis de ambiente.

Por fim é importante configurar os comandos de inicialização. Essa etapa não foi necessário no Heorku, pois esses comandos estão descritos no arquivo **.Procfile** que já está presente no repositório do projeto.

Os comandos seriam:

- Build Command: npm install && node ace build
- Start Command: cd build && npm i && node server.js