UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO DE CIÊNCIAS EXATAS, NATURAIS E DA SAÚDE

DEPARTAMENTO DE COMPUTAÇÃO
CIÊNCIA DA COMPUTAÇÃO

OTTO FREITAS QUINTANILHA

PROPOSTA DE UM APLICATIVO MÓVEL PARA O RESTAURANTE UNIVERSITÁRIO DO CCA-UFES

ALEGRE - ES
DEZEMBRO 2023

PROPOSTA DE UM APLICATIVO MÓVEL PARA O RESTAURANTE UNIVERSITÁRIO DO CCA-UFES

Trabalho de conclusão de curso apresentado ao Departamento de Computação do Centro de Ciências Exatas, Naturais e da Saúde da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

por

OTTO FREITAS QUINTANILHA

Orientador
GIULIANO PRADO DE MORAIS GIGLIO

Universidade Federal do Espírito Santo

ALEGRE - ES

DEZEMBRO 2023

OTTO FREITAS QUINTANILHA

Proposta de um aplicativo móvel para o restaurante universitário do CCA-UFES

Trabalho de Conclusão de Curso apresentado ao Departamento de Computação do Centro de Ciências Exatas, Naturais e da Saúde da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Aprovado em 15 de dezembro de 2023.

COMISSÃO EXAMINADORA

Prof. MSc. Giuliano Prado de Morais Giglio
Universidade Federal do Espírito Santo
Orientador

Prof. Dr. Antonio Almeida de Barros Junior
Universidade Federal do Espírito Santo

Prof. Dr. Marcelo Otone Aguiar
Universidade Federal do Espírito Santo

AGRADECIMENTOS

Agradeço primeiramente a Deus, por jamais me desamparar ao longo de toda a trajetória percorrida até este momento.

À minha família, pelo constante incentivo e apoio, que tem sido fonte de força e estímulo nos estudos.

Ao meu orientador Giuliano pela paciência, suporte e orientação no desenvolvimento do trabalho.

Aos professores da instituição especialmente os professores do departamento de computação pelo valioso conhecimento adquirido ao longo deste percurso acadêmico.

Aos amigos que fiz em Alegre pelo apoio e companheirismo durante a graduação.

SUMÁRIO

Lista de Quadros **11**

1 Introdução **15**

1.1 O problema e sua importância 16

1.2 Objetivos 17

1.2.1 Objetivo Geral 17

1.2.2 Objetivos específicos 18

2 Revisão de Literatura **19**

2.1 *Campus* Inteligente 19

2.2 Aplicações relacionadas 21

2.3 Tecnologias envolvidas 22

2.3.1 ReactJS 23

2.3.2 Gerenciamento de estado e comunicação com o servidor 23

2.3.3 React Native 24

2.4 Backend 25

2.4.1 REST 25

2.4.2 Node.js 26

2.4.3 Middlewares 28

2.4.4 JSON 28

2.4.5 Controllers 28

2.4.6 Rotas 28

2.4.7 Banco de Dados NoSQL 29

2.4.8 MongoDB 30
2.4.9 Schemas 31

3 Metodologia 32
3.1 Arquitetura do Sistema 32
3.1.1 Desenho da Arquitetura 33
3.1.2 Delimitações do projeto 34
3.2 Metodologia do Desenvolvimento 34
3.3 Analise dos requisitos 36
3.3.1 Descrição dos Usuários 36
3.3.2 Requisitos Funcionais 36
3.3.3 Requisitos não funcionais 37
3.3.4 Casos de Uso 37
3.4 Modelagem Banco de Dados 38

4 Desenvolvimento 42
4.1 Metodologia do desenvolvimento 42
4.2 Estrutura do projeto 43
4.2.1 Conexão com o bancos de dados 44
4.2.2 Schemas 44
4.2.3 Middlewares 48
4.2.4 Services 49
4.2.5 Controllers 50
4.2.6 Rotas 51
4.3 Front end 52
4.3.1 Estrutura do front end web 52
4.3.2 Comunicação com o servidor 53

4.3.3 Rotas 57
4.3.4 Estrutura do frontend Mobile 60
4.3.5 Comunicação com o servidor 61
4.3.6 Rotas 64

5 Resultados 67
5.0.1 Interfaces do sistema web 67
5.0.1.1 Login 67
5.0.1.2 Cadastro 68
5.0.1.3 Interface inicial 69
5.0.2 Cadastro de cardápio 71
5.0.2.1 Listagem de cardápios 71
5.0.2.2 Cadastro de notícias 72
5.0.2.3 Listagem de notícias 73
5.0.2.4 Interface de gráficos 74
5.0.2.5 Interfaces da aplicação mobile 75
5.0.2.6 Interface de login 75
5.0.2.7 Questionário 76
5.0.2.8 Interface inicial 86
5.0.2.9 Interface de avaliação 87
5.0.2.10 Interface de noticias 89
5.0.2.11 Interface de informações gerais 91

6 Testes e garantia da qualidade 93
6.1 Teste de desenvolvimento 93
6.1.1 Teste de Integração 93
6.1.2 Teste *End-to-End* 96

6.2 Teste de usuário 98

6.3 Teste de usabilidade 99

6.4 Avaliação dos testes 100

7 Considerações Finais e Conclusões 102

7.1 Trabalhos futuros 104

Referências 105

Apêndice A – Questionário de usabilidade 110

Apêndice B – Questionário de usuários betas 112

Apêndice C – Questionário setor nutrição 114

LISTA DE FIGURAS

Figura 1 Diferenças entre SQL e NoSQL. 29

Figura 2 Exemplo de documento no formato *JSON*. 30

Figura 3 Arquitetura do Sistema 33

Figura 4 Exemplo de quadro kanban. 35

Figura 5 Diagrama de Caso de Uso. 38

Figura 6 Representação do documento Aluno. 39

Figura 7 Representação do documento refeição. 40

Figura 8 Representação do documento aviso. 40

Figura 9 Representação do documento avaliação. 41

Figura 10 Estrutura do projeto 43

Figura 11 Conexão com o banco *MongoDB*. 44

Figura 12 Schema cardápio. 45

Figura 13 Exemplo de *repository* do cardápio. 47

Figura 14 Exemplo de *middleware* de validação do cardápio. 48

Figura 15 Exemplo de *service* de criação de um cardápio. 49

Figura 16 Exemplo de *controller* 50

Figura 17 Exemplo de definição de rotas. 51

Figura 18 Estrutura do projeto web. 52

Figura 19 Módulos do *redux*. 53

Figura 20 *Actions* do cardápio. 54

Figura 21 *Saga* do cardápio 55

Figura 22 *Reducer* do cardápio 56

Figura 23 Página de listagem de cardápios 57

Figura 24 Rotas da aplicação web 59

Figura 25 Estrutura do projeto mobile. 60

Figura 26 Pasta contendo os *services* do projeto mobile. 61

Figura 27 Funções de requisições HTTP relacionadas ao cardápio. 62

Figura 28 *Hooks* customizados relacionados ao cardápio. 63

Figura 29 Código da interface de exibição do cardápio. 64

Figura 30 Rotas da aplicação mobile. 65

Figura 31 Arquivo de rota. 66

Figura 32 Interface de login. 68

Figura 33 Interface de cadastro. 69

Figura 34 Interface inicial. 70

Figura 35 Menu lateral. 70

Figura 36 Interface de cadastro de cardápio. 71

Figura 37 Interface de listagem de cardápios. 72

Figura 38 Interface de cadastro de noticia. 73

Figura 39 Interace de listagem de noticias. 74

Figura 40 Interface de gráficos. 75

Figura 41 Interface de login do aplicativo. 76

Figura 42 Interface de informativa sobre o questionário. 77

Figura 43 Primeira etapa do questionário. 78

Figura 44 Segunda etapa do questionário. 79

Figura 45 Terceira etapa do questionário. 80

Figura 46 Quarta etapa do questionário. 81

Figura 47 Quinta etapa do questionário. 82

Figura 48 Sexta etapa do questionário. 83

Figura 49 Sétima etapa do questionário. 84

Figura 50 Oitava etapa do questionário. 85

Figura 51 Nona e última etapa do questionário. 86

Figura 52 Interface inicial da aplicação móvel. 87

Figura 53 Interface inicial avaliação do cardápio. 88

Figura 54 Janela de avaliação do cardápio. 89

Figura 55 Interface de listagem de noticias. 90

Figura 56 Interface de detalhe da noticia. 91

Figura 57 Interface de informações gerais. 92

Figura 58 Parte 1 do questionário de usabilidade 110

Figura 59 Parte 2 do questionário de usabilidade 111

Figura 60 Parte 1 do questionário para usuários betas 112

Figura 61 Parte 2 do questionário para usuários betas 113

Figura 62 Parte 1 do questionário conexionado pelo setor de nutrição 114

Figura 63 Parte 2 do questionário conexionado pelo setor de nutrição 115

LISTA DE QUADROS

1 Métodos HTTP. 25
2 Testes de integração realizados 95
3 Tabela de testes *end to end* realizados 96
4 Tabela de testes *end to end* realizados 97
5 Problemas encontrados no teste beta 99
6 Problemas encontrados no teste de usabilidade 100

LISTA DE SIGLAS

UFES	Universidade Federal do Espírito Santo
RU	Restaurante Universitário
SPA	*Single Page Aplication*
RN	React Native
REST	*Representational State Transfer*
API	*Application Programming Interface*
NPM	*Node Package Manager*
JSON	*JavaScript Object Notation*
NoSQL	*Not Only SQL*
HTTP	*Hypertext Transfer Protocol*
XML	*Extensible Markup Language*

RESUMO

Esta trabalho apresenta uma proposta de desenvolver uma aplicação móvel para que alunos possam ter acessos ao cardápio do Restaurante Universitário da UFES *campus* Alegre, e avaliar o cardápio do dia, juntamente fornecer uma forma do setor de nutrição recolher dados sobre perfil nutricional dos alunos. E de uma aplicação *web* para que o setor de nutrição do RU possa cadastrar o cardápio, verificar avaliação do cardápio e visualizar os dados colhidos dos alunos. Para o desenvolvimento da proposta, utilizou-se tecnologias como NodeJS, além de um banco de dados *MongoDB*. Para construção da aplicação *web* utilizou-se ReactJS e para a construção do aplicativo móvel a utilização do React Native.

Palavras-chave: Restaurante Universitário; Aplicativos Moveis; NodeJS; ReactJS; React Native.

ABSTRACT

This work presents a proposal to develop a mobile application so that students can access the menu of the University Restaurant at UFES campus Alegre, and evaluate the menu of the day, along with providing a way for the nutrition sector to collect data on students' nutritional profile. And a web application so that the RU nutrition sector can register the menu, check evaluation menu and view the data collected from students. To develop the proposal, technologies such as NodeJS were used, in addition to a MongoDB database. ReactJS was used to build the web application and React Native was used to build the mobile application.
Keywords: University Restaurant; Mobile Applications; NodeJS; ReactJS; React Native.

1 INTRODUÇÃO

Os dispositivos móveis estão cada vez mais presente na sociedade atual. Segundo Lima (2018) o Brasil superou a marca de um *smartphone* por habitante e hoje conta com 220 milhões de celulares inteligentes ativos e que em maio de 2018, a FGV estima que teremos 174 milhões de computadores no Brasil entre computadores de mesa, *notebooks* e *tablets*. A média é de cinco computadores para cada habitante. O grande avanço das tecnologias de informação contribuem para o aumento desse numero.

O uso da tecnologia da informação nas instituições de ensino, vem em crescimento progressivo. Ter acesso as informações, pesquisar e buscar as soluções eficientes e de forma rápida são uns dos benefícios do uso da tecnologia. O desenvolvimento tecnológico para órgãos públicos se tornou algo essencial para a melhor gestão de seus processos. Neste sentido é comum vários *softwares* serem desenvolvidos, cada qual com seu propósito em atender uma determinada área. (SILVA; LAMPKOWSKI, 2017)

Em um ambiente acadêmico, o uso dessas tecnologias podem auxiliar a comunidade na tomada de medidas que possam melhorar o ambiente acadêmico. Segundo Jacoski e Hoffmeister (2018 apud MOREIRA, 2014) á medida que as pessoas interagem com a comunidade auxiliando na tomada de decisões através do uso intensivo e integrado das tecnologias de comunicação, a cidade passa a ser vista como um laboratório vivo, pelo qual são desenvolvidas e testadas ideias em tempo real nas mais variadas áreas do cotidiano.

Assim sendo, as cidades inteligentes permanecem associadas à inovação e a resolução de problemas, defendendo uma inteligência competitiva que promove um desenvolvimento inteligente e sustentável (JACOSKI; HOFFMEISTER, 2018 apud MOREIRA, 2014). Segundo Jacoski e Hoffmeister (2018) levar estes conceitos de Cidades Inteligentes para os *campi*, aparentemente parece uma medida interessante, uma

vez que os próprios espaços das Universidades em muitos casos confundem-se com o meio urbano.

Uma forma de aplicar os conceitos de *campus* inteligente, seria disponibilizar uma aplicação para colher dados sobre os alunos que usam o Restaurante Universitário (RU), visto que a maiorias dos acadêmicos frequentam o restaurante para adquirir suas refeições diárias.

Segundo Benvindo, Pinto e Bandoni (2017) o Brasil conta com uma série de políticas públicas que promovem refeições gratuitas ou subsidiadas, que incentivam a oferta de refeições saudáveis em espaços institucionais. Entre elas está a política de restaurantes universitários, parte do Plano Nacional de Assistência Estudantil.

Benvindo, Pinto e Bandoni (2017) salienta que para grande parte dos estudantes, o ingresso na universidade é marcado por intensas transformações. É o momento em que eles terão que se responsabilizar pela moradia, alimentação, finanças e estudos. A existência do restaurante universitário pode minimizar o impacto que a mudança do domicílio familiar acarreta e contribuir para a manutenção da saúde do indivíduo, com a oferta de cardápios diversificados e nutritivos (BENVINDO; PINTO; BANDONI, 2017 apud OLIVEIRA; GUAGLIANONI; DEMONTE, 2009).

O RU da UFES *campus* Alegre, atende a comunidade acadêmica de Alegre, oferecendo refeições diárias e assistência alimentar para os estudantes universitários. Por ser um espaço da comunidade, o RU é um ambiente de confraternização dos estudantes, um local de troca cultural.

A partir da necessidade de entrega de informações sobre o RU para os estudantes da instituição, surgiu a ideia de implementar uma aplicação como forma de entregar essas informações para os alunos, junto com a necessidade do setor de nutrição do RU coletar dados sobre possíveis distúrbios alimentares dos acadêmicos, a fim de adequar o cardápio ás necessidades nutricionais de todos os alunos, além de usar essas informações para estudos futuros de nutricionistas em suas pesquisas.

1.1 O problema e sua importância

Hoje no RU da UFES campus Alegre existem algumas formas em que o aluno pode verificar o cardápio do dia, sendo que primeira forma é acessando o site da instituição[1]

[1] http://www.alegre.ufes.br/

e navegando até a área onde o setor de nutrição do RU divulga cardápio do dia e a segunda forma é verificando um cartaz que contém o cardápio que fica anexado no local onde se compra as fichas para entrada no restaurante. Existem problemas nessas duas maneiras de divulgar o cardápio, sendo que se o aluno escolhe verificar o cardápio do dia pelo site da instituição, o mesmo deve navegar no menu lateral do site e selecionar a opção de serviços, após essa ação abre-se um submenu onde ao clicar no *link* Restaurante Universitário, o *site* irá redirecionar para a página de informações sobre o RU (UFES, 2019). Ao ser redirecionado a página, o aluno se depara com uma outra página web onde contém informações acerca do RU. Para descobrir o cardápio do dia o aluno deve selecionar uma data em um calendário contido na pagina e só assim é direcionado ao cardápio do dia. Logo, observa-se um longo caminho entre páginas e *links* até o aluno chegar na informação que precisa. No caso do aluno verificar o cardápio no divulgado pela forma de cartaz, o mesmo deve se deslocar ao local de divulgação, ou seja, caminhar até o local de venda de passes de alimentação para verificar uma simples informação.

Outro problema verificado é não existe uma forma do setor de nutrição do RU coletar dados sobre possíveis distúrbios alimentares que os acadêmicos possam ter. Ao coletar esses dados o RU pode interpreta-los e adequar as refeições de uma forma que possa atender a todos os estudantes. Além dos problemas já citados, o RU hoje não deixa disponível uma forma em que os alunos possam avaliar a refeição do dia, ou emitir sugestões que podem melhorar o preparo dos alimentos que serão disponibilizado aos acadêmicos. Visto isso a aplicação em questão visa resolver esses problemas citados.

1.2 Objetivos

1.2.1 Objetivo Geral

Este trabalho tem como objetivo desenvolver um aplicativo que torne o cardápio do RU acessível através de dispositivos móveis e coletar dados sobre distúrbios alimentares dos estudantes para redefinir as políticas nutricionais de acordo com o perfil do aluno e de possíveis problemas de saúde que possuem e que limitam os alimentos, tais como intolerância a a lactose, alergias, diabetes, entre outras.

1.2.2 Objetivos específicos

Além dos objetivos gerais e cumprir com a proposta do trabalho, pretende-se atingir aos seguintes objetivos específicos:

1. Criar a funcionalidade de informar os estudantes sobre o cardápio do dia
2. Criar um sistema de avaliação do cardápio do dia para que o RU possa saber a opinião dos alunos sobre a refeição;
3. Criar uma forma de coletar dados sobre possíveis distúrbios alimentares dos estudantes;
4. Criar um canal de comunicação do RU com os estudantes;
5. Criar uma forma do Setor de Nutrição do RU receber *feedback* dos alunos sobre o cardápio;
6. Estudar novas tecnologias não disponíveis na grade curricular atual.

2 REVISÃO DE LITERATURA

Nesta seção são apresentados os fundamentos teóricos que embasam o desenvolvimento do aplicativo

2.1 *Campus* Inteligente

Segundo Jacoski e Hoffmeister (2018 apud ABUARQOUB et al., 2017) *campus* inteligente implica que a instituição adotará tecnologias avançadas para controlar e monitorar automaticamente instalações no *campus* e fornecer serviços de alta qualidade para a comunidade, ou seja, estudantes e funcionários. Isso levou a aumentar a eficiência e a capacidade de resposta do campus e ter uma melhor tomada de decisão, utilização do espaço e experiência dos alunos.

Ainda de acordo com Jacoski e Hoffmeister (2018 apud YU et al., 2011) diz que *campi* inteligente são construídos para que professores e alunos, gerenciam os recursos disponíveis e melhorarem a experiência do usuário com serviços proativos. Um *campus* inteligente varia de uma sala de aula inteligente, que beneficia o processo de ensino dentro de uma sala de aula, para um *campus* inteligente que fornece muitos serviços proativos em um ambiente em todo o *campus* (...) *campus* é um ambiente social onde estudantes universitários têm muitas interações com seus amigos.

Ferreira e Araujo (2018 apud BURLEA-SCHIOPOIU; BURDESCU, 2017) descrevem universidade inteligente como uma universidade tradicional que implementa gradualmente um sistema interconectado com controle central dos recursos tecnológicos.

Neste contexto, exitem trabalhos que implementaram o conceito de *campus* inteligente nas universidades no mundo. Castro (2018) cita alguns exemplos dessas universidades:

- A Universidade de Wisconsin-Madison está trabalhando com sensores nos se-

máforos ao longo de um corredor de viagem para criar uma zona de testes de veículos autônomos.

- A Universidade de Michigan por sua área de testes de veículos autônomos de 50 mil metros quadrados chamada de *Mcity*.
- A Universidade de Washington, em Seattle, está testando um projeto de *smart grid* com mais de 200 medidores inteligentes instalados em prédios do *campus* que rastreiam o uso de energia em tempo real.
- A Clemson University e a Universidade Estadual da Califórnia Dominguez Hills contam com várias iniciativas na área de economia de energia e energia renovável.
- A Universidade do Texas, em Austin, tem sua própria rede elétrica inteligente para atender às necessidades de eletricidade, aquecimento e resfriamento de todo o *campus*.
- O MIT anunciou para 2019 a criação de uma nova faculdade com cursos combinando Inteligência Artificial, aprendizado de máquina e ciência de dados com outras disciplinas acadêmicas. Com U$$1 bilhão em financiamento, este será o maior investimento financeiro em Inteligência Artificial de qualquer instituição acadêmica dos EUA e provavelmente do mundo.

No Brasil também existem universidades adotando essas práticas. A USP (Universidade de São Paulo) desenvolveu um projeto que tem o objetivo de monitorar e aumentar a segurança dentro e fora do *campus*, por meio da aplicação de tecnologia inteligente. O sistema da USP utiliza dispositivos e câmeras de alta tecnologia para identificar pessoas em atividades suspeitas dentro da Cidade Universitária. Esses dispositivos detectam rostos e objetos e enviam as informações para um banco de dados em nuvem que, por sua vez, é capaz de identificar a pessoa se ela estiver cadastrada no sistema. Se houver indícios de atividades suspeitas, o sistema envia um alerta de segurança (FERREIRA, 2018).

A Facens (Faculdade de Engenharia de Sorocaba) desenvolveu um projeto dentro do *campus*, onde há coleta de dados contínua, por meio de sensores, câmeras e outros dispositivos. O sensores mensuram a água, a energia, a quantidade de carros que estão no *campus* por meio de visão computacional. Há sensores na caixa d'água para monitorar o volume e sinalizam quando é necessário acionar as bombas que puxam

água do poço artesiano,há até um aplicativo que mensura o nível de estresse dentro da faculdade. A análise é feita por meio de perguntas e o aplicativo ainda envia dicas para melhorar o bem-estar do usuário. Todos os dados são apresentados em um painel (FERNANDES, 2018).

A UFCG (Universidade Federal Campo Grande) estabeleceu uma portaria para estabelecer práticas de um *campus* inteligente na universidade, com a criação da portaria a Prefeitura Universitária espera promover uma gestão mais eficiente para a infraestrutura do *campus*, baseada nos dados coletados nos projetos desenvolvidos. Além disso, a participação da comunidade universitária no desenvolvimento de soluções para as questões em que é diretamente afetada, permitirá um estreitamento na sua relação com a Administração, promovendo a transparência e incentivando a colaboração (UFCG, 2019).

Já na Faculdade de Engenharia Elétrica e de Computação da Unicamp existe trabalho de rede de sensores para o controle de ativos de iluminação pública, onde um dispositivo integrado aos pontos de iluminação pública do *campus*, que é capaz de identificar defeitos na lâmpada, monitorar seu tempo de uso e atuar no sistema de iluminação, acendendo e apagando as lâmpadas. Cada ponto da rede será um ativo cadastrado com sua localização geográfica bem definida, permitirá o monitoramento de falhas e simplificará o trabalho dos responsáveis pela manutenção do *campus*. O sistema é capaz de proporcionar relatórios de funcionamento e fornecer dados em tempo real, bem como a atuação programada (ACIDADEON, 2017).

Este trabalho se alinha com o conceito de *campus* inteligente, ao visar aprimorar a experiência dos usuários, proporcionando um acesso fácil às informações sobre refeições para estudantes e funcionários. Além disso, o projeto simplifica o monitoramento do restaurante ao coletar dados sobre a popularidade dos pratos e receber *feedback* dos estudantes. A coleta de dados provenientes do questionário sobre possíveis distúrbios alimentares estabelece uma base para tomada de decisões baseada em dados coletados.

2.2 Aplicações relacionadas

Existem alguns aplicativos que possuem funções semelhantes ao proposto por este trabalho. Serão abordados alguns deles, com suas principais características. Os aplicativos que serão analisados serão: RU UFG, Cardápio+ USP, RUral, RU - UFC.

O aplicativo RU UFG [1] foi desenvolvido para permitir o acesso ao Restaurante Universitário da Universidade Federal de Goiás, possibilitando agilidade na identificação e validação dos usuários. Com funcionalidades como acompanhamento do histórico de compras, monitoramento do consumo, acesso ao cardápio semanal e de avaliação das refeições. O aplicativo está disponível, gratuitamente, para a plataforma *Android* e *iOS*

O aplicativo Cardápio+ USP [2] foi criado com o intuito de oferecer suporte aos estudantes da Universidade de São Paulo, disponibilizando funcionalidades que vão além da simples consulta de cardápios. Além de permitir aos usuários verificar as opções de refeições disponíveis, o aplicativo oferece uma forma de acessar informações sobre o saldo do cartão de acesso. Adicionalmente, destaca-se por fornecer dados essenciais relacionados a diversos serviços oferecidos pela Pró-Reitoria da instituição, abrangendo áreas como moradia, transporte, creche, serviço social e saúde mental. O aplicativo está disponível, gratuitamente, para a plataforma *Android* e *iOS*.

O aplicativo RUral [3] foi criado para auxiliar os usuários da Universidade Federal Rural de Pernambuco (UFRPE). Oferecendo recursos como a visualização do cardápio semanal, a possibilidade de enviar sugestões para o restaurante universitário e a opção de favoritar ingredientes presentes nos cardápios. O aplicativo está disponível, gratuitamente, para a plataforma *Android*.

O aplicativo RU - UFC [4] foi desenvolvido para oferecer suporte aos usuários da Universidade Federal do Ceará (UFC). Com funcionalidades que permitem o acesso ao restaurante universitário, verificação do cardápio semanal e visualização do histórico de utilização do cartão.O aplicativo está disponível, gratuitamente, para a plataforma *Android*.

2.3 Tecnologias envolvidas

Nesta seção são apresentados as tecnologias que foram utilizadas para o desenvolvimento do aplicativo.

[1] https://play.google.com/store/apps/details?id=br.ufg.cercomp.ru
[2] https://play.google.com/store/apps/details?id=br.usp.cardapio.usp
[3] https://play.google.com/store/apps/details?id=com.mobile.rural
[4] https://play.google.com/store/apps/details?id=br.ufc.restaurante

2.3.1 ReactJS

O ReactJS é uma biblioteca *JavaScript* usada para criar interfaces para aprimorar os aplicativos e melhorar o desempenho de renderização de paginas (BRITO et al., 2019). A biblioteca faz com que a criação de UIs[1] interativas seja uma tarefa fácil (FACEBOOK, 2019b). O ReactJS resolve o problema de recarregar páginas sempre que algo foi alterado na interface, com isso foi introduzido o conceito de *Single Page Aplication* (SPA) que são páginas que renderizam apenas conteúdo que foram alterados, reduzindo o custo de recarregar uma nova interface, a navegação na aplicação rola toda em uma única página e todo o conteúdo é carregado de uma vez ou obtido dinamicamente (FIGUEIREDO, 2018). A biblioteca usa o conceito de componentização de elementos, onde podemos criar componentes que gerenciam próprios estados e que se combinam para criar interfaces mais complexas (FACEBOOK, 2019b).

2.3.2 Gerenciamento de estado e comunicação com o servidor

A comunicação com o servidor é estabelecida por meio de requisições HTTP. Para entender como o *front end web* realiza as requisições, precisamos entender o fluxo de gerenciamento de estados da aplicação. Para isso foi utilizado a biblioteca *Redux*.

O *Redux* é uma biblioteca para armazenamento de estados de aplicações *JavaScript*. Ele nasceu através de uma implementação do *Flux*, para contribuir com as aplicações de interface do usuário, utilizando o conceito de fluxo de dados unidirecional. Quando desenvolvemos aplicações temos que lidar com o gerenciamento de estado. O *Redux* veio para suprir essa necessidade de simplificar o controle dos estados de uma aplicação (GUEDES, 2019).

O *Redux* utiliza três principais recursos para gerenciar estados de forma centralizada em uma aplicação:

- *Store*: A *store* é um contêiner centralizado que armazena todos os estados da aplicação de forma imutável. Funciona como um centro de informações que fornece dados conforme solicitado pelos componentes.
- *Actions*: As *actions* são ações específicas disparadas pela aplicação para a *store*. Elas são criadas através de*action creators* e são a única maneira de

[1] Interfaces do usuário

provocar mudanças nos estados na *store*. As *actions* representam eventos que descrevem o que aconteceu.

- *Reducers*: Cada dado na *store* tem seu próprio *reducer*. Os *reducers* são responsáveis por manipular as *actions* e especificam como o estado da aplicação deve mudar em resposta a uma *action* específica. Eles definem as transformações de estado com base nas *actions* recebidas, garantindo que as mudanças ocorram de maneira previsível e controlada.

O *Redux* organiza o gerenciamento de estados de uma aplicação por meio de uma *store* central, *actions* que representam eventos específicos e *reducers* que determinam como o estado da aplicação é alterado em resposta a essas *actions*. Em resumo o fluxo geralmente funciona da seguinte forma: um componente gera uma interação através de um clique dado pelo usuário na interface, um *action creator* é acionado e dispara uma ação para o *store*. Essa ação chega até um *reducer* que irá processar e fazer a alteração do estado no *store*. Assim um novo estado será disponibilizado para o componente. (GUEDES, 2019)

O *Redux*, por padrão, despacha ações de forma síncrona, o que pode ser limitante para aplicativos que precisam lidar com operações assíncronas, como chamadas de API. Para resolver esse problema, o *Redux* permite a inserção de *middleware* entre a ação sendo despachada e a ação que atinge os *reducers*. Isso possibilita a execução de operações assíncronas e efeitos colaterais, tornando o gerenciamento de estados mais flexível e adaptável para cenários complexos.(ALLIGATOR.IO, 2020)

2.3.3 React Native

Segundo Eisenman (2017) React Native (RN) é um *framework JavaScript* para criar aplicativos móveis. Ele é baseado na biblioteca ReactJS (FACEBOOK, 2019b), e segue a maioria dos seus conceitos. Usando o RN podemos compartilhar o mesmo código entre plataformas, simplicando o desenvolvimento para o *Android* e *iOS* e construindo aplicativos que realmente aparentam ser nativos.

Os aplicativos RN são escritos usando *JavaScript* e JSX, uma linguagem de marcação semelhante ao XML. Após ser programado, o RN chama as *APIs* de renderização nativas do Android e do *iOS*, produzindo componentes reais da interface da plataforma mobile. O RN também fornece acesso a APIs de plataformas que permitem o acesso ao recursos do *hardware* dos dispositivos (FACEBOOK, 2019a).

2.4 Backend

O *backend* refere-se à parte de um sistema de *software* que lida com o processamento e armazenamento de dados, além de realizar a lógica de negócios e interação com o banco de dados. É responsável por fornecer informações e funcionalidades para o *front end* (parte visível para os usuários) e geralmente não é acessível diretamente pelos usuários finais (HARVE, 2023).

2.4.1 REST

Conceitualmente falando, o modelo REST (*Representational State Transfer*) representa uma possibilidade para a criação de web services, cujas principais diferenças em relação ao modelo tradicional (SOAP) estão na utilização semântica dos métodos HTTP (*GET, POST, PUT e DELETE*), na leveza dos pacotes de dados transmitidos na rede e na simplicidade, fazendo desnecessária a criação de camadas intermediárias para encapsular os dados (COUTINHO, 2013).

Segundo Ferreira (2017) os recursos gerenciados por uma aplicação, geralmente podem ser manipulados de diversas maneiras. É possível criá-los, atualizá-los, excluí-los, dentre outras operações. Quando um cliente dispara uma requisição HTTP para um servidor, é necessário que ele informe o tipo de manipulação que deseja realizar no recurso. É justamente aí que entra um outro conceito da *Web*, que são os métodos do protocolo HTTP. O protocolo HTTP possui diversos métodos, sendo que cada um possui uma semântica distinta, e devem ser utilizados para indicar o tipo de manipulação a ser realizada em um determinado recurso.

Quadro 1: Métodos HTTP.

Método	**Descrição**
GET	Obter os dados de um recurso.
POST	Criar um novo recurso.
PUT	Substituir os dados de um determinado recurso.
PATCH	Atualizar parcialmente um determinado recurso.
DELETE	Excluir um determinado recurso.

Fonte: (FERREIRA, 2017).

Segundo Gupta (2018) os princípios orientadores do *REST* são:

- Cliente-servidor - Ao separar as preocupações da interface do usuário das preocupações com o armazenamento de dados, melhoramos a portabilidade da interface do usuário em várias plataformas e melhoramos a escalabilidade, simplificando os componentes do servidor.
- Sem estado - Cada solicitação do cliente para o servidor deve conter todas as informações necessárias para entender a solicitação e não pode tirar proveito de nenhum contexto armazenado no servidor. O estado da sessão é, portanto, mantido inteiramente no cliente.
- Armazenamento em cache - as restrições de cache exigem que os dados em uma resposta a uma solicitação sejam implicitamente ou explicitamente rotulados como armazenáveis em cache ou não em cache. Se uma resposta for armazenável em cache, o cache do cliente terá o direito de reutilizar esses dados de resposta para solicitações posteriores equivalentes.
- Interface uniforme - Ao aplicar o princípio de generalidade da engenharia de software à interface do componente, a arquitetura geral do sistema é simplificada e a visibilidade das interações é aprimorada. Para obter uma interface uniforme, são necessárias várias restrições arquiteturais para orientar o comportamento dos componentes. O *REST* é definido por quatro restrições de interface: identificação de recursos; manipulação de recursos através de representações; mensagens auto-descritivas; e hipermídia como o mecanismo do estado do aplicativo.
- Sistema em camadas - O estilo do sistema em camadas permite que uma arquitetura seja composta de camadas hierárquicas, restringindo o comportamento do componente, de modo que cada componente não possa "ver"além da camada imediata com a qual está interagindo.

2.4.2 Node.js

Node.js é uma tecnologia usada para executar código *JavaScript* fora do navegador. Com ele podemos construir aplicações web em geral, desde web sites até APIs e microsserviços. Isso é possível graças a união do ambiente de execução de *JavaScript* fornecido pelo próprio Node.js e o motor de interpretação e execução de *JavaScript* presente no *Google Chrome*, chamado de V8 (DEVMEDIA, 2019).

> O V8 é o *engine open-source* de *JavaScript* e *WebAssembly* de alto desempenho do Google. Ele foi escrito em C++ e usado tanto no Ch-

> rome quanto em ambientes similares ao *Chrome*, e no Node.js. O V8 possui a implementação completa para *ECMAScript* e *WebAssembly*. Mas ele não depende de um navegador, na verdade, o V8 pode ser executado de forma independente e incorporado a qualquer aplicativo C++. (SANTOS, 2019).

Segundo Satheesh, D'mello e Krol (2015) Node.js não é uma linguagem nova, não é apenas um *framework Javascript*, ele nos fornece um contexto em que podemos escrever código *JavaScript* em qualquer plataforma em que o Node.js pode ser instalado, onde usamos a linguagem *JavaScript* no lado do servidor.

O Node.js utiliza o modelo orientado a eventos de entrada e saída não bloqueante,o que o torna leve e eficiente (FOUNDATION, 2019). O Node.js tenta fazer processamento assíncrono em uma única *thread* para fornecer mais desempenho e escalabilidade para aplicativos que contém muito tráfego *web*. O modelo de processamento assíncrono de uma única *thread* tem a vantagem de processar requisições simultâneas com menos número de recurso do lado do servidor (SATHEESH; D'MELLO; KROL, 2015).

Satheesh, D'mello e Krol (2015) afirma que em um esforço para tornar o código mais modular e reutilizável possível,o Node.js usa um sistema de módulos que permite organizar melhor o código. A premissa básica é que você escreva um código cumprindo uma única preocupação e exporte esse código como um módulo que serve a esse único objetivo.

O núcleo do Node.js já vem com alguns módulos, porém existe uma infinidade de módulos de terceiros disponível no npm (*Node Package Manager*).

> O npm é o gerenciador de pacotes padrão do Node.js. Você pode usar o npm para instalar bibliotecas de terceiros (pacotes) e gerenciar dependências entre elas. O registro npm (www.npmjs.com) é um repositório público de todos os módulos publicados por pessoas com o objetivo de compartilhar. (SUBRAMANIAN, 2019).

Usando o gerenciador de pacotes interno do Node.js, o npm, pode-se encontrar literalmente dezenas de milhares de módulos que podem ser instalados e usado. Uma das razões para o maior sucesso do Node.js é o npm, que é um dos melhores gerenciadores de pacotes do mercado, com uma curva de aprendizado muito minuciosa.

2.4.3 Middlewares

As funções de *middlewares* são blocos de código que têm acesso aos objetos de solicitação e resposta em uma aplicação *web*. Além disso, elas também podem chamar a próxima função de *middleware* na sequência. Durante o ciclo de solicitação-resposta, essas funções desempenham um papel vital, permitindo a execução de código personalizado, modificação dos dados da solicitação e resposta, bem como a capacidade de encerrar o ciclo de solicitação-resposta, se necessário. Elas são fundamentais para processar e transformar dados, além de garantir a execução suave das operações em um aplicativo, proporcionando flexibilidade e controle sobre o fluxo de requisições e respostas (EXPRESS, 2023).

2.4.4 JSON

JSON (*JavaScript Object Notation*) é uma formatação leve de troca de dados. Para seres humanos, é fácil de ler e escrever. Para máquinas, é fácil de interpretar e gerar (JSON.ORG, 2019). Segundo Satheesh, D'mello e Krol (2015) *JSON* é a sintaxe padrão usada ao lidar com dados em JavaScript, bem como na maioria dos outros idiomas e serviços da *web*.

JSON.org (2019) define que um objeto é um conjunto desordenado de pares nome/-valor. Um objeto começa com chave de abertura e termina com chave de fechamento. Cada nome é seguido por :dois pontos e os pares nome/valor são seguidos por ,vírgula.

2.4.5 Controllers

Os *controllers* atuam como intermediários entre as solicitações dos usuários e a manipulação dos dados no *backend*. Eles também são responsáveis por preparar e enviar respostas de volta ao cliente, geralmente na forma de dados formatados, como *JSON* (LIMA, 2023).

2.4.6 Rotas

As rotas desempenham um papel vital no processamento dos pedidos e na entrega de respostas às aplicações clientes. Ao receber as demandas *HTTP*, o servidor de *bac-*

kend ativa as rotas pertinentes, que são encarregadas de processar as informações requisitadas (MOZZILA, 2023).

2.4.7 Banco de Dados NoSQL

NoSQL é o termo utilizado para banco de dados não relacionais de alto desempenho, onde geralmente não é utilizado o SQL como linguagem de consulta. O NoSQL foi criado para ter uma performance melhor e uma escalabilidade mais horizontal para suprir necessidades onde os bancos relacionais não são eficazes (GUEDES, 2017).

Uma das diferenças nos bancos de dados NoSQL é que toda a informação é agrupada e guardada no mesmo registro. Já no SQL você precisa ter o relacionamento entre várias tabelas para ter a informação, informação esta disposta no modelo entidade e relacionamento (GUEDES, 2017).

A Figura 1 ilustra as diferenças entre o SQL e NoSQL, onde as tabelas em bancos SQL são coleções em NoSQL, as linhas são documentos e as colunas em bancos SQL são campos em bancos NoSQL.

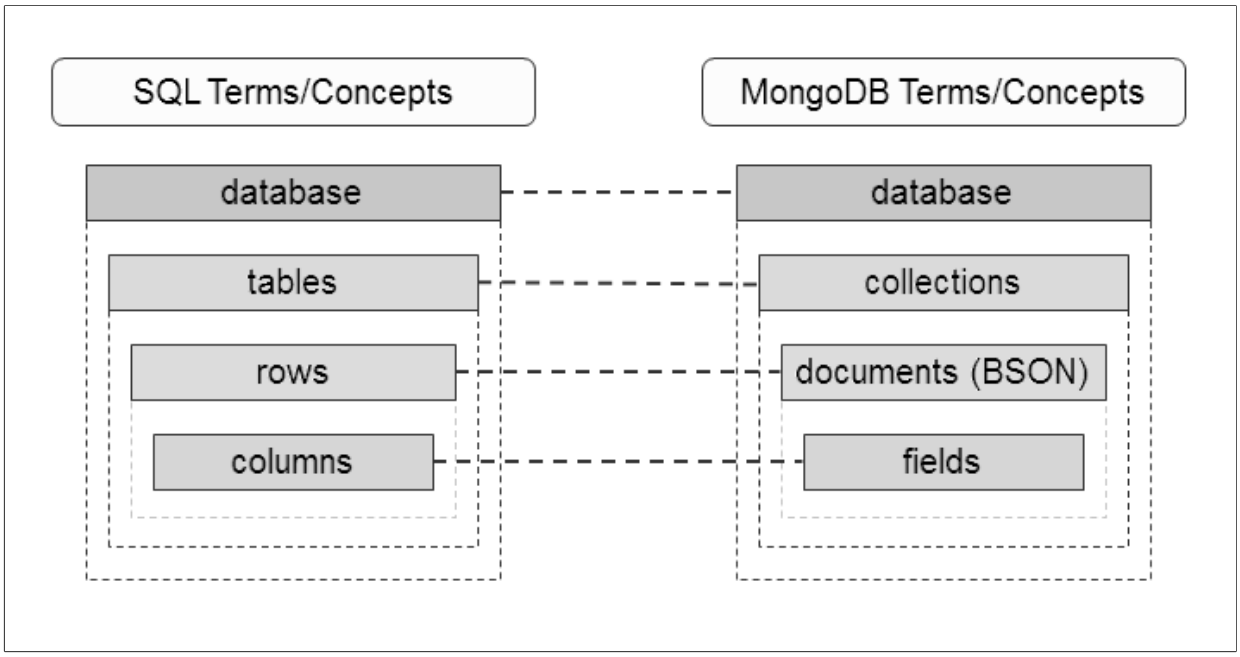


Figura 1: Diferenças entre SQL e NoSQL.
Fonte (GUEDES, 2017).

Segundo Kokay (2012) uma característica notável em bancos de dados NoSQL é a ausência parcial ou total de esquema que define a estrutura de dados. É justamente

essa ausência de esquema que facilita uma alta escalabilidade e alta disponibilidade. Existem algumas formas de guardar dados em bancos NoSQL, uma delas é em formato de documento. Guedes (2017) salienta que os dados são armazenados como documentos, podem ser descritos como dados no formato de chave-valor, como por exemplo, o padrão *JSON*. Um exemplo de banco de dados neste formato é o MongoDB. A Figura 2 ilustra um exemplo de documento *JSON*.

```
{
        "id": "00000234567894",
        "name": "Jane Doe",
        "birthday": "04/18/1978",
        "gender": "female",
        "type": "user",
        "work": [{
           "employer": {
              "id": "106119876543210",
              "name": "Doe Inc."
           },
           "start_date": "2007 - 08"
         },
        {
          "start_date": "2004",
          "end_date": "2007"
        }]
}
```

Figura 2: Exemplo de documento no formato *JSON*.
Fonte: (IMHOF; FROZZA; MELLO, 2017).

É importante referir que o NoSQL não veio para substituir o SQL, mas sim para oferecer uma alternativa mais flexível ao suporte de dados. Podemos sempre usar ambas as soluções para diferentes casos de uso (PERES, 2018).

2.4.8 MongoDB

O *MongoDB* é um banco de dados *NoSQL(Not Only SQL)* orientado a documentos, o que significa que ele armazena dados em documentos semelhantes a *JSON*. Acredita-se que essa é a maneira mais natural de pensar em dados e é muito mais expressiva e poderosa do que o modelo tradicional de linha/coluna (MONGODB, 2019).

Segundo Chodorow (2013) no centro do *MongoDB* está o documento um conjunto ordenado de chaves com valores associados. A representação de um documento varia de acordo com a linguagens de programação, mas a maioria das linguagem possuem uma estrutura de dados que é natural, como um mapa, *hash* ou dicionário.

Em *JavaScript*, por exemplo, os documentos são representados como objetos.

> Um documento é uma estrutura de dados composta por pares de campo e valor. Os valores dos campos podem incluir objetos, vetores e vetores de objetos e assim por diante, tão profundamente aninhados quanto você deseja. Os documentos do *MongoDB* são semelhantes aos objetos *JSON*, portanto, é fácil pensar neles como objetos JavaScript. Comparado a um objeto *JSON*, um documento do *MongoDB* oferece suporte não apenas aos tipos de dados primitivos - booleanos, números e *strings* - mas também a outros tipos de dados comuns, como datas, *timestamps*, expressões regulares e dados binários. (SUBRAMANIAN, 2019)

Os dados armazenados no *MongoDB* são agrupados dentro de uma *Collection*, que equivale às tabelas de bases de dados relacionais, Chodorow (2013) diz que além de agrupar documentos por coleção, o *MongoDB* agrupa coleções em bancos de dados. Uma única instância do *MongoDB* pode hospedar vários bancos de dados, cada um agrupando zero ou mais coleções.

2.4.9 Schemas

Schemas é um objeto *JSON* que define a estrutura e o conteúdo dos dados. Os *schemas* representam tipos de dados em vez de valores específicos. Isso inclui tipos primitivos, como *strings* e números, bem como tipos estruturais, como objetos e matrizes, onde se tem a possibilidade de combinar para criar *schemas* que representam tipos de objetos personalizados. (MONGODB, 2023)

3 METODOLOGIA

A pesquisa utilizada no trabalho foi qualitativa e exploratória, onde a partir da observação e levantamento dos requisitos gerais para atacar o problema da gestão de cardápio do RU em geral, utilizou-se uma entre as metodologias de desenvolvimento de sistemas para a implementação do referido sistema.

Para a resolução do problema proposto, primeiramente foram levantados os principais requisitos para a criação do sistema, mediante a observação do funcionamento atual da disponibilização do cardápio pelo RU e necessidades prementes não contempladas. Além disso, foi desenvolvido um questionário junto ao Departamento de Nutrição da UFES – *campus* Alegre, contendo várias perguntas sobre o perfil nutricional individual do usuário do RU. Este será incorporado ao aplicativo para levantar informações dos usuários para servir de referência à montagem do cardápio pelo RU.

O presente trabalho, do ponto de vista da natureza pode ser classificado como uma pesquisa exploratória aplicada. Do ponto de vista técnico esse trabalho pode ser classificado como um estudo de caso. As metodologias utilizadas consistem em coletar e analisar informações sobre um determinado setor institucional, a fim de estudar aspectos relacionados ao tema de gestão do cardápio e controle nutricional de um RU. Neste método foi buscada a aplicação prática dos conhecimentos para uma solução proposta que resultará no desenvolvimento de um aplicativo móvel que será apresentado no decorrer deste trabalho.

3.1 Arquitetura do Sistema

Nesta seção, será abordada a arquitetura utilizada no desenvolvimento do projeto, ilustrando a arquitetura cliente-servidor e apresentando detalhes sobre como as aplicações clientes estabeleceram comunicação com o servidor.

3.1.1 Desenho da Arquitetura

A arquitetura da aplicação segue o conceito geralmente utilizado em aplicativos desenvolvidos utilizando os *frameworks* ReactJS e React Native. Existe duas interfaces *front ends*[2], onde o primeiro *front end* da aplicação é o sistema *web* usando ReactJS e utilizado pelo setor de Nutrição RU para fornecer os dados do cardápio, verificar avaliação do cardápio e visualizar dados obtidos dos alunos. O segundo *front end* é a aplicação feita usando React Native e instalada nos dispositivos móveis do alunos, afim de responder o questionário inicial da aplicação, verificar o cardápio do dia e avaliar as refeições diárias.

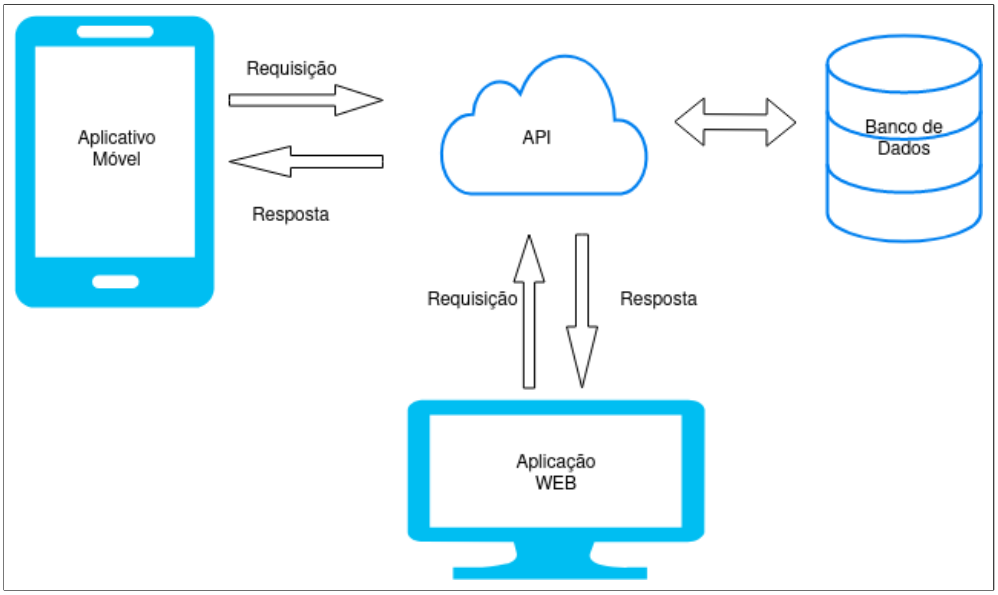


Figura 3: Arquitetura do Sistema
Fonte: (SEOBILITY, 2018)

Conforme a Figura 4, o primero *front end* é o aplicativo móvel e o segundo *front end* é a aplicação web, se comunicam com o *back end*[3] representado na figura a API, que foi feito utilizando a arquitetura *REST*, onde os clientes se comunica através de requisições HTTP, essas requisições serão tratadas pelo servidor que se comunica com o banco de dados.

[2]Interfaces de interação com o usuário
[3]Responsável pela regra de negócios

3.1.2 Delimitações do projeto

Apesar do *framework* React Native permitir a geração de aplicativos para diversas plataformas móveis, este projeto será restrito a construir a aplicação somente para a plataforma *Android*, pois para construir aplicações para *iOS* deve-se possuir o sistema operacional da Apple parar fazer a construção da aplicação. Já para a construções de aplicações *Android*, pode ser feita em qualquer sistema operacional.

3.2 Metodologia do Desenvolvimento

Os processos de desenvolvimento rápido de *software* são concebidos para produzir, rapidamente, *softwares* úteis. O *software* não é desenvolvido como uma única unidade, mas como uma série de incrementos — cada incremento inclui uma nova funcionalidade do sistema (SOMMERVILLE, 2011). O modelo do processo de desenvolvimento adotado nesse projeto, será o método Kanban.

Segundo TOTVS (2022), Kanban é um sistema visual de gestão de trabalho, que busca conduzir cada tarefa por um fluxo predefinido de trabalho. Em geral, o conceito de Kanban pode ser definido pelos seguintes itens:

- O sistema visual: um processo, definido em um quadro com colunas de separação, que permite dividir o trabalho em segmentos ou pelo seu status, fixando cada item em um cartão e colocando em uma coluna apropriada para indicar onde ele está em todo o fluxo de trabalho.
- Os cartões: que descrevem o trabalho real que transita por este processo.
- A limitação do trabalho em andamento: que permite atribuir os limites de quantos itens podem estar em andamento em cada segmento ou estado do fluxo de trabalho (TOTVS, 2022).

O trabalho das equipes que usam a técnica, gira em torno de um quadro kanban, uma ferramenta usada para visualizar o trabalho e otimizar o fluxo do trabalho. Ao contrário da implementação do kanban em um chão de fábrica, que envolveria mudanças nos processos físicos e a adição de materiais, as únicas coisas físicas que as equipes de *software* precisam são um quadro e cartões, que podem até mesmo ser virtuais (RADIGAN, 2023).

No geral Sabino (2023) indica que o sistema kanban possui 3 fases diferentes:

- *To Do* (A fazer): representa as tarefas ou itens que ainda estão por fazer. Essa é a primeira etapa do processo, na qual as demandas são registradas e priorizadas. As tarefas nesta fase podem estar esperando para serem iniciadas ou aguardando a disponibilidade de recursos. Essa coluna ou seção no quadro Kanban serve como um reservatório para futuras atividades.

- *Doing* (Em andamento): nessa etapa, as tarefas antes pertencentes à coluna *"To Do"*, são movidas para a coluna *"Doing"* e atribuídas aos membros da equipe. O trabalho então é iniciado e progride até sua conclusão. Essa coluna no quadro Kanban mostra as tarefas que estão sendo trabalhadas e em processo de execução. É importante limitar o número de tarefas em andamento, para evitar sobrecargas e manter o fluxo de trabalho equilibrado.

- *Done* (Concluído): quando uma tarefa é finalizada, ela é movida para a coluna *"Done"* no quadro Kanban. Essa etapa representa o progresso alcançado e o trabalho realizado. A coluna *"Done"* serve como um indicador visual das tarefas que foram concluídas com sucesso. É uma fonte de motivação para a equipe, pois permite uma visão clara das realizações.

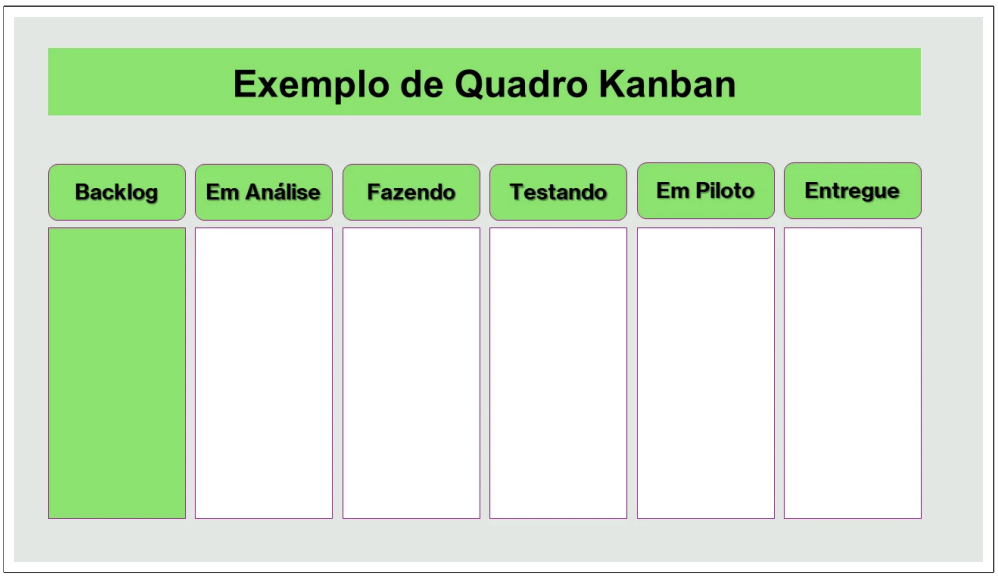


Figura 4: Exemplo de quadro kanban.
Fonte: Sabino (2023).

3.3 Analise dos requisitos

Entre as tarefas de um engenheiro de *software*, perceber os requisitos, ou seja, saber o que o cliente quer, é uma das etapas mais complexas do processo, tanto pela falta de informação, ou muitas vezes, informações tácitas, quanto falta de comunicação entre o cliente e o engenheiro, além do entendimento errado por parte do profissional ou explicação errada pelo lado do cliente (PRESSMAN, 2011). Ainda Pressman (2011) diz que a engenharia de requisitos o levantamento e entendimento dos requisitos, sendo uma ação da engenharia de *software* que se inicia durante a atividade de comunicação e que continua na modelagem. É necessário que se adapte às necessidades do projeto, produto e pessoas que fazem parte do trabalho.

3.3.1 Descrição dos Usuários

Os atores são os usuários de um sistema, ou melhor, um tipo de usuário. No aplicativo proposto, teremos o ator Setor Nutrição do RU, que representa as pessoas que irão se cadastrar rá cadastrar o cardápio do dia, o o Aluno que representa uma pessoa que irá utiliza o aplicativo.

3.3.2 Requisitos Funcionais

Segundo Sommerville (2011) Requisitos funcionais são declarações de serviços que o sistema deve fornecer, de como o sistema deve reagir a entradas específicas e de como o sistema deve se comportar em determinadas situações. Em alguns casos, os requisitos funcionais também podem explicitar o que o sistema não deve fazer. A seguir estão descritos os requisitos funcionais do sistema.

- RF01 - Exibir o cardápio da Semana - O aluno terá acesso ao cardápio do dia do RU.
- RF02 - Preencher o questionário inicial da aplicação - O aluno deve preencher um questionário elaborado pelo setor de nutrição do RU ao abrir a aplicação.
- RF03 - Avaliar o cardápio do dia - O aluno poderá avaliar o cardápio oferecido no dia.

- RF04 - Enviar comentários ou sugestões sobre o RU - O aluno poderá enviar comentários e sugestões sobre a qualidade da refeição.
- RF05 - Cadastrar cardápio do dia - O setor de nutrição cadastra o cardápio do dia.
- RF06 - Postar notícias e avisos - O setor de nutrição pode postar notícias e eventuais avisos.
- RF07 - Consultar os dados obtidos dos alunos - O setor de nutrição pode consultar os dados colhidos dos alunos ao preencher o questionário inicial.
- RF08 - Consultar avaliação do cardápio - O setor de nutrição poderá consultar as avaliações feitas pelos alunos do cardápio do dia.

3.3.3 Requisitos não funcionais

Segundo Sommerville (2011) Requisitos não funcionais são restrições a os serviços ou funções oferecidos pelo sistema. Incluem restrições de *timing*, restrições no processo de desenvolvimento e restrições impostas pelas normas. Ao contrário das características individuais ou serviços do sistema, os requisitos não funcionais, muitas vezes, aplicam-se ao sistema como um todo. A seguir estão descritos os requisitos não funcionais do sistema.

- RNF01 - A aplicação *web* deve ser totalmente compatível com os navegadores mais recentes, como *Google Chrome*, *Mozilla Firefox* e *Microsoft Edge*;
- RNF02 - A aplicação mobile deve ser desenvolvido para plataforma *Android* e compatível com a versão 5.0 (SDK 21) ou superior.
- RNF03 - A aplicação mobile deve ser otimizada para o desempenho específico do *Android*, garantindo uma experiência fluida em dispositivos com diferentes especificações.

3.3.4 Casos de Uso

Os casos de uso são documentados por um diagrama de casos de uso de alto nível. O conjunto de casos de uso representa todas as possíveis interações que serão descritas nos requisitos de sistema. Atores, que podem ser pessoas ou outros sistemas,

são representados como figuras palito. Cada classe de interação é representada por uma elipse. Os casos de uso identificam as interações individuais entre o sistema e seus usuários ou outros sistemas (SOMMERVILLE, 2011). A Figura 5 ilustra os casos de uso do sistema.

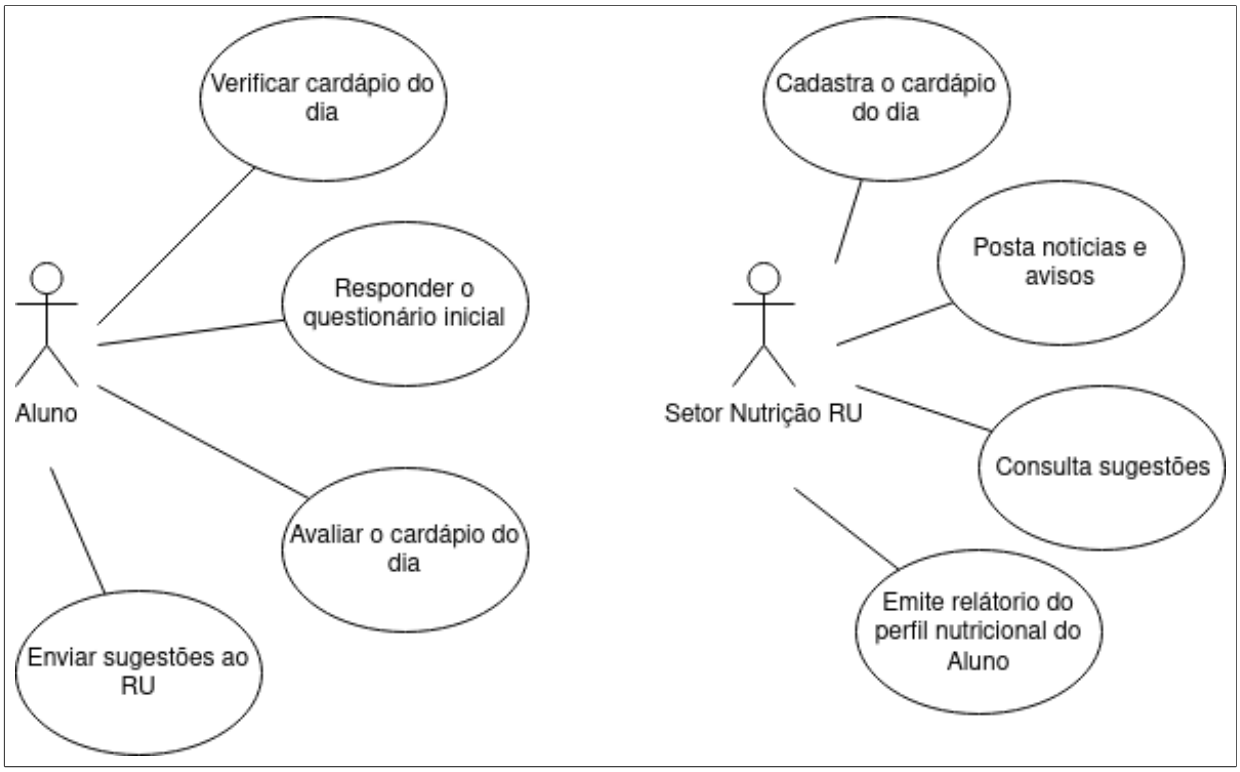


Figura 5: Diagrama de Caso de Uso.
Fonte: Do Autor.

Cenários e casos de uso são técnicas eficazes para elicitar requisitos dos cliente que vão interagir diretamente com o sistema. Cada tipo de interação pode ser representado como um caso de uso (SOMMERVILLE, 2011).

3.4 Modelagem Banco de Dados

O *MongoDB* usa um modelo de dados baseado em documentos binários chamado BSON, que é baseado no padrão *JSON*. Ao invés das tabelas de uma base relacional, o modelo de dados em documentos do Mongo é fortemente alinhado ao modelo de objetos usado nas linguagens de programação modernas e na maioria dos casos não há a necessidade de transações ou *joins*[4] complexos devido às vantagens de se ter

[4]combinação de conjunto de dados

os dados relacionados ao objeto dentro do próprio objeto (DUARTE, 2017).

Após analise dos requisitos e do questionário inicial da aplicação mobile, foi realizada a modelagem do banco de dados como segue:

A Figura 6 ilustra uma representação inicial do documento aluno no banco de dados, um conjunto de chave e valor, em que o valor nessa representação descreve o tipo de dados que a chave deve ter. Nesse documento estão descritos os dados que foram colhidos dos alunos definidos pelo questionário inicial (disponível no apêndice).

```
4   nome: {
5     type: String,
6     required: true,
7   },
8   email: {
9     type: String,
10    required: true,
11    unique: true,
12  },
13  hash_password: {
14    type: String,
15    required: true,
16    select: false,
17  },
18  matricula: {
19    type: Number,
20    required: true,
21    unique: true,
22  },
23  data_nascimento: {
24    type: String,
25    required: true,
26  },
27  curso: {
28    type: String,
29    required: true,
30  },
```

Figura 6: Representação do documento Aluno.
Fonte: Do Autor.

A Figura 7 ilustra a representação inicial do documento refeição no banco de dados, onde os contém os dados do cardápio do dia.

```
4     data: {
5       type: Date,
6       required: true,
7     },
8     tipo: {
9       type: String,
10      required: true,
11    },
12    entrada: {
13      type: String,
14      required: true,
15    },
16    prato_proteico: { type: String, required: true },
17    opcao: {
18      type: String,
19      required: true,
20    },
21    acompanhamento: { type: String, required: true },
22    guarnicao: {
23      type: String,
24      required: true,
25    },
26    sobremesa: { type: String, required: true },
27  });
```

Figura 7: Representação do documento refeição.

Fonte: Do Autor.

A Figura 8 ilustra a representação inicial do documento aviso no banco de dados, onde contém a descrição dos avisos que serão informados pelo setor do RU.

```
4     data: {
5       type: Date,
6       default: Date.now,
7     },
8     titulo: {
9       type: String,
10      required: true,
11    },
12    descricao: {
13      type: String,
14      required: true,
15    },
```

Figura 8: Representação do documento aviso.

Fonte: Do Autor.

A Figura 9 ilustra a representação inicial da avaliação de cardápio no banco de dados, onde contém os dados de avaliação de cada atributo do cardápio, juntamente com o comentário em questão.

```
4   {
5     student_id: {
6       type: Types.ObjectId,
7       required: true,
8     },
9     nome: { type: String },
10    avaliacao: {
11      entrada: { type: Number },
12      prato_proteico: { type: Number },
13      opcao: { type: Number },
14      acompanhamento: { type: Number },
15      guarnicao: { type: Number },
16      sobremesa: { type: Number },
17    },
18    comentario: { type: String },
19  },
20  { _id: false }
```

Figura 9: Representação do documento avaliação.

Fonte: Do Autor.

4 DESENVOLVIMENTO

Nesta seção, será detalhado o desenvolvimento do sistema, destacando as tecnologias utilizadas. Explorando o *backend*, a aplicação *web* e, por fim, a aplicação mobile, será definido e ilustrado cada aspecto do processo do sistema.

4.1 Metodologia do desenvolvimento

Após a conclusão da fase de levantamento de requisitos e a definição das tecnologias escolhidas, foram criados cartões no *kanban* para cada parte do sistema. Inicialmente, o desenvolvimento começou com os cartões de tarefas do *backend*, focando na definição das coleções do banco de dados para as seções de usuários, cardápios e notícias. *Endpoints* foram desenvolvidos para cada uma dessas partes.

Após a finalização dos *endpoints* do *backend*, iniciou-se ao desenvolvimento do sistema *web*. Seguindo a mesma ordem estabelecida para o *backend*, foram criadas interfaces e realizadas integrações com a API para cadastro de usuários, gerenciamento de cardápios e notícias. Após a conclusão e validação dessas seções, foram desenvolvidos os *endpoints* para comentários e avaliações dos cardápios, bem como os *endpoints* responsáveis por calcular estatísticas geradas a partir de questionários respondidos pelos alunos. Posteriormente, iniciou-se ao desenvolvimento das interfaces e integrações com a API para exibição dos comentários, avaliações e gráficos.

Por fim, as interfaces da aplicação mobile foram desenvolvidas, começando pelas de exibição do cardápio, avaliação do cardápio e envio de comentários, integrando-se aos *endpoints* da API. Após a conclusão destas funcionalidades, deu-se início ao desenvolvimento da interface de exibição de notícias e informações gerais. Para concluir o aplicativo mobile, foram criadas as interfaces de login, juntamente com aquelas relacionadas ao questionário. Dessa forma, o desenvolvimento do sistema como um todo foi finalizado.

4.2 Estrutura do projeto

Na Figura 10, é apresentada a estrutura de pacotes organizados por pasta no projeto. Nos próximos tópicos, serão fornecidas as definições correspondentes a cada uma dessas pastas.

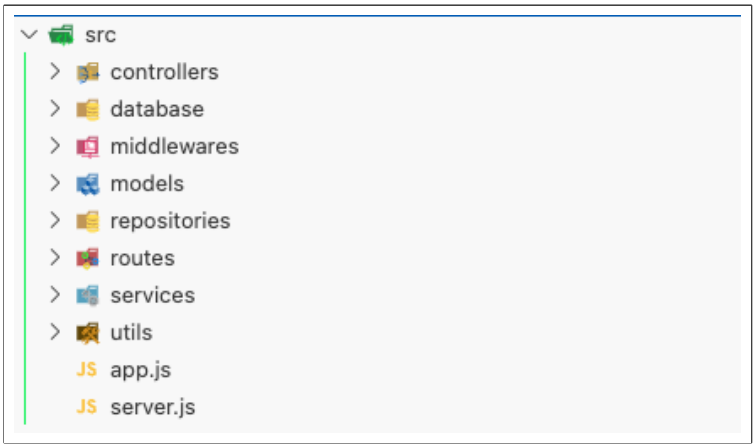


Figura 10: Estrutura do projeto
Fonte: Do Autor

Estrutura de pastas

- *controllers*: Classes responsáveis por tratar requisições HTTP;
- *database*: Arquivos responsáveis por tratar a conexão com o banco;
- *middlewares*: Funções responsáveis por interceptar e validar as requisições HTTP;
- *models*: Arquivos responsáveis por definir os schemas das coleções do banco de dados;
- *repositories*: Classes responsáveis pela comunicação com o banco de dados;
- *routes*: Funções responsáveis por definir as rotas da aplicação;
- *services*: Classes responsáveis por implementar os casos de uso;
- *utils*: Arquivos utilitários.

4.2.1 Conexão com o bancos de dados

Conforme apresentado na Figura 11, através da biblioteca *Mongoose* foi realizada a configuração da conexão com o banco de dados *MongoDB* (MONGODB, 2023). A conexão é obtida pela URL de conexão contendo a endereço do banco e os dados de acesso e configurações necessárias para funcionamento da biblioteca de comunicação com o banco.

```
1  const mongoose = require('mongoose');
2
3  module.exports = mongoose.connect(process.env.MONGO_URL, {
4    useNewUrlParser: true,
5    useUnifiedTopology: true,
6    useCreateIndex: true,
7  });
```

Figura 11: Conexão com o banco *MongoDB*.
Fonte: Do Autor.

4.2.2 Schemas

Foi adotada a prática de utilizar a biblioteca *Mongoose* para a definição dos *schemas* do banco de dados. A Figura 12 ilustra um exemplo de definição de *schema* seguindo essa abordagem.

```
1   const { Schema, model } = require('mongoose');
2
3   const menuSchema = new Schema({
4     data: {
5       type: Date,
6       required: true,
7     },
8     tipo: {
9       type: String,
10      required: true,
11    },
12    entrada: {
13      type: String,
14      required: true,
15    },
16    prato_proteico: { type: String, required: true },
17    opcao: {
18      type: String,
19      required: true,
20    },
21    acompanhamento: { type: String, required: true },
22    guarnicao: {
23      type: String,
24      required: true,
25    },
26    sobremesa: { type: String, required: true },
27  });
28
29  module.exports = model('menu', menuSchema);
```

Figura 12: Schema cardápio.
Fonte: Do Autor.

Na Figura 12 apresentada observa-se a definição a estrutura de um documento em uma coleção do banco de dados, representando uma entidade "cardápio". Ele possui os seguintes campos:

- data: Uma data obrigatória que indica a data do cardápio.
- tipo: Uma *string* obrigatória que especifica o tipo do cardápio (por exemplo, almoço ou jantar).
- entrada: Uma *string* obrigatória que descreve a opção de entrada no menu.

- prato proteico: Uma *string* obrigatória que representa o prato principal proteico no cardápio.
- opcao: Uma *string* obrigatória que indica uma opção adicional no cardápio.
- acompanhamento: Uma *string* obrigatória que descreve o acompanhamento no cardápio.
- guarnicao: Uma *string* obrigatória que representa a guarnição no cardápio.
- sobremesa: Uma *string* obrigatória que especifica a sobremesa no cardápio.

Este *schema* assegura que cada documento na coleção tenha esses campos e que todos sejam obrigatórios, garantindo a consistência dos dados armazenados.

No contexto do *Mongoose*, a validação é geralmente definida diretamente dentro do objeto de *schema*. No exemplo de código apresentado na Figura 12, o objeto *menuSchema* já possui validações incorporadas. Por exemplo:

- *required*: *true*: Indica que o campo é obrigatório, ou seja, cada documento deve ter um valor para esse campo. Se um documento for salvo sem esse campo, ele não será válido.
- *type*: Especifica o tipo de dados que deve ser armazenado no campo. Por exemplo, *type*: *Date* assegura que o campo data deve conter um valor do tipo *Date*, enquanto *type*: *String* assegura que os campos tipo, entrada, prato proteico, opcao, acompanhamento, guarnição e sobremesa devem conter valores do tipo *String*.

A partir disso, foram desenvolvidos os repositórios, que importam os *schemas* específicos e os utilizam para realizar interações com o banco de dados, como inclusões, exclusões e buscas. Na prática, os repositórios servem como uma camada de abstração entre a lógica de negócios e o banco de dados, simplificando a manipulação dos dados. A Figura 13 ilustra um exemplo concreto de implementação de um repositório para o *schema* de cardápio, demonstrando como os dados são persistidos e consultados de maneira eficiente e organizada.

```
4   class MenuRepository {
5     async create(menu) {
6       try {
7         const menuCreated = await menuModel.create(menu);
8
9         return menuCreated;
10      } catch (error) {
11        throw new DBError(error.message);
12      }
13    }
14
15    async list(skip, limit) {
16      try {
17        const menus = await menuModel
18          .find()
19          .skip(skip)
20          .limit(limit)
21          .sort({ _id: -1 });
22
23        return menus;
24      } catch (error) {
25        throw new DBError(error.message);
26      }
27    }
28
29    async findById(id) {
30      try {
31        const menus = await menuModel.findById(id);
32
33        return menus;
34      } catch (error) {
35        throw new DBError(error.message);
36      }
37    }
```

Figura 13: Exemplo de *repository* do cardápio.

Fonte: Do Autor.

Na Figura 13 pode-se observar as seguintes funções:

- A função *create(menu)*: Esta função cria um novo cardápio no banco de dados usando os dados fornecidos no objeto cardápio. Se bem sucedida, retorna o novo registro.
- A função *list(skip, limit)*: Esta função recupera uma lista de cardápios do banco

de dados. *skip* determina quantos registros pular, e *limit* define quantos registros retornar. Os cardápios são ordenados por *ID* e depois paginados.

- A função *findById(id)*: Esta função encontra um cardápio específico no banco de dados com base no *ID* fornecido. Retorna o registro se encontrado.

4.2.3 Middlewares

No projeto, foram desenvolvidas funções de *middlewares* para interceptar as requisições e validar seus corpos. Essas funções desempenham um papel importante ao verificar e garantir a integridade dos dados enviados nas requisições. Elas asseguram que as informações recebidas estejam corretas e formatadas adequadamente antes de serem processadas, contribuindo para a segurança e consistência das operações no sistema. A Figura 14 ilustra um exemplo de implementação de um *middlewares* de validações de um cardápio

```
4   const create = celebrate({
5     [Segments.BODY]: Joi.object().keys({
6       tipo: Joi.string()
7         .valid('Almoço', 'Jantar')
8         .required(),
9       entrada: Joi.string().required(),
10      prato_proteico: Joi.string().required(),
11      opcao: Joi.string().required(),
12      acompanhamento: Joi.string().required(),
13      guarnicao: Joi.string().required(),
14      sobremesa: Joi.string().required(),
15    }),
16  });
17
18  const update = celebrate({
19    [Segments.BODY]: Joi.object().keys({
20      tipo: Joi.string()
21        .valid('Almoço', 'Jantar')
22        .required(),
23      entrada: Joi.string().required(),
24      prato_proteico: Joi.string().required(),
25      opcao: Joi.string().required(),
26      acompanhamento: Joi.string().required(),
27      guarnicao: Joi.string().required(),
28      sobremesa: Joi.string().required(),
29    }),
30  });
31
32  const listAllMenus = celebrate({
33    [Segments.QUERY]: {
34      page: Joi.number(),
35    },
36  });
37
```

Figura 14: Exemplo de *middleware* de validação do cardápio.
Fonte: Do Autor.

Esse *middleware*, criado com a biblioteca *Celebrate*, foca no corpo das requisições, verificando se os dados enviados pelos clientes atendem a critérios específicos antes de serem processados pelo servidor.

Para realizar essa validação, o *middleware* utiliza a biblioteca *Joi* para criar um esquema de validação. No esquema, são definidos os campos que a requisição deve conter. Por exemplo, o campo tipo é esperado como uma *string*, mas não apenas qualquer *string*: ela deve ser ou 'Almoço' ou 'Jantar'. Além disso, esse campo é obrigatório, ou seja, precisa estar presente na requisição para que seja considerada válida. Da mesma forma, outros campos como *entrada*, *prato proteico*, *opcao*, *acompanhamento*, *guarnicao* e *sobremesa* também são esperados como *strings* obrigatórias.

4.2.4 Services

Os *services* são responsáveis por implementar os casos de usos, que é uma parte da lógica de negócios que representa uma única tarefa que o sistema precisa executar. O *service* encapsula as regras e a lógica necessárias para executar a tarefa e define as entradas e saídas necessárias para a operação (ALLIES, 2023).

```
4   class CreateMenuService {
5     async handle(menu) {
6       try {
7         const menuCreated = await menuRepository.create(menu);
8
9         return menuCreated;
10      } catch (error) {
11        throw new InternalServerError('Internal Server Error');
12      }
13    }
14  }
```

Figura 15: Exemplo de *service* de criação de um cardápio.
Fonte: Do Autor.

Na Figura 27 é uma classe de serviço que encapsula a lógica de criação de cardápios do sistema. Ele fornece um método chamado *handle* que recebe os dados do cardápio como entrada e, em seguida, utiliza o *repository* para persistir esses dados no banco de dados. Se a operação for bem sucedida, o serviço retorna o cardápio recém criado. No entanto, se ocorrer algum erro durante o processo, o serviço lança uma exceção indicando um erro interno do servidor. Essa abordagem ajuda a manter a organização e a separação de responsabilidades em um aplicativo, garantindo que a lógica de negócios relacionada à criação de menus seja tratada de maneira coesa e gerenciável.

4.2.5 Controllers

No projeto, foram utilizados (*controllers*) como intermediários entre as solicitações dos usuários e a manipulação de dados. Esses controladores desempenharam um papel importante ao lidar com as respostas aos usuários, garantindo uma entrega de solicitações feitas e as operações de manipulação de dados realizadas no sistema.

```
12  class MenuController {
13    async create(req, res) {
14      const data = Date.now();
15      const {
16        tipo,
17        entrada,
18        prato_proteico,
19        opcao,
20        acompanhamento,
21        guarnicao,
22        sobremesa,
23      } = req.body;
24
25      const menu = {
26        data,
27        tipo,
28        entrada,
29        prato_proteico,
30        opcao,
31        acompanhamento,
32        guarnicao,
33        sobremesa,
34      };
35
36      try {
37        const menuCreated = await createMenuService.handle(menu);
38        await createRatingColletionService.handle(menuCreated._id);
39        return res.status(200).json(menuCreated);
40      } catch (error) {
41        return res.status(error.statusCode).json({ error: error.message });
42      }
43    }
44
```

Figura 16: Exemplo de *controller*
Fonte: Do Autor

A Figura 16 representa uma *controller* que é responsável por lidar com uma requisição de criação de um novo cardápio no sistema. Quando uma solicitação é recebida, o *controller* extrai os dados relevantes do corpo da requisição, incluindo informações como tipo de refeição, pratos principais, acompanhamentos, entre outros. Em seguida, esses dados são agrupados em um objeto menu, que inclui também um *timestamp* representando o momento da criação. Após preparar os dados, o código interage com

o serviço chamado *createMenuService*. Esse serviço é responsável por lidar com a lógica de negócios relacionada à criação de cardápios. Se a criação do cardápio for bem sucedida, o serviço retorna o cardápio criado, e o código então chama outro serviço chamado *createRatingCollectionService*. Após essa etapa, o cardápio criado é enviado como resposta para o cliente que fez a requisição. No entanto, se ocorrer algum erro durante o processo de criação do cardápio, o código captura o erro e retorna uma resposta com um código de status correspondente ao erro ocorrido, juntamente com uma mensagem de erro.

4.2.6 Rotas

No projeto em questão foram implementados funções com as definições de rotas. Essas funções são responsáveis por expor o endereço da rota além definir quais são os *middlewares* responsáveis por interceptar as rotas e quais são os controladores responsável por processar a requisição.

```
8   const menuRoutes = Router();
9
10  menuRoutes.use(auth);
11
12  menuRoutes.get('/', menuValidator.listAllMenus, menuController.readAll);
13  menuRoutes.get('/last', menuController.readLast);
14  menuRoutes.get('/:id', idValidator, menuController.readOne);
15  menuRoutes.get(
16    '/:id/comentarios',
17    idValidator,
18    menuValidator.listComments,
19    ratingController.listComments
20  );
21  menuRoutes.get('/avg/:id', idValidator, ratingController.average);
22  menuRoutes.post('/', menuValidator.create, menuController.create);
23  menuRoutes.post(
24    '/avaliar/:id',
25    idValidator,
26    menuValidator.createRating,
27    ratingController.create
28  );
29  menuRoutes.put(
30    '/:id',
31    idValidator,
32    menuValidator.update,
33    menuController.update
34  );
35  menuRoutes.delete('/:id', idValidator, menuController.delete);
```

Figura 17: Exemplo de definição de rotas.
Fonte: Do Autor.

A Figura 17 é apresentada um trecho de código que define rotas para operações relacionadas as cardápios e avaliações de um cardápio. Essas rotas abrangem operações como leitura de todos os cardápios, do mais recente e de um específico por *ID*. Elas também permitem listar comentários de um cardápio, calcular a média de avaliações,

criar e atualizar cardápios e avaliações, além de excluir cardápios existentes. Cada rota é validada por *middlewares* e manipulada por controladores, garantindo um fluxo seguro e organizado de dados na aplicação.

4.3 Front end

O *Frontend* é a parte de um sistema com a qual os usuários interagem diretamente. Ele representa a interface do usuário, englobando todos os elementos visuais, interativos e experiências que os usuários encontram em um aplicativo ou site. Neste projeto específico, a biblioteca *React* foi utilizada para construir as interfaces. No ambiente *web*, foi criada uma *Single Page Application* (SPA), proporcionando uma experiência de navegação contínua e fluida. Além disso, para dispositivos móveis, o *frontend* foi desenvolvido usando o *framework React Native*, garantindo uma interface otimizada e nativa para dispositivos *Android*.

4.3.1 Estrutura do front end web

Na Figura 18, é apresentada a estrutura do projeto organizada por pastas. Nos próximos tópicos, serão fornecidas as definições correspondentes a cada uma dessas pastas.

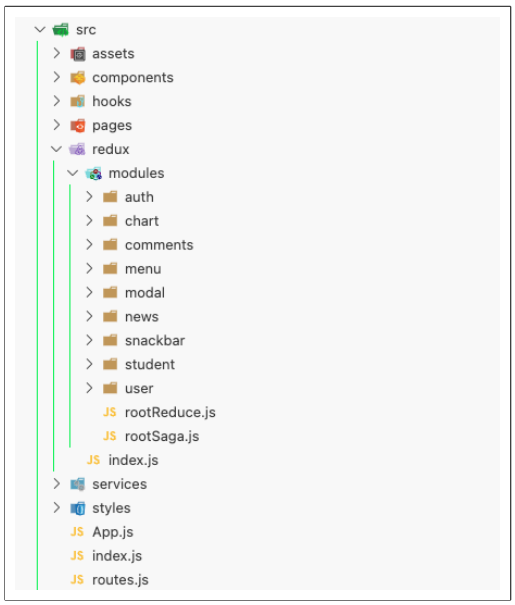


Figura 18: Estrutura do projeto web.

Fonte: Do Autor.

Estrutura de pastas do projeto *web*

- *assets*: Pastas com recursos estáticos da aplicação;
- *components*: Pastas com os componentes que são reutilizados no projeto;
- *hooks*: Pastas com os *hooks* customizados;
- *pages*: Pastas com as páginas da aplicação;
- *redux*: Pastas com as *stores* da aplicação;
- *services*: Pasta com o arquivo de instância do cliente HTTP;
- *styles*: Pasta com estilos globais da aplicação;
- *routes*: Arquivo com as rotas da aplicação.

4.3.2 Comunicação com o servidor

Nesse projeto utilizou-se de *Redux* para gerenciar os estados da aplicação e *Redux Saga* com *middleware* para gerenciar as chamadas HTTP.

A Figura 19 ilustra as *stores* da aplicação separadas por contexto. Cada módulo trata de um aspecto específico da aplicação, como cardápio, notícias e gráficos. Cada contexto é gerenciado por uma *store* individual, permitindo uma organização modular dos estados, facilitando a manutenção e a compreensão do fluxo de dados na aplicação.

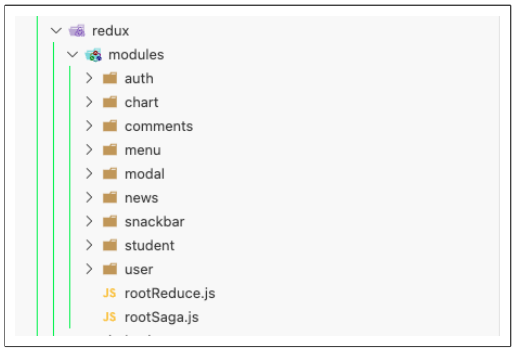


Figura 19: Módulos do *redux*.
Fonte: Do Autor.

Cada módulo contém três arquivos: um arquivo para definir as *actions*, um arquivo para o *reducer* e um arquivo para o saga. A Figura 20 apresenta as *actions* que serão

disparadas por páginas referente ao cardápio. Essas ações do *Redux* são criadas para buscar e gerenciar menus na aplicação. A ação de *fetchMenus* busca menus com base na página especificada, *fetchMenusSuccess* lida com o sucesso da busca, *fetchMenusFailure* trata falhas na busca e *fetchMenu* busca um menu específico pelo ID.

```
1   export const fetchMenus = page => {
2     return {
3       type: 'menu:FETCH_MENUS',
4       payload: { page },
5     };
6   };
7
8   export const fetchMenusSuccess = (menus, total) => {
9     return {
10      type: 'menu:FETCH_MENUS_SUCCESS',
11      payload: { menus, total },
12    };
13  };
14
15  export const fetchMenusFailure = () => {
16    return {
17      type: 'menu:FETCH_MENUS_FAILURE',
18    };
19  };
20
21  export const fetchMenu = id => {
22    return {
23      type: 'menu:FETCH_MENU',
24      payload: { id },
25    };
26  };
27
28  export const fetchMenuSuccess = menu => {
29    return {
30      type: 'menu:FETCH_MENU_SUCCESS',
31      payload: { menu },
32    };
33  };
```

Figura 20: *Actions* do cardápio.

Fonte: Do Autor.

A Figura 21 representa o interceptador de ações que gerencia as chamadas à *API* na

aplicação. A função *loadMenu* é ativada para buscar um cardápio específico com base no ID fornecido. Se a busca for bem sucedida, a ação *fetchMenuSuccess* é acionada, e uma solicitação adicional é enviada para calcular a média das avaliações para esse cardápio. Em caso de erro na busca, a ação *fetchMenuFailure* é acionada para lidar com a falha.

A função *loadMenus*, por sua vez, é utilizada para recuperar uma lista de cardápios com base no número da página fornecido. Se a busca for bem sucedida, a ação *fetchMenusSuccess* é acionada. Em caso de erro na busca, a ação *fetchMenusFailure* é acionada, seguida pela exibição de um alerta de erro na interface.

```
23  function* loadMenu({ payload }) {
24    try {
25      const { id } = payload;
26      const result = yield call(api.get, `/cardapio/${id}`);
27      yield put(fetchMenuSuccess(result.data));
28      yield put(fetchMenuAverage(result.data._id));
29    } catch (error) {
30      yield put(fetchMenuFailure());
31    }
32  }
33
34  function* loadMenus({ payload }) {
35    try {
36      const { page } = payload;
37      const result = yield call(api.get, `/cardapio?page=${page}`);
38
39      yield put(fetchMenusSuccess(result.data.menus, result.data.total_menus));
40    } catch (error) {
41      yield put(fetchMenusFailure());
42      yield put(snackBarFailure('Erro ao carregar os cardápios!'));
43    }
44  }
```

Figura 21: *Saga* do cardápio
Fonte: Do Autor

A Figura 22 apresenta o *reducer* que gerencia o estado dos cardápios na aplicação. Este *reducer*, encarregado de gerenciar o estado dos cardápios, opera em diversas frentes. Quando uma busca por cardápios é iniciada, ele atualiza o estado para refletir o carregamento dos dados, indicando também a página em questão. Se essa busca for bem sucedida, o *reducer* é responsável por atualizar o estado com a lista de cardápios e o total de cardápios disponíveis, marcando que a operação foi concluída com êxito.

```
27  const menu = (state = intialState, action) => {
28    switch (action.type) {
29      case 'menu:FETCH_MENUS':
30        return produce(state, draftState => {
31          draftState.page = action.payload.page;
32          draftState.loadingData = true;
33        });
34
35      case 'menu:FETCH_MENUS_SUCCESS':
36        return produce(state, draftState => {
37          draftState.list = action.payload.menus;
38          draftState.total = action.payload.total;
39          draftState.loadingData = false;
40        });
41      case 'menu:FETCH_MENUS_FAILURE':
42        return {
43          ...state,
44          loadingData: false,
45        };
```

Figura 22: *Reducer* do cardápio

Fonte: Do Autor

Em situações de falha na busca por cardápios, o *reducer* entra em ação novamente, revertendo o indicador de carregamento para falso e indicando que a operação não foi bem sucedida. Além disso, quando a aplicação precisa buscar detalhes específicos de um cardápio, o *reducer* atualiza o estado para refletir o carregamento desses dados. Se essa busca for concluída com sucesso, o *reducer* atualiza o estado com os detalhes do cardápio em questão, garantindo que o estado da aplicação esteja sempre preciso e pronto para fornecer a melhor experiência ao usuário.

A Figura 23 representa uma página do sistema *web*, onde a ação de carregar os cardápios é acionada automaticamente ao ser renderizada. Quando essa operação é bem sucedida, os cardápios são armazenados na *store* e exibidos na página. Além disso, a página possui funcionalidades para deletar um cardápio, e obtém dados da *store*, como a listagem de cardápios e o status de carregamento.

```
20   export default function List() {
21     const [idDeleteMenu, setIdDeleteMenu] = useState(0);
22     const [actualPage, setActualPage] = useState(0);
23
24     const dispatch = useDispatch();
25     const { list: menus, loadingData, loadingDelete, total } = useSelector(
26       state => state.menu
27     );
28
29     useEffect(() => {
30       dispatch(fetchMenus(actualPage + 1));
31     }, [actualPage, dispatch]);
32
33     const handleChangePage = (event, nextPage) => {
34       setActualPage(nextPage);
35     };
36
37     const DeleteMenu = () => {
38       dispatch(deleteMenu(idDeleteMenu));
39     };
40
41     const handleButtonDelete = id => {
42       dispatch(openModal());
43       setIdDeleteMenu(id);
44     };
45
46     return (
47       <S.Container>
48         <S.TableContainer component={Paper}>
49           <Table aria-label="tabela de cardapios">
50 >           <S.TableHead>···
57             </S.TableHead>
58             <S.TableBody>
59 >             {menus.length > 0 ? (···
99               ) : (
100                <S.TableRow loading={loadingData ? 1 : 0}>
101                  <TableCell>
102                    <h1>Vazio</h1>
103                  </TableCell>
104                </S.TableRow>
105              )}
106            </S.TableBody>
107          </Table>
```

Figura 23: Página de listagem de cardápios
Fonte: Do Autor

4.3.3 Rotas

Nesta seção será abordado as rotas da aplicação *web*. Na Figura 24 exibe a configuração das rotas da aplicação utilizando a biblioteca de roteamento do *React*, conhecida como *React Router*.

O *Route* é um componente funcional que define as rotas da aplicação. O *array* de *routes* é um *array* de objetos que contem informações sobre as rotas da aplicação. Cada objeto tem as seguintes propriedades:

- *path*: O caminho da URL que corresponde à rota.
- *component*: O componente *React* que deve ser renderizado quando a rota é acessada.

- *key*: Uma chave única para identificar a rota.
- *exact*: Um *booleano* que indica se o caminho deve ser correspondido exatamente (*true*) ou não (*false*).

Já o *Switch* é componente do *React Router* que renderiza a primeira rota que corresponde ao local atual da URL. Para cada objeto no *array routes*, um componente *Route* é renderizado. Cada *Route* especifica um *path*, *component*, e se deve corresponder exatamente. Quando o caminho da URL corresponde ao *path* especificado em uma *Route*, o componente correspondente é renderizado.

Por exemplo, quando o usuário acessa "/"na URL, o componente Login é renderizado. Se o usuário acessar *"/register"*, o componente *Register* é renderizado. Para qualquer URL que comece com *"/dashboard"*, o componente *Layout* é renderizado, mas apenas se o usuário estiver autenticado. Se a *URL* não corresponder a nenhuma rota especificada, o componente *Page404* é renderizado. Este sistema de roteamento permite a navegação entre diferentes partes da aplicação com base na URL acessada pelo usuário.

```
10  const Routes = () => {
11    const routes = [
12      {
13        path: '/',
14        component: Login,
15        key: 'login',
16        exact: true,
17      },
18      {
19        path: '/register',
20        component: Register,
21        key: 'register',
22        exact: true,
23      },
24      {
25        path: '/dashboard*',
26        component: PrivateRoute(Layout),
27        key: 'main',
28        exact: true,
29      },
30      {
31        path: '*',
32        component: Page404,
33        exact: false,
34        key: '404',
35      },
36    ];
37
38    return (
39      <Switch>
40        {routes.map(route => (
41          <Route
42            key={route.key}
43            path={route.path}
44            exact={route.exact}
45            component={route.component}
46          />
47        ))}
48      </Switch>
49    );
50  };
```

Figura 24: Rotas da aplicação web

Fonte: Do Autor

4.3.4 Estrutura do frontend Mobile

Na Figura 25, é apresentada a estrutura do projeto organizados por pasta. Nos próximos tópicos, serão fornecidas as definições correspondentes a cada uma dessas pastas.

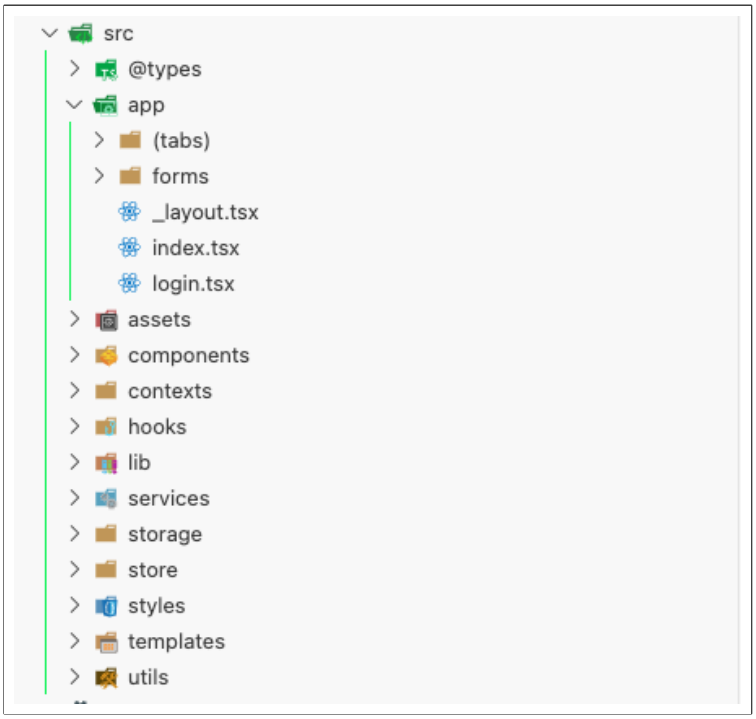


Figura 25: Estrutura do projeto mobile.
Fonte: Do Autor.

Estrutura de pastas

- *@types*: Pasta com utilitários de tipos;
- *app*: Pasta com as rotas do projeto;
- *assets*: Pasta com arquivos estáticos do projeto mobile
- *components*: Pastas com os componentes que são reutilizados no projeto;
- *contexts*: Pastas com arquivos de contextos fornecidos pelas *libs* utilitarias;
- *hooks*: *Custom hooks* da aplicação mobile;
- *lib*: Pasta com bibliotecas utilitárias;

- *services*: Pasta com funções responsáveis por comunicação com o servidor;
- *storage*: Pasta com utilitários de acesso ao *storage* do dispositivo;
- *store*: Pasta com a *store* da aplicação;
- *styles*: Pasta com utilitários de estilo da aplicação;
- *templates*: Pasta contendo as interfaces da aplicação mobile;
- *utils*: Pasta com funções utilitárias.

4.3.5 Comunicação com o servidor

A comunicação com o servidor é realizada por meio de requisições HTTP, sendo facilitada por pequenas funções desenvolvidas para executar essas chamadas e retornar os dados necessários. A Figura 26 apresenta os arquivos relacionados aos *services* da aplicação, organizados por contexto. Cada contexto representa uma área específica da aplicação e possui suas próprias funções de serviço para lidar com as interações com o servidor.

Figura 26: Pasta contendo os *services* do projeto mobile.
Fonte: Do Autor.

Na Figura 27, duas funções são definidas para interações com o servidor. A função *getLastMenu* realiza uma requisição para obter dados do último cardápio cadastrado no servidor e valida o formato dos dados antes de retorná-los. Já a função *getMenuRating* recebe um ID de cardápio como parâmetro, faz uma requisição para obter as avaliações relacionadas a esse ID de cardápio e também valida o formato dos dados antes de retorná-los. Ambas as funções garantem que os dados recebidos do servidor estejam corretamente estruturados para uso no aplicativo.

```
51  export async function getLastMenu() {
52    const { data } = await api.get('/cardapio/last');
53    return menuSchema.parse(data);
54  }
55
56  export async function getMenuRating(id: string) {
57    const { data } = await api.get(`/cardapio/avg/${id}`);
58    return ratingSchema.parse(data);
59  }
```

Figura 27: Funções de requisições HTTP relacionadas ao cardápio.
Fonte: Do Autor.

Foram desenvolvidos *hooks* customizados para cada interface que necessita recuperar dados do servidor. Esses *hooks* encapsulam a lógica de consulta e gerenciamento de estado, proporcionando uma abstração conveniente para as operações de busca de dados em cada interface da aplicação. Dessa forma, a obtenção de dados do servidor é simplificada e reutilizável, promovendo uma estrutura mais organizada e eficiente. Na Figura 28 ilustra duas funções de *hook* customizados são definidas para facilitar a obtenção de dados relacionados a cardápios e avaliações.

A função *useMenu* utiliza o *hook useQuery* do *React Query* para realizar uma consulta aos dados do último cardápio cadastrado no servidor. Ela recebe um objeto de configuração que inclui a chave da consulta, a função que efetua a requisição e uma função de seleção que transforma os dados recebidos do servidor em um formato desejado.

A função *useRating* também utiliza o *hook useQuery* e recebe um ID de cardápio como parâmetro. Ela configura a chave da consulta para incluir o ID recebido, habilitando a consulta apenas se um ID válido estiver presente. Quando ativada, a função *queryFn* realiza uma chamada à função *getMenuRating*, obtendo as avaliações relacionadas a esse ID do servidor.

Ambas as funções *useMenu()* e *useRating(id: string)* encapsulam a lógica de consulta e gerenciamento de estado relacionada a cardápios e avaliações, tornando mais fácil o uso desses dados em componentes *React*.

```
12  export function useMenu() {
13    return useQuery({
14      queryKey: [QueryKeys.lastMenu],
15      queryFn: getLastMenu,
16      select: React.useCallback(
17        (menuData: MenuData): MenuDataDTO => ({
18          ...menuData,
19          id: menuData._id,
20          pratoProteico: menuData.prato_proteico,
21        }),
22        []
23      ),
24    });
25  }
26
27  export function useRating(id: string) {
28    return useQuery({
29      queryKey: [QueryKeys.rating, id],
30      queryFn: () => getMenuRating(id),
31      enabled: !!id,
32    });
33  }
```

Figura 28: *Hooks* customizados relacionados ao cardápio.
Fonte: Do Autor.

Na Figura 29 ilustra o uso do *hook* customizados. Nessa interface, o *hook* customizado *useMenu* é utilizado para obter os dados do servidor relacionados ao cardápio. O *hook useMenu* encapsula a lógica de consulta à API e gerenciamento de estado, permitindo que a interface de exibição de cardápio se concentre na exibição dos dados sem se preocupar com detalhes de implementação de requisições e gerenciamento de carregamento.

Quando a interface é carregada, o *hook useMenu* é chamado, e o componente espera até que a consulta seja concluída (*isLoading* seja falso) antes de exibir o conteúdo. Durante o carregamento (*isLoading* sendo verdadeiro), é exibido um componente de carregamento (*HomeSkeleton*), proporcionando uma experiência visual agradável para o usuário enquanto aguarda os dados.

Assim, os *hooks* customizados ajudam a abstrair a complexidade das operações assíncronas e a centralizar a lógica de busca de dados, facilitando a manutenção e reutilização do código. Eles permitem que a lógica de obtenção de dados seja separada do componente visual, tornando o código mais limpo, modular e fácil de entender.

```
13  export default function Home() {
14    const { data, isLoading, isError, refetch } = useMenu();
15
16  >  function _formatDate(date: string) {…
19    }
20
21    if (isLoading) {
22      return <HomeSkeleton />;
23    }
24
25  >  if (isError) {…
33    }
34
35    if (!data) {
36      return <Empty />;
37    }
38
39    return (
40      <ScrollView
41        contentContainerStyle={styles.gap}
42        className="flex flex-1 bg-gray-50 p-4"
43      >
44        <Text className="text-center text-base">{_formatDate(data.data)}</Text>
45        <MenuItem image="entrada" title={data.entrada} description="Entrada" />
46        <MenuItem
47          image="proteico"
48          title={data.pratoProteico}
49          description="Prato proteico"
50        />
51        <MenuItem image="opcao" title={data.opcao} description="Opção" />
52        <MenuItem
53          image="acompanhamento"
54          title={data.acompanhamento}
55          description="Acompanhamento"
56        />
57        <MenuItem
58          image="guarnicao"
59          title={data.guarnicao}
60          description="Guarnicao"
61        />
62        <MenuItem
63          image="sobremesa"
64          title={data.sobremesa}
65          description="Sobremesa"
```

Figura 29: Código da interface de exibição do cardápio.

Fonte: Do Autor.

4.3.6 Rotas

Nesta seção, serão discutidas as rotas da aplicação mobile. A Figura 30 representa a estrutura de pastas das rotas. No projeto, foi adotada a organização de pastas como estrutura de roteamento da aplicação, onde cada arquivo corresponde a uma interface específica do aplicativo mobile. Essa abordagem facilita a navegação e o entendimento da arquitetura do aplicativo.

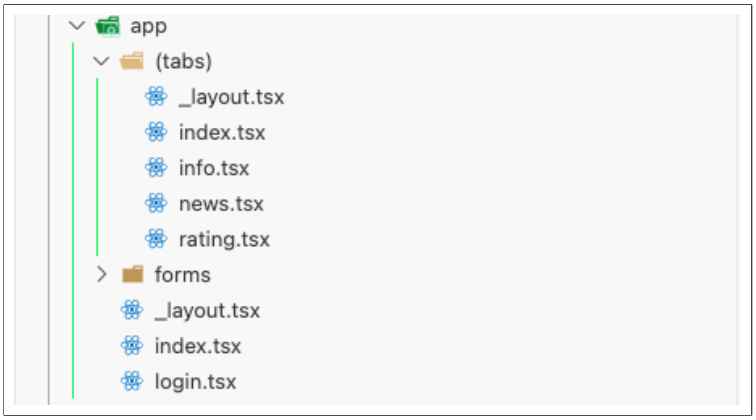


Figura 30: Rotas da aplicação mobile.
Fonte: Do Autor.

Na Figura 31 é definido o *layout* principal da interface inicial do aplicativo mobile. A função *HomeLayout* retorna um componente que representa a estrutura de navegação do aplicativo.

Dentro do componente *Stack*, que é uma parte da biblioteca de navegação utilizada no *React Native*, são definidas as interfaces iniciais do aplicativo. Cada interface é representada por um arquivo separado na estrutura de pastas do projeto. A interface de login é representada pelo arquivo "login", a interface de formulários por pela pasta *"forms"* e as menu inferior do aplicativo pela pasta *"(tabs)"*.

Essa organização de arquivos por pastas facilita a manutenção do código, tornando-o mais modular e fácil de entender. Cada interface tem seu próprio arquivo e, quando necessário, pode-se adicionar mais interfaces simplesmente criando novos arquivos dentro da estrutura de pastas correspondente. O uso do componente *Stack* permite a navegação entre essas interfaces de forma intuitiva.

```
8   export default function HomeLayout() {
9     return (
10      <SafeAreaView className="flex-1">
11        <StatusBar style="dark" />
12        <AppProvider>
13          <Stack
14            initialRouteName="login"
15            screenOptions={{
16              headerShown: false,
17            }}
18          >
19            <Stack.Screen name="login" />
20            <Stack.Screen name="forms" />
21            <Stack.Screen name="(tabs)" />
22          </Stack>
23        </AppProvider>
24      </SafeAreaView>
25    );
26  }
```

Figura 31: Arquivo de rota.

Fonte: Do Autor.

5 RESULTADOS

Nesta seção, será apresentado os resultados do sistema, destacando as interfaces desenvolvidas e as funcionalidades de cada interface. Será ilustrado como o sistema opera, proporcionando uma visão detalhada das interfaces e explicando as diversas funcionalidades incorporadas em cada uma delas.

5.0.1 Interfaces do sistema web

Nesta subseção, serão exibidas as interfaces do sistema *web*, apresentando uma demonstração abrangente de cada uma. Este percurso seguirá o fluxo desde o login na aplicação até as interface de estatísticas gerais, demonstrando os objetivos de cada interface ao longo desse caminho.

5.0.1.1 Login

Na Figura 32, é apresentada a interface de login do sistema, representando a primeira interação do usuário com a plataforma. Nesse cenário, a autenticação é realizada por meio do fornecimento do *email* e senha associados à conta. A interface oferece dois botões de opções: um para efetuar o login após inserir as credenciais e outro para direcionar o usuário à interface de cadastro. Ao selecionar a opção de cadastro, o usuário é redirecionado para a interface onde pode criar uma nova conta no sistema.

Figura 32: Interface de login.
Fonte: Do Autor.

5.0.1.2 Cadastro

Na Figura 33, é exibida a interface de cadastro no sistema, onde os usuários são solicitados a inserir seu nome, senha e endereço de *email*. Após um cadastro bem sucedido, o sistema redireciona automaticamente o usuário para a interface de login.

Figura 33: Interface de cadastro.
Fonte: Do Autor.

5.0.1.3 Interface inicial

Após realizar o login com sucesso, o sistema redireciona os usuários para a interface inicial da aplicação. Na Figura 35, é possível visualizar uma seção exibindo o cardápio do dia, acompanhado das respectivas avaliações feitas pelos alunos. Além disso, há uma lista de comentários feitos pelos alunos sobre o cardápio do dia.

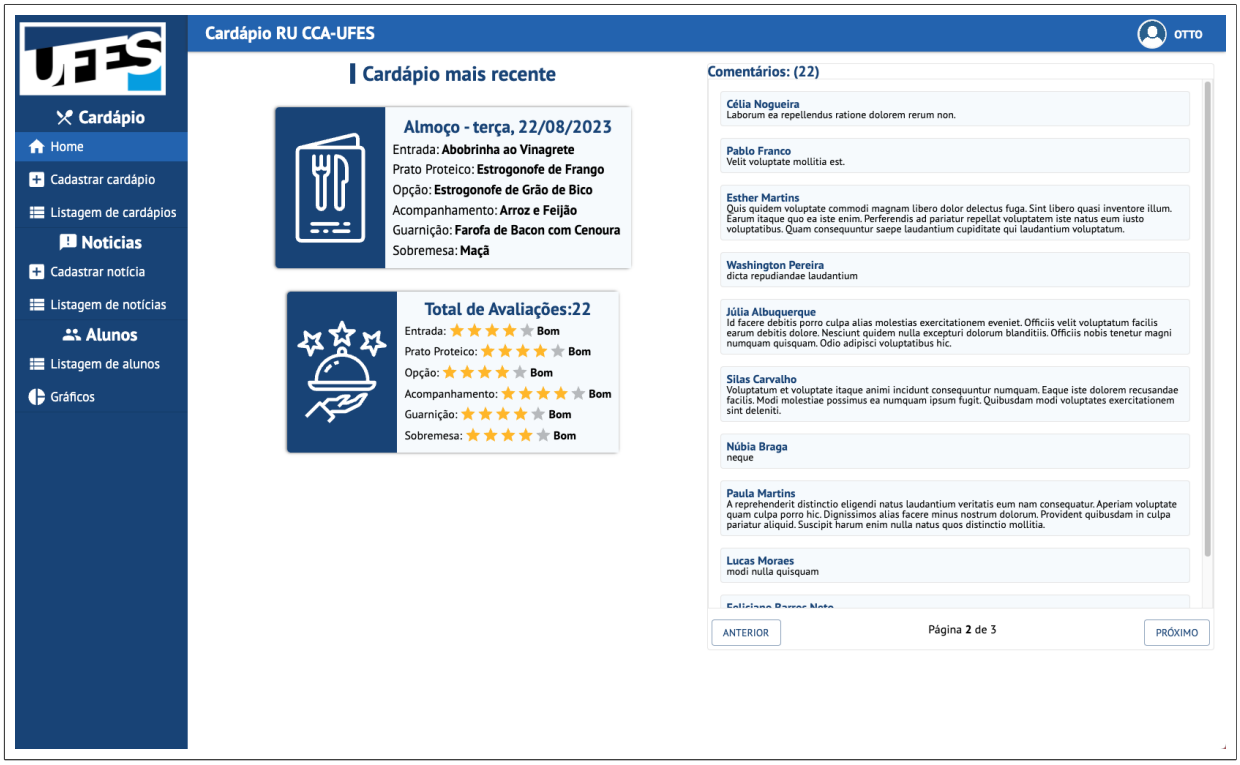


Figura 34: Interface inicial.

Fonte: Do Autor.

Na Figura 35, são apresentadas as opções para gerenciar cardápios, administrar notícias e visualizar gráficos.

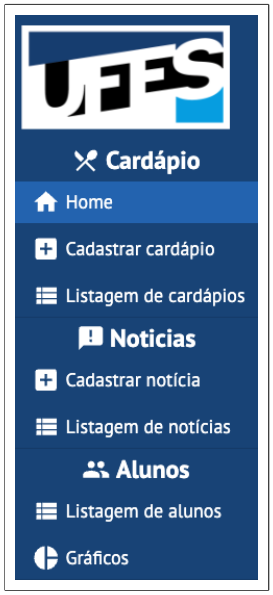


Figura 35: Menu lateral.

Fonte: Do Autor.

5.0.2 Cadastro de cardápio

Na interface de cadastro de cardápio, representada na Figura 36, os usuários têm a capacidade de registrar o cardápio do dia. Após concluir o processo de cadastro, o sistema redireciona automaticamente para a interface inicial.

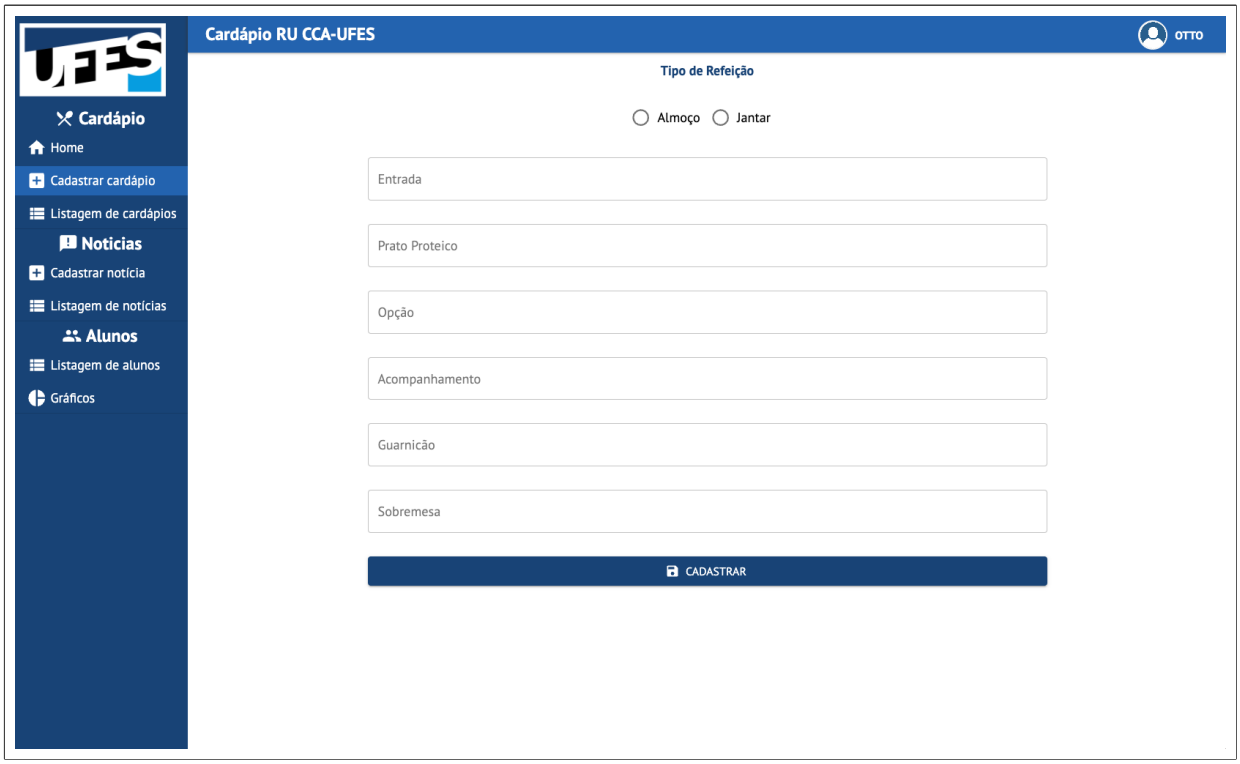


Figura 36: Interface de cadastro de cardápio.
Fonte: Do Autor.

5.0.2.1 Listagem de cardápios

A interface de listagem de cardápios, representada pela Figura 37, desempenha um papel importante no gerenciamento das opções de cardápios. Nessa interface, os usuários têm a opção de visualizar detalhes específicos de um cardápio, bem como editar ou excluir cardápios individuais. Essa funcionalidade oferece um controle preciso sobre as informações dos cardápios, permitindo ajustes conforme necessário.

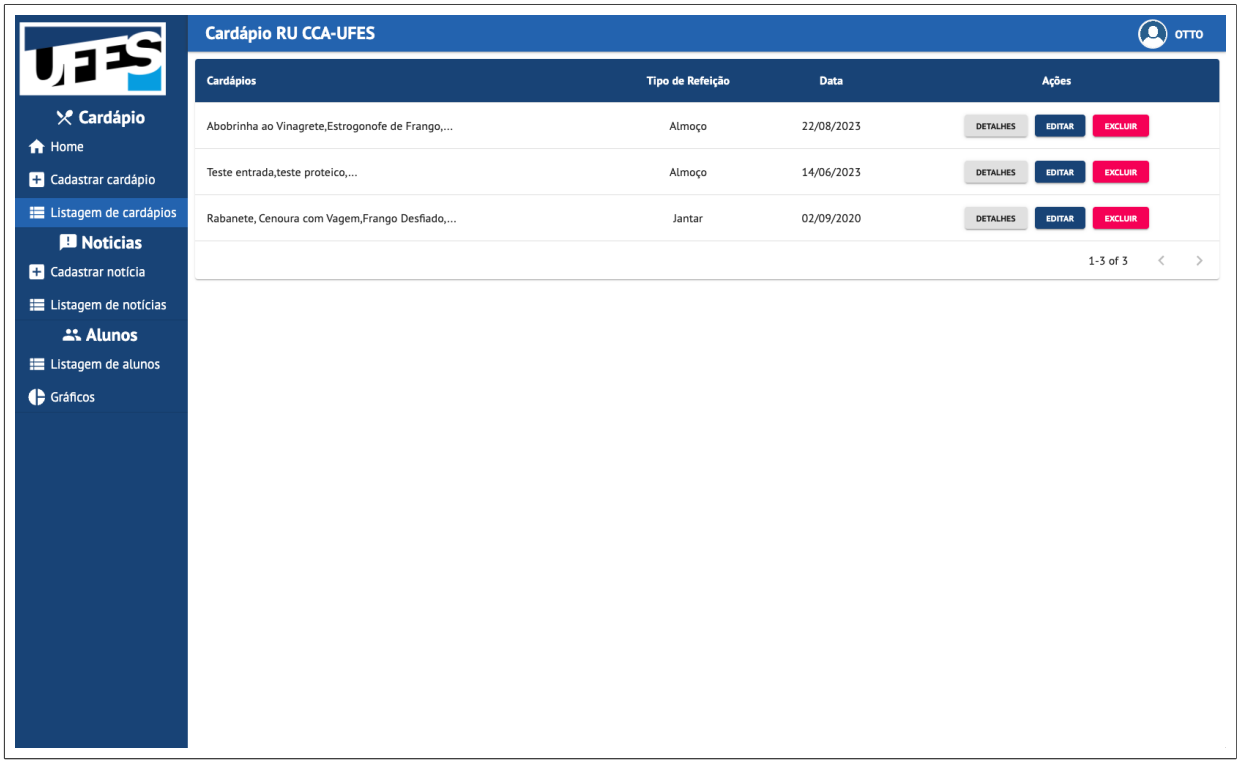


Figura 37: Interface de listagem de cardápios.
Fonte: Do Autor.

5.0.2.2 Cadastro de notícias

Na Figura 38, é apresentada a interface de cadastro de notícias e avisos que serão exibidos na aplicação móvel. Nessa interface, os usuários têm acesso a uma variedade de opções de formatação de texto, permitindo uma personalização detalhada do conteúdo.

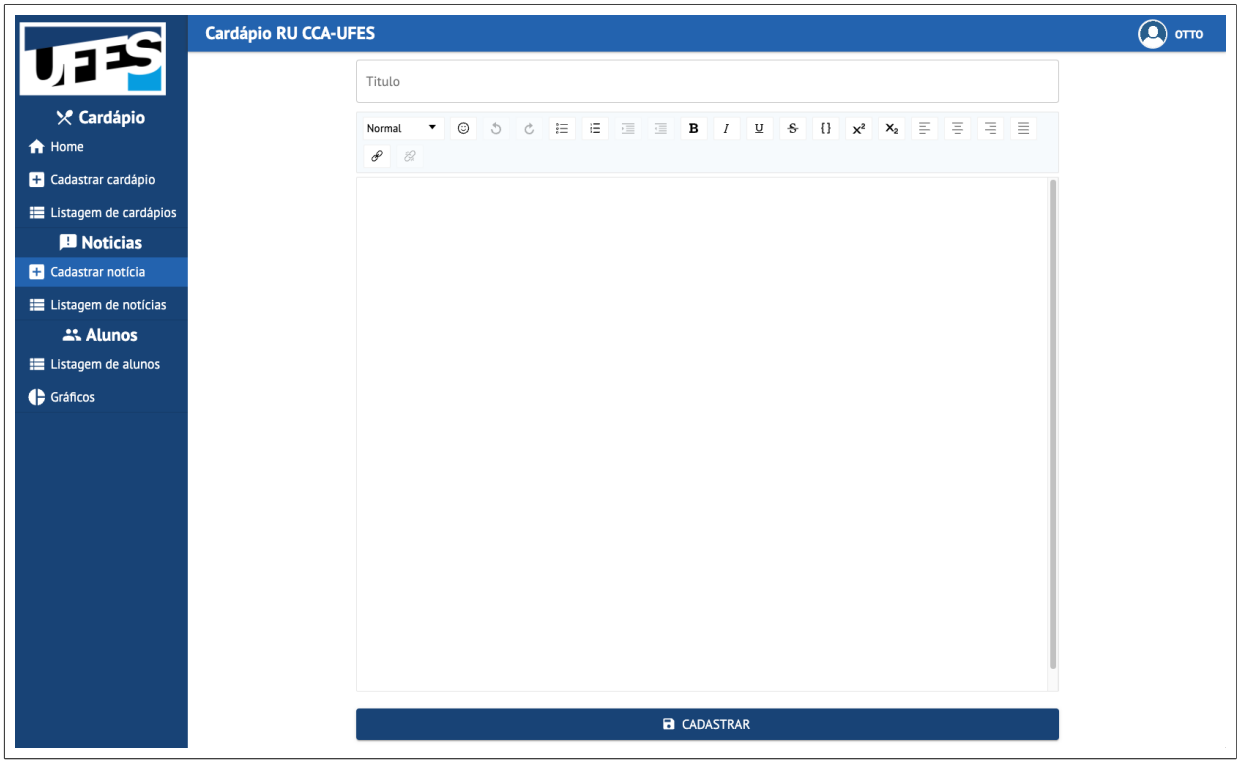


Figura 38: Interface de cadastro de noticia.
Fonte: Do Autor.

5.0.2.3 Listagem de notícias

A interface de listagem de noticias, representada pela Figura 39, é uma importante interface para o gerenciamento das opções de noticias. Nesse espaço, os usuários têm a opção de visualizar detalhes de uma noticias, bem como editar ou excluir noticias.

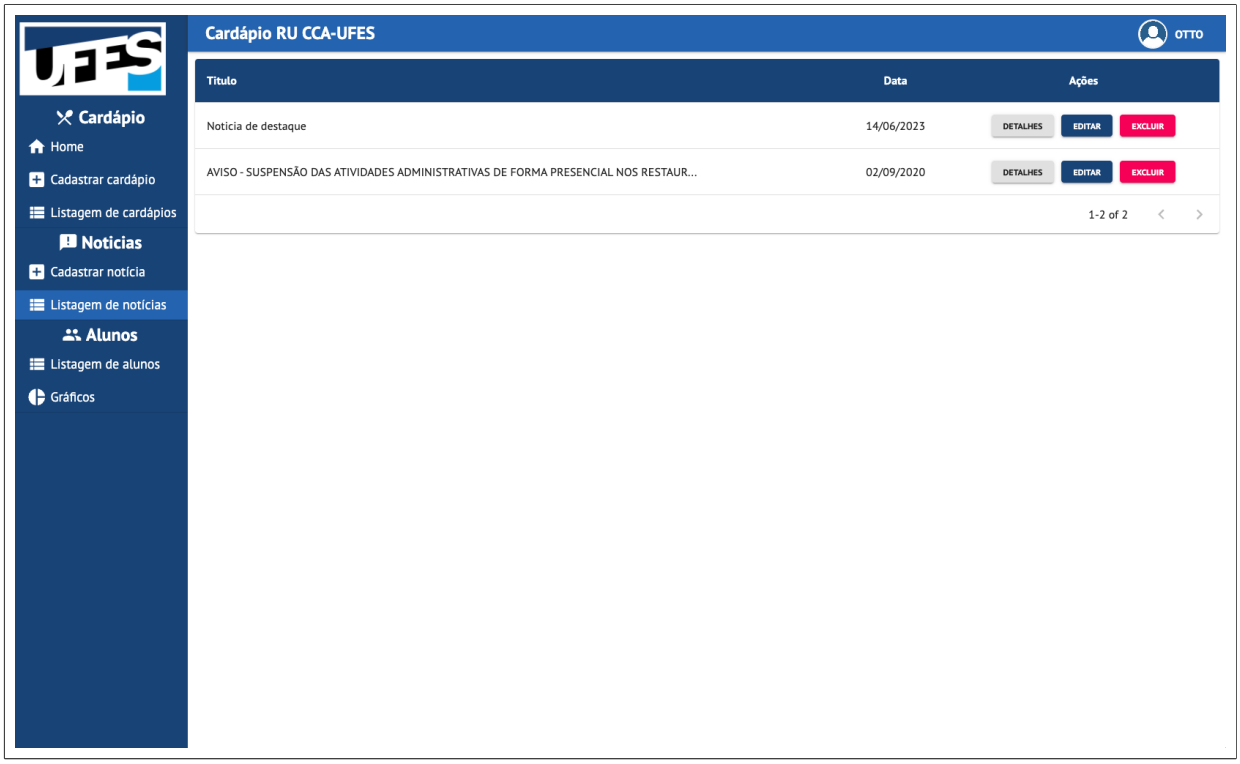


Figura 39: Interace de listagem de noticias.
Fonte: Do Autor.

5.0.2.4 Interface de gráficos

A Figura 40 ilustra a interface de gráficos gerados a partir das respostas dos alunos ao preencherem questionários na aplicação móvel. Nessa interface, os usuários podem visualizar uma variedade de estatísticas apresentadas por meio de gráficos de pizza, barras e outros formatos. Isso oferece uma compreensão visual das avaliações, permitindo uma análise abrangente das estatísticas gerais. A interface proporciona uma visão detalhada dos resultados, facilitando a interpretação das respostas dos alunos de forma clara.

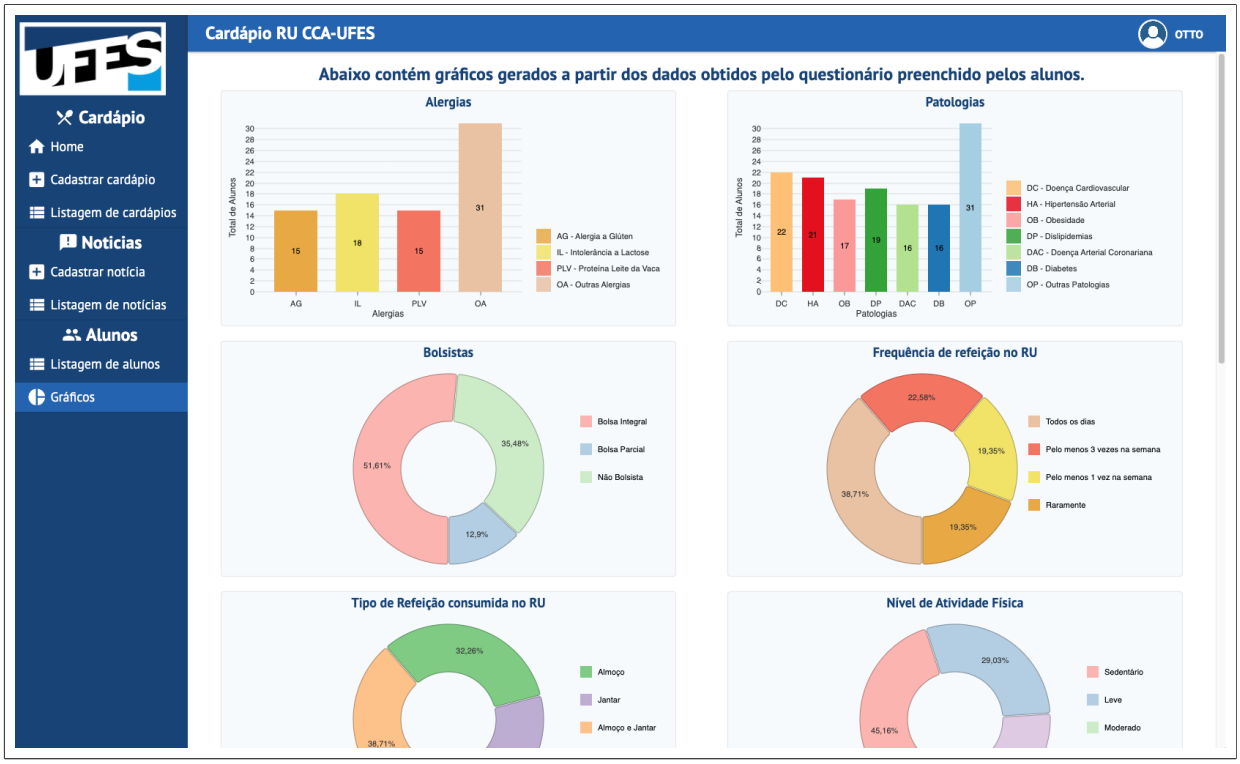


Figura 40: Interface de gráficos.

Fonte: Do Autor.

5.0.2.5 Interfaces da aplicação mobile

Nesta seção, serão apresentadas as interfaces da aplicação mobile, proporcionando uma visão abrangente de cada uma. Este percurso abrangerá desde o processo de cadastro e preenchimento do questionário de nutrição até as interfaces de informações gerais, oferecendo uma compreensão completa do fluxo.

5.0.2.6 Interface de login

Na Figura 41, é exibida a interface de login do aplicativo, marcando a primeira interação do usuário com a plataforma móvel. Nesse contexto, a autenticação é realizada ao fornecer o *email* e senha associados à conta do usuário. A interface inclui dois botões: um para efetuar o login após inserir as credenciais e outro para direcionar o usuário à interface de cadastro e preenchimento do questionário. Além disso, há um terceiro botão que leva o usuário à interface inicial do aplicativo.

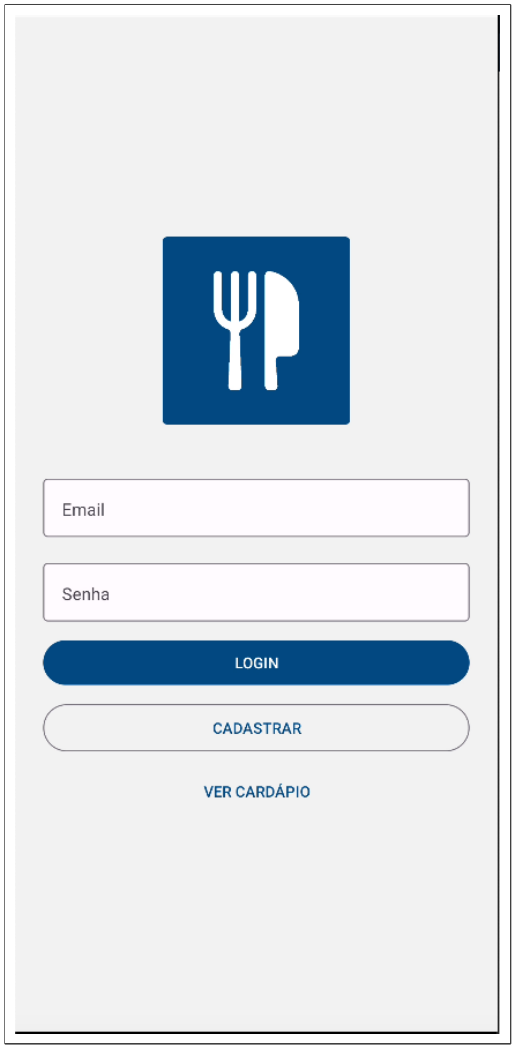


Figura 41: Interface de login do aplicativo.
Fonte: Do Autor.

5.0.2.7 Questionário

Ao clicar na opção de cadastro, o usuário é direcionado para uma interface com informações sobre o preenchimento do questionário elaborado pelo setor de nutrição, como ilustrado na Figura 42. Nesta interface, os usuários recebem orientações sobre o procedimento de preenchimento do questionário.

Figura 42: Interface de informativa sobre o questionário.
Fonte: Do Autor.

Na Figura 43, é exibida a interface da primeira etapa do questionário. Nessa etapa inicial, os usuários são solicitados a fornecer informações como nome, *email*, curso, número de matrícula e sexo.

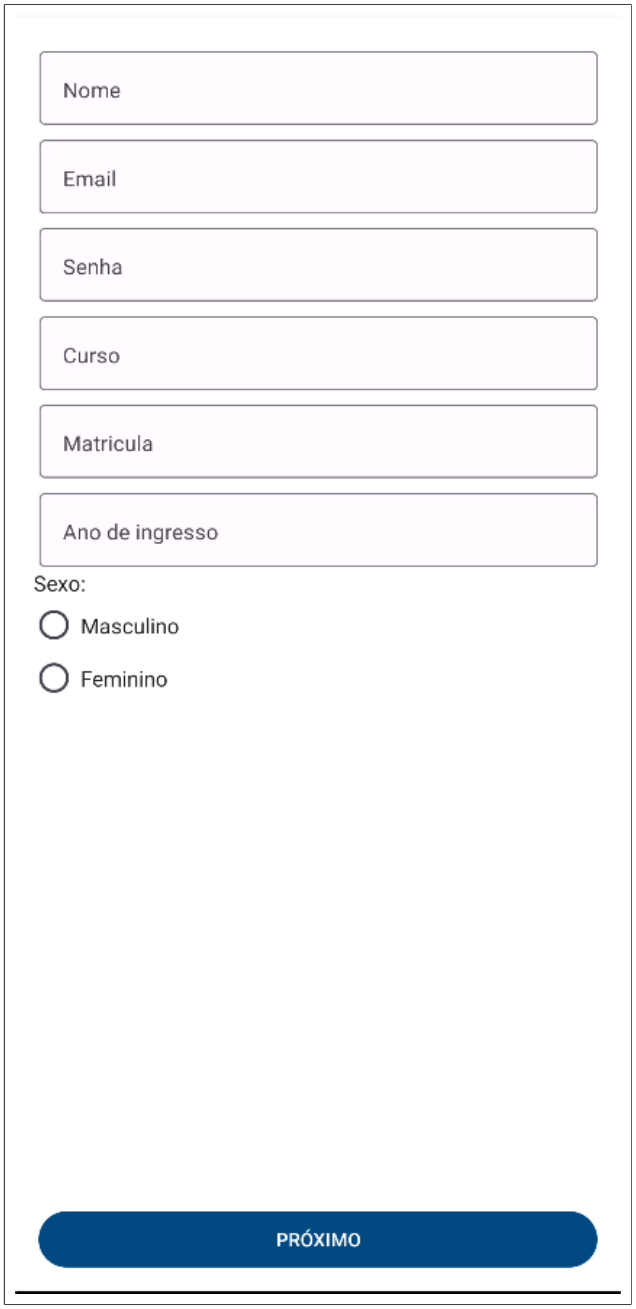


Figura 43: Primeira etapa do questionário.
Fonte: Do Autor.

Na Figura 44, é apresentada a interface da segunda etapa do questionário. Nessa fase, os usuários são solicitados a preencher informações relacionadas à bolsa do restaurante universitário, frequência de refeições e nível de atividade física.

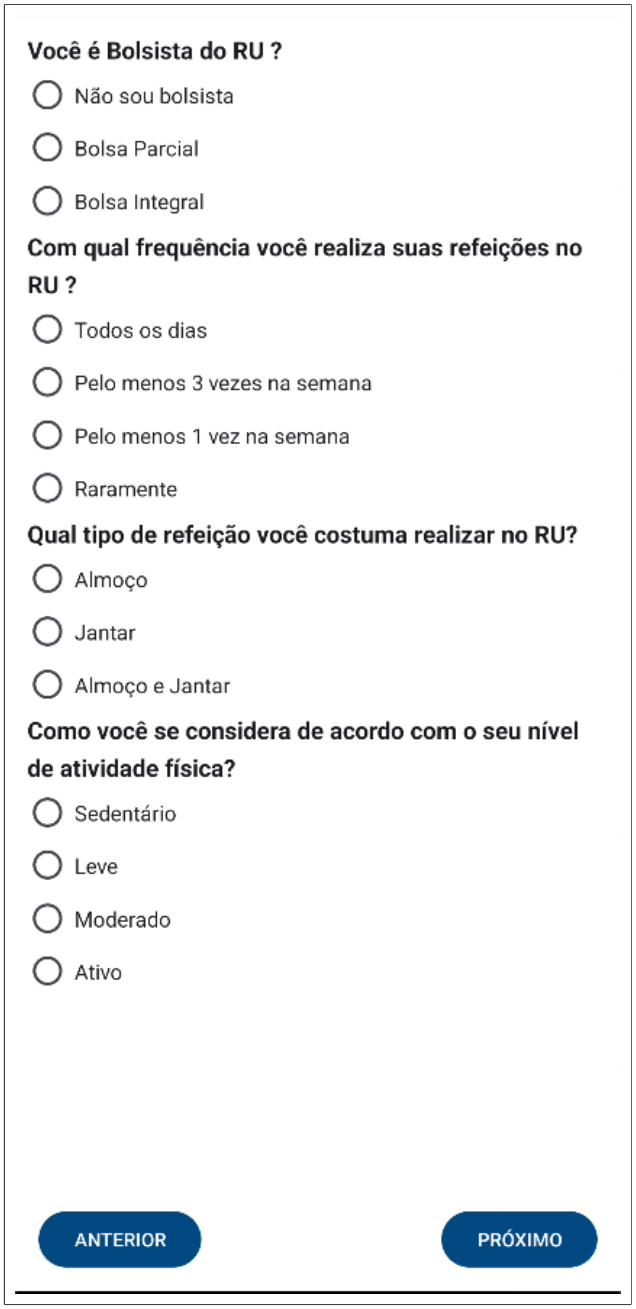


Figura 44: Segunda etapa do questionário.
Fonte: Do Autor.

Na Figura 45, é exibida a interface da terceira etapa do questionário. Nesse estágio, os usuários são solicitados a fornecer informações como peso ideal, se seguem uma dieta vegana ou vegetariana, e se possuem alguma intolerância alimentar.

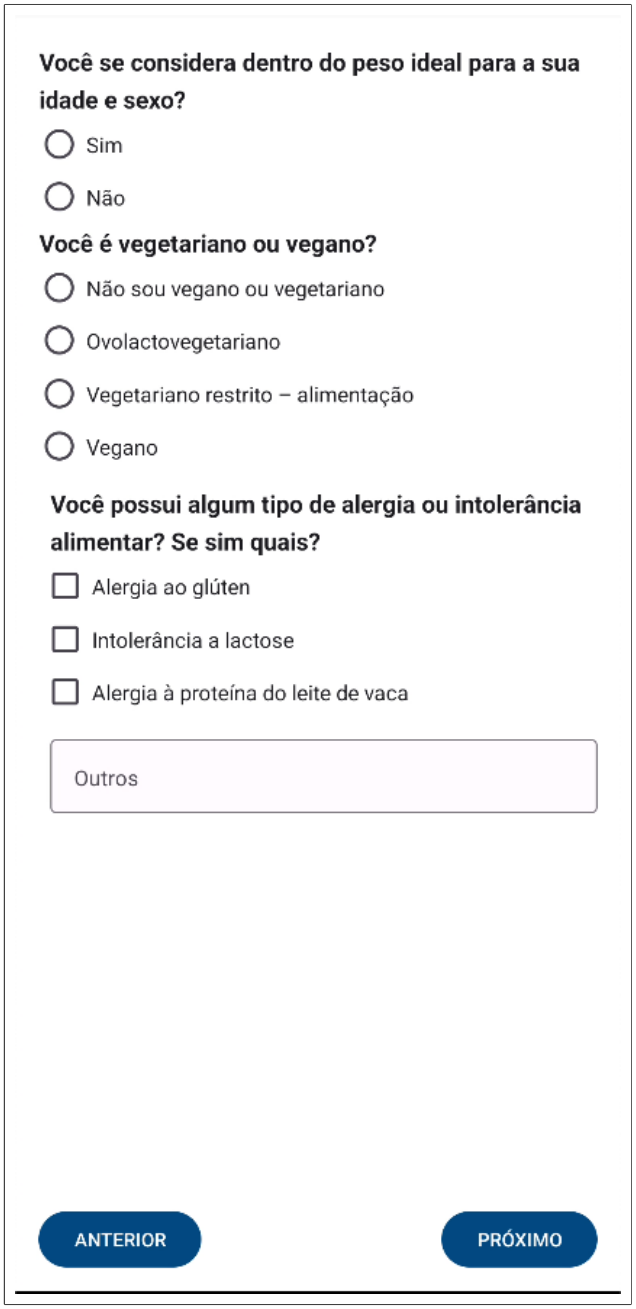


Figura 45: Terceira etapa do questionário.
Fonte: Do Autor.

Na Figura 46, é apresentada a interface da quarta etapa do questionário. Nesse estágio, os usuários são solicitados a fornecer informações sobre hábitos específicos, como o uso de saleiro à mesa, o uso de óleo composto, consumo de bebidas alcoólicas e se são tabagistas.

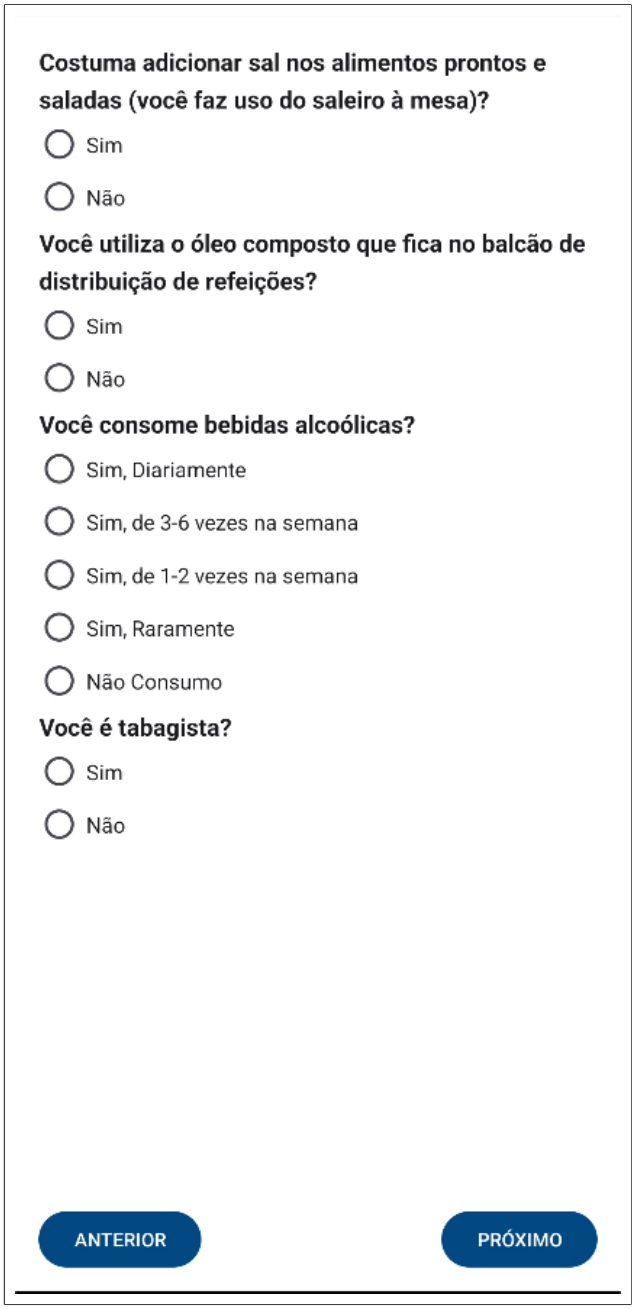


Figura 46: Quarta etapa do questionário.
Fonte: Do Autor.

Na Figura 47, é exibida a interface da quinta etapa do questionário. Nesse estágio, os usuários são solicitados a fornecer informações sobre histórico de patologias ou condições de saúde já diagnosticadas.

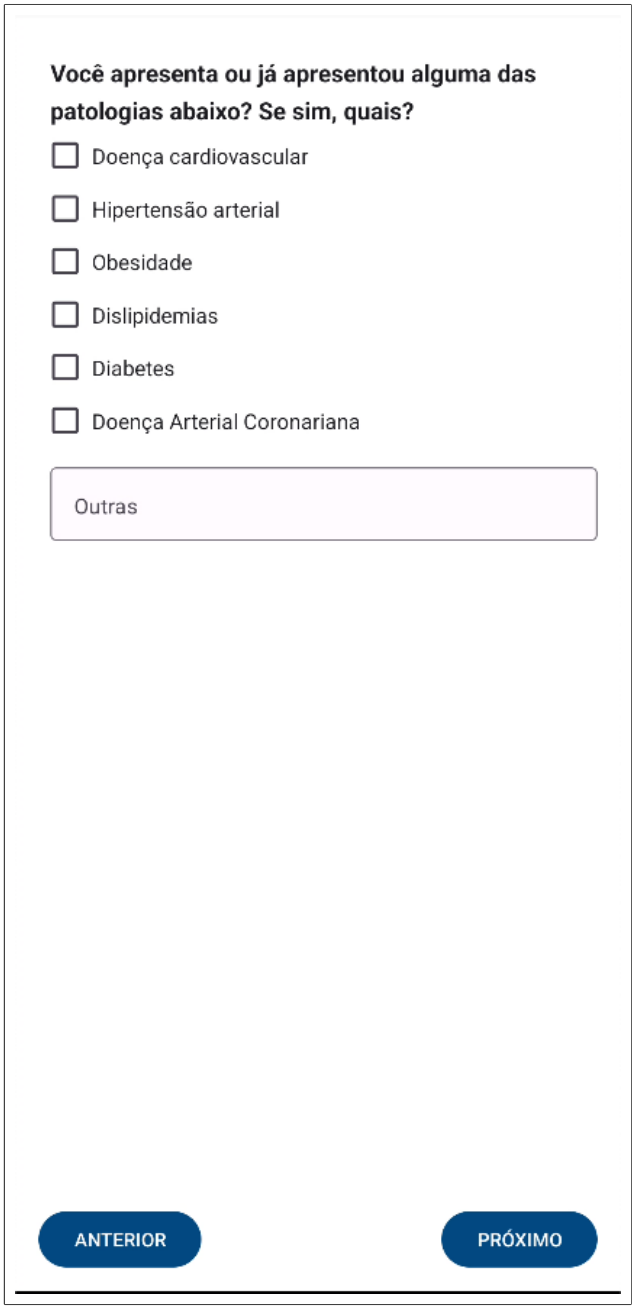


Figura 47: Quinta etapa do questionário.
Fonte: Do Autor.

Na Figura 48, é apresentada a interface da sexta etapa do questionário. Nesse estágio, os usuários são solicitados a fornecer informações sobre o histórico de patologias ou condições de saúde em sua família, bem como detalhes sobre o uso contínuo de medicamentos.

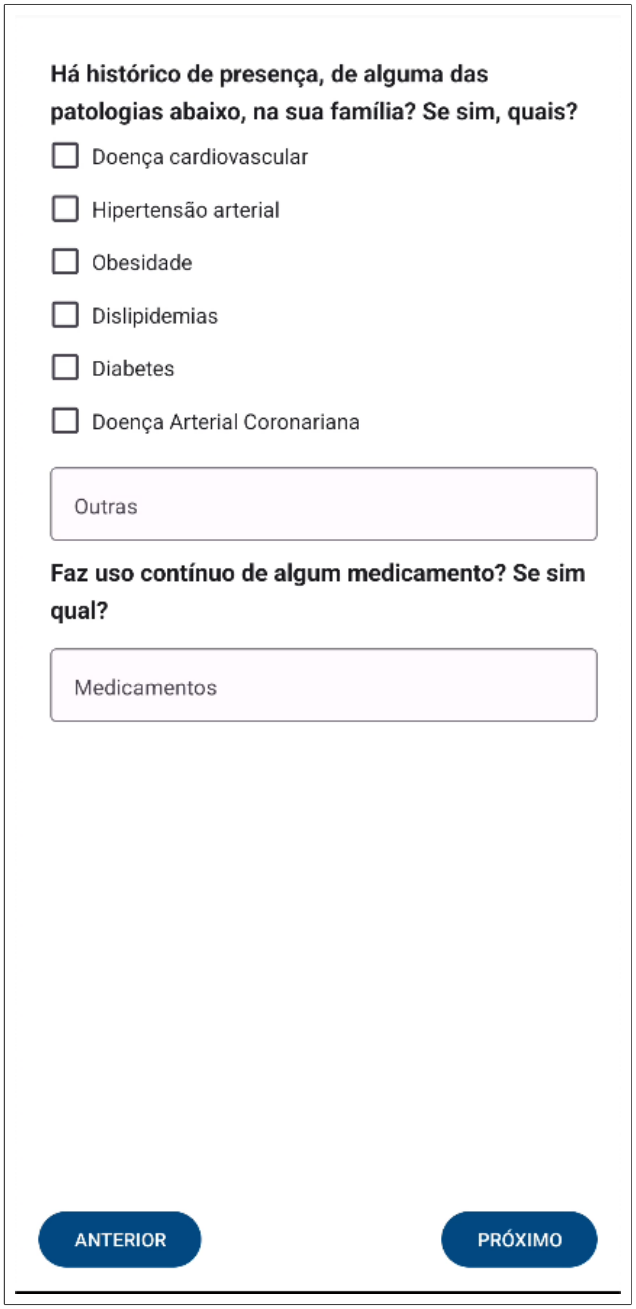


Figura 48: Sexta etapa do questionário.
Fonte: Do Autor.

A partir da sétima etapa do questionário são apresentada perguntas referente a avaliação do restaurante universitário. Na Figura 49, é exibida a interface da sétima etapa do questionário, onde os usuários são convidados a avaliar aspectos específicos do restaurante universitário. Nesse estágio, os participantes são solicitados a avaliar o aroma e a coloração do cardápio oferecido.

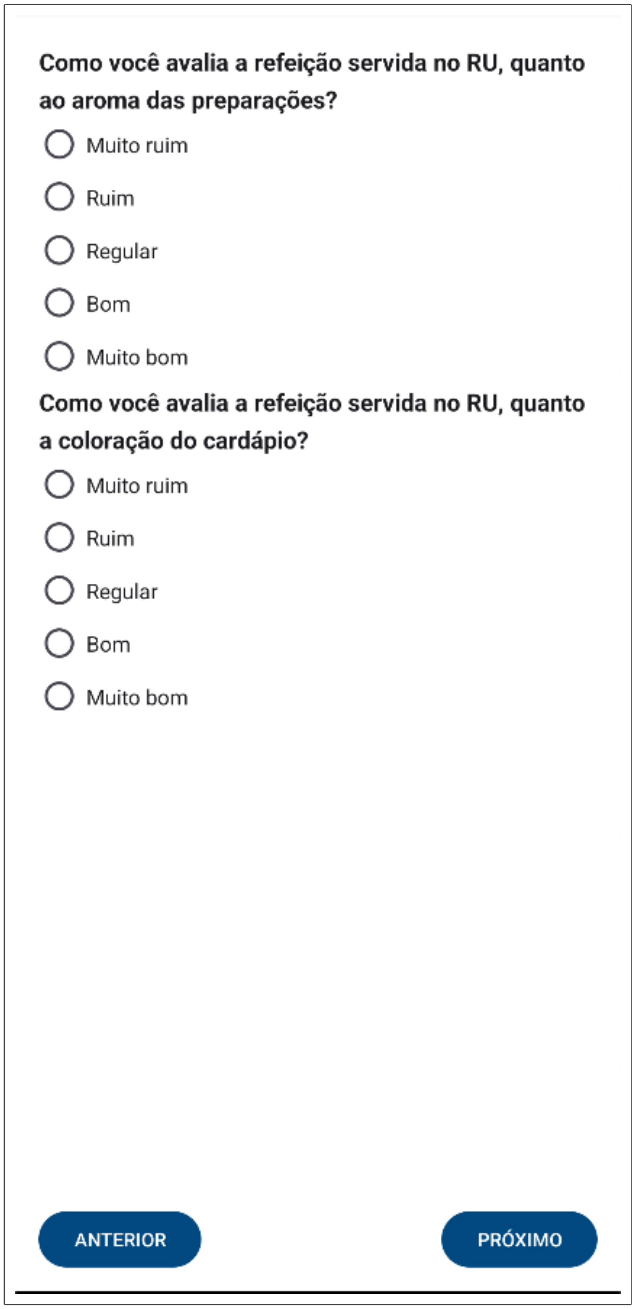


Figura 49: Sétima etapa do questionário.
Fonte: Do Autor.

Na Figura 50, é exibida a interface da oitava etapa do questionário. Nesse estágio, os usuários são convidados a avaliar a textura e o sabor das preparações oferecidas.

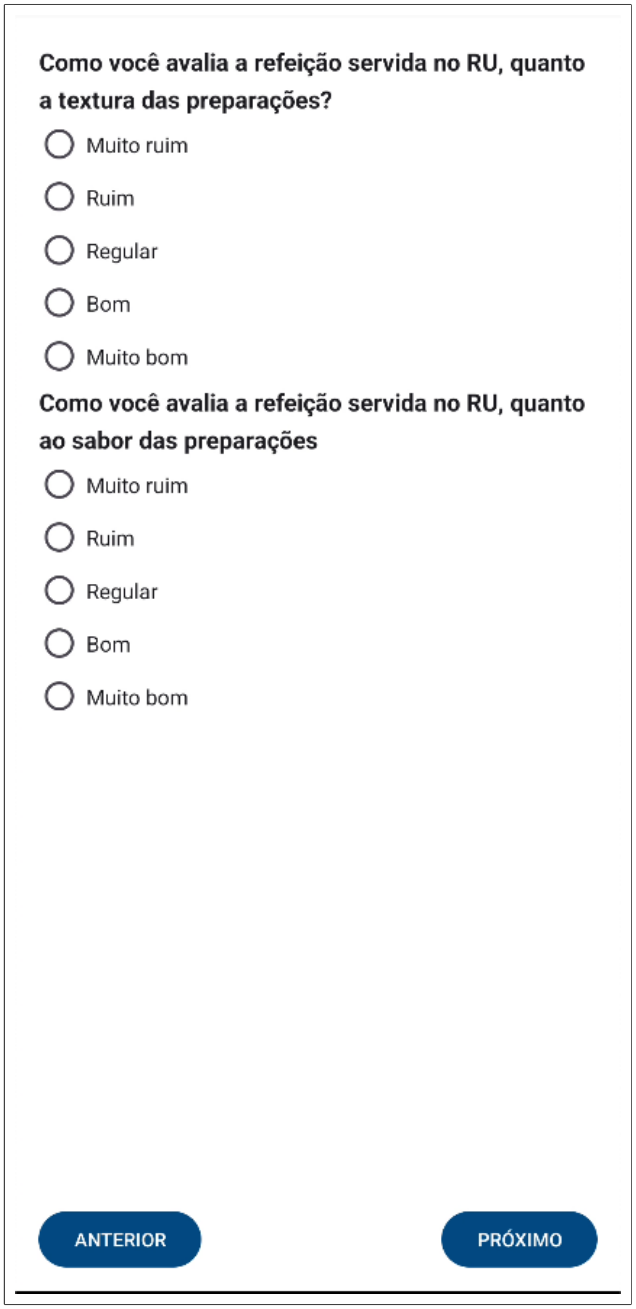


Figura 50: Oitava etapa do questionário.
Fonte: Do Autor.

Na Figura 51, é apresentada a interface da nona e última etapa do questionário. Nesse estágio final, os usuários são convidados a fornecer uma avaliação geral do restaurante, além de sugestões para possíveis melhorias. Após completar o questionário, a aplicação redireciona o usuário para a interface inicial do aplicativo.

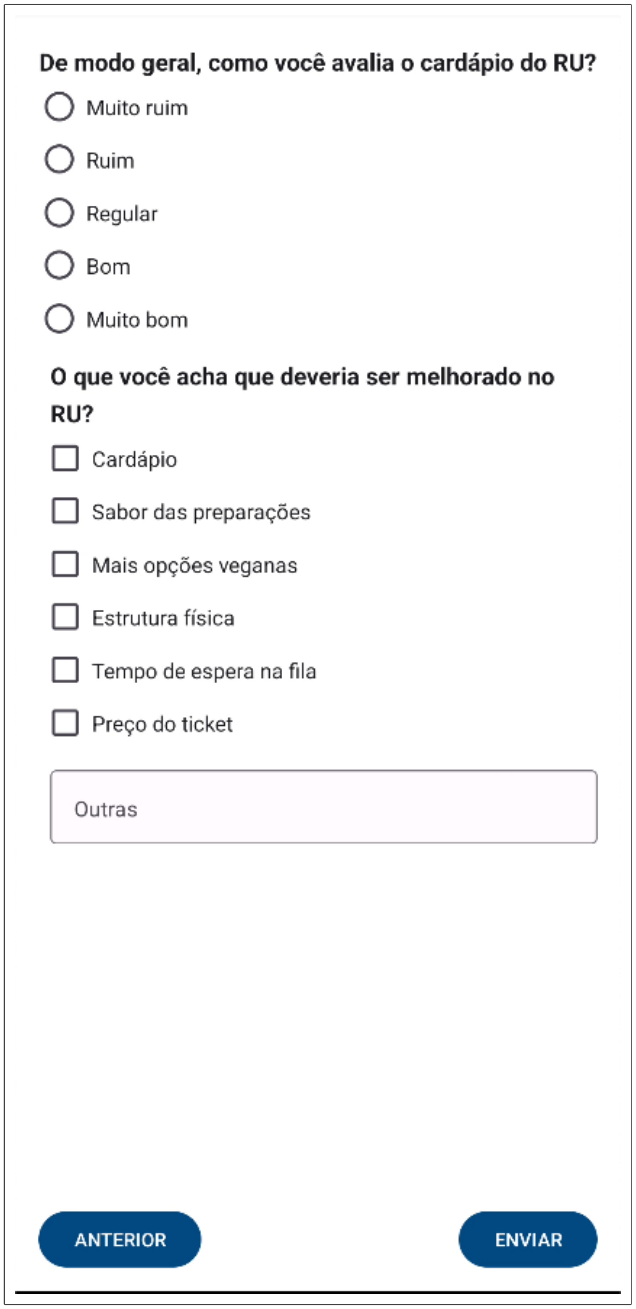


Figura 51: Nona e última etapa do questionário.
Fonte: Do Autor.

5.0.2.8 Interface inicial

Na Figura 52, é exibida a interface inicial da aplicação móvel. Nela, o cardápio do dia é destacado, e há opções no menu inferior, tais como: cardápio, avaliação, notícias e informações gerais

Figura 52: Interface inicial da aplicação móvel.
Fonte: Do Autor.

5.0.2.9 Interface de avaliação

Ao selecionar a opção de avaliação, uma interface é exibida conforme mostrado na Figura 53. Nessa interface, o cardápio do dia é apresentado junto com avaliação do cardápio do dia. Cada nível de avaliação, desde "muito ruim"até "muito bom", é representado por uma quantidade específica de estrelas.

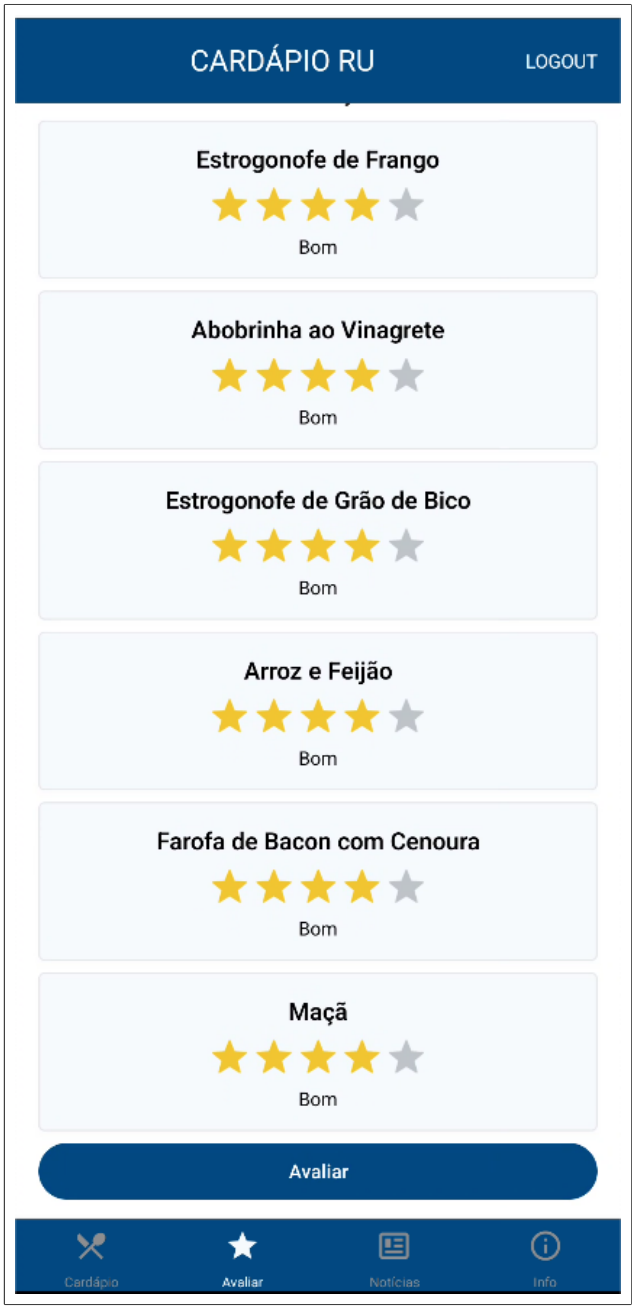


Figura 53: Interface inicial avaliação do cardápio.

Fonte: Do Autor.

Ao clicar no botão de avaliação, uma janela é exibida conforme mostrado na Figura 54. Nesse espaço, o cardápio do dia é apresentado, e os usuários têm a opção de avaliar individualmente os itens do menu usando um sistema de estrelas. Além disso, há a possibilidade de deixar um comentário sobre o cardápio.

Figura 54: Janela de avaliação do cardápio.
Fonte: Do Autor.

5.0.2.10 Interface de noticias

Na Figura 63, é apresentada a interface de listagem de notícias. Nesse cenário, os títulos das notícias são exibidos junto com as datas de seus cadastros, proporcionando aos usuários uma visão clara e organizada das informações disponíveis.

Figura 55: Interface de listagem de noticias.
Fonte: Do Autor.

Ao clicar em um item de notícia, uma janela é aberta exibindo os detalhes completos da notícia, conforme ilustrado na Figura 56. Nessa interface, os usuários têm acesso a informações detalhadas sobre a notícia selecionada.

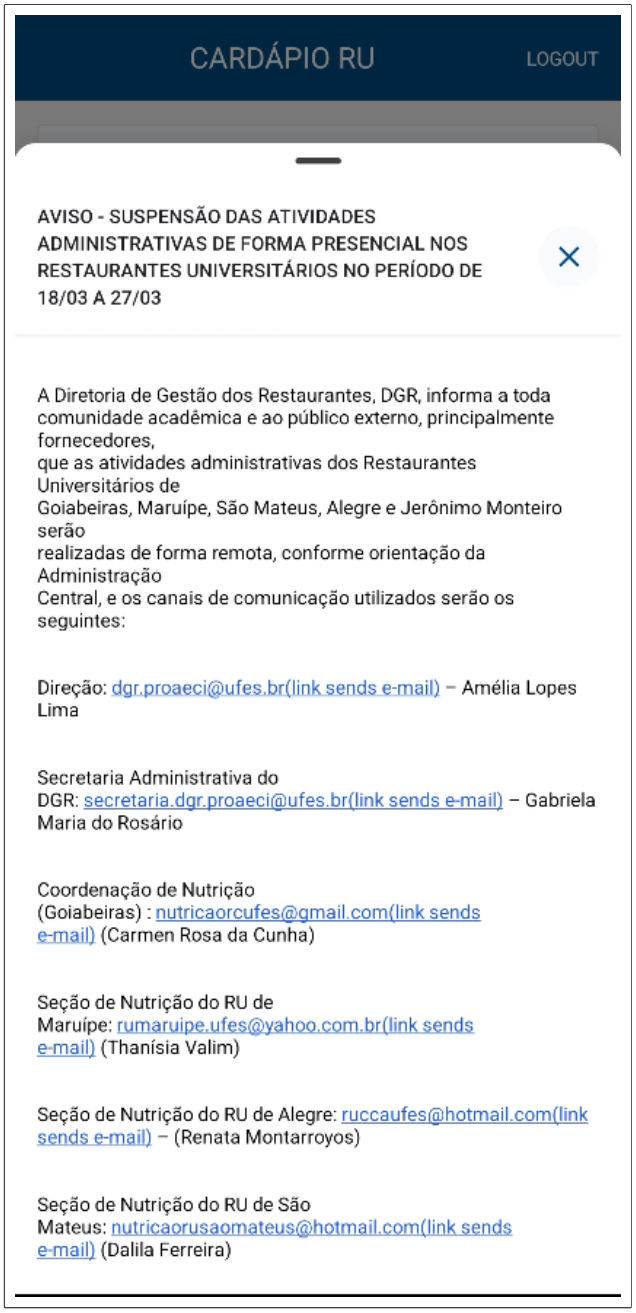


Figura 56: Interface de detalhe da noticia.

Fonte: Do Autor.

5.0.2.11 Interface de informações gerais

Na Figura 57, é apresentada a interface que contém informações gerais sobre o funcionamento do restaurante universitário. Nessa interface, são exibidos dados essenciais, como horário de funcionamento, valores dos *tickets* e informações sobre identificação.

Figura 57: Interface de informações gerais.

Fonte: Do Autor.

6 TESTES E GARANTIA DA QUALIDADE

Segundo Sommerville (2011) o teste é destinado a mostrar que um programa faz o que é proposto a fazer e para descobrir os defeitos do programa antes do uso. Os resultados do teste são verificados à procura de erros, anomalias ou informações sobre os atributos não funcionais do programa.

O teste de *software* é um elemento de um tópico mais amplo, muitas vezes conhecido como verificação e validação. Verificação refere-se ao conjunto de tarefas que garantem que o *software* implementa corretamente uma função especifica. Validação refere-se a um conjunto de tarefas que asseguram que o *software* foi criado e pode ser rastreado segundo os requisitos do cliente (PRESSMAN, 2011). Nesta seção serão demonstrados os testes realizados nas aplicações.

6.1 Teste de desenvolvimento

Para Sommerville (2011) os testes de desenvolvimento incluem todas as atividades de testes que são realizadas pela equipe de desenvolvimento do sistema. O testador do *software* geralmente é o programador que o desenvolveu. Testes de desenvolvimento são essencialmente um processo de teste de defeitos, em que o objetivo do teste é descobrir *bugs* no *software*.

A aplicação em desenvolvimento seguiu uma abordagem de testes contínuos, realizando verificações específicas para cada funcionalidade à medida que eram implementadas.

6.1.1 Teste de Integração

Segundo Objective (2023) teste de integração é uma etapa do processo de desenvolvimento de *software* em que módulos ou componentes são combinados e testados

em grupo. Esse tipo de teste visa verificar a eficiência e a segurança da comunicação entre sistemas. Essa etapa é essencial para garantir que o *software* funcione sem erros de integração.

Existem diferentes tipos de teste de integração, Objective (2023) os define da seguinte forma:

- Teste de integração vertical: verifica a integração entre camadas de um sistema, ou seja, entre as diferentes unidades que compõem a arquitetura do *software*. Nele, são analisadas as interações entre as camadas, bem como a consistência dos dados que são transmitidos entre elas.
- Teste de integração horizontal: utilizado para verificar a integração entre diferentes módulos ou componentes que atuam no mesmo nível de abstração. Nesse tipo de teste, são avaliadas as interações entre os componentes e a compatibilidade entre eles.

Foram testados a integração entre as camadas de validação (*middlwares*), intermediador de requisições (*controllers*), lógicas de negócio (*services*) e interação com o banco de dados (*repositories*). Para isso foram desenvolvidos cenários de testes para validar um fluxo completo e como esses módulos se comportariam transmitindo as informações necessárias, Ao final de cada teste foram inspecionados os dados gravados no banco de dados e verificado se a informação estava consistente. Foram identificados e corrigidos os seguintes itens:

Quadro 2: Testes de integração realizados

Teste realizado	Falha encontrada	Motivo	Solução
Cadastro de cardápio	Ao cadastrar um cardápio o campo opção estava preenchendo com os mesmos dados do campo prato proteico	O objeto passado para função de gravar um cardápio estava recebendo o valor do campo prato proteico no campo opção	Alterado o objeto de cardápio para receber os campos corretamente
Cadastro de cardápio	Permitido cadastrar um cardápio sem enviar um tipo de cardápio	Falta de validação do envio do tipo do cardápio	Adicionado a validação do tipo de cardápio
Cadastro de cardápio	Ao editar o cardápio o campo tipo de cardápio não estava sendo atualizado	A função responsável por atualizar o cardápio não estava considerando o campo tipo	Adicionado o campo tipo na função responsável por atualizar o cardápio.
Avaliação do cardápio	Ao avaliar o cardápio estava permitindo avaliar o mesmo cardápio múltiplas vezes	A função responsável por salvar uma avaliação de cardápio não levava em conta se o usuário já tinha avaliado o cardápio em questão	Alterado a lógica da função responsável por salvar uma avaliação para levar em conta se o usuário já tinha feito uma avaliação prévia.
Gráficos	Dados inconsistentes no gráfico de frequência do restaurante universitário	A contagem do campo "Pelo menos 1 vez na semana" não estava sendo contabilizada	Adicionado a contabilidade do campo "Pelo menos 1 vez na semana" no cálculo de porcentagem do uso do restaurante universitários

Fonte: Do Autor

O Quadro 2 apresentada destaca os resultados dos testes de integração realizados,

evidenciando os erros identificados e as correções aplicadas. Estes testes desempenharam um papel importante ao avaliar a integração entre as diversas camadas da aplicação, assegurando o funcionamento adequado e coeso de cada componente. A análise desses testes contribuiu para a validação das funcionalidades de cada camada, reforçando assim a confiabilidade do sistema como um todo.

6.1.2 Teste *End-to-End*

Segundo Marques (2018) teste *end-to-end* é uma metodologia utilizada para testar se o fluxo de um aplicativo está sendo executado conforme o projeto do início ao fim. O objetivo da realização de testes *end-to-end* é identificar dependências do sistema e garantir que a informação certa seja passada entre vários componentes e sistemas.

Os testes abrangeram ambos os *fronts ends* do sistema, tanto a versão *web* quanto a mobile. Essas avaliações foram conduzidas com o objetivo de verificar os fluxos completos, seguindo os cenários definidos pelos casos de uso da aplicação. Durante esses testes, a ênfase foi colocada na verificação da apresentação correta dos dados e na consistência do fluxo de interações. O foco principal era garantir que ambas as interfaces estivessem exibindo os dados de maneira apropriada e que os fluxos de usuário estivessem alinhados com as expectativas estabelecidas. Com isso foram identificados e corrigidos os seguintes itens:

Quadro 3: Tabela de testes *end to end* realizados

Teste realizado	**Falha encontrada**	**Motivo**	**Solução**
Detalhe da notícia	Layout inconsistente no dispositivo móvel ao clicar em um detalhe da notícia	A notícia cadastrada em questão estava com caracteres especiais	Adicionado uma etapa de tratamento de texto ao criar uma notícia
Edição de cardápio	Após edição do cardápio não o sistema não redirecionava para tela de listagem de cardápio	A função responsável por navegar para tela de listagem, estava recebendo a rota da edição de cardápio	Alterado a rota de navegação para a listagem de cardápio

Fonte: Do Autor

Quadro 4: Tabela de testes *end to end* realizados

Teste realizado	**Falha encontrada**	**Motivo**	**Solução**
Preenchimento questionário primeira etapa	Campo email permitindo email não válido	Falta de validação da string como email	Adicionado validação no input de email
Preenchimento questionário segunda etapa	Permitia passar para próxima etapa do questionário sem a resposta da pergunta sobre atividade física	A pergunta em questão não estava obrigatória	Adicionado obrigatoriedade na pergunta em questão
Preenchimento questionário terceira etapa	Teclado do dispositivo móvel aparecia por cima do campo de "Outros" na pergunta sobre alergia	O componente utilizado não permitia a rolagem da tela para que o campo fosse exibido	Adicionado o componente de rolagem na tela em questão
Preenchimento questionário sexta etapa	Ao avançar para próxima etapa sem responder as questões não mostrava nada na tela que indicava erro	Falta do componente de erro com a mensagem em vermelho indicando erro na etapa em questão	Adicionado o componente de texto responsável por indicar o erro na etapa
Avaliação do cardápio quando usuário está deslogado	Caixa de aviso não era exibida informando ao usuário que somente usuários logados podem avaliar o cardápio	A caixa de aviso estava atrás da tela de avaliação	Adicionado atributo para que o componente de aviso fosse exibido acima da tela de avaliação quando o usuário não esteja logado
Exibição do cardápio do dia	Lentidão ao exibir o cardápio do dia	O servidor em questão em alguns momentos entrava em estado de espera	Removido a configuração que deixava o servidor entrar em modo de espera para economizar recursos

Fonte: Do Autor

Os Quadros 3 e 4 apresentados ilustram os resultados dos testes *end-to-end* realizados, destacando o teste realizado, a falha identificada, o motivo e a solução aplicada para corrigir o problema. Estes testes desempenharam um papel importante na validação do fluxo completo, assegurando o funcionamento integral de cada etapa e fornecendo descobertas sobre a localização exata das falhas ao longo do processo. A análise desses testes não apenas identificou e corrigiu as falhas, mas também contribuiu para a validação do fluxo como um todo, aprimorando a confiabilidade e a eficácia do sistema.

6.2 Teste de usuário

De acordo com Sommerville (2011) teste de usuário é um estágio no processo de teste em que os usuários fornecem entradas e conselhos sobre o teste de sistema. Isso pode envolver o teste formal de um sistema que foi aprovado por um fornecedor externo ou processo informal em que os usuários experimentam um produto de *software* novo para ver se gostam e verificar sua aderência ás suas necessidades. O teste de usuário é essencial, mesmo em sistemas abrangentes ou quando testes de *release* tenham sido realizados. A razão para isso é que as influências do ambiente de trabalho do usuário têm um efeito importante sobre a confiabilidade, o desempenho, a usabilidade e a robustez de um sistema.

Existem três tipos de testes de usuário:

- Teste alfa, em que os usuários do *software* trabalham com a equipe de desenvolvimento para testar o *software* no local do desenvolvedor.
- Teste beta, em que um *release* do *software* é disponibilizado aos usuários para que possam experimentar e levantar os problemas que eles descobriram com os desenvolvedores do sistema.
- Teste de aceitação, em que os clientes testam um sistema para decidir se está ou não pronto para ser aceito pelos desenvolvedores de sistemas e implantado no ambiente do cliente.

No contexto do sistema *web*, o professor orientador deste trabalho desempenhou o papel de testador alfa, identificando uma área de aprimoramento específica:

- Observou-se que os dados dos alunos estavam sendo acessados por meio do menu à esquerda, expondo informações sensíveis dos alunos aos usuários do sistema *web*. Para abordar essa preocupação, essa opção foi removida.

Na fase de testes beta da aplicação mobile, uma versão foi disponibilizada para a avaliação de três alunos. Durante esse processo, eles desempenharam o papel de testadores beta, para isso foi contextualizado o que seria aplicação mobile, e os requisitos implementados na aplicação mobile. Após o teste foi disponibilizado um questionário (disponível no capitulo de apêndice), para obter *feedback* sobre o teste em questão. Após a análise do *feedback* foram identificados os seguintes pontos de aprimoramento:

Quadro 5: Problemas encontrados no teste beta

Problemas encontrados	**Solução**
O campo destinado ao ano de ingresso não possuía um limite de caracteres	Foi adicionado uma limitação de 4 caracteres para o input do ano de ingresso
O campo de matrícula estava permitindo a inserção de letras como caracteres válidos.	Foi adicionado uma propriedade de *number pad* no componente responsável por obter a matrícula do aluno

Fonte: Do Autor

O Quadro 5 apresenta as questões identificadas pelos testadores beta da aplicação, assim como as soluções implementadas para corrigir cada problema identificado.

6.3 Teste de usabilidade

Existem diferentes tipos de testes em aplicações móveis entre eles, estão os testes de usabilidade que devem ser realizados para garantir a experiência do usuário, validando que a aplicação possui um visual agradável e acesso as ações pretendidas de forma simples e facilitada (MAMEDE, 2018).

No contexto do sistema *web*, a avaliação da usabilidade ficou a cargo do professor orientador, enquanto, para a aplicação mobile, essa responsabilidade foi assumida

pelos três alunos. Novamente foi disponibilizado um questionário após o conclusão do teste, e foram identificados os seguintes pontos:

Quadro 6: Problemas encontrados no teste de usabilidade

Problemas encontrados	**Solução**
Os botões de navegação para a próxima etapa não estavam sendo exibidos corretamente	Os estilos dos botões de navegação foram ajustados, considerando o dimensionamento para adequação à tela.
Na tela de informações gerais as informações estavam muitos próximas dificultando a leitura	Ajustes nos estilos dos títulos e dos textos informativos proporcionando uma exibição mais clara e legível

Fonte: Do Autor

O Quadro 6 acima expõe as questões identificadas durante o teste de usabilidade, juntamente com as soluções implementadas para mitigar os problemas encontrados.

6.4 Avaliação dos testes

Durante o desenvolvimento do sistema, foram realizadas avaliações abrangentes pelo desenvolvedor. A participação do professor orientador foi importante para o projeto, envolvendo testes gerais e validações, especialmente focados na usabilidade da seção *web* do sistema. Ele desempenhou o papel de gerenciador, representando o restaurante universitário como o usuário final.

Na aplicação mobile, os *feedbacks* provenientes dos três alunos selecionados para testar o sistema foram de extrema importância, uma vez que representam os usuários finais. No teste da aplicação mobile, foi considerável o *feedback* positivo recebido. Em particular, foi destacada a utilidade da funcionalidade que permite avaliar o cardápio, evidenciando o valor dessa característica para os usuários. Além disso, foi observado que não foram encontradas dificuldades na busca por informações, e a navegabilidade do aplicativo foi avaliada de forma positiva, indicando uma experiência fluida e intuitiva para os usuários durante a interação com a aplicação.

Durante a fase de desenvolvimento dos testes, foram identificadas melhorias e sugestões que foram prontamente acolhidas, desenvolvidas e implementadas no sistema

como um todo. Essas correções resultaram em aprimoramentos significativos na usabilidade e resiliência do sistema.

7 CONSIDERAÇÕES FINAIS E CONCLUSÕES

O projeto, de maneira abrangente, mostrou-se uma solução potencialmente eficaz para auxiliar tanto os gestores responsáveis pelo cardápio do restaurante universitário quanto os alunos frequentadores da instituição. Para os gestores, as funcionalidades oferecidas pelo projeto simplificam e aprimoram o gerenciamento do cardápio, fornecendo informações gerais essenciais e, dados sobre a nutrição dos alunos que utilizam o restaurante universitário.

Além disso, para os alunos, o projeto representa uma ferramenta útil para acessar de forma rápida e prática informações detalhadas sobre o cardápio diário, assim como visualizar dados gerais sobre o funcionamento do restaurante universitário. Além disso, a funcionalidade de avaliação do cardápio oferece aos alunos a oportunidade de expressar suas opiniões e contribuir para a melhoria contínua das refeições oferecidas no restaurante universitário. Esta abordagem não apenas aprimora a experiência dos usuários, mas também contribui para uma gestão mais eficiente e informada ao setor de nutrição da instituição.

O sistema apresenta um conjunto de funcionalidades que visam facilitar a gestão do cardápio no ambiente do restaurante universitário. Entre essas funcionalidades, destaca-se o cadastro diário do cardápio, fornecendo uma ferramenta potencialmente eficaz para os gestores atualizarem as opções disponíveis. Além disso, o projeto permite a exibição das avaliações dos alunos sobre o cardápio do dia, oferecendo aos gestores *feedbacks* valiosos sobre a recepção e preferências dos usuários.

A gestão de notícias e avisos representa outra funcionalidade essencial, possibilitando a comunicação eficiente de informações relevantes aos usuários, como eventuais mudanças no cardápio, eventos especiais ou qualquer outra atualização relevante. Adicionalmente, a exibição de estatísticas geradas a partir dos questionários disponibilizados aos alunos oferece uma visão sobre a satisfação e as necessidades nutricionais dos alunos da instituição.

Essas funcionalidades, integradas de forma consistente, não apenas facilitam o dia a dia dos gestores do RU, mas também capacitam decisões embasadas em dados concretos. Ao proporcionar uma visão das preferências alimentares e das avaliações dos usuários, o projeto possibilita que os gestores avaliem continuamente a eficácia do cardápio atual e considerem ajustes para atender às expectativas e necessidades da comunidade acadêmica. Essa abordagem baseada em dados promove uma gestão mais estratégica e alinhada às demandas reais dos usuários.

A visualização de notícias e avisos, juntamente com informações gerais, enriquece ainda mais a experiência dos alunos, mantendo-os informados sobre eventos especiais, mudanças no cardápio, ou quaisquer atualizações relevantes. Essa funcionalidade promove a transparência e cria uma experiência satisfatória e bem informada.

Ao disponibilizar uma plataforma que atende às necessidades dos alunos, o projeto visa aprimorar a experiência no restaurante universitário. A combinação de acesso fácil às informações do cardápio, a capacidade de oferecer *feedbacks* e a exibição de informações gerais estabelece um ambiente que potencialmente melhora positivamente a experiência dos alunos no que diz respeito à alimentação na instituição.

Adicionalmente, o desenvolvimento deste projeto representou para o autor uma valiosa oportunidade de aplicar de forma prática os conhecimentos adquiridos durante a graduação. A utilização de tecnologias como *React.js*, *React Native*, *Node.js* e *MongoDB* não apenas enriqueceu o repertório do autor, mas também o preparou de maneira sólida para enfrentar as demandas do mercado de trabalho.

A exploração abrangente das áreas de *backend*, *front end web* e *front end mobile*, embora desafiadora, proporcionou ao autor uma experiência prática em elementos fundamentais empregados nos sistemas atuais. Este processo não apenas consolidou a compreensão teórica adquirida durante os estudos, mas também permitiu uma imersão prática nas ferramentas e metodologias essenciais para o desenvolvimento eficaz de soluções tecnológicas. O projeto, assim, serviu como um campo de treinamento valioso, capacitando o autor com habilidades e competências essenciais para enfrentar os desafios do cenário profissional.

Como pontos de melhoria incluem a realização de testes com uma amostra de teste mais representativa ao avaliar o *front end mobile*, especialmente com uma maior taxa de participação dos alunos. Além disso, não foi possível envolver todos os interessados nos testes, incluindo gestores do setor de nutrição, para garantir uma opinião do sistema do *front end web*. Adicionalmente, a automação dos testes mencionados é

considerada benéfica, embora não seja crucial, pois poderia proporcionar um auxílio na validação do sistema. Vale ressaltar que, os pontos de melhoria não foram implementados devido a restrições de tempo na execução dos requisitos em todas as três áreas em questão: *front end mobile*, *front end web* e *back end*. A inclusão dos gestores do setor de nutrição teria contribuído para uma validação mais abrangente do sistema como um todo. Essa participação adicional poderia resultar em aprimoramentos no *front-end web*, contribuindo para melhorias na interface, incluindo observações relevantes e possibilitando a sugestões de implementação de futuras funcionalidades.

7.1 Trabalhos futuros

No projeto, foram identificados aspectos que sinalizam possíveis desenvolvimentos e aprimoramentos a serem explorados em trabalhos futuros. A seguir, foram destacadas algumas dessas funcionalidades:

- Adaptar a aplicação móvel para o sistema *iOS*.
- Desenvolver os requisitos do sistema *web* para a aplicação móvel, consolidando de forma integrada os requisitos gerais do projeto em uma única plataforma.
- Implementar a funcionalidade de cadastro de múltiplos cardápios para a semana corrente.
- Criar a funcionalidade de notificações para informar os usuários sobre a disponibilidade de novos cardápios diários e atualizações de notícias relevantes.
- Integrar a funcionalidade de compra de *tickets* diretamente na aplicação e autenticação de alunos com diferentes do tipos de usuário.
- Gerar estatísticas das avaliações gerais para comparação de cardápios.
- Estudos futuros podem ser direcionados à análise mais aprofundada dos dados obtidos por meio das respostas do questionário.
- Testar com uma amostra mais significativa o uso do sistema, principalmente pelos docentes do departamento de Nutrição responsáveis pelo questionário e pelos gestore do Restaurante Universitário, obtendo ajustes e melhoria na aplicação.

REFERÊNCIAS

ABUARQOUB, A. et al. A survey on internet of things enabled smart campus applications. In: *CONFERENCE ON FUTURE NETWORKS AND DISTRIBUTED SYSTEMS*. New York,USA: [s.n.], 2017. p. 50:1–50:7. Citado na página 19.

ACIDADEON. *Estudantes criam sistemas para solucionar problemas do campus*. 2017. Disponível em: <https://portal.ufcg.edu.br/ultimas-noticias/878-ufcg-cria-o-smart-campus.html>. Acesso em: 2019-12-04. Citado na página 21.

ALLIES, P. *Clean Architecture: Use Cases*. 2023. Disponível em: <https://nanosoft.co.za/blog/post/clean-architecture-use-cases>. Acesso em: 2023-10-15. Citado na página 49.

ALLIGATOR.IO. *Compreendendo ações assíncronas do Redux com o Redux Thunk*. 2020. Disponível em: <https://www.digitalocean.com/community/tutorials/redux-redux-thunk-pt>. Acesso em: 2023-10-15. Citado na página 24.

BENVINDO, J. L. da S.; PINTO, A. M. de S.; BANDONI, D. H. Qualidade nutricional de cardápios planejados para restaurantes universitários de universidades federais do brasil. *Demetra: alimentação, nutrição saúde*, v. 12, n. 2, p. 447–464, 2017. Citado na página 16.

BRITO, H. et al. Análise de aprendizagem de frameworks móveis javascript. In: *14th Iberian Conference on Information Systems and Technologies (CISTI)*. Coimbra,Portugal: [s.n.], 2019. Citado na página 23.

BURLEA-SCHIOPOIU, A.; BURDESCU, D. D. The development of the critical thinking as strategy for transforming a traditional university into a smart university. In: *Smart Education and e-Learning 2017*. [S.l.: s.n.], 2017. p. 67–74. Citado na página 19.

CASTRO, R. de. *Universidade do futuro: você vai querer estudar nestes campi tecnológicos*. 2018. Disponível em: <https://cidadesmaisinteligentes.blogosfera.uol.com.br/2018/10/29/a-universidade-do-futuro-em-que-voce-vai-querer-estudar/>. Acesso em: 2019-12-04. Citado na página 19.

CHODOROW, K. *MongoDB: The Definitive Guide*. [S.l.]: Apress, 2013. Second Edition. Citado 2 vezes nas páginas 30 e 31.

COUTINHO, P. C. *REST Tutorial*. 2013. Disponível em: <https://www.devmedia.com.br/rest-tutorial/28912>. Acesso em: 2019-11-25. Citado na página 25.

DEVMEDIA. *Como funcionam as aplicações web*. 2019. Disponível em: <https://www.devmedia.com.br/guia/node-js/40312>. Acesso em: 2019-11-21. Citado na página 26.

DUARTE, L. *Boas práticas com MongoDB*. 2017. Disponível em: <https://blog.umbler.com/br/boas-praticas-com-mongodb>. Acesso em: 2019-12-02. Citado na página 39.

EISENMAN, B. *Learning React Native Building Native Mobile Apps with JavaScript*. [S.l.]: O'Reilly, 2017. Second Edition. Citado na página 24.

EXPRESS. *Usando middlewares*. 2023. Disponível em: <https://expressjs.com/pt-br/guide/using-middleware.html>. Acesso em: 2023-10-15. Citado na página 28.

FACEBOOK. *React Native Learn once, write anywhere*. 2019. Disponível em: <https://facebook.github.io/react-native/>. Acesso em: 2019-11-04. Citado na página 24.

FACEBOOK. *React – Uma biblioteca JavaScript para criar interfaces de usuário*. 2019. Disponível em: <https://pt-br.reactjs.org/>. Acesso em: 2019-11-04. Citado 2 vezes nas páginas 23 e 24.

FERNANDES, P. *Smart Campus tem tecnologia a serviço do bem-estar coletivo*. 2018. Disponível em: <https://www2.jornalcruzeiro.com.br/materia/881622/smart-campus-tem-tecnologia-a-servico-do-bem-estar-coletivo>. Acesso em: 2019-12-04. Citado na página 21.

FERREIRA, F. *USP implanta tecnologia inteligente no campus*. 2018. Disponível em: <https://revistasegurancaeletronica.com.br/usp-implanta-tecnologia-inteligente-no-campus/>. Acesso em: 2019-12-04. Citado na página 20.

FERREIRA, F. H. C.; ARAUJO, R. M. de. Campus inteligentes: Conceitos, aplicações,tecnologias e desafios. In: *Relatórios Técnicos do Departamento de Informática Aplicada da UNIRIO*. [S.l.: s.n.], 2018. Citado na página 19.

FERREIRA, R. *REST: Princípios e boas práticas*. 2017. Disponível em: <https://blog.caelum.com.br/rest-principios-e-boas-praticas/>. Acesso em: 2019-11-25. Citado na página 25.

FIGUEIREDO, E. *O que é uma SPA: Single-Page Application?* 2018. Disponível em: <https://blog.schoolofnet.com/o-que-e-uma-spa-single-page-application/>. Acesso em: 2019-11-04. Citado na página 23.

FOUNDATION, N. *Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine.* 2019. Disponível em: <https:///nodejs.org/>. Acesso em: 2019-11-04. Citado na página 27.

GUEDES, M. *SQL vs NoSQL, qual usar?* 2017. Disponível em: <https://www.treinaweb.com.br/blog/sql-vs-nosql-qual-usar/>. Acesso em: 2019-12-04. Citado 2 vezes nas páginas 29 e 30.

GUEDES, M. *O que é Redux?* 2019. Disponível em: <https://www.treinaweb.com.br/blog/o-que-e-redux>. Acesso em: 2023-10-15. Citado 2 vezes nas páginas 23 e 24.

GUPTA, L. *What is REST*. 2018. Disponível em: <https://restfulapi.net/>. Acesso em: 2019-11-25. Citado na página 25.

HARVE. *O que é backend: Guia completo*. 2023. Disponível em: <https://harve.com.br/blog/desenvolvimento-web/o-que-e-backend-guia-completo/>. Acesso em: 2023-10-15. Citado na página 25.

IMHOF, R.; FROZZA, A. A.; MELLO, R. Um survey sobre extração de esquemas de documentos json. In: *XIII Escola Regional de Informática de Banco de Dados*. Passo Fundo-RS,Brasil: [s.n.], 2017. p. 26–35. Citado na página 30.

JACOSKI, C. A.; HOFFMEISTER, L. M. *CAMPUS INTELIGENTE:UM NOVO PARADIGMA NA ORGANIZAÇÃO DAS UNIVERSIDADES*. 2018. Disponível em: <https://repositorio.ufsc.br/bitstream/handle/123456789/190551/101_00104.pdf>. Citado 2 vezes nas páginas 15 e 19.

JSON.ORG. *Introducing JSON*. 2019. Disponível em: <https://www.json.org/>. Acesso em: 2019-11-21. Citado na página 28.

KOKAY, M. C. *Banco de dados NoSQL: Um novo paradigma - Revista SQL Magazine*. 2012. Disponível em: <https://www.devmedia.com.br/banco-de-dados-nosql-um-novo-paradigma-revista-sql-magazine-102/25918>. Acesso em: 2019-12-04. Citado na página 29.

LIMA, C. *Spring Boot - Trabalhando com Controllers*. 2023. Disponível em: <https://www.treinaweb.com.br/blog/spring-boot-trabalhando-com-controllers>. Acesso em: 2023-10-15. Citado na página 28.

LIMA, M. *Brasil já tem mais de um smartphone ativo por habitante, diz estudo da FGV*. 2018. Disponível em: <https://link.estadao.com.br/noticias/geral,brasil-ja-tem-mais-de-um-smartphone-ativo-por-habitante-diz-estudo-da-fgv,70002275238>. Acesso em: 2019-11-25. Citado na página 15.

MAMEDE, E. *Quais os tipos de teste que devo fazer em aplicações móveis?* 2018. Disponível em: <https://medium.com/@elisabethmamede/quais-os-tipos-de-teste-que-devo-fazer-em-aplica%C3%A7%C3%B5es-m%C3%B3veis-abbfb71c9aa1>. Acesso em: 2019-12-04. Citado na página 99.

MARQUES, R. *Teste end-to-end*. 2018. Disponível em: <https://www.cedrotech.com/blog/teste-end-to-end/>. Acesso em: 2023-11-30. Citado na página 96.

MONGODB. *The most popular database for modern apps*. 2019. Disponível em: <https://www.mongodb.com/>. Acesso em: 2019-11-11. Citado na página 30.

MONGODB. *MongoDB documentation*. 2023. Disponível em: <https://www.mongodb.com/docs/atlas/app-services/schemas/>. Acesso em: 2023-10-15. Citado 2 vezes nas páginas 31 e 44.

MOREIRA, J. da Conceição da R. *O paradigma das cidades inteligentes*. 2014. Disponível em: <repositorio.ulusiada.pt/bitstream/11067/1870/1/DissertaÃğÃčo%20de%20mestrado.pdf>. Citado na página 15.

MOZZILA. *Introdução Express/Node*. 2023. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Learn/Server-side/Express_Nodejs>. Acesso em: 2023-10-15. Citado na página 29.

OBJECTIVE. *Testes de integração: conheça os tipos e as vantagens em automatizar*. 2023. Disponível em: <https://www.objective.com.br/insights/teste-de-integracao/>. Acesso em: 2023-11-30. Citado 2 vezes nas páginas 93 e 94.

OLIVEIRA, R. B. de; GUAGLIANONI, D. G.; DEMONTE, A. Perfil do usuÁrio, composiÇÃo e adequaÇÃo nutricional do cardÁpio oferecido em um restaurante universitÁrio. *Alimentos e Nutrição*, v. 16, 03 2009. Citado na página 16.

PERES, R. *SQL vs NoSQL*. 2018. Disponível em: <https://www.revista-programar.info/artigos/sql-vs-nosql/>. Acesso em: 2019-12-04. Citado na página 30.

PRESSMAN, R. S. *Engenharia de Software: Uma abordagem profissional*. [S.l.]: AMGH, 2011. v. 7º edição. Citado 2 vezes nas páginas 36 e 93.

RADIGAN, D. *kanban,Como a metodologia Kanban é aplicada ao desenvolvimento de software*. 2023. Disponível em: <https://www.atlassian.com/br/agile/kanban>. Acesso em: 2023-08-28. Citado na página 34.

SABINO, R. *Kanban: o que é, o Método Kanban, principais conceitos e como funciona no dia a dia*. 2023. Disponível em: <https://www.alura.com.br/artigos/metodo-kanban#quais-sao-as-tres-fases-do-kanban?> Acesso em: 2023-08-28. Citado na página 35.

SANTOS, L. *Node.js Por Baixo Dos Panos 4 - Vamos Falar do V8*. 2019. Disponível em: <https://dev.to/khaosdoctor/node-js-por-baixo-dos-panos-4-vamos-falar-do-v8-4pai>. Acesso em: 2019-11-11. Citado na página 27.

SATHEESH, M.; D'MELLO, B. J.; KROL, J. *Web Development with MongoDB and NodeJS*. [S.l.]: Packt Publishing, 2015. Second Edition. Citado 2 vezes nas páginas 27 e 28.

SEOBILITY. *REST API*. 2018. Disponível em: <https://www.seobility.net/en/wiki/REST_API>. Acesso em: 2019-11-02. Citado na página 33.

SILVA, J. H.; LAMPKOWSKI, M. *INTEGRAÇÃO DE SISTEMAS DE INFORMAÇÃO:a importância das integrações e como realizá-las com interoperabilidade de forma a produzirem e consumirem informações reutilizáveis*. 2017. Disponível em: <https://fgp.edu.br/wp-content/uploads/2017/01/P%C3%93S-Integra%C3%A7%C3%A3o-de-Sistemas-de-Informa%C3%A7%C3%A3o-Jose-Henrique-Silva.pdf>. Acesso em: 2019-11-28. Citado na página 15.

SOMMERVILLE, I. *Engenharia de Software*. [S.l.]: Pearson Prentice Hall, 2011. v. 9º edição. Citado 6 vezes nas páginas 34, 36, 37, 38, 93 e 98.

SUBRAMANIAN, V. *Pro MERN Stack*. [S.l.]: Apress, 2019. Second Edition. Citado 2 vezes nas páginas 27 e 31.

TOTVS, E. *Kanban: conceito, como funciona, vantagens e implementação*. 2022. Disponível em: <https://www.totvs.com/blog/negocios/kanban/#:~:text=O%20termo%20%E2%80%9CKanban%E2%80%9D%20%C3%A9%20de,ele%20se%20move%20pelo%20processo.> Acesso em: 2023-08-28. Citado na página 34.

UFCG. *UFCG cria o Smart Campus*. 2019. Disponível em: <https://portal.ufcg.edu.br/ultimas-noticias/878-ufcg-cria-o-smart-campus.html>. Acesso em: 2019-12-04. Citado na página 21.

UFES. *Restaurante Universiário - Alegre-ES*. 2019. Disponível em: <http://ru.alegre.ufes.br/>. Citado na página 17.

YU, Z. et al. Towards a smart campus with mobile social networking. In: *2011 IEEE INTERNATIONAL CONFERENCES ON INTERNET OF THINGS AND CYBER, Physical and Social Computing,iThings/CPSCom 2011*. Dalian, China: [s.n.], 2011. p. 162–169. Citado na página 19.

APÊNDICE A – QUESTIONÁRIO DE USABILIDADE

Nome

Sua resposta

Você enfrentou alguma dificuldade específica durante o uso do aplicativo? Se sim, descreva brevemente.

Sua resposta

Alguma terminologia ou palavra utilizada no aplicativo não estava clara para você? Se sim, forneça exemplos.

Sua resposta

Identifique as principais barreiras que você encontrou durante a interação com o aplicativo. Exemplos podem incluir falta de fluidez, intencionalidade e demora para realizar tarefas.

Sua resposta

Figura 58: Parte 1 do questionário de usabilidade

Identifique quaisquer falhas na indicação de textos que possam ter sido confusas ou mal interpretadas.

Sua resposta

Os botões essenciais estavam bem localizados e fáceis de encontrar? Caso contrário, aponte exemplos.

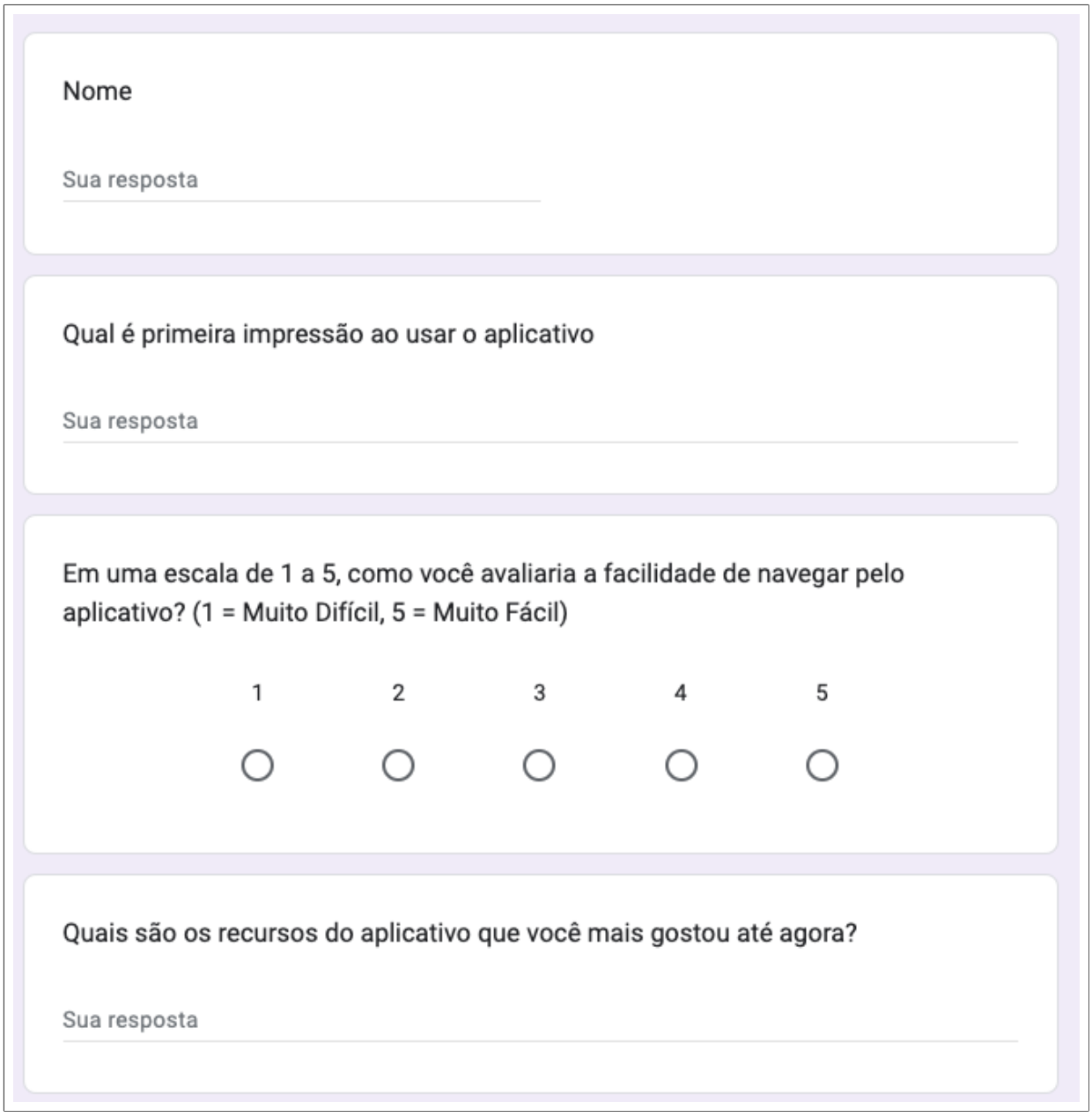
Sua resposta

Com base na sua experiência, que melhorias ou adições você sugeriria para tornar o aplicativo mais fácil de usar?

Sua resposta

Figura 59: Parte 2 do questionário de usabilidade

APÊNDICE B – QUESTIONÁRIO DE USUÁRIOS BETAS



Nome

Sua resposta

Qual é primeira impressão ao usar o aplicativo

Sua resposta

Em uma escala de 1 a 5, como você avaliaria a facilidade de navegar pelo aplicativo? (1 = Muito Difícil, 5 = Muito Fácil)

1 2 3 4 5

Quais são os recursos do aplicativo que você mais gostou até agora?

Sua resposta

Figura 60: Parte 1 do questionário para usuários betas

Em uma escala de 1 a 5, como você avaliaria o design visual do aplicativo? (1 = Muito Ruim, 5 = Excelente)

1 2 3 4 5

Há alguma melhoria específica que você gostaria de sugerir para tornar o aplicativo mais útil ou agradável?

Sua resposta

Em uma escala de 1 a 10, qual seria sua classificação geral para o aplicativo beta? (1 = Muito Insatisfeito, 10 = Muito Satisfeito)

1 2 3 4 5 6 7 8 9 10

Você recomendaria este aplicativo ao alunos? Por quê?

Sua resposta

Figura 61: Parte 2 do questionário para usuários betas

APÊNDICE C – QUESTIONÁRIO SETOR NUTRIÇÃO

Questionário Aplicativo RU

Nome: **Data de Nascimento**
Curso: **Ano de ingresso:**
Sexo: ( ) Masculino ( ) Feminino
Bolsista RU: ( ) Não ( ) Sim
Em caso afirmativo: ( ) bolsa parcial ( ) bolsa integral (almoço e jantar)

1- Com qual frequência você realiza suas refeições no RU (Restaurante Universitário)?
( ) Todos os dias ( ) Pelo menos 3 vezes na semana ( ) Pelo menos 1 vez na semana ( ) Raramente

2- Qual tipo de refeição você costuma realizar no RU?
( ) Almoço ( ) Jantar ( ) Almoço e Jantar

3- Como você se considera de acordo com o seu nível de atividade física?
( ) Sedentário ( ) Leve ( ) Moderado ( ) Ativo

4- Você se considera dentro do peso ideal para a sua idade e sexo?
( ) Sim ( ) Não

5- Você possui algum tipo de alergia ou intolerância alimentar?
( ) Sim ( ) Não

6- Se sim, qual:
( ) Alergia ao glúten ( ) Alergia à proteína do leite de vaca
( ) Intolerância à lactose ( ) Outros:_____

7- Você é vegetariano ou vegano?
( ) Sim ( ) Não

Caso sim, como você se classifica?
( ) ovolactovegetariano
( ) vegetariano restrito – alimentação
( ) vegano

8- Costuma adicionar sal nos alimentos prontos e saladas (você faz uso do saleiro à mesa)?
( ) Sim ( ) Não

Figura 62: Parte 1 do questionário conexionado pelo setor de nutrição

Questionário Aplicativo RU

Nome: **Data de Nascimento**

Curso: **Ano de ingresso:**

Sexo: ( ) Masculino ( ) Feminino

Bolsista RU: ( ) Não ( ) Sim

Em caso afirmativo: ( ) bolsa parcial ( ) bolsa integral (almoço e jantar)

1- Com qual frequência você realiza suas refeições no RU (Restaurante Universitário)?
( ) Todos os dias ( ) Pelo menos 3 vezes na semana ( ) Pelo menos 1 vez na semana ( ) Raramente

9- Você utiliza o óleo composto que fica no balcão de distribuição de refeições?
( ) Sim ( ) Não

10- Você consome bebidas alcoólicas?
( ) Sim ( ) Não

11- Com qual frequência?
( ) Diariamente ( ) de 3-6 vezes na semana ( ) de 1-2 vezes na semana ( ) Raramente

12- Você é tabagista?
( ) Sim ( )Não

13- Você apresenta ou já apresentou alguma das patologias abaixo? Se sim, quais?
( ) Doença cardiovascular ( ) Hipertensão arterial ( ) Obesidade ( ) Dislipidemias
( ) Doença Arterial Coronariana ( ) Diabetes ( ) Outras: _____

14- Há histórico de presença, de alguma das patologias abaixo, na sua família? Se sim, quais?
( ) Doença cardiovascular ( ) Hipertensão arterial ( ) Obesidade
( ) Dislipidemias ( ) Doença Arterial Coronariana ( ) Diabetes ( ) Outras: _____

15- Faz uso contínuo de algum medicamento?
Qual:_____

Como você avalia a refeição servida no RU, quanto aos diferentes atributos:
Aroma das preparações: _____

Legenda:
(1) Muito ruim
(2) Ruim

Figura 63: Parte 2 do questionário conexionado pelo setor de nutrição