



Universidade Federal de Itajubá - *campus* de Itabira
Instituto de Ciências Tecnológicas

SISTEMA DE DIVULGAÇÃO E INTEGRAÇÃO ENTRE USUÁRIOS E RESTAURANTES UNIVERSITÁRIOS

FELIPE WALLACE PINTO COELHO

Itabira MG
2024

FELIPE WALLACE PINTO COELHO

SISTEMA DE DIVULGAÇÃO E INTEGRAÇÃO ENTRE USUÁRIOS E RESTAURANTES UNIVERSITÁRIOS

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia de Computação da Universidade Federal de Itajubá - *campus* de Itabira como parte dos requisitos necessários para a obtenção do grau em Bacharel em Engenharia de Computação.

Orientador: Eduardo Ribeiro Felipe

Itabira - MG
20 de novembro de 2024

Agradecimentos

Gostaria de expressar minha gratidão a todos que, em algum momento da minha trajetória acadêmica, contribuíram para que eu chegassem a esta etapa.

Primeiramente, agradeço a Deus, por sua atuação maravilhosa ao longo de todos esses anos. Apesar das adversidades, senti Sua presença nos momentos em que mais precisei de ajuda, guiando-me rumo às melhores escolhas.

Agradeço também aos meus familiares, em especial ao meu pai e à minha mãe, que foram pilares fundamentais para que eu não desistisse do caminho e dos sonhos que escolhi para minha vida. À minha mãe, em particular, dedico um agradecimento especial, por assumir o papel de psicóloga, conselheira, amiga e tantos outros, sempre com amor e dedicação. Ela é, sem dúvida, a definição perfeita de mãe.

Aos meus amigos, expresso minha gratidão pela companhia, não apenas nos momentos de descontração, que trouxeram leveza à rotina, mas também nas ocasiões de trabalho em grupo, que foram essenciais para nossa progressão conjunta na universidade.

Por fim, agradeço ao Eduardo, que atuou como um verdadeiro orientador ao longo deste projeto. Sua presença constante, apoio e disposição para sanar dúvidas e inseguranças foram fundamentais. Além de ser um excelente professor, Eduardo é um verdadeiro formador de profissionais.

Resumo

O Restaurante Universitário é uma infraestrutura de grande importância nas instituições de ensino. O processo de comunicação e interação com a comunidade é igualmente importante. Neste trabalho pode-se acompanhar o desenvolvimento de uma aplicação web para a divulgação e integração entre usuários e o Restaurante Universitário da Universidade Federal de Itajubá - *Campus Itabira*. A solução proposta visa substituir os métodos tradicionais e pouco eficientes de divulgação do cardápio, atualmente feitos por meio de impressões e e-mails, e implementar um sistema moderno e interativo que permita atualizações diárias e avaliações diretas dos pratos servidos. A aplicação permitirá que os estudantes accessem o cardápio atualizado em tempo real e forneçam feedback sobre as refeições. A integração de uma interface amigável visa melhorar a experiência dos usuários e otimizar a operação do restaurante universitário, promovendo uma comunicação mais direta e eficaz entre os estudantes e a gestão do restaurante.

Palavras-chave: Restaurante Universitário, Cardápio Digital, Avaliação de Usuários, Aplicação Web, Gestão de Cardápios.

Abstract

The University Restaurant is a vital infrastructure within educational institutions, playing a significant role in supporting the academic community. Equally important is the process of communication and interaction with users. This study presents the development of a web application aimed at improving the dissemination and integration between users and the University Restaurant at the Federal University of Itajubá - Campus Itabira. The proposed solution seeks to replace traditional and inefficient methods of menu distribution, currently relying on printed materials and emails, with a modern and interactive system that enables daily updates and direct feedback on the served dishes. The application allows students to access the updated menu in real time and provide feedback on the meals. By incorporating a user-friendly interface, the solution aims to enhance user experience and streamline the operation of the university restaurant, fostering more direct and effective communication between students and restaurant management.

Keywords: University Restaurant, Digital Menu, User Evaluation, Web Application, Menu Management.

Lista de figuras

Figura 1 – Diagrama de Arquitetura Inicial do Projeto	18
Figura 2 – Diagrama Entidade-Relacionamento Inicial do Banco de Dados	19
Figura 3 – Relacionamento entre as tabelas Cardapio e Item	21
Figura 4 – Diagrama Entidade-Relacionamento Final do Banco de Dados	22
Figura 5 – Diagrama de Arquitetura do Projeto	28
Figura 6 – Página principal com exibição do cardápio do dia	29
Figura 7 – Exibição detalhada das notas e comentários sobre os cardápios	30
Figura 8 – Campos de Avaliação	30
Figura 9 – Menu administrativo na lateral da interface	32
Figura 10 – Página exclusiva para a manipulação de cardápios	33
Figura 11 – Componente para adição de itens aos cardápios	34
Figura 12 – Componente para a remoção de itens do cardápio	34
Figura 13 – Página de Gerenciamento de Itens	35
Figura 14 – Página de Configuração de Avisos	36

Listas de abreviaturas e siglas

API	<i>Application Programming Interface</i>
CSS	<i>Cascading Style Sheets</i>
DER	Diagrama Entidade-Relacionamento
DOM	<i>Document Object Model</i>
HTML	<i>HyperText Markup Language</i>
HTTP	<i>Hypertext Transfer Protocol</i>
NPM	<i>Node Package Manager</i>
PNAES	Plano Nacional de Assistência Estudantil
REST	<i>Representational State Transfer</i>
RU	Restaurante Universitário
SQL	<i>Structured Query Language</i>
Unifei	Universidade Federal de Itajubá

Sumário

1	INTRODUÇÃO	1
1.0.1	Justificativa	2
1.0.2	Objetivos	2
1.0.3	Organização do Trabalho	3
2	REFERENCIAL TEÓRICO	5
2.1	Trabalhos Relacionados	5
2.2	Restaurante Universitário	5
2.3	Banco de Dados	6
2.3.1	Relacional	7
2.3.2	NoSQL	7
2.4	<i>Application Programming Interface</i>	8
2.4.1	REST	9
2.4.2	GraphQL	10
2.5	Interfaces Web	10
2.5.1	HTML	11
2.5.2	CSS	11
2.5.3	JavaScript	12
2.6	Node.js	13
2.7	Framework	14
2.7.1	Express	14
2.7.2	React	15
2.7.3	Next.js	16
2.8	OAuth	17
3	METODOLOGIA	18
3.1	Desenvolvimento do protótipo	20
3.1.1	Back-end	20
3.1.1.1	Banco de Dados	20
3.1.1.2	API	22
3.1.2	Autenticação	26
3.1.3	Front-end	28
4	DISCUSSÃO	37
5	CONCLUSÃO	39

Referências	40
Anexos	42
ANEXO A – SCRIPT DE CRIAÇÃO DO BANCO DE DADOS	43
ANEXO B – REPOSITÓRIOS DO PROJETO	46
B.1 API	46
B.2 Front-end	46

1 Introdução

Por muito tempo, apenas os mais ricos tinham acesso ao ensino superior no Brasil, devido à grande desigualdade social e aos altos custos das universidades. Além disso, a discrepância entre a qualidade do ensino público e privado, juntamente com a baixa retenção de estudantes vulneráveis, representava um grande desafio (ARIOVALDO, 2016).

Com o intuito de enfrentar essa realidade, foi implementado em 2010 o Plano Nacional de Assistência Estudantil (PNAES), cujo objetivo era fornecer suporte para a permanência de estudantes de baixa renda matriculados em cursos de graduação presencial nas instituições federais de ensino superior (IFES). Dessa forma, buscou-se promover a igualdade de oportunidades entre os estudantes, contribuindo para a melhoria do desempenho acadêmico e para a redução da disparidade econômica no Brasil (SOUZA; FAVA; CINTRA, 2023).

Em uma instituição de ensino superior, onde é fundamental o desenvolvimento da criatividade, da capacidade de resolver problemas e do aspecto social do indivíduo, é essencial garantir que esses aspectos sejam protegidos de influências externas, como o custo de frequentar a universidade ou a falta de acesso a um ambiente propício para o progresso acadêmico. Isso inclui o acesso a conhecimentos e espaços de estudo adequados e também a uma nutrição apropriada.

É amplamente reconhecido que uma alimentação saudável tem impacto no desempenho geral de uma pessoa, e isso não é diferente em um ambiente acadêmico. Por isso, é comum que as universidades integrem restaurantes de baixo custo em suas instalações para atender às necessidades nutricionais dos alunos.

Os Restaurantes Universitários (RUs) são uma opção acessível e conveniente para os estudantes do *campus*, oferecendo uma alimentação adequada e financeiramente viável. Eles desempenham um papel fundamental no desenvolvimento profissional, otimizando o tempo dos estudantes e facilitando suas rotinas diárias, especialmente quando têm horários variados de aula.

Os RUs são atraentes porque oferecem refeições de qualidade a preços acessíveis, proporcionando uma experiência vantajosa em termos de custo-benefício. Além disso, o serviço de alimentação coletiva garante refeições equilibradas e preparadas com os devidos cuidados higiênicos, atendendo às necessidades dos alunos que têm períodos limitados para se deslocar durante as refeições.

No entanto, isso não os isenta das críticas. Um caminho para promover o aperfeiçoamento desse ambiente, está na construção de um canal de comunicação entre os usuários deste sistema e os responsáveis por gerenciá-lo a fim de tornar o restaurante continuamente melhor.

Atualmente, na Universidade Federal de Itajubá (Unifei), no *campus* de Itabira, o cardápio é divulgado de forma tradicional, por meio de uma impressão fixada na porta do restaurante, e

sua versão digital, no site da universidade, que é pouco difundida entre os estudantes e demais membros da comunidade, não só é algo pouco usual para estes, mas é igualmente ruim para os responsáveis pelo empreendimento, que veem a divulgação como pouco convidativa.

Uma alternativa de comunicar o cardápio via e-mail também é feita, mas esta muitas vezes passa despercebida e, por um tempo, foi descontinuada. De forma parecida, o sistema de avaliação nunca teve meios efetivos para ser realizado de forma contínua por aqueles que recorrem ao refeitório, se limitando ao máximo a botões físicos com poucas opções em suas instalações, algo que também foi suspenso.

Uma pesquisa de satisfação é realizada semestralmente com os utilizadores do RU, entretanto esta não fornece informações de cardápios específicos, levando em consideração a qualidade do serviço na totalidade naquele determinado período. Em casos mais extremos de insatisfação, os estudantes precisam recorrer aos canais de comunicação da universidade para apresentarem suas queixas. Esse tipo de situação também é vista como incomoda para a gestão do restaurante, uma vez que as queixas dos estudantes poderiam ser resolvidas de forma mais amigável se houvesse um canal direto de comunicação entre as partes.

Diante desse cenário surge o seguinte questionamento: como desenvolver uma solução tecnológica que atenda às necessidades dos estudantes universitários e dos responsáveis pelo restaurante do *campus* de Itabira da Unifei, proporcionando uma comunicação mais eficiente e transparente do cardápio?

Alia-se a esse questionamento inicial, também, a possibilidade desta solução trabalhar com um sistema de avaliação acessível e interativo, visando melhorar a experiência dos usuários e facilitar a resolução de problemas e queixas relacionados ao serviço de alimentação.

1.0.1 Justificativa

Este trabalho tem como objetivo desenvolver uma alternativa para que estudantes e todos os que são diretamente afetados pelo restaurante universitário tenham acesso a uma aplicação tecnológica. Neste contexto, a aplicação será um software no modelo web de comunicação, que permitirá uma melhor interação entre as pessoas que utilizam o RU. Essa solução deve iniciar com elementos de comunicação básica como a divulgação do cardápio de maneira direta e simplificada e que, ao mesmo tempo, dê voz para os usuários poderem opinar, seja avaliando a qualidade dos pratos servidos, do ambiente e até os preços praticados. Desta forma, procura-se promover uma constante busca pela qualidade das refeições oferecidas.

1.0.2 Objetivos

O RU é uma instituição que fornece grande apoio aos estudantes, servidores e demais profissionais envolvidos na jornada acadêmica da universidade, servindo como um dos pilares do

bem-estar em uma instituição de ensino, portanto, é importante que este atenda as necessidades de seus usuários e tenha ferramentas eficazes de comunicação com seus clientes.

Propondo tornar essa comunicação eficiente e clara entre os indivíduos que fazem uso e fornecem o serviço do RU, este trabalho será direcionado por meio do objetivo geral em: *implementar uma aplicação web para facilitar o acesso dos estudantes universitários ao cardápio do restaurante da instituição*, promovendo uma experiência alimentar mais transparente, conveniente e interativa, como também fornecer aos administradores do restaurante as ferramentas que podem ajudar na gestão de tomadas de decisões que melhor atendam à comunidade.

Dentre os objetivos específicos deste trabalho, destaca-se:

- Compreender a estrutura organizacional do restaurante universitário e suas características.
- Analisar as necessidades e requisitos dos estudantes universitários em relação à divulgação do cardápio do restaurante universitário e avaliação deste.
- Projetar e implementar uma interface de usuário intuitiva e responsiva para a aplicação web, garantindo facilidade de acesso e navegação para os usuários, seja em navegadores para computadores ou de dispositivos móveis.
- Construir uma estrutura envolvendo uma base de dados para o sistema.
- Preparar uma *Application Programming Interface* (API) que permita o desenvolvimento escalável para outras formas de interação com o banco de dados, a exemplo de aplicações mobile em conjunto com a aplicação web.
- Desenvolver um sistema de gerenciamento de cardápio que permita a atualização diária ou semanal das opções de refeições oferecidas pelo restaurante da universidade.
- Implementar um mecanismo de avaliação do cardápio que permita aos usuários compartilharem feedbacks sobre a qualidade e a satisfação com as refeições servidas.

1.0.3 Organização do Trabalho

Este trabalho está estruturado em capítulos que exploram de forma organizada e detalhada as diferentes etapas do desenvolvimento de um sistema de divulgação e integração entre usuários e restaurantes universitários, que tem como base o RU da Unifei - *Campus Itabira*. Cada capítulo aborda aspectos fundamentais do projeto, desde a motivação inicial e os conceitos teóricos até a implementação e análise dos resultados obtidos.

O Capítulo 2 explora os conceitos e tecnologias que fundamentam a solução proposta, como bancos de dados, APIs, interfaces web, frameworks modernos como Express, React e Next.js, além do protocolo de autenticação OAuth. Essa seção serve como base para o entendimento das escolhas técnicas realizadas ao longo do desenvolvimento.

O [Capítulo 3](#) detalha as etapas práticas da criação do sistema, divididas entre as camadas de back-end, autenticação e front-end. Essa seção descreve a implementação do banco de dados, o design e desenvolvimento da API, a integração de sistemas de autenticação e a construção de uma interface interativa e responsiva para os usuários.

O [Capítulo 4](#) reflete sobre os resultados alcançados e os desafios enfrentados durante o desenvolvimento. Essa seção avalia a eficácia das escolhas feitas e identifica as limitações do sistema.

Por fim, o [Capítulo 5](#) sintetiza os principais resultados, avaliando o cumprimento dos objetivos propostos e destacando as contribuições do trabalho. Ela também oferece considerações finais e propõe possíveis melhorias e expansões para trabalhos futuros.

2 Referencial Teórico

2.1 Trabalhos Relacionados

No desenvolvimento do trabalho, dedicou-se tempo e esforços para conhecer outras pesquisas que dissertaram sobre o tema e puderam contribuir para o entendimento deste campo.

Dentre os trabalhos encontrados, pode-se destacar os seguintes autores: ([SIQUEIRA, 2020](#)) que publicou um artigo e demonstrou como desenvolveu uma aplicação móvel para o RU da Universidade de Brasília, visando combater as grandes filas que se formavam durante os períodos de almoço na instituição. Ainda que tivesse objetivos em comum com este trabalho, seu foco na compra de tickets para o refeitório demonstra que cada universidade por si só têm dinâmicas únicas de funcionamento, o que evidencia a necessidade de soluções exclusivas para cada uma delas, tendo em vista a rotina dos estudantes e funcionamento do *campus*.

([TOTO, 2021](#)) demonstrou que a necessidade de aprimoramentos não é algo único das universidades ou intuições públicas. Em seu artigo, a autora propôs e desenvolveu um cardápio virtual para um restaurante de uma mineradora, visando validar aspectos nutricionais e a satisfação dos funcionários. Seu projeto compartilha funcionalidades em comum com as que serão implementadas neste trabalho, como a disponibilização de um cardápio acessível aos usuários, além de ter usado tecnologias de desenvolvimento atuais que serão de grande utilidade e também serão discutidas nos próximos tópicos.

2.2 Restaurante Universitário

O RU é um serviço de alimentação subsidiado oferecido por diversas universidades brasileiras aos seus alunos. No caso do *campus* de Itabira da Universidade Federal de Itajubá (Unifei), este auxílio é realizado por meio da isenção de custos estruturais aos vencedores da licitação, como os custos de limpeza e aluguel, o que possibilita uma tarifa menor em relação aos outros estabelecimentos da cidade. Futuramente este subsídio também será feito com o auxílio de custo em cada refeição, tornando-as ainda mais acessíveis.

O funcionamento dos RUs varia de acordo com cada universidade, mas geralmente segue um modelo similar: o cardápio do RU é elaborado por nutricionistas e oferece opções variadas de pratos, incluindo alternativas vegetarianas e as refeições custam um valor inferior ao praticado em restaurantes convencionais.

Restaurantes Universitários tem como principal missão produzir e fornecer refeições de qualidade higiênico-sanitária e nutricional satisfatória a um custo reduzido e/ou sem custo para os estudantes ([CARDOSO et al., 2018](#)), contribuindo para a segurança alimentar e o bem-estar

da comunidade acadêmica que utiliza esta instituição, via iniciativas como a promoção de uma alimentação saudável, a oferta de opções variadas e balanceadas de refeições e a implementação de práticas de higiene e segurança alimentar rigorosas.

Os RUs desempenham um papel crucial na vida dos estudantes universitários, especialmente para aqueles que possuem dificuldades financeiras. Além de fornecer alimentação nutritiva a preços baixos, os RUs também servem como espaços de socialização e integração entre os estudantes. Eles atuam como ferramentas importantes na assistência estudantil, especialmente para os alunos com baixo poder aquisitivo ou que residem a grandes distâncias do ambiente onde estudam (ROHR; MASIERO; NETO, 2010).

Para o desenvolvimento de uma solução de software para a melhor comunicação entre a comunidade acadêmica e o RU vamos considerar no próximo tópico os Bancos de Dados como software para gravação e leitura das informações.

2.3 Banco de Dados

Um banco de dados é projetado, construído e populado com dados para uma finalidade específica. Ele possui um grupo definido de usuários e algumas aplicações previamente concebidas nas quais esses usuários estão interessados (ELMASRI; NAVATHE, 2011). Em outras palavras, é um conjunto organizado de dados estruturados, armazenados e gerenciados eletronicamente. O sistema é protegido contra acessos não autorizados e alterações indevidas mediante mecanismos de segurança.

(DATE, 2004) compara o conceito de um sistema de banco de dados a um armário de arquivamento eletrônico, ressaltando que é um local onde se armazena uma coleção de arquivos de dados computadorizados.

Algumas operações típicas que podem ser realizadas em um banco de dados, incluem adicionar e remover arquivos, inserir, buscar, excluir e alterar dados em arquivos já existentes. Essas operações são fundamentais para a gestão de informações em sistemas informatizados, permitindo que os usuários interajam com os dados de maneira eficiente e organizada.

Segundo (SILVA; FERREIRA, 2017) a utilização de um banco de dados tornou-se um requisito essencial na gestão da informação para a maioria das empresas. Geralmente, as empresas utilizam um software de gestão para administrar todas as suas transações ao longo dos anos. A maioria delas opta pelo modelo de banco de dados Relacional, que é amplamente aceito em situações em que a base de dados é menor e as buscas de informações são mais simples. Esse modelo oferece uma estrutura organizada e eficiente para armazenar e recuperar dados de forma confiável.

No entanto, à medida que a base de dados começa a crescer devido ao aumento das organizações e à necessidade de manter um histórico das transações, os modelos não relacionais,

ou NoSQL, tornam-se cada vez mais eficientes como veremos adiante.

2.3.1 Relacional

Bancos de dados relacionais armazenam dados em formas estruturadas denominadas tabelas, com linhas e colunas, onde cada linha representa uma entidade (registro) e cada coluna representa um atributo (característica) dessa entidade. As tabelas estão interligadas por meio de relacionamentos definidos por chaves primárias, que são identificadores únicos, e chaves estrangeiras que permitem o acesso aos dados entre as tabelas, por meio deste mesmo tipo de identificador.

Esta arquitetura oferece uma estrutura organizada e lógica, facilitando a consulta e manipulação de dados, além de fornecer suporte a transações ACID (atomicidade, consistência, isolamento e durabilidade). Bancos de dados relacionais são mais apropriados para aplicações que demandam consistência e integridade nos dados. Exemplos populares incluem: MySQL e PostgreSQL.

Segundo (OLIVEIRA, 2014):

[...] esse modelo ainda é amplamente utilizado pelo fato de prover acesso facilitado aos dados, possibilitando aos usuários utilizar uma grande variedade de abordagens no tratamento das informações, além da possibilidade de uso dos sistemas gerenciadores de bancos de dados, que executam comandos na linguagem SQL (*Structured Query Language*) e têm a responsabilidade de gerenciar o acesso, a manipular e a organizar os dados, principalmente no que diz respeito à segurança.

Além do modelo relacional, outra opção são os bancos de dados NoSQL. A escolha entre os modelos deve ser guiada pelos requisitos específicos do projeto. Bancos de dados relacionais são uma escolha sólida para aplicativos que exigem consistência de dados, transações complexas e integridade referencial. Por outro lado, bancos de dados NoSQL oferecem uma solução flexível e escalável como será detalhado no próximo tópico.

2.3.2 NoSQL

Também conhecido como banco de dados não-relacional, esta categoria armazena dados em estruturas não tabulares, como documentos, grafos ou chave-valor, sendo mais flexíveis e escaláveis do que os bancos de dados relacionais. São adequados para armazenar grandes volumes de dados não estruturados, devido à flexibilidade no modelo de dados e alta escalabilidade. No entanto, possuem uma estrutura menos organizada, tornando as consultas e manipulações de dados mais complexas. (DIANA; GEROSA, 2010) explicam outro fator.

[...] Assim como o termo não-relacional, o termo NoSQL não ajuda a definir o que esses bancos são de fato. Além do problema da falta de precisão, esse termo também tem contribuído para uma grande confusão em torno dessa categoria de bancos de dados, já que a princípio a linguagem SQL não é sinônimo de bancos de dados relacionais, nem representa as limitações desses bancos de dados. Devido a isso, o termo NoSQL tem sido usado com o significado de “Não apenas SQL” numa tentativa da comunidade de reconhecer a utilidade dos modelos tradicionais e não divergir as discussões.

Após compreender a importância dos bancos de dados no armazenamento e gerenciamento de grandes volumes de informações estruturadas e não estruturadas, é crucial explorar como essas informações são acessadas e utilizadas pelas aplicações. É neste ponto que entra a *Application Programming Interface* (API) que funciona como uma ponte entre o banco de dados e a interface do usuário.

2.4 Application Programming Interface

Interface de Programação de Aplicativos (API) é um termo em inglês que se refere a interfaces de programação que permitem a conexão e a troca de informações entre diferentes programas, estabelecendo uma série de regras e especificações que possibilitam a interação entre eles. Isso permite que os desenvolvedores criem aplicações que integrem funcionalidades de outras, sem a necessidade de reimplementá-las, o que acelera o desenvolvimento.

API serve para interligar, fornecer suas funcionalidades ou rotinas, para mais de um sistema, sem que as partes saibam detalhes internos de sua implementação (JUNIOR; ROCHA; MACIEL, 2021). É importante destacar que uma API não se limita a uma única linguagem de programação, uma vez que sua capacidade ultrapassa as barreiras linguísticas. Por exemplo, uma API desenvolvida para fornecer acesso a um serviço específico pode ser utilizada por uma variedade de sistemas e aplicativos, independentemente da linguagem em que foram desenvolvidos. Isso significa que uma API pode ser criada em uma linguagem e ser facilmente integrada a diferentes sistemas que utilizam linguagens distintas, proporcionando interoperabilidade entre eles.

Também existem APIs baseadas em Serviços da Web, que desempenham um papel crucial na integração de sistemas distribuídos, proporcionando uma forma padronizada de comunicação entre aplicativos pela internet ou redes locais. Conforme explicam (JUNIOR; ROCHA; MACIEL, 2021):

[...] De modo prático, os recursos da API são compartilhados através de Web Services e utiliza XML ou JSON como formato de comunicação, ou seja, quando um cliente faz uma requisição, é retornado uma resposta no formato XML ou JSON, quando a API precisa de alguma informação do cliente, o formato enviado também precisa ser o mesmo.

Elas facilitam a troca de dados e a realização eficiente e confiável de operações entre diferentes sistemas de software, independentemente das tecnologias utilizadas para implementá-las. São amplamente adotadas em diversos cenários, como integração de sistemas empresariais, desenvolvimento de aplicativos móveis e construção de ecossistemas de serviços na nuvem. Por meio do uso de protocolos como GraphQL e *Representational State Transfer* (REST), essas APIs proporcionam uma maneira flexível e interoperável para que aplicativos clientes acessem funcionalidades e recursos disponibilizados por servidores remotos.

2.4.1 REST

Em uma API REST, os recursos são representados por URIs (*Uniform Resource Identifiers*), que são os identificadores únicos dos recursos disponibilizados pelo servidor. REST foi introduzido por Roy Fielding em sua tese de doutorado em 2000 e se tornou uma das abordagens mais populares para a construção de APIs web devido à sua simplicidade, escalabilidade e flexibilidade. Os clientes interagem com esses recursos por meio de operações padrão da web, como GET, POST, PUT e DELETE, que correspondem aos métodos HTTP.

- GET: o método GET é usado para recuperar dados do servidor, sendo uma operação segura e idempotente, o que significa que executar o mesmo método GET várias vezes produzirá o mesmo resultado e não causará alterações no servidor.
- POST: este método é usado para submeter dados ao servidor para processamento, como criar um novo registro em um banco de dados. No entanto, o uso repetido pode causar sobrecarga nos recursos do servidor.
- PUT: tem como objetivo atualizar ou substituir de forma completa a representação de um recurso existente com os dados fornecidos na solicitação. Assim como o método GET, executá-lo várias vezes produzirá o mesmo resultado.
- DELETE: O método DELETE é utilizado para excluir permanentemente um recurso específico do servidor. Após a exclusão, a operação pode ser realizada diversas vezes sem que nada seja alterado, uma vez que o item deixa de existir após a primeira utilização.

A API do tipo REST é simples e intuitiva, fácil de entender e implementar. Cada requisição é independente, sem necessidade de manter o estado da sessão e é flexível, suportando diversos formatos de dados (JSON, XML, etc.), o que a torna adequada para aplicações complexas e com alto volume de tráfego. Por outro lado, requer implementação de mecanismos de segurança adicionais.

2.4.2 GraphQL

GraphQL é uma linguagem de consulta para APIs, criada pelo Facebook em 2012 e aberta ao público em 2015. Sua principal vantagem é oferecer uma alternativa eficiente e flexível às APIs tradicionais, permitindo que os clientes solicitem exatamente os dados de que precisam, e nada mais.

Conforme explica (BITTENCOURT, 2021a), GraphQL permite especificar a estrutura exata da resposta. Isso é possível graças à sua natureza declarativa, onde os clientes enviam uma única consulta ao servidor, especificando os campos e aninhamentos de dados desejados. O servidor, por sua vez, processa essa consulta e retorna exatamente os dados solicitados, otimizando a transferência de dados e reduzindo a necessidade de múltiplas requisições. A transmissão dos dados ocorre de duas maneiras: pelo método GET, os dados são enviados por meio de uma consulta com parâmetros; pelo método POST, os dados são transferidos dentro do corpo da solicitação.

Outro benefício significativo do GraphQL é sua tipagem forte e autodescritiva. Um esquema GraphQL define os tipos de dados e as relações entre eles, permitindo que os clientes saibam exatamente quais consultas são possíveis e quais dados podem ser retornados. Um cliente pode fazer uma única consulta que reúne dados de várias fontes. Isso é particularmente útil em arquiteturas de microsserviços, onde os dados podem estar distribuídos por diferentes serviços.

2.5 Interfaces Web

Interface é o nome dado a toda a porção de um sistema (hardware e software) com a qual um usuário mantém contato ao utilizá-lo, tanto passiva quanto ativamente (BARBOSA et al., 2002).

Em outras palavras, uma interface, também conhecida como interface gráfica do usuário (GUI), é a parte visual de um aplicativo web com a qual os usuários interagem. Ela é responsável por apresentar de forma visual e interativa os dados e funcionalidades do aplicativo, permitindo que os usuários compreendam e manipulem as informações de maneira intuitiva. A interface é a primeira coisa que os usuários veem e interagem, sendo fundamental para a experiência. Ela traduz as informações e funcionalidades em elementos visuais, como botões, menus, campos de texto, entre outros, tornando a interação mais amigável e acessível.

Embora os termos interface web e *front-end* sejam frequentemente usados de forma intercambiável, eles possuem distinções sutis. Enquanto interface web se concentra na parte visual e funcional com a qual os usuários interagem em um site ou aplicativo web, *front-end* abrange o processo de desenvolvimento dessa interface. Isso envolve a criação e implementação dos aspectos visuais e interativos, bem como a utilização de tecnologias como *HyperText Markup Language* (HTML), *Cascading Style Sheets* (CSS) e JavaScript para construir a experiência do

usuário.

A base de qualquer interface web é o HTML, a linguagem de marcação que estrutura e organiza o conteúdo nas páginas web. A seguir, vamos aprofundar nossa compreensão sobre o HTML, detalhando como ele serve de alicerce para a construção de interfaces web eficazes.

2.5.1 HTML

Desenvolvida por Tim Berners-Lee em 1991, o HTML é essencialmente a espinha dorsal de qualquer página na web, fornecendo a estrutura básica sobre a qual outros elementos podem ser aplicados.

HTML é a linguagem padrão utilizada para criar e estruturar páginas da web. Ela fornece uma estrutura básica e semântica para organizar o conteúdo de um documento web, permitindo que os navegadores interpretem e exibam essas páginas corretamente para os usuários. Como explica (SANTOS, 2023):

[...]O HTML funciona como uma linguagem de marcação, utilizando tags para descrever a estrutura e o conteúdo dos documentos web. Essas tags são responsáveis pela formatação e organização de diversos elementos, como texto, imagens, áudio, vídeo e gráficos. Ao utilizar as tags do HTML, os desenvolvedores podem apresentar de forma eficaz recursos variados na web.

Os elementos são utilizados para definir diferentes partes de uma página, como cabeçalhos, parágrafos, listas, links, imagens e muito mais. Cada elemento é representado por uma tag, inserida no código HTML e envolve o conteúdo ao qual se aplica.

Embora o HTML forneça a estrutura básica e o conteúdo de uma página web, por si só não é suficiente para criar uma interface visualmente atraente e esteticamente agradável. Neste ponto, o CSS desempenha um papel fundamental, como será abordado no próximo tópico.

2.5.2 CSS

CSS é uma linguagem de estilo utilizada para controlar a apresentação e o layout de páginas da web. Ela permite definir o visual e o design de elementos HTML, possibilitando criar páginas com estilos visualmente atraentes e consistentes. Os estilos são aplicados aos elementos HTML por meio de regras. Cada regra de estilo consiste em um seletor, que especifica quais elementos serão estilizados, e um conjunto de propriedades e valores, que definem como esses elementos serão apresentados.

Com o CSS, é possível definir propriedades como cores, alinhamento, fundo e outros atributos visuais. Ele trabalha por meio de seletores, onde cada tag do HTML ou ID utilizados na construção da página pode ser selecionado e estilizado individualmente (SANTOS, 2023).

As propriedades CSS podem controlar uma ampla variedade de características visuais, como tamanho, fonte, espaçamento e efeitos especiais. Além disso, o CSS permite a criação de layouts complexos e responsivos, permitindo que os desenvolvedores controlem a aparência da página em diferentes dispositivos e tamanhos de tela.

Uma de suas principais vantagens é a capacidade de separar o conteúdo (HTML) da apresentação (CSS), o que torna o código mais limpo, modular e fácil de manter. Além disso, o CSS também oferece recursos avançados, como herança, cascata e seletores específicos, que permitem uma maior flexibilidade e controle sobre o estilo de uma página da web.

2.5.3 JavaScript

JavaScript é uma linguagem de programação de alto nível, interpretada e multiplataforma, utilizada principalmente para tornar páginas da web interativas e dinâmicas. Ele é uma das tecnologias fundamentais da web moderna, permitindo aos desenvolvedores criar uma ampla variedade de funcionalidades e recursos que tornam a experiência do usuário mais rica e envolvente.

O código era limitado a execução no navegador do usuário, o que implica em uma interação rápida e responsiva sem a necessidade de recarregar a página. Ele é frequentemente usado para manipular elementos HTML, responder a eventos do usuário, validar formulários, realizar chamadas de API e criar animações e efeitos visuais. Uma das características mais poderosas do JavaScript é sua capacidade de trabalhar com o Modelo de Objeto de Documento (DOM), que é uma representação em árvore de todos os elementos de uma página da web. Isso permite manipular dinamicamente a estrutura e o conteúdo de uma página, criando experiências interativas e personalizadas para os usuários.

A linguagem está em constante evolução, com uma comunidade ativa e uma vasta biblioteca de frameworks e bibliotecas disponíveis, como React, Angular e Vue.js, que facilitam o desenvolvimento de aplicativos web complexos e sofisticados. Além disso, pode ser usada tanto no lado do cliente quanto no lado do servidor, graças ao Node.js. Inclusive existe uma extensão da linguagem chamada de TypeScript, de código aberto e desenvolvida pela Microsoft, a sintaxe adiciona tipos estáticos opcionais ao JavaScript, permitindo escrever códigos mais robustos e escaláveis.

O TypeScript foi criado para auxiliar os desenvolvedores a evitarem erros recorrentes de programação, fornecendo uma verificação de tipo durante o desenvolvimento. Isso significa que é possível declarar tipos para variáveis, parâmetros de função e retornos de função. Essa prática auxilia na detecção de erros de tipagem e facilita a manutenção do código em projetos de grande escala. (RABELO, 2018) acrescenta:

[...] O poder real no TypeScript vem de suas ferramentas. Os tipos são um meio de levar ferramentas de classe mundial à linguagem JavaScript, o que permite projetos melhores estruturados e mais fáceis de manter. Isso é especialmente

importante à medida que os projetos JavaScript aumentam de tamanho (seja em linhas de código ou desenvolvedores no projeto). Ter uma conclusão rápida e precisa, recursos de refatoração e feedback imediato faz do TypeScript a linguagem ideal para JavaScript em grande escala.

Embora o JavaScript tenha sido originalmente desenvolvido para ser executado no lado do cliente, evoluiu para ser uma linguagem versátil, capaz de operar no lado do servidor graças ao Node.js. A seguir, vamos examinar o Node.js com mais cuidado, analisando como ele permite o desenvolvimento de aplicações completas utilizando uma única linguagem de programação.

2.6 Node.js

Node.js é um ambiente de execução JavaScript de código aberto, baseado no motor V8 do Google Chrome, que permite executar JavaScript no lado do servidor. Desenvolvido por Ryan Dahl em 2009, é projetado para construir aplicações de rede escaláveis e de alto desempenho, oferecendo uma alternativa eficiente para criar servidores web e aplicações em tempo real.

Conforme explicado por (PEREIRA, 2014), uma das características mais importantes do Node.js é seu modelo de E/S (entrada/saída) não-bloqueante e orientado a eventos. Isso significa que o Node.js pode gerenciar várias conexões simultâneas com alta eficiência, sem bloquear o thread principal. Em vez de esperar que uma operação de E/S, como a leitura de um arquivo ou uma consulta a um banco de dados, seja concluída, o Node.js registra um *callback* e continua executando outras operações. Esse modelo é adequado para aplicações que requerem alto desempenho e escalabilidade, como servidores web, APIs e serviços em tempo real, como chats e jogos online.

O ecossistema do Node.js é enriquecido por seu gerenciador de pacotes, o *Node Package Manager* (NPM), que permite que os desenvolvedores instalem, compartilhem e reutilizem pacotes de software, facilitando a construção e manutenção de aplicações complexas. Com milhares de pacotes disponíveis, os desenvolvedores podem facilmente integrar funcionalidades adicionais às suas aplicações.

Além disso, Node.js é amplamente utilizado em combinação com frameworks e bibliotecas populares, como Express para construção de APIs, aplicações web e para interações com bancos de dados. Isso permite criar aplicações robustas e multifuncionais com facilidade. O Node.js também se destaca por sua arquitetura modular e sua capacidade de utilizar JavaScript no lado do servidor, proporcionando uma linguagem comum para o desenvolvimento full-stack. Isso reduz a complexidade e melhora a colaboração entre as equipes de desenvolvimento, já que os mesmos conceitos e ferramentas podem ser aplicados tanto no front-end quanto no back-end.

2.7 Framework

Um framework de desenvolvimento de software é um conjunto de componentes que fornece uma estrutura básica para a criação de aplicações. Ele funciona como um esqueleto pré-construído, poupando tempo e esforço, já que não é necessário implementar funcionalidades comuns, pois inclui componentes e módulos reutilizáveis que podem ser aproveitados em diferentes partes de uma aplicação, reduzindo a quantidade de código que precisa ser escrito do zero. (BITTENCOURT, 2021b) contextualiza:

[...] um frame representa a estrutura de uma casa, na qual só existem as paredes de tijolos levantadas ou mesmo um carro somente com a lataria, como vemos nas montadoras. Essas estruturas estão aguardando o desenvolvimento (work), que no exemplo da casa, corresponderia à escolha da massa corrida que será colocada nas paredes, as cores das tintas, os tipos de pisos. No caso do carro, se serão colocados bancos de tecido ou couro, qual será a cor do painel, vidros escuros ou claros e demais componentes.

Os frameworks disponibilizam bibliotecas e componentes prontos para uso, isso contribui para a padronização e eficiência no desenvolvimento de software, permitindo que os desenvolvedores se concentrem mais na lógica específica da aplicação em vez de lidar com questões genéricas de implementação. Eles impõem uma estrutura organizada e padrões de desenvolvimento, o que ajuda a manter o código consistente e fácil de manter. Isso inclui convenções sobre a organização de arquivos e a arquitetura da aplicação.

Frameworks abstraem a complexidade de tarefas comuns, como gerenciamento de rotas, manipulação de banco de dados, autenticação de usuários e comunicação com APIs, simplificando o trabalho no desenvolvimento da aplicação. A maioria dos frameworks populares possui uma comunidade ativa e extensa documentação, o que facilita a resolução de problemas, a obtenção de ajuda e a aprendizagem de novas funcionalidades. Existem frameworks focados em diferentes áreas de desenvolvimento, entre eles se destacam:

- Frameworks web: focados no desenvolvimento de aplicações web back-end e APIs, como Express (JavaScript), Django (Python), Ruby on Rails (Ruby).
- Frameworks mobile: destinados ao desenvolvimento de aplicativos mobile para diferentes plataformas, como React Native (JavaScript), Flutter (Dart), Xamarin (C#).
- Frameworks front-end: especializados na criação de interfaces web interativas e dinâmicas, como React, Angular e Vue.js (JavaScript).

2.7.1 Express

Express é um framework minimalista e flexível amplamente utilizado para desenvolver aplicações de web back-end e APIs em JavaScript, com base na plataforma Node.js. Como um

dos frameworks mais populares para Node.js, oferece uma série de funcionalidades essenciais sem a necessidade de uma estrutura rígida, permitindo o controle total sobre a arquitetura e o fluxo de suas aplicações.

O framework oferece um sistema de roteamento robusto que permite definir como as aplicações respondem às solicitações HTTP em diferentes URLs. É possível criar rotas para manipular os métodos HTTP GET, POST, PUT, DELETE e outros, organizando de forma eficiente a lógica de roteamento, o que o torna amplamente utilizado para a construção de APIs REST, pois permite a criação de endpoints¹ que podem ser consumidos por clientes front-end, aplicativos móveis ou outros serviços.

Uma das características-chave do Express é o seu design modular, que permite aos desenvolvedores usar somente os componentes que precisam e facilmente estender o framework com um middleware personalizado (PAVANELI, 2023). Os middleware podem executar tarefas como manipulação de solicitações, autenticação, manipulação de erros, logging e muito mais. Eles podem ser aplicados globalmente, em rotas específicas ou em grupos de rotas.

Apesar de suas muitas vantagens, o Express também tem suas limitações. Alguns desenvolvedores criticaram a falta de suporte embutido para determinados recursos, como engines de template e conectividade com banco de dados. No entanto, essas limitações podem ser facilmente superadas usando módulos ou plugins de terceiros (PAVANELI, 2023).

O Express integra-se facilmente com várias bibliotecas e frameworks para adicionar funcionalidades como autenticação, banco de dados, validação de dados, segurança e muito mais. Isso permite estender as capacidades do Express conforme necessário para atender às necessidades específicas da aplicação.

2.7.2 React

O React é uma biblioteca JavaScript de código aberto criada e mantida pelo Facebook, utilizada para a construção de interfaces web interativas e reutilizáveis. Ao invés de ser um framework completo, o React foca na camada de visualização (UI) da aplicação, possibilitando a criação de componentes reutilizáveis que descrevem como a interface deve se comportar e renderizar.

A biblioteca permite a construção de interfaces de usuário utilizando componentes, que são blocos de construção reutilizáveis. Cada componente encapsula a sua própria lógica e renderização, tornando o desenvolvimento modular e facilitando a manutenção e a reutilização do código. Os componentes podem ser compostos para formar interfaces complexas. O React utiliza JSX, uma extensão de sintaxe para JavaScript que permite escrever HTML dentro do

¹ Enquanto "rota" se refere ao URL que o cliente requisita, "endpoint" é uma combinação da rota com o método HTTP (GET, POST, etc.) associado a uma ação específica no servidor.

código da linguagem, o que torna o código mais legível e expressivo, facilitando a criação de interfaces de usuário complexas de forma declarativa.

Como explica (SILVA, 2021), uma vez que a linguagem JavaScript é complexa, qualquer alteração pode causar problemas em toda a estrutura da árvore do DOM, podendo causar desde pequenas instabilidades até travamentos da página, o que afeta negativamente a performance da aplicação. Ou seja, pequenas alterações na página podem causar grandes problemas na interface.

Para solucionar essa questão, React foi projetado para trabalhar com o DOM Virtual, que é uma abstração da árvore do DOM. Todas as atualizações e modificações na interface são aplicadas e processadas no DOM Virtual e, depois de tudo pronto no DOM Virtual, a alteração é aplicada na árvore do DOM, sem interferir com o restante da árvore do documento (SILVA, 2021).

Por fim, a biblioteca também faz uso de Hooks, que permitem o uso de estado e outras funcionalidades do React em forma de componentes funcionais, simplificando o código e tornando-o mais legível. Hooks como useState e useEffect são amplamente utilizados para gerenciar estado e efeitos colaterais em componentes funcionais.

2.7.3 Next.js

Next.js é um framework web de código aberto, criado pela Vercel, que amplia as funcionalidades do React, fornecendo recursos e ferramentas prontas para uso que agilizam o desenvolvimento de aplicações web modernas e otimizadas. O Next.js suporta múltiplos métodos de renderização, incluindo a renderização no lado do servidor, a geração estática e a renderização no lado do cliente. Isso permite escolher a estratégia que melhor se adapta às necessidades da aplicação.

Um dos principais benefícios do Next.js é o *Server-Side Rendering*. Enquanto o React é uma SPA (*Single Page Application*), onde todo o conteúdo é gerado no navegador do usuário, o SSR permite que o conteúdo seja renderizado no servidor e entregue pronto ao usuário. Isso é especialmente útil para projetos React de áreas públicas não logadas, pois facilita a indexação por mecanismos de busca, melhorando a visibilidade da aplicação na web (CLEMENTE, 2023).

Além disso, o framework oferece um sistema de roteamento baseado em arquivos, onde qualquer arquivo adicionado ao diretório "pages" se torna automaticamente uma rota disponível na aplicação. Isso elimina a necessidade de configurar rotas manualmente, simplificando o desenvolvimento.

Por padrão, os componentes dentro desse diretório são definidos como Server Components, o que nos dá a capacidade de criar aplicativos que abrangem tanto o servidor quanto o cliente, combinando a interatividade avançada dos

aplicativos do lado do cliente com o desempenho aprimorado da renderização de servidor tradicional ([CLEMENTE, 2023](#)).

Além de servir páginas React, o Next.js permite a criação de endpoints API diretamente dentro da aplicação. Isso permite a criação de um back-end leve sem a necessidade de configurar um servidor separado. É possível criar aplicações em JavaScript ou TypeScript usando a integração nativa com o Node.js para criar tanto o front-end quanto o back-end.

2.8 OAuth

OAuth (*Open Authorization*) é um protocolo de autorização aberto amplamente utilizado para permitir que aplicativos acessem recursos de outros sites de forma segura, sem a necessidade de compartilhar senhas, por exemplo. Ele foi projetado para oferecer uma experiência segura e conveniente, permitindo que os usuários concedam permissões específicas para aplicativos e serviços sem comprometer suas credenciais.

O OAuth não compartilha dados de senha, mas usa tokens de autorização para provar uma identidade entre consumidores e provedores de serviços. O OAuth é um protocolo de autenticação que permite que você aprove um aplicativo interagindo com outro em seu nome [...] ([SOBERS, 2022](#)).

Por exemplo, ao conectar um aplicativo a uma conta do Google, o OAuth permite que a aplicação acesse informações do usuário, como email, nome e foto do perfil, sem jamais compartilhar dados sensíveis.

A evolução do OAuth do 1.0 para o 2.0 trouxe avanços significativos em termos de simplicidade e aplicabilidade. A primeira versão, embora segura, apresentava requisitos criptográficos complexos e era mais difícil de implementar. A versão 2.0 simplificou o processo, eliminando a necessidade de criptografia explícita e confiando em HTTPS para proteger os dados em trânsito.

Além disso, o OAuth 2.0 foi projetado para ser expansível, com suporte para uma variedade maior de cenários, incluindo dispositivos móveis, aplicações web, e até mesmo dispositivos conectados. Essa evolução o consolidou como o padrão de autorização mais utilizado na atualidade. Sua flexibilidade e ampla adoção o tornaram indispensável no ecossistema de desenvolvimento de software, especialmente em contextos onde a segurança e a facilidade de uso são cruciais.

3 Metodologia

O processo metodológico está conectado aos objetivos declarados na [subseção 1.0.2](#) deste trabalho e inicia-se com a análise do funcionamento do RU e a identificação das necessidades de informação dos usuários, como foi explicado em tópicos anteriores. Além disso, é necessário possuir conhecimento técnico para propor um artefato tecnológico que possa solucionar esse problema.

Um dos passos iniciais consistiu em realizar um levantamento das funcionalidades do sistema em conjunto com a gestora do RU, por meio de uma entrevista. Essa etapa permitiu obter a visão de uma profissional da área em relação ao sistema atual e suas expectativas em relação a uma nova aplicação para o restaurante. Além disso, foram consideradas as deficiências identificadas pelos clientes do restaurante, incluindo os estudantes e demais frequentadores do campus.

Com base nas ideias levantadas, a arquitetura do sistema foi definida conforme o diagrama apresentado na [Figura 1](#). A aplicação será dividida em três camadas principais: front-end, back-end e banco de dados.

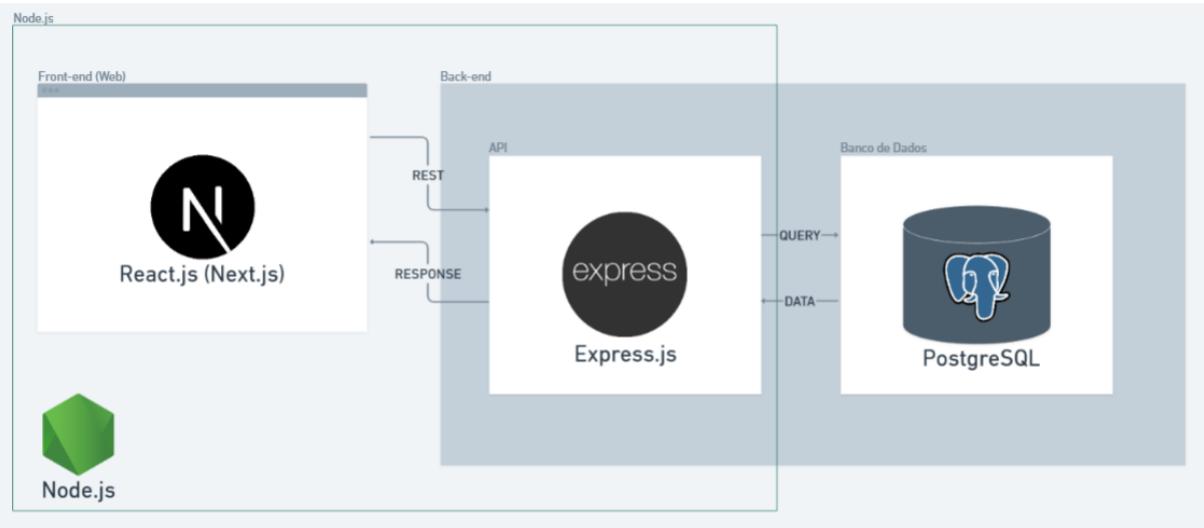


Figura 1 – Diagrama de Arquitetura Inicial do Projeto

O desenvolvimento do front-end será feito utilizando o React juntamente com o framework Next.js. Essa combinação será utilizada para criar uma aplicação web que permitirá aos estudantes acessar e avaliar o cardápio. Além disso, proporcionará aos responsáveis pelo restaurante uma interface intuitiva para a publicação do cardápio semanal, bem como acesso às avaliações e a um portal de comunicação. Esse portal permitirá a comunicação de notícias e informações pertinentes aos usuários do RU.

O back-end será desenvolvido utilizando o framework Express.js para Node.js, que fornecerá uma API REST. Essa API não só se conectará ao sistema web que será desenvolvido neste projeto, mas também permitirá uma futura compatibilidade com um front-end mobile, que poderá ser desenvolvido em React Native, por exemplo. A API será responsável por:

- Gerenciar a autenticação e autorização dos usuários;
- Receber e processar as requisições de consulta ao cardápio;
- Coletar e armazenar as avaliações dos pratos feitas pelos usuários.

Na camada de persistência, o sistema utilizará o banco de dados PostgreSQL para armazenar todas as informações, como dados dos usuários, cardápios diários e avaliações dos pratos. A integração com o banco de dados será feita por meio de consultas SQL executadas pela API. Na Figura 2, é possível observar uma visão inicial do diagrama entidade-relacionamento do banco de dados.

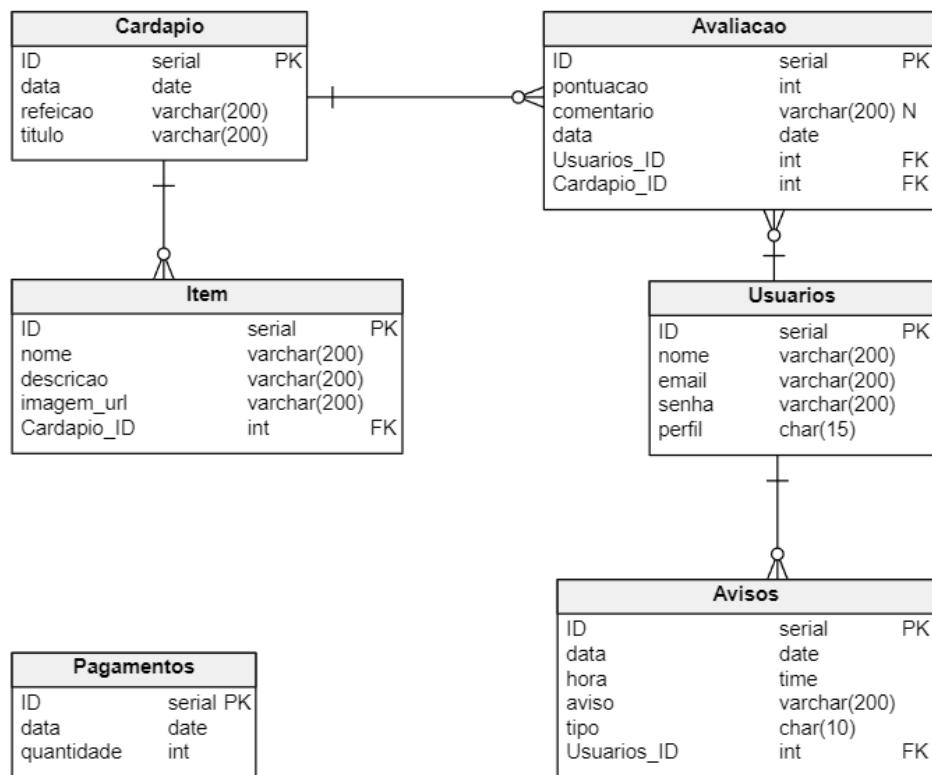


Figura 2 – Diagrama Entidade-Relacionamento Inicial do Banco de Dados

Após a conclusão dos testes e a validação do sistema, será avaliada a viabilidade de sua implementação nos servidores da universidade. Serão fornecidas instruções e demonstrações aos responsáveis pelo restaurante, a fim de garantir que possam atualizar o cardápio diariamente. Os

técnicos administrativos da universidade que lidam com as questões do restaurante terão acesso às avaliações concedidas pelos usuários, para poderem atuar diretamente com essas informações. Além disso, deseja-se que o sistema atenda às expectativas e, acima de tudo, seja intuitivo.

3.1 Desenvolvimento do protótipo

O desenvolvimento do protótipo foi estruturado em três grandes etapas: banco de dados, API e front-end. Embora essas etapas tenham sido iniciadas de forma independente, o avanço do projeto exigiu aprimoramentos conjuntos, uma vez que os desafios e ajustes surgiram conforme as regras de negócio e os objetivos da aplicação foram sendo alinhados e consolidados.

3.1.1 Back-end

No back-end, duas principais estruturas suportam o funcionamento do sistema: o banco de dados e a API, como ilustrado na [Figura 1](#). O banco de dados é responsável pelo armazenamento de todas as informações e pela aplicação das regras de negócio que garantem a integridade dos dados e o cumprimento dos requisitos do sistema. A API, por sua vez, atua como uma camada intermediária entre o banco de dados e a interface do usuário, promovendo uma comunicação eficiente e segura entre as partes, facilitando tanto o acesso aos dados quanto a implementação de operações complexas que exigem manipulação controlada das informações.

Essa arquitetura modular possibilita um sistema mais flexível, escalável e preparado para evoluções futuras, garantindo que tanto a camada de dados quanto a de interface possam ser aprimoradas conforme novas demandas forem surgindo.

3.1.1.1 Banco de Dados

O script descrito no Anexo [A.1](#) define a estrutura do banco de dados relacional projetado para este sistema, especificando a criação de tabelas, chaves primárias, chaves estrangeiras e restrições de unicidade, essenciais para a integridade dos dados.

O banco foi organizado em seis tabelas principais, entre as quais se destacam `Cardapio`, `Item` e `Avaliacao`. A tabela `Cardapio` é a estrutura central, onde se registram dados como a data, o tipo de refeição e o título do cardápio, formando a base do projeto. Já a tabela `Item` armazena informações detalhadas sobre cada prato disponível, incluindo nome, descrição e imagem representativa.

Para permitir que um cardápio possua múltiplos itens, cuja quantidade pode variar, foi necessária a introdução de uma tabela intermediária denominada `Cardapio_Item`. Essa tabela estabelece uma relação muitos-para-muitos entre `Cardapio` e `Item`, possibilitando que um único item seja vinculado a diferentes cardápios, promovendo a reutilização eficiente de itens frequentes em cardápios distintos.

A estrutura relacional estabelecida entre as tabelas `Item`, `Cardapio` e `Cardapio_Item`, ilustrada pela [Figura 3](#), facilita a consistência do sistema, proporcionando flexibilidade na criação de cardápios variados que incluem itens comuns. Essa abordagem se alinha à necessidade de um sistema adaptável, onde pratos regulares possam compor diferentes refeições sem a duplicação de dados.

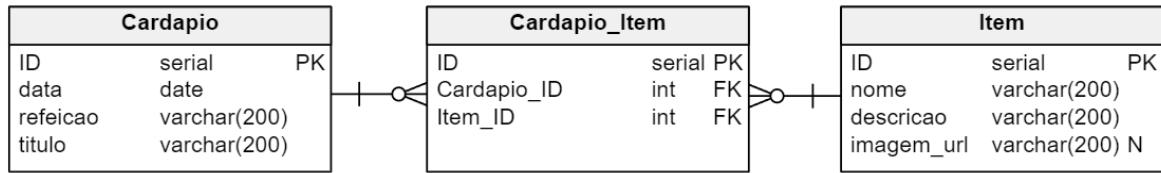


Figura 3 – Relacionamento entre as tabelas `Cardapio` e `Item`

A inclusão da tabela `Cardapio_Item` impactou o Diagrama Entidade-Relacionamento (DER) representado na [Figura 2](#). No entanto, o total de tabelas foi mantido em seis devido à exclusão da tabela `Pagamentos`.

A tabela `Pagamentos` foi descontinuada ao longo do desenvolvimento, pois não era indispensável para atender aos requisitos atuais e possuía pouca relevância no escopo definido. A adição de um fluxo de pagamentos, porém, é uma possibilidade futura e pode agregar valor ao sistema, especialmente para fins de gestão administrativa.

Como resultado dessas modificações, obteve-se o DER final, ilustrado na [Figura 4](#), refletindo uma estrutura de dados eficiente e alinhada às necessidades operacionais e de manutenção do Restaurante Universitário.

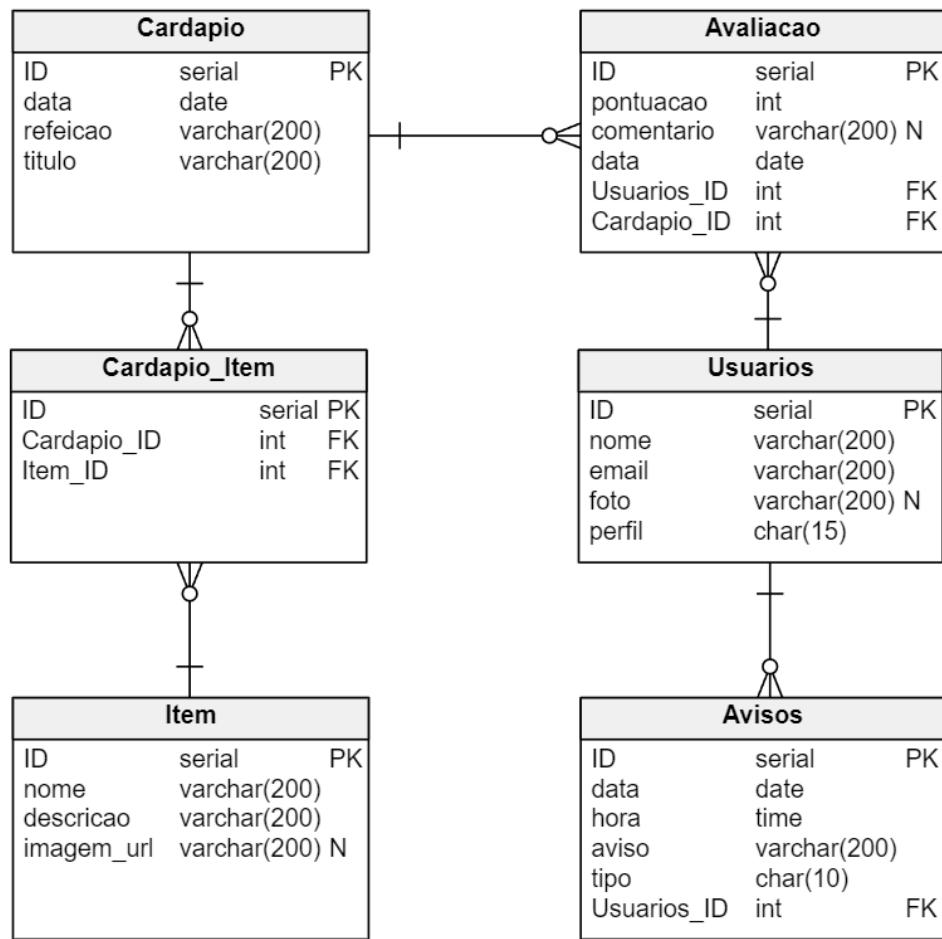


Figura 4 – Diagrama Entidade-Relacionamento Final do Banco de Dados

3.1.1.2 API

A API foi desenvolvida utilizando o framework Express¹ e integra-se diretamente com o banco de dados PostgreSQL para fornecer e gerenciar as informações do sistema. Toda a estruturação desta parte do projeto pode ser acessada por meio do repositório no anexo B.1.

Para cada tabela presente no banco representado pela Figura 4 foram criadas rotas na API que permitissem tanto a leitura de todos os dados de forma geral e individual, como também a criação de novos dados e a edição e a exclusão de cada um deles por índice. O protocolo utilizado neste contexto foi melhor explicado na subseção 2.4.1.

Embora algumas rotas não sejam acessíveis aos usuários no projeto final, sua criação e funcionamento foram essenciais para o desenvolvimento e podem ser úteis para melhorias e implementações futuras. Um exemplo é o caso dos *endpoints* que permitem a manipulação dos usuários registrados no sistema, atualmente restrito até para os administradores, mas que, dada a existência dessas rotas, podem ser melhor aproveitados em uma expansão do projeto.

¹ <https://expressjs.com/>

Entre os *endpoints* implementados, destaca-se o DELETE com validação no qual a exclusão de cardápios que já possuem avaliações foi impedida, com retornos específicos de erro. No [Código 3.1](#) está definida uma rota de exclusão para registros da tabela cardapio na qual, antes de proceder com a exclusão, é realizada uma verificação para garantir que o cardápio não tenha sido previamente avaliado, garantindo a integridade dos dados no sistema.

```

1   app.delete("/cardapio/:id", (req, res) => {
2     try {
3       const cardapioId = req.params.id;
4       console.log("Rota: delete/" + cardapioId);
5       client.query(
6         "SELECT COUNT(*) FROM avaliacao WHERE cardapio_id = $1",
7         [cardapioId],
8         (err, result) => {
9           if (err) {
10             return console.error("Erro ao verificar avaliacoes", err);
11           }
12           const avaliacaoCount = parseInt(result.rows[0].count, 10);
13           if (avaliacaoCount > 0) {
14             return res.status(400).json({ error: "Nao e possivel excluir
um cardapio que ja foi avaliado." });
15           }
16           client.query(
17             "DELETE FROM cardapio WHERE id = $1",
18             [cardapioId],
19             (err, result) => {
20               if (err) {
21                 return console.error("Erro ao executar a query de DELETE"
, err);
22               }
23               if (result.rowCount === 0) {
24                 res.status(404).json({ info: "Registro nao encontrado."
});
25               } else {
26                 res.status(200).json({ info: "Registro excluido." });
27               }
28             }
29           );
30         );
31       );
32     } catch (error) {
33       console.log(error);
34       res.status(500).json({ error: "Ocorreu um erro no servidor." });
35     }
36   });

```

Código 3.1 – *Endpoint* de exclusão de um cardápio

A primeira query executada, `SELECT COUNT(*) FROM avaliacao WHERE cardapio_id = $1`, tem como objetivo contar o número de registros na tabela `avaliacao` que estão associados ao `cardapio_id` especificado. Se o contador retornar um valor maior que zero, indica que o cardápio já possui avaliações e, por isso, a exclusão é impedida. Nesse caso, a resposta retornada ao cliente é uma mensagem de erro, informando justamente que a remoção de cardápios avaliados não é permitida.

Caso a contagem de avaliações seja zero, a segunda query é executada para proceder com a exclusão. A query `DELETE FROM cardapio WHERE id = $1` remove o registro da tabela `cardapio`.

Essas duas operações realizadas em sequência asseguram que somente cardápios sem avaliações associadas possam ser excluídos do banco de dados, promovendo a consistência dos dados e a integridade referencial no sistema de gerenciamento de cardápios.

Outro *endpoint* de destaque é o usado na listagem de itens em cada cardápio: A query apresentada no [Código 3.2](#) visa recuperar todos os itens associados a um cardápio específico no sistema. Essa consulta SQL utiliza uma combinação de seleção e junção de tabelas para identificar e retornar os itens que pertencem a um determinado cardápio. A estrutura inicial, `SELECT i.*`, indica que todas as colunas da tabela `Item` devem ser retornadas no resultado da consulta.

```

1 app.get("/cardapio/:id/itens", (req, res) => {
2   try {
3     console.log("Rota: cardapio/" + req.params.id + "/itens");
4     client.query(
5       "SELECT i.* FROM Item i INNER JOIN Cardapio_Item ci ON i.ID = ci.
6       Item_ID WHERE ci.Cardapio_ID = $1",
7       [req.params.id],
8       (err, result) => {
9         if (err) {
10           return res.status(500).json({ error: "Erro ao executar a qry
11             de SELECT itens por cardapio id" });
12         }
13         res.send(result.rows);
14       }
15     );
16   } catch (error) {
17     res.status(500).json({ error: "Ocorreu um erro no servidor." });
18   }
19 });

```

Código 3.2 – Endpoint para listar itens presentes em um determinado cardápio

A tabela `Item`, referenciada pelo alias `i`, é definida como a tabela principal, enquanto a operação `INNER JOIN` com a tabela `Cardapio_Item` permite a combinação de registros entre as duas tabelas onde houver correspondência. A tabela `Cardapio_Item` funciona como uma tabela intermediária que relaciona os identificadores de itens (`Item_ID`) aos identificadores de

cardápios (Cardapio_ID), essencial para mapear quais itens pertencem a quais cardápios. O alias ci é utilizado para simplificar a referência a Cardapio_Item, otimizando a legibilidade da query.

A condição ON i.ID = ci.Item_ID define o critério de junção, especificando que a coluna ID da tabela Item deve corresponder à coluna Item_ID de Cardapio_Item, garantindo que apenas itens específicos associados ao cardápio sejam incluídos no resultado.

Por fim, o *endpoint* de criação de uma nova avaliação, [Código 3.3](#), permite que os usuários submetam suas pontuações e comentários para um cardápio específico. Para realizar esta ação, o sistema requer que o cliente envie os parâmetros pontuacao, usuarios_id e cardapio_id.

```

1 app.post('/avaliacoes', (req, res) => {
2   const { pontuacao, comentario, usuarios_id, cardapio_id } = req.body;
3
4   if (!pontuacao || !usuarios_id || !cardapio_id) {
5     return res.status(400).json({ error: 'Pontuacao, Usuarios_ID e
6     Cardapio_ID sao obrigatorios.' });
7   }
8
9   client.query(
10     'SELECT * FROM Avaliacao WHERE Usuarios_ID = $1 AND Cardapio_ID = $2',
11     [usuarios_id, cardapio_id],
12     (err, result) => {
13       if (err) {
14         return res.status(400).json({ error: 'Erro ao verificar
15         avaliacao existente.' });
16       }
17
18       if (result.rowCount > 0) {
19         return res.status(400).json({ error: 'Usuario ja avaliou este
20         cardapio.' });
21     }
22
23     client.query(
24       'INSERT INTO Avaliacao (pontuacao, comentario, data,
25       Usuarios_ID, Cardapio_ID) VALUES ($1, $2, CURRENT_DATE, $3, $4)
26       RETURNING *',
27       [pontuacao, comentario, usuarios_id, cardapio_id],
28       (err, result) => {
29         if (err) {
30           return res.status(500).json({ error: 'Erro ao criar
31           avaliacao.' });
32         }
33         res.status(201).json(result.rows[0]);
34       }
35     );
36   );
37 }
```

```

30     }
31   );
32 });

```

Código 3.3 – *Endpoint* de avaliações

Em seguida, o sistema verifica se já existe uma avaliação feita pelo mesmo usuário para o mesmo cardápio. Caso um resultado seja encontrado, o sistema retorna uma mensagem de erro, indicando que o usuário já avaliou o cardápio, impedindo duplicação de avaliações para o mesmo par de usuário e cardápio. Caso a verificação indique que o usuário ainda não avaliou o cardápio, a aplicação prossegue para a inserção da nova avaliação.

Dessa forma, o *endpoint* implementa uma verificação de unicidade das avaliações por usuário e cardápio, assegurando a consistência dos dados e promovendo um registro controlado e único de avaliações.

Vale citar também, o retorno de status adicionado em cada um dos *endpoints*, que retornam mensagens de sucesso ou erro para cada uma das requisições feitas ao banco de dados. Estas mensagens também foram usadas diretamente no front-end, através de um componente do projeto, para que os usuários fossem notificados quando suas ações foram aceitas ou recusadas.

3.1.2 Autenticação

A autenticação dos usuários foi implementada com a biblioteca NextAuth, que utiliza OAuth 2.0 para realizar o login por meio de contas Google. Esse método elimina a necessidade de armazenamento de senhas, conforme explicado na [seção 2.8](#), o que trouxe uma mudança sutil a tabela apesentada na [Figura 4](#), e facilita o gerenciamento de permissões no banco de dados.

A configuração do NextAuth inclui dois principais *callbacks*: `session` e `signIn`. O `session` adiciona o ID do usuário à sessão, caso ele já exista no banco de dados, enquanto o `signIn` verifica a existência do usuário e, se ele não estiver registrado, cria um novo registro na tabela `usuarios`. Essas operações de leitura e escrita são realizadas com consultas SQL parametrizadas, que previnem a inserção de dados duplicados.

```

1 import NextAuth from "next-auth";
2 import GoogleProvider from "next-auth/providers/google";
3 import { connectToDB, pool } from "@/utils/database";
4
5 const handler = NextAuth({
6   providers: [
7     GoogleProvider({
8       clientId: process.env.GOOGLE_ID,
9       clientSecret: process.env.GOOGLE_CLIENT_SECRET,
10      }),
11    ],
12   callbacks: {

```

```
13  async session({ session }) {
14    try {
15      await connectToDB();
16      const query = 'SELECT * FROM usuarios WHERE email = $1;';
17      const values = [session.user.email];
18      const result = await pool.query(query, values);
19
20      if (result.rows.length > 0) {
21        const user = result.rows[0];
22        session.user.id = user.id;
23      }
24
25      return session;
26    } catch (error) {
27      console.error("Erro ao recuperar a sessao:", error);
28      return session;
29    }
30  },
31
32  async signIn({ profile }) {
33    try {
34      await connectToDB();
35      const query = 'SELECT * FROM usuarios WHERE email = $1;';
36      const values = [profile.email];
37      const result = await pool.query(query, values);
38
39      if (result.rows.length > 0) {
40        return true;
41      }
42
43      const insertQuery = `
44        INSERT INTO usuarios (nome, email, foto, perfil)
45        VALUES ($1, $2, $3, 'user') RETURNING *;
46      `;
47      const insertValues = [
48        profile.name,
49        profile.email,
50        profile.picture,
51      ];
52
53      const insertResult = await pool.query(insertQuery, insertValues);
54      console.log("Novo usuario criado:", insertResult.rows[0]);
55
56      return true;
57    } catch (error) {
58      console.error("Erro ao autenticar usuario:", error);
59      return false;
60    }
61  }
62
63  module.exports = { session, signIn };
64
```

```

60      }
61    },
62  },
63 });
64
65 export { handler as GET, handler as POST };

```

Código 3.4 – Configuração do NextAuth para autenticação

O sistema de autenticação implementado na arquitetura do projeto, conforme ilustrado na Figura 5, garante a segurança e escalabilidade do sistema, proporcionando um fluxo de login intuitivo para os usuários, facilitando a gestão de contas e permissões, e minimizando o risco de exposição de dados sensíveis.

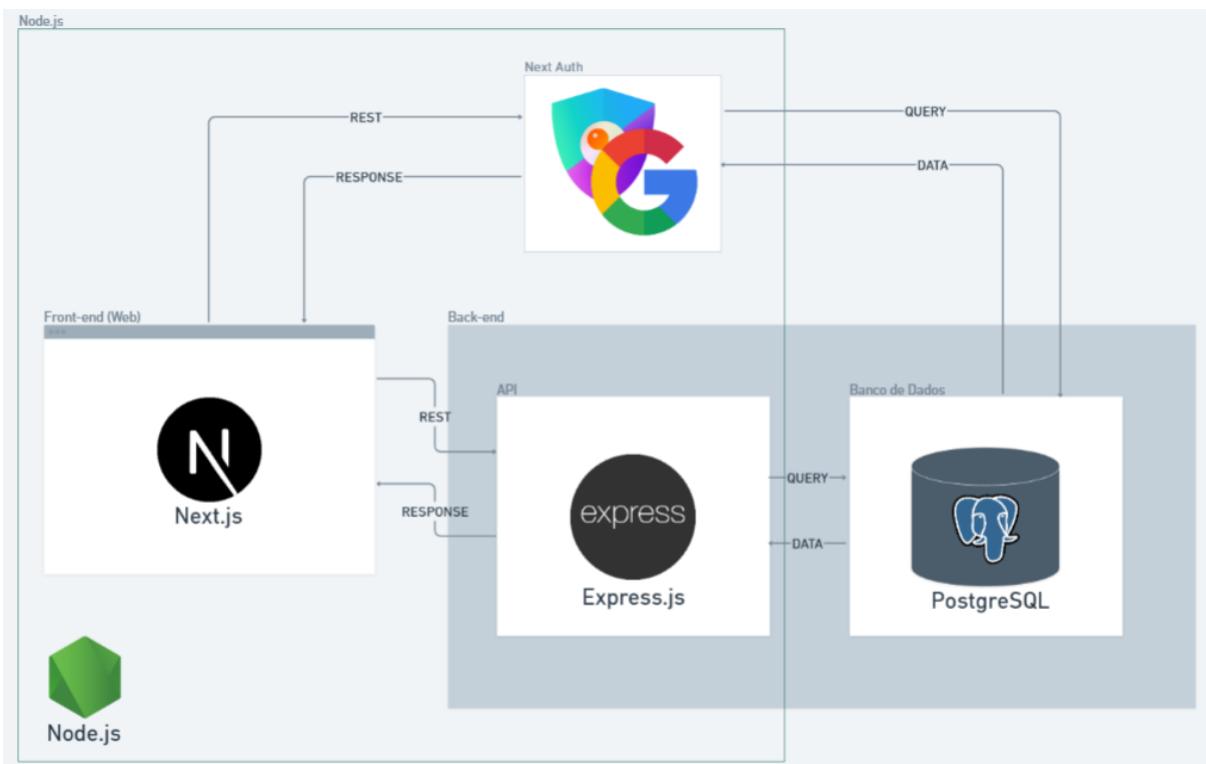


Figura 5 – Diagrama de Arquitetura do Projeto

3.1.3 Front-end

O front-end do projeto foi desenvolvido utilizando o framework Next.js, escolhido para construir uma interface dinâmica, interativa e responsiva, que proporciona uma navegação eficiente e intuitiva aos usuários do Restaurante Universitário. A organização modular das páginas e componentes permite flexibilidade na manutenção e no aprimoramento da aplicação, e a estrutura completa do repositório pode ser visualizada no Anexo B.2.

A página inicial da aplicação, ilustrada na Figura 6, exibe o cardápio do dia de forma centralizada e permite o compartilhamento em formato de imagem para *download*. Qualquer visitante pode conferir o cardápio e as avaliações já realizadas, mesmo não sendo um usuário que se cadastrou no sistema. Esse acesso público garante maior transparência na qualidade dos serviços oferecidos.



Figura 6 – Página principal com exibição do cardápio do dia

Usuários autenticados podem acessar o sistema de avaliações, registrando comentários e classificações para os pratos servidos, o que facilita o feedback contínuo para a gestão e permite ajustes no menu com base nas preferências registradas. As avaliações, por sua vez, são compostas por notas e comentários destes usuários, como mostrado na Figura 7.

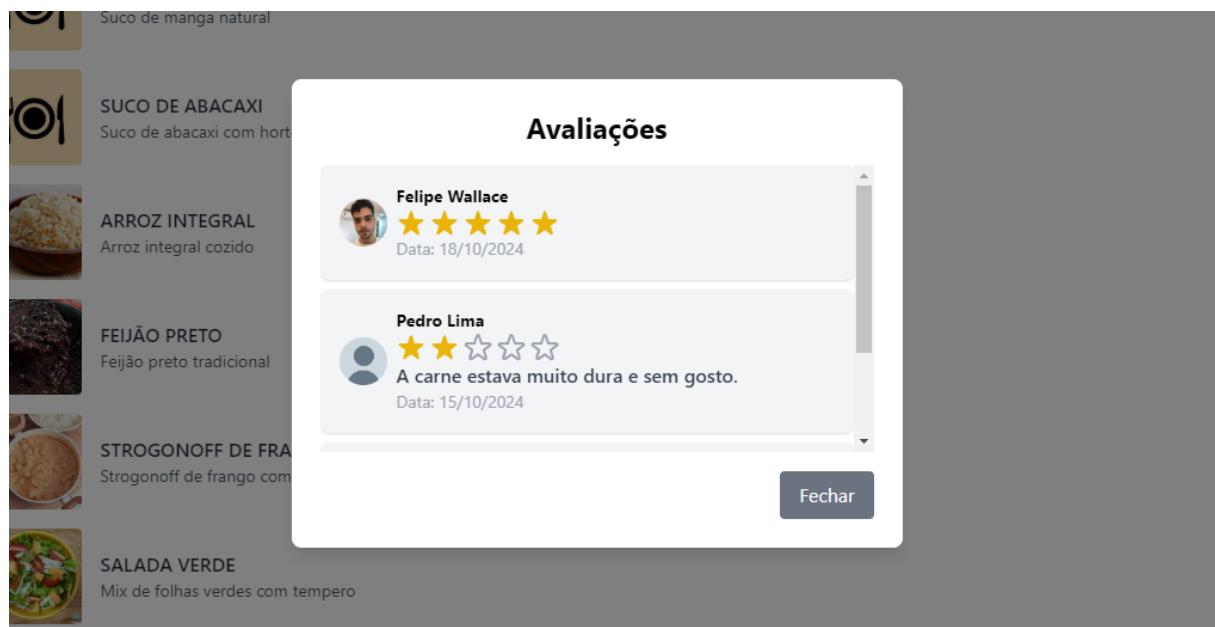


Figura 7 – Exibição detalhada das notas e comentários sobre os cardápios

O sistema de avaliações é viável graças à integração detalhada na subseção 3.1.2. Ao realizar o login na plataforma, os usuários comuns têm permissão apenas para essa ação. Para registrar notas e comentários, é necessário preencher os campos exibidos na Figura 8. Atualmente, no projeto, os tipos de comentários permitidos estão restritos a algumas opções, visando evitar manifestações que não atendam às expectativas ou que possam ser consideradas ofensivas.



Figura 8 – Campos de Avaliação

Além disso, foi configurado um cargo específico para administradores, atribuído manualmente através da API ou diretamente no banco de dados, que permite acesso a páginas com funcionalidades exclusivas, como criação e gerenciamento de cardápios. Para proteger essas páginas, foi criado o componente AdminGuard, que encapsula e verifica as permissões de acesso, garantindo segurança nas operações administrativas.

```

1 const AdminGuard = ({ children }) => {
2   const { data, status } = useSession();
3   const router = useRouter();
4   const [isUserAdmin, setIsUserAdmin] = useState(null);
5   const url = process.env.NEXT_PUBLIC_API_URL;
6
7   useEffect(() => {
8     const checkIfUserIsAdmin = async () => {
9       if (session && session.user) {
10         try {
11           const response = await fetch(`/${url}usuarios/${session.
12             user.id}`);
13           const data = await response.json();
14           const perfil = data.perfil.trim();
15           setIsUserAdmin(perfil === "admin");
16         } catch (err) {
17           console.log("Erro ao buscar usuário:", err);
18           setIsUserAdmin(false);
19         }
20       } else {
21         setIsUserAdmin(false);
22       }
23     };
24     checkIfUserIsAdmin();
25   }, [session]);
26
27   useEffect(() => {
28     if (session && !isUserAdmin) {
29       const timeout = setTimeout(() => {
30         router.push('/');
31       }, 5000);
32
33       return () => clearTimeout(timeout);
34     }
35   }, [isUserAdmin, session, router]);
36
37   if (status === 'loading') {
38     return <p>Carregando...</p>;
39   }
40   if (status === 'unauthenticated' || !session) {
41     return (

```

```

41   <div>
42     <p>Você precisa estar logado para acessar esta pagina.</p>
43     <div className="flex justify-center">
44       <button onClick={() => signIn()}>
45         Fazer login
46       </button>
47     </div>
48   </div>
49 );
50 }
51 if (isUserAdmin === null) {
52   return <p>Carregando...</p>;
53 }
54 if (!isUserAdmin) {
55   return (
56     <p>Você não tem permissão para acessar esta pagina.</p>
57   );
58 }
59 return <>{children}</>;
60 };
61 export default AdminGuard;

```

Código 3.5 – Código base do componente AdminGuard

Por padrão, as rotas restritas são acessíveis apenas a partir do menu administrativo para usuários autorizados. A seguir, na Figura 9, é possível visualizar o menu lateral da interface administrativa.



Figura 9 – Menu administrativo na lateral da interface

As páginas administrativas incluem a aba "Cardápio", que permite o gerenciamento das refeições disponibilizadas no restaurante. Essa página, ilustrada na Figura 10, oferece ferramentas para adicionar e remover itens dos cardápios de forma simplificada.

The screenshot shows a web-based application for managing restaurant menus. At the top, there's a button labeled "Novo Cardápio". Below it, a search bar allows filtering by date or name, with fields for "dd/mm/aaaa" and "Pesquise pelo cardápio...".

The main content area displays two menu entries:
27/11/2024 - Jantar - Lasanha
Média de Avaliações: 4.75

A button "Exibir Itens" is shown below this entry.

28/11/2024 - Almoço - Pizza
Média de Avaliações: 5.00

This entry lists four items with small icons:

- ARROZ BRANCO**
Arroz cozido tradicional
- FEIJOADA**
Feijoada completa com carnes e farofa
- PURÊ DE BATATA**
Purê de batata com manteiga
- SUCO DE LIMÃO**
Suco de limão gelado

At the bottom right of the menu list are four small icons: a green plus sign, a red minus sign, a blue edit icon, and a red trash icon.

Figura 10 – Página exclusiva para a manipulação de cardápios

Adicionalmente, foram criados componentes específicos para inclusão e exclusão de itens, como ilustrado nas Figuras 11 e 12. Essa abordagem melhora a usabilidade e eficiência ao modificar o conteúdo dos cardápios.

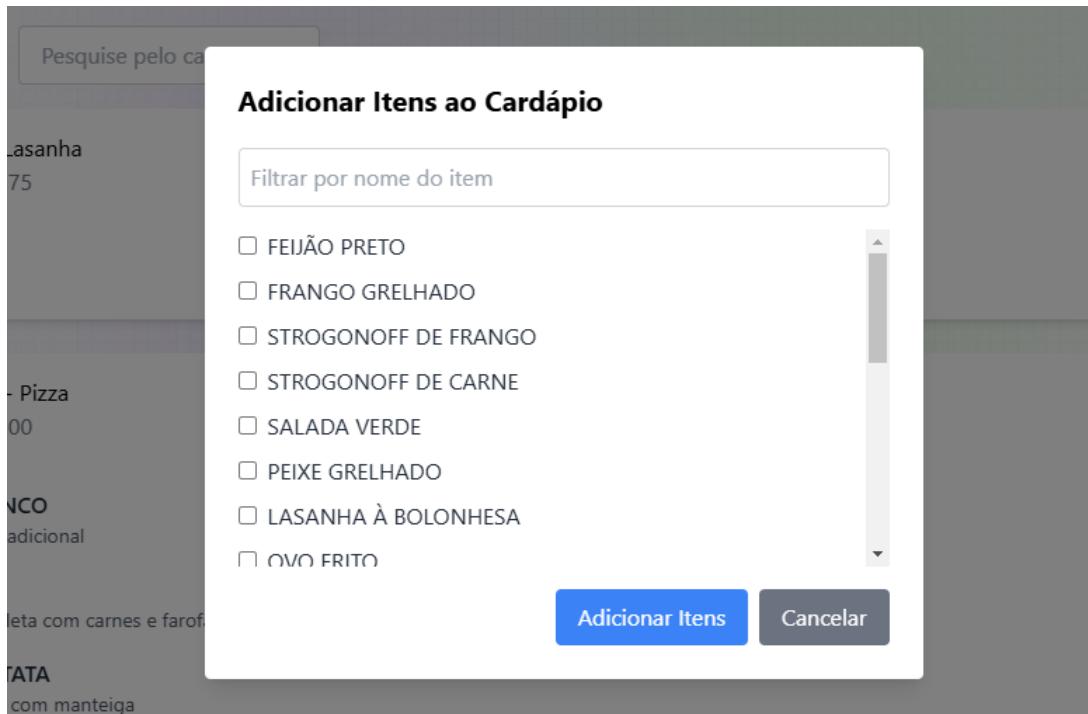


Figura 11 – Componente para adição de itens aos cardápios



Figura 12 – Componente para a remoção de itens do cardápio

Apesar de existirem páginas específicas para realizar esse tipo de associação, a criação desses componentes mostrou-se necessária como uma forma de melhorar a usabilidade do

sistema. Além disso, para aprimorar a usabilidade, foram adicionados filtros nas principais páginas administrativas, seja por nome ou datas, nos recursos que utilizam essas informações, a fim de facilitar e tornar a interação com o sistema mais dinâmica.

A página "Itens", Figura 13, permite a criação, edição e exclusão de itens do cardápio. A exclusão de um item só é possível se ele não estiver associado a nenhum cardápio. Ao excluir um cardápio, todos os itens relacionados são removidos, evitando registros órfãos no banco de dados. No entanto, os cardápios não podem ser excluídos após serem avaliados.

Nome do Item	Descrição	Opções
ARROZ BRANCO	Arroz cozido tradicional	
BATATA FRITA	Batata frita crocante	
FEIJOADA		

Figura 13 – Página de Gerenciamento de Itens

A página de "Avisos", ilustrada na Figura 14, será de grande importância, pois permitirá a publicação de conteúdo, informações e notícias relevantes aos usuários do RU. Foram criadas diferentes categorias de avisos, que variam na forma como a notificação é entregue aos usuários, desde notificações simples com cores mais suaves até alertas urgentes exibidos em pop-ups no site.

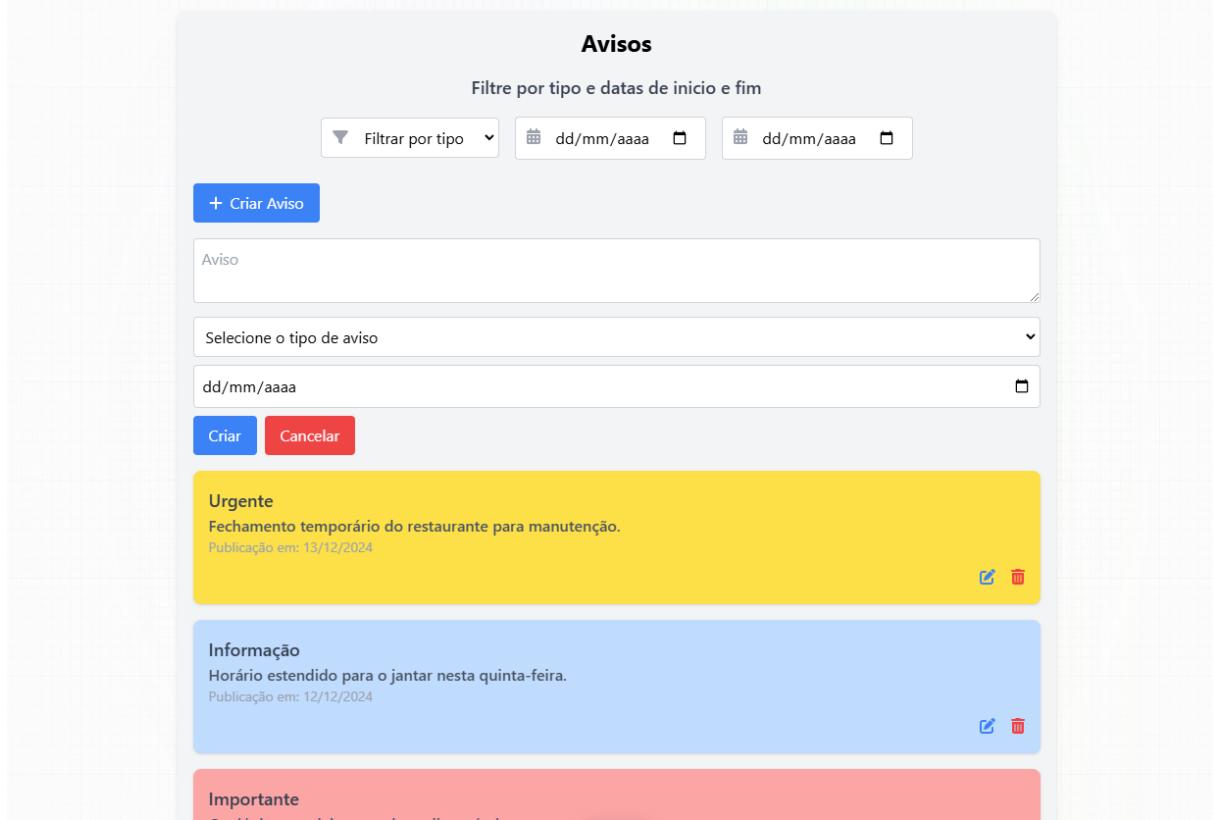


Figura 14 – Página de Configuração de Avisos

Os avisos são notificados apenas nas datas selecionadas durante a criação, podendo ser editadas posteriormente. Eles ficam listados por sete dias no ícone de sino na barra de navegação do site, permitindo que os usuários consultem informações que possam ter perdido dentro desse período.

Outras páginas importantes são "Usuários" e "Avaliações", que permitem a consulta de dados relevantes para a administração. Na página "Usuários", os gestores podem visualizar uma lista de usuários e acompanhar as médias de avaliação feitas por cada um. A página "Avaliações" exibe uma média geral das avaliações do restaurante, com filtros por data para análises detalhadas e temporais.

4 Discussão

Este sistema foi desenvolvido com uma arquitetura prática, utilizando um banco de dados relacional, uma API REST e um front-end moderno, projetados para atender às necessidades específicas do Restaurante Universitário. Durante a implementação, foram realizadas diversas etapas de planejamento e execução, mas, como em qualquer projeto de desenvolvimento, surgiram desafios e oportunidades de aprimoramento que merecem uma análise mais aprofundada.

A arquitetura adotada mostrou-se bastante adequada para um sistema que exige alta flexibilidade e capacidade de expansão. A estrutura modular das camadas de dados e de API permitiu isolar as responsabilidades, facilitando futuras modificações sem impactar drasticamente as demais partes do sistema. No entanto, embora a estrutura de banco de dados tenha sido funcional e eficiente, a ausência de funcionalidades avançadas, como um fluxo de pagamentos, foi uma limitação identificada durante o desenvolvimento, mas em concordância com as especificações levantadas. Em retrospectiva, a inclusão de uma camada de transações financeiras poderia ter enriquecido o sistema, proporcionando dados financeiros e de avaliação dos usuários em um único ambiente, útil para análises administrativas.

Outro ponto que merece reflexão foi o uso das tecnologias para autenticação como o OAuth com NextAuth para realização do login no sistema, o que trouxe vantagens em termos de segurança e praticidade, mas também gerou um aumento na complexidade inicial do projeto. A integração com provedores de login externos, como o Google, exigiu uma curva de aprendizado e cuidados adicionais com configurações de API e permissões. Embora essa abordagem tenha eliminado a necessidade de armazenamento de senhas, simplificando a gestão de contas, a integração não foi completamente intuitiva e exigiu múltiplas tentativas e erros. No entanto, o esforço foi recompensado com um sistema de login seguro e confiável, que se mostrou vantajoso para a experiência do usuário.

Em relação ao front-end, a utilização do Next.js contribuiu para uma interface responsiva e moderna. O framework permitiu o desenvolvimento de páginas dinâmicas, o que aprimorou a experiência de navegação dos usuários. No entanto, durante a implementação, surgiram desafios na adaptação do layout para todos os dispositivos. Em alguns casos, a interface precisou de ajustes manuais para garantir uma experiência visual consistente. Embora esses desafios tenham sido superados, a complexidade inicial da implementação da interface poderia ser reduzida com o uso de uma biblioteca de componentes pré-definidos, acelerando o desenvolvimento de um layout consistente e responsivo.

Além disso, foi possível perceber que a configuração do sistema para administrar e proteger as rotas dos administradores, com a criação de um componente denominado AdminGuard, atendeu a demanda de segurança de forma satisfatória. O desenvolvimento desse componente trouxe

flexibilidade no controle de acesso e na segurança das informações confidenciais. No entanto, o processo de definir e ajustar as permissões foi mais desafiador do que o previsto, o que pode indicar a necessidade de maior preparação para o gerenciamento de acessos em projetos futuros.

Em uma autoanálise, o sistema cumpriu os objetivos elencados no início do projeto, proporcionando uma aplicação capaz de gerenciar cardápios e coletar avaliações de forma eficiente. Contudo, há espaço para aprimoramentos. Entre os avanços, a construção de um sistema de estatísticas mais detalhado e a reinclusão de uma tabela para pagamentos seriam valiosas adições para aumentar a abrangência do sistema. Adicionalmente, a criação de filtros avançados na área de avaliações poderia fornecer informações mais detalhadas aos administradores, como a qualidade dos pratos em dias específicos ou a preferência dos estudantes.

Em suma, o desenvolvimento deste projeto foi satisfatório em termos de funcionalidade e desempenho. O sistema final representa uma plataforma capaz de atender às demandas do RU, e o aprendizado acumulado ao longo do desenvolvimento forneceu percepções valiosos para otimizar projetos futuros. Essa experiência reforça a importância de uma estrutura modular e de um planejamento detalhado das camadas de dados e acesso, pontos fundamentais para a escalabilidade e usabilidade de um sistema voltado ao atendimento de necessidades específicas.

5 Conclusão

Este trabalho desenvolveu uma aplicação web para a divulgação e integração entre usuários e o Restaurante Universitário da Universidade Federal de Itajubá - *Campus Itabira*, substituindo métodos tradicionais de divulgação por uma interface moderna, eficiente e interativa, procurando responder à indagação inicial do trabalho.

A implementação do sistema abrangeu desde a modelagem do banco de dados até a criação de uma API e uma interface de front-end, resultando em um protótipo funcional que permite atualizações diárias de cardápio e avaliação direta dos pratos oferecidos. A integração de uma interface intuitiva não apenas aprimorou a experiência dos usuários, mas também contribuiu para uma comunicação mais transparente entre os estudantes e a gestão do restaurante, facilitando o atendimento às necessidades dos frequentadores do RU.

Os resultados obtidos mostram um sistema funcional, que cumpre as demandas de comunicação e interação propostas e oferece uma estrutura flexível para adaptações futuras. Com uma base sólida, o projeto está preparado para expandir suas funcionalidades a partir de algumas propostas de continuidade, que podem aumentar ainda mais sua relevância no contexto universitário.

Como trabalhos futuros pode-se indicar o desenvolvimento de uma aplicação móvel que faça uso do back-end existente, ampliando o acesso e a usabilidade da aplicação por meio de um front-end específico aos aparelhos celulares. A reinclusão da tabela Pagamentos também representa um passo importante, permitindo o desenvolvimento de um sistema de pagamentos integrado ao cardápio. Essa funcionalidade abrirá possibilidades para um levantamento detalhado de estatísticas, relacionando as avaliações dos usuários ao faturamento do restaurante, facilitando a análise dos impactos de satisfação sobre o desempenho financeiro.

Outro ponto a ser aprimorado é a metodologia de avaliação, com a inclusão de critérios mais extensos e específicos, que abrangem desde a qualidade dos pratos até aspectos como higiene e limpeza do ambiente. Esse aprimoramento nas avaliações permitirá uma coleta de dados mais completa, fornecendo à administração informações valiosas para a melhoria contínua dos serviços oferecidos pelo restaurante universitário.

Assim, o projeto desenvolvido estabelece uma base para a evolução contínua do sistema, servindo de apoio tanto para a gestão quanto para a experiência dos usuários. A implementação das propostas apresentadas permitirá ao sistema acompanhar as necessidades da comunidade acadêmica de forma ainda mais precisa e eficiente, consolidando o restaurante universitário como uma estrutura essencial no apoio ao bem-estar dos estudantes.

Referências

- ARIOVALDO, T. Política de moradia estudantil: Experiências de moradoras nos alojamentos da universidade federal de viçosa. *COLÓQUIO INTERNACIONAL DE GESTIÓN UNIVERSITARIA*, 2016. 1
- BARBOSA, C. M. O. et al. Using the underlying discourse unveiling method to understand organizations of social volunteers. *Anais do V Simpósio sobre Fatores Humanos em Sistemas Computacionais*, 2002. 10
- BITTENCOURT, A. Uma comparação de performance entre arquitetura graphql e rest. 2021. 10
- BITTENCOURT, J. *Framework: o que é e pra que serve essa ferramenta?* 2021. Acesso em: 29 mai. 2024. Disponível em: <<https://www.alura.com.br/artigos/framework-o-que-e-pra-que-serve-essa-ferramenta>>. 14
- CARDOSO, C. et al. Restaurante universitário da ufpe: uma abordagem sistêmica. *Revista Gestão Universitária na América Latina*, 2018. Acesso em: 7 mai. 2024. Disponível em: <<https://periodicos.ufsc.br/index.php/gual/article/view/1983-4535.2018v11n3p211>>. 5
- CLEMENTE, P. *Introdução ao Next.js - Um Framework para Desenvolvedores React*. 2023. Acesso em: 3 jun. 2024. Disponível em: <<https://blog.rocketseat.com.br/introducao-ao-next-js>>. 16, 17
- DATE, C. J. *Introdução a Sistemas de Bancos de Dados*. 8. ed. [S.l.]: Elsevier, 2004. 6
- DIANA, M.; GEROSA, M. Nosql na web 2.0: Um estudo comparativo de bancos não-relacionais para armazenamento de dados na web 2.0. *IX Workshop de Teses e Dissertações em Banco de Dados*, 2010. 7
- ELMASRI, R.; NAVATHE, S. B. *Sistemas de Banco de Dados*. 6. ed. [S.l.]: Pearson, 2011. 6
- JUNIOR, E.; ROCHA, R.; MACIEL, R. Desenvolvimento de api rest com spring boot. *Revista Científica do UniRios*, 2021. 8
- OLIVEIRA, S. Bancos de dados não-relacionais: Um novo paradigma para armazenamento de dados em sistemas de ensino colaborativo. *Revista da Escola de Administração Pública do Amapá*, 2014. Acesso em: 8 mai. 2024. Disponível em: <<https://www2.unifap.br/oliveira/files/2016/02/35-124-1-PB.pdf>>. 7
- PAVANELI, T. Desenvolvimento de uma api e uma aplicação web para auxiliar na análise de dados providos por aplicação móvel. 2023. 15
- PEREIRA, C. *Aplicações web real-time com Node.js*. [S.l.]: Casa do Código, 2014. 13
- RABELO, M. *TypeScript: O guia definitivo*. 2018. Acesso em: 17 mai. 2024. Disponível em: <<https://oieduardorabelo.medium.com/typescript-o-guia-definitivo-1a63b04259cc>>. 12
- ROHR, A.; MASIERO, M.; NETO, F. Proposta de um sistema de gestão de custos para o restaurante universitário da universidade federal do rio grande do sul. *Encontro Nacional de Engenharia de Produção*, 2010. Acesso em: 7 mai. 2024. Disponível em: <https://abepro.org.br/biblioteca/enegep2010_TN_STP_115_753_15326.pdf>. 6

- SANTOS, M. Sistema de comunicação organizacional em rotinas empresariais. 2023. 11
- SILVA, G.; FERREIRA, J. Análise comparativa de desempenho de consultas entre um banco de dados relacional e um banco de dados não relacional. 2017. Acesso em: 8 mai. 2024. Disponível em: <<https://dspace.uniube.br:8443/handle/123456789/178>>. 6
- SILVA, M. *React Aprenda Praticando*. [S.l.]: Novatec, 2021. 16
- SIQUEIRA, B. Runb: Aplicativo para o restaurante universitário da universidade de brasília. 2020. 5
- SOBERS, R. *O que é o OAuth? Definição e como funciona*. 2022. Acesso em: 10 out. 2024. Disponível em: <<https://www.varonis.com/pt-br/blog/what-is-oauth>>. 17
- SOUZA, G. V.; FAVA, H. L.; CINTRA, R. F. Restaurante universitário no contexto da assistência estudantil: Análise da produção científica (2010-2021). *Revista Ibero-Americana de Estudos em Educação*, 2023. 1
- TOTO, C. Cardápio digital - uma aplicação para criar e validar o cardápio de uma mineradora. 2021. 5

Anexos

ANEXO A – Script de Criação do Banco de Dados

```

1 -- Table: Cardapio
2 CREATE TABLE Cardapio (
3     ID serial NOT NULL,
4     data date NOT NULL,
5     refeicao varchar(200) NOT NULL,
6     titulo varchar(200) NOT NULL,
7     CONSTRAINT Cardapio_pk PRIMARY KEY (ID)
8 );
9
10 -- Table: Cardapio_Item
11 CREATE TABLE Cardapio_Item (
12     ID serial NOT NULL,
13     Cardapio_ID int NOT NULL,
14     Item_ID int NOT NULL,
15     CONSTRAINT Cardapio_Item_pk PRIMARY KEY (ID),
16     CONSTRAINT Cardapio_Item_unique UNIQUE (Cardapio_ID , Item_ID)
17 );
18
19 -- Table: Item
20 CREATE TABLE Item (
21     ID serial NOT NULL,
22     nome varchar(200) NOT NULL CONSTRAINT nome_unico UNIQUE,
23     descricao varchar(200) NOT NULL,
24     imagem_url varchar(200) NULL,
25     CONSTRAINT Item_pk PRIMARY KEY (ID)
26 );
27
28 -- Table: Avaliacao
29 CREATE TABLE Avaliacao (
30     ID serial NOT NULL,
31     pontuacao int NOT NULL,
32     comentario varchar(200) NULL,
33     data date NOT NULL,
34     Usuarios_ID int NOT NULL,
35     Cardapio_ID int NOT NULL,
36     CONSTRAINT Avaliacao_pk PRIMARY KEY (ID),
37     CONSTRAINT Avaliacao_Cardapio UNIQUE (Cardapio_ID , Usuarios_ID)
38 );
39
40 -- Table: Avisos

```

```
41 CREATE TABLE Avisos (
42     ID serial NOT NULL,
43     data date NOT NULL,
44     aviso varchar(200) NOT NULL,
45     tipo char(10) NOT NULL,
46     Usuarios_ID int NOT NULL,
47     CONSTRAINT Avisos_pk PRIMARY KEY (ID)
48 );
49
50 -- Table: Usuarios
51 CREATE TABLE Usuarios (
52     ID serial NOT NULL,
53     nome varchar(200) NOT NULL,
54     email varchar(200) NOT NULL,
55     foto varchar(200) NULL,
56     perfil char(15) NOT NULL,
57     CONSTRAINT Usuarios_pk PRIMARY KEY (ID),
58     CONSTRAINT email_unico UNIQUE (email)
59 );
60
61 -- Foreign keys
62 -- Reference: Avaliacao_Cardapio (table: Avaliacao)
63 ALTER TABLE Avaliacao ADD CONSTRAINT Cardapio_Avaliacao
64     FOREIGN KEY (Cardapio_ID)
65     REFERENCES Cardapio (ID)
66     NOT DEFERRABLE
67     INITIALLY IMMEDIATE;
68
69 -- Reference: Avaliacao_Usuarios (table: Avaliacao)
70 ALTER TABLE Avaliacao ADD CONSTRAINT Avaliacao_Usuarios
71     FOREIGN KEY (Usuarios_ID)
72     REFERENCES Usuarios (ID)
73     NOT DEFERRABLE
74     INITIALLY IMMEDIATE;
75
76 -- Reference: Avisos_Usuarios (table: Avisos)
77 ALTER TABLE Avisos ADD CONSTRAINT Avisos_Usuarios
78     FOREIGN KEY (Usuarios_ID)
79     REFERENCES Usuarios (ID)
80     NOT DEFERRABLE
81     INITIALLY IMMEDIATE;
82
83 -- Reference: Cardapio_Item_Item (table: Cardapio_Item)
84 ALTER TABLE Cardapio_Item ADD CONSTRAINT Cardapio_Item_Item
85     FOREIGN KEY (Item_ID)
86     REFERENCES Item (ID)
87     NOT DEFERRABLE
```

```
88    INITIALLY IMMEDIATE;  
89  
90 -- Reference: Item_Cardapio (table: Cardapio_Item)  
91 ALTER TABLE Cardapio_Item ADD CONSTRAINT Item_Cardapio  
92     FOREIGN KEY (Cardapio_ID)  
93     REFERENCES Cardapio (ID)  
94     NOT DEFERRABLE  
95    INITIALLY IMMEDIATE;
```

Código A.1 – Script de criação do banco de dados

ANEXO B – Repositórios do Projeto

B.1 API

O código-fonte da API do projeto pode ser acessado no GitHub através do link: <<https://github.com/FelipeWallace/RU-API>>

B.2 Front-end

O código-fonte do front-end do projeto pode ser acessado no GitHub através do link: <https://github.com/FelipeWallace/cardapio_ru>