RI-IFBA: Sistema Digital para o Refeitório Institucional do IFBA - Campus Salvador

Trabalho de Conclusão de Curso

Emmanuel Vitor Pereira de Jesus

Flávia Maristela Santos Nascimento

Orientadora

Instituto Federal da Bahia – IFBA
Curso de Análise e Desenvolvimento de Sistemas
Campus Salvador
Salvador, Bahia, Brasil

Janeiro 2026

Lista de abreviaturas e siglas

CONSUP
Conselho Superior

DPAAE
Diretoria de Políticas Afirmativas e Assuntos Estudantis

IFBA
Instituto Federal de Educação, Ciência e Tecnologia da Bahia

PNAES
Política Nacional de Assistência Estudantil

PNAE
Programa Nacional de Alimentação Escolar

PAAE
Programa de Assistência e Apoio ao Estudante

ORM
Object-Relational Mapping (ou Mapeamento Objeto-Relacional)

MVC
Model – View – Controller

Sumário

1 Visão Geral

As políticas públicas (PP) estabelecem um conjunto de programas, ações e decisões adotadas pelo Estado, tendo como objetivo o enfrentamento de problemas coletivos da sociedade e a promoção do bem-estar social. Embora a Constituição Federal de 1988 não apresente uma definição explícita de políticas públicas, seus fundamentos permitem delineá-las. O Artigo 3º define como objetivos da República a construção de uma sociedade livre, justa e solidária, bem como a redução das desigualdades sociais (BRASIL, 1988, s/p). Enquanto o Artigo 6º estabelece os direitos sociais, os princípios fundamentais e as finalidades que norteiam a ação do Estado para garantir as condições essenciais à dignidade humana a todo brasileiro em situação de vulnerabilidade social.[1] Assim, as políticas públicas legitimam-se nesses princípios constitucionais, orientadas por seu caráter estratégico para garantir a efetivação dos direitos fundamentais e assegurar o cumprimento das funções essenciais do Estado (SOUSA; SOARES, 2024).

No cenário das redes de ensino superior, a Política Nacional de Assistência Estudantil (PNAES), regulamentada pelo Decreto Federal nº 7.234, de 19 de julho de 2010,[2] configura-se como uma política pública voltada à ampliação das condições de permanência dos estudantes regularmente matriculados em cursos de graduação presenciais nas instituições federais de ensino superior (BRASIL, 2010, s/p). O decreto estabelece objetivos e diretrizes que orientam o desenvolvimento de ações de assistência em áreas estratégicas, como moradia estudantil, alimentação, transporte, atenção à saúde e inclusão social. A PNAES prioriza o atendimento a estudantes oriundos da rede pública de educação básica ou àqueles com renda familiar per capita de até um salário mínimo e meio.

No Instituto Federal da Bahia (IFBA), a Diretoria de Políticas Afirmativas e Assuntos Estudantis (DPAAE) atua diretamente na formulação, implementação e consolidação das Política de Assistência Estudantil, instituída pela Resolução CONSUP nº 25, de 23 de maio de 2016. Essa política organiza um conjunto de princípios, diretrizes voltados a assegurar o acesso, a permanência e a conclusão dos cursos contemplando especialmente aqueles em situação de vulnerabilidade socioeconômica. Essa política tem como eixo central o Programa de Assistência e Apoio ao Estudante (PAAE)[3], composto por diversas

[1] Art. 6º São direitos sociais a educação, a saúde, a alimentação, o trabalho, a moradia, o transporte, o lazer, a segurança, a previdência social, a proteção à maternidade e à infância, a assistência aos desamparados, na forma desta Constituição. (Redação dada pela Emenda Constitucional nº 90, de 2015). Parágrafo único. Todo brasileiro em situação de vulnerabilidade social terá direito a uma renda básica familiar, garantida pelo poder público em programa permanente de transferência de renda, cujas normas e requisitos de acesso serão determinados em lei, observada a legislação fiscal e orçamentária (Incluído pela Emenda Constitucional nº 114, de 2021) (Vide Lei nº 14.601, de 2023).

[2] DECRETO Nº 7.234, DE 19 DE JULHO DE 2010.

[3] Conforme Art. 2º da Resolução nº 25/2016: "A Política de Assistência Estudantil do IFBA está dividida em três eixos: I – Programa de Assistência e Apoio ao Estudante (PAAE): destina-se a estudantes em comprovada situação de vulnerabilidade social, tendo como obrigatória a participação em processo de seleção socioeconômica" (IFBA, 2016, p. 2).

modalidades de auxílios.[4]

É importante ressaltar que este auxílio é regulamentado de forma detalhada na Subseção V – Auxílio Alimentação, constante nos Artigos 40 a 45 da referida resolução.

1.1 Declaração do Problema

O Instituto Federal da Bahia (IFBA), Campus Salvador, dispõe em suas dependências de um Refeitório Institucional (RI) destinado à oferta de refeições (almoço e jantar) aos estudantes beneficiários do Programa de Assistência e Apoio ao Estudante (PAAE) e à comunidade acadêmica em geral. Segundo Sousa e Soares (2024), os RUs constituem um espaço de convivência e integração social, além de desempenhar papel central na garantia da segurança alimentar e nutricional dos discentes.

Atualmente, o cardápio é elaborado e executado por uma empresa terceirizada, seguindo supervisão e aprovação da equipe de Nutrição do campus. Diariamente, são oferecidos prato principal, guarnições, duas opções de proteína, sobremesa e bebida, conforme cardápio semanal disponibilizado em formato impresso no mural do RI.

Apesar da importância estratégica do RI para a permanência e o desempenho acadêmico dos estudantes, a gestão do atendimento aos bolsistas apresenta fragilidades operacionais que dialogam diretamente com a Subseção V da Resolução 25/2016, especialmente com os Artigos 40 a 45 (IFBA, 2016, art. 40–45). Segundo a normativa, o uso do auxílio deve ser registrado pelo próprio estudante, em folha de controle ou sistema informatizado (Art. 42, § 1º); as listas de frequência devem ser encaminhadas semanalmente à Gestão de Assistência Estudantil (Art. 42, § 2º); faltas consecutivas sem justificativa ensejam convocação e possível cancelamento do benefício (Art. 43); e refeições não utilizadas devem ser ofertadas a outros estudantes (Art. 44).

Entretanto, o controle atual é manual: o estudante apresenta documento e o funcionário responsável registra a presença em uma planilha eletrônica. Tal procedimento aumenta o risco de erros, perda de dados e retrabalho, contrariando a recomendação do uso de sistemas informatizados prevista na normativa.

O processo de justificativa de faltas também é fragmentado. Atualmente, o bolsista acessa um formulário eletrônico para comunicar ausências. Faltas posteriores exigem apresentação de atestado médico, e três faltas injustificadas no mês podem acarretar perda do benefício. No entanto, o controle e a validação dessas justificativas ocorrem sem integração com o registro de presença, onde posteriormente esses dados são cruzados e checados manualmente.

Da mesma forma, o gerenciamento das vagas excedentes (“extras”) depende de um segundo formulário do Google, no qual os interessados registram nome e horário. A lista

[4] O PAAE é composto pelos seguintes auxílios: Auxílio Transporte, Auxílio Moradia, Auxílio para Aquisições, Auxílio Cópia e Impressão e Auxílio Alimentação; e pelas bolsas: Bolsa Estudo e Bolsa vinculada a Projetos de Incentivo à Aprendizagem – PINA (IFBA, 2016, p. 3, art. 4º).

é organizada por ordem de preenchimento e os candidatos são chamados presencialmente nos minutos finais do turno, conforme disponibilidade. A ausência de integração entre os processos — presença, justificativas, vagas extras e cardápio — gera morosidade, inconsistências e limita o acesso da gestão a dados sistematizados, contrastando com os princípios de eficiência e inovação previstos na Política de Assistência Estudantil (IFBA, 2016), com os requisitos operacionais normativos da Subseção V e com a própria vocação tecnológica do IFBA.

1.2 Proposta de Solução de Software

A proposta é o desenvolvimento de um sistema web responsivo, acessível em múltiplos dispositivos, que centralize as operações do RI em uma única plataforma. A solução é estruturada com diferentes níveis e perfis de acesso, permitindo que cada usuário visualize apenas os módulos e funcionalidades pertinentes às suas atribuições.

Nesse contexto, a unificação dessas atividades em uma única plataforma contribui para tornar a operação mais ágil, segura e eficiente. Além disso, reforça a transparência e o controle sobre recursos, reduz erros manuais e melhora a experiência dos estudantes e colaboradores do refeitório institucional, garantindo um processo de gestão mais moderno, integrado e confiável.

1.3 Tecnologias Adotadas

1.3.1 Laravel (Framework PHP)

Laravel foi escolhido como framework *backend* devido à sua robustez, segurança e facilidade de manutenção. É um dos frameworks PHP mais utilizados no desenvolvimento web moderno, oferecendo recursos como ORM Eloquent para manipulação de banco de dados, sistema de autenticação integrado, validação de dados, e uma arquitetura MVC (*Model-View-Controller*) que facilita a organização do código. Laravel também possui excelente documentação e uma comunidade ativa, facilitando a resolução de problemas durante o desenvolvimento.

1.3.2 Vue.js (Framework JavaScript)

Vue.js foi selecionado para o desenvolvimento do frontend devido à sua curva de aprendizado pequena, reatividade eficiente e capacidade de criar interfaces de usuário dinâmicas e responsivas. A integração do Vue.js com Laravel é bem estabelecida, e sua arquitetura baseada em componentes facilita a reutilização de código e manutenção da aplicação.

1.3.3 TypeScript

O TypeScript é uma linguagem baseada em JavaScript que adiciona tipagem estática ao código. Ele foi adotado no *frontend* por ajudar com a organização da aplicação e reduzir erros durante o desenvolvimento, além de trazer mais segurança na integração com *backend*.

1.3.4 PostgreSQL

O PostgreSQL foi adotado como SGBD relacional por ser um software livre, estável e amplamente utilizado em aplicações institucionais, oferecendo mecanismos sólidos de integridade e segurança dos dados. No contexto deste projeto, isso é importante porque operações como cadastro de estudantes, registro de presença, justificativas e controle de filas exigem que o banco mantenha os dados corretos mesmo quando há muitos acessos simultâneos.

Além disso, o banco de dados integra-se de forma eficiente ao Laravel por meio do Eloquent ORM, facilitando a implementação das operações de CRUD (Create, Read, Update e Delete), como a criação de registros (por exemplo, cadastro de estudantes), a consulta de informações (visualização de cardápio e histórico), a atualização de dados (alteração de itens de refeição) e a exclusão de registros quando necessário, mantendo os relacionamentos entre tabelas de forma organizada.

1.3.5 Docker (Ambiente de Desenvolvimento)

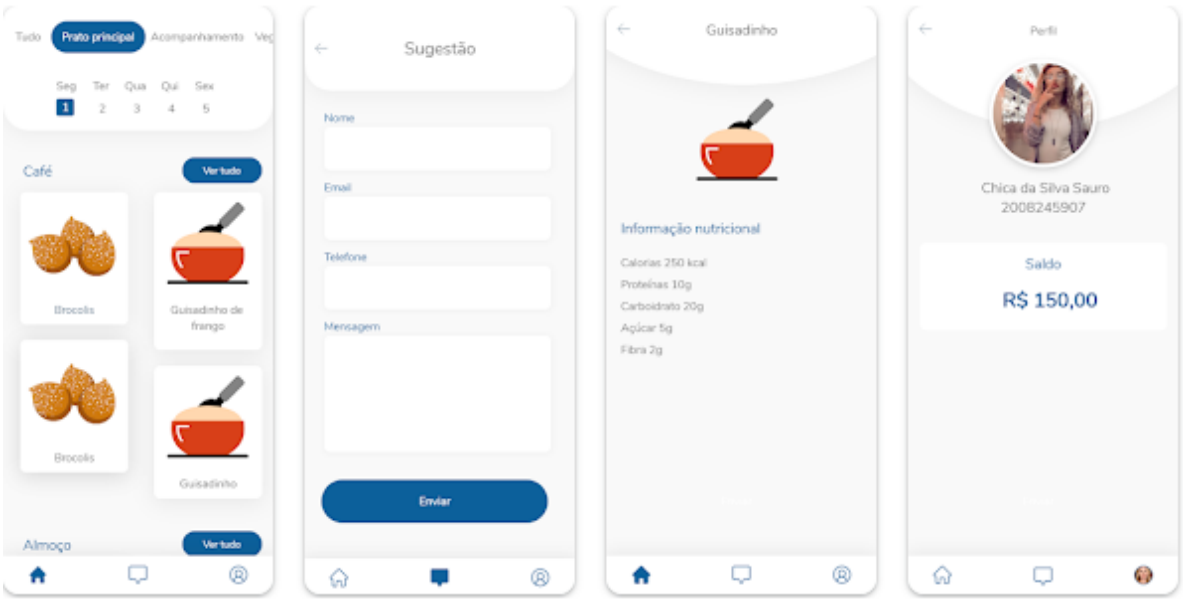
Docker foi utilizado para conteinerização da aplicação, garantindo que o ambiente de desenvolvimento seja consistente entre diferentes máquinas e facilitando o processo de implantação. Isso elimina problemas relacionados a diferenças de configuração entre ambientes de desenvolvimento, teste e produção.

1.4 Trabalhos Relacionados

1.4.1 Sistema RU Digital – UFRN

A Universidade Federal do Rio Grande do Norte (UFRN) implementou um sistema digital para gestão de seu restaurante universitário. O RU Digital da UFRN permite que estudantes e servidores realizem o pagamento de refeições através de um aplicativo móvel ou sistema web, utilizando saldo pré-carregado.

Figura 1 – Página do aplicativo RU UFRN na Google Play Store.



Fonte: SUPERINTENDÊNCIA DE TI DA UFRN (2025).

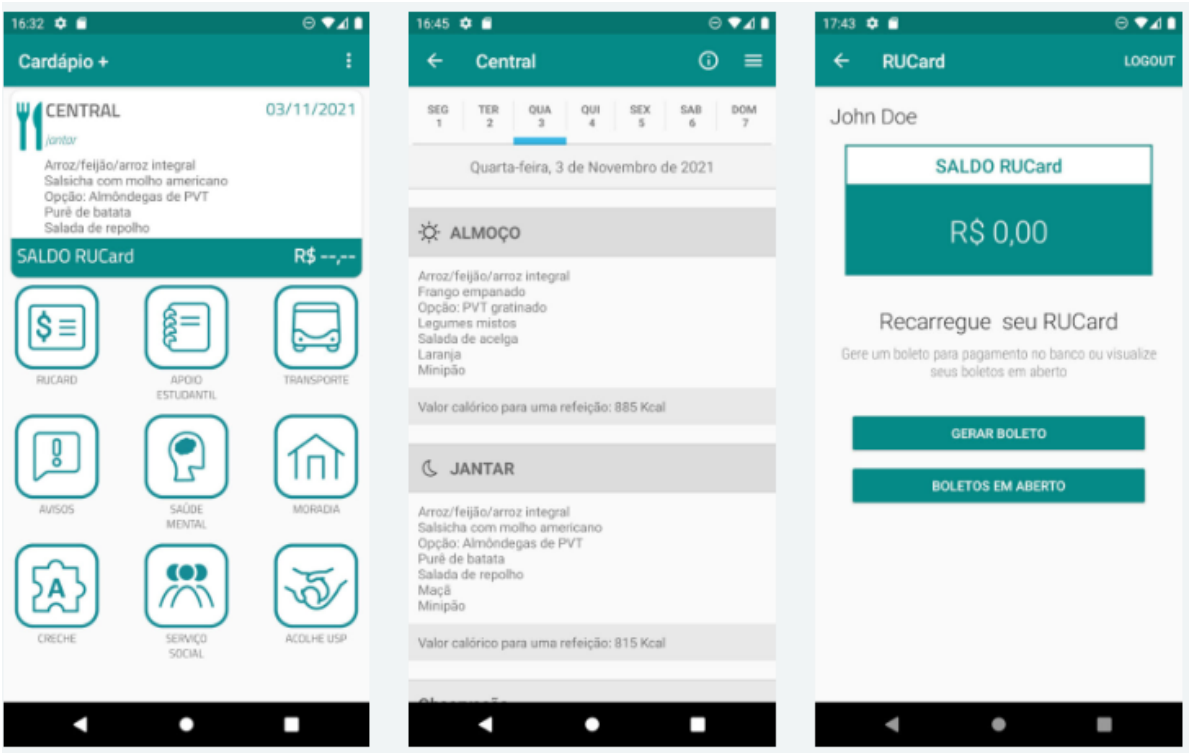
Embora o sistema da UFRN seja uma referência importante em automação de restaurantes universitários, seu modelo difere significativamente do contexto do IFBA. Na UFRN, todas as refeições têm custo, mesmo que subsidiado, enquanto no IFBA as refeições são gratuitas para bolsistas do PAAE. Além disso, o sistema da UFRN não precisa lidar com a gestão de filas de espera e disponibilização de vagas ociosas, que são características específicas do modelo de assistência estudantil do IFBA.

1.4.2 Cardápio+ USP

A Universidade de São Paulo (USP) possui diversos Restaurantes Universitários, conhecidos como "bandejões", que oferecem refeições nutricionalmente balanceadas a preços subsidiados para a comunidade acadêmica. Parte dos estudantes é contemplada pelo Programa de Apoio à Permanência e Formação Estudantil (PAPFE), que concede auxílio alimentação a discentes em situação de vulnerabilidade socioeconômica.

Com o objetivo de centralizar informações institucionais, a USP disponibiliza o aplicativo Cardápio+ USP, lançado em 2011. O sistema permite a consulta aos cardápios dos Restaurantes Universitários, a verificação do saldo do RUCard e a aquisição de créditos por meio de Pix ou boleto bancário. Adicionalmente, o aplicativo reúne informações sobre serviços oferecidos pela Pró-Reitoria de Inclusão e Pertencimento, como transporte, moradia estudantil, serviço social, saúde mental e apoio estudantil.

Figura 2 – Interface do aplicativo Cardápio+ USP.



Fonte: UNIVERSIDADE DE SÃO PAULO (2025).

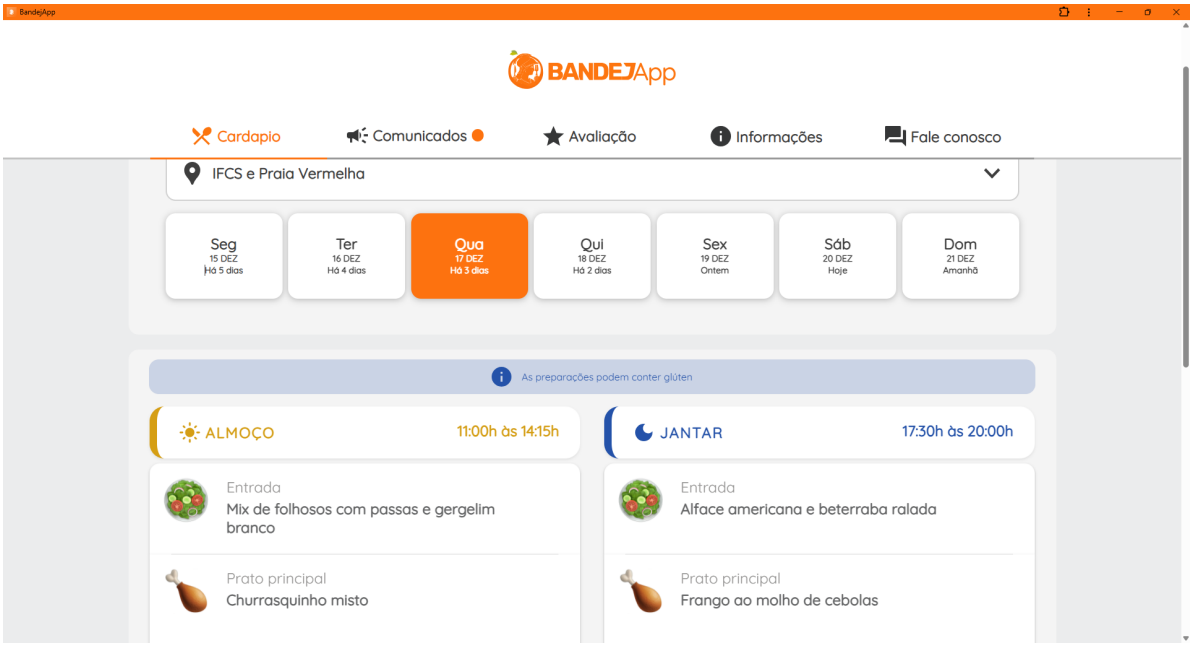
Apesar de sua relevância institucional, o Cardápio+ USP não é voltado especificamente à gestão operacional dos Restaurantes Universitários. Diferentemente da solução proposta neste projeto, o sistema da USP não contempla funcionalidades como controle de acesso às refeições, gestão de beneficiários, gerenciamento de vagas ociosas ou filas de espera.

1.4.3 BandejApp

O BandejApp é um aplicativo desenvolvido no âmbito do Instituto de Computação da Universidade Federal do Rio de Janeiro (UFRJ), com a participação de discentes dos cursos de Ciência da Computação e Comunicação Visual e Design.

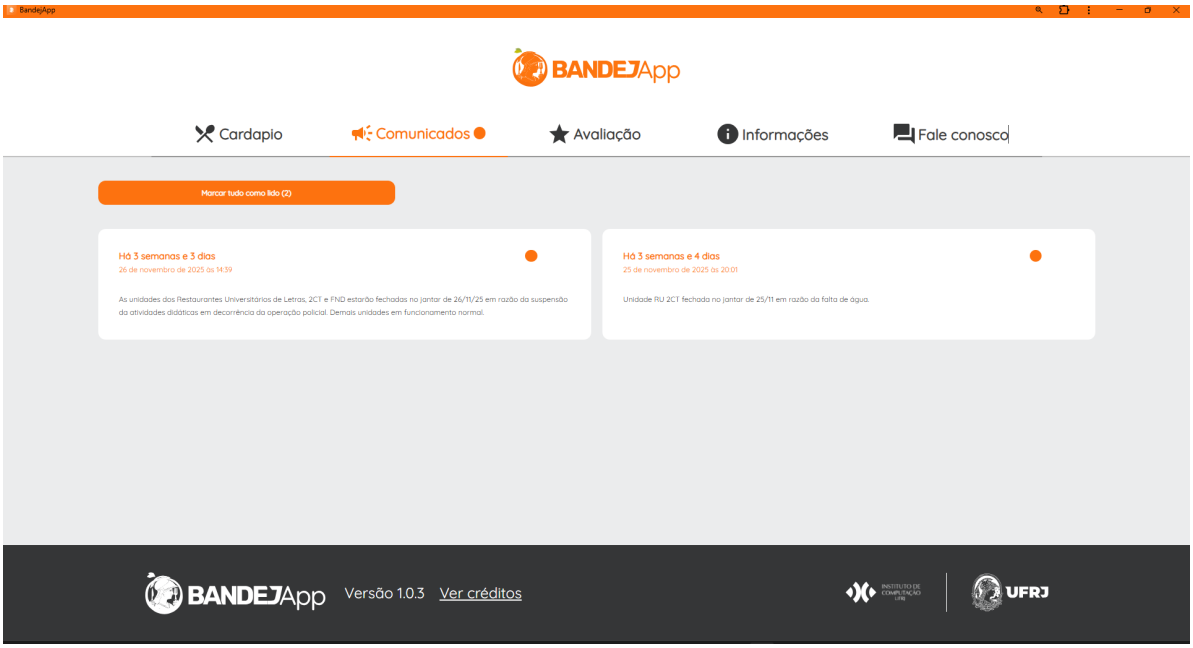
A principal funcionalidade do aplicativo consiste em oferecer um acesso mais prático e eficiente às informações referentes ao cardápio do restaurante universitário. Além disso, o sistema possibilita a consulta aos horários de funcionamento, aos valores das refeições, a avaliação dos pratos pelos usuários e o recebimento de comunicados institucionais. O aplicativo é compatível com diferentes dispositivos móveis, apresenta funcionamento mesmo sem conexão contínua à internet e está integrado aos diversos campi da universidade.

Figura 3 – Interface do aplicativo BandejApp.



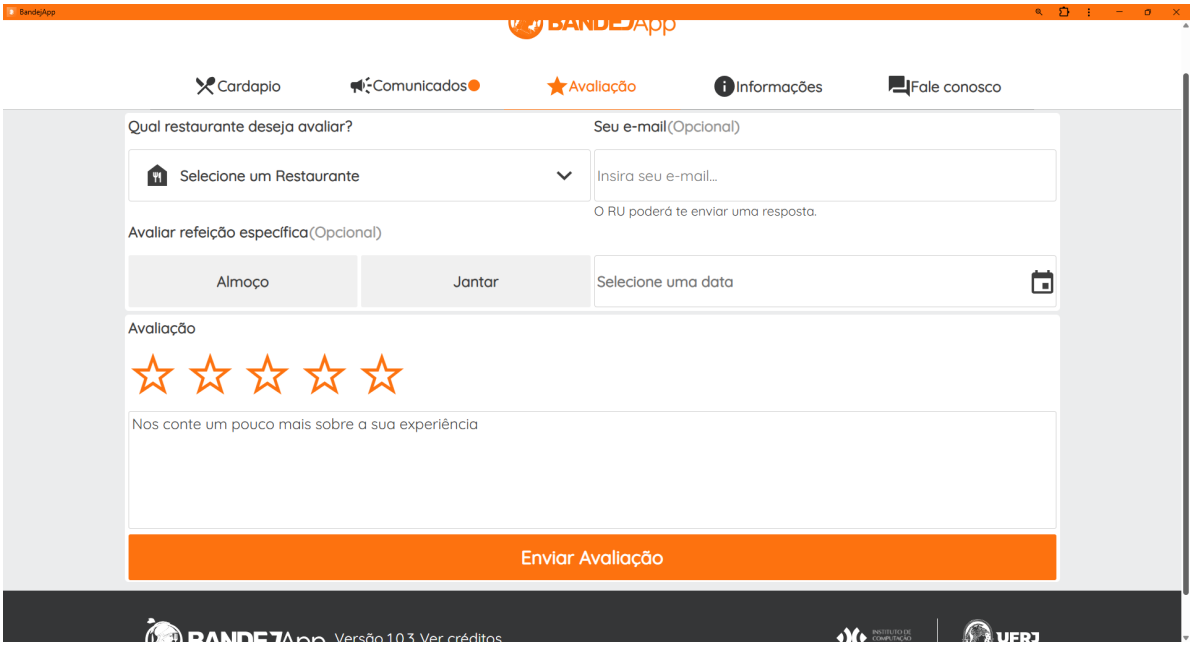
Fonte: Elaboração própria a partir do aplicativo BandejApp (UFRJ, 2025).

Figura 4 – Tela de comunicados do BandejApp.



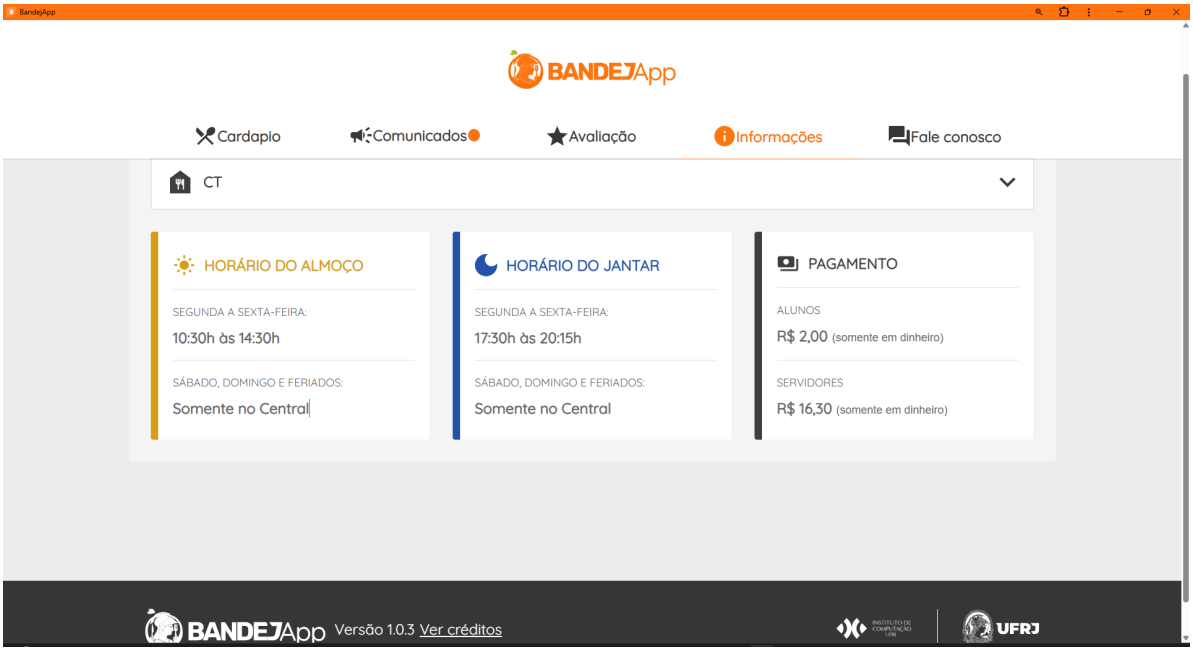
Fonte: Elaboração própria a partir do aplicativo BandejApp (UFRJ, 2025).

Figura 5 – Tela de avaliação de refeição do BandejApp.



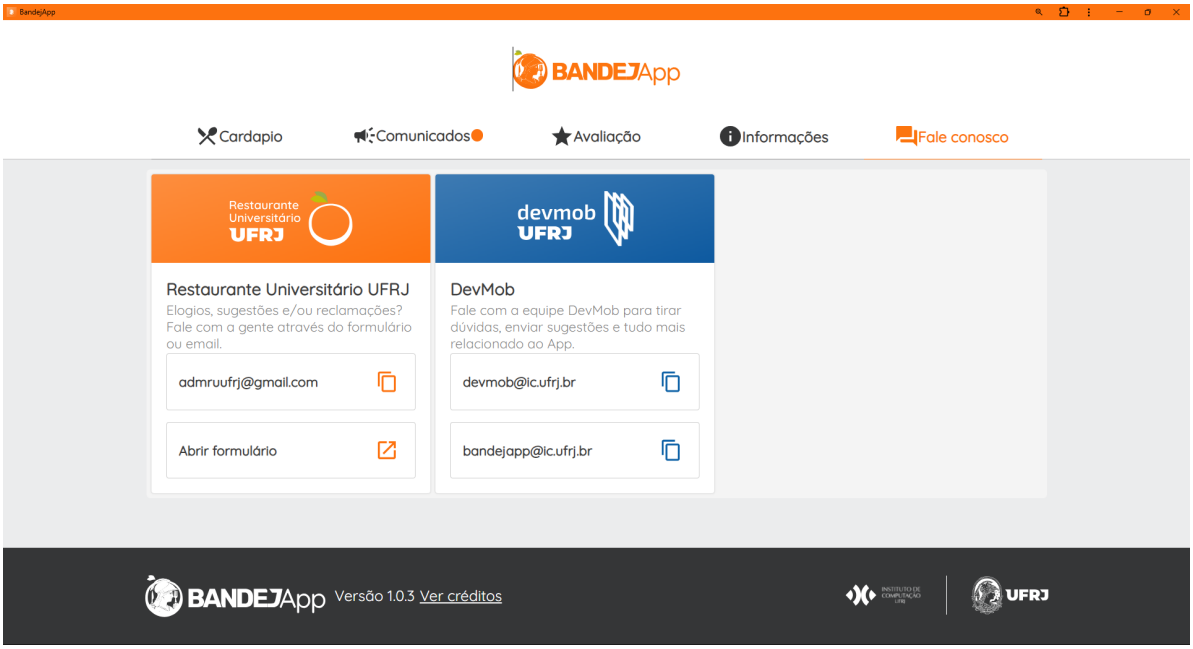
Fonte: Elaboração própria a partir do aplicativo BandejApp (UFRJ, 2025).

Figura 6 – Informações de horário e valores das refeições.



Fonte: Elaboração própria a partir do aplicativo BandejApp (UFRJ, 2025).

Figura 7 – Contatos do Restaurante Universitário e equipe DevMob.



Fonte: Elaboração própria a partir do aplicativo BandejApp (UFRJ, 2025).

1.4.4 Diferenciação da Proposta

A solução proposta se diferencia dos trabalhos relacionados nos seguintes aspectos:

- **Foco em assistência estudantil:** Sistema projetado especificamente para programas de assistência que oferecem refeições gratuitas, não para modelos baseados em pagamento ou subsídio.
- **Justificativa de faltas:** Módulo específico para registro e validação de justificativas com anexo de documentos, permitindo controle mais rigoroso do uso do benefício.

2 Requisitos

2.1 Requisitos Funcionais

2.1.1 Estudantes

RF01 – Cadastro de bolsista
Ator: Estudante bolsista.
Descrição: Como estudante bolsista, desejo realizar meu cadastro na plataforma vinculando minha matrícula, para que minhas informações fiquem registradas no sistema e eu possa acessar os serviços do RI.
Critérios de Aceitação:

1. O estudante deve informar os dados solicitados.
2. A matrícula deve ser validada comparando com a lista de bolsistas aprovados.
3. O sistema deve impedir cadastros com matrículas já registradas.

RF02 – Justificar faltas

Ator: Estudante bolsista.

Descrição: Como estudante bolsista, desejo justificar minhas faltas (antecipadamente ou após a ocorrência) através da plataforma, com possibilidade de anexar atestados ou documentos comprobatórios, para manter meu benefício regular.

Critérios de Aceitação:

1. O estudante pode registrar justificativas antecipadas ou posteriores.
2. O estudante deve informar a data da justificativa.
3. O sistema deve permitir anexar documentos em PDF, JPG ou PNG.
4. A justificativa deve incluir um campo de texto para descrição do motivo.
5. O estudante pode acompanhar o status (Pendente, Aprovada, Rejeitada).
6. O estudante deve receber notificação in-app sobre a decisão.

RF03 – Visualizar cardápio

Ator: Estudante (bolsista e não bolsista).

Descrição: Como estudante, desejo visualizar o cardápio semanal (opções comum e vegetariana) e o cardápio detalhado do dia, para planejar minhas refeições.

Critérios de Aceitação:

1. O cardápio semanal deve exibir as refeições de segunda a sexta-feira.
2. Cada dia deve apresentar as opções: comum e ovolactovegetariana.
3. O cardápio do dia deve destacar a refeição atual com mais detalhes.
4. O cardápio deve ser acessível sem necessidade de login.

RF04 – Consultar histórico de refeições e faltas

Ator: Estudante bolsista.

Descrição: Como estudante bolsista, desejo consultar meu histórico de refeições (presenças) e faltas, para acompanhar minha utilização do benefício.

Critérios de Aceitação:

1. O histórico deve exibir data, tipo de refeição e status (presente, falta, falta justificada).
2. O estudante pode filtrar o histórico por período (semana, mês).
3. O sistema deve exibir resumo com totais de presenças e faltas.
4. O histórico deve ser ordenado por data, do mais recente ao mais antigo.

RF05 – Informar preferência alimentar

Ator: Estudante bolsista.

Descrição: Como estudante bolsista, desejo informar minha preferência alimentar (co-

mum ou ovolactovegetariano), para que o refeitório possa planejar a quantidade adequada de refeições.

Critérios de Aceitação:

1. O estudante pode selecionar entre: Comum e ovolactovegetariano.
2. A alteração entra em vigor a partir do próximo dia útil.
3. O sistema deve exibir a preferência atual no perfil do estudante.

RF06 – Inscrever-se na fila diária de refeições extras

Ator: Estudante não bolsista.

Descrição: Como estudante não bolsista, desejo me inscrever na fila diária de refeições extras, dentro do prazo estabelecido, para concorrer às vagas remanescentes do dia.

Critérios de Aceitação:

1. O estudante deve estar cadastrado e logado na plataforma.
2. A inscrição só pode ser realizada dentro do horário permitido.
3. O sistema deve registrar a ordem de inscrição.
4. O estudante só pode ter uma inscrição ativa por dia/refeição.
5. O estudante deve receber confirmação, in-app, da sua incrição.
6. O estudante pode cancelar sua inscrição a qualquer momento.
7. O sistema deve exibir a posição atual do estudante na fila.

2.1.2 Administração do RI

RF08 – Manter cardápio

Ator: Administrador do RI.

Descrição: Como administrador do RI, desejo cadastrar e atualizar o cardápio, de forma manual ou através de upload via planilha Excel, para disponibilizar as informações aos estudantes.

Critérios de Aceitação:

1. O administrador pode cadastrar cardápio manualmente por dia/refeição.
2. O sistema deve aceitar importação via arquivo Excel.
3. O sistema deve validar o formato do arquivo antes da importação.
4. O administrador pode editar ou excluir itens do cardápio.

RF09 – Visualizar bolsistas do dia

Ator: Administrador do RI.

Descrição: Como administrador do RI, desejo visualizar a lista de bolsistas esperados para o dia, para ter controle da demanda prevista.

Critérios de Aceitação:

1. O sistema deve exibir a lista de bolsistas ativos por turno do dia.
2. A lista deve mostrar nome, matrícula e preferência alimentar.
3. O sistema deve exibir o total de bolsistas esperados.

4. O administrador pode filtrar por turno (almoço/jantar), por nome e por matrícula.

RF10 – Validar justificativas

Ator: Administrador do RI.

Descrição: Como administrador do RI, desejo validar (aprovar ou rejeitar) as justificativas de faltas dos bolsistas, podendo visualizar detalhes, histórico e baixar os anexos comprobatórios.

Critérios de Aceitação:

1. O administrador pode visualizar lista de justificativas pendentes.
2. O sistema deve exibir detalhes da justificativa e histórico do bolsista.
3. O administrador pode visualizar e baixar os anexos comprobatórios.
4. O administrador pode aprovar ou rejeitar com campo de observação.
5. O sistema deve notificar o estudante sobre a decisão via notificação in-app.

RF11 – Acessar dashboard de utilização

Ator: Administrador do RI.

Descrição: Como administrador do RI, desejo acessar um dashboard com estatísticas de utilização (taxa de presença, faltas justificadas/injustificadas, extras atendidos), para análise e planejamento.

Critérios de Aceitação:

1. O dashboard deve exibir taxa de presença diária/semanal/mensal.
2. O sistema deve apresentar gráficos de faltas justificadas vs injustificadas.
3. O dashboard deve mostrar quantidade de refeições extras atendidas.
4. O administrador pode filtrar por período e tipo de refeição.

RF12 – Gerar relatórios

Ator: Administrador do RI.

Descrição: Como administrador do RI, desejo gerar relatórios por período para prestação de contas, controle e identificação de padrões.

Critérios de Aceitação:

1. O administrador pode selecionar tipo de relatório e período.
2. O sistema deve gerar relatórios em Excel.
3. Os relatórios devem incluir dados detalhados e resumos estatísticos.

RF13 – Validar acesso

Ator: Administrador do RI.

Descrição: Como administrador do RI, desejo validar a entrada dos estudantes no momento da refeição através da busca por matrícula/nome ou via leitura de QR Code, para garantir o controle de acesso eficiente e registro adequado de presença.

Critérios de Aceitação:

1. O sistema deve permitir a busca de estudantes por matrícula ou nome, filtrando

automaticamente pelo dia e turno atual.
2. O sistema deve validar se o estudante possui autorização ativa para a refeição do turno correspondente.
3. O sistema deve validar automaticamente a autorização ao realizar a leitura do QR Code do estudante.
4. O sistema deve registrar automaticamente a presença do estudante após validação bem-sucedida.
5. O sistema deve impedir o registro duplicado de presença para o mesmo estudante no mesmo turno e dia.
6. O sistema deve exibir mensagem de confirmação ou erro após cada tentativa de validação.

RF14 – Gerenciar bolsistas
Ator: Administrador do RI.
Descrição: Como administrador do RI, desejo cadastrar, editar e desativar bolsistas para manter a base de dados atualizada.
Critérios de Aceitação:

1. O administrador pode cadastrar novo bolsista.
2. O administrador pode editar informações de bolsistas existentes.
3. O administrador pode desativar bolsistas.

RF15 – Importar lista de bolsistas por turno
Ator: Administrador do RI.
Descrição: Como administrador do RI, desejo importar a lista de bolsistas por turno via arquivo Excel, para atualização em lote da base de dados.
Critérios de Aceitação:

1. O sistema deve aceitar arquivos Excel.
2. O sistema deve validar o formato e dados obrigatórios antes da importação.
3. O sistema deve reportar erros detalhados em caso de falha.

2.2 Requisitos Não-Funcionais

RNF01 – Responsividade
O sistema deve ser totalmente responsivo, adaptando-se a diferentes tamanhos de tela (desktop, tablet e smartphone) para permitir acesso de qualquer dispositivo.

RNF02 – Compatibilidade
O sistema deve ser compatível com os principais navegadores web modernos (Chrome, Firefox, Safari, Edge) em suas versões mais recentes.

RNF03 – Performance
O tempo de resposta para operações comuns (login, confirmação de presença, consulta de

cardápio) não deve exceder 3 segundos em condições normais de rede.

RNF04 – Segurança
O sistema deve implementar autenticação segura, proteção contra CSRF (Cross-Site Request Forgery) e validação de dados no backend.

RNF05 – Disponibilidade
O sistema deve estar disponível aos usuários.

RNF06 – Usabilidade
A interface deve seguir com navegação intuitiva e feedback claro para ações do usuário.

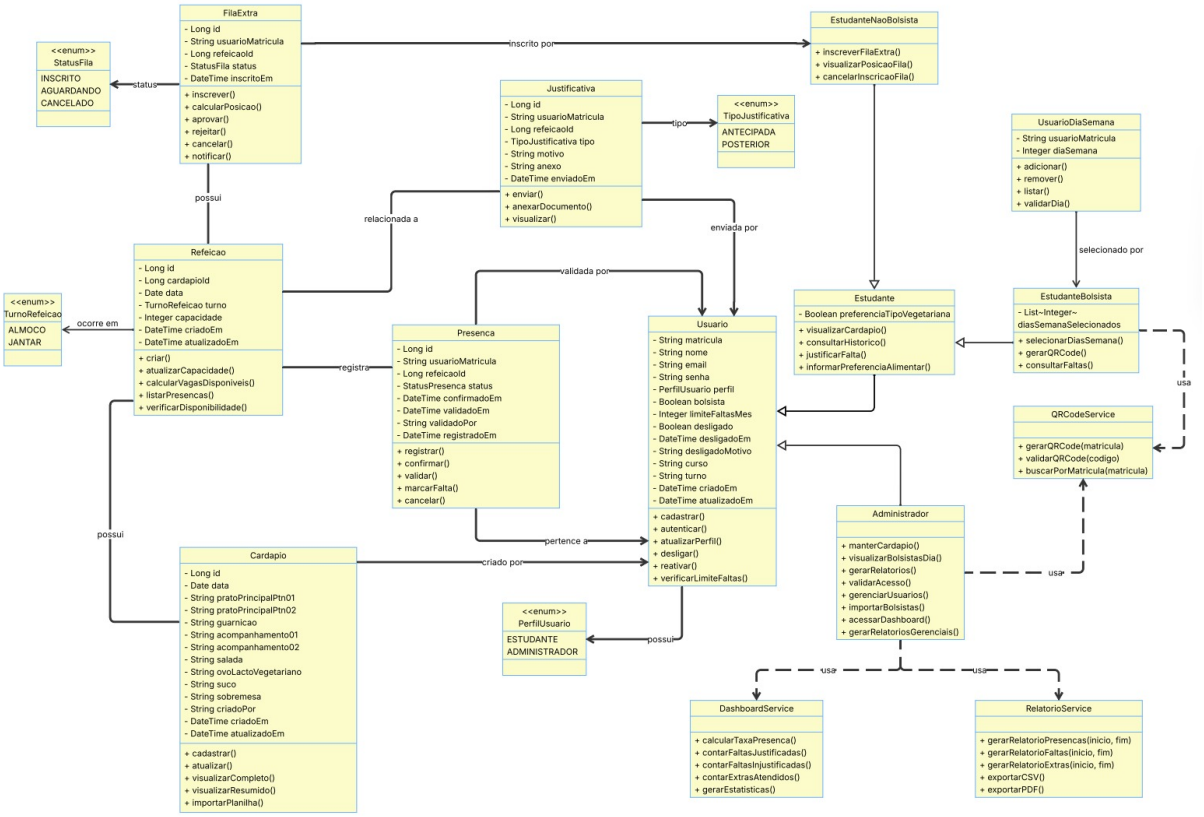
RNF07 – Escalabilidade
A arquitetura do sistema deve permitir crescimento do número de usuários sem degradação significativa de performance.

3 Design

3.1 Projeto UML

Diagrama de Classe

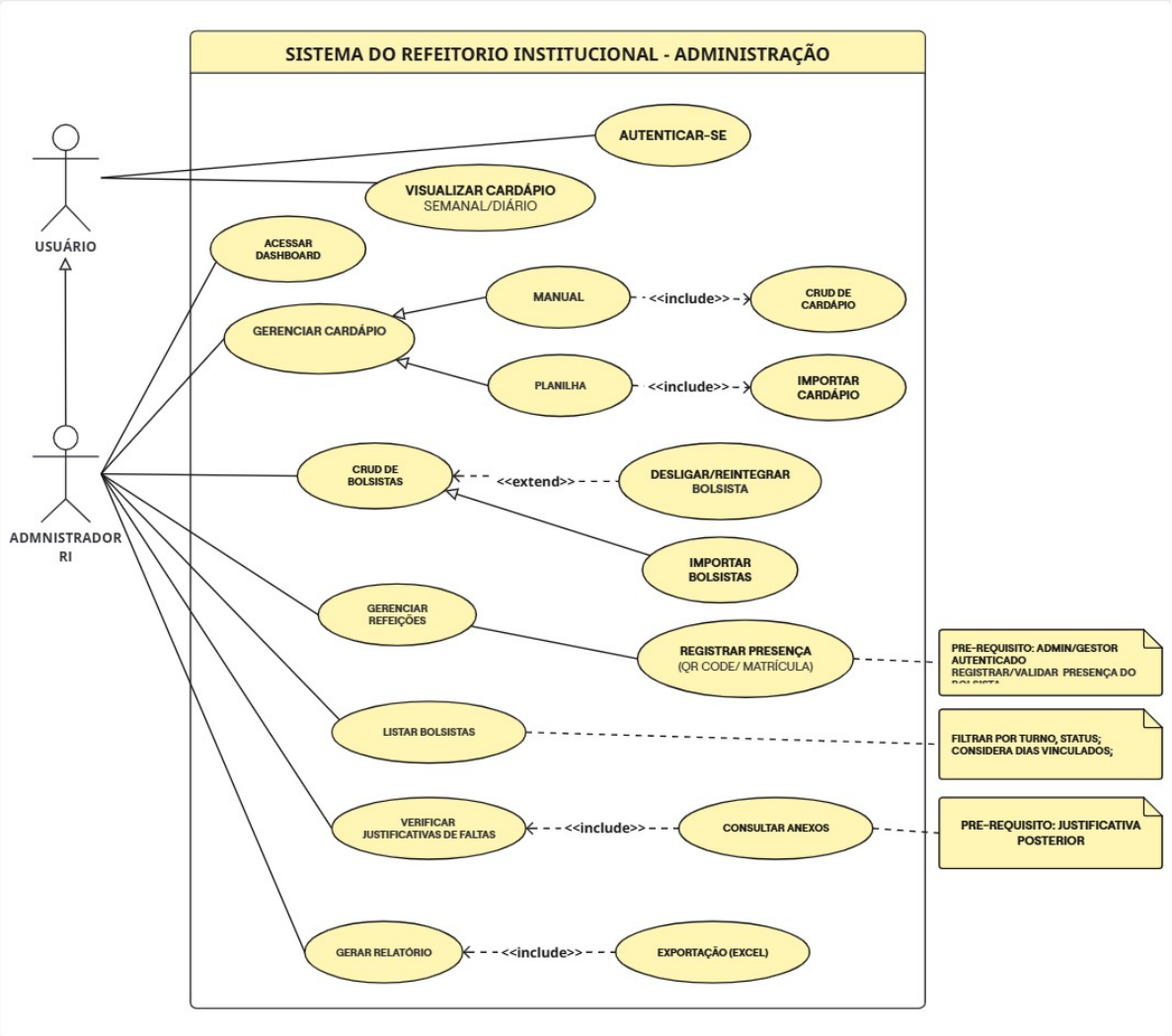
Figura 8 – Diagrama de classes.



Fonte: Elaboração própria do autor.

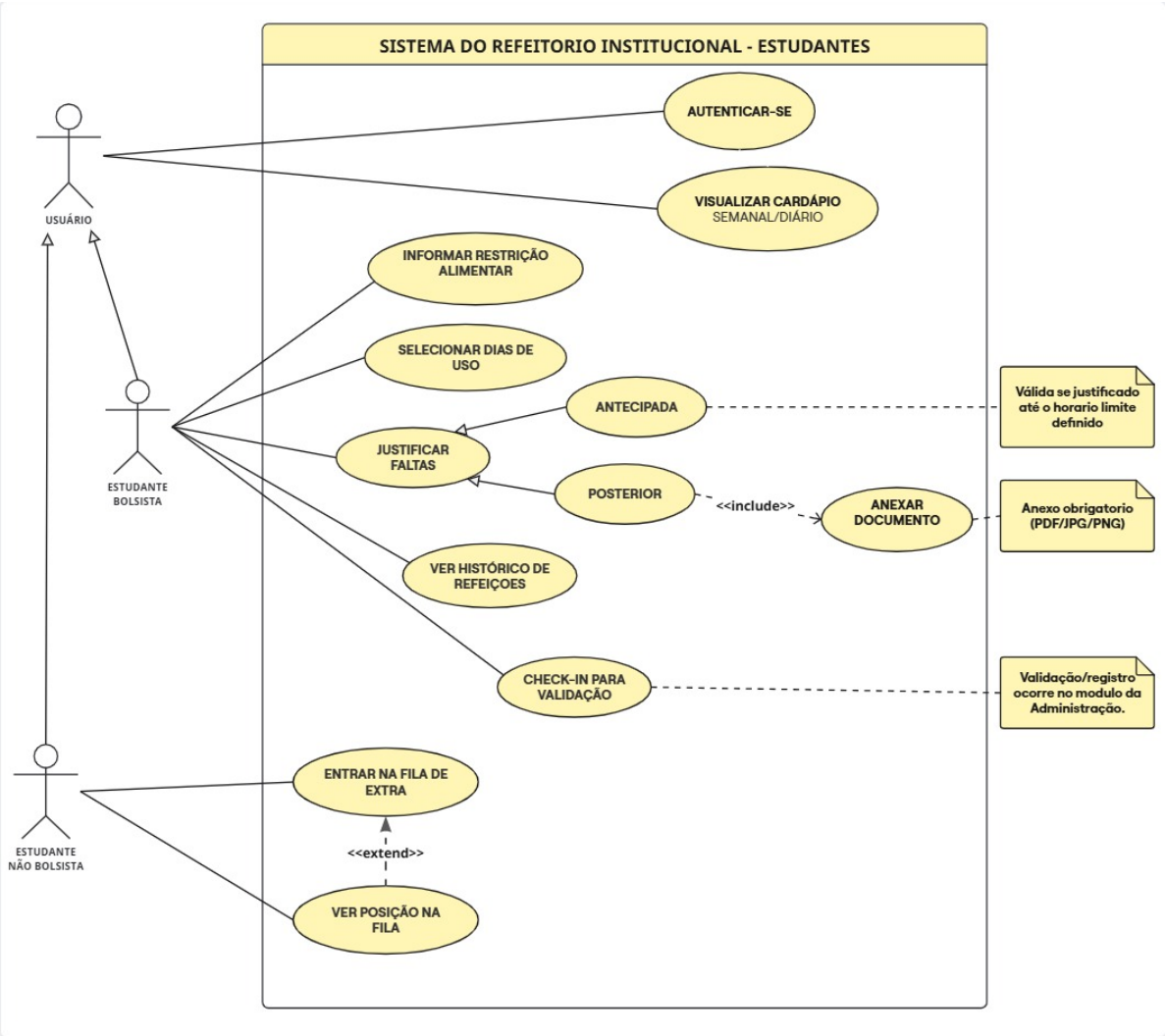
Diagrama de Casos de Uso

Figura 9 – Diagrama de casos de uso de usuários administrativos.



Fonte: Elaboração própria do autor.

Figura 10 – Diagrama de casos de uso de usuários estudantes.



Fonte: Elaboração própria do autor.

3.2 Visão Arquitetural

3.2.1 Visão geral do sistema

A figura 11 permite compreender o contexto da aplicação ao evidenciar as interações essenciais que ocorrem no sistema.

O usuário acessa o sistema por meio de um dispositivo (desktop, notebook ou smartphone), utilizando um navegador. Consome a aplicação e interage com a interface, disparando ações (ex.: autenticar, solicitar dados, enviar formulários).

A camada de apresentação, o *front-end*, desenvolvido em Vue.js, é responsável por exibir as telas no navegador e tratar as interações do usuário. A cada operação realizada (por exemplo, consultas e envio de formulários), o *front-end* se comunica com o servidor por meio de requisições HTTP e atualiza a interface conforme as respostas retornadas.

A camada de aplicação é implementada em Laravel e concentra as regras do sis-

tema, com separação de responsabilidades no fluxo Controlador–Serviço–Modelo (*Controller–Service–Model*). Os controladores atuam como ponto de entrada das requisições, reali-

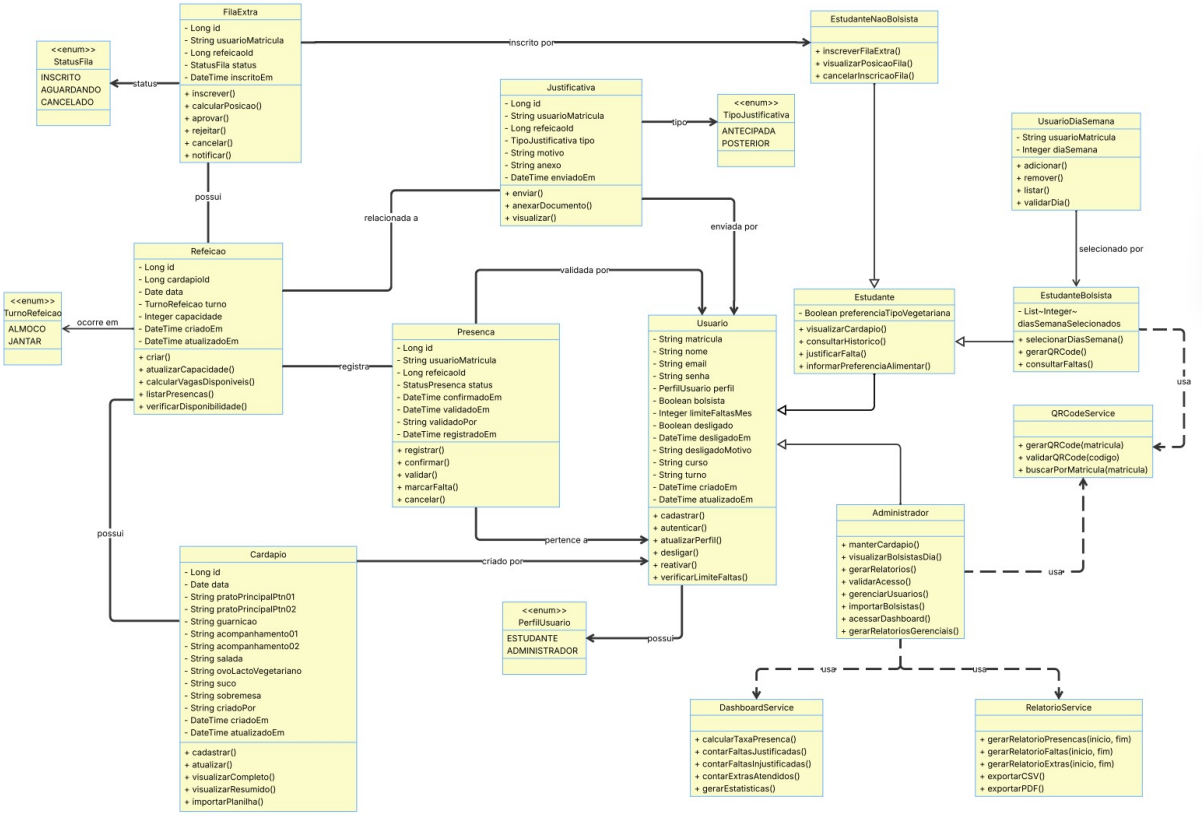
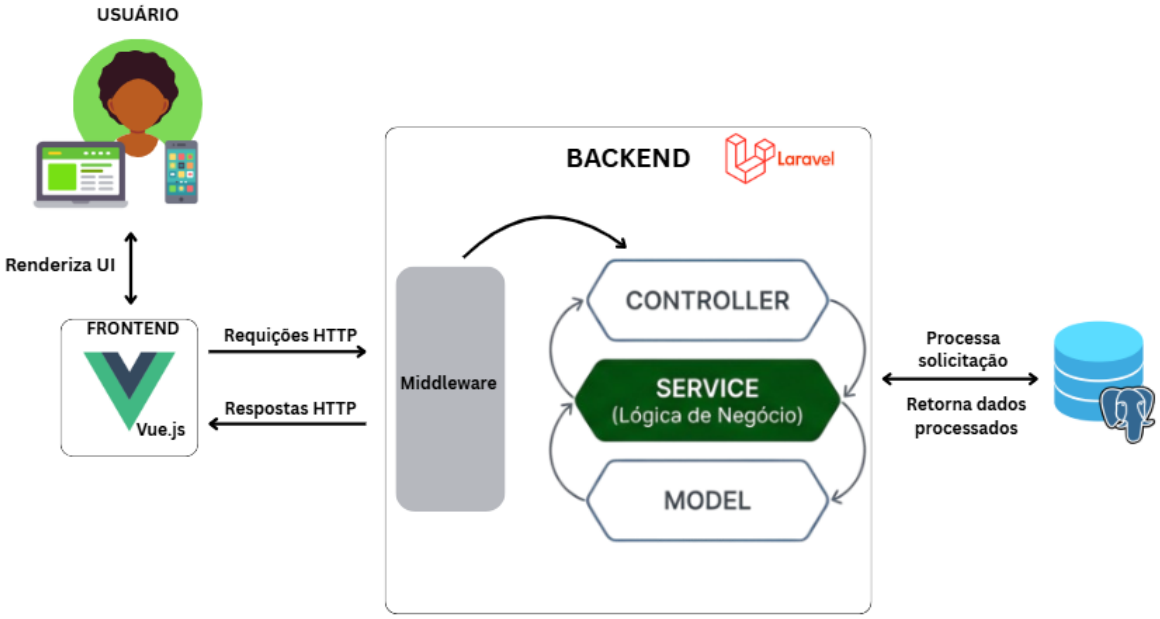


Figura 11 – 1.1

zando validações iniciais e encaminhando o processamento. A lógica de negócio é centralizada na camada de services, responsável por aplicar regras do domínio e coordenar as operações necessárias. Os modelos representam as entidades do sistema e fazem a integração com o banco de dados por meio do ORM Eloquent, permitindo executar consultas e operações de criação, atualização e remoção de registros.

Por fim, a camada de persistência utiliza o PostgreSQL como sistema gerenciador de banco de dados relacional. As informações são armazenadas e recuperadas pelo *back-end* por intermédio dos models, garantindo organização e consistência dos dados. Após o processamento de cada solicitação, o servidor devolve ao *front-end* uma resposta estruturada, que é utilizada para atualizar a interface apresentada ao usuário.

Figura 12 – Visão geral do sistema.

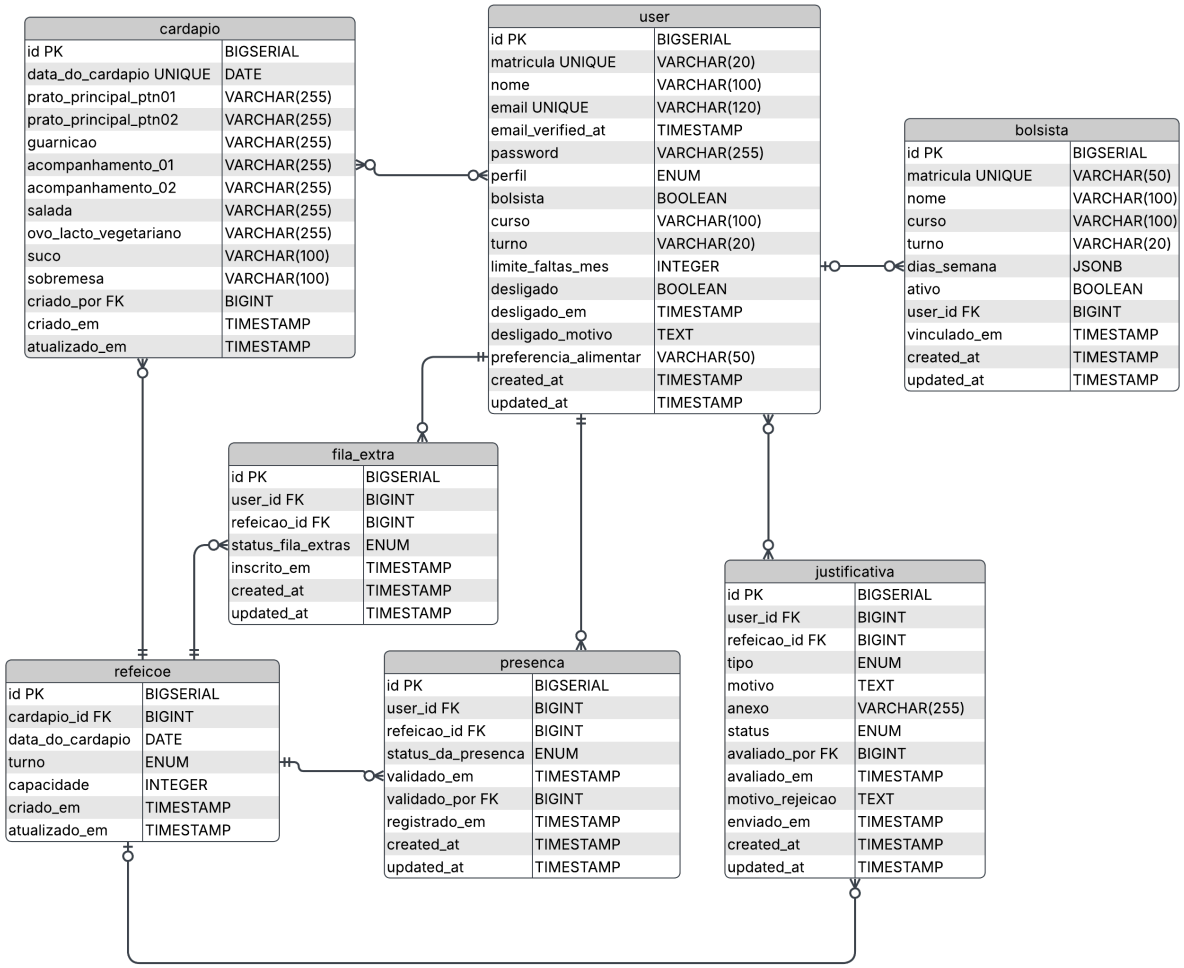


Fonte: Elaboração própria do autor.

3.3 Modelo de Banco de Dados

A estrutura de dados do sistema foi modelada por meio do diagrama apresentado na Figura 13, que representa o modelo lógico e serve de base para a implementação física no PostgreSQL (em sua versão 18).

Figura 13 – Modelo lógico relacional do banco de dados.



Fonte: Elaboração própria do autor.

No modelo relacional proposto, a tabela *usuario* é utilizada para centralizar dados de autenticação e informações cadastrais comuns a todos os perfis, utilizando a coluna *perfil* para diferenciar estudantes e administradores por meio do padrão *Single Table Inheritance* (STI). Essa abordagem permite que a autenticação seja gerenciada de forma unificada e facilita o controle de acesso. Para estudantes, a tabela armazena atributos específicos como curso, turno e a flag *bolsista*, que possibilita a verificação rápida do status de beneficiário sem necessidade de consultar tabelas adicionais.

Informações detalhadas sobre bolsas são mantidas na tabela *bolsista*, relacionada a *usuario*. Essa separação otimiza consultas frequentes ao status de bolsista, permitindo o acesso aos dados detalhados apenas quando necessário. Complementarmente, a tabela *estudante_dias_semana* armazena os dias da semana vinculados a cada estudante bolsista, permitindo a aplicação de regras de acesso específicas por dia.

A tabela *cardapio* possibilita registrar o cardápio associado a uma data específica, contendo informações como prato principal (com duas opções), guarnição, acompanha-

mentos, salada, sobremesa, suco e observações adicionais. A tabela *refeicao* materializa a execução desse cardápio em cada turno.

Para o controle do comparecimento, ou não, de um estudante bolsita é utilizada a tabela de *presenca*. E para apoiar a gestão de ausências, a tabela *justificativa*, permitindo vincular as justificativas de faltas, sejam elas posteriores ou antecipadas, do estudantes bolsistas a uma refeição. Cada justificativa registra o tipo (antecipada ou posterior), motivo textual, possível anexo de arquivo comprobatório. E, para estudantes não bolsistas, a tabela *fila_extra* controla a inscrição na fila diária, relacionando-se com *usuario* (estudante) e *refeicao*, garantindo que um estudante não possa se inscrever mais de uma vez na mesma refeição, evitando duplicidades no sistema de filas.

Figura 14 – Modelo físico do banco de dados.

```sql
-- PRESENÇAS
CREATE TABLE presencas (
  id                  BIGSERIAL PRIMARY KEY,
  usuario_matricula   VARCHAR(32)      NOT NULL REFERENCES usuarios(matricula) ON DELETE CASCADE,
  refeicao_id         BIGINT           NOT NULL REFERENCES refeicoes(id) ON DELETE CASCADE,
  status_da_presença  status_presenca  NOT NULL,
  validado_em         TIMESTAMP,
  validado_por        VARCHAR(32) REFERENCES usuarios(matricula) ON DELETE SET NULL,
  registrado_em       TIMESTAMP        DEFAULT CURRENT_TIMESTAMP,
  UNIQUE(usuario_matricula, refeicao_id)
);
CREATE INDEX idx_presencas_usuario     ON presencas(usuario_matricula);
CREATE INDEX idx_presencas_refeicao    ON presencas(refeicao_id);
CREATE INDEX idx_presencas_validado    ON presencas(validado_em);

-- JUSTIFICATIVAS
CREATE TABLE justificativas (
  id                  BIGSERIAL PRIMARY KEY,
  usuario_matricula   VARCHAR(32)      NOT NULL REFERENCES usuarios(matricula) ON DELETE CASCADE,
  refeicao_id         BIGINT           REFERENCES refeicoes(id) ON DELETE SET NULL,
  tipo                tipo_justificativa NOT NULL,
  motivo              TEXT             NOT NULL,
  anexo               VARCHAR(255),
  enviado_em          TIMESTAMP        DEFAULT CURRENT_TIMESTAMP
);
CREATE INDEX idx_justificativas_usuario   ON justificativas(usuario_matricula);
CREATE INDEX idx_justificativas_refeicao  ON justificativas(refeicao_id);
CREATE INDEX idx_justificativas_tipo      ON justificativas(tipo);
```

Fonte: Elaboração própria do autor.

No modelo físico, foram aplicados mecanismos do PostgreSQL para garantir consistência e desempenho: chaves estrangeiras para garantir integridade nas referências, restrições *UNIQUE* para evitar registros duplicados e índices nos campos mais consultados (por exemplo, matrícula, turno e refeição), otimizando consultas.

4 Projeto de Testes

Os testes foram planejados com foco nas funcionalidades principais do sistema, combinando testes unitários no *back-end*, utilizando o framework PHPUnit, para validação de regras essenciais e testes visuais no *front-end* para verificação do comportamento da interface. Para tornar a validação aderente ao contexto institucional, os cenários e os dados utilizados foram baseados em exemplos e informações representativas fornecidas pelo setor responsável no IFBA, com adoção de cuidados quanto à confidencialidade e à anonimização no uso acadêmico.

4.1 Estratégia de Testes

1. **Teste Unitário (*Unit Tests*):** verificação de regras de negócio em componentes centrais do sistema, como métodos da camada de Modelos e Serviços, com foco em comportamentos essenciais.
2. **Testes Visuais no *front-end*:** validação manual da interface e dos fluxos de navegação, observando consistência visual, exibição de mensagens, estados de carregamento e comportamento dos componentes nas principais funcionalidades.

4.2 Evidência de Execução

As asserções correspondem às verificações realizadas em cada caso de teste, indicando a validação de critérios definidos por cenário (Figura 14).

Listing 1: Teste da camada de serviço de cardápio.

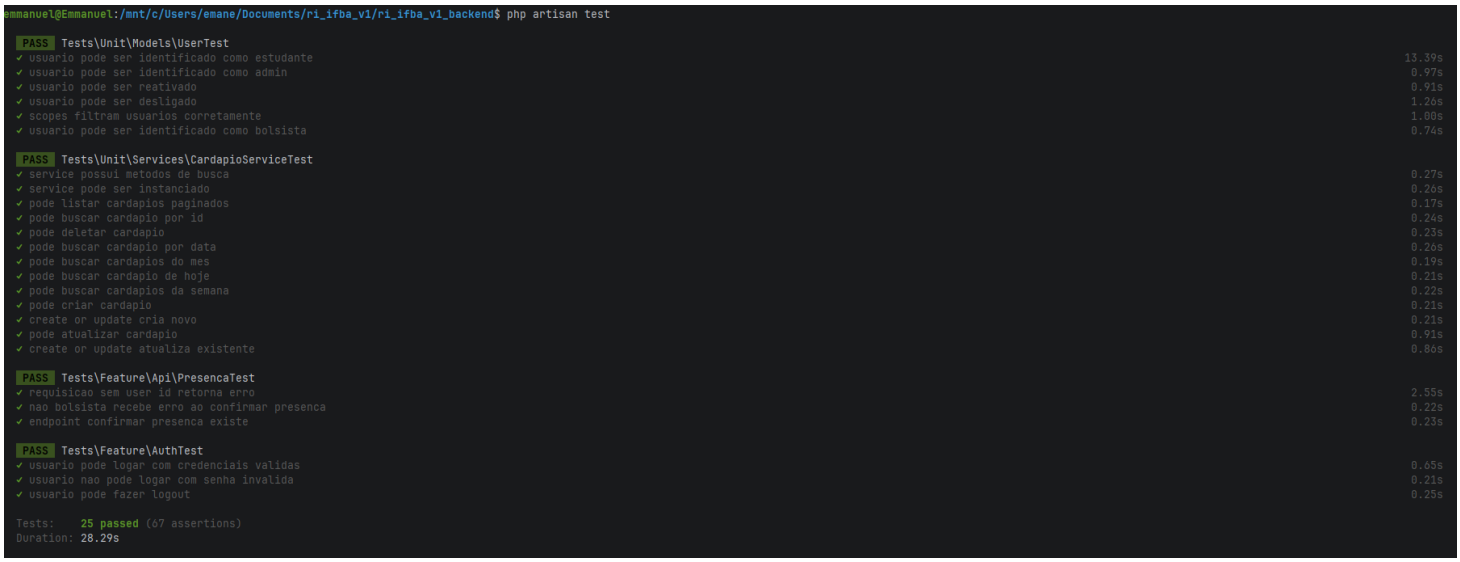
```php
<?php

class CardapioServiceTest extends TestCase
  /**
     * CT14 - Testa criação de um novo cardápio.
     */
  public function test_pode_criar_cardapio(): void
    {
        $user = User::factory()->create();
        $data = [
            'data_do_cardapio' => '2035-03-01',
            'prato_principal_ptn01' => 'Frango Assado',
            'prato_principal_ptn02' => 'Omelete',
            'guarnicao' => 'Purê',
            'acompanhamento_01' => 'Arroz Branco',
            'acompanhamento_02' => 'Feijão Tropeiro',
```

```
            'salada' => 'Alface',
            'turno' => 'almoco',
            'capacidade' => 300
        ];

        $cardapio = $this->service->create($data, $user->id);

        $this->assertInstanceOf(Cardapio::class, $cardapio);
        $this->assertDatabaseHas('cardapios', [
            'data_do_cardapio' => '2035-03-01',
            'prato_principal_ptn01' => 'Frango Assado',
            'criado_por' => $user->id
        ]);
        $this->assertDatabaseHas('refeicoes', [
            'cardapio_id' => $cardapio->id,
            'turno' => 'almoco',
            'capacidade' => 300
        ]);
    }
```

Fonte: Elaboração própria.

Complementarmente, os testes visuais no *front-end* foram conduzidos de forma manual, verificando os fluxos de navegação e a apresentação de estados de interface (carregamento, sucesso e erro) dos componentes nas telas principais.

Figura 15 – Log de execução dos testes (PHPUnit).



Fonte: Elaboração própria.

5 Implementação

5.1 Organização do repositório

A implementação foi organizada em módulos, separando *back-end* e *front-end*, com o objetivo de manter responsabilidades isoladas e facilitar a manutenção e a evolução do sistema.

5.1.1 Repositório do Back-end

O *back-end* foi desenvolvido em Laravel e está disponível em: github.com/emmanueljesus-cyber/ri_ifba_v1_backend.

Estrutura do repositório (back-end):

```
ri_ifba_v1_backend/
+-- app/                        # Nucleo da aplicacao
|   +-- Enums/                  # Estados e tipos centralizados
|   +-- Http/
|   |   +-- Controllers/
|   |   |   +-- Api/V1/         # Versionamento e separacao por dominio
|   |   |       +-- Admin/      # Gestao de bolsistas, cardapios, relatorios
|   |   |       +-- Estudante/  # Perfil, historico, justificativas
|   |   |       +-- Publico/    # Endpoints abertos (cardapio do dia)
|   |   +-- Middleware/         # Camada de seguranca
|   |   +-- Requests/           # Validacoes de entrada
|   |   +-- Resources/          # Transformacao de dados
|   |   +-- Responses/          # Respostas padronizadas
|   +-- Models/                 # Entidades do banco
|   +-- Services/               # Regras de negocio
|   +-- Providers/              # Injecao de dependencias
+-- config/                     # Configuracoes do sistema
+-- database/                   # Persistencia de dados
|   +-- migrations/             # Historico de alteracoes
|   +-- seeders/                # Dados iniciais
+-- routes/                     # Definicao das rotas
|   +-- api.php                 # Rotas por prefixos
+-- tests/                      # Testes automatizados
|   +-- Feature/                # Testes de integracao
|   +-- Unit/                   # Testes de logica isolada
+-- Dockerfile                  # Configuracao da imagem
+-- docker-compose.yml          # Orquestracao dos containers
+-- .env                        # Variaveis de ambiente
```

5.1.2 Perfis e tipagem do domínio

Para padronizar valores de domínio e mitigar inconsistências associadas ao uso de *strings* nas regras de negócio, foram adotados *enums*. Como exemplo, o enum `PerfilUsuario`, apresentado no Código 2, define de forma tipada os perfis de acesso suportados pela aplicação. Essa abordagem centraliza a definição dos valores válidos, evitando divergências de nomenclatura e simplificando verificações de autorização e validações.

Listing 2: Enum de perfis de usuário no back-end.

```php
<?php

namespace App\Enums;

enum PerfilUsuario: string
{
    case ADMIN     = 'admin';
    case ESTUDANTE = 'estudante';
}
```

Fonte: Elaboração própria.

5.1.3 Autorização e controle de acesso às rotas (*middlewares*)

O controle de acesso às rotas do sistema foi realizado por meio de middlewares, aplicados conforme o contexto de execução (público, estudante e administrativo). Eles são como "filtros"que inspecionam e interceptam requisições HTTP antes que cheguem aos controllers, ou, antes mesmo que as respostas sejam enviadas para o cliente, verificando requisitos como autenticação e perfil de acesso. No RI-IFBA, são utilizadas para restringir a execução de operações sensíveis a usuários autenticados com permissões compatíveis a cada um de seus domínios, negando o processamento quando as condições de acesso não são atendidas. O Código 3 apresenta um exemplo de middleware utilizado para essa finalidade.

Listing 3: Middleware de autorização para rotas administrativas.

```php
<?php

class EnsureIsAdmin
{
    public function handle(Request $request, Closure $next): Response
    {
```

```
7           // Verifica se o usuário autenticado tem perfil de admin
8           if (!$request->user() || !$this->isAdmin($request->user())) {
9               return response()->json(
10                  ['message' => 'Acesso negado. Usuário não possui permissão
                    ↪  de administrador.'],
11                  403
12              );
13          }
14          return $next($request);
15      }
16
17      /**
18       * Verifica se o usuário é administrador.
19       */
20      private function isAdmin($user): bool
21      {
22          return $user->perfil === PerfilUsuario::ADMIN;
23      }
24  }
```

Fonte: Elaboração própria.

5.1.4 Modelos (models)

No projeto, o modelo User foi configurado com os principais recursos do Eloquent para garantir segurança e consistência no tratamento dos dados. O atributo fillable define explicitamente quais campos podem ser preenchidos por meio de atribuição em massa. O atributo hidden especifica os campos que devem ser omitidos nas respostas como senhas e tokens , evitando a exposição de informações sensíveis. Já o atributo casts permite a conversão automática de tipos, assegurando que determinados campos sejam sempre retornados no formato esperado pela aplicação, como booleanos, datas ou arrays. Adicionalmente, o processo de autenticação foi ajustado para utilizar a matrícula como identificador de login, substituindo o campo padrão de e-mail do Laravel. Esses elementos são exemplificados no Código 4.

Listing 4: Modelo de usuário (*User*).

```
1   <?php
2
3   class User extends Authenticatable
4   {
5       protected $fillable = [ 'matricula', 'nome', 'email', 'password',
```

```
        'perfil', 'bolsista', 'limite_faltas_mes', 'desligado',
        'desligado_em','desligado_motivo', 'curso', 'turno',
        'preferencia_alimentar', 'restricoes_alimentares',
        'foto_perfil'        ];

    protected $hidden = ['password', 'remember_token'];

    protected $casts = [
        'email_verified_at' => 'datetime',
        'password' => 'hashed',
        'bolsista' => 'boolean',
        'desligado' => 'boolean',
        'desligado_em' => 'datetime',
        'limite_faltas_mes' => 'integer',
        'perfil'            => PerfilUsuario::class,
        'restricoes_alimentares' => 'array',
    ];

    /** IMPORTANTE: Login via matrícula */
    public function getAuthIdentifierName(){return 'matricula'; }
}
```

Fonte: Elaboração própria.

5.1.5 Camada de serviços (services)

A camada de serviços foi adotada como estratégia de organização da lógica de aplicação, concentrando regras de negócio e operações de consulta fora da camada HTTP. Com isso, os controladores permaneceram responsáveis pela orquestração das requisições, enquanto decisões de domínio foram tratadas em componentes específicos. Essa separação reduziu o acoplamento, permitiu a reutilização de regras de negócio entre diferentes pontos do sistema. O Código 5 apresenta um exemplo de implementação de serviço, evidenciando a aplicação de critérios de filtragem, busca e paginação na recuperação de registros.

Listing 5: Service para listagem e busca de usuários com filtros e paginação.

```
<?php

class UserService
{
    public function listarUsuarios(array $filtros = [], int $perPage = 15)
    {
```

```php
        $query = User::query();

        if (isset($filtros['perfil'])) {
            $query->where('perfil', $filtros['perfil']);
        }

        if (isset($filtros['busca'])) {
            $query->where(function ($q) use ($filtros) {
                $q->where('nome', 'ILIKE', "%{$filtros['busca']}%")
                  ->orWhere('matricula', 'ILIKE', "%{$filtros['busca']}%");
            });
        }

        return $query->paginate($perPage);
    }
}
```

Fonte: Elaboração própria.

5.1.6 Camada de Controle (Controllers)

A camada de controle foi responsável por receber as requisições HTTP e direcionar o fluxo de processamento, extraindo parâmetros e filtros, acionando validações de entrada quando aplicável e encaminhando a execução para a camada de serviços. Dessa forma, a lógica de negócio permaneceu concentrada nos serviços, enquanto os controles mantiveram-se focados na condução do ciclo requisição-resposta. O Código 6 exemplifica um controlador administrativo relacionado ao gerenciamento de usuários.

Listing 6: Controller.

```php
<?php

class UserController extends Controller
{
    public function __construct(private UserService $service) {}

    public function index(Request $request): JsonResponse
    {
        $filtros = $request->only(['perfil', 'busca']);
        $usuarios = $this->service->listarUsuarios($filtros);

        return ApiResponse::success(
```

```
            data: $usuarios->items(),
            message: 'Usuários recuperados com sucesso'
        );
    }

    public function store(StoreUserRequest $request): JsonResponse
    {
        try {
            $usuario = $this->service->criarUsuario($request->validated());
            return ApiResponse::created($usuario);
        } catch (\Exception $e) {
            return ApiResponse::error($e->getMessage(), null, 422);
        }
    }
}
```

Fonte: Elaboração própria.

5.2 Rotas e padronização da API

No Laravel, o arquivo de rotas é o responsável por centralizar os *endpoints* da aplicação, associando métodos e caminhos HTTP aos controladores. Esse mecanismo permite aplicar prefixos, agrupar rotas por contexto e vincular middlewares.

No RI-IFBA, a API foi versionada e organizada sob o prefixo `/api/v1`, com estruturação das rotas por contexto de uso: *endpoints* públicos e autenticados (estudante e administração). Além disso, foi adotado um padrão de resposta no formato `{data, errors, meta}`, com o objetivo de padronizar o tratamento de respostas e facilitar o consumo pelo *front-end*. O Código 7 apresenta o arquivo de rotas aplicando os middlewares conforme perfil e contexto de execução.

Listing 7: Arquivo de rotas da API(trecho do `routes/api.php`).

```
<?php
/*
| API Routes
| Prefixo: /api/v1
*/

Route::prefix('v1')->group(function () {

    // Rotas públicas
    Route::prefix('cardapio')->group(function () {
```

```
11          Route::get('hoje', [PublicoCardapioController::class, 'hoje']);
12          Route::get('semanal', [PublicoCardapioController::class,
            ↪  'semanal']);
13      });
14
15      // Autenticação
16      Route::prefix('auth')->group(function () {
17          Route::post('login', [AuthController::class, 'login']);
18
19          Route::middleware('auth:sanctum')->group(function () {
20              Route::post('logout', [AuthController::class, 'logout']);
21              Route::get('me', [AuthController::class, 'me']);
22          });
23      });
24
25      // Rotas do estudante (autenticadas)
26      Route::prefix('estudante')
27          ->middleware('auth:sanctum')
28          ->group(function () {
29              Route::get('perfil', [PerfilController::class, 'show']);
30              Route::put('perfil', [PerfilController::class, 'update']);
31              Route::get('historico', [HistoricoController::class, 'index']);
32          });
33
34      // Rotas administrativas
35      Route::prefix('admin')
36          ->middleware(['auth:sanctum', 'ensure.is.admin'])
37          ->group(function () {
38              Route::prefix('cardapios')->group(function () {
39                  Route::get('/', [AdminCardapioController::class, 'index']);
40                  Route::post('/', [AdminCardapioController::class, 'store']);
41              });
42
43              Route::prefix('presencas')->group(function () {
44                  Route::post('/confirmar', [AdminPresencaController::class,
                    ↪  'confirmar']);
45              });
46          });
47  });
48  });
```

Fonte: Elaboração própria.

5.3 Infraestrutura e ambiente (Docker e variáveis)

Para garantir reprodutibilidade do ambiente, foi adotado Docker com serviços separados para aplicação e banco. O *docker-compose.yml* define a aplicação Laravel e uma instância PostgreSQL, para assegurar disponibilidade do banco antes da inicialização do serviço de API (Código 9).

As variáveis de ambiente foram documentadas no arquivo `.env.example` (Código 8), incluindo orientações sobre comportamento de autenticação em rotas administrativas em ambientes de teste e produção. O arquivo Dockerfile do projeto (Código **??**) descreve a imagem base, dependências e extensões do PHP, instalação do Composer e o comando de inicialização da aplicação.

Listing 8: Arquivo de configuração de ambiente (.env.example).

```
APP_NAME=ri_ifba_v1_backend
APP_ENV=local
APP_KEY=
APP_DEBUG=true
APP_URL=http://127.0.0.1:8000

# Toggle de autenticação nas rotas admin:
# - APP_DEBUG=true   rotas admin SEM autenticação (ambiente de teste)
# - APP_DEBUG=false  rotas admin COM auth:sanctum + ensure.is.admin
 ↪  (produção)

DB_CONNECTION=pgsql
DB_HOST=postgres
DB_PORT=5432
DB_DATABASE=ri_ifba_v1
DB_USERNAME=postgres
DB_PASSWORD= SENHA_DO_BANCO

IMPORT_MAX_FILE_SIZE=5120
IMPORT_MAX_ROWS=0
IMPORT_DEBUG=false
```

Fonte: Elaboração própria.

Listing 9: Orquestração local do ambiente com Docker Compose.

```
services:
  app:
```

```
    build: .
    container_name: ri-ifba-app
    working_dir: /var/www/html
    ports:
      - "8000:8000"
    volumes:
      - .:/var/www/html
    depends_on:
      postgres:
        condition: service_healthy
    environment:
      - DB_HOST=postgres
      - DB_PORT=5432
      - DB_DATABASE=ri_ifba_v1
      - DB_USERNAME=postgres
      - DB_PASSWORD= SENHA_DO_BANCO
    restart: unless-stopped

  postgres:
    image: postgres:16-alpine
    container_name: ri-ifba-postgres
    ports:
      - "5433:5432"
    environment:
      POSTGRES_DB: ri_ifba_v1
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: SENHA_DO_BANCO
    volumes:
      - postgres_data:/var/lib/postgresql/data
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U postgres"]
      interval: 5s
      timeout: 5s
      retries: 5

volumes:
  postgres_data:
```

Fonte: Elaboração própria.

5.4 Implementação do Front-end

O *front-end* foi desenvolvido utilizando Vue.js 3, priorizando reatividade e experiência do usuário. O código-fonte está disponível em: <https://github.com/emmanueljesus-cyber/ri_ifba_v1_frontend>

5.4.1 Organização e estrutura

O projeto segue a estrutura padrão do Vue.js, organizando código em camadas de responsabilidade distintas: componentes de interface, serviços de comunicação, gerenciamento de estado e configuração de rotas.

Listing 10: Estrutura base do repositório front-end:

```
ri_ifba_v1_frontend/
+-- src/
|   +-- assets/                # Estilos globais e recursos estáticos
|   +-- components/            # Componentes reutilizáveis da interface
|   +-- layouts/               # Templates de página por perfil de usuário
|   +-- router/                # Configuração de rotas e navegação
|   +-- services/              # Camada de comunicação com a API
|   +-- stores/                # Gerenciamento de estado global (Pinia)
|   +-- types/                 # Definições de interfaces TypeScript
|   +-- views/                 # Páginas da aplicação
|   |   +-- admin/             # Área administrativa
|   |   +-- estudante/         # Área do estudante
|   |   +-- auth/              # Autenticação
|   |   +-- public/            # Páginas públicas
+-- package.json               # Dependências do projeto
+-- vite.config.ts             # Configuração do ambiente de
↪  desenvolvimento
```

5.4.2 Tecnologias utilizadas

A interface foi construída com Vue 3 usando PrimeVue 4 para componentes de interface e Tailwind CSS para estilização. TypeScript foi adotado para garantir segurança de tipos e o Vite como ferramenta de build.

5.4.3 Comunicação com o back-end

A comunicação com a API foi implementada usando requisições centralizadas em uma camada de serviços, semelhante ao backend. O Código 11 exemplifica a estrutura de um serviço.

Listing 11: Exemplo de *services* para consumo da API (cardapio.service.ts).

```typescript
import api from './api';
import type { Cardapio } from '../types/cardapio';

export const cardapioService = {
  async listarCardapioSemanal(params?: object): Promise<Cardapio[]> {
    const response = await api.get('/cardapio/semanal', { params });
    return response.data.data;
  },

  async buscarCardapioHoje(): Promise<Cardapio> {
    const response = await api.get('/cardapio/hoje');
    return response.data.data;
  },

  async criar(dados: Partial<Cardapio>): Promise<Cardapio> {
    const response = await api.post('/admin/cardapios', dados);
    return response.data.data;
  },
};
```

Fonte: Elaboração própria.

5.4.4 Gerenciamento de estado e rotas

O gerenciamento de estado global foi implementado com Pinia, mantendo informações de autenticação e dados do usuário acessíveis em toda a aplicação. O Código 12 apresenta a store de autenticação, responsável por controlar login, logout e verificação de perfis.

Listing 12: Store de autenticação com Pinia (stores/auth.ts).

```typescript
import { defineStore } from 'pinia';
import { ref, computed } from 'vue';
import type { User } from '@/types/user';

export const useAuthStore = defineStore('auth', () => {
  const user = ref<User | null>(null);
  const token = ref<string | null>(localStorage.getItem('auth_token'));

  const isAuthenticated = computed(() => !!token.value);
```

```
  const isAdmin = computed(() => user.value?.perfil === 'admin');
  const isBolsista = computed(() => user.value?.bolsista === true);

  async function login(credentials: { matricula: string; password: string
  ↪ }) {
    const response = await authService.login(credentials);
    token.value = response.token;
    user.value = response.user;
    localStorage.setItem('auth_token', response.token);
  }

  async function logout() {
    token.value = null;
    user.value = null;
    localStorage.removeItem('auth_token');
  }

  return { user, token, isAuthenticated, isAdmin, isBolsista, login, logout
  ↪ };
});
```

Fonte: Elaboração própria.

O Vue Router gerencia a navegação entre as diferentes áreas do sistema (pública, estudante e administrativa), aplicando guardas de navegação para controlar acesso baseado em autenticação e perfil do usuário. O Código 13 demonstra a configuração de rotas e a implementação das guardas de navegação.

Listing 13: Configuração de rotas e guardas de navegação (router/index.ts).

```
import { createRouter, createWebHistory } from 'vue-router';
import { useAuthStore } from '@/stores/auth';

const routes = [
  {
    path: '/',
    name: 'Home',
    component: () => import('@/views/public/Home.vue'),
  },
  {
    path: '/estudante',
    name: 'EstudanteDashboard',
    component: () => import('@/views/estudante/Dashboard.vue'),
```

```
    meta: { requiresAuth: true, role: 'estudante' },
  },
  {
    path: '/admin',
    name: 'AdminDashboard',
    component: () => import('@/views/admin/Dashboard.vue'),
    meta: { requiresAuth: true, role: 'admin' },
  },
];

const router = createRouter({
  history: createWebHistory(),
  routes,
});

// Guarda de navegação global
router.beforeEach((to, from, next) => {
  const authStore = useAuthStore();

  if (to.meta.requiresAuth && !authStore.isAuthenticated) {
    next({ name: 'Login' });
    return;
  }

  if (to.meta.role === 'admin' && !authStore.isAdmin) {
    next({ name: 'EstudanteDashboard' });
    return;
  }

  next();
});

export default router;
```

Fonte: Elaboração própria.

5.5 Relação com a Implantação

A implementação foi planejada para suportar publicação a partir do repositório, mantendo separação entre camadas. No cenário proposto, o *front-end* seria hospedado em plataforma de publicação estática, Netlify, enquanto o *back-end* e o banco PostgreSQL seriam executados em plataforma gerenciada, Railway. As variáveis de ambiente seriam

configuradas conforme o arquivo `.env.example`. Esse arranjo reduziria a complexidade operacional no estágio inicial e permitiria evolução incremental conforme requisitos de disponibilidade, segurança e escala.

6 Implantação

6.1 Projeto de Implantação

O projeto de implantação define a separação das camadas de apresentação, aplicação e dados em serviços gerenciados na nuvem. A arquitetura proposta considera a utilização de planos gratuitos disponibilizados pelas plataformas no estágio inicial, permitindo migração para planos pagos conforme necessidade de escalonamento.

6.1.1 Requisitos de Hardware

Por tratar-se de uma aplicação hospedada em nuvem, os requisitos são apresentados em dois contextos:

1. **Ambiente de servidor (nuvem):**
   - **CPU:** mínimo de 1 vCPU.
   - **Memória RAM:** mínimo de 512 MB.
   - **Armazenamento:** 1 GB para aplicação e *logs*; 1 GB para o banco de dados relacional.
2. **Ambiente de operação (cliente):**
   - **Dispositivos móveis:** *smartphones* ou *tablets* com acesso à internet (Wi-Fi/4G/5G).
   - **Desktops:** computadores com configuração básica e, no mínimo, 2 GB de RAM, com navegador atualizado.

6.1.2 Requisitos de Software

As especificações de software necessárias para operação do sistema são:

1. **Servidor (*back-end* e banco de dados):**
   - **Sistema operacional:** Linux (via contêiner).
   - **Runtime:** PHP 8.4 ou superior.
   - **Servidor web:** Nginx ou Apache.
   - **SGBD:** PostgreSQL 16 ou superior.
2. **Cliente (*front-end*):**

- **Navegadores:** Chrome, Firefox, Safari ou Edge (versões atuais).
- **Permissões:** acesso a recursos do dispositivo, como câmera e armazenamento local, para funcionalidades de captura e seleção de imagens de perfil.

6.1.3 Plataformas de Hospedagem e *Deploy*

Para a implantação da aplicação (em ambiente de homologação), foram selecionadas duas plataformas de hospedagem gerenciada na nuvem. O Netlify é um serviço especializado na hospedagem de aplicações *front-end* e sites estáticos, oferecendo integração direta com repositórios Git, deploy automático a cada atualização do código-fonte e distribuição de conteúdo por meio de rede CDN global, dispensando a necessidade de configuração manual de servidores web. O Railway é uma plataforma de hospedagem voltada a aplicações *back-end* e bancos de dados, que simplifica o provisionamento de infraestrutura ao abstrair configurações de servidor, oferecendo suporte a múltiplas linguagens e ambientes de execução, gerenciamento de variáveis de ambiente e serviços de banco de dados relacionais integrados ao mesmo ambiente de execução da aplicação.

- **Netlify:** hospedagem da aplicação Vue.js com deploy automático a partir de atualizações na branch principal do repositório, permitindo que alterações no código sejam publicadas automaticamente.
- **Railway:** plataforma de hospedagem da API Laravel, responsável pelo processamento das requisições e execução da lógica de negócio do sistema.
- **Banco de dados (Railway PostgreSQL):** serviço de banco de dados PostgreSQL gerenciado pela mesma plataforma do *back-end*, facilitando a comunicação entre as camadas da aplicação.

Configurações de Ambiente A aplicação requer a definição de variáveis de ambiente específicas para cada camada, conforme modelo disponibilizado no arquivo `.env.example` do repositório:

1. **Railway (PostgreSQL e *back-end*):**
   - Utilização da variável `DATABASE_URL` fornecida automaticamente pelo serviço gerenciado de PostgreSQL.
   - Mapeamento da `DATABASE_URL` para as variáveis do Laravel: `DB_HOST`, `DB_PORT`, `DB_DATABASE`, `DB_USERNAME` e `DB_PASSWORD`.
   - Configuração de `APP_URL` com o domínio público gerado pela plataforma.
   - Definição de `APP_KEY` para criptografia de dados sensíveis.
2. **Netlify (*front-end*):**

- Definição de `VITE_API_BASE_URL` apontando para a URL pública da API hospedada no Railway (por exemplo, `https://ri-ifba.up.railway.app/api/v1`).

6.1.4 Procedimento de *Deploy*

O procedimento de implantação proposto segue as seguintes etapas:

1. **Preparação do repositório:**
   - Garantir que o código está na *branch* principal (`main` ou `master`).
   - Verificar presença dos arquivos de configuração necessários (`.env.example`, `netlify.toml`, etc.).
2. **Configuração do banco de dados (Railway):**
   - Criar novo projeto no Railway e provisionar serviço PostgreSQL.
   - Copiar a `DATABASE_URL` gerada pela plataforma.
3. **Implantação do *back-end* (Railway):**
   - Conectar o repositório GitHub ao Railway.
   - Configurar as variáveis de ambiente necessárias.
   - Executar *deploy* automático, incluindo migração do banco de dados (`php artisan migrate`).
4. **Implantação do *front-end* (Netlify):**
   - Conectar o repositório GitHub ao Netlify.
   - Configurar comando de *build* (`npm run build`) e diretório de publicação (`dist`).
   - Definir variável `VITE_API_BASE_URL` com a URL pública da API.
   - Executar *deploy* automático.
5. **Validação:**
   - Verificar comunicação entre *front-end* e *back-end*.
   - Testar funcionalidades principais da aplicação.
   - Monitorar *logs* das plataformas para identificar possíveis erros.

6.2 Considerações sobre Escalabilidade

A arquitetura proposta permite evolução incremental conforme crescimento da demanda. Os planos gratuitos das plataformas escolhidas oferecem recursos suficientes para operação inicial, com possibilidade de migração para planos pagos que oferecem:

- Maior capacidade de processamento e memória.
- Domínio personalizado.
- *Backups* automáticos do banco de dados.
- Suporte a múltiplas instâncias para balanceamento de carga.
- Monitoramento avançado e alertas de disponibilidade.

A separação clara entre camadas facilita também a migração futura para outras plataformas ou para infraestrutura própria, caso necessário.

7 Manual do Usuário

Este capítulo apresenta as diretrizes operacionais para utilização do sistema Refeitório Institucional (RI-IFBA). O sistema foi projetado com módulos específicos para diferentes perfis de usuário, garantindo que cada tipo de usuário tenha acesso apenas às funcionalidades pertinentes às suas necessidades e responsabilidades.

7.1 Tela Inicial Pública

Antes de realizar o login, todos os usuários têm acesso à página inicial do sistema, que apresenta informações gerais e permite o acesso ao cardápio sem necessidade de autenticação.

7.1.1 Funcionalidades da tela inicial Pública

- **Apresentação do sistema:** Descrição das funcionalidades principais.
- **Acesso ao cardápio:** Botão para visualizar o cardápio do dia sem necessidade de login.
- **Links de acesso:** Opções para login e cadastro de novos usuários.

Figura 16 – Tela inicial pública do sistema RI-IFBA.

Fonte: Elaboração própria.

7.2 Módulo Público (Acesso sem Autenticação)

Além da tela inicial, o sistema disponibiliza funcionalidades públicas que podem ser acessadas sem necessidade de login.

7.2.1 Consulta Pública ao Cardápio

Qualquer usuário pode consultar o cardápio institucional através dos seguintes recursos:

- **Cardápio do dia:** Visualização detalhada das opções de almoço e jantar do dia corrente.
- **Cardápio semanal:** Planejamento das refeições da semana atual, com opções comuns e vegetarianas.

Para acessar o cardápio público, o usuário pode clicar no botão *"Ver Cardápio"* na tela inicial ou acessar diretamente a URL específica para consulta pública.

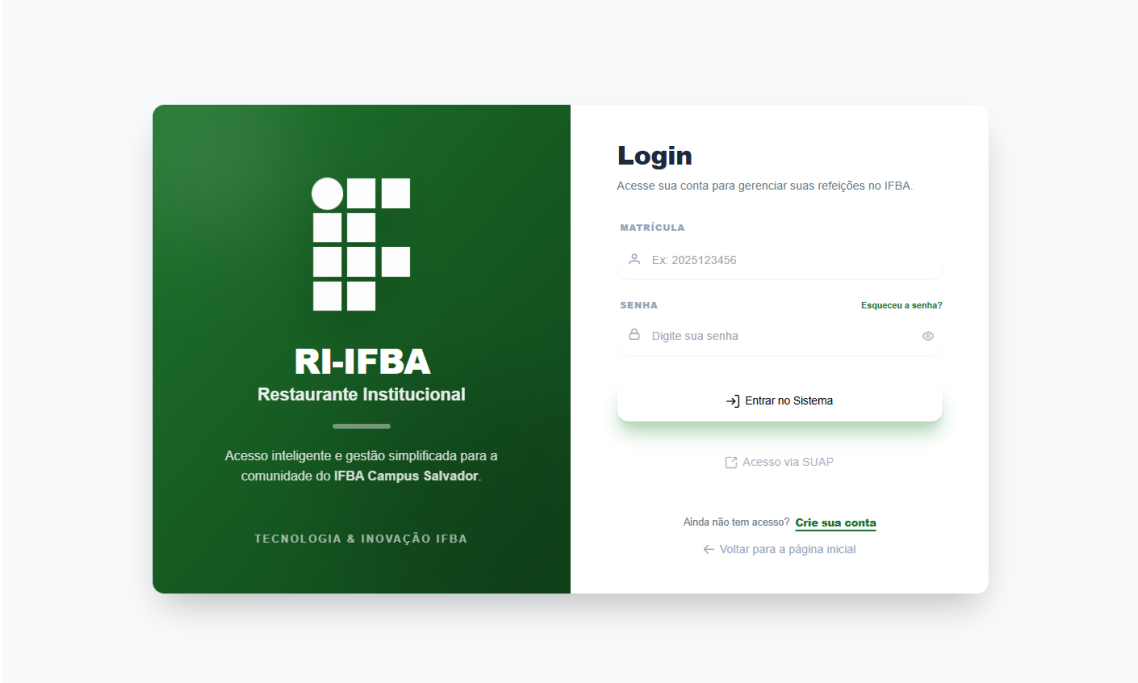
7.3 Acesso Inicial e Autenticação

O sistema utiliza a matrícula institucional como identificador único para autenticação. O primeiro acesso é realizado através do portal web, disponível os dispositivos com navegador moderno.

7.3.1 Primeiro Acesso e Cadastro

1. **Acesso ao sistema:** Navegue até o endereço web do RI-IFBA.
2. **Login:** Insira sua matrícula institucional e senha. O sistema identifica automaticamente o perfil do usuário (Administrador, Estudante Bolsista ou Estudante Não-Bolsista).
3. **Cadastro de novo usuário:** Caso não possua conta, clique em *"Crie sua conta"*. Para bolsistas recém-aprovados, o sistema realiza automaticamente o vínculo com a lista oficial de beneficiários após o preenchimento do formulário.

Figura 17 – Tela de login do sistema RI-IFBA.



Fonte: Elaboração própria.

7.4 Módulo do Estudante (Após Autenticação)

Após o login, todos os estudantes (bolsistas e não-bolsistas) têm acesso ao painel principal e suas funcionalidades.

7.4.1 Dashboard Principal

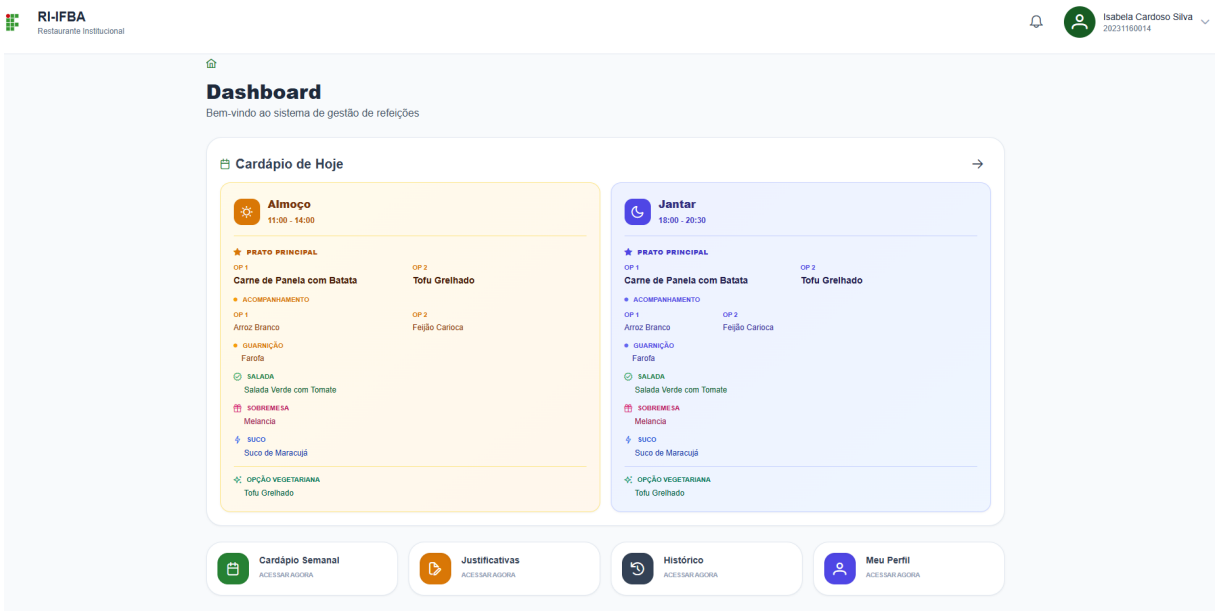
O dashboard apresenta informações consolidadas e personalizadas de acordo com o perfil do usuário:

- **Cardápio do dia:** Detalhamento das refeições do turno atual (almoço e jantar).
- **Status da refeição:** Para bolsistas, indicação se já realizou a refeição do turno.
- **Fila de extras:** Para não-bolsistas, exibe a posição atual na fila e a disponibilidade de vagas.
- **Menu rápido:** Acesso direto às principais funcionalidades do sistema.
- **Dashboard:** O dashboard adapta-se dinamicamente ao tipo de usuário.

Acompanhamento da Fila de Extras no Dashboard Para estudantes não-bolsistas, o dashboard fornece a situação na fila de extras:

- **Status da inscrição:** Indicação visual.
- **Vagas disponíveis:** Quantidade de vagas extras liberadas para o turno.
- **Ações rápidas:** Possibilidade de cancelar a inscrição.

Figura 18 – Dashboard principal do estudante bolsista.

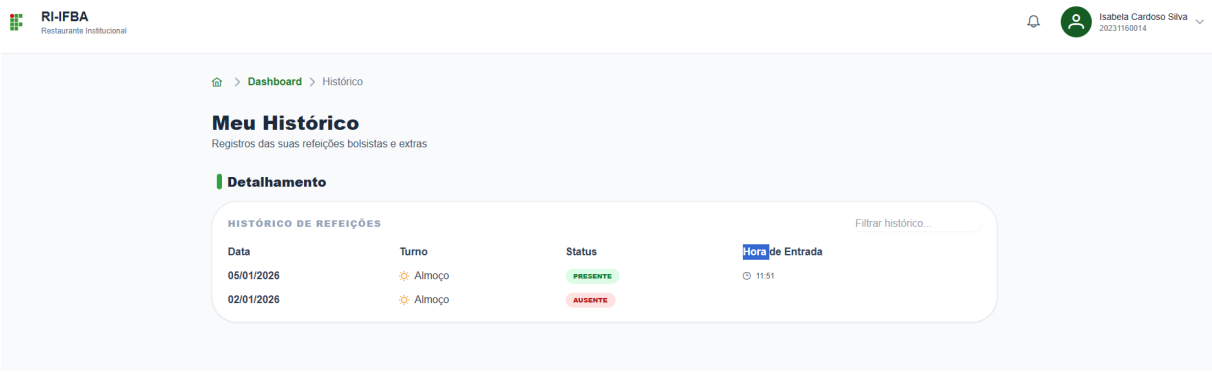


Fonte: Elaboração própria.

7.4.2 Histórico de Refeições

Os estudantes podem consultar o histórico de refeições:

Figura 19 – Histórico de refeições do estudante.



Fonte: Elaboração própria.

7.5 Módulo do Estudante Bolsista

Estudantes beneficiários do Programa de Assistência e Apoio ao Estudante (PAAE) têm acesso a funcionalidades específicas relacionadas ao controle de frequência e justificativas.

7.5.1 Gestão de Justificativas

Bolsistas que não puderem comparecer devem registrar justificativa para evitar acúmulo de faltas. Existem dois tipos de justificativas:

- **Ausência Programada (Antecipada):** Quando o estudante sabe previamente que não comparecerá (ex: atividade acadêmica externa). Não exige comprovação documental.
- **Ausência Posterior:** Quando ocorre um imprevisto (ex: motivo de saúde), exigindo envio de documento comprobatório.

Regras para Justificativas

- **Prazos:**
  - Antecipada: Almoço até 13:30h, Jantar até 19:00h do dia da ausência.
  - Posterior: Até o penúltimo dia letivo do mês.
- **Motivos aceitos:**
  - Ausência por motivo de saúde (com atestado médico).

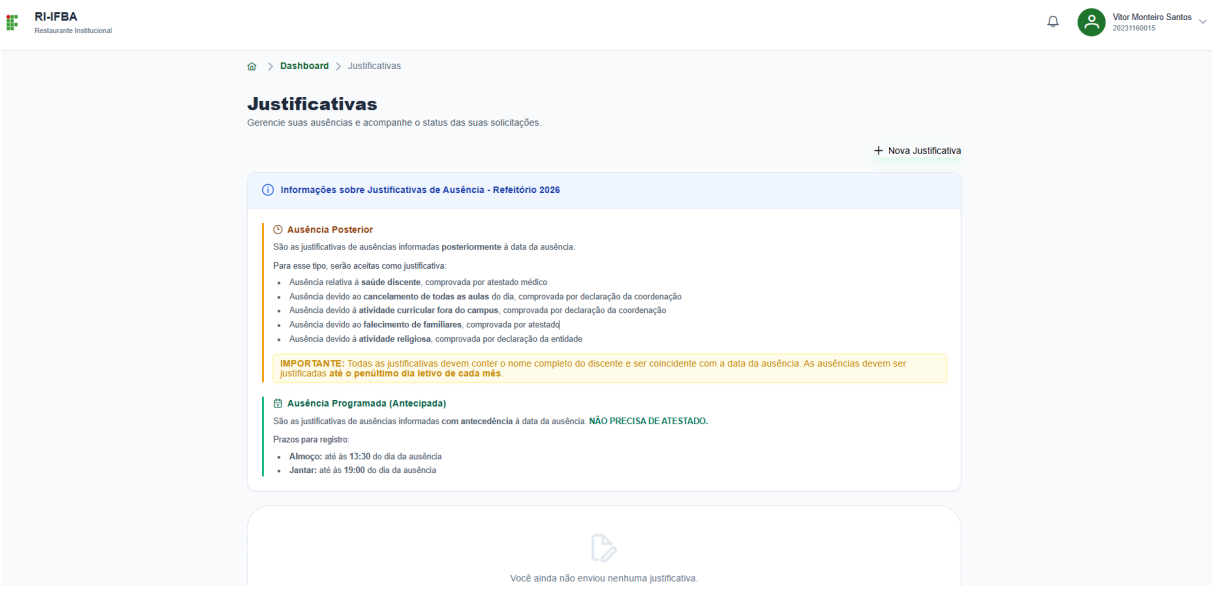
– Cancelamento de todas as aulas do dia (com declaração).

– Atividade curricular externa (com declaração).

– Falecimento de familiar (com atestado).

– Atividade religiosa (com declaração da entidade).

Figura 20 – Tela de informações sobre justificativas.

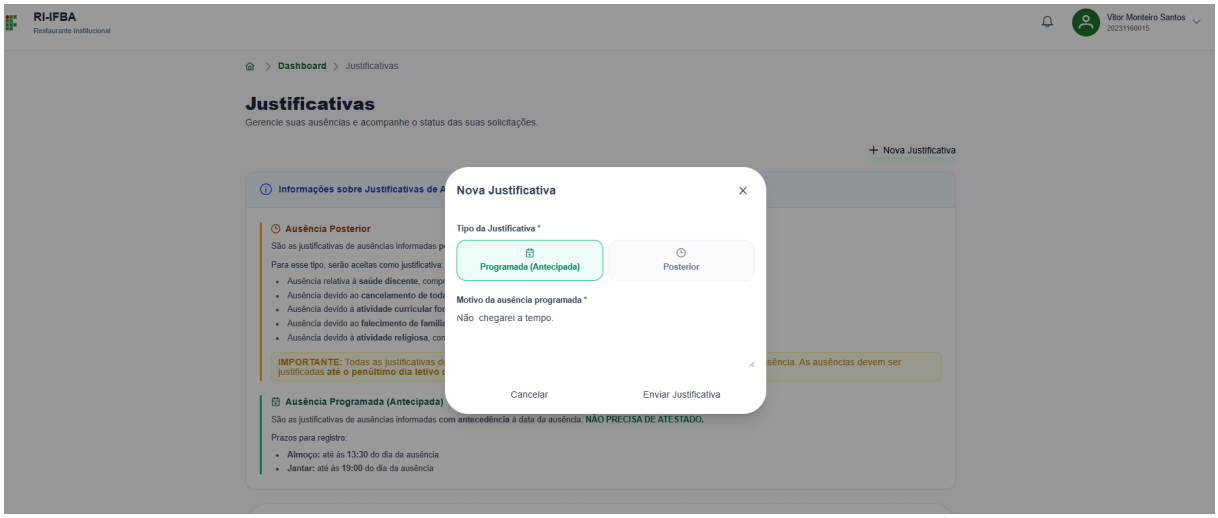


Fonte: Elaboração própria.

7.5.2 Envio de Justificativa Antecipada

1. Acesse *"Justificativas"* no menu.
2. Clique em *"+ Nova Justificativa"*.
3. Selecione *"Programada (Antecipada)"*.
4. Escolha data, turno e informe o motivo.
5. Clique em *"Enviar Justificativa"*.

Figura 21 – Formulário de justificativa antecipada.

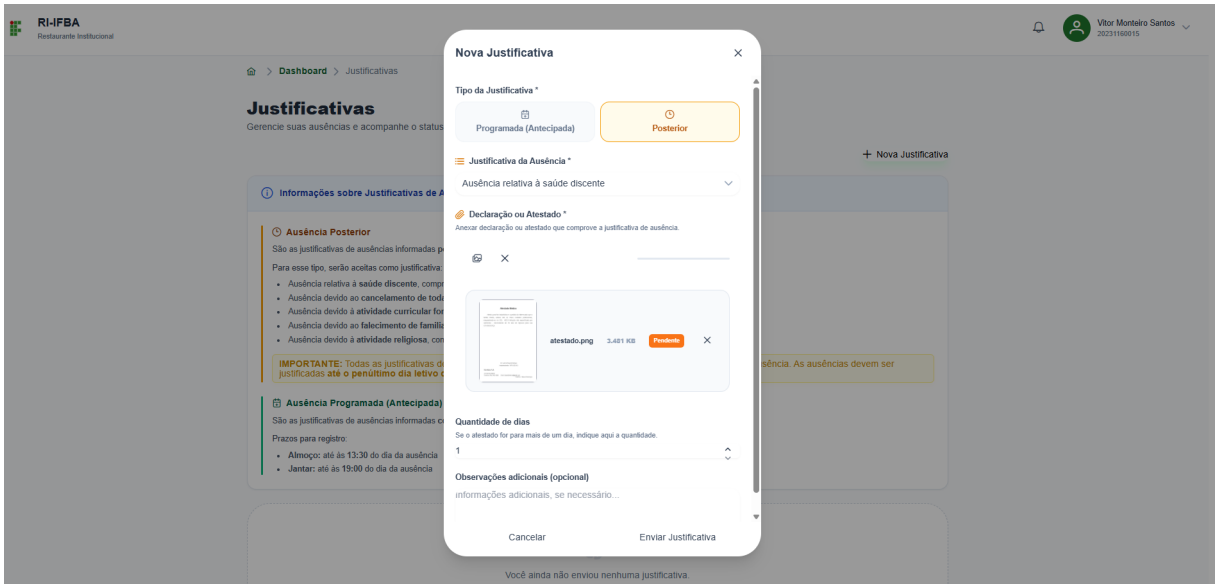


Fonte: Elaboração própria.

7.5.3 Envio de Justificativa Posterior

1. Acesse *"Justificativas"* no menu.
2. Clique em *"+ Nova Justificativa"*.
3. Selecione *"Posterior"*.
4. Escolha data, turno, motivo e anexe o documento comprobatório.
5. Informe quantidade de dias (se aplicável) e observações.
6. Clique em *"Enviar Justificativa"*.

Figura 22 – Formulário de justificativa posterior com upload de documento.



Fonte: Elaboração própria.

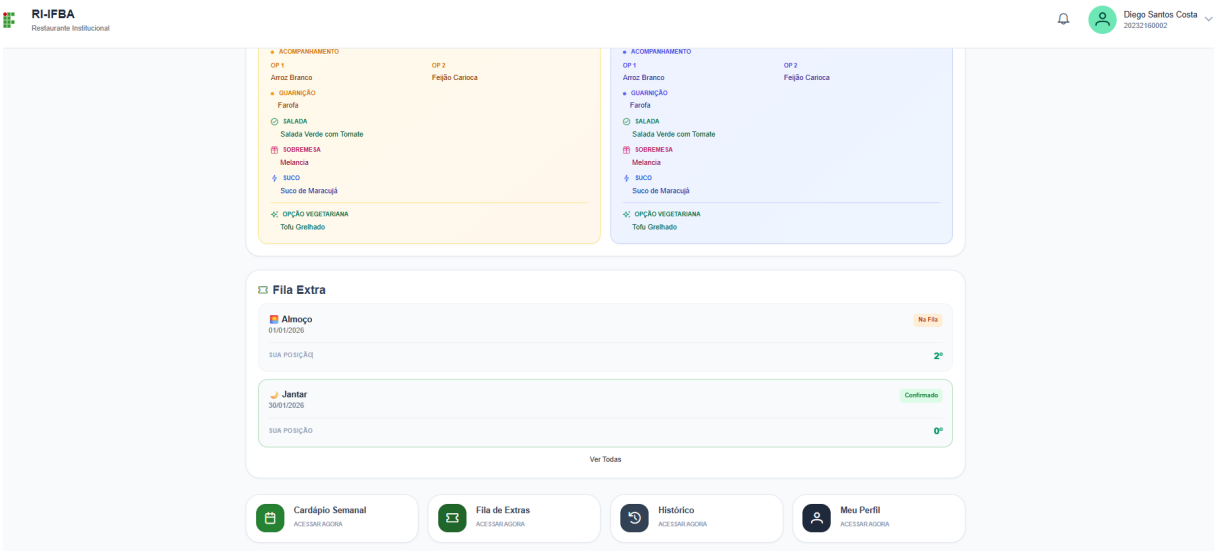
7.6 Módulo do Estudante Não-Bolsista

Estudantes que não são beneficiários do PAAE têm acesso à funcionalidade de inscrição na fila de refeições extras.

7.6.1 Fila de Refeições Extras

Funcionalidade destinada a estudantes que não possuem bolsa alimentação e desejam concorrer às vagas remanescentes.

Figura 23 – Dashboard do estudante não bolsista.



Fonte: Elaboração própria.

Regras de Funcionamento

- **Inscrição:** Realizada diariamente, dentro do horário estabelecido.
- **Seleção:** Ordem de chegada (primeiro a se inscrever, primeiro a ser atendido).

Procedimento de Inscrição

1. Acesse *"Fila de Extras"* no menu.
2. Verifique a disponibilidade de vagas para o turno desejado.
3. Clique em *"Inscrever-se"*.
4. Acompanhe sua posição na fila através do painel.
5. Compareça ao refeitório se sua posição estiver dentro do número de vagas disponíveis.

7.7 Módulo do Administrador

O perfil de Administrador é responsável pela gestão completa do sistema, incluindo manutenção de dados base, validação de refeições e geração de relatórios.

7.7.1 Gestão de Cardápios

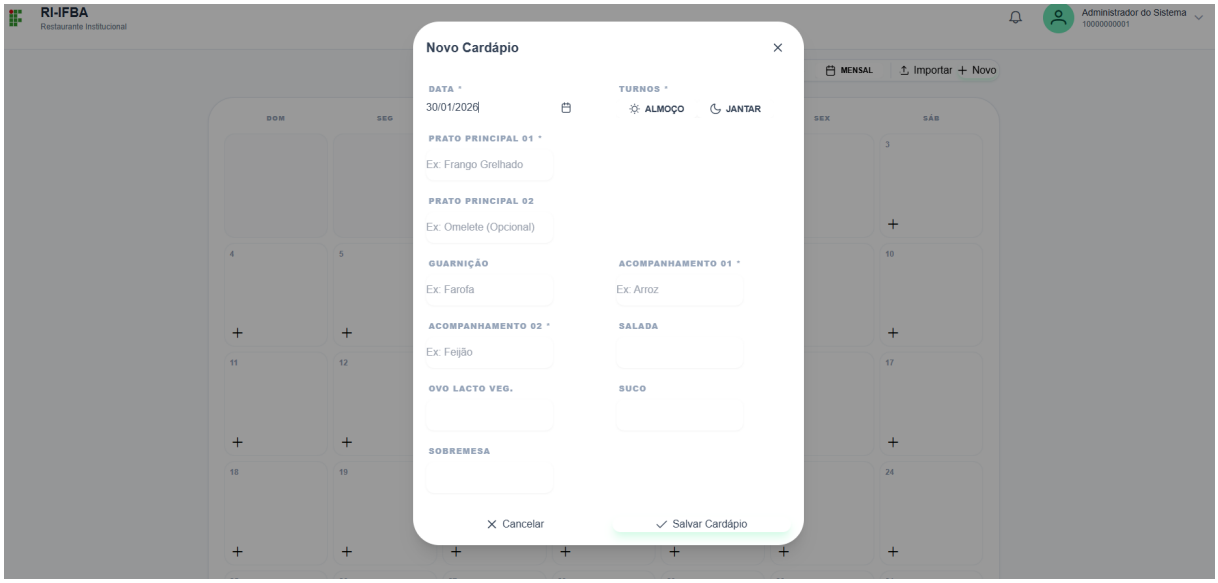
O sistema oferece múltiplas formas de gerenciamento de cardápios:

- **Modo calendário:** Visualização mensal com adição rápida em dias sem cardápio.
- **Importação de cardápio:** Carregamento de cardápios via planilha Excel.

Criação Manual de Cardápio

1. Acesse *"Gestão de Cardápio"*.
2. Clique em *"Novo Cardápio"*.
3. Defina data e turnos (Almoço, Jantar ou ambos).
4. Preencha os componentes: proteína, guarnição, salada, sobremesa, suco.
5. Clique em *"Salvar"*. O sistema cria automaticamente as refeições vinculadas.

Figura 24 – Criação Manual de Cardápio.



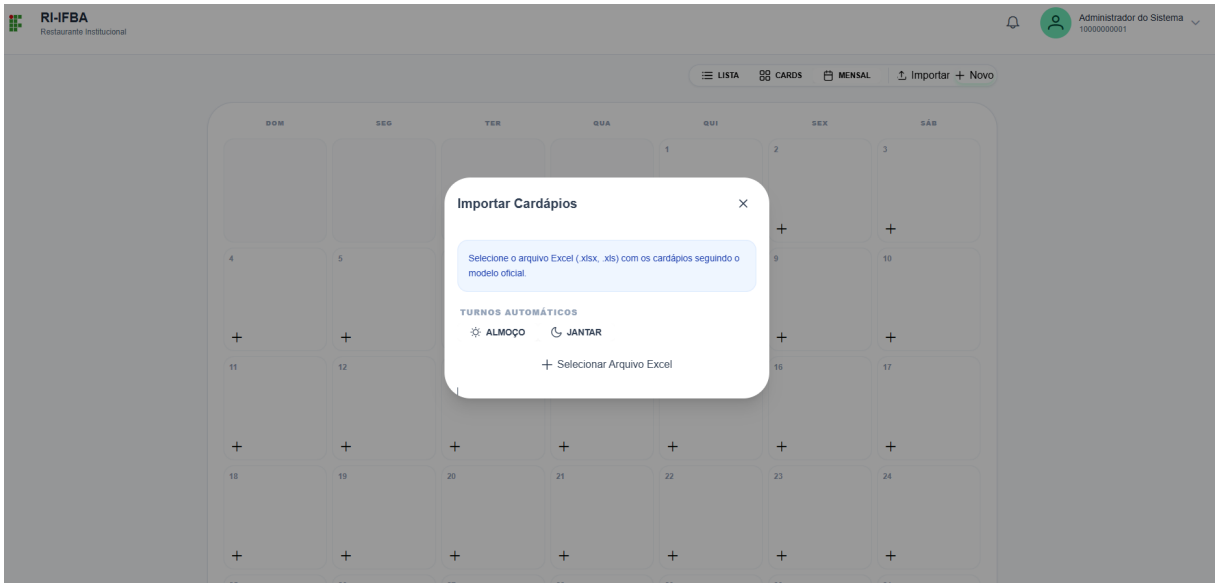
Fonte: Elaboração própria.

Importação via Planilha

1. Acesse *"Gestão de Cardápio"*.
2. Clique em *"Importar"*.
3. Baixe o modelo oficial e preencha com os dados.
4. Selecione o arquivo e os turnos desejados.

5. Clique em *"Importar"*. O sistema valida e cria os cardápios.

Figura 25 – Importação via Planilha.



Fonte: Elaboração própria.

7.7.2 Controle de Presenças

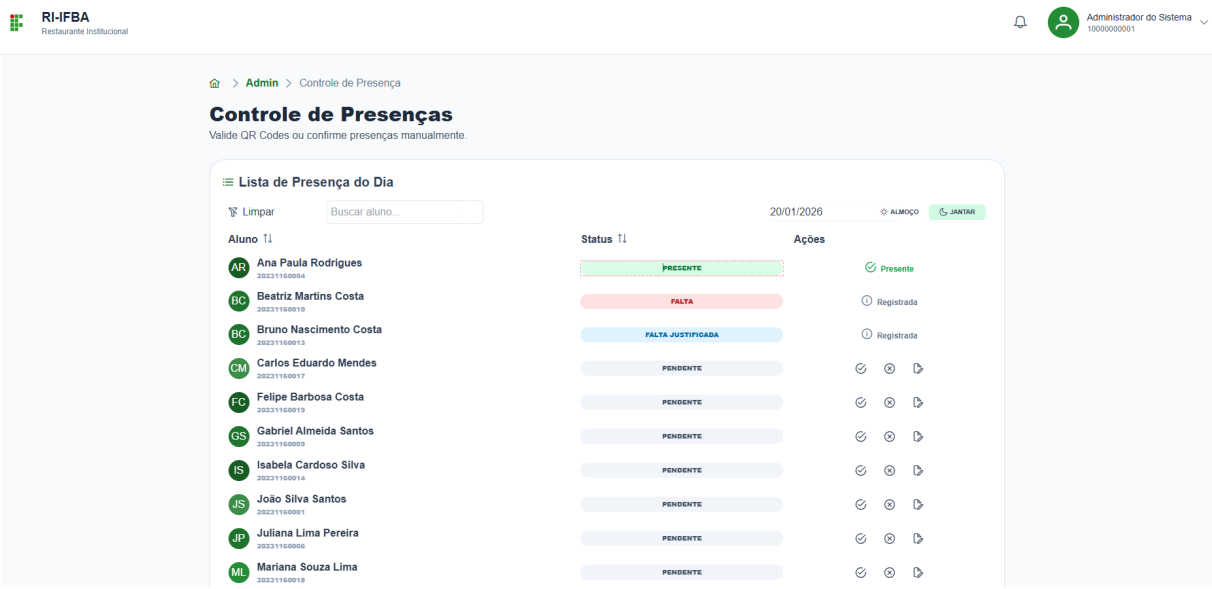
Funcionalidade utilizada na entrada do refeitório durante os horários de refeição.

1. Clique em *"Buscar Manualmente"*.
2. Informe matrícula ou nome do estudante.
3. Selecione o estudante na lista e confirme a presença.

Lista do Dia Visualização em tempo real dos bolsistas:

- **Esperados:** Bolsistas que ainda não compareceram.
- **Presentes:** Bolsistas que já realizaram a refeição.
- **Faltas:** Bolsistas ausentes (com indicador de justificativa pendente).

Figura 26 – Painel de controle de presenças.

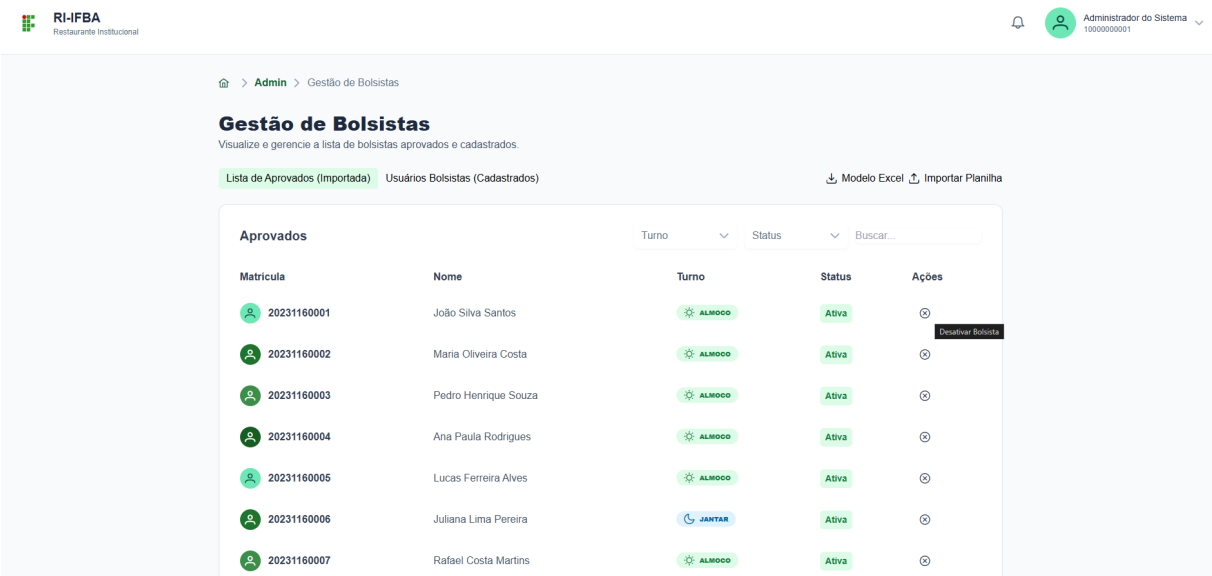


Fonte: Elaboração própria.

7.7.3 Gestão de Bolsistas

- **Importação de lista:** Carregamento da lista oficial de bolsistas aprovados.
- **Ativação/Desativação:** Controle manual do status do benefício.

Figura 27 – Lista de bolsistas com indicadores de alerta.



Fonte: Elaboração própria.

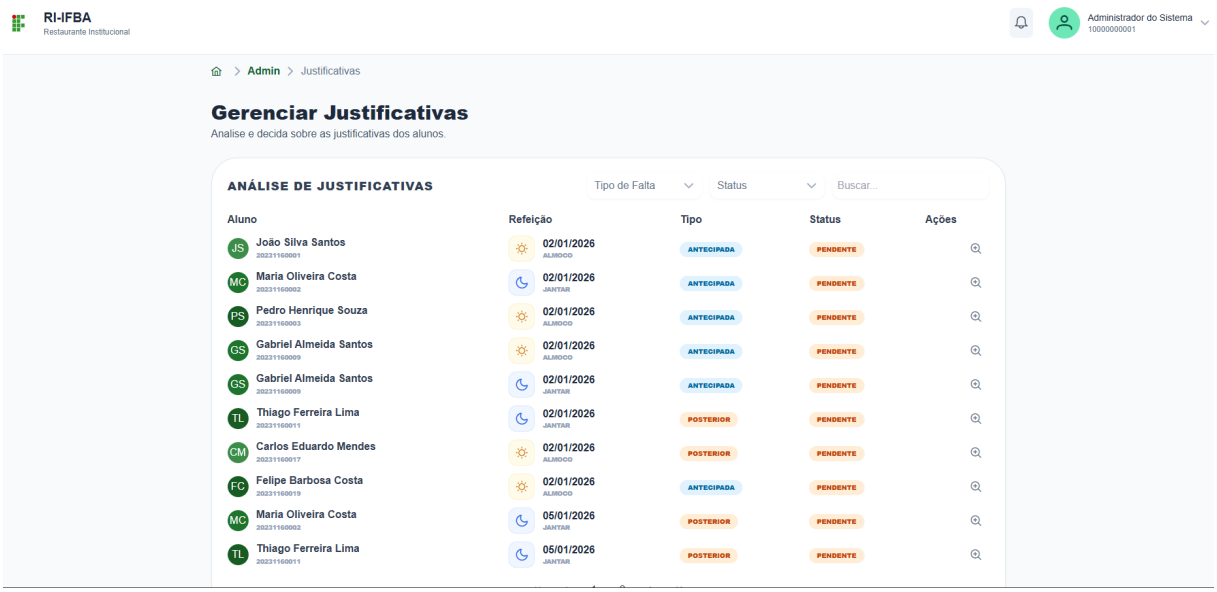
7.7.4 Validação de Justificativas

O administrador analisa e decide sobre as justificativas enviadas pelos bolsistas:

Procedimento de Análise

1. Acesse *"Justificativas Pendentes"*.
2. Visualize detalhes da solicitação e documento anexado.
3. Decida por:
    - **Aprovar:** A falta é considerada justificada.
    - **Rejeitar:** A falta permanece como injustificada.
4. Registre observações se necessário.
5. O estudante é notificado automaticamente sobre a decisão.

Figura 28 – Interface de gerenciamento de justificativas.



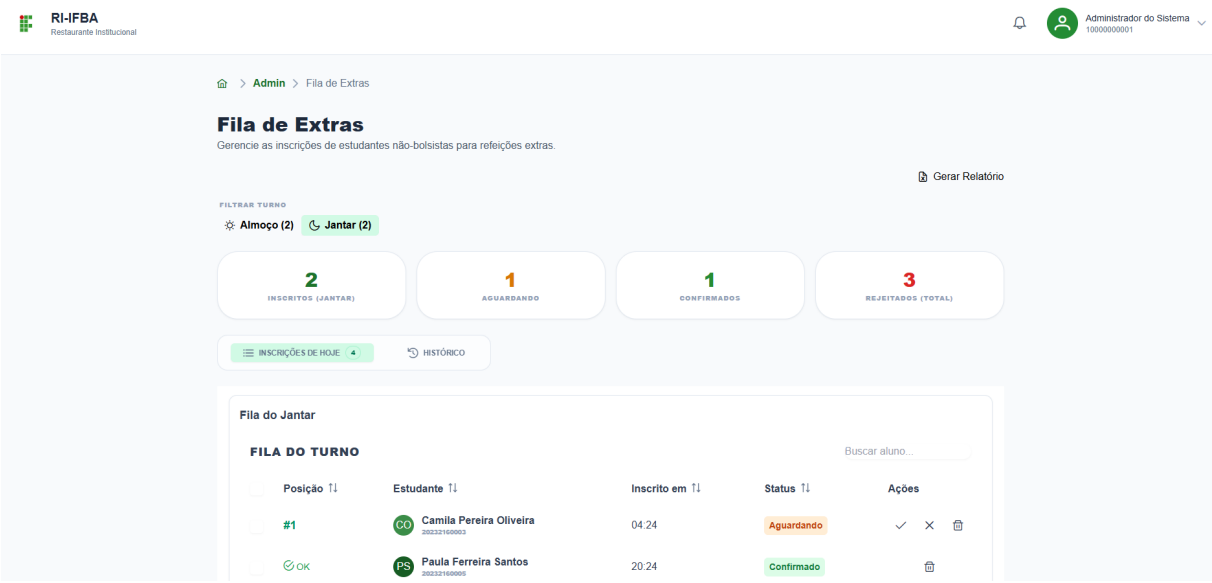
Fonte: Elaboração própria.

7.7.5 Gestão da Fila de Extras

Controle das inscrições de não-bolsistas na fila de vagas remanescentes:

- **Visualização:** Lista de inscritos por turno.
- **Aprovação/Reprovação:** Controle manual quando necessário.
- **Relatório:** Relatório de utilização das vagas extras.

Figura 29 – Interface de gerenciamento da fila de extras.



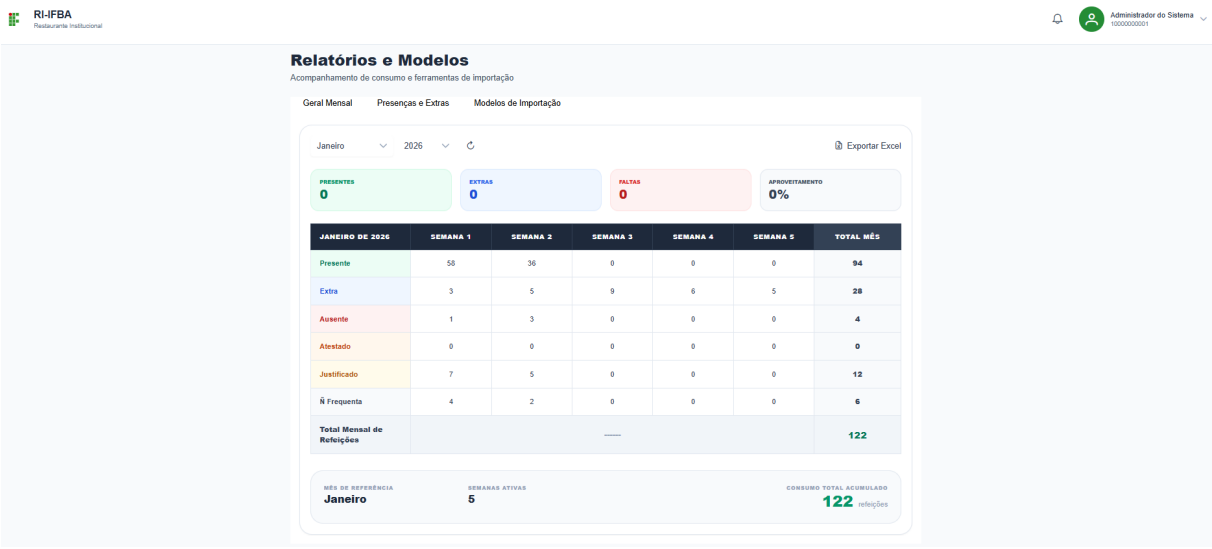
Fonte: Elaboração própria.

7.7.6 Relatórios e Auditoria

Funcionalidade exclusiva para apoio à prestação de contas e acompanhamento operacional:

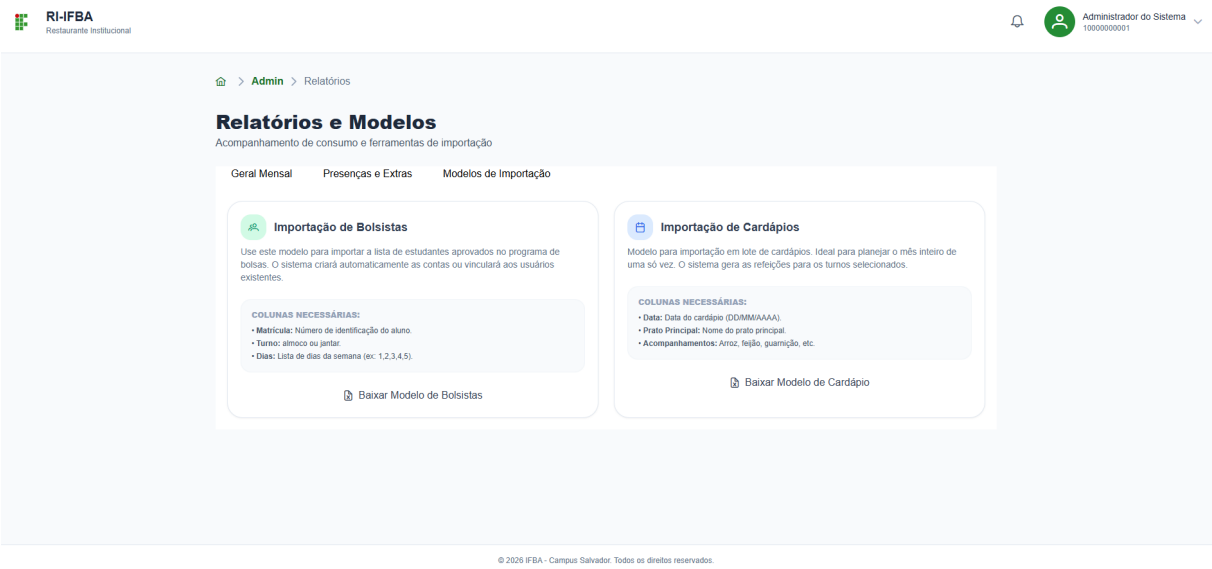
- **Relatório diário:** Consolidado de presenças e faltas.
- **Relatório mensal:** Estatísticas de utilização por período.
- **Relatório por bolsista:** Histórico individual de frequência.
- **Modelos para importação:** Modelo de referência .
- **Exportação:** Geração de arquivos Excel para análise externa.

Figura 30 – Tela de relatórios com filtros por período.



Fonte: Elaboração própria.

Figura 31 – Modelos para importação.



Fonte: Elaboração própria.

7.8 Resumo de Funcionalidades por Perfil

A Tabela 1 apresenta uma síntese das funcionalidades disponíveis para cada perfil de usuário, evidenciando a granularidade do controle de acesso implementado.

Tabela 1: Matriz com algumas funcionalidades por perfil de usuário.

Funcionalidade	**Público**	**Administrador**	**Bolsista**	**Não-Bolsista**
Visualizar tela inicial	Sim	Sim	Sim	Sim
Consultar cardápio público	Sim	Sim	Sim	Sim
Visualizar cardápio detalhado	Não	Sim	Sim	Sim
Gerenciar cardápios	Não	Sim	Não	Não
Enviar justificativas	Não	Não	Sim	Não
Validar justificativas	Não	Sim	Não	Não
Inscrever-se na fila de extras	Não	Não	Não	Sim
Gerenciar fila de extras	Não	Sim	Não	Não
Registrar presenças	Não	Sim	Não	Não
Consultar histórico pessoal	Não	Não	Sim	Sim
Gerar relatórios institucionais	Não	Sim	Não	Não
Gerenciar bolsistas	Não	Sim	Não	Não

Fonte: Elaboração própria.

7.9 Solução de possíveis Problemas

A Tabela 2 reúne possíveis problemas operacionais e suas respectivas soluções, servindo como referência rápida para usuários e administradores.

Tabela 2: Problemas frequentes e soluções.

Problema	**Causa Provável**	**Solução Recomendada**
Matrícula não reconhecida	Cadastro não encontrado na base institucional ou inconsistência de dados.	Solicitar à administração a verificação do vínculo com a lista oficial de bolsistas (quando aplicável).
Erro no upload de documento	Arquivo excede limite de tamanho (5MB) ou formato incompatível.	Comprimir a imagem ou converter para PDF antes do envio. Verificar se o arquivo não ultrapassa 5MB.
Justificativa rejeitada	Documento incompleto, ilegível ou fora do prazo.	Reenviar com documento válido dentro do prazo estabelecido.
Fila de extras indisponível	Período de inscrição encerrado ou não há vagas remanescentes.	Tentar inscrição no horário correto ou em outro turno/-dia.
Não consigo visualizar cardápio sem login	Problema de cache no navegador ou conexão instável.	Limpar cache do navegador ou tentar acesso em modo anônimo/incógnito.

Fonte: Elaboração própria.

7.10 Considerações Finais do Manual

O sistema RI-IFBA foi desenvolvido com foco na usabilidade, eficiência operacional e aderente aos requistos reais. A estrutura garante que cada usuário interaja apenas com as funcionalidades relevantes ao seu perfil. As telas foram projetadas buscando ser responsiva e intuitiva, buscando trazer uma experiência adequada em dispositivos móveis e desktops.

Agradecimentos

Agradeço primeiramente a Deus e a Nossa Senhora de Fátima pela força e perseverança concedidas ao longo desta jornada acadêmica.

À minha mãe, Irenice Araujo Pereira, e à minha companheira, Raquel Souza dos Santos, pela compreensão, apoio incondicional e incentivo constante, especialmente nos momentos mais desafiadores.

À Prof.ł Flávia Maristela Santos Nascimento, minha orientadora, pela paciência,

dedicação e valiosas contribuições que tornaram possível a realização deste trabalho.

Referências

BRASIL. **Constituição da República Federativa do Brasil de 1988**. Disponível em: <https://www.planalto.gov.br/ccivil_03/constituicao/constituicao.htm>. Acesso em: 10 dez. 2025.

BRASIL. **Decreto nž 7.234, de 19 de julho de 2010**. Dispõe sobre o Programa Nacional de Assistência Estudantil – PNAES. Disponível em: <https://www.planalto.gov.br/ccivil_03/_ato2007-2010/2010/decreto/d7234.htm>. Acesso em: 10 dez. 2025.

BRASIL. **Lei nž 11.947, de 16 de junho de 2009**. Dispõe sobre o atendimento da alimentação escolar e do Programa Dinheiro Direto na Escola aos alunos da educação básica. Disponível em: <https://www.planalto.gov.br/ccivil_03/_ato2007-2010/2009/lei/l11947.htm>. Acesso em: 04 dez. 2025.

BRASIL. **Lei nž 14.914, de 3 de julho de 2024**. Institui a Política Nacional de Assistência Estudantil (PNAES). Disponível em: <https://www.planalto.gov.br/ccivil_03/_ato2023-2026/2024/lei/L14914.htm>. Acesso em: 04 dez. 2025.

DOCKER INC. **Documentação oficial do Docker**. Disponível em: <https://docs.docker.com/>. Acesso em: 15 dez. 2025.

INSTITUTO FEDERAL DA BAHIA. **Resolução nž 25, de 14 de julho de 2016**. Aprova a Política de Assistência Estudantil do IFBA. Disponível em: <https://portal.ifba.edu.br/dpaae/a-dpaae/dpaae>. Acesso em: 10 dez. 2025.

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA BAHIA. **Resolução nž 25, de 23 de maio de 2016**. Aprova as Diretrizes e Normas da Política de Assistência Estudantil do IFBA. 2016.

LARAVEL. **Documentação Laravel (versão 12.x)**. Disponível em: <https://laravel.com/docs/12.x>. Acesso em: 04 dez. 2025.

POSTGRESQL GLOBAL DEVELOPMENT GROUP. **Documentação oficial do PostgreSQL**. Disponível em: <https://www.postgresql.org/docs/>. Acesso em: 15 dez. 2025.

SOUSA, Laiana Paula Severo de; SOARES, Maria Elias. Políticas de permanência estudantil no ensino superior: a importância do Programa Restaurante Universitário. **SciELO Preprints**, 2024. Disponível em: <https://doi.org/10.1590/SciELOPreprints.10208>. Acesso em: 16 dez. 2025.

UNIVERSIDADE DE SÃO PAULO. **Aplicativo Cardápio – PRIP**. Disponível em: <https://prip.usp.br/servicos/aplicativo-cardapio/>. Acesso em: 16 dez. 2025.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE. **Restaurante Universitário (RU): sobre**. Disponível em: <https://ru.ufrn.br/sobre/>. Acesso em: 16 dez. 2025.

VUE.JS. **Introdução ao Vue.js**. Disponível em: <https://vuejs.org/guide/introduction.html>. Acesso em: 05 dez. 2025.