UNIVERSIDADE FEDERAL RURAL DO RIO DE JANEIRO
INSTITUTO MULTIDISCIPLINAR

LENNON FERREIRA MACHADO
PEDRO NUNES CARDOSO

Capiwallet: Um Sistema Web de Carteira Digital para o Restaurante Universitário da UFRRJ

Profa. Natália Chaves Lessa, D.Sc.
Orientadora

Nova Iguaçu, Julho de 2025

Capiwallet: Um Sistema Web de Carteira Digital para o Restaurante Universitário da UFRRJ

Lennon Ferreira Machado

Pedro Nunes Cardoso

Projeto Final de Curso submetido ao Departamento de Ciência da Computação do Instituto Multidisciplinar da Universidade Federal Rural do Rio de Janeiro como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Apresentado por:

_____

Lennon Ferreira Machado

_____

Pedro Nunes Cardoso

Aprovado por:

_____

Profa. Natália Chaves Lessa, D.Sc.

_____

Profa. Juliana Mendes Nascente e Silva Zamith, D.Sc.

_____

Prof. Marcel William Rocha da Silva, D.Sc.

NOVA IGUAÇU, RJ - BRASIL

Julho de 2025

DOCUMENTOS COMPROBATÓRIOS Nº 15013/2025 - CoordCGCC (12.28.01.00.00.98)

(Nº do Protocolo: NÃO PROTOCOLADO)

(Assinado digitalmente em 09/07/2025 14:47)
JULIANA MENDES NASCENTE E SILVA ZAMITH
COORDENADOR CURS/POS-GRADUACAO - TITULAR
CoordCGCC (12.28.01.00.00.98)
Matrícula: ###731#0

(Assinado digitalmente em 08/07/2025 10:14)
MARCEL WILLIAM ROCHA DA SILVA
PROFESSOR DO MAGISTERIO SUPERIOR
PPGIHD (11.39.00.16)
Matrícula: ###807#6

(Assinado digitalmente em 08/07/2025 10:48)
NATALIA CHAVES LESSA
PROFESSOR DO MAGISTERIO SUPERIOR
DeptCC/IM (12.28.01.00.00.83)
Matrícula: ###435#4

(Assinado digitalmente em 08/07/2025 17:21)
LENNON FERREIRA MACHADO
DISCENTE
Matrícula: 2019######0

(Assinado digitalmente em 08/07/2025 17:13)
PEDRO NUNES CARDOSO
DISCENTE
Matrícula: 2019######2

Visualize o documento original em https://sipac.ufrrj.br/documentos/ informando seu número: ***15013****, ano:* ***2025****, tipo:* ***DOCUMENTOS COMPROBATÓRIOS****, data de emissão:* ***07/07/2025*** *e o código de verificação:* ***21fce3f056***

Agradecimentos

Lennon Ferreira Machado

Primeiramente, agradeço a Deus por me conceder força, saúde, sabedoria e resiliência ao longo desta jornada. Sem Sua presença em minha vida, nada disso teria sido possível.

Agradeço ao meu amigo Pedro, pela amizade e parceria construídas desde o ensino fundamental, passando pelo ensino médio e agora na faculdade, compartilhando comigo mais uma conquista ao realizarmos juntos este Trabalho de Conclusão de Curso.

Expresso também minha sincera gratidão à Professora Natália, pela orientação, apoio e dedicação ao longo do curso. Sua contribuição foi fundamental para o meu desenvolvimento acadêmico e para a realização deste trabalho.

Agradeço ainda aos colegas da faculdade que fizeram parte dessa trajetória e contribuíram, de alguma forma, para o meu crescimento.

E, por fim, deixo meu agradecimento aos meus amigos, pelo apoio, incentivo e por estarem ao meu lado, ajudando-me direta e indiretamente ao longo dessa caminhada.

Pedro Nunes Cardoso

Agradeço, em primeiro lugar, à minha mãe, por ter me incentivado a retornar à faculdade mesmo diante das dificuldades que enfrentamos em casa. Sua força e apoio foram fundamentais para que eu recomeçasse essa jornada.

Ao meu pai, que me apoiou com entusiasmo desde o momento em que entrei na universidade. Sua motivação foi essencial no início dessa caminhada. Infelizmente, ele partiu no mesmo ano em que iniciei o curso e não poderá ver minha formatura, mas sua presença e inspiração seguirão comigo para sempre.

Ao meu padrasto, agradeço por todo o incentivo e apoio diário, sempre acreditando na minha capacidade e me motivando a concluir essa etapa da minha vida.

Agradeço também à Universidade Federal Rural do Rio de Janeiro e a todos os professores do curso de Ciência da Computação, que contribuíram significativamente com seus ensinamentos ao longo da minha formação acadêmica.

Em especial, deixo minha sincera gratidão à Prof.ª Natália, por toda orientação, paciência e apoio durante o desenvolvimento deste trabalho. Sua dedicação foi essencial para que este projeto se concretizasse.

Tenho uma gratidão imensa pelo meu amigo e colega de dupla, Lennon Ferreira, com quem compartilho uma amizade que começou ainda no ensino fundamental. Ingressamos juntos na mesma universidade, no mesmo curso, na mesma turma — e é uma alegria enorme termos chegado juntos até aqui, agora como parceiros no nosso Trabalho de Conclusão de Curso.

Também agradeço ao Bruno Rodrigues, amigo de longa data, que gentilmente criou a identidade visual do nosso sistema com a capivarinha, dando um toque único e pessoal ao projeto.

Por fim, agradeço a todos os meus amigos. Em especial, àqueles que compartilham comigo o dia a dia na nossa comunidade no Discord — mesmo com a distância física, nunca deixamos de nos comunicar, fortalecer nossa amizade e tornar a rotina mais leve e divertida.

RESUMO

Capiwallet: Um Sistema Web de Carteira Digital para o Restaurante Universitário da UFRRJ

Lennon Ferreira Machado e Pedro Nunes Cardoso

Julho/2025

Orientadora: Natália Chaves Lessa, D.Sc.

O desenvolvimento de *software* tem ganhado cada vez mais destaque globalmente, impulsionado pela crescente demanda por soluções digitais em diversas áreas. Nesse cenário, a arquitetura de *software* desempenha um papel crucial, sendo a base essencial para a criação de sistemas robustos, escaláveis e eficientes. Investir em uma arquitetura bem estruturada é fundamental para garantir a qualidade e a longevidade de qualquer aplicação. Este trabalho tem como objetivo propor e desenvolver uma solução digital para otimizar o processo de compra e utilização de *tickets* no restaurante universitário da Universidade Federal Rural do Rio de Janeiro (UFRRJ), que atualmente é realizado de forma manual e por meio de *tickets* em papel. Inicialmente, foi realizado um estudo sobre arquiteturas de *software*, com foco nos modelos monolítico e de microsserviços. A partir desse conhecimento, foram analisadas as características, vantagens e desvantagens de cada abordagem. Com base nessa análise, foi desenvolvido um sistema *web*, denominado Capiwallet, que funciona como uma carteira digital para compra e utilização de *tickets*, oferecendo uma alternativa prática e segura ao modelo tradicional. A escolha da arquitetura do sistema foi fundamentada nos estudos teóricos realizados, considerando aspectos como escalabilidade, manutenção e flexibilidade. Os testes e validações da aplicação foram conduzidos por meio de um formulário baseado no modelo *Technology Acceptance Model* (TAM), e os resultados obtidos demonstraram uma recepção positiva quanto à utilidade e facilidade de uso do sistema, evidenciando sua viabilidade e potencial para aplicação prática no ambiente universitário.

ABSTRACT

Capiwallet: Um Sistema Web de Carteira Digital para o Restaurante Universitário da UFRRJ

Lennon Ferreira Machado and Pedro Nunes Cardoso

Julho/2025

Advisor: Natália Chaves Lessa, D.Sc.

Software development has gained increasing global prominence, driven by the growing demand for digital solutions across various sectors. In this context, software architecture plays a crucial role, serving as the foundational base for building robust, scalable, and efficient systems. Investing in a well-structured architecture is essential to ensure the quality and longevity of any application. This work aims to propose and develop a digital solution to optimize the process of purchasing and using meal tickets at the university restaurant of the Federal Rural University of Rio de Janeiro (UFRRJ), which is currently done manually through paper tickets. An initial study was conducted on software architectures, focusing on monolithic and microservices models. Based on this knowledge, the characteristics, advantages, and disadvantages of each approach were analyzed. From this analysis, a web system named Capiwallet was developed, functioning as a digital wallet for purchasing and using meal tickets, offering a practical and secure alternative to the traditional model. The choice of system architecture was based on the theoretical studies, considering aspects such as scalability, maintainability, and flexibility. The system was tested and validated using a questionnaire based on the Technology Acceptance Model (TAM), and the results showed a positive reception regarding the system's usefulness and ease of use, demonstrating its viability and potential for practical application in the university environment.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

HTTP *Hypertext Transfer Protocol*

API *Application Programming Interface*

HTML *Hypertext Markup Language*

CSS *Cascading Style Sheets*

TAM *Technology Acceptance Model*

SIGAA Sistema Integrado de Gestão de Atividades Acadêmicas

UFRRJ Universidade Federal Rural do Rio de Janeiro

Sumário

Capítulo 1

Introdução

A tecnologia da informação tem desempenhado um papel fundamental na modernização dos serviços voltados ao cotidiano acadêmico, proporcionando soluções digitais que facilitam o acesso a recursos e otimizam diversos processos institucionais. Ao integrar ferramentas tecnológicas ao ambiente universitário, busca-se promover maior eficiência, agilidade e comodidade tanto para os alunos quanto para a gestão administrativa.

Nas universidades públicas brasileiras, é possível observar que alguns processos administrativos ainda utilizam práticas manuais, o que pode representar um desafio adicional diante da crescente demanda por digitalização e automação de serviços. Um exemplo pontual é o sistema de aquisição e utilização de *tickets* no restaurante universitário da Universidade Federal Rural do Rio de Janeiro (UFRRJ), que atualmente é realizado por meio de *tickets* físicos impressos em papel. Esse modelo exige a presença física dos alunos para a compra e o uso das refeições, envolvendo etapas operacionais que podem ser otimizadas com o uso de tecnologias digitais.

Neste cenário, o desenvolvimento de sistemas de informação pode ser amplamente beneficiado por decisões estruturadas que orientem seu processo de construção. Uma das decisões mais importantes nessa etapa é a escolha da arquitetura de *software*, pois ela define a organização fundamental do sistema, seus componentes e as interações entre eles. De acordo com Bass, Clements e Kazman (2003), uma arquitetura

bem definida contribui para a qualidade do *software* e facilita sua manutenção, escalabilidade e evolução ao longo do tempo. Além disso, a arquitetura influencia diretamente atributos de qualidade como desempenho, segurança, reutilização e modularidade, sendo, portanto, um elemento estratégico para o sucesso de projetos ao viabilizar soluções robustas e alinhadas às necessidades dos usuários.

1.1 Objetivo

Diante desse cenário, este trabalho propõe o desenvolvimento de uma aplicação *web* chamada Capiwallet, que funciona como uma carteira digital voltada para a compra e utilização de *tickets* no restaurante universitário. A solução visa não apenas digitalizar o processo, mas também oferecer uma experiência mais fluida aos usuários e permitir à administração maior controle sobre os dados e operações.

Para o desenvolvimento desse sistema busca-se identificar qual modelo arquitetural é mais adequado para resolver o problema real, promovendo melhorias na experiência dos alunos, maior controle no processo de distribuição de *tickets* e maior eficiência operacional. O estudo também busca compreender como diferentes arquiteturas impactam o desempenho, escalabilidade e manutenção do sistema.

1.2 Organização do Trabalho

O trabalho está organizado em cinco capítulos, além deste introdutório, conforme descrito a seguir:

a) Capítulo 2: corresponde à fundamentação teórica, onde são abordados os conceitos técnicos e arquiteturais utilizados como base para o desenvolvimento do sistema. Este capítulo inclui uma discussão sobre as arquiteturas monolítica e de microsserviços, bem como os critérios utilizados para a escolha da abordagem mais adequada ao contexto do problema;

b) Capítulo 3: descreve a proposta do sistema, com foco na resolução do problema identificado. São apresentadas as etapas de Engenharia de *Software*, incluindo

os requisitos do sistema, modelagens e diagramas que sustentam a concepção da solução;

c) Capítulo 4: detalha o sistema implementado e justifica as tecnologias escolhidas para sua construção. Além disso, apresenta as principais telas da aplicação desenvolvida, evidenciando seu funcionamento;

d) Capítulo 5: aborda a avaliação do sistema, realizada por meio da aplicação de um formulário baseado no modelo *Technology Acceptance Model* (TAM). Os dados coletados são analisados estatisticamente, com o apoio de gráficos que ilustram o *feedback* dos usuários que testaram a aplicação;

e) Capítulo 6: apresenta a conclusão do trabalho, destacando os resultados obtidos, as limitações encontradas durante o desenvolvimento e sugestões para trabalhos futuros.

Capítulo 2

Arquitetura de Software

Softwares são construídos com base em um tipo de arquitetura. Entretanto, é importante destacar que não existe uma definição exata para esse conceito. Apesar disso, especialistas na área conseguem estabelecer uma descrição semelhante desta idealização. Levando isso em consideração, pode-se destacar as seguintes definições:

> "A arquitetura é definida pela prática recomendada como a organização fundamental de um sistema, incorporada em seus componentes, suas relações entre si e com o ambiente, e os princípios que regem seu projeto e evolução." (COMMITTEE et al., 2000).

> "A arquitetura de *software* de um programa ou sistema de computação é a estrutura ou estruturas do sistema, que compreende elementos de *software*, as propriedades visíveis externamente desses elementos e os relacionamentos entre eles." (BASS; CLEMENTS; KAZMAN, 2003).

Utilizando como base essas definições, é possível avaliar que a arquitetura de *software* está diretamente relacionada à organização e estrutura de um sistema, onde seus componentes possuem relacionamentos entre si. Nesse sentido, a escolha de uma arquitetura influencia totalmente na qualidade do *software*. A seleção de um padrão arquitetural pode acarretar tanto benefícios quanto desafios, uma vez que tais estruturas buscam otimizar o desempenho, garantir a segurança, promover a usabilidade e viabilizar a manutenibilidade do *software* (BASS; CLEMENTS; KAZMAN, 2003).

Bass, Clements e Kazman (2003) definem alguns atributos de qualidade, entre os quais destacam-se a manutenibilidade, segurança e usabilidade, como sendo essenciais para uma arquitetura.

A manutenibilidade assume um papel de extrema importância. Quanto maior a coesão e menor o acoplamento na arquitetura, mais facilmente e economicamente as tarefas de manutenção podem ser executadas. A redução dos custos e dos impactos negativos se traduz em uma vantagem. Além disso, a manutenção eficaz da arquitetura é uma necessidade fundamental, permitindo ajustes e melhorias ao sistema (BASS; CLEMENTS; KAZMAN, 2003).

A segurança do *software* refere-se à habilidade do sistema de evitar a entrada em estados que possam causar danos ou colocar em risco vidas das partes envolvidas no ambiente do *software*. Além disso, busca-se a capacidade de recuperar e limitar os danos quando o *software* entra em estados adversos (BASS; CLEMENTS; KAZMAN, 2003).

A usabilidade concentra-se na facilidade com que os usuários podem realizar tarefas e no suporte oferecido pelo sistema, englobando estratégias para tornar a interação com o sistema mais intuitiva, eficiente e satisfatória para o usuário, abordando desde a fase de aprendizado até a confiança contínua nas ações realizadas (BASS; CLEMENTS; KAZMAN, 2003).

Existem diversos tipos de arquitetura: Camadas, Cliente-Servidor, Orientada a Serviços, Baseada em Eventos, Monolíticos e Microsserviços dentre outros (VALENTE, 2020). Nesse estudo, serão abordados dois tipos: Monolíticas e de Microsserviços, tendo em vista que as empresas de destaque internacional têm optado por implementar estas arquiteturas em suas operações.

2.1 Monolítico

Em uma arquitetura monolítica, a aplicação é composta por um único componente que contém todo o código e dados necessários para sua execução. Todas as operações da aplicação são realizadas dentro desse componente, que é executado como um

único processo (FOWLER, 2014).

Os aplicativos monolíticos se caracterizam pela integração de diversos componentes, formando uma única unidade de *software*. Essa abordagem unificada incorpora todas as funcionalidades e serviços, destacando-se pela interconexão dos elementos (AWATI, 2022).

Aplicações monolíticas são uma boa opção para projetos iniciais, pois não exigem um planejamento detalhado. É possível iniciar a aplicação com um conjunto básico de funcionalidades e, à medida que as necessidades forem aumentando, adicionar novos módulos de código. No entanto, aplicações monolíticas podem se tornar complexas com o tempo, o que pode dificultar a atualização ou alteração do código (AWS, 2023).

2.2 Microsserviços

A arquitetura de microsserviços é caracterizada pela implantação independente de serviços em um *software*, modelado sobre um domínio de negócio, estabelecendo comunicação entre eles por meio de redes (NEWMAN, 2020). Em outras palavras, essa abordagem arquitetônica envolve a criação de um aplicativo composto por diversos serviços distintos, cada um operando como um processo autônomo e interagindo através de solicitações *Hypertext Transfer Protocol* (HTTP) (FOWLER, 2014).

A implantação independente de serviços oferece flexibilidade na modificação ou criação de novos serviços. No entanto, essa flexibilidade só é alcançada quando os serviços possuem baixo acoplamento, ou seja, são independentes e não fortemente interligados. Isso facilita a manutenção e evolução do sistema, tornando-o mais ágil e adaptável às mudanças (NEWMAN, 2020). Além dessa característica, os microsserviços possuem outras características que são brevemente descritas a seguir (FOWLER, 2014):

a) Componentização via serviços: um componente é visto como uma unidade de *software* que é substituível e atualizável de forma independente. No cenário

de microsserviços, os componentes podem ser bibliotecas ou serviços. As bibliotecas são componentes ligados ao programa e acionados por meio de chamadas de função na memória, enquanto os serviços são componentes independentes que se comunicam por meio de solicitações. O uso de serviços como componentes nas arquiteturas de microsserviços oferece alguns benefícios, o principal é a implantação independente dos serviços, que permite atualizações específicas sem a necessidade de reimplantar todo o aplicativo.

b) Governança descentralizada: como na arquitetura de microsserviços os serviços são implantados de forma independente e desacoplada, a governança se distancia da padronização comum para o projeto e desenvolvimento dos serviços. Ao decompor um sistema monolítico em serviços independentes, cada microsserviço ganha autonomia, permitindo ser construído com a tecnologia mais adequada para resolver um problema específico. Essa abordagem flexível dá liberdade às equipes de desenvolvimento para escolherem as tecnologias mais apropriadas para cada serviço, em vez de serem restritas a uma única tecnologia para todo o sistema.

c) Gestão descentralizada de dados: na estrutura de microsserviços, a descentralização do armazenamento é um princípio fundamental. Isso significa que cada serviço é responsável por seu próprio banco de dados, permitindo o uso de diferentes bancos de dados para resolver problemas específicos. Essa abordagem descentralizada proporciona maior independência e flexibilidade aos serviços, possibilitando a escolha de tecnologias de armazenamento mais adequadas para as necessidades de cada serviço.

d) *Design* para falhas: a incorporação de microsserviços em aplicações destaca a importância de abordar possíveis falhas nos serviços, considerando que um serviço pode enfrentar períodos de indisponibilidade. Para enfrentar esse desafio, é essencial implementar um monitoramento em tempo real, visando identificar e corrigir prontamente tais falhas.

e) *Design* evolucionário: a arquitetura de microsserviços, ao favorecer a decomposição em serviços, possibilita mudanças rápidas na aplicação, priorizando a substituição independente e a capacidade de atualização. Ao colocar com-

ponentes em serviços, há a vantagem de um planejamento de liberação mais detalhado, permitindo alterações específicas sem a necessidade de uma implantação completa. Isso assegura que os serviços sejam projetados para suportar alterações sem causar impactos negativos em todo o sistema.

Considerando estas principais características de microsserviços, Richards e Ford (2020) e Newman (2015) destacam algumas vantagens, tais como:

a) Estabilidade: a estabilidade nos microsserviços é efetivamente mantida pela governança descentralizada, que favorece o desacoplamento entre serviços, e pelo *design* voltado para falhas, que garante o isolamento de eventuais falhas. Essa combinação assegura que falhas em um serviço não afetem os outros, proporcionando assim uma melhor estabilidade;

b) Tolerância a falhas: o *design* para falhas contribui para que os microsserviços sejam projetados para lidar com falhas de maneira mais eficaz. Se um serviço falhar, outros podem continuar funcionando sem interrupção;

c) Escalabilidade: a escalabilidade é diretamente beneficiada pela abordagem de componentização via serviços. Dada a independência dos serviços, é possível projetá-los de maneira que possam ser replicados e escalados de forma autônoma, resultando em um aumento eficiente na capacidade do sistema;

d) Evolucionabilidade: o *design* evolucionário contribui para a substituição e atualização de serviços rapidamente, permitindo alterações sem impacto no sistema;

e) Heterogeneidade: a governança descentralizada permite uma diversidade tecnológica entre os microsserviços, proporcionando flexibilidade para selecionar as ferramentas mais apropriadas a cada necessidade específica.

Por outro lado, Fowler (2015) destacam algumas desvantagens no uso de microsserviços, tais como:

a) Complexidade operacional: a implementação de microsserviços pode trazer consigo uma complexidade substancial para as operações e a infraestrutura, principalmente devido à administração de múltiplas instâncias de serviços,

à distribuição de implantações e à necessidade de coordenação eficaz. Esses desafios, embora inerentes ao modelo de microsserviços, requerem uma abordagem estratégica e eficiente para garantir uma transição e operação bem-sucedidas.

b) Distribuição: embora os microsserviços busquem aprimorar a modularidade através de um sistema distribuído, essa abordagem também introduz complexidades consideráveis. Os desafios de desempenho são evidenciados, indicando que chamadas remotas podem ser lentas, especialmente em um cenário onde várias chamadas são encadeadas, resultando em latência significativa.

c) Consistência de eventual: a lógica de negócios pode tomar decisões com base em informações inconsistentes. Os microsserviços são apontados como potenciais causadores desses problemas, devido à ênfase na descentralização do gerenciamento de dados, demandando maior conscientização dos desenvolvedores em relação a questões de consistência.

d) Testabilidade: a gestão de ambientes de teste com diferentes serviços evoluindo em ritmos distintos e lançados frequentemente, apresenta desafios consideráveis. A complexidade é ampliada pela assincronicidade e cargas de mensagens dinâmicas, tornando difícil reproduzir ambientes de maneira consistente para testes manuais ou automatizados.

2.3 Comparação entre Monolítico e Microsserviços

Com o avanço dos sistemas, as arquiteturas monolítica e de microsserviços emergem como as principais opções para o desenvolvimento de *software*. Considerando as características previamente discutidas, faz-se necessário uma análise comparativa entre essas duas abordagens.

As aplicações monolíticas apresentam acoplamento em um único módulo, caracterizando-se por uma base de código unificada. Por outro lado as aplicações de microsserviços, cada componente opera para executar uma única funcionalidade ou lógica de negócios. Ao contrário da troca de dados dentro de uma base de código única, os microssservi-

ços se comunica por meio de uma *Application Programming Interface* (API) (AWS, 2023).

A implantação de sistemas monolíticos é mais simples do que a de microsserviços devido à consolidação da base de código da aplicação e de suas dependências em um único ambiente. Por outro lado, a implantação de aplicações baseadas em microsserviços é mais complexa, uma vez que cada microsserviço constitui um pacote de *software* implantável de forma independente (AWS, 2023).

Pequenas alterações em uma aplicação monolítica têm impacto significativo, pois a codificação fortemente acoplada gera interdependência entre várias funções do *software*. Introduzir modificações demanda retestar e reimplantar o sistema inteiro no servidor, aumentando a complexidade do processo (AWS, 2023).

Já o microsserviços proporcionam flexibilidade ao permitir modificações mais fáceis na aplicação. Ao contrário da abordagem monolítica, os desenvolvedores podem fazer alterações específicas em funções, evitando a necessidade de modificar todos os serviços. A implantação independente de serviços facilita a adoção de práticas de implantação contínua, possibilitando ajustes frequentes sem impactar a estabilidade do sistema (AWS, 2023).

A arquitetura monolítica enfrenta desafios significativos de escalabilidade ao expandir. Com todas as funcionalidades integradas em uma única base de código, a necessidade de escalonamento abrange toda a aplicação quando ocorrem mudanças nos requisitos (AWS, 2023).

A arquitetura de microsserviços, ao contrário, sustenta sistemas distribuídos, proporcionando a cada componente de *software* seus próprios recursos computacionais. Isso permite a escalabilidade independente, com alocação flexível de recursos conforme as necessidades específicas de cada serviço (AWS, 2023).

2.4 Trabalhos Relacionados

Nos últimos anos, tem havido um crescente interesse e pesquisa na arquitetura de microsserviços. Nesse cenário, Mendes (2021) realizou uma pesquisa exploratória, a partir de um levantamento teórico baseado em autores importantes na área de arquitetura de *software*, como Martin Fowler, Sam Newman e Mark Richards. Nesse levantamento, foi realizado o fichamento e agrupamento de pontos de vistas de tais autores, sendo juntamente comparado a quatro casos práticos de empresas que realizaram migrações entre as arquiteturas de microsserviços e monolíticas, levando em consideração a problemática a ser resolvida, os recursos necessários, características arquiteturais, a manutenibilidade e a evolucionabilidade das empresas.

Mendes (2021) analisou empresas que adotaram arquiteturas tanto monolíticas quanto de microsserviços, avaliando os pontos positivos e negativos. Foram apresentados os casos de quatro empresas, a saber:

a) KN Login que possuia um monolítico legado, com alta heterogeneidade de tecnologias, alta complexidade e baixa confiabilidade, se tornando um sistema instável e difícil de manter. Visto estes pontos negativos, foi transformado em uma série de sistemas autocontidos, com o intuito de caminhar em direção a uma arquitetura de microsserviços. Mendes (2021) concluiu que, diante da combinação de um contexto complexo e a heterogeneidade de tecnologias, a arquitetura monolítica não satisfazia as necessidades da empresa.

b) Otto, que inicialmente possuía um sistema monolítico simples, enfrentou problemas relacionados à escalabilidade e complexidade à medida que sua base de código evoluía. Para lidar com esses desafios, foi decidido realizar uma reimplementação do zero, migrando de um sistema monolítico para uma arquitetura de sistemas autocontidos que posteriormente foi evoluída para microsserviços. Ao identificar os problemas de escalabilidade e complexidade, ficou claro que os sistemas monolíticos têm dificuldades em lidar com a evolução do sistema, enquanto os sistemas baseados em microsserviços são mais adequados para esse cenário. Contudo, essa transição também introduziu desafios adicionais, como a necessidade de automatizar todos os processos. Mendes

(2021) ressalta a importância de considerar cuidadosamente as compensações entre os diferentes estilos arquiteturais e adaptar a abordagem arquitetural de acordo com as necessidades específicas do projeto e do contexto empresarial.

c) Segment, que transitou de um monolítico para uma arquitetura de microsserviços, seguido do retorno para a arquitetura monolítica após as várias dificuldades encontradas. Inicialmente, a empresa contava um sistema monolítico dado que possuíam uma equipe pequena e estavam no início do projeto. Ao enfrentar problemas referentes à tolerância a falhas, na arquitetura adotada, a empresa decidiu migrar para uma arquitetura de microsserviços. No entanto, deparou-se com processos operacionais não automatizados, resultando na incapacidade de gerenciar a crescente demanda. Isso, juntamente com o aumento no número de serviços em funcionamento e sobrecarga da equipe, a arquitetura se tornou impraticável de gerenciar. Sendo assim, optaram por voltar para o monolítico, tendo em vista que é uma arquitetura na qual eram conhecidos os seus pontos negativos e positivos. Mendes (2021) afirma que a escolha de retornar ao modelo monolítico, embasada em uma compreensão mais consciente e estratégias para enfrentar adversidades, foi crucial para a construção de um sistema que proporcionasse escalabilidade, tolerância a falhas e produtividade. Além disso, Mendes (2021) salienta que é necessário que a equipe se empenhe para conseguir manter essas características a medida que sua base de código cresce.

d) RuaDois, um empresa inicialmente voltada para facilitar o trabalho das imobiliárias, adotou uma arquitetura de microsserviços visando maior estabilidade, independência entre sistemas e tolerância a falhas. No entanto, enfrentando desafios com um time pequeno de apenas 4 membros e custos de manutenção elevados, a opção pela arquitetura de microsserviços se mostrou inviável. Assim, a equipe optou por migrar para uma abordagem monolítica. A migração para a abordagem monolítica se mostrou bem-sucedida para a RuaDois. Após a transição, a empresa experimentou maior estabilidade no sistema e uma redução nos custos de manutenção. Nesse sentido, Mendes (2021) então destacou que, na arquitetura monolítica, o custo da aplicação seria mais

adequado ao porte da empresa e a testabilidade do sistema seria de certa forma simplificada, sem necessidade de integração entre sistemas. Apesar disso, o desenvolvimento de novas funcionalidades poderia se tornar mais complexo.

Após uma análise comparativa dos casos mencionados, aliada ao estudo teórico, Mendes (2021) concluiu que, de forma geral, os autores compartilhavam uma visão semelhante quanto à utilização das arquiteturas, e que os estudos de caso enriqueceram essa perspectiva. Nesse contexto, as arquiteturas monolíticas emergem como uma opção de menor custo, sendo recomendadas especialmente para a fase inicial de um sistema. No entanto, é importante destacar que essa arquitetura tende a apresentar desafios em termos de manutenção e evolução do sistema, sendo necessária uma exploração cuidadosa para adotar uma arquitetura mais sustentável ao longo do processo de desenvolvimento.

Por outro lado, no que diz respeito aos microsserviços, percebe-se que o modelo arquitetônico tende a ser mais sustentável. Entretanto, sua implementação demanda um investimento significativo, tanto em termos de manutenção e custo quanto de domínio sobre o problema a ser resolvido. Isso implica que nem todas as empresas conseguirão usufruir plenamente dos benefícios dessa arquitetura (MENDES, 2021).

De forma semelhante ao estudo realizado por Mendes (2021), Lucio (2017) conduziu uma análise comparativa entre as arquiteturas monolíticas e de microsserviços com foco na diferenciação do desenvolvimento de ambas. Para isso, implementou um sistema gerenciador de uma rede de cinemas, em ambas as arquiteturas. O objetivo foi investigar as vantagens e desvantagens de cada modelo, bem como suas particularidades e os cenários nos quais cada um se mostra mais eficaz.

Das análises realizadas, os critérios incluíram comparações teóricas das vantagens e desvantagens dos microsserviços. Segundo Lucio (2017), enquanto os microsserviços oferecem agilidade no desenvolvimento e permitem atualizações independentes, também introduzem complexidade na criação de sistemas distribuídos, aumentando os desafios de implantação e gerenciamento. Durante sua construção, é comum surgirem preocupações não previstas. Também foram realizadas análises que apresentaram

a estrutura de diretórios e as diferenças encontradas na organização dos projetos de forma qualitativa, além de uma verificação quantitativa das linhas de código entre ambos os projetos. Além disso, foram considerados aspectos de desempenho, avaliando a performance obtida por meio de execução exaustiva.

Lucio (2017) observou que a arquitetura de microsserviços aumentou a complexidade do projeto devido à quantidade de diretórios criados, totalizando 71, em comparação com os 17 do monolítico. Isso ocorre devido à necessidade de uma estrutura de diretórios para cada serviço, possibilitando seu desenvolvimento, teste e implantação de forma independente.

Quanto às linhas de código, a aplicação monolítica totalizou 1133, enquanto a de microsserviços teve 3618, indicando que a arquitetura de microsserviços exigiu três vezes mais esforço na escrita de código.

No teste de carga, no sistema monolítico, o tempo médio total das requisições foi de 2,659 segundos, com uma mediana de 1,1158 segundos. Já na arquitetura de microsserviços, o tempo médio total foi de 11,900 segundos, com uma mediana de 5,867 segundos.

A alta taxa de tempo de resposta para as requisições entre os serviços da arquitetura de microsserviços pode ser justificada pela utilização de requisições internas, o que resulta em um aumento significativo no tempo de comunicação entre os processos.

Lucio (2017) destaca que a arquitetura de microsserviços oferece vantagens em termos de agilidade no desenvolvimento e implantação de projetos, liberdade na escolha de tecnologias e melhor organização dos serviços. A independência entre os serviços permite maior flexibilidade e facilita a troca de tecnologias. Além disso, os serviços de baixo acoplamento são mais fáceis de reaproveitar e reconfigurar para diferentes propósitos. Por outro lado, a arquitetura monolítica simplifica preocupações relacionadas ao registro de novos serviços e oferece potenciais vantagens de desempenho devido ao acesso mais rápido à memória. No entanto, ela pode apresentar limitações em termos de segurança e comunicação entre componentes.

Al-Debagy e Martinek (2018) realizaram uma análise comparativa entre as arqui-

teturas monolítica e microsserviços focando na análise de desempenho das aplicações. Foram utilizados três tipos de testes diferentes: teste de carga, teste de simultaneidade e teste de resistência. O objetivo era determinar qual das duas arquiteturas apresentava o melhor desempenho. A seguir, é apresentado um resumo de como cada teste foi realizado e os principais resultados obtidos.

a) Teste de carga: consiste em monitorar os efeitos do aumento do número de usuários no aplicativo e como isso afetará o rendimento e o tempo de resposta. Foi utilizada como metodologia a verificação da aceleração, tempo de resposta, taxa de transferência e quantas solicitações cada aplicativo pode lidar em um determinado período de tempo. Foram utilizados 100 processos, aumentando gradativamente até 7000 processos. Os resultados indicaram desempenho semelhante entre microsserviços e arquiteturas monolíticas. A aplicação monolítica teve melhor rendimento com poucos usuários, devido à necessidade de maior comunicação entre serviços e bancos de dados nos microsserviços. Porém, à medida que o número de usuários aumenta, o rendimento das duas arquiteturas se equipara. Assim, a aplicação monolítica se destaca apenas em cenários de baixa carga. A diferença média de desempenho entre elas foi de 0,87%, não sendo estatisticamente significativa.

b) Teste de simultaneidade: avaliou-se o comportamento do sistema diante do uso simultâneo de todos os serviços. Inicialmente, foram enviadas 100 solicitações para cada serviço sem um tempo de aceleração específico, com o número de solicitações aumentando gradualmente até atingir 1000. O objetivo foi verificar como o sistema responderia quando todos os serviços fossem utilizados ao mesmo tempo, através de suas APIs expostas. Todos os serviços foram invocados simultaneamente, sem aceleração específica, resultando em um desempenho superior da arquitetura monolítica em relação ao *throughput*. Houve uma melhoria média de 6% em comparação com a arquitetura de microsserviços em todos os testes.

c) Teste de resistência: avaliou-se a resistência do sistema com 10.000 procesos, um período de aceleração de 10 minutos e um tempo de espera de 10 minutos. Além disso, foram incluídas diferentes configurações de arquitetura de mi-

> crosserviços, utilizando tecnologias como Consul e Eureka para a descoberta de serviços. O objetivo foi analisar como o sistema responderia sob cargas intensas, explorando diferentes abordagens de arquitetura de microsserviços. Foram usados 10.000 processos em 20 minutos, com o aumento de 10 minutos divididos em cinco etapas de 2 minutos cada, começando em 2.000 processos e aumentando gradualmente até 10.000 processos, seguido de 10 minutos de espera. Não houve diferença significativa entre as arquiteturas, mantendo-se em torno de 1790 milissegundos com 100 processos e aumentando para cerca de 15.000 milissegundos com 1000 processos após repetição do teste três vezes.

Desta fomra, Al-Debagy e Martinek (2018) concluíram que, em condições de carga normal, ambos os tipos de aplicativos apresentaram desempenho comparável, embora os monolíticos tenham demonstrado uma pequena vantagem em cargas leves. Recomenda-se, portanto, o uso de aplicativos monolíticos para aplicações de menor escala, com poucos usuários. Além disso, os autores comentam que a implementação de um microsserviço levando em consideração a escolha de sua tecnologia, pode afetar diretamente seu desempenho.

Capítulo 3

Especificação do Sistema Proposto

A mudança de arquiteturas monolíticas para microsserviços emergiu como uma área de pesquisa significativa, impulsionada por transformações tecnológicas e pela busca incessante das empresas por flexibilidade, escalabilidade e eficiência na manutenção. O estudo de Mendes (2021), por exemplo, ofereceu uma análise aprofundada dessas transformações, abordando os desafios e benefícios percebidos por empresas que abraçaram essa abordagem.

Essa linha de pesquisa é crucial não apenas para compreender os motivos por trás das transições entre arquiteturas, mas também para fornecer uma base valiosa para empresas que consideram a migração ou adoção de uma dessas estruturas arquitetônicas. Ao examinar as experiências e práticas bem-sucedidas, bem como os obstáculos enfrentados por essas empresas, o estudo de Mendes (2021) contribuiu para o estabelecimento de um alicerce sólido na tomada de decisões relacionadas a essas arquiteturas.

Dentro desse contexto, busca-se neste trabalho aprofundar ainda mais a compreensão, explorando as implicações específicas para o contexto local e investigando como as decisões arquiteturais podem ser moldadas, levando em conta as motivações específicas das empresas estudadas por Mendes (2021) e as pesquisas de desempenho relacionadas aos sistemas monolíticos e microsserviços, conforme discutido por Lucio (2017) e Al-Debagy e Martinek (2018).

Nesse contexto, foi escolhido para este trabalho propor uma solução tecnológica para o restaurante universitário da UFRRJ que apresenta limitações que dificultam a aquisição de *tickets*. Este trabalho visa destacar os problemas enfrentados pelo restaurante e propor uma solução utilizando os estudos arquiteturais descritos a fim de realizar uma aplicação prática de uma arquitetura.

3.1 O Problema

A gestão do restaurante universitário na UFRRJ enfrenta desafios significativos devido ao sistema de venda de *tickets* físicos em períodos limitados, resultando em dificuldades de acesso e opções limitadas de pagamento, podendo ocasionar longas filas durante os horários de pico.

O principal desafio identificado é a ineficiência do sistema de venda de *tickets* físicos no restaurante universitário, caracterizado pelos problemas a seguir destacados:

a) Horários das aquisições de *tickets*: a venda de *tickets* em períodos específicos cria barreiras de acesso para os alunos, pois uma parte dos horários disponíveis para compra coincide com outras atividades.

b) Método de pagamento: o único método de pagamento aceito é o dinheiro em espécie. Isso pode resultar em pequenas demoras durante as transações, devido à verificação da autenticidade do dinheiro, a necessidade de fornecer troco e possíveis dificuldades em transações de alto valor. Além disso, observa-se que, ao realizar compras, a maioria das pessoas atualmente não utiliza mais o pagamento físico como método principal, preferindo opções digitais pela praticidade e segurança.

c) Filas: devido aos horários limitados para a compra de *tickets* e à aceitação apenas de dinheiro em espécie, há uma concentração de pessoas em determinados períodos, resultando em filas longas.

Dados esses problemas faz necessário um estudo para criação de um sistema de compra e venda de *tickets* a fim de facilitar a vida das pessoas que utilizam o

restaurante.

3.2 Levantamento de Requisitos

A etapa de levantamento de requisitos desempenha um papel crucial na garantia da construção completa e eficaz de um sistema de *software*. Nesse contexto, serão delineados os requisitos funcionais, referentes às funcionalidades do sistema apresentados na Tabela 3.1, os requisitos não funcionais relacionados a aspectos como desempenho, segurança e usabilidade, apresentados na Tabela 3.2 e as regras de negócios, pertinente às restrições ou condições que devem ser seguidas em uma organização, apresentados na Tabela 3.3.

ID	**Descrição**
RF01	O sistema deve permitir que o usuário solicite o seu cadastro.
RF02	O sistema deve permitir que os usuários cadastrados façam *login*.
RF03	O sistema deve permitir que os usuários possam atualizar seu cadastro.
RF04	O sistema deve permitir que os usuários façam compra de *tickets*.
RF05	O sistema deve permitir o técnico debitar *tickets* de usuários.
RF06	O sistema deve permitir a visualização da carteira virtual do usuário.
RF07	O sistema deve permitir que o administrador valide usuários.
RF08	O sistema deve permitir que o técnico realize a venda de *ticket*.
RF09	O sistema deve permitir que o técnico realize o reembolso de *ticket*.
RF10	O sistema deve permitir que o administrador modifique as informações do usuário.
RF11	O sistema deve permitir a visualização do histórico de compras do usuário.
RF12	O sistema deve permitir a visualização do histórico de vendas do técnico.
RF13	O sistema deve permitir a emissão de relatórios de vendas de *tickets*.

Tabela 3.1: Requisitos Funcionais

ID	Descrição
RNF01	O sistema deve suportar 400 usuários simultaneamente.
RNF02	O sistema deve ficar ativo em 90% do dia.
RNF03	O sistema deve ser compatível com os navegadores Google Chrome(a partir da versão 112.0), Mozila Firefox(a partir da versão 114), Microsoft Edge(a partir da versão 113).
RNF04	O sistema deve permitir pagamentos via Pix.
RNF05	O sistema deve ser responsivo e adaptar-se automaticamente para dispositivos móveis e computadores.

Tabela 3.2: Requisitos Não Funcionais

3.3 Casos de Uso

A seguir, serão detalhados os casos de uso correspondentes aos requisitos funcionais apresentados anteriormente. Os casos de uso, juntamente com seus respectivos diagramas, são uma técnica amplamente utilizada para documentar e especificar os requisitos funcionais de um sistema. Essa abordagem descreve as funcionalidades do sistema a partir da perspectiva do usuário final, fornecendo uma visão clara das interações típicas entre os usuários e o sistema (BOOCH, 2005).

O diagrama de casos de uso do sistema em questão está ilustrado na Figura 3.1, o qual visualiza de forma concisa as principais interações entre os usuários e os componentes do sistema. Esse diagrama é essencial para compreender as funcionalidades e os fluxos de interação de maneira geral.

Além do diagrama, a especificação detalhada de cada caso de uso é apresentado no Apêndice A, proporcionando uma descrição mais aprofundada de como cada funcionalidade é executada. Essas tabelas incluem os seguintes elementos para cada caso de uso: os atores envolvidos, o fluxo de eventos, as pré-condições, as pós-condições, as referências cruzadas e a descrição. A especificação completa visa fornecer uma documentação técnica e precisa, que serve como referência para todo o

desenvolvimento do sistema.

ID	Descrição
RN01	O usuário deve fornecer dados como nome completo, endereço de e-mail e matrícula.(RF01)
RN02	O sistema deve permitir o cadastro de dois tipos de usuários: usuários e técnicos.(RF01)
RN03	A senha deve conter no mínimo 8 caracteres.(RF03)
RN04	O sistema deve notificar o usuário após validação do cadastro pelo administrador.(RF07)
RN05	O *login* deverá ser realizado fornecendo sua matrícula e senha.(RF03)
RN06	Os *tickets* disponíveis para compra devem incluir opções para café da manhã e refeição.(RF04)
RN07	Os *tickets* adquiridos devem ser armazenados na carteira virtual do usuário para uso futuro.(RF05)
RN08	O sistema deve descontar um *ticket* da carteira do usuário com base no tipo de *ticket* utilizado.(RF05)
RN09	O sistema deve emitir os relatórios em formato PDF e XLSX.(RF13)
RN10	O sistema deve notificar o usuário quando houver apenas um *ticket* em sua carteira.(RF04)
RN11	O sistema deve permitir o acesso via *QR Code* e matrícula.
RN12	O sistema deve registrar o dia e o tipo de *ticket* utilizado.(RF05)

Tabela 3.3: Regras de Negócios

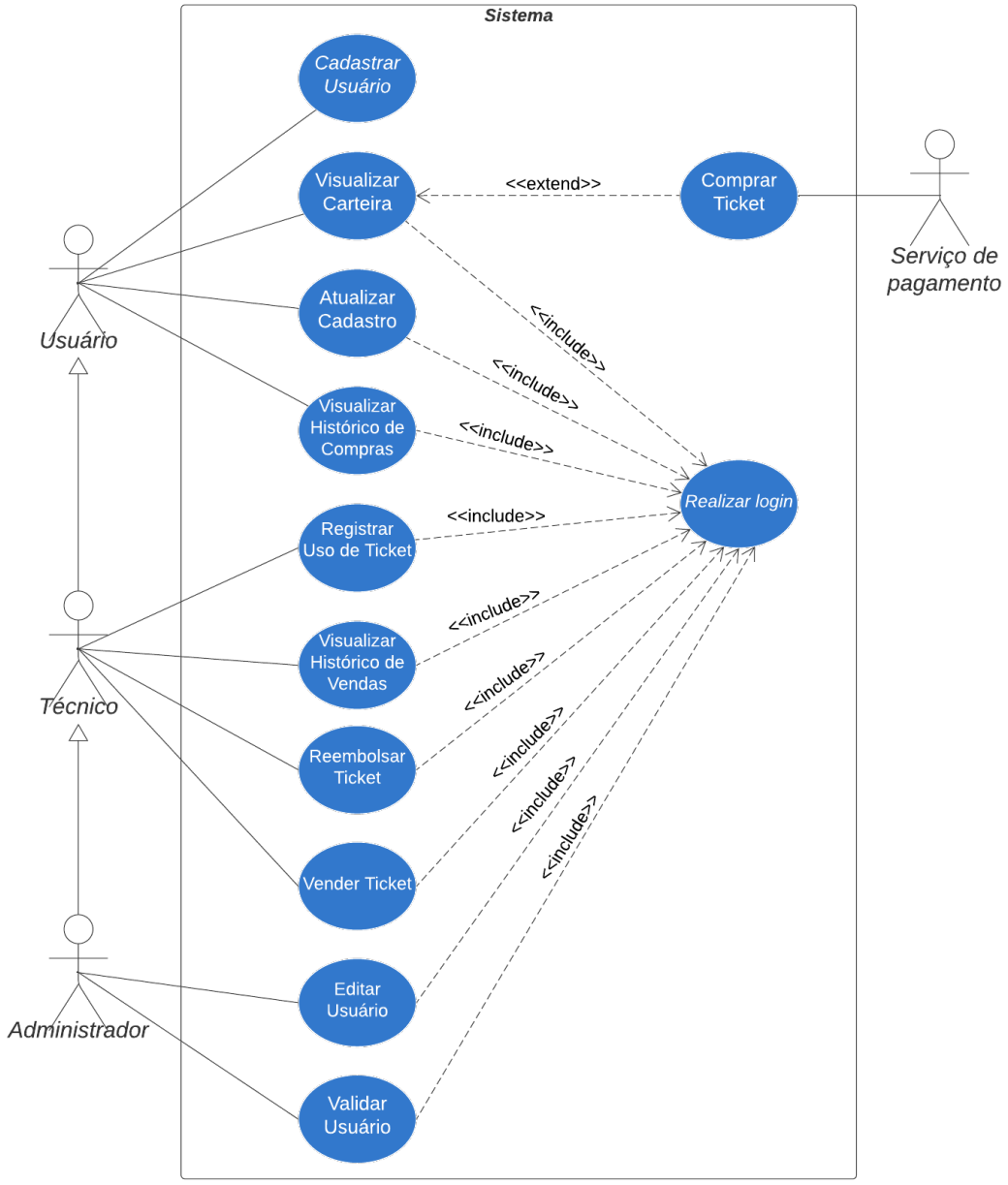

Figura 3.1: Diagrama de Casos de Uso.

3.4 Diagrama de Classe

Um diagrama de classe é um tipo de diagrama usado na modelagem de sistemas orientados a objetos. Ele representa as classes dentro de um sistema, suas propriedades (atributos), métodos (operações) e os relacionamentos entre elas. Esse diagrama é útil para visualizar a arquitetura de um sistema, facilitando o entendimento, projeto e documentação do *software* (BOOCH, 2005).

O diagrama de classe do sistema é apresentado na Figura 3.2 e contém:

a) Classes: representadas como retângulos, que definem as entidades do sistema. As classes do sistema incluem Usuario, Carteira, Administrador, Identificador, TipoTicket e Compra.

b) Atributos: representam as propriedades das entidades. Por exemplo, a classe Usuario possui os atributos nome, e email, matrícula, senha, tipoUsuario.

c) Métodos: são as operações que podem ser realizadas pelas classes. Por exemplo, a classe Usuario possui o método atualizarCadastro().

d) Relacionamentos: as linhas entre as classes representam os relacionamentos. Por exemplo, a relação entre Usuario e Compra indica que um usuário pode realizar múltiplas compras. Esses relacionamentos podem ser de diferentes tipos:

e) Generalização: representa uma relação hierárquica em que uma classe mais específica herda características e comportamentos de uma classe mais genérica. No sistema a ser desenvolvido, a criação de um usuário com funcionalidades básicas facilita a reutilização de código, permitindo que a classe administrador herde todos os comportamentos da classe usuário e, além disso, incorpore suas próprias funcionalidades.

f) Composição: define uma relação de dependência forte entre duas classes, em que a existência de uma parte está completamente vinculada à existência do todo. Se o todo for destruído, suas partes também deixam de existir. No contexto do sistema a ser desenvolvido, o usuário está associado a uma carteira, estabelecendo uma relação de forte dependência entre as duas entidades. Dessa

forma, caso o usuário seja excluído, a carteira vinculada a ele também será removida, uma vez que sua existência está diretamente ligada à do usuário. Da mesma forma que a compra está ligada a um usuário, se o usuário não existir, a compra também não pode existir.

g) Agregação: estabelece uma relação de dependência mais fraca entre duas classes, onde uma parte pode continuar existindo independentemente do todo. Não há relação de agregação identificada no contexto do sistema proposto.

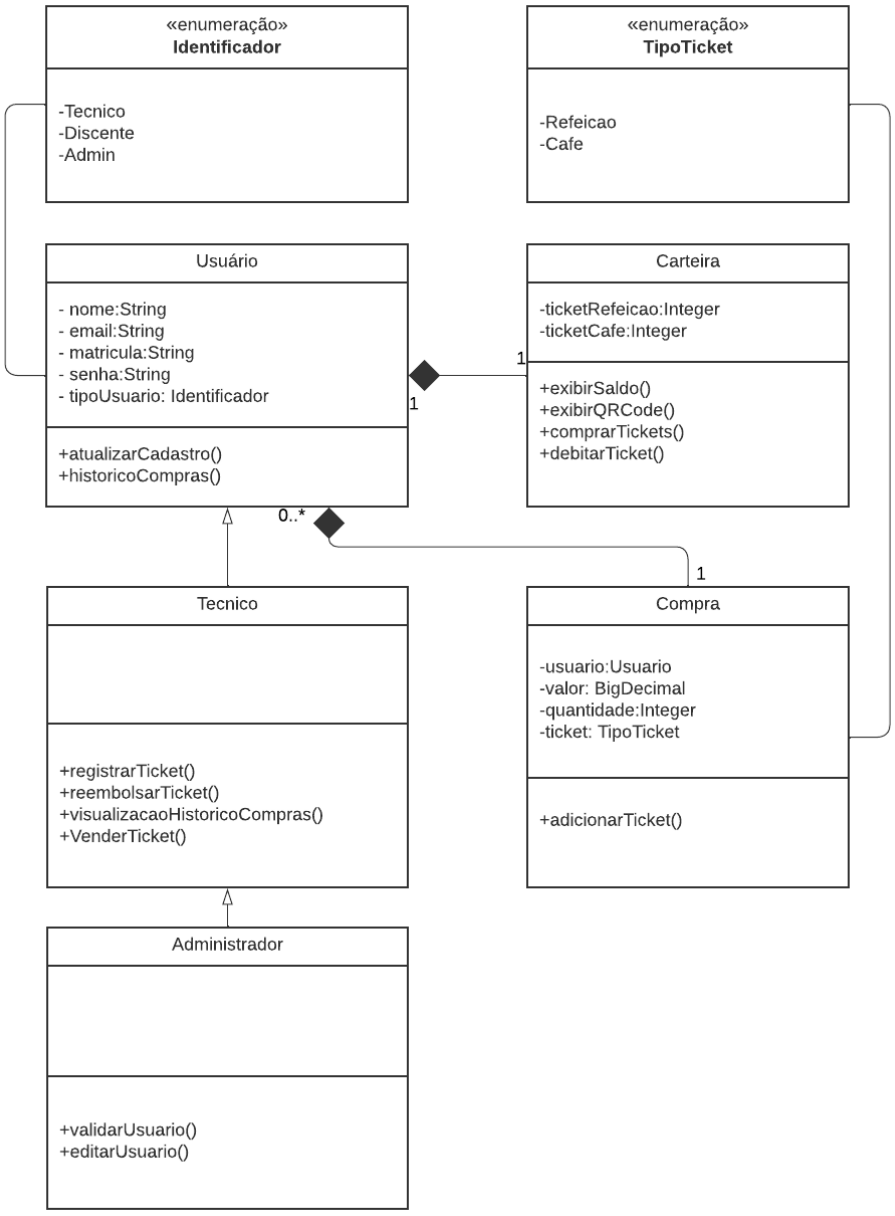

Figura 3.2: Diagrama de classe do sistema.

3.5 Escolhendo a arquitetura

A partir do levantamento de requisitos e dos casos de uso apresentados, é importante destacar algumas características essenciais que foram consideradas para a escolha da arquitetura ideal para o sistema proposto. Entre essas características, pode-se citar:

a) Considerando uma média de 500 usuários diários que utilizam o serviço no almoço e no jantar, e 110 usuários no café da manhã, pode-se inferir que o sistema seria voltado para um pequeno nicho;

b) Em relação à variedade de funcionalidades, o sistema se concentraria principalmente na compra de *tickets* e na validação de usuários, o que implica em um conjunto limitado de recursos;

c) Quanto à capacidade de evolução ao longo do tempo, o sistema não demandaria escalabilidade significativa, pois suas funcionalidades básicas se destinam a resolver um problema específico, sem a necessidade de expansão futura;

d) Considerando o escopo limitado de funcionalidades, o tamanho do sistema seria pequeno, adequado ao seu propósito específico e não exigiria uma estrutura de grande porte.

Esses aspectos são cruciais para garantir que a arquitetura escolhida seja capaz de atender às necessidades do sistema e proporcionar uma experiência satisfatória para os usuários.

A partir dessas características apresentadas e do estudo das arquiteturas realizado no Capítulo 2, foi possível realizar algumas análises. Como por exemplo, em um sistema pequeno e com poucas funcionalidades a serem implementadas, uma arquitetura monolítica seria uma boa opção, especialmente para projetos iniciais. Isso ocorre porque a implantação é mais simples e abrange um conjunto básico de funcionalidades. Em contraste, a arquitetura de microsserviços, embora ofereça alta escalabilidade e flexibilidade, apresenta maior complexidade na implementação.

No caso do sistema em questão, a necessidade de escalabilidade e flexibilidade

proporcionada pelos microsserviços não é um fator determinante, pois o sistema a ser implementado será um pequeno projeto, destinado a atender inicialmente os alunos e técnicos do Instituto Multidisciplinar da Universidade Federal Rural do Rio de Janeiro. Portanto, uma arquitetura monolítica se mostra mais adequada para este projeto, devido ao tamanho e à complexidade do projeto, facilitando o desenvolvimento inicial, na manutenção e na implantação.

Caso o sistema fosse desenvolvido para atender a outra universidade, a arquitetura monolítica ainda seria uma boa opção, já que se trataria de um projeto com escopo limitado, um público restrito e funcionalidades básicas. Isso garantiria simplicidade no desenvolvimento, manutenção e implantação, mantendo a eficiência. No entanto, caso o sistema fosse expandido para atender diversas universidades do Brasil, a arquitetura de microsserviços se tornaria mais adequada. Com um público muito maior e a necessidade de escalar o sistema de maneira eficiente, os microsserviços permitiriam uma maior flexibilidade, facilitando a implementação de novas funcionalidades e a distribuição de cargas de trabalho. Essa possibilidade de expansão exigiria uma estrutura capaz de suportar grandes volumes de dados e usuários simultâneos, tornando a escalabilidade e a independência entre os serviços fatores cruciais para garantir a performance e a disponibilidade do sistema em nível nacional. Além disso, a arquitetura de microsserviços facilitaria a integração com diferentes sistemas, a implementação de atualizações sem impactar todo o sistema e a possibilidade de adotar diferentes tecnologias para diferentes serviços, otimizando ainda mais o desempenho geral do sistema.

Capítulo 4

Desenvolvimento do Sistema Proposto

Neste capítulo, são apresentadas as principais tecnologias utilizadas no desenvolvimento do sistema. A definição das tecnologias utilizadas considerou os requisitos previamente levantados, buscando alinhar as soluções implementadas às necessidades do sistema. Adicionalmente, são incluídas representações visuais da aplicação com o objetivo de demonstrar, de forma prática, o comportamento das funcionalidades desenvolvidas e a composição da interface do sistema.

4.1 Tecnologias utilizadas

No desenvolvimento de *software*, diversas ferramentas, como linguagens de programação, *frameworks* e bibliotecas, desempenham um papel fundamental ao facilitar a implementação de sistemas complexos. Essas ferramentas fornecem abstrações e funcionalidades pré-definidas que permitem aos desenvolvedores criar, gerenciar e escalar aplicações de forma eficiente. Segundo Pressman e Maxim (2021), a utilização de tecnologias adequadas ao projeto possibilita uma implementação mais estruturada e modular, facilitando tanto o desenvolvimento quanto a manutenção do *software*.

Dessa forma, a linguagem Java foi selecionada para o *back-end* devido à sua popularidade e vasta documentação, o que facilita a manutenção e a segurança do sistema. Java é uma linguagem orientada a objetos e possui uma forte tipagem,

características que contribuem para o desenvolvimento seguro e eficiente (DEITEL et al., 2016).

Aproveitando essas características, o *framework* Spring Boot foi escolhido porque simplifica a criação de aplicações em Java, fornecendo uma estrutura de desenvolvimento que facilita a construção de sistemas *web* e a integração com bancos de dados. Segundo Walls (2015), o Spring Boot permite uma configuração mínima, com suporte integrado para o gerenciamento de dependências e injeção de dependências, além de oferecer diversos módulos para controle de segurança e persistência de dados. Essas características foram determinantes para a escolha do Spring Boot, uma vez que ele proporciona rapidez e eficiência no desenvolvimento de aplicações *web*.

Para o armazenamento de dados, optou-se pelo banco de dados relacional MySQL, devido à sua estabilidade e suporte para gerenciamento de dados relacionais. O MySQL oferece um desempenho eficiente para operações de leitura e escrita e suporta a modelagem de relacionamentos complexos, essenciais para a organização dos dados no sistema desenvolvido (DUBOIS, 2013). Além disso, para garantir a segurança das credenciais dos usuários, as senhas são armazenadas de forma segura utilizando o algoritmo de hash bcrypt. O bcrypt é baseado na cifra Blowfish e foi projetado especificamente para proteger senhas, aplicando uma função de derivação de chave com sal (salt) embutido e fator de custo ajustável, tornando-o resistente a ataques de força bruta e dicionário, aumentando assim a segurança geral do sistema (PROVOS; MAZIERES, 1999).

Com os dados devidamente estruturados no banco de dados, é necessário apresentá-los de forma clara, funcional e interativa no *front-end*. Para isso, foram utilizadas tecnologias como *Hypertext Markup Language* (HTML), *Cascading Style Sheets* (CSS) e JavaScript, responsáveis, respectivamente, pela estruturação, estilização e interatividade das páginas *web*.

Nesse contexto, o Thymeleaf, um motor de *templates* moderno do lado do servidor, foi escolhido para facilitar a renderização dinâmica das informações. Projetado para ambientes *web* e autônomos, seu principal objetivo é oferecer *templates* naturais e elegantes que podem ser exibidos corretamente em navegadores e funcionar como

protótipos estáticos, facilitando a colaboração entre equipes de desenvolvimento (Thymeleaf, 2024). Com módulos para o Spring Boot, diversas integrações e a possibilidade de adicionar funcionalidades próprias, o Thymeleaf se apresenta como uma opção viável para o desenvolvimento de aplicações *web*.

No desenvolvimento de páginas *web*, o HTML é usado para estruturar o conteúdo na *web*, fornecendo a base para a criação de textos, imagens, *links* e outros elementos essenciais de uma página (DUCKETT, 2011). Com o CSS, é possível definir a apresentação visual do conteúdo HTML, controlando o *layout*, cores, fontes e outros aspectos estéticos da página (DUCKETT, 2011). O JavaScript, por sua vez, é a linguagem de programação utilizada para adicionar interatividade à página *web*, permitindo a manipulação de elementos dinâmicos, validação de formulários, animações e atualizações de conteúdo sem a necessidade de recarregar a página, proporcionando uma experiência de usuário mais rica e interativa (DUCKETT, 2011).

Além das tecnologias mencionadas, foi utilizado como serviço externo a API do Pix do Banco do Brasil, acessada exclusivamente em seu ambiente de testes (*sandbox*) [1]. Esse serviço foi fundamental para simular a geração e o gerenciamento de cobranças via Pix, possibilitando a validação das funcionalidades desenvolvidas sem a necessidade de operar em ambiente de produção.

4.2 Exemplos de Uso

A aplicação desenvolvida foi denominada Capiwallet, uma junção dos termos "Capi", em referência à capivara, mascote oficial da universidade, e "*wallet*", palavra em inglês que significa carteira. A escolha do nome visa integrar a identidade institucional à principal funcionalidade do sistema, que consiste na compra e gerenciamento de *tickets* por meio de uma carteira digital.

O sistema contempla três perfis distintos de usuários: usuário aluno, com acesso às funcionalidades básicas da plataforma; usuário técnico, responsável por funções operacionais específicas; e usuário administrador, que detém privilégios avançados

[1] https://apoio.developers.bb.com.br/sandbox/spec/post/5fe9e7f13b02bd0012eca9f1

para controle, configuração e gerenciamento geral do sistema.

A seguir, serão apresentadas as principais funcionalidades do sistema, detalhando a estrutura e os recursos oferecidos. A página principal do sistema, mostrada na Figura 4.1, permite que os usuários realizem o *login* através da matrícula e senha, servindo como o ponto de acesso para os usuários.

Figura 4.1: Tela de login.

Caso o usuário do perfil aluno não possua conta criada no sistema, é possível solicitar sua criação na página principal. Uma vez solicitado, o usuário tem acesso à página de cadastro, mostrada na Figura 4.2, que permite o preenchimento de um formulário com nome completo, matrícula e senha. Após o envio das informações, o cadastro fica pendente de aprovação ou rejeição, apresentada na Figura 4.3. A solicitação será analisada pela equipe responsável, que verificará se a matrícula e o nome completo informados são válidos. Caso estejam corretos, o acesso ao sistema será autorizado; caso contrário, a solicitação será recusada, garantindo a segurança e a integridade dos dados.

Figura 4.2: Tela de cadastro.

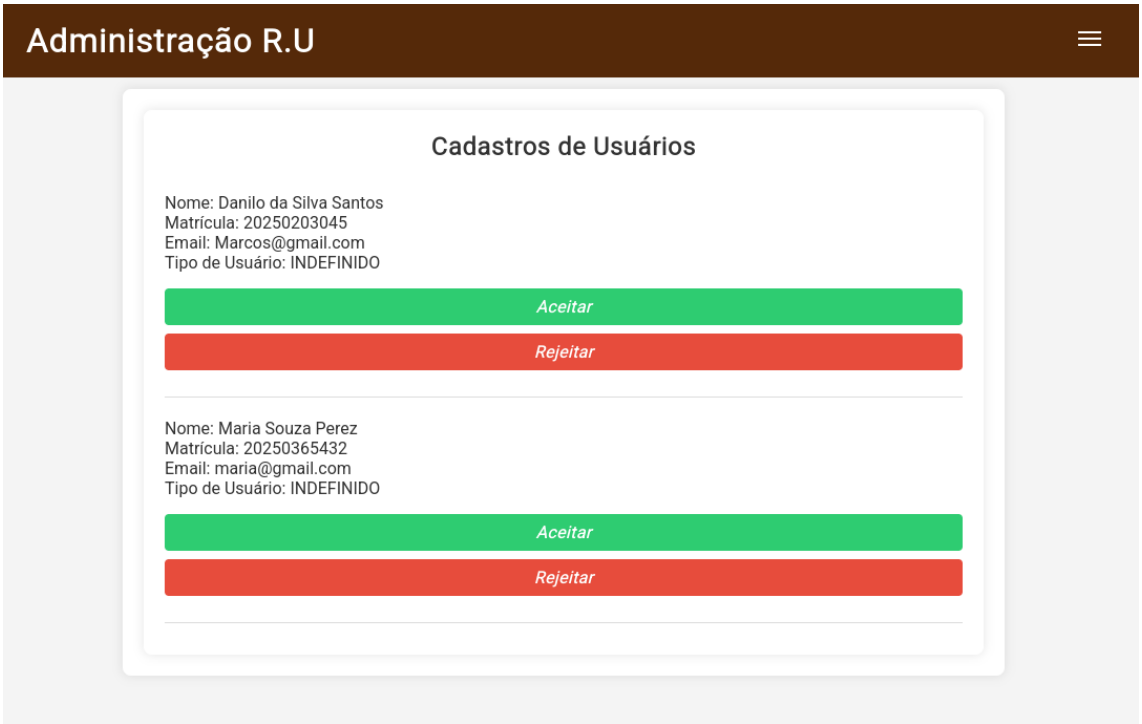

Figura 4.3: Tela de usuários pendentes para validação.

Após a aprovação do cadastro, ao realizar o *login*, o usuário de perfil aluno acessa a página principal, mostrada na Figura 4.4, que exibe o *QR Code* do usuário. Essa escolha visa facilitar o acesso ao restaurante universitário, permitindo que o usuário apresente rapidamente o código para leitura para o técnico na entrada, agilizando o processo e evitando etapas desnecessárias de navegação no sistema.

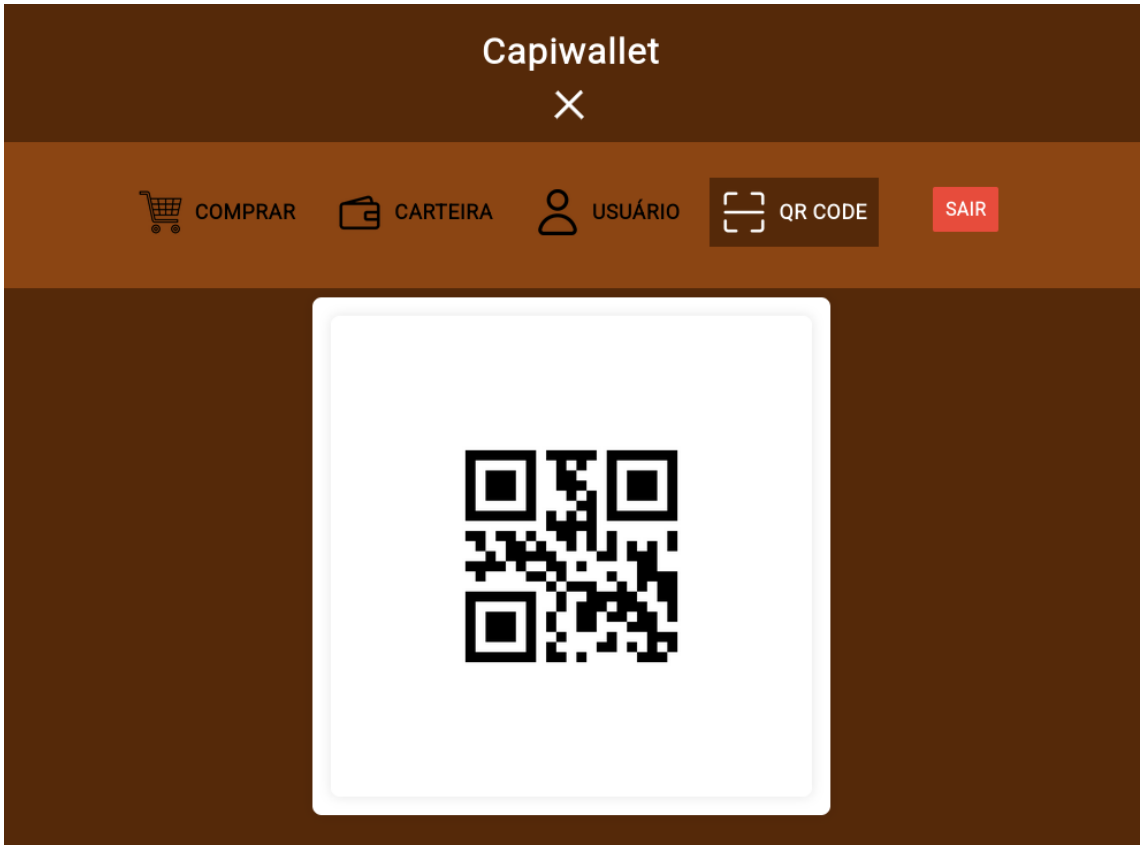

Figura 4.4: Tela do *QR Code*.

O sistema permite ao discente selecionar diferentes opções de navegação, tais como "Comprar", "Carteira", "Usuário" e "*QR Code*". Ao selecionar a opção "Comprar", exibe a página correspondente, conforme na Figura 4.5.

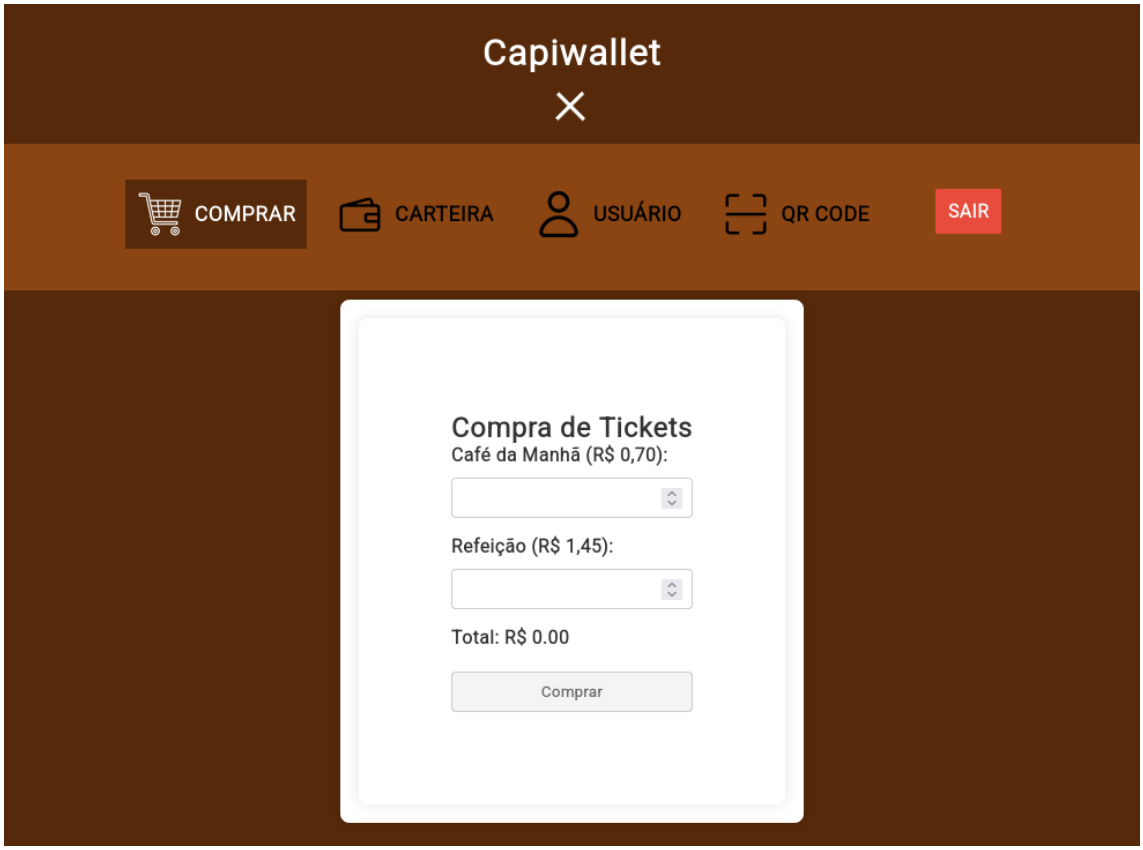

Figura 4.5: Tela de comprar *tickets*.

Após a realização de uma compra, na qual o usuário seleciona o tipo e a quantidade de *tickets* desejados, é possível consultar, na seção "Carteira", o total de *tickets* adquiridos, conforme apresentado na Figura 4.6.

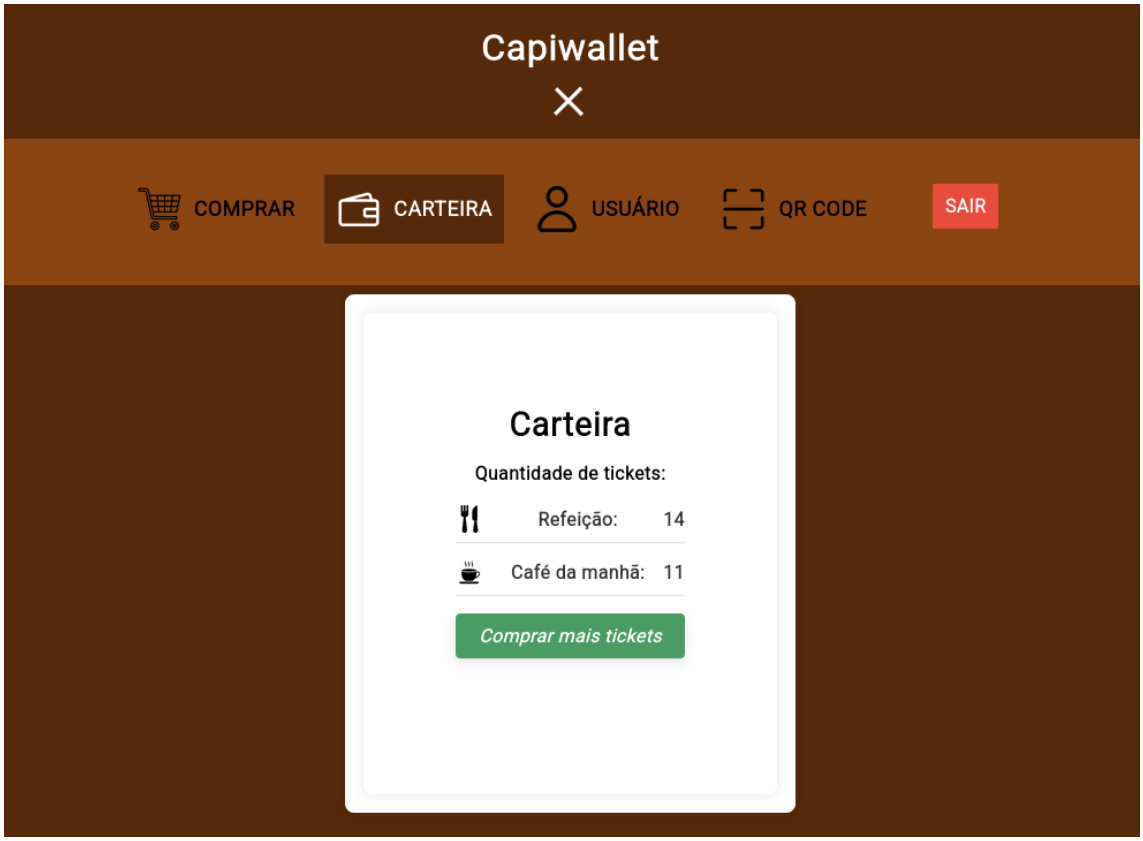

Figura 4.6: Tela de *tickets* comprados.

O usuário também pode visualizar suas informações pessoais acessando a seção "Usuário", onde estão disponíveis as opções para alterar a imagem de perfil, trocar a senha, modificar o e-mail e consultar o histórico de compras, conforme apresentado na Figura 4.7.

Figura 4.7: Tela de informações do usuário.

Ao acionar a opção "Histórico de Compras", o sistema exibe, na mesma página, uma lista com todas as compras realizadas pelo usuário, incluindo transações efetuadas via Pix e compras presenciais, como mostra a Figura 4.8.

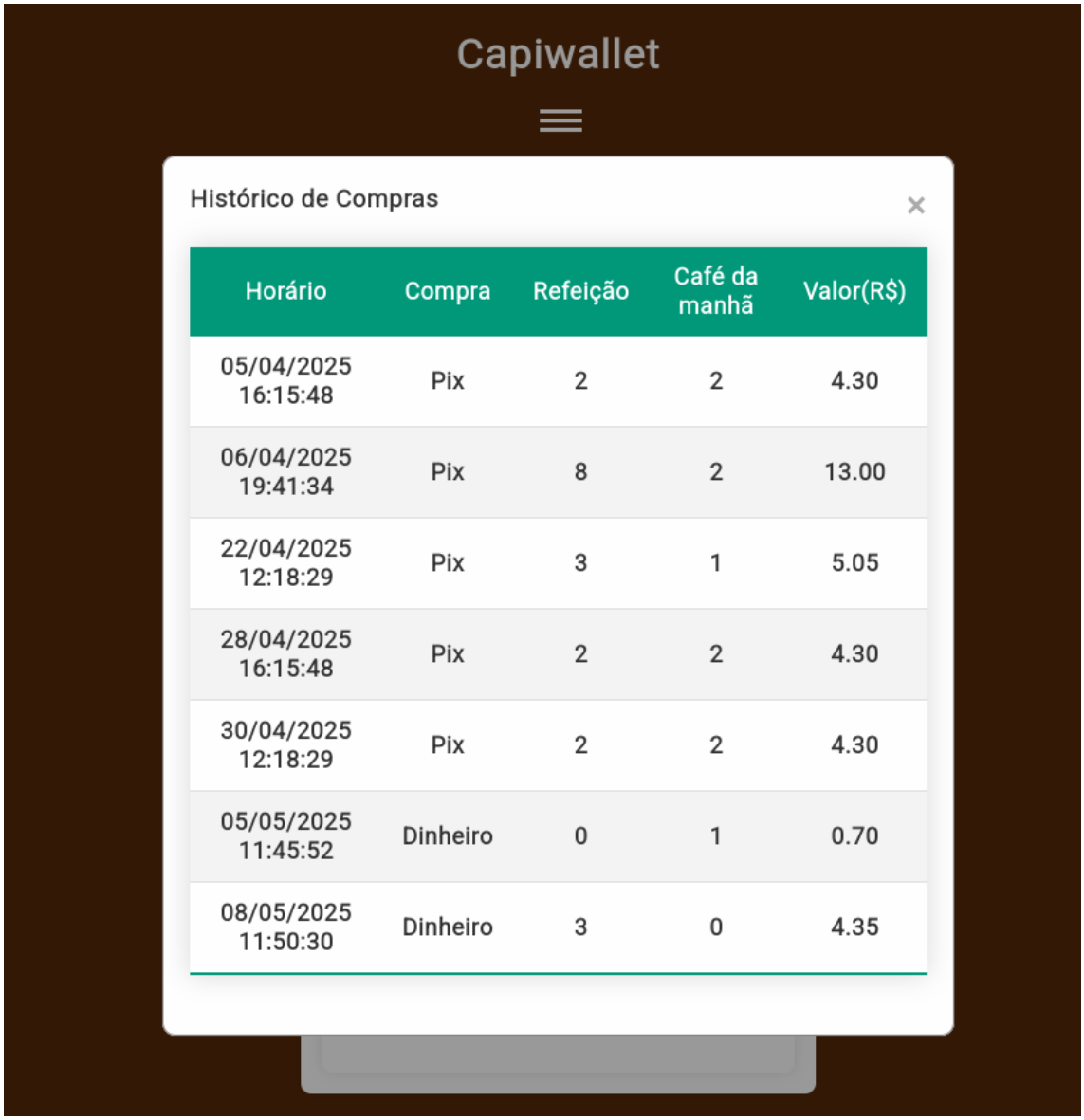

Figura 4.8: Tela do histórico de compras.

A seguir, apresenta-se a interface dos técnicos, previamente cadastrados pelo administrador. Após realizar o *login*, o técnico é direcionado à sua página inicial, podendo realizar a busca por *QR Code*, conforme apresentado na Figura 4.9. A partir dessa funcionalidade, o técnico pode solicitar a leitura do *QR Code* exibido pelo usuário com perfil de aluno, utilizado no momento de acesso ao restaurante para a utilização dos *tickets*. Nesta página, o técnico também possui acesso às funcionalidades de busca por matrícula, busca por *QR Code* e compra presencial.

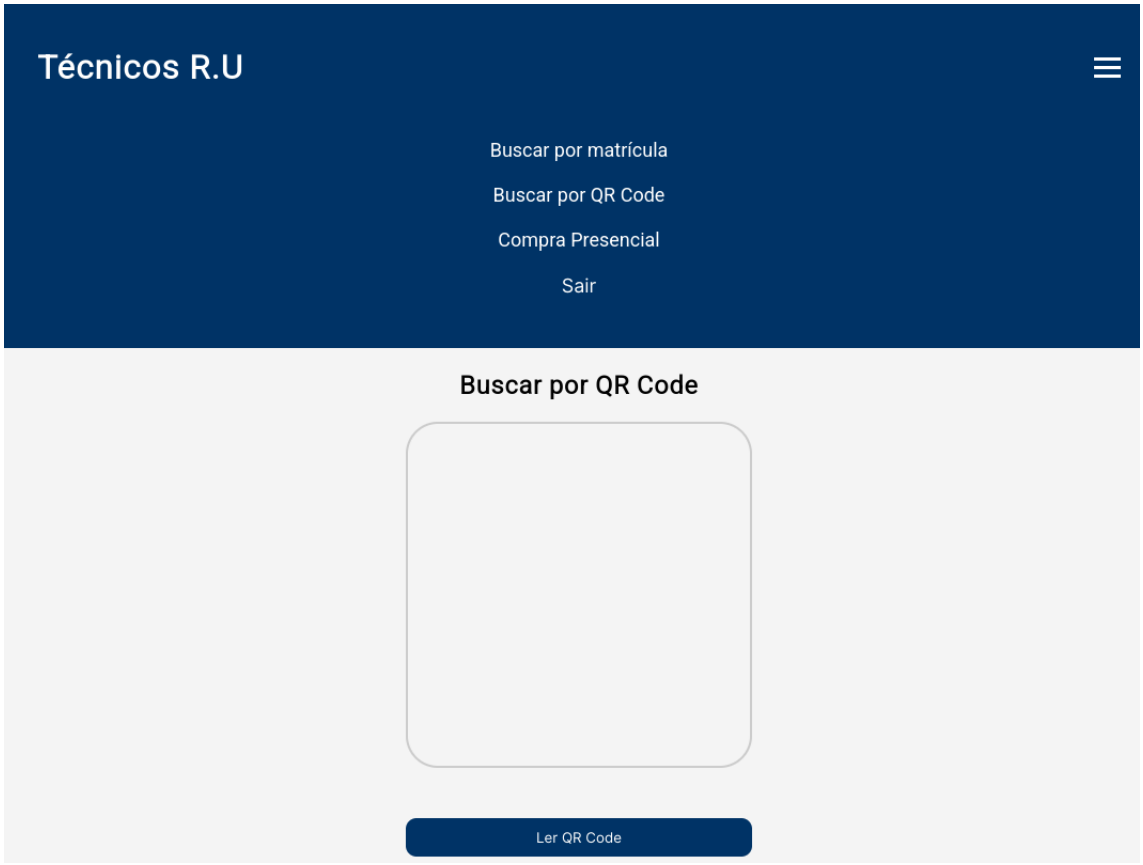

Figura 4.9: Tela de leitura por *QR Code*.

Ao selecionar a opção de busca por matrícula, apresentada na Figura 4.10, o técnico insere manualmente o número da matrícula do usuário no sistema, o qual retorna suas informações, desde que o usuário esteja devidamente cadastrado. A partir disso, é possível realizar o desconto do *ticket* correspondente. Esta opção foi disponibilizada para o caso de não ser possível a leitura do *QR Code* por algum motivo.

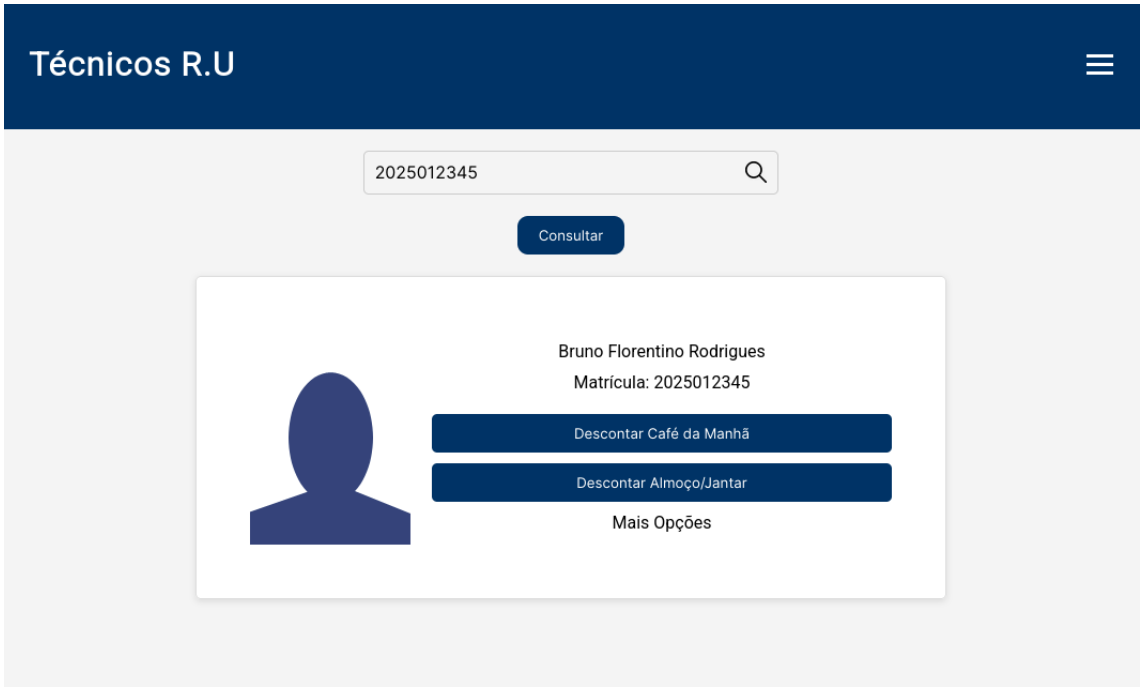

Figura 4.10: Tela de leitura por matrícula.

Em seguida, são exibidas as páginas na versão móvel do sistema, mostradas nas Figuras 4.11 e 4.12. A versão móvel foi desenvolvida para oferecer maior praticidade e acessibilidade aos usuários, permitindo o acesso às funcionalidades do sistema a qualquer momento, por meio de dispositivos móveis. Este formato assegura uma experiência de uso otimizada, com uma interface adaptada para telas reduzidas.

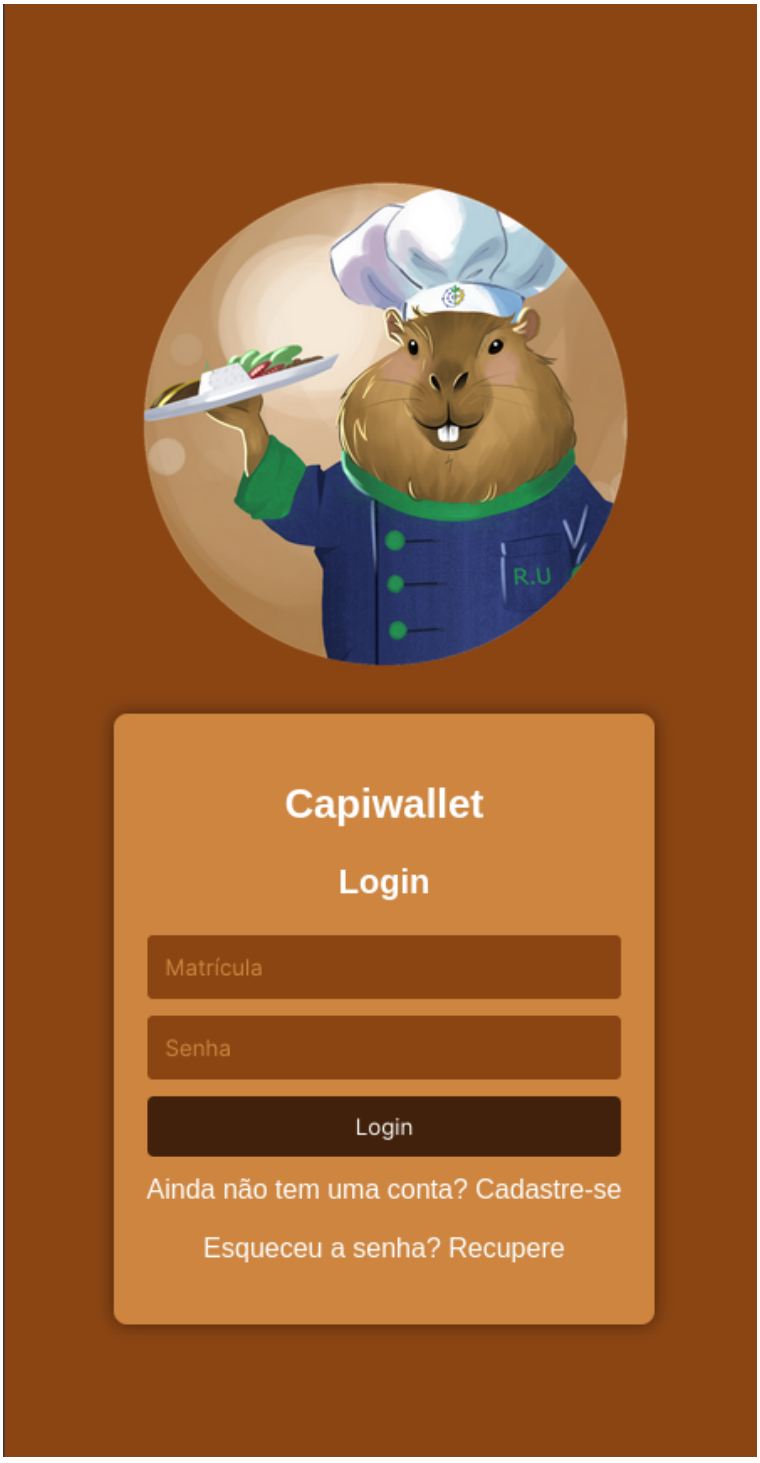

Figura 4.11: Tela de *login* versão móvel.

Figura 4.12: Tela da carteira versão móvel.

Capítulo 5

Avaliação do Sistema

Este capítulo apresenta a avaliação do sistema, realizada com base no modelo *Technology Acceptance Model* (TAM). São descritos o planejamento da avaliação, o processo de coleta dos dados com os participantes e a análise dos resultados obtidos.

5.1 Planejamento

Para avaliar a proposta desenvolvida neste trabalho, foi elaborado um formulário *online* com o objetivo de coletar opiniões qualitativas e quantitativas dos possíveis usuários. A coleta de dados visou compreender a usabilidade, eficiência e relevância da solução proposta, a partir da perspectiva dos participantes. Esta avaliação foi baseada no modelo *Technology Acceptance Model* (TAM).

O TAM foi desenvolvido com o objetivo de explicar os principais fatores que influenciam a aceitação de sistemas de tecnologia por parte dos usuários (DAVIS, 1989). Sua aplicação é amplamente reconhecida na literatura científica pela capacidade de prever a intenção de uso com base em dois fatores principais: a utilidade percebida (*perceived usefulness*) e a facilidade de uso percebida (*perceived ease of use*).

Este modelo tem sido estendido e validado em diversos contextos, incluindo ambientes corporativos, acadêmicos e plataformas digitais, como demonstrado por Venkatesh e Davis (2000) e, posteriormente, em uma proposta mais ampla de

unificação de modelos de aceitação de tecnologia (VENKATESH et al., 2003).

Ao utilizar o modelo TAM neste trabalho, busca-se identificar e mensurar a percepção dos usuários sobre o Capiwallet, permitindo a análise de fatores que influenciam sua aceitação. Essa abordagem fornece uma base teórica sólida para compreender o comportamento dos usuários em relação ao uso da tecnologia, o que é essencial para propor melhorias e aumentar a adoção do sistema em questão (DAVIS, 1989).

O formulário *online* foi elaborado usando o Google Forms [1] devido à praticidade e acesso gratuito disponibilizado por esta ferramenta. O formulário elaborado é apresentado no Apêndice B.

As perguntas que compõem o formulário foram baseadas nas duas dimensões do modelo TAM, utilidade percebida e facilidade percebida, com relação à dimensão "Utilidade Percebida" por exemplo, foi elaborada a pergunta "Em relação à sua satisfação com a utilidade do *software*, você considera que ele atende às suas necessidades?" apresentada na Figura 5.1. Essa questão visa avaliar a percepção dos usuários sobre o grau em que o *software* contribui para o desempenho de suas tarefas ou objetivos, sendo um dos principais indicadores da intenção de uso da tecnologia.

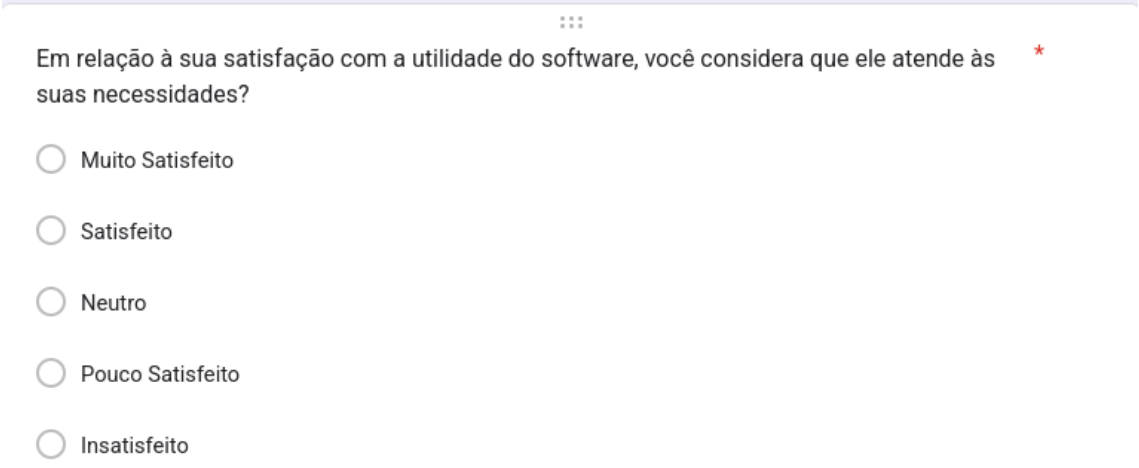

Figura 5.1: Exemplo de pergunta baseada na utilidade percebida.

Com relação à dimensão "Facilidade de Uso Percebida" foram elaboradas per-

[1] https://docs.google.com/forms

guntas semelhantes à pergunta "Em relação à sua satisfação com a visualização e o gerenciamento de *tickets* adquiridos, você acha que o sistema oferece uma forma fácil de gerenciar suas compras?" como apresentada na Figura 5.2. O objetivo dessa questão é avaliar se o usuário percebe o sistema como intuitivo e de fácil navegação, especialmente no processo de acesso e controle dos *tickets* adquiridos. Essa dimensão é essencial para entender como a usabilidade influencia a aceitação da tecnologia.

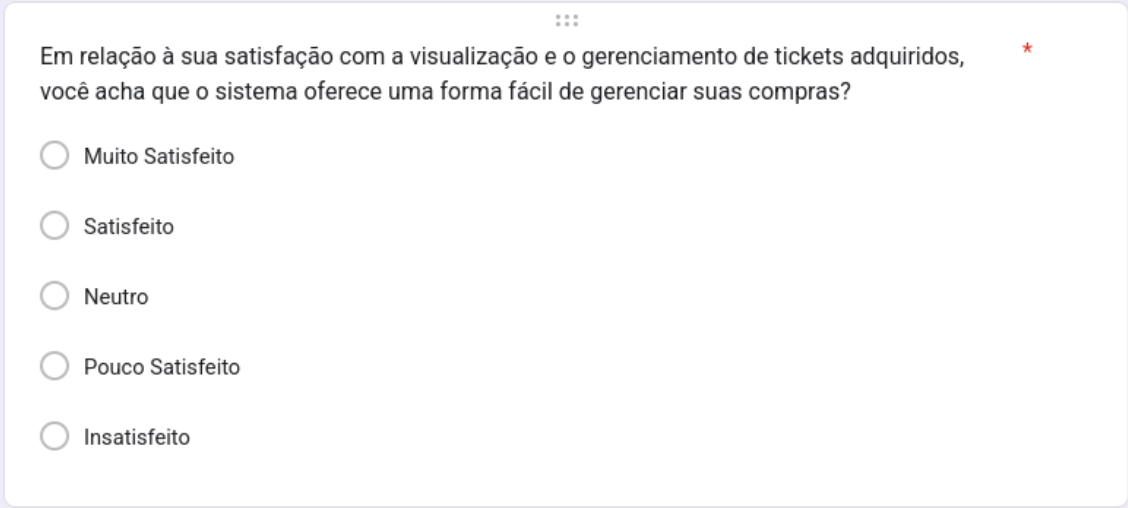

Figura 5.2: Exemplo de pergunta baseada em facilidade de uso percebida.

As perguntas aplicadas tiveram como foco principal a avaliação das funcionalidades centrais do sistema desenvolvido, buscando identificar o nível de satisfação dos usuários em relação ao seu desempenho. Para isso, utilizou-se, em todas as perguntas, a escala de Likert de 5 pontos, que permite mensurar atitudes e percepções com base em um conjunto de respostas ordenadas (LIKERT, 1932). Neste caso, os participantes responderam variando de 1 (Insatisfeito) a 5 (Muito Satisfeito). Além das questões fechadas, também foi disponibilizada uma pergunta aberta, permitindo que os participantes expressassem livremente suas opiniões, críticas ou sugestões em relação ao sistema.

5.2 Coleta dos Dados

A coleta de dados foi realizada com um total de 18 participantes, sendo composta por 17 alunos da UFRRJ, campus Nova Iguaçu e pela responsável pelo restaurante universitário. Para os alunos, foi aplicado um formulário contendo questões fechadas e uma pergunta aberta,conforme apresentado na seção anterior, com o objetivo de avaliar o nível de satisfação em relação às funcionalidades do sistema proposto. Já com a responsável pelo restaurante, foi realizada uma reunião, na qual foi possível obter um retorno qualitativo sobre a aplicação.

Antes do preenchimento do formulário, os participantes tiveram a oportunidade de utilizar o sistema e testar suas principais funcionalidades, de modo a proporcionar uma avaliação mais concreta e embasada. O procedimento de coleta foi realizado por meio de abordagem direta, na qual os alunos foram convidados a interagir com o sistema e, em seguida, registrar suas percepções por meio do formulário.

Os alunos que participaram da pesquisa pertencem a diferentes cursos de graduação da universidade, incluindo História, Pedagogia e Ciência da Computação, o que possibilitou uma diversidade de perspectivas quanto à usabilidade e à efetividade do sistema desenvolvido.

5.3 Análise dos Dados Coletados

Os dados coletados foram consolidados e apresentados em gráficos para facilitar a interpretação dos resultados. A Figura 5.3 mostra a distribuição das respostas para cada pergunta, permitindo avaliar a percepção geral dos usuários em relação aos diferentes aspectos do sistema.

A fim de facilitar a visualização do gráfico, cada pergunta foi substituída por um código, conforme apresentado a seguir:

P1. Experiência geral: “Em relação à experiência geral de uso, você considera que o *software* atendeu às suas expectativas?”

P2. Processo de compra: "Em relação à sua satisfação com o processo de compra de *tickets*, você acha que ele é rápido e fácil de realizar?"

P3. Gerenciamento de *tickets*: "Em relação à sua satisfação com a visualização e o gerenciamento de *tickets* adquiridos, você acha que o sistema oferece uma forma fácil de gerenciar suas compras?"

P4. Facilidade de aprendizado: "Em relação à sua satisfação com a facilidade de aprendizado, como você avaliaria a experiência de aprender a usar o *software*?"

P5. Interface: "Em relação à sua satisfação com a interface do *software*, você considera que ela é intuitiva e fácil de usar?"

P6. Utilidade: "Em relação à sua satisfação com a utilidade do *software*, você considera que ele atende às suas necessidades?"

P7. Design e estética: "Em relação à sua satisfação com o *design* e a estética do *software*, você acha que a aparência é agradável?"

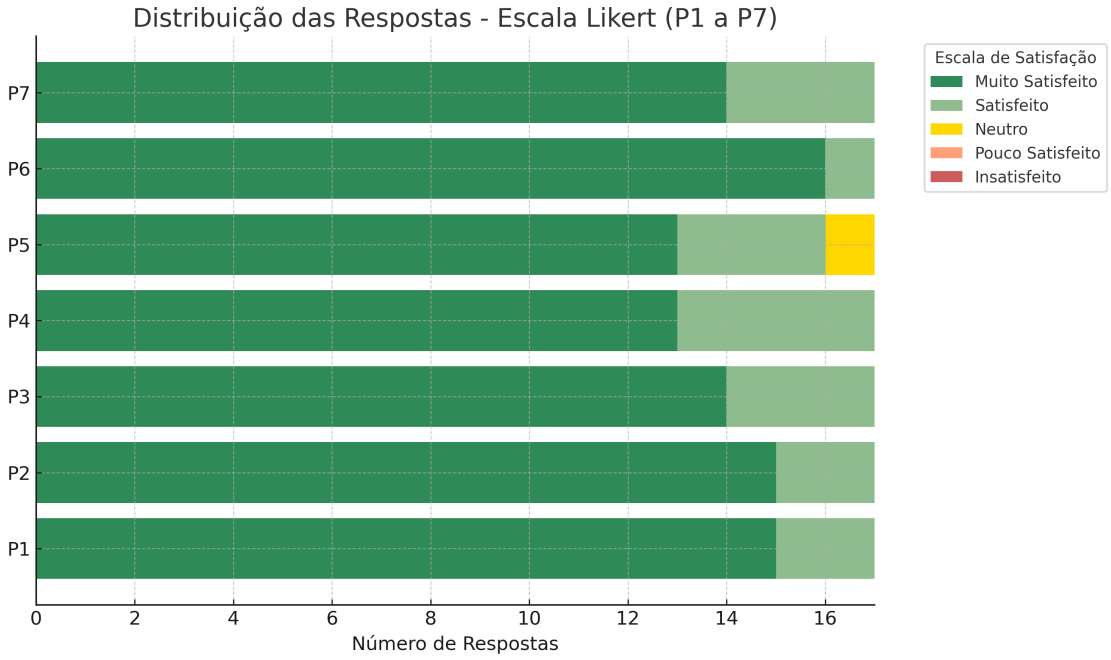

Figura 5.3: Resultado da avaliação.

De forma geral, observou-se uma predominância de respostas nas categorias "Muito Satisfeito" e "Satisfeito", o que indica uma aceitação positiva do sistema por parte dos participantes. Em particular, os itens relacionados à facilidade de uso e à clareza na visualização dos *tickets* adquiridos obtiveram altos índices de satisfação. A presença de respostas "Neutras" em algumas questões também sugere pontos de possível aprimoramento. Ressalta-se ainda que não foram registradas respostas nas categorias "Pouco Satisfeito" ou "Insatisfeito", o que reforça a percepção geral de que o sistema atende adequadamente às expectativas dos usuários.

As respostas à pergunta aberta tiveram como objetivo captar percepções mais detalhadas dos usuários sobre o sistema, permitindo comentários livres a respeito de pontos positivos, negativos ou sugestões de melhoria. A análise qualitativa dessas respostas evidenciou, de forma geral, uma recepção positiva quanto à proposta e à usabilidade da aplicação. Muitos participantes destacaram que o sistema é “prático e fácil de usar”, “fácil e intuitivo” e que “iria modernizar o processo de compra”. Um dos participantes comentou: “Adorei a proposta. Espero que vire realidade!”. Outro destacou de forma bem-humorada: “Eu adorei a capivara, e o sistema vai ajudar muito na fila!”, embora tenha pontuado que a conexão com a Internet nas proximidades do restaurante universitário poderia ser um desafio para sua utilização efetiva. Também foram registrados comentários demonstrando o desejo de uso real do sistema, como: “Muito bom! Eu quero poder usar de fato no IM.”

Ainda, durante a reunião em que o sistema foi apresentado, a responsável pelo restaurante demonstrou satisfação com a solução de forma geral, destacando sua praticidade e organização. Ela ressaltou, em especial, a funcionalidade de vendas, considerando-a simples e eficiente para o uso no dia a dia. Além disso, evidenciou a utilidade da funcionalidade de geração de relatórios, que permite acompanhar os registros e facilita a gestão do restaurante universitário. Por sua vez, os alunos demonstraram contentamento com o sistema, ressaltando a facilidade de uso no processo de compra dos *tickets*.

Capítulo 6

Conclusão

O presente trabalho teve como objetivo o estudo comparativo entre as arquiteturas monolítica e de microsserviços, destacando suas principais características, vantagens e desvantagens. A análise teórica foi seguida da aplicação prática dos conhecimentos adquiridos no desenvolvimento de um sistema *web* voltado à compra e utilização de *tickets* digitais no restaurante universitário da UFRRJ.

O sistema foi implementado utilizando a arquitetura monolítica, escolhida com base nas etapas do processo de Engenharia de *Software*, que indicaram ser a abordagem mais adequada ao contexto do problema. O estudo comparativo entre arquiteturas foi realizado previamente, e os resultados apontaram que a solução monolítica atendia melhor aos requisitos do sistema, foi possível observar que, embora a arquitetura de microsserviços ofereça escalabilidade e independência de componentes, a arquitetura monolítica se mostrou mais adequada ao problema proposto. Isso se deve ao fato de tratar-se de uma aplicação voltada a um nicho específico, com funcionalidades bem definidas e público-alvo restrito, o que favorece a adoção de uma arquitetura mais simples, centralizada e de fácil manutenção.

Por sua vez, a avaliação do sistema foi realizada por meio de uma pesquisa de satisfação aplicada aos alunos da universidade, público diretamente envolvido com o uso da solução. Os resultados da pesquisa foram amplamente positivos, indicando que o sistema atende bem às necessidades dos usuários.

Com base nos dados obtidos, acredita-se que a arquitetura escolhida está em conformidade com as diretrizes discutidas ao longo do estudo, demonstrando que decisões arquiteturais devem sempre considerar o escopo, o público-alvo e a complexidade do sistema. Espera-se que este trabalho possa servir de referência para futuros projetos acadêmicos ou profissionais que enfrentem desafios semelhantes no desenvolvimento de sistemas *web*.

6.1 Limitações e Trabalhos Futuros

A principal limitação encontrada no desenvolvimento do sistema foi a utilização do pagamento via Pix exclusivamente em ambiente de teste. Nesse contexto, as transações foram simuladas por meio de contas de teste, o que não permitiu validar a integração sob as condições reais de operação. Isso impossibilitou a verificação de aspectos críticos como segurança, conformidade regulatória e desempenho sob grande volume de transações. Além disso, o sistema pode apresentar falhas de performance ou segurança quando exposto a transações reais, uma vez que os testes não reproduzem completamente o impacto de pagamentos autênticos.

Além da limitação do pagamento via Pix em ambiente de teste, outra questão relevante foi a falta de coleta de *feedback* dos técnicos responsáveis pela operação do sistema. Embora tenha sido realizada uma pesquisa com os alunos, os profissionais técnicos, essenciais para a manutenção e ajustes do sistema, não foram ouvidos. A visão técnica é crucial para avaliar a usabilidade do sistema e a adequação da arquitetura às demandas operacionais reais. Sem esse *feedback*, não foi possível identificar potenciais falhas que poderiam impactar a performance e escalabilidade do sistema em produção, evidenciando a necessidade de incluir a pesquisa com os técnicos em futuras avaliações.

Considerando as limitações identificadas neste estudo, diversas possibilidades de aprimoramento e expansão do sistema desenvolvido podem ser exploradas em estudos futuros. Uma das principais propostas é a integração do sistema com o Sistema Integrado de Gestão de Atividades Acadêmicas (SIGAA), utilizado pela UFRRJ para

centralizar as informações acadêmicas dos alunos. Por meio dessa integração, seria possível validar automaticamente se um aluno possui matrícula ativa, eliminando a necessidade de autorização manual por parte da administração. Isso poderia ser viabilizado por meio do consumo seguro de uma API disponibilizada pelo SIGAA, tornando o processo mais eficiente e confiável.

Outra proposta relevante consiste na integração do sistema com dispositivos físicos de controle de acesso, como catracas eletrônicas. A ideia seria que o aluno, ao apresentar um *QR Code* gerado pelo sistema, pudesse ter sua entrada liberada de forma automática, com base na validação do *ticket* comprado previamente. Essa funcionalidade agregaria praticidade e segurança ao processo de entrada no restaurante universitário, automatizando completamente a verificação de *tickets*.

Além disso, uma evolução natural do sistema seria o desenvolvimento de uma versão *mobile*. Embora o sistema atual seja uma aplicação *web* responsiva, uma aplicação móvel dedicada poderia oferecer melhor desempenho, usabilidade aprimorada e a oportunidade de incorporar funcionalidades exclusivas, como notificações em tempo real, armazenamento local de dados e integração com serviços do dispositivo, como a câmera para leitura de *QR Code*.

Essas propostas visam não apenas aprimorar a experiência do usuário final, mas também aumentar a robustez e escalabilidade do sistema, tornando-o mais alinhado com as demandas tecnológicas contemporâneas.

Referências

AL-DEBAGY, O.; MARTINEK, P. A comparative review of microservices and monolithic architectures. In: IEEE. *2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI)*. [S.l.], 2018. p. 000149–000154.

AWATI, R. *monolithic architecture*. 2022. Disponível em: <https://www.techtarget.com/whatis/definition/monolithic-architecture>. Acesso em: 27 de outubro de 2023.

AWS. 2023. Disponível em: <https://aws.amazon.com/pt/compare/the-difference-between-monolithic-and-microservices-architecture/>. Acesso em: 3 de novembro 2023.

BASS, L.; CLEMENTS, P.; KAZMAN, R. *Software architecture in practice*. Boston: Addison-Wesley Professional, 2003.

BOOCH, G. *The unified modeling language user guide*. [S.l.]: Pearson Education India, 2005.

COMMITTEE, A. W. G. of the S. E. et al. Recommended practice for architectural description of software intensive systems. *IEEE Standards Department, Piscataway, New Jersey, USA*, 2000.

DAVIS, F. D. Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Quarterly*, v. 13, n. 3, p. 319–340, 1989.

DEITEL, H. M. et al. Java: como programar. Biblioteca Hernán Malo González, 2016.

DUBOIS, P. Mysql. Addison-Wesley, 2013.

DUCKETT, J. *HTML & CSS: design and build websites*. [S.l.]: Wiley Indianapolis, IN, USA:, 2011. v. 15.

FOWLER, M. *Microservices*. 2014. Disponível em: <https://martinfowler.com/articles/microservices.html>. Acesso em: 26 de Outubro 2023.

FOWLER, M. *Microservice Trade-Offs*. 2015. Disponível em: <https://martinfowler.com/articles/microservice-trade-offs.html>. Acesso em: 11 de Dezembro 2023.

LIKERT, R. A technique for the measurement of attitudes. *Archives of psychology*, 1932.

LUCIO, J. P. D. Análise comparativa entre arquitetura monolítica e de microsserviços. 2017.

MENDES, I. S. Arquitetura monolítica vs microsserviços: uma análise comparativa. 2021.

NEWMAN, S. *Building Microservices: Designing Fine-Grained Systems*. Boston: O'Reilly Media, 2015. ISBN 9781491950333.

NEWMAN, S. *Migrando Sistemas monolíticos para microsserviços*. São Paulo: Novatec, 2020.

PRESSMAN, R. S.; MAXIM, B. R. Engenharia de software-9. McGraw Hill Brasil, 2021.

PROVOS, N.; MAZIERES, D. A future-adaptable password scheme. *USENIX Annual Technical Conference*, 1999.

RICHARDS, M.; FORD, N. *Fundamentals of Software Architecture: An Engineering Approach*. Boston: O'Reilly Media, Incorporated, 2020. ISBN 9781492043454.

Thymeleaf. *What is Thymeleaf?* 2024. Disponível em: <https://www.thymeleaf.org/>.

VALENTE, M. T. *Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade*. Belo Horizonte: Editora: Independente, 2020.

VENKATESH, V.; DAVIS, F. D. A theoretical extension of the technology acceptance model: Four longitudinal field studies. *Management Science*, v. 46, n. 2, p. 186–204, 2000.

VENKATESH, V. et al. User acceptance of information technology: Toward a unified view. *MIS Quarterly*, v. 27, n. 3, p. 425–478, 2003.

WALLS, C. Spring boot in action. Simon and Schuster, 2015.

Apêndice A

Especificação dos casos de uso

Este apêndice apresenta os diagramas e descrições dos casos de uso desenvolvidos durante a elaboração do sistema.

Identificador	UC01
Nome	Cadastrar Usuário
Atores	Usuário
Descrição	O caso de uso é iniciado quando o usuário deseja ter uma conta no sistema.
Referências Cruzadas	RF01, RN01, RN02, RN03
Pré-condições	-
Pós-condições	O usuário é registrado no sistema.
Fluxo principal	1. O usuário solicita o cadastro. 2. O sistema exibe o formulário de cadastro. 3. O usuário preenche o formulário com matrícula, e-mail e nome completo. 4. O usuário envia o formulário. 5. O sistema valida os dados fornecidos. 6. O sistema recebe os dados do usuário. 7. Fim do caso de uso.
Fluxo alternativo	-
Fluxo de exceção	FE01: Usuário já cadastrado. 1. No passo 5 do fluxo principal, o sistema verifica a matrícula. 2. O sistema informa que matrícula já está cadastrada. 3. O caso de uso retorna ao passo 2 do fluxo principal. FE02: Usuário não preenche um ou mais campos de dados. 1. No passo 5 do fluxo principal, o sistema verifica a falta de dados. 2. O sistema informa que está faltando dados. 3. O caso de uso retorna ao passo 2 do fluxo principal.

Tabela A.1: Cadastrar Usuário

Identificador	UC02
Nome	Atualizar Cadastro
Atores	Usuário
Descrição	O caso de uso é iniciado quando o usuário deseja atualizar o cadastro no sistema.
Referências Cruzadas	RF03, RN03, UC03
Pré-condições	O usuário deve estar autenticado no sistema.(UC03)
Pós-condições	O cadastro do usuário é atualizado no sistema.
Fluxo principal	1. O usuário acessa a página de configurações do perfil. 2. O sistema exibe um formulário com as informações do usuário. 3. O usuário modifica as informações que deseja atualizar no formulário. 4. O usuário envia o formulário. 5. O sistema valida as informações fornecidas. 6. O sistema atualiza o cadastro do usuário. 7. Fim do caso de uso.
Fluxo alternativo	-
Fluxo de exceção	FE01: Informações inválidas. 1. No passo 4 do fluxo principal, o sistema verifica alterações inválidas. 2. Retorna ao passo 2 do fluxo princial.

Tabela A.2: Atualizar Cadastro

Identificador	UC03
Nome	Realizar *Login*
Atores	Usuário
Descrição	O caso de uso é iniciado quando o usuário deseja acessar o sistema.
Referências Cruzadas	RF02, RN05
Pré-condições	O usuário deve possuir cadastro validado pelo administrador.
Pós-condições	O usuário estará autenticado no sistema.
Fluxo principal	1. O usuário acessa a página de *login*. 2. O sistema exibe um formulário de *login* solicitando matrícula e senha. 3. O usuário preenche o formulário com matrícula e senha. 4. O usuário envia o formulário. 5. O sistema verifica as credenciais fornecidas. 6. O sistema autentica o usuário. 7. Fim do caso de uso.
Fluxo alternativo	-
Fluxo de exceção	FE01: Matrícula ou senha inválida. 1. No passo 5 do fluxo principal, o sistema verifica que a matrícula ou a senha não é válida. 2. O sistema informa que a matrícula ou a senha está incorreta. 3. O caso de uso retorna ao passo 2 do fluxo principal.

Tabela A.3: Realizar *Login*

Identificador	UC04
Nome	Visualizar Carteira
Atores	Usuário
Descrição	O caso de uso inicia quando o usuário reliza *login*(RF02).
Referências Cruzadas	RF06, RF02, RN07, UC03, UC05
Pré-condições	O usuário deve estar autenticado no sistema.(UC03)
Pós-condições	O sistema apresenta as informações do usuários
Fluxo principal	1. O usuário acessa a página de visualização da carteira virtual. 2. O sistema exibe as informações da carteira virtual do usuário, com o saldo atual e opções de compra. 3. Fim do caso de uso.
Fluxo alternativo	FA01: O usuário deseja realizar compra de *tickets*. 1. No passo 2 do fluxo principal, o usuário seleciona a opção para compra de *tickets*. 2. O sistema encaminha o usuário para o passo 2 do fluxo principal do caso de uso UC05.
Fluxo de exceção	-

Tabela A.4: Visualizar Carteira

Identificador	UC05
Nome	Comprar *Tickets*
Atores	Usuário, API externa
Descrição	O caso de uso é iniciado quando o usuário deseja comprar *tickets*.
Referências Cruzadas	RF04, RN06, RN09, RN10, UC03, UC07
Pré-condições	O usuário deve estar autenticado no sistema.(UC03)
Pós-condições	O usuário recebe a quantidade de *ticket* solicitado.
Fluxo principal	1. O usuário solicita a compra de *tickets*. 2. O sistema exibe opções de café da manhã e almoço/janta para a compra de *tickets*. 3. O usuário seleciona a opção desejada e a quantidade. 4. O sistema calcula o total da compra com base nas seleções do usuário. 5. O usuário confirma a compra. 6. O usuário efetua o pagamento. 7. O sistema envia os dados de pagamentos para a API de pagamento externa. 8. A API externa processa a transação. 9. O sistema recebe a confirmação da transação. 10. O sistema informa o status da compra. 11. Fim do caso de uso.
Fluxo alternativo	FA01: O usuário cancela a compra. 1. No passo 4 do fluxo principal, o usuário desiste da compra. 2. O sistema solicita a confirmação. 3. O usuário confirma o cancelamento da compra. 4. Fim do caso de uso. FA02: Alterar a quantidade/tipo de *ticket*: 1. No passo 4 do fluxo principal, o usuário deseja alterar a quantidade de *tickets* ou o tipo. 2. Retorna ao passo 2 do fluxo princial.
Fluxo de exceção	FE01: Falha no pagamento. 1. No passo 8 do fluxo principal, a API falha em processar a transação. 2. O sistema informa a falha no pagamento. 3. Retorna ao passo 2 do fluxo princial.

Tabela A.5: Comprar *Tickets*

Identificador	UC06
Nome	Visualizar Histórico de Compras
Atores	Usuário
Descrição	O caso de uso inicia quando o usuário deseja visualizar o histórico de compras.
Referências Cruzadas	RF11, UC03
Pré-condições	O usuário deve estar autenticado no sistema.(UC03)
Pós-condições	-
Fluxo principal	1. O usuário acessa a página de dados do usuário e escolhe a opção de visualizar histórico de compras. 2. O sistema exibe o histórico de compras do usuário. 3. Fim do caso de uso.
Fluxo alternativo	-
Fluxo de exceção	-

Tabela A.6: Visualizar Histórico de Compras

Identificador	UC07
Nome	Registrar uso de *Ticket*
Atores	Técnico
Descrição	O caso de uso é iniciado quando o técnico deseja registrar o uso de um *ticket* pelo usuário.
Referências Cruzadas	RF05, RN08, UC03
Pré-condições	O usuário deve estar autenticado no sistema.(UC03)
Pós-condições	Um *ticket* é descontado da conta do usuário
Fluxo principal	1. O técnico acessa a página de descontar *tickets*. 2. O sistema solicita a matrícula do usuário para utilização de um *ticket* 3. O técnico informa a matrícula do usuário e confirma. 4. O Sistema exibe opções de *ticket* para descontar. 5. O técnico escolhe a opção e confirma. 6. O sistema debita o *ticket* do usuário. 7. O sistema registra o uso do *ticket* do usuário. 8. Fim do caso de uso.
Fluxo alternativo	-
Fluxo de exceção	FE01: Conta não possui *ticket*. 1. No passo 4 no fluxo principal, o sistema não debita pela falta de saldo. 2. O sistema apresenta uma mensagem informando a falta de saldo. 3. Fim do caso de uso.

Tabela A.7: Registrar uso de *Ticket*

Identificador	UC08
Nome	Visualizar histórico de Vendas
Atores	Técnico
Descrição	O caso de uso inicia quando o técnico deseja visualizar o histórico de vendas realizadas.
Referências Cruzadas	RN09, RF12, RF13, UC03
Pré-condições	O usuário deve estar autenticado no sistema. (UC03)
Pós-condições	-
Fluxo principal	1. O técnico acessa a página de histórico de vendas. 2. O sistema exibe o histórico de vendas do realizadas. 3. Fim do caso de uso.
Fluxo alternativo	FA01: Emitir relatório de vendas. 1. No passo 2 do fluxo principal, o técnico deseja exportar as informações das vendas realizadas. 2. O técnico solicita a exportação do histórico de vendas em PDF ou XLSX. 3. O sistema emite o relatório de vendas no formato desejado. 4. Fim do caso de uso.
Fluxo de exceção	-

Tabela A.8: Visualizar Histórico de Vendas

Identificador	UC09
Nome	Vender *Ticket*
Atores	Técnico
Descrição	O caso de uso inicia quando o técnico deseja vender *tickets* para o usuário.
Referências Cruzadas	RF08, UC03
Pré-condições	O usuário deve estar autenticado no sistema.(UC03)
Pós-condições	O(s) *ticket(s)* são adicionados na carteira do usuário.
Fluxo principal	1. O técnico acessa a página de vender *tickets*. 2. O sistema exibe opções de café da manhã e refeição para a compra de *tickets*. 3. O técnico seleciona a opção desejada, informando a matrícula do usuário e quantidade de *tickets*. 4. O sistema calcula o total da compra com base nas seleções do técnico e exibe o nome e a foto do usuário a receber a compra. 5. O técnico confirma a venda. 6. O sistema adiciona os *tickets* à carteira do usuário. 7. Fim do caso de uso.
Fluxo alternativo	FE01: Matrícula inválida. 1. No passo 3 do fluxo principal, o sistema verifica que a matrícula não é válida. 2. O sistema informa que a matrícula está incorreta. 3. O caso de uso retorna ao passo 2 do fluxo principal.
Fluxo de exceção	-

Tabela A.9: Vender *Ticket*

Identificador	UC10
Nome	Reembolsar *Ticket*
Atores	Técnico
Descrição	O caso de uso inicia quando o técnico deseja reembolsar *ticket* para o usuário.
Referências Cruzadas	RF12, UC03
Pré-condições	O usuário deve estar autenticado no sistema.(UC03)
Pós-condições	O *ticket* é adicionado na carteira do usuário.
Fluxo principal	1. O técnico acessa a página de reembolsar *tickets*. 2. O sistema solicita a matrícula do usuário. 3. O técnico informa a matrícula do usuário. 4. O sistema exibe informações do usuário e opção de reembolsar *ticket*. 5. O técnico seleciona o *ticket* para rembolsar e confirma. 6. O sistema adiciona o *ticket* na carteira do usuário. 7. Fim do caso de uso.
Fluxo alternativo	-
Fluxo de exceção	FE01: Matrícula inválida. 1. No passo 3 do fluxo principal, o sistema verifica que a matrícula não é válida. 2. O sistema apresenta uma mensagem informando que a matrícula está incorreta. 3. O caso de uso retorna ao passo 2 do fluxo principal.

Tabela A.10: Reembolsar *Ticket*

Identificador	UC11
Nome	Validar Usuário
Atores	Administrador
Descrição	O caso de uso inicia quando o administrador deseja validar o cadastro de um usuário no sistema.
Referências Cruzadas	RF08, RN03
Pré-condições	O usuário deve ter preenchido o formulário de cadastro com as informações obrigatórias, como nome, e-mail, matrícula e senha.
Pós-condições	Usuário é validado e seu acesso ao sistema é liberado.
Fluxo principal	1. O administrador solicita a validação do usuário. 2. O sistema exibe os usuários cadastrados que estão pendentes de validação. 3. O administrador seleciona o usuário desejado. 4. O sistema solicita confirmação do usuário. 5. O administrador confirma a validação. 7. O sistema cadastra o usuário validado pelo administrador. 8. O sistema envia um e-mail informando que seu acesso ao sistema foi concedido. 9. Fim do caso de uso.
Fluxo alternativo	FA01: Dados inconsistentes. 1. No passo 3 do fluxo principal a matrícula ou nome completo do usuário estão incorretos. 2. O administrador informa que o usuário não será validado. 3. O sistema solicita confirmação da não validação do usuário. 4. O adminstrador confirma a não validação do usuário, informando o motivo da não validação. 3. O sistema envia um e-mail informando ao usuário que o nome ou a matrícula está errado. 4. Fim do caso de uso.
Fluxo de exceção	-

Tabela A.11: Validar Usuário

Identificador	UC12
Nome	Editar Usuário
Atores	Administrador
Descrição	O caso de uso inicia quando o administrador deseja alterar informações cadastradas do usuário.
Referências Cruzadas	RF10, UC03
Pré-condições	O usuário deve estar autenticado no sistema.(UC03)
Pós-condições	Dados do usuário alterado.
Fluxo principal	1. O administrador acessa a página de alterar usuário. 2. O sistema solicita a matrícula do usuário. 3. O técnico informa a matrícula. 4. O sistema exibe informações do usuário. 5. O administrador altera as informações desejadas e confirma. 6. O sistema salva as alterações. 7. Fim do caso de uso.
Fluxo alternativo	-
Fluxo de exceção	FE01: Informações inválidas. 1. No passo 5 do fluxo principal, o sistema verifica alterações inválidas. 2. Retorna ao passo 4 do fluxo princial.

Tabela A.12: Editar Usuário

Apêndice B

Formulário aplicado para coleta de dados

O formulário a seguir foi elaborado com o intuito de avaliar a experiência dos usuários com o sistema desenvolvido. As perguntas são fechadas, do tipo escala de satisfação, e foram organizadas para cobrir aspectos fundamentais como usabilidade, funcionalidade e estética da aplicação.

Formulário de Feedback sobre o sistema

Olá!

Seja bem-vindo(a)!

Este sistema foi desenvolvido com foco no Trabalho de Conclusão de Curso (TCC), a partir do estudo de uma arquitetura teórica e sua implementação em um sistema real. Seu objetivo é oferecer uma solução prática, rápida e segura para a compra e o gerenciamento de tickets, permitindo ao usuário visualizar o histórico de compras e acessar informações do perfil..

Para continuarmos evoluindo e oferecendo a melhor experiência possível, gostaríamos de contar com a sua opinião. Preparamos um breve formulário para entender como o sistema está atendendo às suas expectativas.

Antes de responder às perguntas, utilize o sistema por alguns minutos, explore as funcionalidades disponíveis e avalie a usabilidade e eficiência da plataforma.

Sua participação é muito importante para nós. Agradecemos desde já pelo seu tempo e contribuição!

Link: site

Sugestão de uso:

1. Acesse o sistema através deste link: .
2. Clique em **cadastre-se** para realizar o cadastro.
3. Faça login com a sua matrícula e senha.
4. Acesse a seção **Compras**, escolha o tipo e a quantidade de ticket que deseja e realize uma compra.
5. Acesse a seção **Carteira** para verificar a quantidade e o tipo de tickets disponíveis na sua conta.
6. Acesse a seção **Usuário** e clique em **Histórico de Compras** para verificar suas compras realizadas.

* Indica uma pergunta obrigatória

1. Em relação à experiência geral de uso, você considera que o software atendeu às suas expectativas? *

Marcar apenas uma oval.

- ◯ Muito Satisfeito
- ◯ Satisfeito
- ◯ Neutro
- ◯ Pouco Satisfeito
- ◯ Insatisfeito

2. Em relação à sua satisfação com o processo de compra de tickets, você acha que ele é rápido e fácil de realizar? *

Marcar apenas uma oval.

- ◯ Muito Satisfeito
- ◯ Satisfeito
- ◯ Neutro
- ◯ Pouco Satisfeito
- ◯ Insatisfeito

3. Em relação à sua satisfação com a visualização e o gerenciamento de tickets adquiridos, você acha que o sistema oferece uma forma fácil de gerenciar suas compras? *

Marcar apenas uma oval.

- ◯ Muito Satisfeito
- ◯ Satisfeito
- ◯ Neutro
- ◯ Pouco Satisfeito
- ◯ Insatisfeito

4. Em relação à sua satisfação com a facilidade de aprendizado, como você avaliaria a experiência de aprender a usar o software? *

Marcar apenas uma oval.

- Muito Satisfeito
- Satisfeito
- Neutro
- Pouco Satisfeito
- Insatisfeito

5. Em relação à sua satisfação com a interface do software, você considera que ela é intuitiva e fácil de usar? *

Marcar apenas uma oval.

- Muito Satisfeito
- Satisfeito
- Neutro
- Pouco Satisfeito
- Insatisfeito

6. Em relação à sua satisfação com a utilidade do software, você considera que ele atende às suas necessidades? *

Marcar apenas uma oval.

- Muito Satisfeito
- Satisfeito
- Neutro
- Pouco Satisfeito
- Insatisfeito

7. Em relação à sua satisfação com o design e a estética do software, você acha que a aparência é agradável? *

Marcar apenas uma oval.

- ◯ Muito Satisfeito
- ◯ Satisfeito
- ◯ Neutro
- ◯ Pouco Satisfeito
- ◯ Insatisfeito

8. Comentários adicionais: