UNIVERSIDADE FEDERAL DE ALAGOAS
CAMPUS A. C. SIMÕES
INSTITUTO DE COMPUTAÇÃO
CURSO DE ENGENHARIA DE COMPUTAÇÃO

WILAMIS MICAEL DE ARAUJO AVIZ

Uma Linha de Produto de Software para Sistemas de Gerenciamento de Refeições utilizando uma Arquitetura Baseada em Microsserviços

MACEIÓ - AL
2024

WILAMIS MICAEL DE ARAUJO AVIZ

Uma Linha de Produto de Software para Sistemas de Gerenciamento de Refeições utilizando uma Arquitetura Baseada em Microsserviços

Trabalho de Conclusão de Curso apresentado ao Curso do Instituto de Computação da Universidade Federal de Alagoas, como requisito parcial à obtenção do título de Bacharelado em Engenharia de Computação.

Orientador: Prof. Dr. Arturo Hernandez Dominguez.

MACEIÓ - AL

2024

Catalogação na fonte
Universidade Federal de Alagoas
Biblioteca Central
Divisão de Tratamento Técnico
Bibliotecária: Helena Cristina Pimentel do Vale – CRB4 - 661

A959u Aviz, Wilamis Micael de Araujo.
Uma linha de produto de software para sistemas de gerenciamento de refeições utilizando uma arquitetura baseada em microsserviços / Wilamis Micael de Araujo Aviz. – 2024.
100 f.: il.

Orientador: Arturo Hernandez Dominguez.
Monografia (Trabalho de Conclusão de Curso em Engenharia de Computação) – Universidade Federal de Alagoas, Instituto de Computação. Maceió, 2024.

Bibliografia: f. 66-69.
Apêndices: f. 70-100.

1. Linhas de produto de software. 2. Sistema de agendamento de refeição. 3. Microsserviços. I. Título.

CDU: 004.41

Folha de Aprovação

UNIVERSIDADE FEDERAL DE ALAGOAS/UFAL
Instituto de Computação – IC
Campus A. C. Simões - Av. Lourival de Melo Mota, BL 12
Tabuleiro do Martins, Maceió/AL - CEP: 57.072-970
Telefone: (082) 3214-1401

Trabalho de Conclusão de Curso – TCC

Formulário de Avaliação

Curso: _____

Nome do Aluno
Wilamis Michael de Araujo Aviz

Nº de Matrícula
19210704-

Título do TCC (Tema)
"Uma Linha de Produto de Software para Sistemas de Gerenciamento de Refeições utilizando uma Arquitetura Baseada em Microsserviços"

Banca Examinadora

Arturo Hernández Domínguez
_____
Nome do Orientador

Documento assinado digitalmente
gov.br ARTURO HERNANDEZ DOMINGUEZ
Data: 04/12/2024 05:24:02-0300
Verifique em https://validar.iti.gov.br
_____
Assinatura

Reinaldo Cabral Silva Filho
_____
Nome do Professor

Documento assinado digitalmente
gov.br REINALDO CABRAL SILVA FILHO
Data: 04/12/2024 11:20:05-0300
Verifique em https://validar.iti.gov.br

Patrick Henrique da Silva Brito_____
Nome do Professor

Documento assinado digitalmente
gov.br PATRICK HENRIQUE DA SILVA BRITO
Data: 04/12/2024 16:41:37-0300
Verifique em https://validar.iti.gov.br

Data da Defesa	Nota Obtida
03 /12 / 2024	9,0 (nove inteiros)

Observações

Condicionada a nota final à entrega da versão final após a atualização da monografia de TCC, com os comentários e sugestões dos membros da banca (Reinaldo Cabral Silva Filho e Patrick Henrique da Silva Brito)

Coordenador do Curso De Acordo

Documento assinado digitalmente
gov.br JOBSON DE ARAUJO NASCIMENTO
Data: 05/12/2024 10:42:35-0300
Verifique em https://validar.iti.gov.br
_____
Assinatura

Curso de ENGENHARIA DE COMPUTAÇÃO - CAMPUS MACEIÓ

AGRADECIMENTOS

Primeiramente, agradeço a Deus, que me deu forças e sabedoria para superar todos os desafios encontrados ao longo desta jornada. Agradeço especialmente a minha mãe, ao meu namorado que esteve ao meu lado, aos meus colegas, a meu orientador e aos professores.

RESUMO

Empresas de software enfrentam uma crescente necessidade de desenvolver soluções de forma cada vez mais rápida e eficiente, visando atender à demanda por automatização de tarefas e otimização de processos. Para ter agilidade na construção de softwares, técnicas de reuso e a adoção de Linhas de Produto de Software (LPS) mostram-se estratégias promissoras. A LPS contém um código base que será comum para toda a família de software e as partes variantes. As variações nos componentes permitem a criação de novos produtos. A LPS é implementada em um domínio, neste trabalho o domínio foi para sistema de agendamento de refeição utilizando a arquitetura de microsserviços. O objetivo foi facilitar a construção de sistemas personalizados. As tecnologias usadas na implementação incluem *Nest.js* e *React.js*, utilizando *TypeScript* como linguagem principal. Para validar a instância da LPS, foi desenvolvido um sistema de agendamento de refeições com alguns componentes e funcionalidades que a linha pode oferecer, sendo aplicado ao contexto da UFAL. Estudantes da universidade participaram de um estudo de caso preliminar, interagindo com o sistema e, posteriormente, respondendo a um questionário de satisfação. Os resultados do teste indicam que houve aceitação da solução pelos usuários, destacando a importância de a UFAL adotar um sistema de agendamento de refeições baseado na arquitetura da LPS desenvolvida.

Palavras-chave: linhas de produto de software; sistema de agendamento de refeição; microsserviços.

ABSTRACT

Software companies face a growing need to develop solutions faster and more efficiently, aiming to meet the demand for task automation and process optimization. To achieve agility in software development, reuse techniques and the adoption of Software Product Lines (SPL) have proven to be promising strategies. The SPL contains a code base that will be common to the entire software family and its variant parts. Variations in the components allow the creation of new products. The SPL is implemented in a domain; in this work, the domain was a meal scheduling system using the microservices architecture. The goal was to facilitate the construction of customized systems. The technologies used in the implementation include Nest.js and React.js, using TypeScript as the main language. To validate the SPL instance, a meal scheduling system was developed with some components and functionalities that the line can offer, and was applied to the context of UFAL. Students from the university participated in a preliminary case study, interacting with the system and, later, answering a satisfaction questionnaire. The test results indicate that the solution was accepted by users, highlighting the importance of UFAL adopting a meal scheduling system based on the LPS architecture developed.

Keywords: software product lines; meal scheduling system; microservices.

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Program Interface*
CIED	Coordenadoria Institucional de Educação à Distância
CSS	*Cascading Style Sheets*
FODA	*Feature Oriented Domain Analysis*
HTML	*HyperText Markup Language*
IC	Instituto da Computação
JSON	*JavaScript Object Notation*
REST	*Representational State Transfer*
RU	Restaurante Universitário
SCSS	*Sassy Cascading Style Sheets*
SIGAA	Sistema Integrado de Gestão de Atividades Acadêmicas
SQL	*Structured Query Language*
TCP	*Transmission Control Protocol*
UML	*Unified Modeling Language*

SUMÁRIO

1 INTRODUÇÃO

Atualmente, o uso de sistemas e softwares é amplamente adotado por empresas para automatizar tarefas diversas, substituindo o trabalho manual por soluções digitais com o intuito de melhorar a eficiência e produtividade.

A indústria de software está cada vez mais globalizada, o que pressiona esse setor relativamente jovem a atender altas expectativas e uma demanda crescente por produtos de qualidade. Diversas pesquisas estão propondo soluções, especialmente onde os desafios práticos impactam a engenharia de software. Esses desafios se intensificam com a subutilização de ferramentas que poderiam otimizar o desenvolvimento (TUAPE et al., 2022).

No entanto, o desenvolvimento de software enfrenta desafios significativos, como o esforço repetitivo em tarefas comuns, a baixa reutilização sistemática de componentes existentes e uma crescente demanda por soluções personalizadas. Essas questões não apenas aumentam o custo e o tempo de desenvolvimento, mas também dificultam a manutenção e evolução dos sistemas.

Como forma de otimizar o desenvolvimento de sistemas, dentro da área de engenharia de software temos o reuso de software que é uma abordagem que sugere que soluções já aplicadas em projetos anteriores possam ser aproveitadas no desenvolvimento de novos projetos (PRESSMAN & MAXIM, 2016).

No contexto de reuso, temos a origem de LPS sendo uma abordagem que aproveita não apenas o reuso de código, mas também de diversos artefatos relacionados ao desenvolvimento de software. Ela funciona como uma "fábrica" para a criação de uma família de sistemas relacionados, permitindo que características comuns e variáveis sejam geridas e reaproveitadas de maneira estruturada. Dessa forma, facilita o desenvolvimento de novos produtos dentro de uma mesma linha, oferecendo flexibilidade e eficiência na criação de soluções específicas para diferentes necessidades, mas baseadas em uma infraestrutura comum (CLEMENTS & NORTHROP, 2001).

Este trabalho de conclusão de curso busca combinar os principais benefícios de uma LPS com a arquitetura de microsserviços. Assim, foi desenvolvida uma LPS com essa arquitetura para um sistema de agendamento de refeições, permitindo que produtos de software sejam construídos de acordo com as necessidades institucionais por meio da seleção de *features*. Para validação de uma instância, foi gerado um

sistema com algumas features disponíveis da linha, o qual foi positivamente avaliado pelo público participante em um estudo de caso preliminar.

1.1 JUSTIFICATIVA

Discutir sobre o uso de uma LPS para sistemas de gerenciamento de refeições utilizando uma arquitetura baseada em microsserviços é relevante devido à crescente diversidade de necessidades de instituições e organizações no gerenciamento desses processos. Uma abordagem tradicional, como a criação de um aplicativo único, pode atender a casos específicos, mas carece da flexibilidade proporcionadas por uma LPS, que permite a customização e reutilização de componentes em diferentes contextos.

Além disso, as necessidades de gerenciamento variam significativamente entre tipos de instituições. Por exemplo, universidades podem demandar um sistema que integre o controle de refeições ao gerenciamento acadêmico e financeiro, enquanto restaurantes populares podem priorizar funcionalidades como a simplificação de cadastros e monitoramento de consumo. Ao adotar uma LPS, é possível atender a essas diferentes demandas de maneira eficiente, mantendo um equilíbrio entre personalização e controle de custos. A utilização de uma arquitetura de microsserviços permite modularidade e adaptabilidade, características fundamentais para o desenvolvimento de sistemas que atendam a realidades específicas, como universidades, onde o fluxo de estudantes demanda um sistema ágil e seguro.

Para tanto, este trabalho visa desenvolver uma LPS que possibilite a customização dos sistemas de refeição para diversos contextos institucionais. Ao final, pretende-se, com base nos resultados obtidos, propor a adoção do sistema pela UFAL, contribuindo significativamente para a melhoria dos processos de controle de refeições na instituição.

1.2 OBJETIVOS

1.2.1 Objetivo geral

Desenvolver uma LPS para sistemas de gerenciamento de refeições utilizando uma arquitetura baseada em microsserviços. Essa LPS deverá possibilitar a rápida construção de sistemas personalizados, no domínio estabelecido.

1.2.2 Objetivos específicos

- Identificar e analisar os requisitos específicos de sistemas de gerenciamento

de refeições, com foco em uma arquitetura de microsserviços;

- Projetar uma LPS que permita a customização de sistemas de refeições para diversos contextos, assegurando modularidade e flexibilidade;
- Gerar um sistema de agendamento de refeições para a UFAL utilizando a LPS desenvolvida, garantindo a adequação do sistema às demandas institucionais;
- Avaliar uma instância da LPS para avaliar o potencial de utilidade na perspectiva do usuário final.

1.3 METODOLOGIA

Este trabalho utiliza uma abordagem quantitativa, conforme descrito por SOARES (2019, p.164): “No tocante aos métodos de pesquisa quantitativa, estes são utilizados quando se quer medir opiniões, reações, sensações, hábitos e atitudes etc.”.

Inicialmente, foi desenvolvida a LPS para o domínio de sistema de agendamento de refeições. Em seguida, a partir da LPS, foi gerada uma instância de produto com o intuito de obter a percepção do usuário final por meio de um estudo de caso preliminar. Para isso, era explicado aos participantes acerca da pesquisa e do sistema, eles testavam o fluxo navegando entre os recursos disponíveis como: agendamento de refeição, consulta de cardápio, avaliação de refeição, editar perfil de acesso.

Após o uso do sistema, eles responderam a um questionário via *Google Forms*, onde relataram suas impressões sobre a aplicação. Essa abordagem permitiu coletar dados para obter a percepção dos usuários quanto à instancia do sistema. Os resultados são apresentados em formatos de gráfico de pizza e em barra, juntamente com o total de participantes e o percentual de cada resposta. As perguntas do formulário foram elaboradas baseado em perguntas utilizadas pelos autores BRITO (2020) e SIQUEIRA (2020).

Além disso, a pesquisa classifica-se como tecnológica, pois visa à criação de um produto tangível, o que se alinha à definição de SOUZA et al. (2013, p. 14):

> A pesquisa tecnológica parte de um conhecimento pré-existente e, através da pesquisa e/ou experiência prática, busca a produção de novos materiais, produtos e aparelhagens, novos processos, sistemas e serviços ou aperfeiçoamento de sistemas, processos já existentes.

Nesse contexto, o desenvolvimento de uma LPS para sistemas de gerenciamento de refeições com uma arquitetura baseada em microsserviços

representa a aplicação e adaptação de conceitos avançados de engenharia de *software* para a construção de um sistema tangível.

A LPS proporciona uma plataforma flexível e escalável, permitindo a geração de soluções específicas para o gerenciamento de refeições. Essa abordagem, ao fazer uso da modularidade dos microsserviços e seleção de features, permite criar e configurar diferentes variações do sistema de acordo com necessidades específicas, como a inclusão de funcionalidades de pagamento ou gestão de avaliação dos usuários. Assim, o projeto não apenas inova ao aplicar a tecnologia LPS a um contexto específico de *software*, mas também aprimora e aperfeiçoa processos de desenvolvimento, resultando em um produto que agrega praticidade e customização ao gerenciamento de refeições, alinhando-se com o propósito da pesquisa tecnológica de desenvolver soluções concretas e aprimoradas.

A engenharia de domínio da LPS foi realizada a partir de estudo dos trabalhos relacionados (ver Seção 3).

1.4 ORGANIZAÇÃO DO TRABALHO

O trabalho está dividido em seis capítulos, organizados da seguinte forma:

No Capítulo II, aborda-se a fundamentação teórica que sustenta o desenvolvimento do sistema. Neste capítulo, são discutidos conceitos essenciais como LPS, arquitetura de microsserviços e sistemas de gerenciamento de refeições, fornecendo a base conceitual necessária para a construção da solução.

O Capítulo III é dedicado à revisão de trabalhos relacionados, onde serão apresentados estudos e sistemas que serviram como referência ou inspiração para o desenvolvimento do *software* proposto. Esta seção busca contextualizar o trabalho dentro do estado da arte.

No Capítulo IV, é apresentada a LPS proposta. Serão descritos os principais aspectos dessa linha de produto, como a modelagem de domínio, a variabilidade que ela oferece, o diagrama de features, as tecnologias utilizadas no desenvolvimento frontend, backend e na persistência de dados, bem como a arquitetura do sistema. Além disso, o capítulo detalha o processo de criação de um produto específico (aplicação) a partir da linha de produto desenvolvida.

O Capítulo V apresenta um estudo de caso, utilizando o sistema de gerenciamento de refeições criado através da LPS proposta, para ser utilizado no contexto da Universidade Federal de Alagoas (UFAL), juntamente com as telas do

sistema. Serão apresentados os resultados e discussões obtidos através de um estudo de caso preliminar.

Por fim, no Capítulo VI, são apresentadas as conclusões do trabalho, incluindo um resumo dos principais resultados alcançados, as limitações encontradas e sugestões para futuras pesquisas e implementações.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados os conceitos necessários ao entendimento deste trabalho.

A Seção 2.1 aborda as Linhas de Produto de *Software*, destacando suas principais características e benefícios. A Seção 2.2 introduz o conceito de microsserviços, discutindo sua arquitetura e vantagens em relação a sistemas monolíticos. Por fim, a Seção 2.3 apresenta os Sistemas de Gerenciamento de Refeições, de outras Universidades e Institutos Federais do Brasil.

2.1 Linhas de produto de *software*

2.1.1 Definição

Uma Linha de Produto de *Software* (LPS), segundo CLEMENTS & NORTHROP (2001), é uma coleção de sistemas que fazem uso intensivo de *software*, compartilhando características comuns e gerenciadas, atendendo às demandas de um segmento específico do mercado ou de uma missão. Esses sistemas são criados a partir de um conjunto comum de ativos principais e seguem um processo previamente estabelecido.

POHL, BOCKLE e LINDEN (2005, p.4) em sua obra intitulada de *Software Product Line Foundations, Principles, and Techniques*, define linha de produto de *software* como "um paradigma de desenvolvimento de aplicações usando plataformas e customização em massa". As áreas de contribuições da LPS ocorrem em quatro pilares distintos: negócios, arquitetura, processo e organização. Autores como LINDEN, SCHMID e ROMMES (2007) e POHL, BOCKLE e LINDEN (2005) definem que:

- **Área de negócios**: O campo de negócios refere-se à estratégia institucional, que define como a LPS será aplicada no desenvolvimento de *softwares* voltados a um domínio específico. A escolha das áreas envolvidas na linha de produtos é algo peculiar a cada organização, considerando o mercado que se deseja atingir e os produtos que serão desenvolvidos para atendê-lo.
- **Área de arquitetura**: A arquitetura lida com a estrutura interna do sistema. No contexto de uma LPS, ela é composta por elementos comuns que podem ser reutilizados, como o código-fonte, promovendo eficiência.
- **Área de processo**: A LPS também influencia o processo de desenvolvimento, definindo como serão realizadas as etapas de análise de requisitos,

implementação, entre outras. O reaproveitamento da arquitetura acontece de maneira indireta para os desenvolvedores, mas os ganhos em termos de economia de tempo e esforço são significativos.

- **Área de organização**: A LPS impacta a organização das equipes de trabalho, permitindo a definição clara das responsabilidades de cada membro, dividindo tarefas de acordo com funcionalidades. Isso possibilita que o desenvolvimento ocorra em partes variáveis e com funcionalidades comuns aos produtos da mesma linha, permitindo a especialização de produtos para casos específicos e o reaproveitamento por outros membros da família de produtos. Embora as mudanças necessárias para a adoção de uma LPS possam gerar um custo inicial elevado, elas trazem benefícios ao longo do tempo.

Essa divisão evidencia que a implementação de uma LPS não é apenas uma questão técnica, mas também uma decisão estratégica e organizacional, com impactos que se refletem em diversas áreas da empresa, ampliando sua capacidade de inovação e competitividade no mercado.

2.1.2 Motivação para utilização de uma LPS em projetos

O uso de LPS traz diversos benefícios para o desenvolvimento de produtos de *software*. De acordo com APEL et al. (2013), podemos destacar os seguintes benefícios:

- **Personalização do produto de *software***: A abordagem de linhas de produto para o desenvolvimento de *software* permite uma maior customização para clientes específicos. Em vez de oferecer um produto único ou um conjunto limitado de versões pré-definidas, como as edições *community*, *professional* e *enterprise*, o fornecedor pode criar uma gama completa de produtos ajustados às necessidades individuais de cada cliente.
- **Economia nos custos**: Fornecedores de linhas de produtos podem economizar ao desenvolver partes reutilizáveis que podem ser combinadas de diferentes formas, em vez de criar cada produto do zero. Embora exija um investimento inicial maior para projetar essas partes e suas variações, a economia é significativa a longo prazo, especialmente quando há demanda por vários produtos personalizados.
- **Qualidade melhorada**: A produção em massa eleva a qualidade dos produtos, pois suas peças são padronizadas e passam por verificações e testes frequentes,

especialmente as mais utilizadas. Isso resulta em produtos mais estáveis e confiáveis do que os feitos manualmente. Além disso, as peças podem ser reutilizadas em diferentes produtos, o que diminui o esforço de desenvolvimento e amplia a capacidade de atender diversos clientes.

- **Agilidade na entrega do *software*:** A LPS permite selecionar componentes predefinidos para atender às necessidades do cliente, facilitando a produção rápida de *software* com funcionalidades já existentes. Mesmo para novas demandas, o processo é ágil devido à reutilização de partes planejadas. Entretanto, isso exige um investimento inicial significativo para preparar os componentes reutilizáveis.

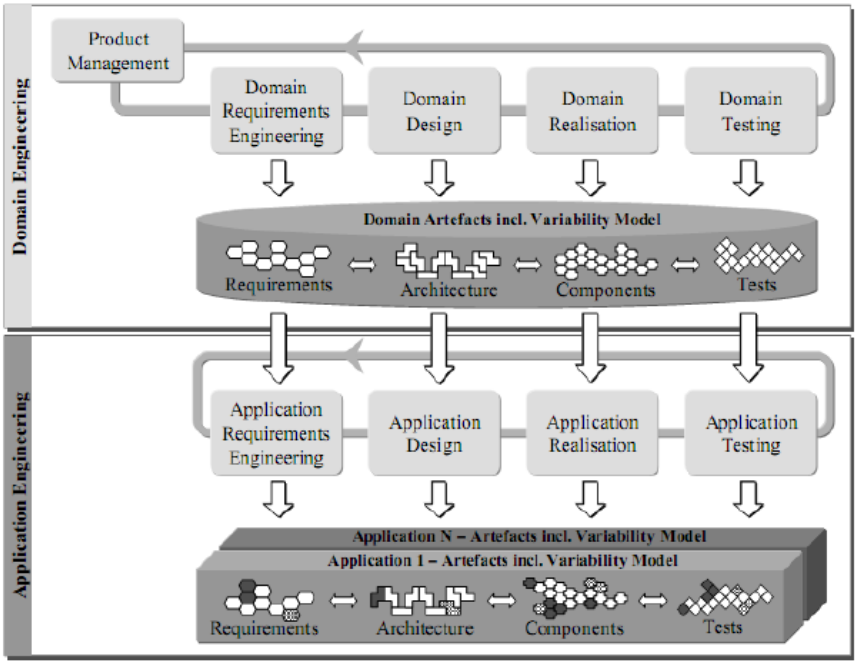
2.1.3 Processos de desenvolvimento de uma linha de produto de *software*

Existem dois processos quando falamos no paradigma de desenvolvimento de *software* com o método de LPS (POHL, BOCKLE e LINDEN, 2005):

1. **Engenharia de domínio**: Responsável por criar a base para a reutilização, definindo os elementos comuns e as variações na linha de produtos. A plataforma abrange diversos tipos de artefatos de *software*, como requisitos, design, arquitetura de *software* (artefato principal) e testes, entre outros, também conhecidos como ativos principais. Além disso, inclui os componentes de *software*, que são utilizados para compor a arquitetura da linha de produto, armazenados e organizados em um repositório de componentes de *software*.
2. **Engenharia de aplicação**: Esse processo é encarregado de gerar aplicações específicas a partir da plataforma definida pela engenharia de domínio. Ele se baseia no reuso de *software*, principalmente nos artefatos reutilizáveis, como a arquitetura da LPS e os componentes de *software* criados na engenharia de domínio. A arquitetura reutilizável é customizada (configurada) de acordo com os requisitos da aplicação. Caso surjam novos requisitos que não sejam contemplados pela arquitetura existente, novos componentes de software devem ser projetados e desenvolvidos para atender às demandas da nova aplicação.

Essa separação busca promover a variabilidade, permitindo que um processo se concentre na construção de uma plataforma robusta, enquanto o outro foca no desenvolvimento ágil de aplicações específicas. Para garantir a eficácia, é fundamental que ambos os processos trabalhem de forma integrada e colaborativa. Abaixo, na Figura 1, é possível observar como esse processo funciona:

Figura 1 - Framework de linhas de produto de *software*: fluxo da engenharia de domínio e aplicação



Fonte: POHL, BOCKLE e LINDEN (2005).

2.1.3.1 Engenharia de domínio

Os principais objetivos ligados ao processo de engenharia de domínio incluem:

- Identificar os aspectos comuns e as variações da linha de produtos de *software*;
- Estabelecer o conjunto de aplicações para as quais a linha de produtos de *software* será desenvolvida, ou seja, definir seu escopo;
- Criar e desenvolver artefatos reutilizáveis que atendam às necessidades de variabilidade.

2.1.3.2 Engenharia de aplicação

Os principais objetivos ligados ao processo de engenharia de aplicação incluem:

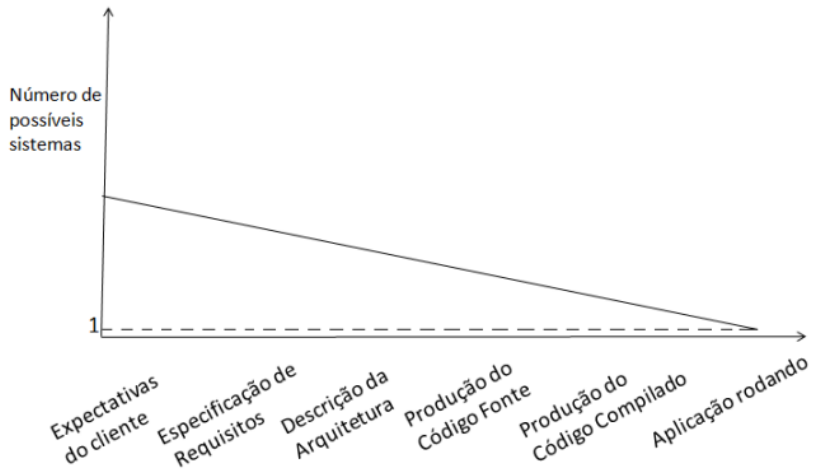
- Maximizar a reutilização dos ativos de domínio ao definir e desenvolver um aplicativo de linha de produtos;
- Aproveitar a comunalidade e a variabilidade da linha de produtos de *software* durante o desenvolvimento do aplicativo;
- Registrar os artefatos do aplicativo, como requisitos, arquitetura, componentes e testes, estabelecendo uma conexão com os artefatos de domínio;
- Ajustar a variabilidade conforme as necessidades do aplicativo, desde os requisitos até a arquitetura, componentes e casos de teste;
- Avaliar os impactos das diferenças entre os requisitos do aplicativo e do domínio

na arquitetura, componentes e testes.

2.1.4 Variabilidade

O conceito de variabilidade refere-se às opções de modificar ou adaptar um sistema. A Figura 2, demonstra como a variabilidade de um sistema de *software* se reduz ao longo de seu desenvolvimento. No início do processo, há diversas possibilidades, pois as restrições ainda são poucas. No entanto, à medida que o projeto avança e o sistema é construído, essas opções se tornam cada vez mais limitadas, até que, na fase de execução, resta apenas um sistema final. Em cada etapa do desenvolvimento, são tomadas decisões de design, e cada decisão diminui a quantidade de sistemas possíveis. (SILVA, SILVEIRA NETO e GARCIA, 2011).

Figura 2 - Quantidade de produtos possíveis



Fonte: SILVA, SILVEIRA NETO e GARCIA (2011).

A variabilidade é uma característica essencial dos artefatos de domínio em linhas de produto de *software*, pois possibilita maior suporte ao desenvolvimento e reutilização desses artefatos. Ela é incorporada durante o gerenciamento do produto, quando são identificados os recursos comuns e variáveis da LPS.

2.1.4.1 Pontos de variabilidade

Um ponto de variabilidade refere-se a um aspecto de variação funcional em um elemento de *software*. Ele define um conjunto de possíveis alternativas, além do mecanismo de variabilidade utilizado para sua implementação e o momento em que essas alternativas serão ativadas, como na instanciação da arquitetura do produto, compilação ou execução do *software* (SILVA, SILVEIRA NETO e GARCIA, 2011).

Os pontos de variabilidade representam um subconjunto das possíveis variações de um determinado domínio no mundo real, sendo essenciais para a implementação de um *software* específico. Eles capturam a variabilidade dentro de artefatos de domínio, incorporando informações contextuais relevantes para garantir flexibilidade no desenvolvimento (POHL, BOCKLE e LINDEN, 2005).

Para BABAR, CHEN e SHULL (2010), os pontos de variabilidade podem ser representados em diferentes níveis de abstração:

- **Descrição da arquitetura**: Normalmente, o sistema é representado por uma combinação de documentos de design de alto nível, linguagens específicas de descrição de arquitetura e documentação em texto;
- **Documentação com diagramas**: Neste estágio, o sistema pode ser representado usando diferentes notações da UML;
- **Código-fonte**: Nessa etapa, o sistema é descrito completamente por meio do código-fonte;
- **Código compilado**: O código-fonte é transformado em código compilado por meio de um compilador. O resultado dessa compilação pode ser alterado por diretivas de pré-processamento antes de ser finalmente combinado;
- **Código ligado**: Durante a fase de ligação, os resultados da compilação são integrados. Isso pode ocorrer de forma estática (em tempo de compilação) ou dinâmica (em tempo de execução);
- **Código em execução**: Na fase de execução, o sistema já construído é inicializado e configurado. Diferente das fases anteriores, o sistema em execução é dinâmico e está sempre mudando.

2.1.4.2 Variante

Uma variante, em LPS, refere-se a uma representação de um objeto variável, identificando uma única opção dentro de um ponto de variação. Ela pode ser associada a outros artefatos, indicando que esses artefatos correspondem a uma opção específica, representando uma das possíveis instâncias de variação que um ponto de variabilidade pode gerar.

2.1.4.3 Identificação de pontos de variação e variantes

Os passos para estabelecer os pontos de variação e suas respectivas variantes são resumidos a seguir:

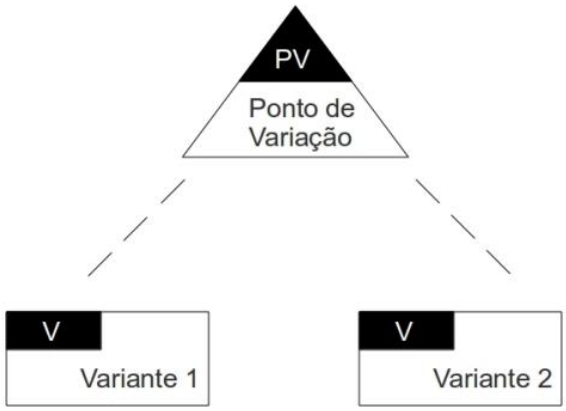
- Identificar um elemento do mundo real que seja passível de variação;

- Determinar um ponto de variação no contexto da LPS;
- Com o ponto de variação definido, o próximo passo é especificar as variantes. A inclusão de variantes fornece à aplicação instâncias específicas.

2.1.4.4 Diagrama de variabilidade

Um diagrama de variabilidade (Figura 3) é utilizado para representar visualmente onde ocorre a variação em um sistema e quais são as possíveis alternativas ou variantes. Nele, o Ponto de Variação é representado no topo de um triângulo, indicando uma decisão ou local no projeto onde variações são introduzidas. As duas caixas conectadas abaixo do ponto de variação são as Variantes (Variante 1 e Variante 2), que representam diferentes implementações ou alternativas para o mesmo ponto de variação.

Figura 3 - Diagrama de variabilidade de uma LPS



Fonte: ANJOS (2011).

2.1.4.5 Modelagem de features

Para BOSCH (2000), o conceito de feature foi inicialmente introduzido pelo método *Feature Oriented Domain Analysis* (FODA). Uma feature representa uma característica do sistema que é perceptível pelo usuário final. A comunalidade e a variabilidade entre produtos de uma linha podem ser descritas por meio de features. Elas são definidas como uma unidade lógica de comportamento especificada por um conjunto de requisitos funcionais e de qualidade.

POHL, BOCKLE e LINDEN (2005), defendem que o conceito de feature é empregado para diferenciar os diversos produtos em uma LPS, identificando as funcionalidades que são comuns e as que variam entre os produtos. Além disso, uma feature pode se referir a requisitos não funcionais. Os artefatos de uma LPS também

podem ser denominados como features.

Ao organizarmos as features de um sistema em um gráfico, criamos um modelo de features. Esse modelo tem como objetivo fornecer uma visão geral em alto nível das principais características que são compartilhadas ou que variam em uma linha de produtos.

2.1.4.6 Notação F.O.D.A.

O método FODA, criado no *Software Engineering Institute*, introduziu uma representação gráfica para o modelo de feature. Sua abordagem detalhada da Análise de Domínio o tornou amplamente utilizado nos anos 1990. Além disso, FODA impulsionou a análise orientada a modelos, ao utilizar visões complementares para fornecer uma compreensão mais completa do domínio (KANG et. al,1990).

O modelo de features, organiza as características de um domínio em uma hierarquia, mostrando as relações estruturais entre elas. Essa técnica facilita a reutilização de artefatos de *software*, destacando as features comuns e variáveis, além de suas dependências. O modelo inclui um diagrama com informações adicionais, como descrições, pontos variáveis, prioridades e regras de dependência (SILVA, SILVEIRA NETO e GARCIA, 2011).

No contexto das LPS, um *feature model* representa a estrutura da linha de produtos. As features podem ser classificadas nos seguintes tipos:

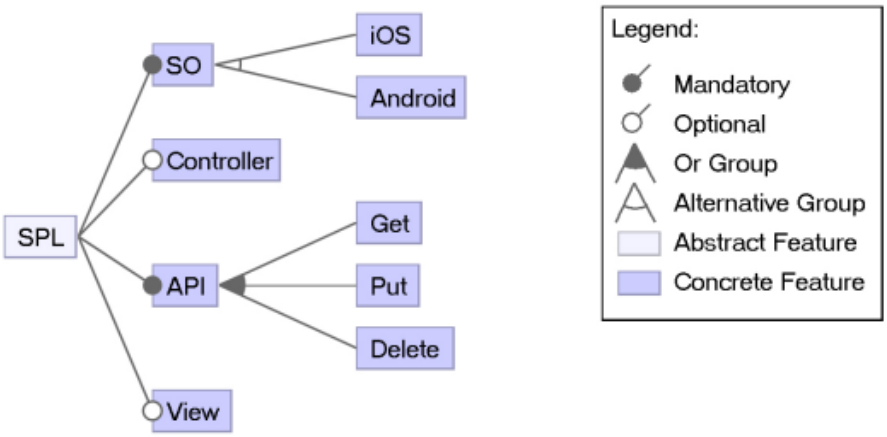
- **Obrigatórias**: devem estar presentes em todas as instâncias do produto.
- **Opcionais**: podem ou não estar presentes, dependendo do produto;
- **Alternativas**: formada por um conjunto de features em que se deve escolher uma ou mais. É importante especificar se apenas uma opção pode ser selecionada ou se várias são permitidas. Nesse contexto, distingue-se entre alternativas do tipo OR, que permitem selecionar múltiplas features, e XOR, que indica que apenas uma pode ser escolhida, devido à exclusão mútua;

Uma feature obrigatória é ilustrada por uma linha conectada a um círculo preto preenchido. Já a feature opcional é indicada por uma linha ligada a um círculo vazio. As features alternativas são representadas por linhas interligadas por um arco. Se o arco for vazio, apenas uma alternativa deve ser escolhida (XOR); se o arco for preenchido, é possível selecionar mais de uma alternativa (OR) (DEURSEN & KLINT, 2001).

MENDONÇA JÚNIOR (2022), em seu trabalho, apresentam uma possível

representação do modelo de *features* feita por BARROS (2022), conforme é possível ver abaixo na Figura 4:

Figura 4 – Exemplo de modelo de features



Fonte: BARROS (2022).

No projeto da figura acima, é possível perceber os diferentes tipos de features e os relacionamentos que compõem um modelo de diagrama de features. Destaca-se uma funcionalidade abstrata denominada SPL, além das features concretas: SO, iOS, Android, Controller, API, Get, Put, Delete e View. Entre essas, SO e API são features obrigatórias, enquanto Controller e View são opcionais. Há um relacionamento XOR entre as features iOS e Android, no qual apenas uma delas deve ser escolhida. Também existe um relacionamento OR, onde ao menos uma das features Get, Put ou Delete deve estar presente.

2.2 Microsserviços

2.2.1 Arquitetura monolítica

Segundo BECKER (2019), para compreender a arquitetura de microserviços, é essencial primeiro entender o conceito de arquitetura monolítica. A característica principal de um sistema baseado na arquitetura monolítica é que ela consiste em uma única aplicação que desempenha todas as funções do sistema, possuindo uma interface gráfica, uma API e um banco de dados único, sem depender de outras aplicações. Além disso, essa arquitetura não é projetada com modularidade externa, logo, não serve como base para outras aplicações. Quanto à modularidade, ela pode ser implementada internamente, onde os módulos são vinculados dentro da mesma

aplicação e, ao final, é necessário compilar todos os módulos para gerar uma aplicação única.

Para CHIARADIA, MACEDO e DUTRA (2018), em um sistema monolítico, conforme novas funcionalidades são incorporadas, o tamanho do código cresce, e chega um ponto em que a manutenção se torna onerosa. Isso ocorre, porque princípios da Engenharia de *Software*, como coesão e desacoplamento, não foram seguidos adequadamente durante o desenvolvimento, resultando em uma aplicação de difícil manutenção.

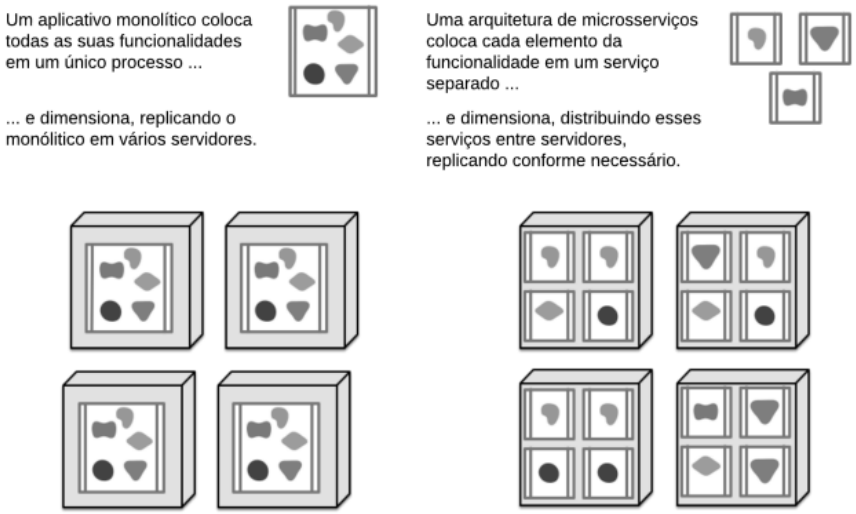
2.2.2 Arquitetura de microsserviços

LEWIS e FOWLER (2014) definem a arquitetura de microserviços como uma abordagem de desenvolvimento em que uma aplicação é constituída por um conjunto de pequenos serviços, cada um rodando em seu próprio processo ou servidor dedicado. Esses serviços são independentes, escaláveis e projetados de acordo com as regras de negócios. A comunicação entre eles ocorre por mecanismos leves como barramento de mensagens do RabbitMQ ou por APIs e o protocolo HTTP.

O termo *Microservice Architecture* tem sido utilizado recentemente para descrever esse modelo, onde aplicações são concebidas como conjuntos de serviços que podem ser implementados de forma autônoma. NEWMAN (2015) complementa, explicando que microsserviços são serviços pequenos e autônomos que operam de forma colaborativa, sendo cada um uma entidade separada, o qual respeita o princípio da responsabilidade única, capaz de ser implementada isoladamente em uma plataforma como serviço (PaaS).

Abaixo (Figura 5), é possível observar como funciona as arquiteturas de *software* baseadas em monolíticas e com microsserviços (LEWIS e FOWLER, 2014):

Figura 5 - Diferença entre as arquiteturas de *software* baseadas em uma aplicação monolítica e com microsserviços



Fonte: LEWIS e FOWLER (2014, adaptado).

No lado esquerdo da imagem, temos a aplicação desenvolvida com arquitetura monolítica. Nessa arquitetura, todas as funcionalidades (representadas pelas diferentes formas geométricas) estão contidas em um único bloco, ou seja, todas as partes do sistema estão interligadas em um único processo. Para aumentar a capacidade do sistema (diminuir a carga), o monolito é replicado e distribuído em vários servidores, mas ainda assim, cada instância replicada contém todas as funcionalidades completas. Já no lado direito, temos a arquitetura baseada em microserviços, onde cada funcionalidade é separada em serviços individuais, representados pelas diferentes formas geométricas que estão fora do bloco maior. Em vez de replicar todo o sistema como um monolito, os serviços são distribuídos entre diferentes servidores e replicados conforme necessário. Isso permite uma maior flexibilidade, escalando apenas os serviços que exigem mais recursos.

A autonomia e o isolamento entre os serviços possibilitam que eles sejam desenvolvidos com diferentes linguagens de programação e tecnologias. Isso permite que o processo de implantação de cada serviço ocorra de forma completamente independente, o que possibilita que as aplicações sejam instaladas em máquinas distintas. (MALIPENSE, 2018).

2.2.3 Vantagens e desvantagens dos microsserviços

As vantagens do uso de microsserviços, conforme NEWMAN (2015) são:

- **Uso de diversas tecnologias**: a flexibilidade de utilizar uma ampla gama de tecnologias diferentes. Isso possibilita aos desenvolvedores a escolha das melhores ferramentas para alcançar o desempenho necessário ou solucionar problemas específicos do projeto;
- **Responsabilidade única e tolerância a falhas**: cada microsserviço é responsável por uma função específica dentro de um escopo restrito. Se ocorrer uma falha, apenas uma parte da aplicação é impactada, sem comprometer o sistema como um todo, aumentando a tolerância a falhas. Se o endpoint de cadastro de produto ficar indisponível, as demais funções e endpoints do sistema continuarão funcionando normalmente;
- **Escalabilidade**: característica de aplicações em nuvem, onde facilita a adaptação de componentes individuais ou do sistema inteiro conforme a demanda. Como os microsserviços são independentes, os recursos podem ser ajustados para atender melhor às suas necessidades operacionais.

Embora a arquitetura de microsserviços ofereça diversas vantagens, ela não é indicada para todos os tipos de projeto. Sua adoção deve ser cuidadosamente avaliada, pois pode introduzir complexidades desnecessárias em sistemas menores ou menos complexos, onde uma arquitetura monolítica mais simples pode ser mais eficiente e fácil de manter. (DOMINGOS e FARINA, 2020).

MOREIRA & BEDER (2016) e FAMILIAR (2015) destacam algumas desvantagens no uso de microsserviços:

- **Complexidade no desenvolvimento**: a criação de microsserviços exige um esforço adicional, pois o sistema é fragmentado em várias partes menores, o que pode complicar o controle e a coordenação entre os serviços.
- **Necessidade de lidar com chamadas remotas e o gerenciamento de múltiplos bancos de dados e transações**: como cada microsserviço possui seu próprio banco de dados e realiza comunicações por meio de redes, isso aumenta a complexidade no gerenciamento de dados e na manutenção de transações distribuídas.
- **Gestão e suporte operacional dos microsserviços**: É fundamental contar com uma equipe experiente em operações, já que será necessário coordenar e gerenciar vários serviços que se intercomunicam de diferentes maneiras.

2.3 Sistemas de gerenciamento de refeições

Segundo SIQUEIRA (2020), no Brasil, há 112 universidades públicas em funcionamento, sendo 68 federais e 44 estaduais, muitas das quais contam com Restaurantes Universitários (RU’s) que variam em preços e estrutura. Entretanto, apenas algumas disponibilizam aplicativos móveis voltados para a comunidade acadêmica. Em sua pesquisa, a autora identificou 21 aplicativos relacionados a esses restaurantes, desenvolvidos por alunos ou ex-alunos de instituições como UFT, UFRJ, USP, UFPA, UFC, UnB, entre outras. Como a responsabilidade pela manutenção fica a cargo dos próprios criadores, muitos desses aplicativos acabam desatualizados e deixam de ser utilizados ao longo do tempo.

Existem também algumas universidades que desenvolvem e mantêm seus próprios aplicativos. Entre os exemplos estão o “Cardápio” da USP, o “UFMA Mobile” da UFMA, o “RU UFRN” da UFRN e o “UFMS Digital” da UFMS, "eUFS - RESUM" da UFS. Como são administrados pelas próprias instituições, esses aplicativos funcionam de forma adequada e passam por atualizações frequentes (SIQUEIRA, 2020).

Algumas universidades e institutos federais utilizam sistema de controle próprios no contexto de RU's como é o exemplo da UFSM. Já outras utilizam o SIGAA para atendimento da demanda, como é o exemplo do IFAL e UFC.

Diversas universidades, seja por meio de aplicativos móveis ou aplicações web, têm desenvolvido ferramentas digitais para o controle, agendamento e consulta de cardápios de refeições nos Restaurantes Universitários (RU). Essas soluções são fundamentais para melhorar o atendimento, garantindo uma experiência mais organizada e ágil para os estudantes.

A digitalização desses serviços permite, por exemplo, o controle de demanda, o que ajuda a evitar filas longas e desorganizadas, além de minimizar o desperdício de alimentos ao ajustar a quantidade de refeições produzidas de acordo com a reserva prévia dos usuários. Além disso, essas ferramentas facilitam o acesso à informação sobre o cardápio diário, possibilitando aos estudantes a escolha consciente de suas refeições, considerando restrições alimentares ou preferências pessoais. O uso dessas tecnologias pelas universidades permite atualizações frequentes e a manutenção eficiente do sistema, garantindo que os estudantes sempre tenham acesso a informações precisas e atualizadas.

3 TRABALHOS RELACIONADOS

SIQUEIRA (2020) desenvolveu o RUnB, um aplicativo móvel para facilitar o uso do restaurante universitário da Universidade de Brasília, buscando reduzir filas e melhorar a experiência dos estudantes. Foi desenvolvido na linguagem java com arquitetura monolítica.

BRITO (2020) desenvolveu um aplicativo para apoiar o fornecimento de refeições no IFBA, com funcionalidades como cadastro de cardápios e relatórios anuais. A solução evita deslocamentos desnecessários dos alunos e custos com carteirinhas, além de permitir ao refeitório visualizar, em tempo real, a demanda de refeições, otimizando recursos. O trabalho foi desenvolvido com arquitetura monolítica.

MURTA (2022) desenvolveu um sistema voltado para a confirmação do acesso dos usuários ao restaurante universitário, com o objetivo de diminuir o desperdício de alimentos nesses locais, por meio da coleta de dados mais precisos sobre o número de pessoas que comparecem às refeições. Foi desenvolvido baseado em uma arquitetura monolítica.

CARDOSO (2021) propoe um sistema web voltado para facilitar e automatizar aspectos do sistema atual, visando aprimorar a gestão e o controle do restaurante universitário. O sistema Web RU foi implementado utilizando a linguagem de programação java juntamente com o ecossistema *spring*. Utilizou-se da arquitetura monolítica.

Na literatura, existem outros trabalhos que exploram sistemas de refeições para universidades. Contudo, os autores apresentados já contemplam as principais funcionalidades necessárias, e, para evitar redundâncias, foram destacados apenas os mais relevantes. Um ponto em comum entre esses trabalhos é que nenhum deles foi desenvolvido com uma arquitetura baseada em microsserviços ou derivado de uma LPS. Na Tabela 1, será apresentado um comparativo de funcionalidades e implementações de cada autor:

Tabela 1 - Comparativo de funcionalidade e implementações entre os trabalhos apresentados

Autor	SIQUEIRA (2020)	BRITO (2020)	CARDOSO (2021)	MURTA (2022)
LPS				
Microsserviços				
Agendamentos	X	X		X
Avaliação	X		X	
Cardápio	X	X		
Refeições	X			
Suporte a Pagamentos	X			
Métricas ou relatórios		X	X	
Carteira digital	X			

Fonte: autor.

4 LINHA DE PRODUTO DE *SOFTWARE* PROPOSTA

Neste capítulo, serão apresentados os principais aspectos da Linha de Produto de *Software* Proposta, detalhando suas características, arquitetura e processo de criação de novos produtos.

A seção 4.1 apresenta a modelagem de domínio. A Seção 4.2 explora a variabilidade da LPS proposta, enfatizando como a flexibilidade e as opções configuráveis contribuem para a personalização dos produtos gerados. A Seção 4.3 apresenta o Diagrama de Features, que ilustra visualmente as funcionalidades e variações disponíveis na linha de produto. Na Seção 4.4, é discutida a Arquitetura da LPS utilizando microsserviços, destacando os benefícios de uma arquitetura distribuída, como escalabilidade e manutenibilidade. Por fim, a Seção 4.5 detalha o processo de criação de um produto (aplicação) a partir da LPS proposta, descrevendo os passos necessários para criação de uma nova aplicação.

4.1 Modelagem de domínio

Para a construção da engenharia de domínio, ARAÚJO et al. (2009) destacam que esse método pode ser composto por: criação de um modelo conceitual do domínio, desenvolvimento de um diagrama de features do domínio e construção de um diagrama de casos de uso.

4.1.1 Conceitos-chave do domínio

Para se obter o domínio a ser trabalhado foi utilizado como referências conceitos e termos de domínio dos trabalhos relacionados. Abaixo serão definidos alguns termos centrais de domínio:

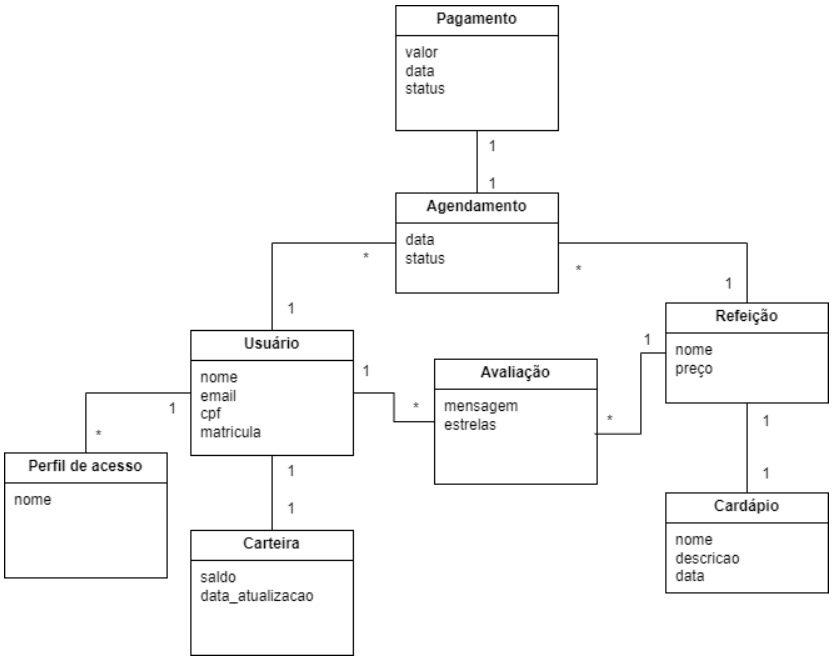
- Refeição: uma unidade de serviço alimentar correspondente a um dos períodos do dia, como café da manhã, almoço ou jantar (SIQUEIRA, 2020).
- Cardápio: conjunto de opções alimentares disponíveis em uma refeição, cuja composição pode variar de acordo com a categoria específica da refeição (SIQUEIRA, 2020).
- Avaliação: processo no qual os alunos fornecem feedback aos administradores do RU sobre a qualidade e experiência do serviço oferecido, contribuindo para melhorias no atendimento (CARDOSO, 2021).
- Carteira digital (saldo e extrato): processo que permite o usuário visualizar suas movimentações financeiras como saldo atual da carteira digital e seu histórico de

transações (SIQUEIRA, 2020).

4.1.2 Modelo conceitual de domínio

MOON et al. (2005) definem que a metodologia empregada na engenharia de domínio inicia com a criação de um modelo conceitual que representa o domínio compartilhado pelas aplicações. Para que o engenheiro de requisitos consiga identificar e detalhar os requisitos do domínio, é essencial estabelecer definições para os conceitos fundamentais e os termos característicos do domínio. Será apresentado abaixo, na Figura 6, o modelo conceitual do domínio para um sistema de agendamento de refeições no qual foi elaborado com base na análise das aplicações desenvolvidas dos trabalhos relacionados:

Figura 6 - Modelo conceitual do domínio de sistema de agendamento de refeição



Fonte: Autor.

4.1.3 Identificação dos requisitos do domínio

Para a construção dos requisitos de domínio para a LPS, é necessário identificar requisitos que serão comuns para toda instância gerada a partir da LPS e aquelas que serão opcionais.

Conforme POHL et al. (2005), as matrizes que relacionam aplicações e requisitos servem como ferramentas para mapear quais requisitos de alto nível estão associados a diferentes aplicações dentro de uma LPS. Essas matrizes permitem visualizar de forma clara as similaridades, identificando requisitos comuns entre as aplicações, e as variabilidades, evidenciando os requisitos específicos de cada uma.

Com base na abordagem proposta por MOON et al. (2005), na análise de similaridade e variabilidade (s/v), é possível calcular, para cada requisito identificado, uma taxa de similaridade (taxa) que indica o nível de uso do requisito no domínio. Essa taxa é determinada pela proporção entre o número de sistemas que incluem o requisito e o total de sistemas analisados. Quando a taxa é elevada, o requisito é classificado como comum (C); caso contrário, é considerado opcional (O). De maneira similar ao trabalho de MOON et al. (2005), foi estabelecido o valor de 0,5 para distinguir entre objetivos comuns e opcionais.

Baseado nas aplicações desenvolvidas pelos autores: CARDOSO (2021), SIQUEIRA (2020), BRITO (2020) e MURTA (2022), pode-se perceber funcionalidades comuns que todos os sistemas no domínio de refeições possuem.

Os diagramas de variabilidade (ver Seção 4.2) e de features (ver Seção 4.3) foram elaborados com base nas informações apresentadas na Tabela 2.

Tabela 2 – Matriz aplicações x requisitos

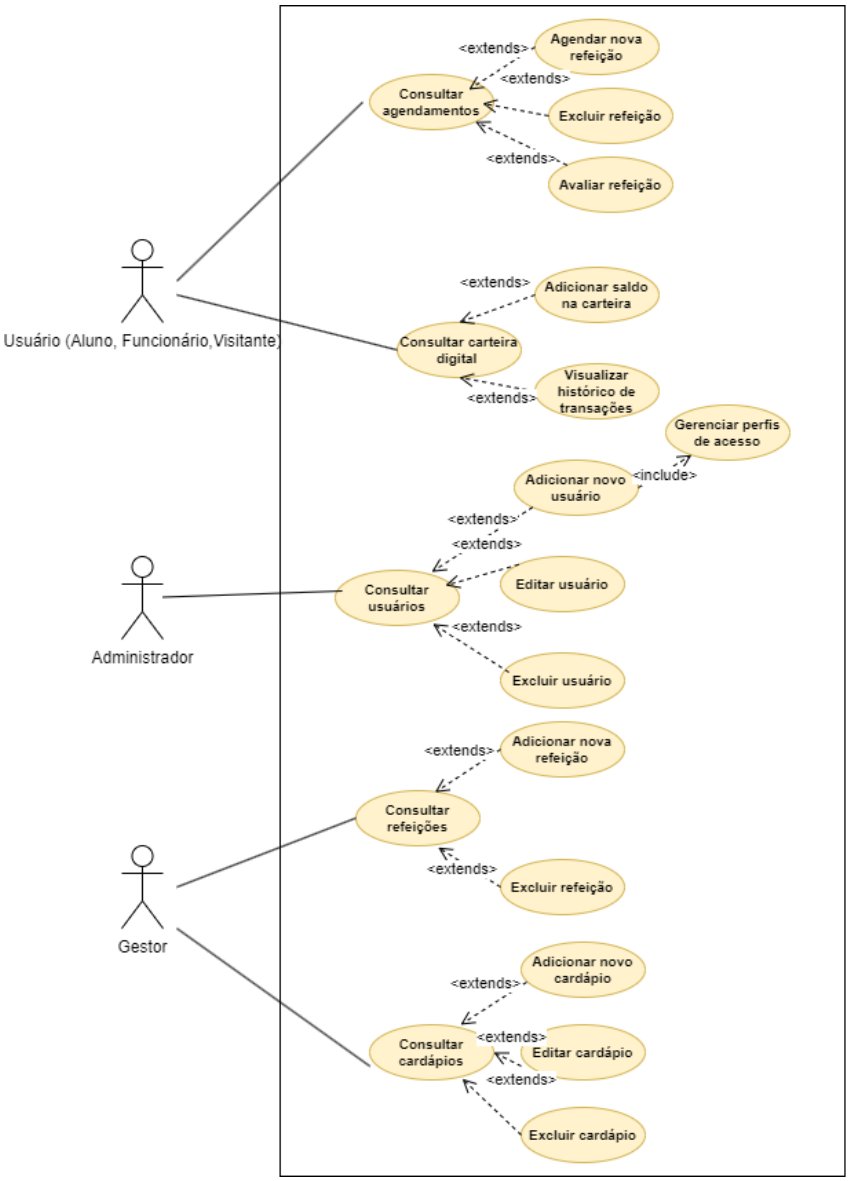
Objetivos/sub-objetivos		s/v	taxa	SIQUEIRA (2020)	BRITO (2020)	CARDOSO (2021)	MURTA (2022)
1.	**Agendar refeição**	**C**	**0,75**	**1**	**1**	**0**	**1**
1.1	**Selecionar o tipo de refeição**	**C**	**0,5**	**1**	**1**	**0**	**0**
2	**Avaliar refeição**	**C**	**0,5**	**1**	**0**	**1**	**0**
3	Pagar refeição	O	0,25	1	0	0	0
4	**Consultar cardápio**	**C**	**0,5**	**1**	**1**	**0**	**0**
5	**Gerenciar perfis de usuário**	**C**	**0,75**	**0**	**1**	**1**	**1**
6	**Gerenciar Usuário**	**C**	**1**	**1**	**1**	**1**	**1**
7	Consultar Métricas	O	0	0	0	0	0

Fonte: autor.

4.1.4 Diagramas UML

Na construção da modelagem do sistema foi utilizado diagramas UML. Abaixo, serão ilustrados os diagramas elaborados. O Diagrama de Casos de Uso (ver Figura 7), conforme definido por Jacobson (1999), visa ilustrar as funcionalidades de um sistema e a relação dessas funcionalidades com os usuários que as utilizam. Esse tipo de diagrama é composto por elementos como Cenário, Ator, Caso de Uso e Comunicação, que conecta o ator ao caso de uso.

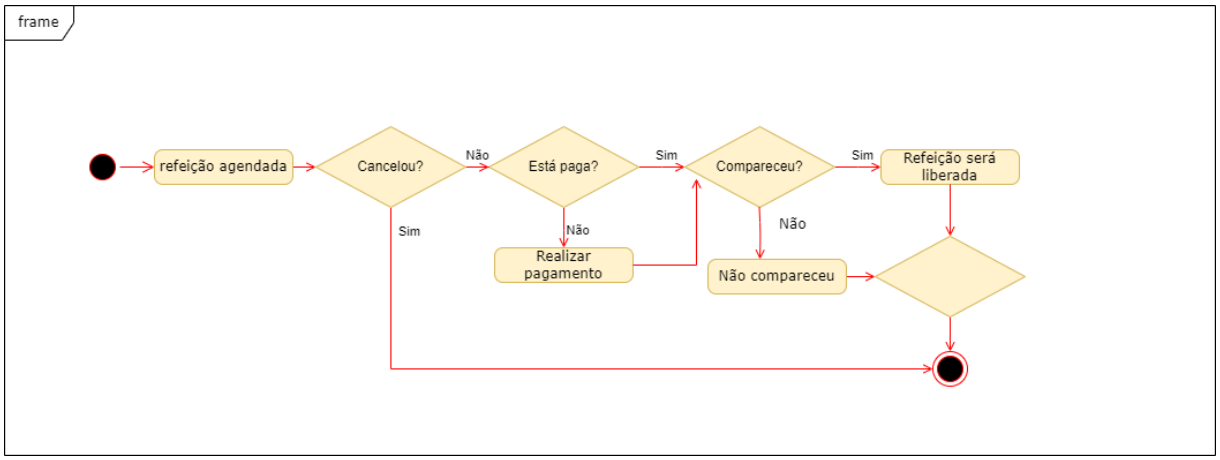
Figura 7 - Diagrama de casos de uso



Fonte: Autor.

Também foi desenvolvido o diagrama de estados (Figura 8), sendo uma ferramenta utilizada para representar comportamentos discretos de sistemas que mudam de estado ao longo do tempo. De acordo com GUEDES (2007), esse diagrama ilustra situações específicas e transições entre diferentes objetos ou elementos, destacando as condições necessárias ou impeditivas para a transição de um estado a outro, até se alcançar o estado final.

Figura 8 - Diagrama de estados (agendamento até utilização da refeição)



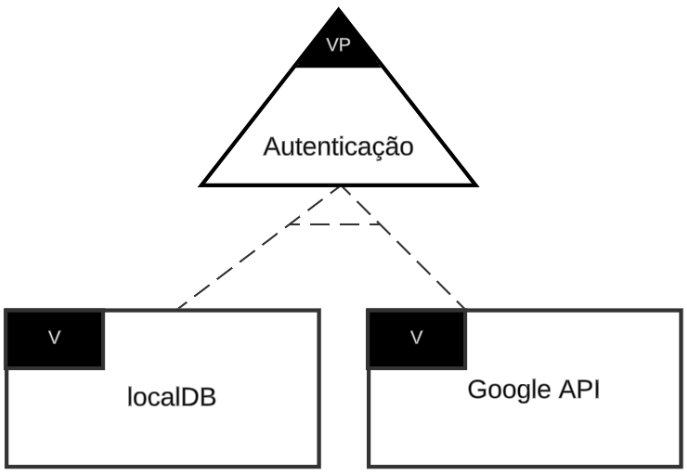
Fonte: autor.

4.2 A variabilidade na linha de produto de *software* proposta

O diagrama de variabilidade da linha de produto modelada será apresentado para cada ponto de variação (VP).

- **Ponto de Variação** - Autenticação (ver Figura 9)
- **Variantes**
  - Google API – Opcional
  - Banco de Dados Local – Opcional
- **Classificação da variabilidade**: Escolha Múltipla (OR).

Figura 9 - Diagrama de variabilidade de autenticação



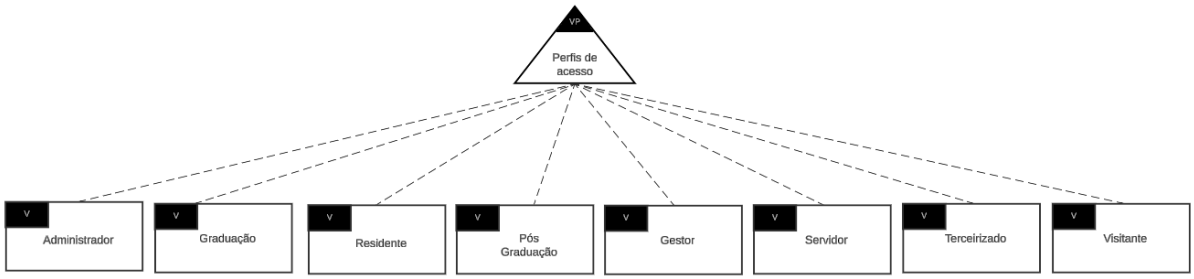
Fonte: Autor.

- **Ponto de Variação** - Perfis de acesso (ver Figura 10)
- **Variantes**

- Administrador – Obrigatório
- Graduação – Opcional
- Residente – Opcional
- Pós-graduação – Opcional
- Gestor – Opcional
- Terceirizado – Opcional
- Servidor – Opcional
- Visitantes – Opcional

- **Classificação da variabilidade**: Escolha Múltipla (OR).

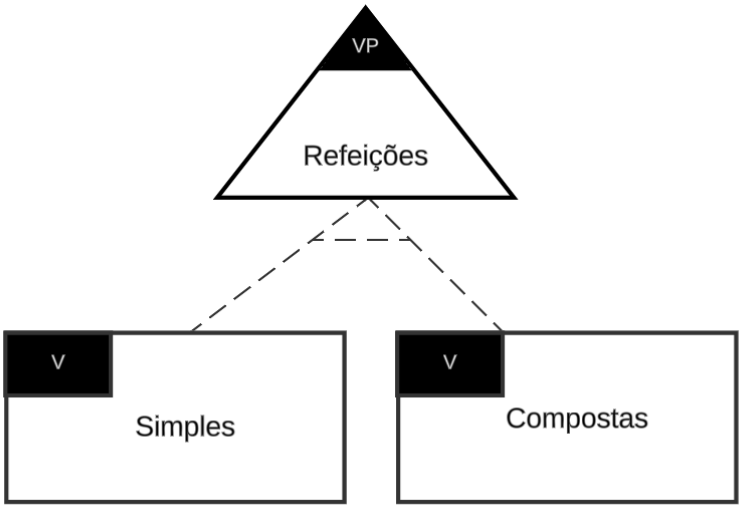
Figura 10 - Diagrama de variabilidade de perfis de acesso



Fonte: Autor.

- **Ponto de Variação** - Refeições (ver Figura 11)
- **Variantes**
  - Simples – Opcional
  - Composta – Opcional
- **Classificação da variabilidade**: Escolha Múltipla (OR).

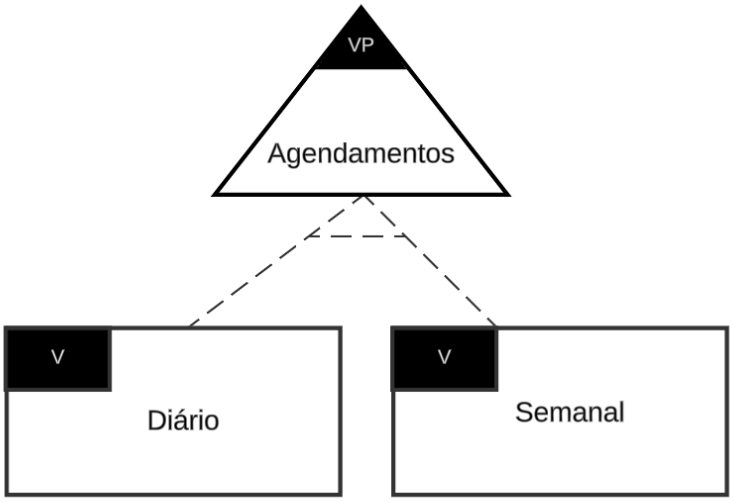
Figura 11 - Diagrama de variabilidade de refeições



Fonte: Autor

- **Ponto de Variação** - Agendamentos (ver Figura 12)
- **Variantes**
  - Diário – Opcional
  - Semanal – Opcional
- **Classificação da variabilidade**: Escolha Múltipla (OR).

Figura 12 - Diagrama de variabilidade de agendamentos

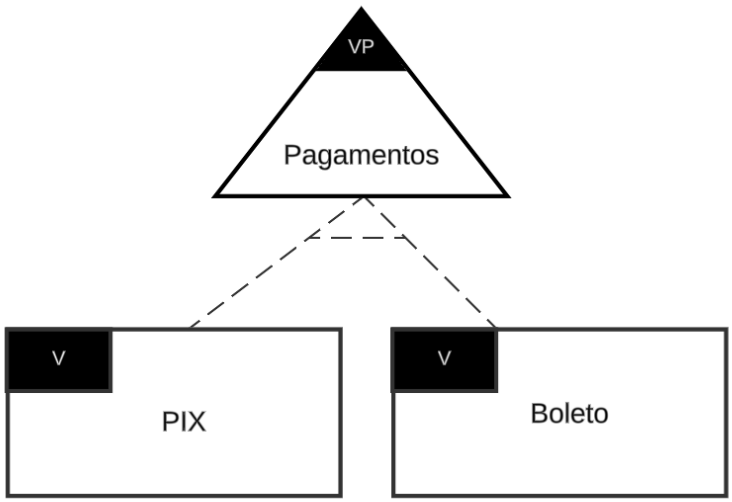


Fonte: Autor

- **Ponto de Variação** - Pagamentos (ver Figura 13)
- **Variantes**
  - PIX– Opcional

  - Boleto – Opcional
- **Classificação da variabilidade**: Escolha Múltipla (OR).

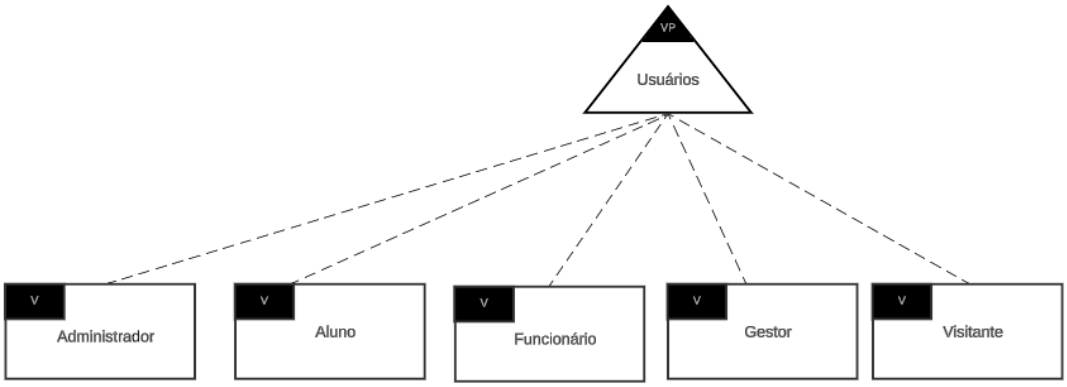
Figura 13 - Diagrama de variabilidade de pagamentos



Fonte: Autor

- **Ponto de Variação** - Usuários (ver Figura 14)
- **Variantes**
  - Administrador – Obrigatório
  - Gestor – Opcional
  - Funcionário – Opcional
  - Aluno – Opcional
  - Visitante – Opcional
- **Classificação da variabilidade**: Escolha Múltipla (OR)

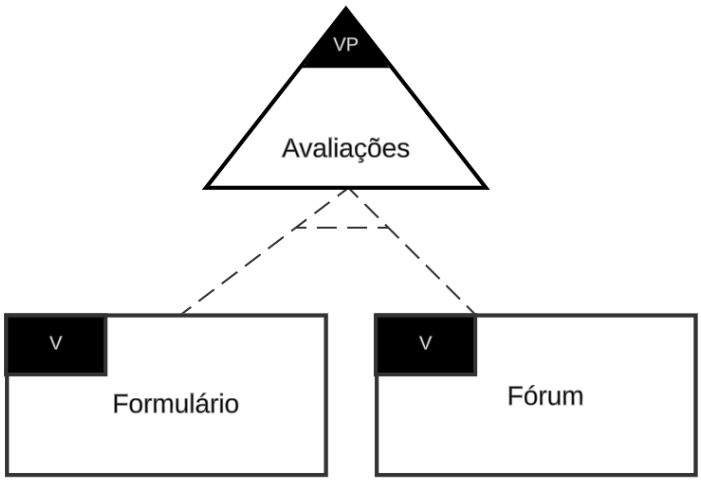
Figura 14 - Diagrama de variabilidade de usuários



Fonte: Autor

- **Ponto de Variação** - Avaliações (ver Figura 15)
- **Variantes**
  - Formulário – Alternativa (escolha única)
  - Fórum – Alternativa (escolha única)
- **Classificação da variabilidade**: Alternativa (XOR)

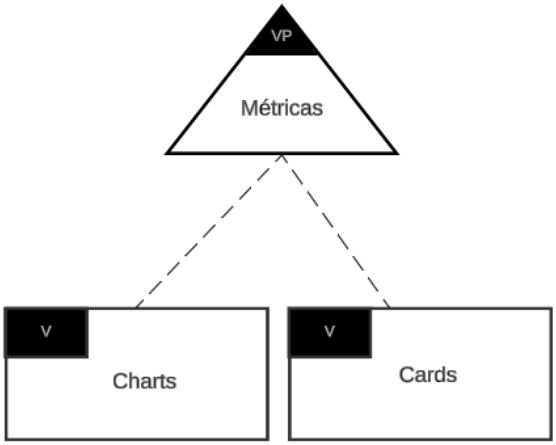
Figura 15 - Diagrama de variabilidade de avaliações



Fonte: Autor

- **Ponto de Variação** - Métricas (ver Figura 16)
- **Variantes**
  - *Cards*– Opcional
  - *Charts*– Opcional
- **Classificação da variabilidade**: Escolha Múltipla (OR)

Figura 16 - Diagrama de variabilidade de métricas

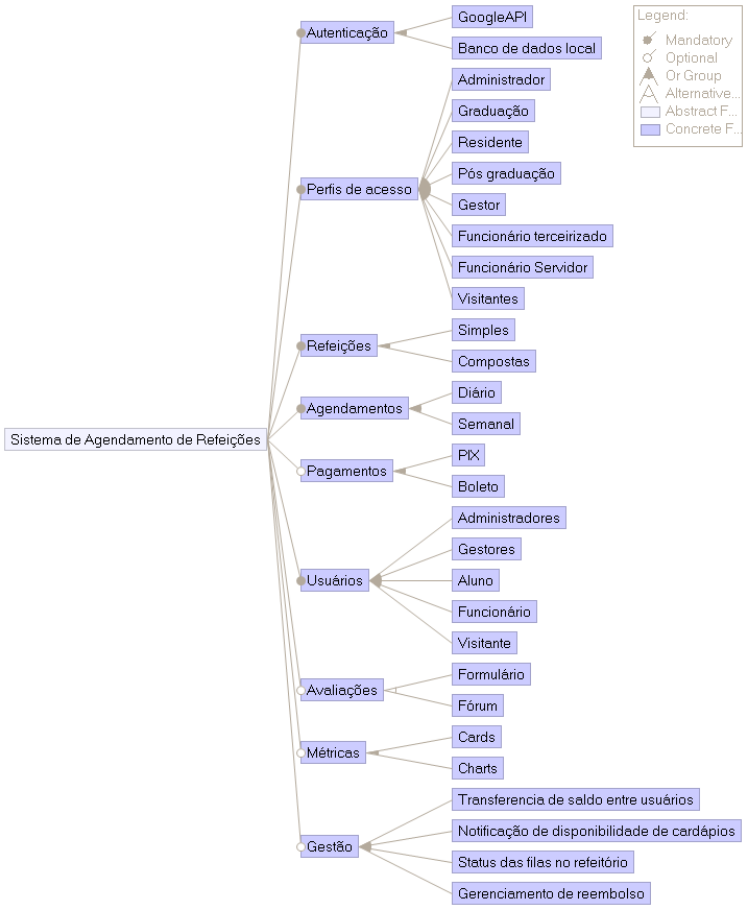


Fonte: Autor

4.3 Diagrama de features

O diagrama de features da linha de produto modelada é mostrado na Figura 17:

Figura 17 - Diagrama de features da LPS



Fonte: Autoria própria.

Detalhamento das features:

Autenticação

- **Obrigatório**
- O sistema precisa de autenticação para acesso. Pode ser feito via GoogleAPI ou banco de dados local.

Perfis de Acesso

- **Obrigatório**
- O sistema define vários perfis de usuários com diferentes níveis de acesso, incluindo:
    - Administrador;
    - Graduação;
    - Residente;
    - Pós-graduação;
    - Gestor;
    - Funcionário terceirizado;
    - Funcionário Servidor;
    - Visitantes.

Refeições

- **Obrigatório**
- O sistema permite a seleção de tipos de refeições:
    - Simples;
    - Compostas.

Agendamentos

- **Obrigatório**
- O agendamento das refeições pode ser:
    - Diário;
    - Semanal.

Pagamentos

- **Opcional**
- O sistema oferece métodos de pagamento:
    - PIX;
    - Boleto.

Usuários

- **Obrigatório**
- Os usuários são classificados em:
    - Administradores;
    - Gestores;
    - Aluno;
    - Funcionário;
    - Visitante.

Avaliações

- **Alternativa**
- O sistema oferece avaliações em dois formatos:
    - Formulário;
    - Fórum.

Métricas

- **Opcional**
- O sistema oferece métricas em dois formatos:
    - Cartões (*cards*);
    - Gráficos em barra (*charts*).

As features de Gestão (transferência de saldo entre usuário, notificação de disponibilidade de cardápios, transferência de saldo entre usuários e gerenciamento de reembolso) não foram realizadas devido ao curto espaço de tempo.

4.4 Arquitetura da linha de produto proposta utilizando microsserviços

Para criação da LPS, foram usadas diversas tecnologias. Nessa seção serão brevemente descritas as tecnologias usadas para sua construção e apresentada sua arquitetura.

4.4.1 Tecnologias utilizadas no frontend

O *ReactJS* [1] é uma biblioteca *JavaScript* desenvolvida pelo Facebook, amplamente utilizada para construir interfaces de usuário (UI) interativas e dinâmicas. Ele se destaca por sua abordagem baseada em componentes, o que facilita a criação e a manutenção de aplicações escaláveis e modulares. O principal objetivo do *React* é otimizar o desenvolvimento de interfaces rápidas e responsivas, permitindo que o desenvolvedor trabalhe de forma declarativa na criação de componentes reutilizáveis que se atualizam de maneira eficiente quando os dados são alterados.

Além disso, o *React* é uma escolha popular para o desenvolvimento de Aplicações de Página Única, ou do ingles, *Single Page Application* (SPA), onde o conteúdo é carregado dinamicamente sem a necessidade de recarregar a página inteira. Isso resulta em uma experiência de usuário mais fluida e rápida, pois as interações são realizadas em tempo real, proporcionando uma navegação mais similar à de aplicativos nativos.

No projeto, foi utilizada a versão 18.2.0 do ReactJS, que oferece suporte a várias melhorias de performance e novos recursos, como o modo de concorrência, que otimiza o gerenciamento de múltiplas atualizações simultâneas.

4.4.2 Tecnologias utilizadas no backend

Para o desenvolvimento do backend, foi utilizado o Node.js[2] e o NestJS[3]. O Node.js (CANTELON et al., 2017) é um ambiente de execução JavaScript de código aberto e assíncrono, que permite a criação de aplicações de alta performance e escaláveis. Sua arquitetura baseada em eventos e o modelo de I/O não bloqueante o tornam ideal para construir aplicações que requerem um alto nível de concorrência, como APIs e microsserviços.

Em conjunto com o Node.js, utilizamos o NestJS, um framework progressivo para construir aplicações de servidor eficientes e escaláveis. A versão utilizada no

[1] https://react.dev/
[2] https://nodejs.org/pt
[3] https://nestjs.com/

projeto foi a 10.3.8. A linguagem utilizada para o desenvolvimento foi o *TypeScript* (TYPESCRIPT, 2024), que traz benefícios como tipagem estática e melhor suporte a ferramentas, aumentando a robustez e a manutenibilidade do código.

Uma característica fundamental do NestJS é sua compatibilidade com microsserviços, permitindo que cada microsserviço seja desenvolvido de forma independente e implementado em projetos separados. Cada um desses serviços pode se comunicar entre si e com a API Gateway, que atua como o ponto central de entrada para as requisições. Essa abordagem modularizada não apenas melhora a escalabilidade da aplicação, mas também facilita o gerenciamento de cada componente de forma isolada, promovendo a agilidade no desenvolvimento e na implementação de novas funcionalidades.

Assim, a combinação do Node.js com o NestJS não apenas suporta a construção de aplicações robustas e escaláveis, mas também proporciona uma comunicação eficiente entre os microsserviços, garantindo que a arquitetura da aplicação permaneça coesa e flexível.

4.4.3 Sistema de gerenciamento de banco de dados

O sistema de gerenciamento de banco de dados (SGBD) adotado neste projeto foi o MySQL[4], que utiliza a linguagem SQL para manipulação de dados. Reconhecido como o banco de dados de código aberto mais amplamente utilizado globalmente, o MySQL se destaca por sua robustez e eficiência. Segundo o ranking do DB-Engines (2024), o MySQL ocupa a segunda posição entre os bancos de dados mais populares, ficando apenas atrás do Oracle Database.

Essa popularidade se deve não apenas à sua flexibilidade e escalabilidade, mas também à extensa comunidade de desenvolvedores que contribuem continuamente para sua evolução, tornando-o uma escolha confiável para uma variedade de aplicações, desde pequenos projetos até sistemas de grande escala. Além disso, sua compatibilidade com diversas plataformas e linguagens de programação facilita a integração em diferentes ambientes de desenvolvimento.

4.4.4 IDE utilizada

A IDE utilizada foi o Microsoft Visual Studio Code (VS Code), versão 1.85.2. Escolhido para este projeto por ser um editor de código leve e altamente

[4] https://www.mysql.com/

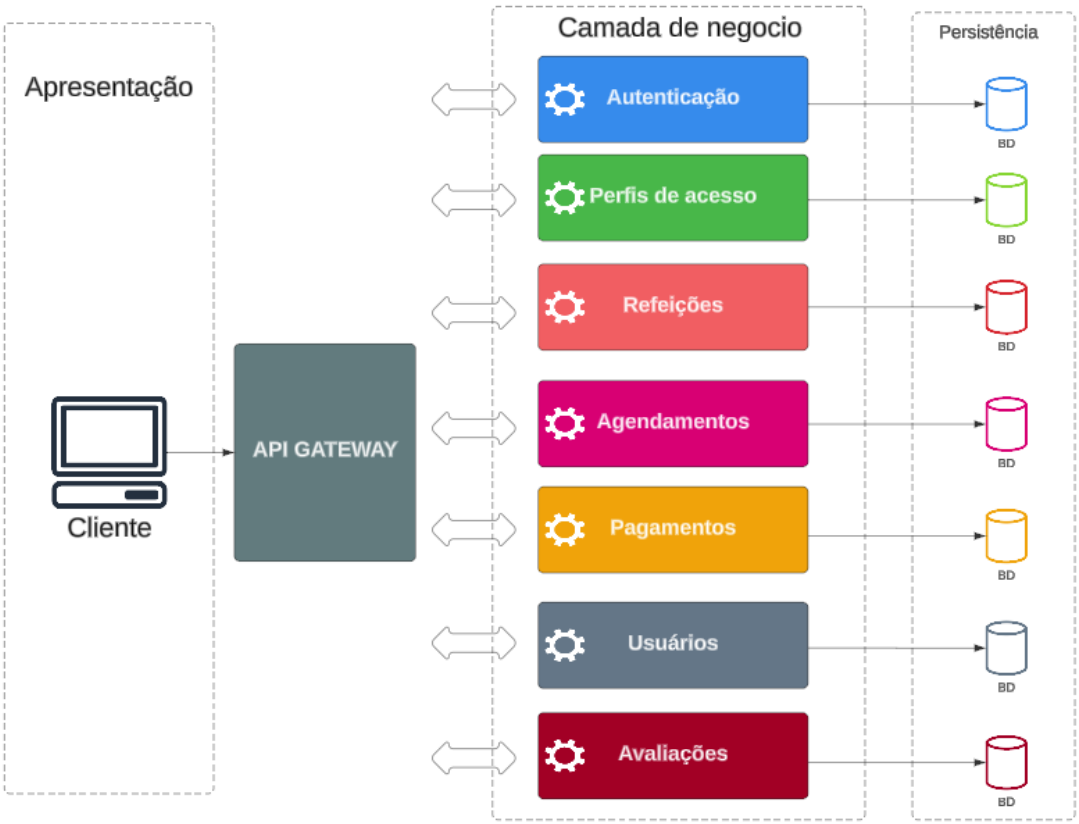
personalizável, o VS Code oferece suporte a plugins que ampliam suas funcionalidades, além de integrar-se facilmente com sistemas de controle de versão, como o Git, facilitando a gestão de projetos e a manutenibilidade.

4.4.5 Arquitetura da aplicação

A Figura 18, representa o diagrama da arquitetura em camadas baseada em microsserviços, onde a API *Gateway* atua como ponto de entrada central para todas as solicitações do cliente. Ela agrega e gerencia a comunicação entre os diferentes microsserviços, garantindo que cada requisição seja encaminhada ao serviço correto.

Os microsserviços são responsáveis por funcionalidades específicas, como autenticação, perfis de acesso, refeições, agendamentos, pagamentos, usuários e avaliações, cada um com seu próprio banco de dados, promovendo isolamento e independência. A comunicação entre a API *Gateway* e os microsserviços utiliza o padrão REST, simplificando a troca de informações por meio de APIs padronizadas e facilitando a escalabilidade, manutenção e evolução do sistema. A interação entre os serviços é feita por meio dos métodos HTTP tradicionais (*GET*, *PUT*, *POST* e *DELETE*), com a troca de dados em formatos multiplataforma, como JSON.

Figura 18 - Arquitetura de microsserviços da linha de produto para sistemas de refeições



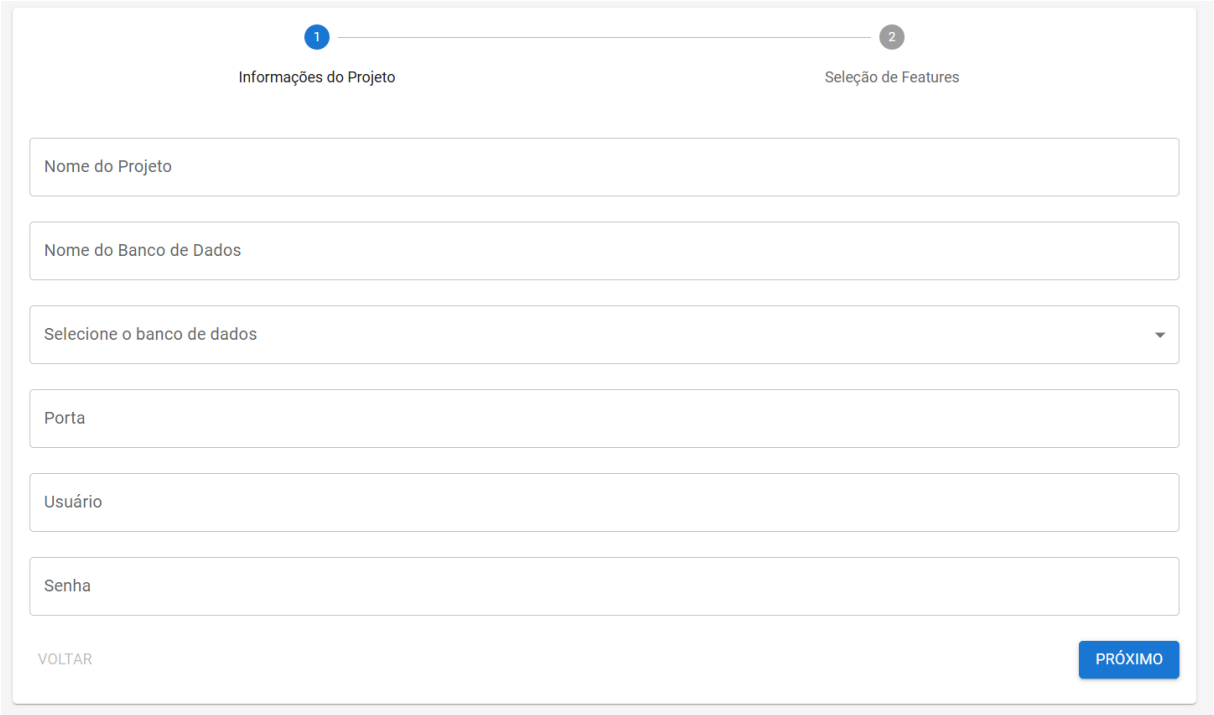
Fonte: Autor.

4.5 Criação de um produto (aplicação) a partir da linha de produto proposta

A nova aplicação de refeições proposta para o RU da UFAL foi realizada a partir do reuso da arquitetura baseada em microsserviços da LPS desenvolvida para sistemas de refeições. A criação de um produto (aplicação) a partir da LPS proposta é feita através da definição de suas configurações e features, utilizando o guia de instalação (*wizard guide*), que pode ser acessado pelo *endpoint* localhost:3000/wizard do próprio projeto.

Essa atividade está dividida em duas fases: (I) informações básicas do projeto; (II) suas features. Para ilustrar o guia de instalação, as Figuras 19 e 20 apresentam as telas onde ocorre a preparação do produto, além de detalharem suas especificidades. Nessa fase, são feitas as configurações dos produtos, que posteriormente serão derivados.

Figura 19 - Guia de instalação LPS: tela das informações básicas do projeto



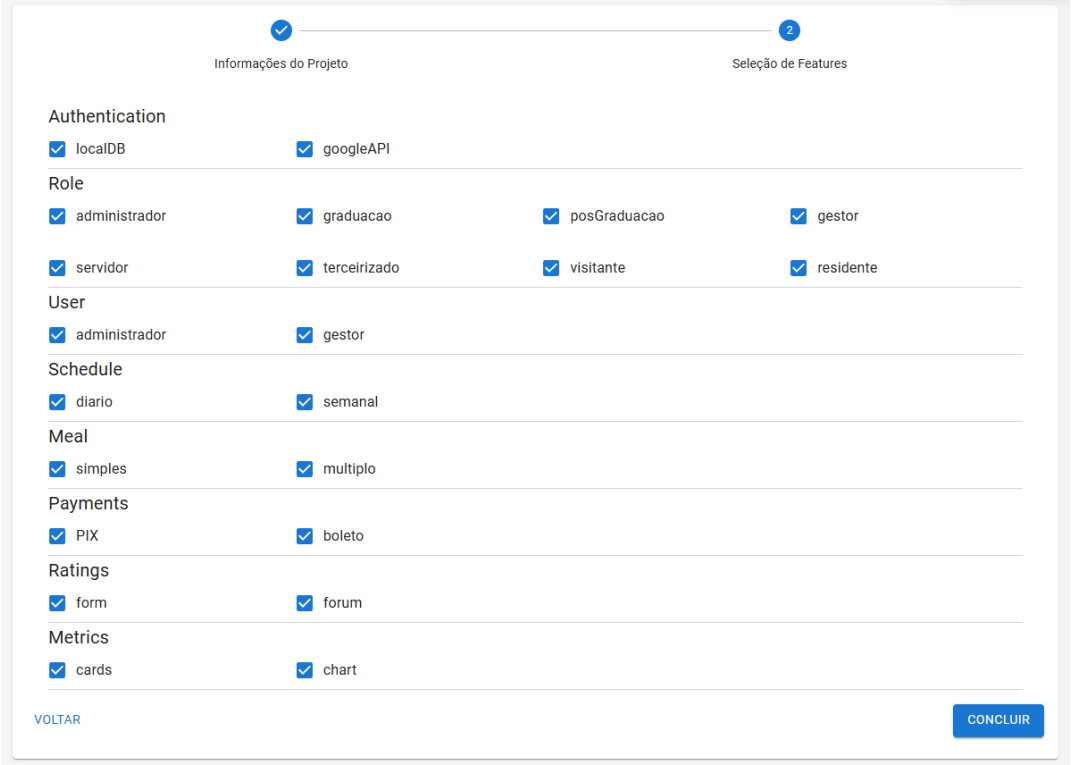
Fonte: autor.

A Figura 18 ilustra a primeira etapa do assistente de criação de projetos, que é dedicada à inserção de informações básicas do projeto. Nesta fase, será solicitado o preenchimento dos seguintes campos:

1. **Nome do projeto**: Este campo é destinado ao nome que você deseja atribuir ao novo projeto. Ele serve como um identificador principal e é a primeira referência que você e outros usuários terão para reconhecer o projeto.
2. **Nome do banco de dados**: Aqui, você deve especificar o nome que será atribuído ao banco de dados associado ao novo projeto. Este nome é importante, pois será utilizado para conectar e gerenciar os dados armazenados.
3. **Porta do banco de dados**: Este campo requer a porta através da qual o banco de dados estará acessível. A porta padrão varia conforme o tipo de banco de dados utilizado (por exemplo, 3306 para MySQL, 5432 para PostgreSQL). Essa informação é crucial para garantir que a aplicação consiga se comunicar corretamente com o banco de dados.
4. **Usuário do banco**: Neste campo, você deve fornecer o nome de usuário que será utilizado para autenticar a conexão com o banco de dados. O usuário deve ter as permissões necessárias para executar operações no banco.
5. **Senha do banco**: Este é o campo onde você insere a senha associada ao usuário do banco de dados. A senha é essencial para garantir a segurança e proteção dos dados, evitando acessos não autorizados.

É fundamental que todos esses campos sejam preenchidos corretamente, pois informações incorretas podem resultar em falhas de conexão ou na criação do projeto. Após preencher todos os campos, o usuário poderá avançar para a próxima etapa do assistente, onde terá a oportunidade de selecionar as features desejadas para o projeto.

Figura 20 - Guia de instalação LPS: tela para seleção de features.



Fonte: Autoria própria.

A Figura 20 ilustra a segunda etapa, dedicada à inserção de seleção de features do projeto. Nesta fase, é possível escolher as features necessárias para a nova LPS. As features são:

1. ***Authentication***: O usuário pode selecionar entre diferentes métodos de autenticação, como:
    - localDB (Opcional)
    - googleAPI (opcional)
2. ***Role***: É possível escolher diferentes perfis de usuários que terão acesso à aplicação. O perfil administrador é obrigatório. Portanto, essa opção deve estar sempre marcada e não será possível desmarcá-la. As outras opções (graduação, posGraduacao, gestor, servidor, terceirizado, visitante, residente) são opcionais, ou seja, o usuário pode marcar ou desmarcar livremente;
3. ***User***: Aqui são escolhidos usuários mestres iniciais, como: administrador, gestor;
4. ***Schedule***: Tipos de agendamentos disponíveis: diário (obrigatório) e semanal (opcional);
5. ***Meal***: Tipos de refeições: simples (obrigatório), múltiplo (opcional);
6. ***Payments***: Formas de pagamento: PIX, boleto;
7. ***Ratings***: Tipos de avaliação: *form*, fórum.

8. ***Metrics***: Tipos métricas: cartões, gráficos.

Após as duas etapas preenchidas e a confirmação por meio do clique no botão "Concluir", o sistema gerará o projeto utilizando o nome previamente informado na primeira etapa. A estrutura do projeto será organizada da seguinte forma: será criada uma pasta principal com o nome informado, contendo duas subpastas principais, denominadas "api" e "front". As pastas básicas e obrigatórias para o funcionamento do sistema, bem como as *features* selecionadas pelo usuário, serão incluídas no projeto.

O sistema será composto tanto pelas pastas necessárias para o desenvolvimento da camada de *frontend* quanto pelas pastas para o *backend*, abrangendo os respectivos microsserviços que compõem a arquitetura do projeto.

Além disso, será gerado um arquivo de configuração do ambiente, com o nome **.env**, onde serão salvas as informações relativas ao banco de dados que foram fornecidas durante o processo de criação do projeto. As informações incluídas nesse arquivo serão referentes aos seguintes parâmetros:

- Endereço do servidor (DB_HOST);
- Porta de conexão (DB_PORT);
- Nome de usuário (DB_USERNAME);
- Senha de acesso (DB_PASSWORD);

Para que o microsserviço de pagamentos funcione corretamente após a criação do projeto, será necessário adicionar manualmente no arquivo .env os seguintes parâmetros de configuração relacionados ao serviço de pagamentos: **EFIPAY_CLIENT_ID_ [DEV ou PROD]**, **EFIPAY_CLIENT_SECRET_ [DEV ou PROD]**, **EFIPAY_PIX_KEY** e **EFIPAY_PIX_CERT_PATH_ [DEV ou PROD].**

Para que a autenticação com a API do google funcione, é necessário que a chave **CLIENT_ID** também seja inserida manualmente no .env.

Significado das variáveis:

- **EFIPAY_CLIENT_ID_[DEV ou PROD]**: Identificador exclusivo do cliente, utilizado para autenticar o acesso à API no ambiente de desenvolvimento/produção;
- **EFIPAY_CLIENT_SECRET_[DEV ou PROD]**: Chave secreta do cliente, usada em conjunto com o *Client* ID para garantir a segurança da autenticação.
- **EFIPAY_PIX_KEY**: Chave Pix associada à conta, necessária para a realização de transações via Pix.

- **EFIPAY_PIX_CERT_PATH_[DEV ou PROD]**: Caminho para o certificado digital necessário para validação de transações Pix no ambiente de desenvolvimento/produção.
- **CLIENT_ID:** Identificador exclusivo que o Google fornece a um aplicativo registrado para que ele possa utilizar seus serviços de autenticação e autorização por meio do protocolo OAuth 2.0. Esse ID é essencial para permitir que os usuários façam login usando suas contas do Google de forma segura.

Essas variáveis referem-se às credenciais necessárias para a integração com a EFI PAY, um sistema de pagamentos que oferece soluções financeiras, incluindo a geração de boletos, transferências via Pix, e outros meios de pagamento online. O administrador ou gerente da aplicação deve realizar o cadastro no site da EFI PAY para obter as credenciais citadas. Por fim, deve-se instalar as dependências do projeto com npm install e executar os projetos.

5 ESTUDO DE CASO

Esta seção apresenta um estudo de caso que visa obter a percepção do usuário final através de uma instância gerada pela LPS desenvolvida, no contexto específico da UFAL. Na seção 5.1, aborda-se o desenvolvimento do sistema de agendamento de refeições e as features implementadas para os usuários da UFAL. Na seção 5.2, são apresentados gráficos e interpretações dos resultados obtidos no questionário respondido pelos participantes após o uso do sistema.

5.1 Sistema de gerenciamento de refeições no contexto da UFAL

Para avaliar o potencial de utilidade do produto gerado pela LPS, foi criado um sistema de agendamento de refeições utilizando os componentes e recursos que a linha pode oferecer, com exceção das features de Gestão (transferência de saldo entre usuário, notificação de disponibilidade de cardápios e transferência de saldo entre usuários, gerenciamento de reembolso), pois não foi desenvolvida.

No contexto de usuários vinculados à UFAL, o sistema permite que, após o login, os usuários acessem a plataforma para agendamento de refeições, consulta ao cardápio disponível, revisão seus agendamentos anteriores, avaliação de refeições já feitas, além de sua carteira digital. Abaixo serão apresentadas imagens com a interface do sistema.

5.1.1 Telas do sistema para os perfis: estudante, funcionário e visitante

A Figura 21, representa a tela inicial ao sistema. A tela de login:

Figura 21 - Tela inicial de login do sistema

Fonte: autor.

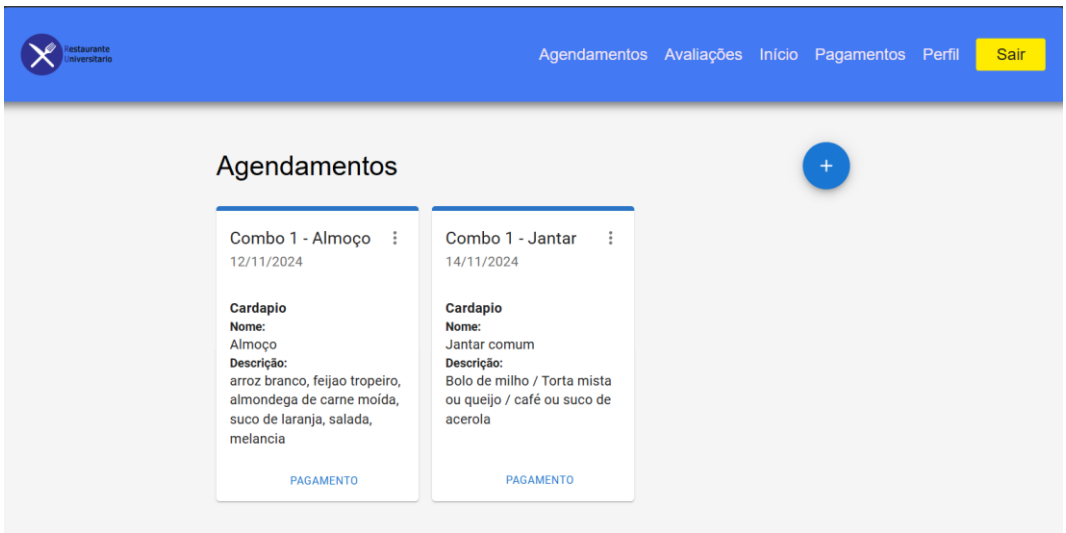
A Figura 22 será apresentado a tela inicial na visão do usuário de perfil estudante, visitante, funcionário.

Figura 22 - Tela inicial do sistema

Fonte: autor.

Para consultar os agendamentos realizados, assim como, agendar novas refeições, o usuário precisa ir à rota de agendamentos no menu do sistema, conforme é possível ver na Figura 23:

Figura 23 - Tela de agendamentos de refeições



Fonte: Autor.

Para agendar uma nova refeição, o usuário deve clicar no botão flutuante azul que possui o símbolo da operação matemática da adição (+). Em seguida, será solicitado o tipo de agendamento podendo ser diário ou semanal. Após a escolha,

deve-se informar a (s) respectiva (s) data (s) e escolher o tipo de refeição, conforme Figura 24:

Figura 24 - Tela de criação de agendamento de refeição

Fonte: Autor.

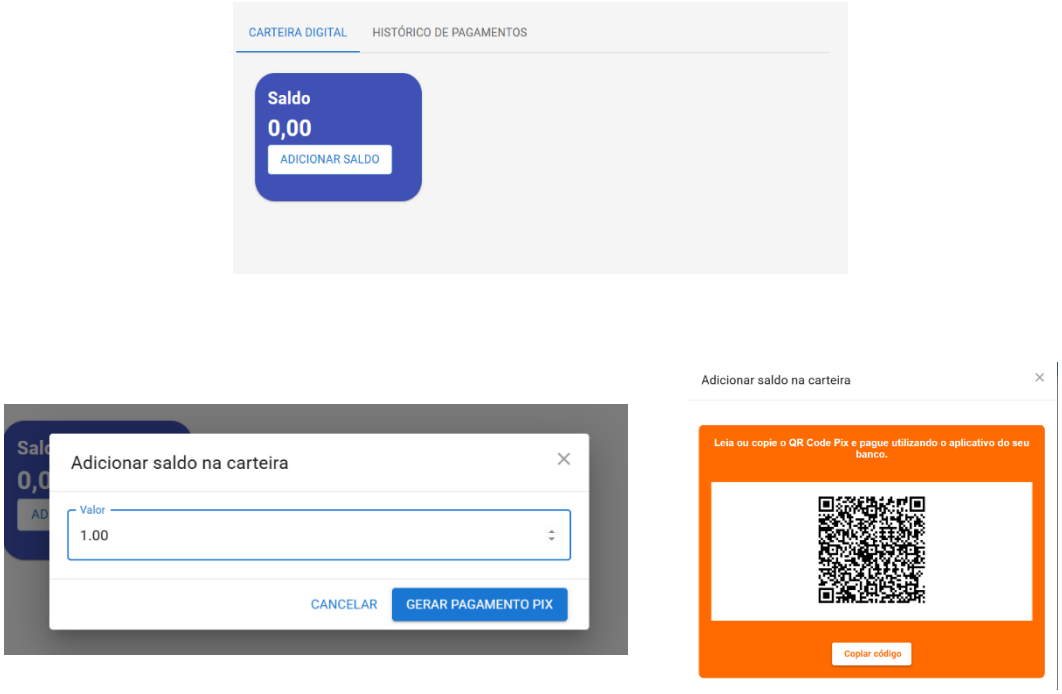
Após o uso da refeição, é possível avaliar o cardápio daquela refeição (ver Figura 25), através da rota de avaliações. O usuário escolhe a refeição que deseja avaliar, em seguida atribui uma nota que varia entre 1 a 5 estrelas e pode deixar um comentário em texto.

Figura 25 - Tela de avaliações de refeições

Fonte: autor.

É possível adicionar créditos à carteira digital (ver Figura 26) na rota de pagamentos. Na carteira digital deve-se clicar em adicionar saldo, em seguida escolher o valor da recarga e escolher a opção pagamento via PIX que será gerado um QR code para o referido pagamento:

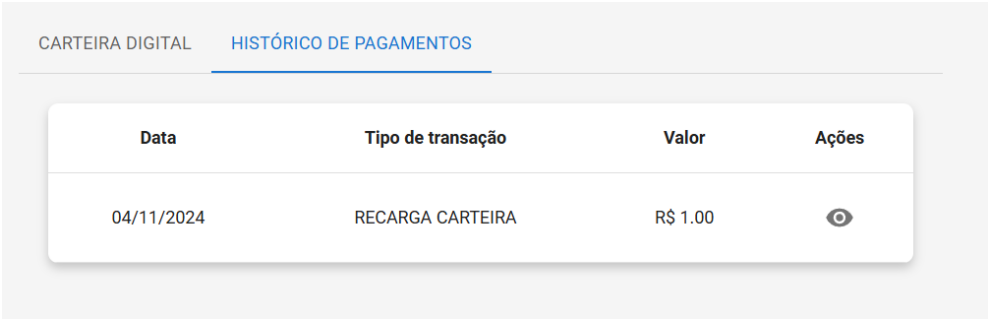
Figura 26 - Adição de saldo na carteira digital



Fonte: Autor.

Também é possível consultar o histórico de tentativas de transações realizadas, conforme Figura 27:

Figura 27 - Tela com o histórico de transações



Fonte: Autor.

O usuário pode alterar a sua senha na rota de perfil, confome Figura 28:

Figura 28 - Tela de perfil do usuário

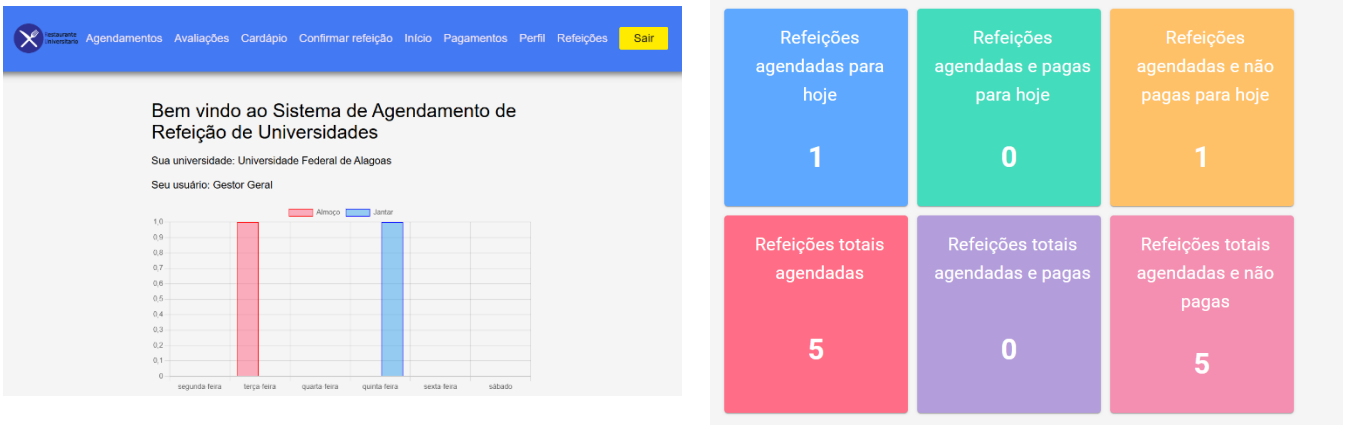
Fonte: Autor.

5.1.2 Telas do sistema para o perfil de gestor

O perfil de usuário gestor possui todos os acessos dos usuários estudante, funcionário e visitante com acréscimo de algumas rotas.

A tela inicial do gestor (ver Figura 29) exibirá um gráfico com as refeições agendadas durante a semana, assim como, métricas das refeições, como é possível notar nas imagens abaixo:

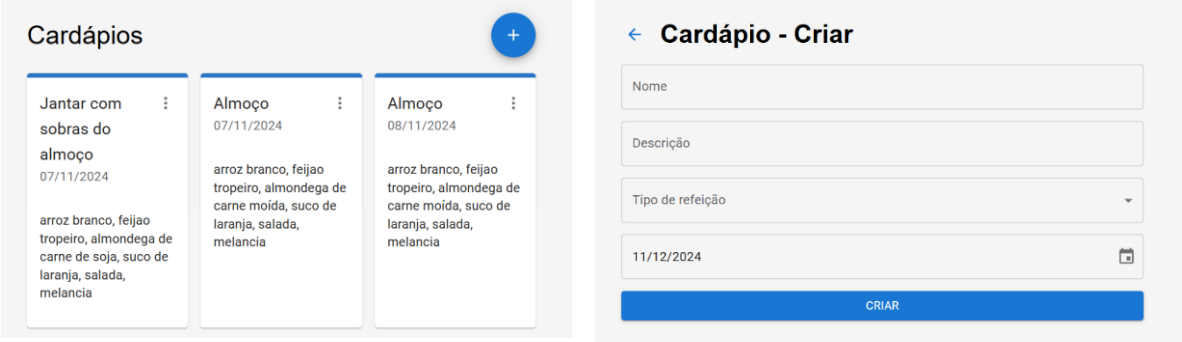
Figura 29 - Tela inicial do sistema para o gestor com gráfico e métricas



Fonte: Autor.

O gestor tem acesso na rota de cardápio (ver Figura 30), a criação, edição e exclusão dos cardápios das refeições disponíveis:

Figura 30 - Tela de listagem dos cardápios e de sua criação



Fonte: autor.

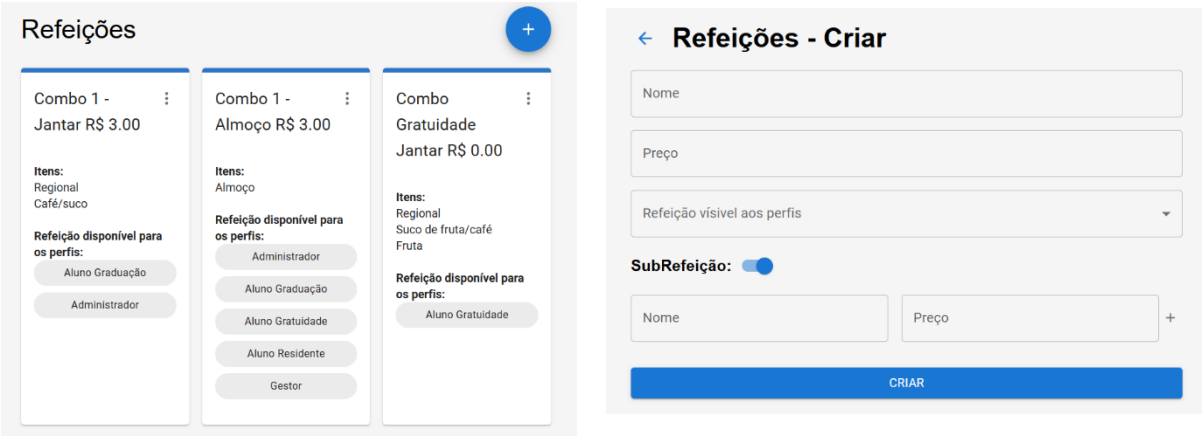
O gestor tem acesso a rota de confirmação de refeição (ver Figura 31) para conferir se o aluno está apto a utilizar a refeição. Para usar essa funcionalidade, deve ser preenchido CPF do aluno que listará as refeições, caso tenham sido previamente agendadas:

Figura 31 - Tela de confirmação de refeição

Fonte: autor.

Também é possível adicionar refeições novas (ver Figura 32), seguindo critérios adotados pela gestão universitária, como: adição de nome da refeição, valor, se haverá itens na respectiva refeição e seu respectivo valor, os perfis de usuários que poderão agendar. Esse recurso está disponível na rota de refeições, conforme ilustra abaixo:

Figura 32 - Tela de listagem de refeições e sua criação



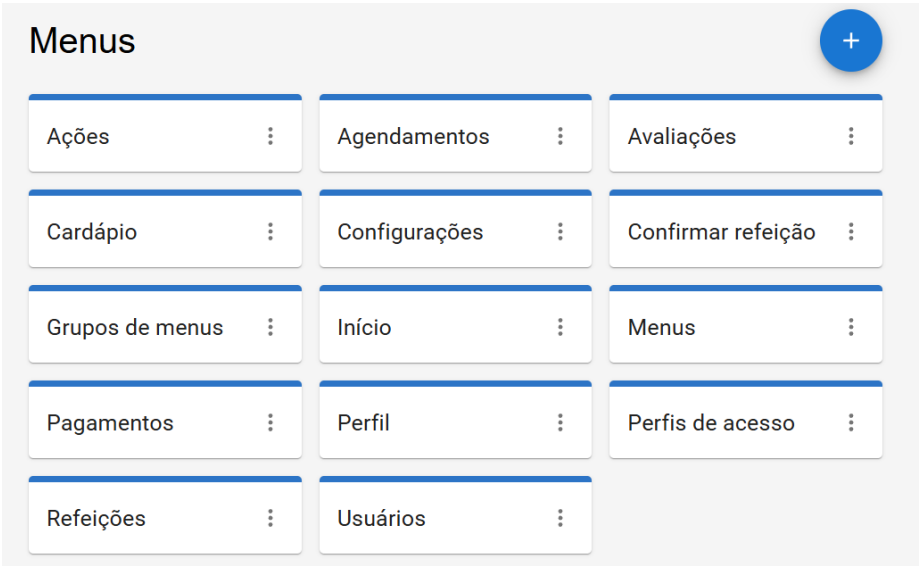
Fonte: autor.

5.1.3 Telas do sistema para o perfil de administrador

O perfil de usuário administrador possui todos os acessos anteriores com acréscimo de algumas rotas de configuração do sistema.

É possível customizar os menus do sistema através da rota menus, conforme Figura 33:

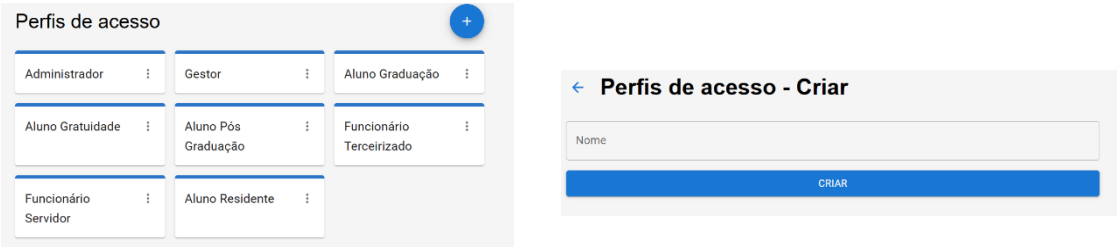
Figura 33 - Listagem dos menus do sistema



Fonte: Autor.

O administrador também pode configurar os perfis de acesso diretamente pelo sistema, através da rota de perfis de acesso, conforme Figura 34:

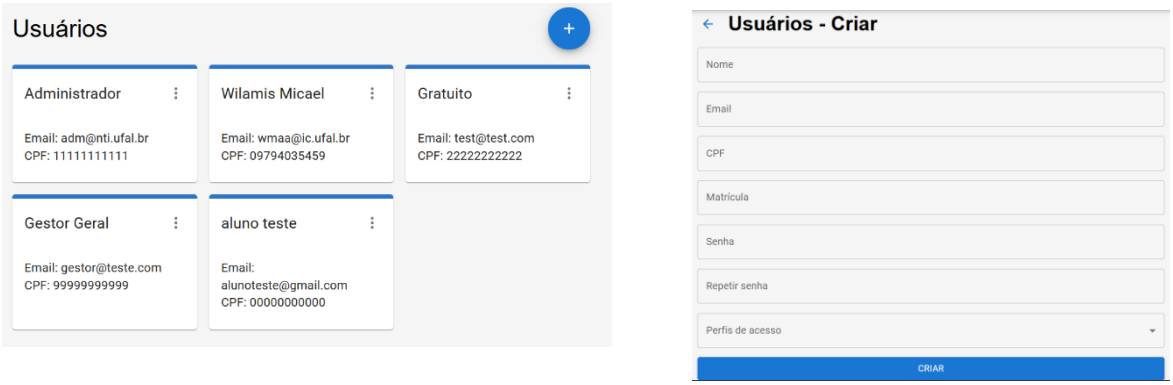
Figura 34 - Listagem dos perfis de acesso e sua adição



Fonte: autor.

É possível gerenciar os usuários cadastros, editar, remover e adicionar novos, na rota de usuários, conforme Figura 35:

Figura 35 - Tela de listagem de usuários e sua inserção



Fonte: Autor.

Também é possível consultar as features do sistema através da rota de configurações, como mostra na Figura 36:

Figura 36 - Tela com as features selecionadas do sistema

Fonte: Autor.

5.1.4 Estudo de caso preliminar do sistema e coleta de feedback com usuários

Devido ao curto tempo disponível, não foi possível colocar em produção o projeto desenvolvido, nem torná-lo público para que a comunidade acadêmica pudesse testá-lo amplamente. Dessa forma, um estudo de caso preliminar foi realizado com o público-alvo do estudo, que foi um total de onze estudantes da universidade dos cursos de Ciência e Engenharia da Computação, Design e Direito.

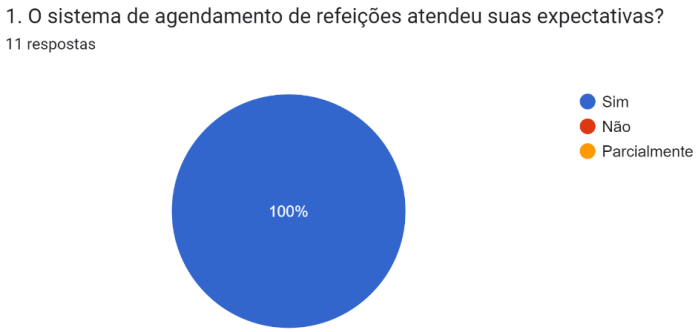
O teste era realizado na sala de convivência do IC ou na CIED. Primeiramente, era explicado aos participantes sobre o objetivo e assunto da pesquisa, logo após, era apresentado o sistema. Em seguida, os usuários puderam explorar o fluxo dele, incluindo funcionalidades como login, visualização do cardápio, agendamento de refeições, pagamentos, avaliação de refeições e acesso ao perfil. Os usuários testaram a aplicação em uma máquina local, em um ambiente de desenvolvimento.

Após essa etapa, eles responderam a um questionário para registrar suas percepções sobre o sistema. As respostas foram obtidas por meio de formulários do Google e incluíram um total de oito perguntas de múltipla escolha. Gráficos com os resultados das respostas serão apresentados na seção seguinte.

5.2 Resultados e discussão

No Gráfico 1, o objetivo foi avaliar se o sistema atendeu às expectativas dos usuários. Observa-se uma resposta unânime de "Sim", indicando um bom alinhamento entre as funcionalidades oferecidas e as expectativas dos estudantes.

Gráfico 1 - Satisfação com o atendimento às expectativas pelo sistema de agendamento de refeições



Fonte: Autor.

Já o Gráfico 2 é mostrado respostas sobre a clareza e simplicidade do sistema. Com 81,8% dos usuários respondendo "Sim, muito claro" e 18,2% "Sim, claro", os resultados indicam uma usabilidade positiva e um processo de agendamento intuitivo.

Gráfico 2 - Clareza e compreensibilidade do processo de agendamento

Fonte: Autor.

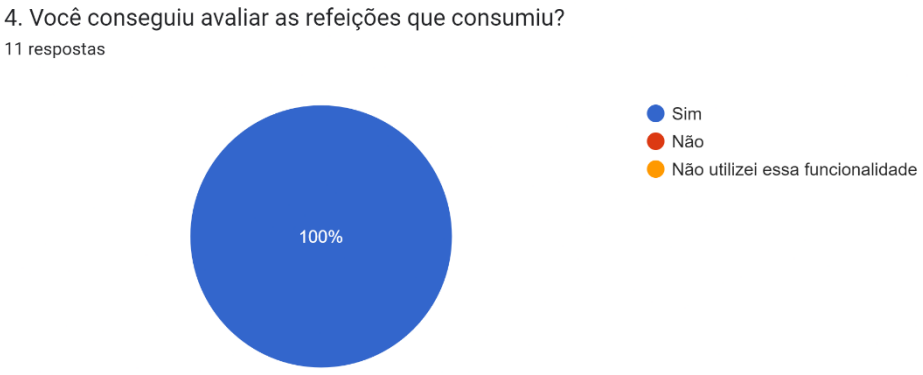
O Gráfico 3 apresenta a experiência geral de uso da funcionalidade principal do sistema. Com 100% dos usuários respondendo "Muito fácil", os dados indicam que o sistema é altamente intuitivo e fácil de operar. Isso signifca que a navegação e o fluxo de uso da funcionalidade estão bem adaptados às necessidades dos usuários, sem apresentar obstáculos significativos.

Gráfico 3 - Avaliação da facilidade de uso do sistema de agendamento

Fonte: Autor.

O Gráfico 4, apresenta respostas da pergunta 4. Esta questão foca no uso da funcionalidade de avaliação das refeições. Com 100% dos usuários respondendo "Sim", o resultado demonstra um alto nível de engajamento com essa funcionalidade. Esse dado indica que os usuários não apenas entendem como utilizar a avaliação, mas também veem valor em dar feedback sobre as refeições, o que pode ser útil para ajustes e melhorias no sistema com base nas preferências dos usuários.

Gráfico 4 - Capacidade dos usuários em avaliar refeições consumidas

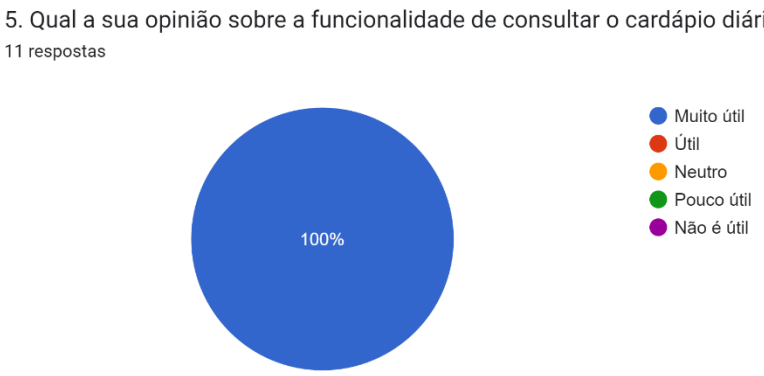


Fonte: Autor.

Também, avaliou-se a utilidade percebida da funcionalidade de consulta ao cardápio. Com 100% dos usuários indicando "Muito útil", o resultado destaca que essa funcionalidade agrega um valor significativo ao sistema, conforme é observado no Gráfico 5. Percebe-se que essa é uma funcionalidade que atende plenamente às expectativas dos estudantes, pois permite que eles visualizem o cardápio com

antecedência, facilitando a decisão sobre o que irão consumir e se o cardápio atende às suas preferências e necessidades nutricionais.

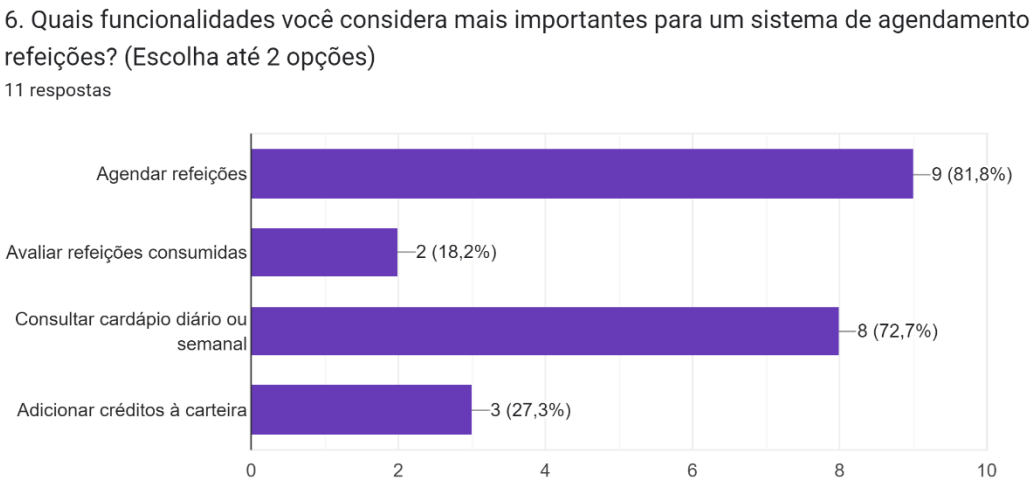
Gráfico 5 - Opinião sobre a funcionalidade de consulta do cardápio diário/semanal



Fonte: Autor.

Abaixo, no Gráfico 6, observa-se a priorização das funcionalidades do sistema, baseada na perspectiva dos usuários. As funcionalidades mais selecionadas indicam áreas consideradas essenciais, auxiliando na alocação de recursos e nos esforços de desenvolvimento. Dentre as 11 respostas coletadas, as funcionalidades mais destacadas foram: agendar refeições (81,8%), consultar cardápio diário ou semanal (72,7%), avaliar refeições consumidas (18,2%) e adicionar créditos à carteira (27,3%).

Gráfico 6 - Funcionalidades mais importantes em um sistema de agendamento de refeições

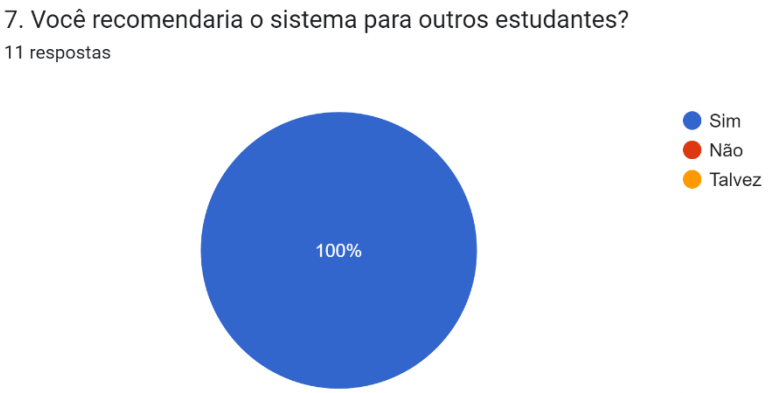


Fonte: Autor.

No Gráfico 7, a pergunta avaliou se os usuários recomendariam o sistema para outras pessoas. Com 100% das respostas indicando "Sim", isso reflete um alto nível

de satisfação e confiança dos usuários, sugerindo que o sistema atende bem às expectativas e possui grande potencial para expansão e aceitação no público-alvo.

Gráfico 7 - Intenção de recomendação do sistema para outros estudantes



Fonte: Autor.

Por fim, o Gráfico 8 apresenta a percepção dos estudantes sobre a importância de um sistema de agendamento de refeições na Universidade Federal de Alagoas. Com 81,8% dos respondentes considerando-o 'muito importante' e 18,2% como 'importante', os resultados reforçam a relevância desse sistema para a comunidade acadêmica.

Gráfico 8 - Percepção sobre a importância de um sistema de agendamento na UFAL

Fonte: Autor.

Em síntese, os resultados obtidos pelas perguntas refletem uma percepção positiva dos usuários, indicando a relevância e a importância da UFAL dispor de um sistema dedicado ao gerenciamento dos agendamentos de refeições. Esses dados

reforçam o valor de uma solução tecnológica que facilite a organização e o acesso às refeições no campus, atendendo às necessidades da comunidade acadêmica.

6 CONCLUSÃO

Este trabalho desenvolveu uma LPS integrada à arquitetura de microsserviços, para a criação de um sistema de agendamento de refeições. O sistema proposto tem potencial de melhorar o gerenciamento de refeições, oferecendo possíveis soluções para desafios comuns como controle de acesso, divulgação de cardápio.

Os resultados obtidos, a partir de uma instância da LPS e da pesquisa feita junto aos estudantes da UFAL, confirmaram a aceitação da solução. Esses dados sugerem que a UFAL deveria adotar um sistema de agendamento de refeições baseado na arquitetura desenvolvida, pois isso pode contribuir para a melhoria na gestão do restaurante universitário.

Na Tabela 3, é apresentado um comparativo entre os trabalhos relacionados previamente analisados e o trabalho atual desenvolvido, destacando as diferenças e contribuições:

Tabela 3 - Comparativo de funcionalidade e implementações entre os trabalhos apresentados e o trabalho proposto

Autor	SIQUEIRA (2020)	BRITO (2020)	CARDOSO (2021)	MURTA (2022)	TRABALHO ATUAL
LPS					X
Microsserviços					X
Agendamentos	X	X		X	X
Avaliação	X		X		X
Cardápio	X	X			X
Refeições	X				X
Suporte a Pagamentos	X				X
Métricas ou relatórios		X	X		X
Carteira digital	X				X

Fonte: autor.

O diferencial deste trabalho foi a implementação de um sistema de agendamento de refeições utilizando uma abordagem baseada em LPS e arquitetura de microsserviços, o que possibilita maior flexibilidade e customização em

comparação com as soluções encontradas na literatura, além de implementar todas as funcionalidades descritas na tabela.

Como limitação, destaca-se que, por não ter sido implantado em ambiente de produção, o *software* foi inteiramente desenvolvido e testado em ambiente de desenvolvimento. Além disso, a funcionalidade de adição de saldo à carteira e pagamento de refeições descontando diretamente do saldo não pôde ser testada, pois dependia do uso de webhooks da API de pagamentos EfiPay, o que exigiria que a aplicação estivesse em um domínio publicado com HTTPS. A necessidade de um domínio com HTTPS deve-se aos requisitos de segurança da API de pagamentos, que realiza duas requisições ao servidor da aplicação.

Para melhorias e trabalhos futuros, é possível avaliar a instância com um perfil heterogêneo de participantes (alunos, servidores), avaliar o esforço para instanciação e evolução da LPS como: custo inicial, custo médio à medida que os números de produtos aumentam, realizar uma revisão sistemática da literatura para avaliação da LPS, avaliar outros requisitos como: escalabilidade (por ser um potencial da arquitetura de microsserviços), a implementação das features não feitas como a de gestão: transferência de saldo entre usuário, notificação de disponibilidade de cardápios, transferência de saldo entre usuários e gerenciamento de reembolso. Na área de *frontend*: a construção de uma LPS para o *frontend* com foco na acessibilidade (surdez, cegueira, baixa visão, dificuldade motora) e a utilização em diferentes dispositivos (*mobile* e *web*).

REFERÊNCIAS BIBLIOGRÁFICAS

ANJOS, Arthur Silva Paes Barreto dos. **Uma arquitetura de linha de produto de software para ambientes virtuais de aprendizagem**. 2011. Monografia (Bacharelado em Ciência da Computação) – Instituto de Computação, Universidade Federal de Alagoas, Maceió, 2011. Disponível em: https://ic.ufal.br/pt-br/graduacao/ciencia-da-computacao/documentos/banco-de-monografias/2011/uma-arquitetura-de-linha-de-produto-de-software-para-ambientes-virtuais-de-aprendizagem/view. Acesso em 07 ago. 2024.

APEL, S. et al. **Feature-Oriented Software Product Lines**. Springer. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.

ARAÚJO, Diego O.; SCHMITZ, Eber A.; CORREA, Alexandre L.; ALENCAR, Antonio J.. **Elaboração de Especificações de Casos de Uso para Linhas de Produto de Software Baseada em Fragmentos**. *In*: SIMPÓSIO BRASILEIRO DE COMPONENTES, ARQUITETURAS E REUTILIZAÇÃO DE SOFTWARE (SBCARS), 3. , 2009, Natal/RN. **Anais** [...]. Porto Alegre: Sociedade Brasileira de Computação, 2009 . p. 24-37. DOI: https://doi.org/10.5753/sbcars.2009.24099.

BABAR, Muhammad Ali; CHEN, Lianping; SHULL, Forrest. **Managing Variability in Software Product Lines**. (2010). IEEE Software, 27(3), 89-91, 94. Disponível em: https://doi.org/10.1109/MS.2010.77.

BARROS, Kaic de Oliveira. **Uma linha de produto de software para o turismo em Sergipe**. 2021. Trabalho de Conclusão de Curso (Graduação em Sistemas de Informação) - Universidade Federal de Sergipe, Itabaiana, 2021. Disponível em: https://ri.ufs.br/jspui/handle/riufs/15159. Acesso em 01 nov. 2024.

BECKER, Alex Malmann. **O impacto do uso de micro serviços na evolução de uma linha de produto de software**. 2019. Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de São Carlos, São Carlos, 2019. Disponível em: https://repositorio.ufscar.br/handle/ufscar/11978. Acesso em 03 out. 2024.

BRITO, Gilzane Caetano de. **Carteirinha na mão: uma proposta de aplicativo para apoiar o fornecimento de refeições no IFBA - Campus Seabra**. Monografia .(Técnico em Informática) Instituto Federal de Ciência e Tecnologia da Bahia. Seabra, BA. 2020. f. 80.

BOSCH, Jan. **Design and use of software architectures: adopting and evolving a product-line approach**. 2000. ACM Press/Addison-Wesley Publishing Co., USA. Disponível em: https://dl.acm.org/doi/10.5555/339362. Acesso em: 30 set. 2024.

CANTELON, M. et al. **Node. js in Action**. [S.l.]: Manning Publications, 2017.

CARDOSO, João Vitor Costa. **Implementação de sistema web para controle de restaurante universitário**. 2021. Monografia (Licenciatura em Computação e Informática) - Universidade Federal Rural do Semi-Árido, Pró-Reitoria de Graduação, Centro Multidisciplinar de Angicos, Angicos, 2021.

CHIARADIA, Luiz Felipe Correa; MACEDO, Douglas Dyllon Jeronimo; DUTRA,

Moisés Lima. **Uma proposta de arquitetura de microsserviços aplicada em um sistema de CRM social**. Encontros Bibli: revista eletrônica de biblioteconomia e ciência da informação, *[S. l.]*, v. 23, n. 53, p. 147–159, 2018. DOI: 10.5007/1518-2924.2018v23n53p147. Disponível em: https://periodicos.ufsc.br/index.php/eb/article/view/1518-2924.2018v23n53p147. Acesso em: 4 out. 2024.

CLEMENTS, P. C.; NORTHROP, L. **Software Product Lines: Practices and Patterns**. Addison-Wesley, 2001. 563 p. (SEI series in software engineering). ISSN 13652648. ISBN 0-201-70332-7. Disponível em: https://books.google.com.br/books?id= tHGFQgAACAAJhttp://dl.acm.org/citation.cfm?id=501065.

DB-Engines. **Ranking de sistemas de gerenciamento de banco de dados**. 2024. Disponível em: https://db-engines.com/en/ranking. Acesso em: 23 set. 2024.

DEURSEN, Arie van; KLINT, Paul. **Domain-specific language design requires feature descriptions**. *Journal of Computing and Information Technology*, v. 10, 2001. Disponível em: http://cit.fer.hr/index.php/CIT/article/view/1465/0. Acesso em: 15 set. 2024.

DOMINGOS, L. L.; FARINA, R. M. **MICROSERVIÇOS: um estudo de caso apontando suas potencialidades**. Revista Interface Tecnológica, [S. l.], v. 17, n. 2, p. 18–30, 2020. DOI: 10.31510/infa.v17i2.851. Disponível em: https://revista.fatectq.edu.br/interfacetecnologica/article/view/851. Acesso em: 4 out. 2024.

FAMILIAR, B. **Microservices, IoT, and Azure: Leveraging DevOps and Microservice Architecture to Delicer SaaS Solutions.** 1.ed. New York: Apress Media, 2015.

GITHUB. **Sistema de agendamento de refeições baseado na arquitetura em microsserviço com LPS criado: University Meal Scheduler (UMS)**. Disponivel em: https://github.com/WilamisAviz15/ums. Acesso em 25 nov. 2024.

GUEDES, T. **UML 2 guia prático**. São Paulo: Novatec, 2007.

JACOBSON, I. **Unified Software Development Process**. Boston: Addison-Wesley, 1999.

KANG, K.; COHEN, S.; HESS, J.; NOVAK, W.; PETERSON, A. **Feature-oriented domain analysis (FODA) feasibility study**. 1990. Carnegie Mellon University, Software Engineering Institute's Digital Library. Technical Report CMU/SEI-90-TR-021. Disponível em: https://insights.sei.cmu.edu/library/feature-oriented-domain-analysis-foda-feasibility-study/. Acesso em: 15 set. 2024.

LEWIS, J.; FOWLER, M. **Microservices: a definition of this new architectural term**. *Mars*, 2014.

LINDEN, F. van der; SCHMID, K.; ROMMES, E. **Software Product Lines in Action**. Science, v. 41, n. 6193, p. 330, 7 2007. ISSN 0036-8075.

MALIPENSE, L. M.; ZUCHI, J. D. **UM ESTUDO SOBRE O CONCEITO E A APLICAÇÃO DA ARQUITETURA DE MICROSERVIÇOS**. Revista Interface Tecnológica, [S. l.], v. 15, n. 1, p. 122–134, 2018. DOI: 10.31510/infa.v15i1.324. Disponível em: https://revista.fatectq.edu.br/interfacetecnologica/article/view/324. Acesso em: 3 out. 2024.

MENDONÇA JÚNIOR, João Alves. **Uma linha de produto de software para micro e pequenas empresas**. 2022.Trabalho de Conclusão de Curso (Graduação em Sistemas de Informação) - Universidade Federal de Sergipe, Itabaiana, 2022. Disponível em: http://ri.ufs.br/jspui/handle/riufs/16910. Acesso em 01 nov. 2024.

MOREIRA, P. F. M.; BEDER, D. M. **Desenvolvimento de aplicações e micro serviços: Um estudo de caso**. Revista TIS, v. 4, n. 3, 2016.

MOON, M.; YEOM, K.; CHAE, H. **An approach to developing domain requirements as a core asset based on commonality and variability analysis in a product line**. IEEE Transactions on Software Engineering, v. 31, n. 7, p. 551–569, 2005.

MURTA, Lucas Pimenta Santana. **Desenvolvimento de sistema para confirmação do uso do restaurante universitário pela comunidade acadêmica**. 2022. 23 f. Trabalho de Conclusão de Curso (Graduação em Sistemas de Informação) - Instituto de Computação, Universidade Federal Fluminense, Niterói, 2022.

NAVARRO, Tiago adelino; REINEHR, Sheila dos Santos; PALUDO, Marco Antônio. **Linhas de produto de software no desenvolvimento de ERP: estudos de caso em empresas no Brasil**. 2019. 260 f. Dissertação (Mestrado) - Pontifícia Universidade Católica do Paraná, Curitiba, 2019 Disponível em: https://archivum.grupomarista.org.br/pergamumweb/vinculos//000080/00008040.pdf. Acesso em: 11 out. 2024.

NEWMAN, S. ***Building microservices: designing fine-grained systems***. [S.l.]: "O'Reilly Media, Inc.", 2015.

POHL, KLAUS; BOCKLE, GUNTER; LINDEN, FRANK VAN DER. **Software Product Line Engineering: Foundations, Principles, and Techniques**. Springer; Softcover reprint of hardcover 1st ed. 2005. Disponível em: https://books.google.com.br/books?id=lsX8_O_TRkEC&printsec=copyright#v=onepage&q&f=false. Acesso em 01 out. 2024.

PRESSMAN, R.; MAXIM, B. **Engenharia de Software**. 8. ed. São Paulo: McGraw Hill Brasil, 2016.

SILVA, F. A. P. ; SILVEIRA NETO, P. A. M. ; GARCIA, Vinicius Cardoso ; MUNIZ, P. F. **Linhas de produto de software: Uma tendência da indústria**. In: André Macêdo Santana; Pedro de Alcântara dos Santos Neto; Raimundo Santos Moura. (Org.). Escola Regional de Informática Ceará, Maranhão, Piauí: Livro texto dos minicursos,

07 e 08 de novembro de 2011. Teresina: Sociedade Brasileira de Computação, 2011, p. 7-31.

SIQUEIRA, Bruno Viana de. **RUnB: aplicativo para o restaurante universitário da Universidade de Brasília**. 2020. 77 f., il. Trabalho de Conclusão de Curso (Bacharelado em Engenharia da Computação) — Universidade de Brasília, Brasília, 2020. Disponível em: https://bdm.unb.br/handle/10483/36296. Acesso em 01 out. 2024.

SOARES, Thiago Coelho; SOARES, João Coelho; SOARES, Sandro Vieira. **Pesquisa quantitativa em turismo: os dados gerados são válidos e confiáveis?**. RITUR - Revista Iberoamericana de Turismo, *[S. l.]*, v. 9, n. 1, p. 162–174, 2019. Disponível em: https://www.seer.ufal.br/index.php/ritur/article/view/6974. Acesso em: 30 out. 2024.

SOUZA, Dalva Inês de; MÜLLER, Deise Margô; FRACASSI, Maria Angélica Hiele; ROMEIRO, Solange Bianco Borges. **Manual de orientações para projetos de pesquisa**. Novo Hamburgo: FESLSVC, 2013. Disponível em: https://www.liberato.com.br/manual-de-trabalhos-cientificos/. Acesso em: 30 out. 2024.

TUAPE, Micheal; HASHEELA-MUFETI, Victoria; IIYAMBO, Petrus; KAYANDA, Anna; MENSAH, Sharon; KASURINEN, Jussi. **Software Practice in Small Software Companies: Development Context Constraints on Process Adoption**. In: Proceedings of the 2022 European Symposium on Software Engineering (ESSE '22), New York: Association for Computing Machinery, 2023. p. 1–9. Disponível em: https://doi.org/10.1145/3571697.3571698. Acesso em: 01 nov. 2024.

TYPESCRIPT. **TypeScript for the New Programmer**. 2024. Disponível em: https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html. Acesso em: 01 out 2024.

APÊNDICE A – Pesquisa de experiência do usuário: sistema de refeições da UFAL

Pesquisa de Experiência do Usuário: Sistema de Refeições da UFAL

wmaa@ic.ufal.br Mudar de conta

Não compartilhado

* Indica uma pergunta obrigatória

1. O sistema de agendamento de refeições atendeu suas expectativas? *

- ◯ Sim
- ◯ Não
- ◯ Parcialmente

2. O processo de agendar uma refeição foi claro e compreensível para você? *

- ◯ Sim, muito claro
- ◯ Sim, claro
- ◯ Neutro
- ◯ Não, confuso
- ◯ Não, muito confuso

3. Como você avaliaria a facilidade de uso da funcionalidade de agendamento de refeições? *

- ◯ Muito fácil
- ◯ Fácil
- ◯ Neutro
- ◯ Difícil
- ◯ Muito difícil

4. Você conseguiu avaliar as refeições que consumiu? *

- ◯ Sim
- ◯ Não
- ◯ Não utilizei essa funcionalidade

5. Qual a sua opinião sobre a funcionalidade de consultar o cardápio diário ou semanal? *

- ◯ Muito útil
- ◯ Útil
- ◯ Neutro
- ◯ Pouco útil
- ◯ Não é útil

6. Quais funcionalidades você considera mais importantes para um sistema de agendamento de refeições? (Escolha até 2 opções) *

- [ ] Agendar refeições
- [ ] Avaliar refeições consumidas
- [ ] Consultar cardápio diário ou semanal
- [ ] Adicionar créditos à carteira

7. Você recomendaria o sistema para outros estudantes? *

- ◯ Sim
- ◯ Não
- ◯ Talvez

8. Você acha importante que a Universidade Federal de Alagoas possua um sistema de agendamento de refeições como este? *

- ◯ Muito importante
- ◯ Importante
- ◯ Neutro
- ◯ Pouco importante
- ◯ Não é importante

Enviar Página 1 de 1 Limpar formulário

APÊNDICE B – Orquestração de microsserviços com *docker swarm* e API *gateway*

Neste apêndice, será detalhado como a orquestração de microsserviços é realizada utilizando *docker swarm* e uma API g*ateway*. Será apresentado, a configuração técnica e a interação entre os microsserviços gerenciada pelo gateway.

A API Gateway atua como o ponto único de entrada para o sistema, recebendo as requisições dos clientes, roteando-as para os microsserviços adequados. No projeto, o *NestJS* foi utilizado para implementar o gateway, enquanto os microsserviços foram gerenciados pelo d*ocker swarm*, que realiza a orquestração e *deploy* dos serviços em contêineres.

O *gateway* está implantado para se comunicar com os microsserviços via transporte TCP, fornecendo um modelo de comunicação síncrono. Ao utilizar o *NestJS*, a abstração *ClientProxy* permite que o *gateway* envie mensagens diretamente aos microsserviços através do método *send*().

Abaixo está a configuração do *gateway* para interação com o microsserviço de autenticação:

```
      @Module({
        imports: [
          ClientsModule.register([
            {
              name: 'AUTHENTICATION',
              transport: Transport.TCP,
              options: { port: 4006 },
            },
          ]),
        ],
      })
```

Onde:

- *name*: Nome usado para injetar o cliente proxy. Para cada microsserviço será uma assinatura diferente;
- *transport*: Define o transporte como TCP.
- *options*: Porta onde o microsserviço de autenticação está escutando.

Ao usuário fazer uma requisição para a rota `auth/login`, ela será encaminhada para o microsserviço de autenticação, através do método *send()* :

```
```

```
  constructor(@Inject('AUTHENTICATION') private readonly msAu-
thentication: ClientProxy) {}
  login(data: LoginDto) {
    return this.msAuthentication.send('login', data);
  }
```

No microsserviço de autenticação, a mensagem enviada pelo gateway é processada por meio do decorador **@MessagePattern**:

```
@Controller()
export class AuthenticationController {
  constructor(private readonly service: AuthenticationService)
{}
  @MessagePattern('login')
  login(
    @Body() @Body() data: LoginDto,
  ): Promise<{ user: UserJwtInterface; accessToken: string }> {
    return this.service.login(data);
  }
}
```

Isso acontece para todos os microsserviços.

Orquestrador de microsserviços: *docker swarm*

Cada microsserviço está em um container no *docker*. A sua infraestrutura utiliza *docker swarm*, um orquestrador nativo do *docker*, para gerenciar o ciclo de vida das aplicações. No arquivo “docker-swarm.yml”, cada microsserviço é configurado como um serviço independente, permitindo escalabilidade e gerenciamento centralizado.

O arquivo de configuração do orquestrador é extenso. Portanto, a seguir será apresentado apenas um microsserviço, sendo que a lógica se aplica da mesma forma aos demais:

```
ms-authentication:
    image: itswillmica/ums:authentication
    ports:
      - '3006:3000'
      - '4006:3006'
    deploy:
      replicas: 1
      resources:
        limits:
          cpus: '0.5'
          memory: 100M
```

```
      update_config:
        parallelism: 1
        delay: 10s
      restart_policy:
        condition: on-failure
    networks:
      - app-network
```

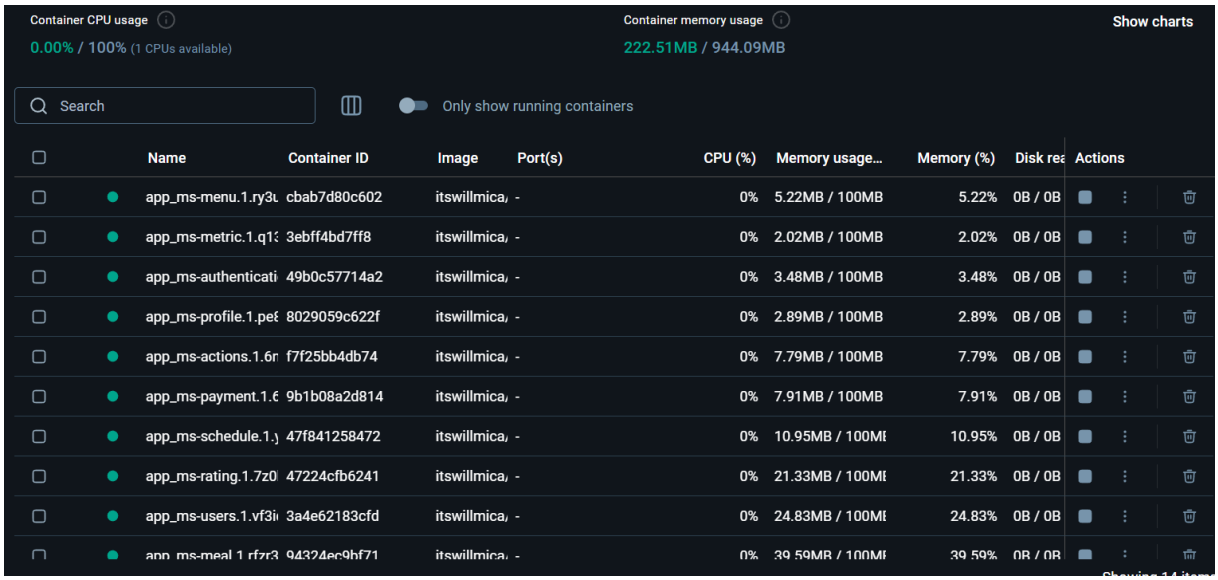
Onde:

- *Image*: Define a imagem *docker* que será usada para criar o container. A imagem já foi criada e está armazenada no *docker hub* no repositório de nome itswillmica/ums e a tag authentication;
- *Ports*: Mapeia as portas dentro do container para uma respectiva porta na máquina host (onde o Docker está rodando);
- *Deploy*: Esta seção é específica para Docker Swarm, que permite definir parâmetros de deploy para os serviços:
    - *replicas*: Define o número de réplicas (instâncias) do serviço que será executado;
    - *resources*: define o número de cpu e tamanho de memória que será utilizado;
    - update_config: Define a estratégia de atualização para o serviço;
    - restart_policy: Define a política de reinício do container, como em caso de falha.
- *Networks*: Define a rede em que o serviço estará conectado. app-network deve ser uma rede previamente definida no arquivo docker swarm. Isso permite que o serviço ms-authentication se comunique com outros serviços dentro da mesma rede.

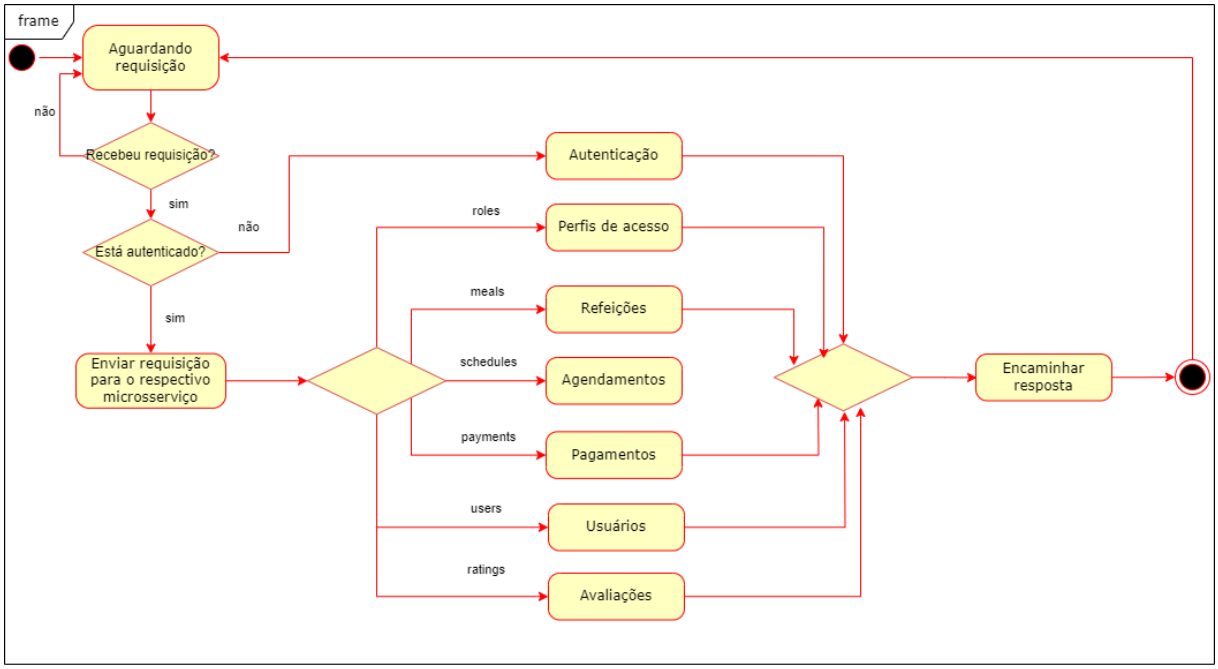
É possível consultar microsserviços que estão rodando pelo comando `docker service ls`, através do terminal:

```
PS C:\Users\Wilamis Aviz> docker service ls
ID             NAME                    MODE         REPLICAS   IMAGE                                PORTS
kru1mj05wp4x   app_ms-actions          replicated   1/1        itswillmica/ums:action               *:3003->3000/tcp, *:4003
->3003/tcp
wpwhc3jsqx56   app_ms-authentication   replicated   1/1        itswillmica/ums:authentication       *:3006->3000/tcp, *:4006
->3006/tcp
3r0kt8v85sda   app_ms-meal             replicated   1/1        itswillmica/ums:meal                 *:3007->3000/tcp, *:4007
->3007/tcp
krdib45zhtrf   app_ms-menu             replicated   1/1        itswillmica/ums:menu                 *:3004->3000/tcp, *:4004
->3004/tcp
lstd893w7uax   app_ms-metric           replicated   1/1        itswillmica/ums:metric               *:3013->3000/tcp, *:4013
->3013/tcp
rhl01h551rzw   app_ms-payment          replicated   1/1        itswillmica/ums:payment              *:3011->3000/tcp, *:4011
->3011/tcp
wp5cqeljnq9m   app_ms-profile          replicated   1/1        itswillmica/ums:profile              *:3005->3000/tcp, *:4005
->3005/tcp
qx143e3ltzt6   app_ms-rating           replicated   1/1        itswillmica/ums:rating               *:3010->3000/tcp, *:4010
->3010/tcp
5v7fs1ha5ckb   app_ms-role             replicated   1/1        itswillmica/ums:role                 *:3002->3000/tcp, *:4002
->3002/tcp
z86qhzncxnbf   app_ms-schedule         replicated   1/1        itswillmica/ums:schedule             *:3008->3000/tcp, *:4008
->3008/tcp
2h05h1zcug0t   app_ms-user-role        replicated   1/1        itswillmica/ums:user-role            *:3009->3000/tcp, *:4009
->3009/tcp
4uh5tmukk0mq   app_ms-users            replicated   1/1        itswillmica/ums:users                *:3001->3000/tcp, *:4001
->3001/tcp
```

Outra maneira de consultar os containers é pelo aplicativo Docker desktop:



APÊNDICE C – Diagrama de estados API *gateway*



APÊNDICE D – Código da criação de uma aplicação a partir da LPS

Será apresentado o código que realiza a criação de uma aplicação a partir da LPS gerada.

Quando o usuário preenche as informações do projeto e seleciona as features no *wizard guide* (guia de instalação), a requisição é enviada para a rota `config/generate`, no qual é gerado um projeto novo de acordo com os parâmetros preenchidos (nome do projeto, nome do banco, tipo de banco, porta, usuário e senha) e de acordo com as features selecionadas, elas serão adicionadas à pasta do novo projeto. O método responsável por lidar com a requisição citada é mostrado abaixo:

```
  async createProject(config: ConfigInterface) {
    const { name, database, selectedModules } = config;
    const newProjectPath = path.join(process.cwd(), name, 'api');
    fs.mkdirSync(newProjectPath, { recursive: true });
    execSync(`nest new . --package-manager npm`, {
      cwd: newProjectPath,
      stdio: 'inherit',
    });
    const basePath = path.join(process.cwd());
    const newSrcPath = path.join(newProjectPath, 'src');
    this.copyCoreFiles(basePath, newProjectPath);
    this.copySelectedModules(basePath, newSrcPath, selectedMod-
ules, newProjectPath);
    this.configureDatabase(newProjectPath, database);
     execSync(`cd ${newProjectPath} && npm install`, { stdio:
'inherit' });
    if (selectedModules.RoleModule?.active) {
       this.overwriteRoleSeedFileInMicroservice(newProjectPath,
selectedModules.RoleModule.options);
    }
    if (selectedModules.UserModule?.active) {
       this.overwriteUserSeedFileInMicroservice(newProjectPath,
selectedModules.UserModule.options);
    }

    const filteredModules = {
       AuthenticationModule: selectedModules.AuthenticationMod-
ule,
      RoleModule: selectedModules.RoleModule,
      ScheduleModule: selectedModules.ScheduleModule,
      MealModule: selectedModules.MealModule,
      RatingModule: selectedModules.RatingModule,
      PaymentsModule: selectedModules.PaymentsModule,
    };
     const modulesConfigPath = path.join(newSrcPath, 'config',
'modules-config.json');
```

```
     fs.mkdirSync(path.dirname(modulesConfigPath), { recursive:
true });
       fs.writeFileSync(modulesConfigPath, JSON.stringify(fil-
teredModules, null, 2), 'utf8');
    this.copyReactFrontend(name, selectedModules);
    return {
       message: `Projeto ${name} criado com sucesso dentro de
${name}/api!`,
    };
  }
```

A função acima, `createProject()`, tem a responsabilidade de coletar os dados enviados pela *wizard guide* e gerar um novo projeto.

A reutilização da arquitetura baseada em microsserviços é feita através dos métodos: `copyCoreFiles()`, `copySelectedModules()`, `configureDatabase()`, `overwriteRoleSeedFileInMicroservice()`,`copyReactFrontend()`, que são invocados dentro do `createProject()`.

Serão apresentados dois métodos (`copySelectedModules()`, `configureDatabase()`) e os demais serão otimidos, pois são extensos, mas estão disponíveis no link do repositório do projeto: https://github.com/WilamisAviz15/ums/blob/main/api/src/config/config.service.ts.

Método de configurar informações do banco de dados e de clonagem das features selecionadas juntamente com os respectivos módulos necessários:

```
private configureDatabase(newProjectPath: string, databaseName:
DatabaseConfigInterface) {
    const envPath = path.join(newProjectPath, '.env');
    const envContent = `DATABASE_TYPE=${databaseName.type}\nDA-
TABASE_NAME=${databaseName.name}\nDATABASE_HOST=localhost\nDA-
TABASE_PORT=${databaseName.port}\nDATABASE_USER=${database-
Name.username}\nDATABASE_PASSWORD=${databaseName.password}`;
    fs.writeFileSync(envPath, envContent);
  }

private copySelectedModules(basePath: string, newSrcPath:
string, selectedModules: SelectedModules, newProjectPath?:
string) {
    const modulesPath = path.join(basePath, 'src', 'modules');
     const microservicesPath = path.join(basePath, 'microsser-
vices');
    const ignoredModulesForMicroservices = ['submeals', 'menus-
group', 'menus-meals', 'meals-user-roles'];
```

```
    for (const [moduleName, moduleConfig] of Object.entries(selectedModules)) {
      if (moduleConfig.active) {
          const moduleFolderName = moduleConfig.name.toLowerCase();
          const moduleFolder = path.join(modulesPath, moduleFolderName);
          const destinationFolder = path.join(newSrcPath, 'modules', moduleFolderName);
           const moduleFolderMicroservices = path.join(microservicesPath, moduleFolderName);
         const destinationFolderMicroservices = path.join(newProjectPath, 'microsservices', moduleFolderName);

        if (fs.existsSync(moduleFolder)) {
          fs.mkdirSync(destinationFolder, { recursive: true });
          this.copyDirectory(moduleFolder, destinationFolder);
        } else {
         console.warn(`Módulo não encontrado: ${moduleFolder}`);
        }
           if (!ignoredModulesForMicroservices.includes(moduleFolderName)) {
          if (fs.existsSync(moduleFolderMicroservices)) {
             fs.mkdirSync(destinationFolderMicroservices, { recursive: true });
             this.copyDirectory(moduleFolderMicroservices, destinationFolderMicroservices);
          } else {
             console.warn(`Módulo não encontrado: ${moduleFolderMicroservices}`);
          }
        }
      }
    }
  }
```

APÊNDICE E – Código do microsserviço de autenticação

Será apresentado o código de um dos microsserviços criados. Neste caso, o de autenticação no sistema.

Arquivo authentication.controller.ts:

```
import { Body, Controller } from '@nestjs/common';
import { MessagePattern } from '@nestjs/microservices';

import { AuthenticationService } from './authentication.service';
import { LoginDto } from './dtos/login.dto';
import { UserJwtInterface } from './interfaces/user-jwt.inter-
face';

@Controller()
export class AuthenticationController {
  constructor(private readonly service: AuthenticationService)
{}

  @MessagePattern('login')
  login(
    @Body() @Body() data: LoginDto,
  ): Promise<{ user: UserJwtInterface; accessToken: string }> {
    return this.service.login(data);
  }
}

```

Arquivo authentication.service.ts:

```
import { HttpException, HttpStatus, Injectable, Unauthor-
izedException } from '@nestjs/common';
import { HttpService } from '@nestjs/axios';
import { JwtService } from '@nestjs/jwt';
import { Repository, In } from 'typeorm';
import { InjectRepository } from '@nestjs/typeorm';
import * as bcrypt from 'bcrypt';

import { UserInterface } from './interfaces/user.interface';
import { environment } from './environment/environment';
import { LoginDto } from './dtos/login.dto';
import { ViewMenuByUserRolesEntity } from './entities/view-
menu-by-user-roles.entity';
import { ViewPrivilegesByUserRolesEntity } from './enti-
ties/view-privileges-by-user-roles.entity';
import { UserJwtInterface } from './interfaces/user-jwt.inter-
face';

@Injectable()
export class AuthenticationService {
  constructor(
    @InjectRepository(ViewMenuByUserRolesEntity)
    private readonly viewMenusByRolesRepository: Reposi-
tory<ViewMenuByUserRolesEntity>,

    @InjectRepository(ViewPrivilegesByUserRolesEntity)
    private readonly viewPrivilegesByRolesRepository: Reposi-
tory<ViewPrivilegesByUserRolesEntity>,

    private readonly http: HttpService,
    private readonly jwtService: JwtService,
  ) {}

  async findByEmail(email: string, user?: UserInterface |
{ id: number }): Promise<UserInterface> {
    return await this.getFindByEmail(email);
  }

  async getFindByEmail(email: string): Promise<UserInterface>
{
    return new Promise((resolve, reject) => {
      this.http.post(`${environment.api}/users/findUserByE-
mail`, email).subscribe({
        next: (user) => {
          resolve(user.data);
        },
        error: (rej) => {
          reject(rej);
```

```
        },
      });
    });
  }

  async getFindByRegister({ register: string }): Prom-
ise<UserInterface> {
    return new Promise((resolve, reject) => {
      this.http.post(`${environment.api}/users/findUs-
erByLogin`, { register: string }).subscribe({
        next: (user) => {
          resolve(user.data);
        },
        error: (rej) => {
          reject(rej);
        },
      });
    });
  }

  async getFindByCPF(cpf: string): Promise<any> {
    return new Promise((resolve, reject) => {
      this.http.post(`${environment.api}/users/findUserByCpf`,
{ cpf }).subscribe({
        next: (user) => {
          resolve(user.data);
        },
        error: (rej) => {
          reject(rej);
        },
      });
    });
  }

  removeMask(cpf: string) {
    return cpf.replace(/[^\d]+/g, '');
  }

  async login({ username, password }: LoginDto) {
    try {
      const user = await this.getFindByCPF(username);
      if (!user || !(await this.checkPassword(password,
user.password))) {
        throw new UnauthorizedException('Essas credencias
estão incorretas');
      }

      delete user.password;
      const { accessToken, payload } = await this.signTo-
ken(user);
      return { user: payload, accessToken };
```

```
    } catch (error) {
      if (error instanceof HttpException) {
        throw error;
      }
      throw new HttpException({ message: `Não foi possivel re-
alizar o login. ${error}` }, HttpStatus.INTERNAL_SERVER_ER-
ROR);
    }
  }

  async signToken(user: UserInterface): Promise<{ accessToken:
string; payload: UserJwtInterface }> {
    const menus = await this.viewMenusByRolesRepository.find({
      where: { roleId: In(user.roles.map((role: any) =>
role.id)) },
    });

    const uniqueMenus = menus.filter(
      (menu, index, self) => index === self.findIndex((m) =>
m.menu === menu.menu && m.menuGroup === menu.menuGroup &&
m.route === menu.route && m.menuKey === menu.menuKey),
    );

    const privileges = await this.viewPrivilegesByRolesReposi-
tory.find({
      where: {
        roleId: In(user.roles.map((role: any) => role.id)),
      },
      select: ['key'],
    });
    const rolesId = user.roles.map((role) => role.id);
    const payload = {
      id: user.id,
      name: user.name,
      email: user.email,
      register: user.register,
      cpf: user.cpf,
      rolesId: rolesId,
      menus: uniqueMenus,
      privileges,
      createdAt: user.createdAt,
      updatedAt: user.updatedAt,
    };

    return { accessToken: this.jwtService.sign(payload), pay-
load };
  }

  async checkPassword(plainPassword: string, password:
string): Promise<boolean> {
```

```
    return (await bcrypt.compare(plainPassword, password)) as
boolean;
  }
}


```

Arquivo authentication.module.ts:

```
import { Module } from '@nestjs/common';
import { HttpModule } from '@nestjs/axios';
import { PassportModule } from '@nestjs/passport';
import { JwtModule } from '@nestjs/jwt';
import { TypeOrmModule } from '@nestjs/typeorm';
import { AuthenticationController } from './authentica-
tion.controller';
import { AuthenticationService } from './authentication.ser-
vice';
import { EnvironmentProviderModule } from './environment/envi-
ronment.provider';
import { ViewMenuByUserRolesEntity } from './entities/view-
menu-by-user-roles.entity';
import { ViewPrivilegesByUserRolesEntity } from './enti-
ties/view-privileges-by-user-roles.entity';
import { JwtStrategy } from './providers/authentica-
tion/jwt.strategy';
import { DatabaseProviderModule } from './providers/data-
base/database.provider';

@Module({
  imports: [
    HttpModule,
    DatabaseProviderModule,
    TypeOrmModule.forFeature([
      ViewMenuByUserRolesEntity,
      ViewPrivilegesByUserRolesEntity,
    ]),
    PassportModule.register({ defaultStrategy: 'jwt' }),
    JwtModule.register({
      secret: process.env.APP_SECRET,
      signOptions: {
        expiresIn: '1h',
        algorithm: 'HS384',
      },
      verifyOptions: {
        algorithms: ['HS384'],
      },
    }),
  ],
```

```
  controllers: [AuthenticationController],
  providers: [JwtStrategy, EnvironmentProviderModule, Authen-
ticationService],
  exports: [TypeOrmModule],
})
export class AuthenticationModule {}


```

Arquivo main.ts:

```
import { NestFactory } from '@nestjs/core';
import { Transport } from '@nestjs/microservices';
import { AuthenticationModule } from './authentication.mod-
ule';
import { Logger } from '@nestjs/common';

async function bootstrap() {
  const app = await NestFactory.create(AuthenticationModule);
  app.connectMicroservice({
    transport: Transport.TCP,
    options: { host: '0.0.0.0', port: 3006 },
  });
  await app.startAllMicroservices();
  app.enableCors();
  await app.listen(3000, () => {
    Logger.log(`Listening at http://localhost:3000`);
  });
}
bootstrap();


```

APÊNDICE F – Geração e utilização do arquivo JSON de features para adição ou edição da LPS

Arquivo config.service.ts (front):

```
import { BehaviorSubject } from "rxjs";
import http from "../../shared/services/axios";
import { ConfigInterface } from "./interfaces/config.interface";
class ConfigService {
  private config$ = new BehaviorSubject<ConfigInterface | null>(null);
  constructor() {}
  getConfig(): ConfigInterface | null {
    return this.config$.getValue();
  }
  async httpGet() {
    const currentConfig = this.config$.getValue();

    if (currentConfig) {
      return currentConfig;
    }
    const configData = (await http.get<ConfigInterface, any>("config/")).data as ConfigInterface;
    this.config$.next(configData);
    return configData;
  }
  async httpPut(data: ConfigInterface): Promise<{ config: ConfigInterface; message: string }> {
    const response = await http.put<ConfigInterface, any>(`config/`, { data });
    this.config$.next(response.data);
    return response.data;
  }
}
export default new ConfigService();
```

```

Arquivo index.tsx do config-form:

```
import React, { useEffect, useState } from 'react';
import { IconButton, Button, Container } from '@mui/material';
import { ArrowBack } from '@mui/icons-material';
import { useNavigate } from 'react-router-dom';
import styles from './ConfigForm.module.scss';
import configService from '../config.service';
import { ConfigInterface } from
'../interfaces/config.interface';
import authService from '../../auth/auth.service';
import ConfigCheckbox from '../../../components/config-
checkbox';
const initialConfig: ConfigInterface = {
  AuthenticationModule: {
    active: true,
    options: {
      localDB: true,
      googleAPI: false,
    },
  },
  RoleModule: {
    active: true,
    options: {
      administrador: true,
      graduacao: true,
      posGraduacao: true,
      gestor: true,
      servidor: true,
      terceirizado: true,
      visitante: true,
      residente: true,
```
```

```
    },
  },
  UserModule: { active: true, options: { admin: true, manager:
false, student: true, employee: false, visitor: false } },
  UserRoleModule: { active: true, options: {} },
  ScheduleModule: { active: true, options: { diario: true,
semanal: true } },
  ActionModule: { active: true, options: {} },
  MenuModule: { active: true, options: {} },
  MenuGroupModule: { active: true, options: {} },
  MealModule: { active: true, options: { simples: true,
multiplo: false } },
  ProfileModule: { active: true, options: {} },
  MenuMealModule: { active: true, options: {} },
  RatingModule: { active: true, options: { forum: true, form:
false } },
  SubMealsModule: { active: true, options: {} },
  MealsUserRolesModule: { active: true, options: {} },
  PaymentsModule: { active: true, options: { PIX: true,
boleto: false } },
};
const ConfigForm = () => {
  const navigate = useNavigate();
  const [systemName, setSystemName] = useState<string>('');
  const [features, setFeatures] = useState<ConfigInterface>();
  const handleCheckboxChange = (moduleKey: string, optionKey?:
string) => {
    setFeatures((prevConfig) => {
      const newConfig = { ...prevConfig };
      if (optionKey) {
        moduleKey = moduleKey.concat('Module');
        if (newConfig[moduleKey].options) {
          newConfig[moduleKey].options[optionKey]
= !newConfig[moduleKey].options[optionKey];
        }
      } else {
```

```
        moduleKey = moduleKey.concat('Module');
        newConfig[moduleKey].active
= !newConfig[moduleKey].active;
      }
      return newConfig;
    });
  };
  useEffect(() => {
    const loadConfig = async () => {
      try {
        const config = await configService.httpGet();
        console.log(config);
        setFeatures(config);
      } catch (error) {
        console.error('Erro ao carregar configurações:',
error);
      }
    };
    loadConfig();
  }, []);

  const handleSystemNameChange = (event:
React.ChangeEvent<HTMLInputElement>) => {
    setSystemName(event.target.value);
  };

  const handleSubmit = (event:
React.FormEvent<HTMLFormElement>) => {
    event.preventDefault();
    if (features) {
      configService.httpPut(features).then((res) => {
        if (res) {
          authService.logout();
          navigate('/auth/login');
          window.location.reload();
```

```
          return;
        }
      });
    }
  };

  if (!features) {
    return <div>Carregando...</div>;
  }

  return (
    <>
      <div className={styles.title}>
        <IconButton size="small" color="primary" onClick={() =>
navigate(-1)}>
          <ArrowBack />
        </IconButton>
        <h1>Configuração Geral do Sistema</h1>
      </div>
      <form onSubmit={handleSubmit}>
        {/* <TextField label="Nome do Sistema"
variant="outlined" fullWidth value={systemName}
onChange={handleSystemNameChange} margin="normal" /> */}
        <h3>Selecionar Features</h3>
        <Container>
          {Object.keys(features).map((module) => {
            const moduleKey = module.replace('Module', '');
            const moduleData = features[module as keyof
ConfigInterface];

            if (!moduleData.options ||
Object.keys(moduleData.options).length === 0) {
              return null;
            }

            return <ConfigCheckbox key={module}
```

```
label={moduleKey} active={moduleData.active}
options={moduleData.options} onChange={(optionKey) =>
handleCheckboxChange(moduleKey, optionKey)} />;

          })}

        </Container>


        <Button type="submit" variant="contained"
color="primary" fullWidth>

          Salvar Configurações

        </Button>

      </form>

    </>

  );

};


export default ConfigForm;
```

Na rota de config:

```
getConfig() {

    try {

      const config = fs.readFileSync(this.configFilePath,
'utf-8');

      return JSON.parse(config);

    } catch (error) {

      console.error('Erro ao ler o arquivo de configuração:',
error);

      throw error;

    }

  }


 updateConfig(newConfig: any) {

    try {

      fs.writeFileSync(this.configFilePath,
JSON.stringify(newConfig, null, 2));
```

```
      console.log('Arquivo de configuração atualizado com
sucesso.');
      this.restartApplication();
    } catch (error) {
      console.error('Erro ao atualizar o arquivo de
configuração:', error);
      throw error;
    }
  }
```

APÊNDICE G – Código do *wizard guide* (guia de instalação) para gerar uma nova aplicação

Arquivos de interfaces:

```
export interface ConfigInterfaceInit {
  name: string;
  database: DatabaseConfigInterface;
  selectedModules: SelectedModules;
}

export interface SelectedModules {
  [moduleName: string]: ModuleConfig;
}

export interface ModuleConfig {
  active: boolean;
  name: string;
  options: {
    [optionName: string]: boolean;
  };
}

export type DatabaseType = "mysql" | "postgres" | "mssql" |
"sqlite";

export interface DatabaseConfigInterface {
  name: string;
  type: DatabaseType;
  port: number;
  username: string;
  password: string;
}
```

Arquivo wizard.service.ts:

```
import http from '../../shared/services/axios';
import { ConfigInterfaceInit } from './interfaces/config.inter-
face';

class WizardService {
  constructor() {}

  async httpPost(data: ConfigInterfaceInit): Promise<{ message:
string }> {
      const response = await http.post<ConfigInterfaceInit,
any>('config/generate', { data });
```

```
    return response.data;
  }
}

export default new WizardService();
```

Arquivo index.tsx:

```
import React, { useState } from 'react';
import {
  Stepper,Step,StepLabel,Button,Typography,TextField,Checkbox,
  FormGroup,FormControlLabel,Box,Container,Grid,CircularProgess,
  FormControl,InputLabel,Select,MenuItem,SelectChangeEvent,
  Card,CardContent,Divider,
} from '@mui/material';
import wizardService from '../wizard.service';
import { ConfigInterfaceInit, DatabaseType } from '../inter-
faces/config.interface';

interface ConfigInterface {
  [key: string]: {
    active: boolean;
    name: string;
    options: { [key: string]: boolean };
  };
}

const initialConfig: ConfigInterface = {
  AuthenticationModule: {
    active: true,
    name: 'authentication',
    options: {
      localDB: true,
      googleAPI: false,
    },
  },
  RoleModule: {
    active: true,
    name: 'roles',
    options: {
      administrador: true,
      graduacao: true,
      posGraduacao: true,
      gestor: true,
      servidor: true,
      terceirizado: true,
      visitante: true,
      residente: true,
    },
  },
```

```
  UserModule: { active: true, name: 'users', options: { admin-
istrador: true, gestor: true } },
   ScheduleModule: { active: true, name: 'schedules', options:
{ diario: true, semanal: true } },
  MealModule: { active: true, name: 'meals', options: { simples:
true, multiplo: false } },
   PaymentsModule: { active: true, name: 'payments', options:
{ PIX: true, boleto: false } },
  RatingsModule: { active: true, name: 'ratings', options: { form:
true, forum: false } },
   MetricsModule: { active: true, name: 'metrics', options:
{ cards: true, chart: true } },
};

const steps = ['Informações do Projeto', 'Seleção de Features'];

function Wizard() {
  const [activeStep, setActiveStep] = useState(0);
  const [projectName, setProjectName] = useState('');
  const [databaseName, setDatabaseName] = useState('');
  const [selectedDatabase, setSelectedDatabase] = useState<Databa-
seType>('mysql');
  const [port, setPort] = useState('');
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');

  const [modules, setModules] = useState<ConfigInterface>(initi-
alConfig);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState('');

  const handleSelectChange = (event: SelectChangeEvent<Database-
Type>) => {
    setSelectedDatabase(event.target.value as DatabaseType);
  };

  const handleNext = () => {
    setActiveStep((prevActiveStep) => prevActiveStep + 1);
  };

  const handleBack = () => {
    setActiveStep((prevActiveStep) => prevActiveStep - 1);
  };

  const handleCheckboxChange = (moduleKey: string, optionKey:
string) => {
    setModules((prevModules) => {
      const updatedModule = {
        ...prevModules[moduleKey],
        options: {
          ...prevModules[moduleKey].options,
```

```
              [optionKey]:  !prevModules[moduleKey].options[op-
tionKey],
        },
      };

       const allOptionsFalse = Object.values(updatedModule.op-
tions).every((option) => option === false);
      updatedModule.active = !allOptionsFalse;

      return {
        ...prevModules,
        [moduleKey]: updatedModule,
      };
    });
  };

  const handleSubmit = async () => {
    setLoading(true);
    setError('');

    const data: ConfigInterfaceInit = {
      name: projectName,
       database: { name: databaseName, type: selectedDatabase,
port: +port, username: username, password: password },
      selectedModules: modules,
    };
    console.log(data);

    try {
      const response = await wizardService.httpPost(data);
      if (response) {
        console.log(response);
      } else {
        throw new Error('Erro ao criar o projeto.');
      }
    } catch (error) {
      console.error('Erro: ', error);
      setError('Erro ao criar o projeto.');
    } finally {
      setLoading(false);
    }
  };

  const renderStepContent = (step: number) => {
    switch (step) {
      case 0:
        return (
          <div>
           <TextField label="Nome do Projeto" value={projectName}
onChange={(e) => setProjectName(e.target.value)} fullWidth mar-
gin="normal" />
```

```
            <TextField label="Nome do Banco de Dados" value={da-
tabaseName} onChange={(e) => setDatabaseName(e.target.value)}
fullWidth margin="normal" />

            <FormControl fullWidth margin="normal">
               <InputLabel id="select-database-label">Selecione
o banco de dados</InputLabel>
              <Select labelId="select-database-label" value={se-
lectedDatabase} onChange={handleSelectChange}>
                <MenuItem value="mysql">MySQL</MenuItem>
               <MenuItem value="postgres">PostgreSQL</MenuItem>
                <MenuItem value="mssql">SQL Server</MenuItem>
                <MenuItem value="sqlite">SQLite</MenuItem>
              </Select>
            </FormControl>

            <TextField label="Porta" value={port} onChange={(e)
=> setPort(e.target.value)} fullWidth margin="normal" />
              <TextField label="Usuário" value={username} on-
Change={(e) => setUsername(e.target.value)} fullWidth mar-
gin="normal" />
           <TextField label="Senha" type="password" value={pass-
word} onChange={(e) => setPassword(e.target.value)} fullWidth
margin="normal" />
          </div>
        );
      case 1:
        return (
          <Container>
            {Object.keys(modules).map((module) => {
              const moduleKey = module.replace('Module', '');
               const moduleData = modules[module as keyof Con-
figInterface];

                if (!moduleData.options || Object.keys(mod-
uleData.options).length === 0) {
                return null;
              }

              return (
                <div key={module}>
                  <FormGroup key={module}>
                      <Typography variant="h6">{moduleKey}</Ty-
pography>
                    <Grid container spacing={2}>
                      {Object.keys(moduleData.options).map((op-
tionKey) => (
                        <Grid item xs={6} sm={4} md={3} key={op-
tionKey}>
                            <FormControlLabel control={<Checkbox
checked={moduleData.options[optionKey]}     onChange={()     =>
```

```
handleCheckboxChange(module, optionKey)} />} label={optionKey}
/>
                        </Grid>
                      ))}
                    </Grid>
                  </FormGroup>
                 <Divider style={{ backgroundColor: '#f3f3f3' }}
/>
                </div>
              );
            })}
          </Container>
        );
      default:
        return 'Unknown step';
    }
  };

  return (
    <Box display="flex" flexDirection="column" alignItems="cen-
ter" justifyContent="center" minHeight="100vh">
      <Card style={{ margin: '30px' }}>
        <CardContent>
          <Stepper activeStep={activeStep} alternativeLabel>
            {steps.map((label, index) => (
              <Step key={index}>
                <StepLabel>{label}</StepLabel>
              </Step>
            ))}
          </Stepper>
          <Box mt={4}>
            {activeStep === steps.length ? (
              <div>
                {loading ? (
                   <CircularProgress size={60} style={{ margin:
'20px', color: 'blue' }} />
                ) : (
                  <>
                    {error ? (
                      <Typography color="error">{error}</Typog-
raphy>
                    ) : (
                      <>
                        <Typography variant="h6" gutterBottom>
                          Projeto Criado com Sucesso!
                        </Typography>
                           <Button onClick={handleSubmit}>Fina-
lizar</Button>
                      </>
                    )}
                  </>
```

```
                )}
              </div>
            ) : (
              <div>
                {renderStepContent(activeStep)}
                 <Box mt={2} style={{ display: 'flex', justify-
Content: 'space-between' }}>
                      <Button disabled={activeStep === 0} on-
Click={handleBack}>
                    Voltar
                  </Button>
                    <Button variant="contained" color="primary"
onClick={activeStep === steps.length - 1 ? handleSubmit :
handleNext}>
                  {activeStep === steps.length - 1 ? 'Concluir' :
'Próximo'}
                  </Button>
                </Box>
              </div>
            )}
          </Box>
        </CardContent>
      </Card>
    </Box>
  );
}
export default Wizard;
```