**PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS**

**PUC Minas Virtual**

**Pós-graduação *Lato Sensu* em Arquitetura de *Software* Distribuído**

Projeto Integrado

Relatório Técnico

Saborito - Plataforma de Vendas para Cantinas Escolares

Adriano dos Santos Laureano

Belo Horizonte

Abril de 2024.

# Projeto Integrado – Arquitetura de Software Distribuído

**Sumário**

[Projeto Integrado – Arquitetura de Software Distribuído 2](#_Toc116844011)

[1. Introdução 3](#_Toc116844012)

[2. Especificação Arquitetural da solução 5](#_Toc116844013)

[2.1 Restrições Arquiteturais 5](#_Toc116844014)

[2.2 Requisitos Funcionais 5](#_Toc116844015)

[2.3 Requisitos Não-funcionais 6](#_Toc116844016)

[2.4 Mecanismos Arquiteturais 6](#_Toc116844017)

[3. Modelagem Arquitetural 7](#_Toc116844018)

[3.1 Diagrama de Contexto 7](#_Toc116844019)

[3.2 Diagrama de Container 8](#_Toc116844020)

[3.3 Diagrama de Componentes 8](#_Toc116844021)

[4. Avaliação da Arquitetura (ATAM) 9](#_Toc116844022)

[4.1 Análise das abordagens arquiteturais 10](#_Toc116844023)

[4.2 Cenários 10](#_Toc116844024)

[4.3 Evidências da Avaliação 11](#_Toc116844025)

[5. Avaliação Crítica dos Resultados 12](#_Toc116844026)

[6. Conclusão 13](#_Toc116844027)

[Referências 14](#_Toc116844028)

## Introdução

Nos últimos anos, tem havido um crescente interesse em soluções tecnológicas que melhorem a eficiência e a conveniência em vários aspectos da vida cotidiana. Uma área que vem experimentando mudanças significativas é a forma como as transações comerciais são realizadas, especialmente em ambientes escolares. Com o objetivo de otimizar o processo de compra e pagamento de lanches em cantinas escolares, foi desenvolvida a plataforma Saborito.

A plataforma Saborito surge como resposta às demandas por maior praticidade e controle nas transações realizadas em cantinas escolares. Seu desenvolvimento foi motivado pela necessidade de modernizar e simplificar o processo de compra de alimentos nas escolas, proporcionando uma experiência mais eficiente tanto para os alunos quanto para os proprietários das cantinas.

Os objetivos específicos propostos são:

* Definir a experiência do usuário na plataforma.
* Documentar um projeto arquitetural que converse com o negócio proposto.
* Desenvolver uma aplicação de acordo com os mecanismos e requisitos definidos.

## Especificação Arquitetural da solução

Esta seção apresenta a especificação básica da arquitetura da solução a ser desenvolvida, incluindo diagramas, restrições e requisitos definidos pelo autor, tal que permitam visualizar a macro arquitetura da solução.

## Restrições Arquiteturais

Requisitos Arquiteturais são todos que, funcionais ou não funcionais impactam de forma direta na arquitetura do sistema. A seguir estão listados os requisitos de arquitetura identificados para a implementação desta solução.

|  |
| --- |
| R1: O software deve ser desenvolvido utilizando tecnologias de código aberto ou de licença de comercialização gratuitas.  R2: Deve ser codificado em tecnologia que permita execução em ambiente de servidor Linux.  R3: Deve utilizar autenticação por token JWT, não se limitando mas podendo utilizar protocolos comuns como OpenID Connect e OAuth 2.0. |
| RA4: Deve se utilizar um banco de dados relacional do tipo SQL como PostgreSQL ou outro similar. |
| RA5: Deve ser utilizado no frontend tecnologia compatível com os padrões web 2.0 |
| RA6: Deve ser utilizado api de terceiro, Stripe, para fornecer o processamento de pagamentos, transferências e saques na plataforma. |
| RA7: Deve ser desenvolvida api própria ou utilizar apis de terceiros, Notazz, Asaas, para fornecer a interface de emissão de notas fiscais, para o devido recolhimento de tributo das recargas na plataforma. |
|  |
|  |

## Requisitos Funcionais

Enumere todos os requisitos funcionais previstos para a sua aplicação. Lembre-se de listar todos os requisitos necessários para garantir cobertura arquitetural, segundo a especificação fornecida. Esta seção deve conter uma lista de requisitos visando produzir a **modelagem completa** da solução.

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Descrição Resumida** | **Dificuldade (B/M/A)\*** | **Prioridade**  **(B/M/A)\*** |
| RF01 | Permitir o registro de novos usuários (aluno, responsável e cantina) | M | A |
| RF02 | Permitir o login de usuários (aluno, responsável e cantina) |  |  |
| RF03 | Permitir que usuários cadastrem e gerenciem produtos(cantina) |  |  |
| RF04 | Processar pagamentos de recargas pelas formas de pagamento aceitas pelo gateway de pagamentos implementado (Cartão de crédito) |  |  |
| RF05 | Exibir resumo de informações da conta do usuário (aluno, responsável e cantina) |  |  |
| RF06 | Exibir histórico de vendas do usuário (cantina) |  |  |
| RF07 | Exibir histórico de compras do usuário (aluno, responsável) |  |  |
| RF08 | Permitir saques para conta corrente dos valores das vendas feitas pelo usuário (cantina) |  |  |
| RF09 | Exibir histórico de saques do usuário (cantina) |  |  |
| RF010 | Integrar com plataforma de terceiro para emitir notas fiscais sobre as vendas dos usuários vendedores |  |  |
| RF011 | Enviar telemetria de dados sensoriais e de análise para plataforma New Relic ou Grafana |  |  |

\*B=Baixa, M=Média, A=Alta.

**Obs:** acrescente quantas linhas forem necessárias.

## Requisitos Não-funcionais

Enumere todos os requisitos não-funcionais previstos para a sua aplicação. Entre os requisitos não funcionais inclua todos os requisitos que julgar importantes do ponto de vista arquitetural, ou seja, os requisitos que terão impacto na definição da arquitetura. Os requisitos devem ser descritos de forma completa e preferencialmente quantitativa (ex: tempo de resposta de “x” segundos).

|  |  |  |
| --- | --- | --- |
| **ID** | **Descrição** | **Prioridade**  **B/M/A** |
| RNF01 | O sistema deve ser apresentar disponibilidade 24 X 7 X 365 | A |
| RNF02 | O sistema deve estar disponível para os navegadores de internet mais atuais do mercado. |  |
| ... | O sistema deve exigir autenticação do usuário através de token jwt através do protocolo OpenID Connect. |  |
| ... | O sistema deve ser capaz de se integrar a api de sistema terceiro (gateway de pagamentos) para processar os pagamentos. |  |
|  | O layout deve ser responsivo para as telas de uso comum (desktop, notebooks, tablets e smartphones) |  |
|  |  |  |

**Obs**: acrescente quantas linhas forem necessárias.

## Mecanismos Arquiteturais

Esta seção deve apresentar uma visão geral dos mecanismos que compõem a arquitetura do software, baseando-se em três estados: (1) análise, (2) design e (3) implementação. Em termos de Análise devem ser listados os aspectos gerais que compõem a arquitetura do software, como: persistência, integração com sistemas legados, geração de logs do sistema, ambiente de front end, tratamento de exceções, formato dos testes, formato de distribuição/implantação (deploy), dentre outros. Em Design deve-se identificar o padrão tecnológico a seguir para cada mecanismo identificado na análise. Em Implementação deve-se identificar o produto a ser utilizado na solução, caso ela fosse implementada.

|  |  |  |
| --- | --- | --- |
| **Análise** | ***Design*** | **Implementação** |
| Persistência | ORM | Prisma |
| Persistência | Banco de Dados | PostgreSQL |
| Front end | Single Page Application | Angular |
| Back end | API Rest | Node.js com Typescript |
| Integração | Fila de tarefas assíncronas | Rabbitmq |
| Autenticação | Protocolo OpenID Connect | Node.js com Typescript |
| Teste de Software | Testes Unitários | Jest |
| Deploy | Pipeline CI/CD | Github Actions |
| Notificações | Envio de E-mail | Protocolo SMTP |

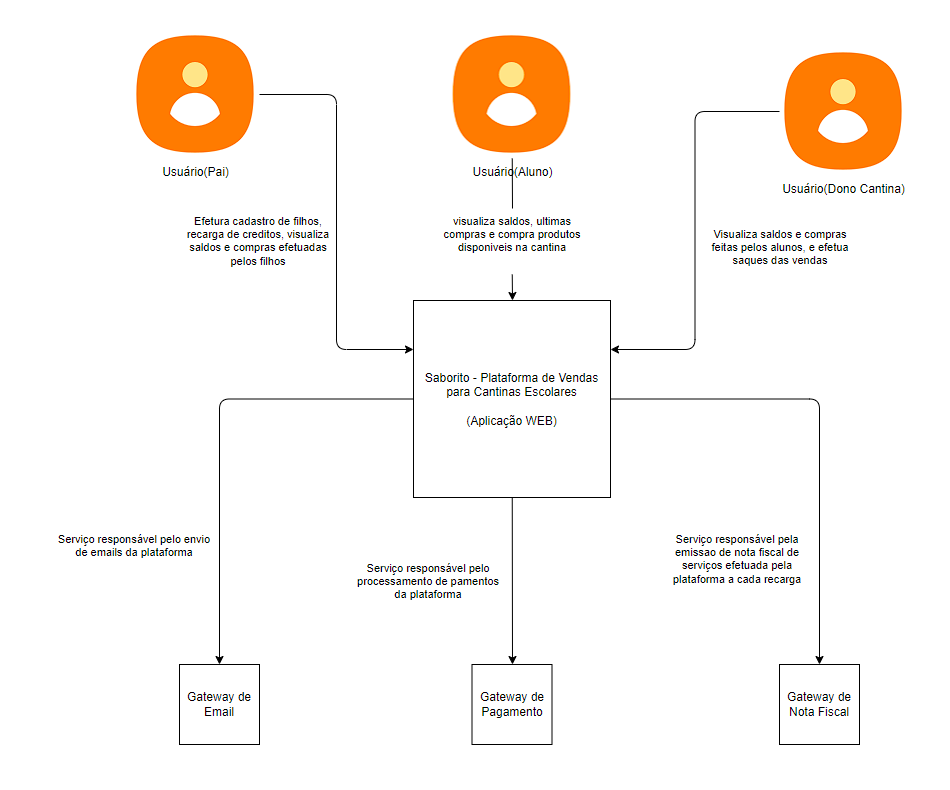
## Modelagem Arquitetural

Coloque um texto explcativo aqui, tal como: “Esta seção apresenta a modelagem arquitetural da solução proposta, de forma a permitir seu completo entendimento visando à futura implementação” (que não está incluída no escopo deste trabalho).

Para esta modelagem arquitetural optou-se por utilizar o modelo C4 para documentação de arquitetura de software. Mais informações a respeito podem ser encontradas aqui: <https://c4model.com/> e aqui: <https://www.infoq.com/br/articles/C4-architecture-model/>. Dos quatro nível que compõem o modelo C4 três serão apresentados aqui e somente o Código será apresentado na próxima seção (5).

## 3.1 Diagrama de Contexto

Apresente um diagrama que mostre a visão geral da solução proposta e, em seguida, explique-o brevemente, de forma textual. Este diagrama não precisa seguir os padrões da UML, e deve ser completo e tão simples quanto possível, apresentando a **macroarquitetura** da solução, como no exemplo:



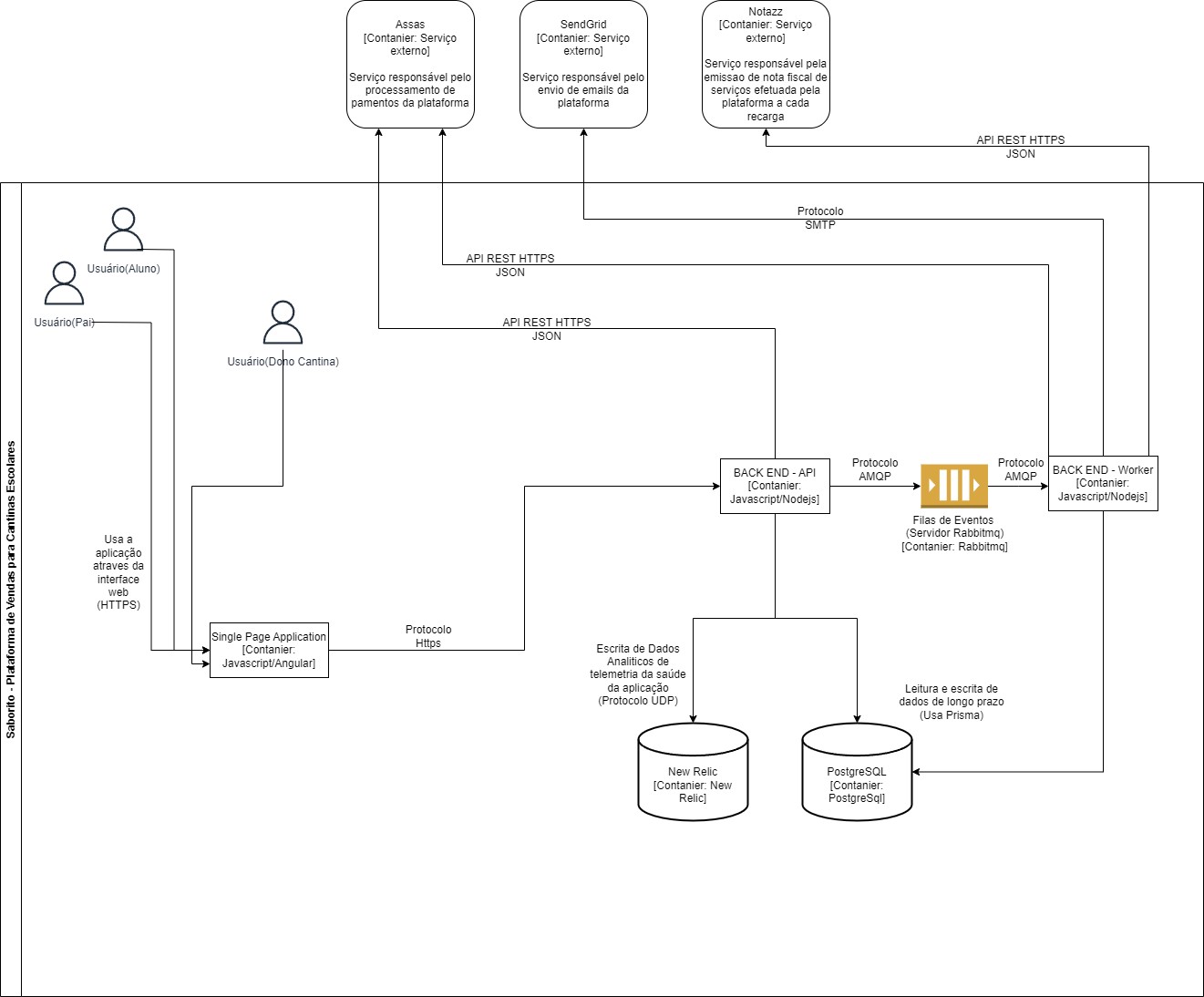
**Figura 1 - Visão Geral da Solução. Fonte:** <https://www.infoq.com/br/articles/C4-architecture-model/>

A figura 1 mostra a especificação o diagrama geral da solução proposta, com todos seus principais módulos e suas interfaces.

O ponto de entrada de qualquer forma de usuário (aluno, responsável e cantina) será a Aplicação Web, que conversa com serviços externo para processa pagamentos, enviar e-mails e emitir notas fiscais.

## 3.2 Diagrama de Container

O Diagrama de Container da aplicação, indica como os componentes (aplicativos, armazenamentos de dados, microservices, etc.) que compõem esse sistema de software estão distribuídos e organizados.



**Figura 2 – Diagrama de container. Fonte:** <https://www.infoq.com/br/articles/C4-architecture-model/>

A figura 2 apresenta os containers da aplicação e a forma como estão organizados, a arquitetura visa manter somente uma fonte de verdade tratando-se dos dados da aplicação como um todo.

A aplicação é constituída por 2 bancos de dados, sendo o PostgreSql para dados persistentes de longo prazo, entendendo-se por isso os dados do usuário, produtos, vendas e outros dados semelhantes pela facilidade para trabalhar com esquemas relacionais, também tendo uma integração excelente com a plataforma NodeJS. O segundo banco de dados que será utilizado para telemetria, é oferecido pelo serviço New Relic. Este pool de dados deverá armazenar quaisquer dados enviados pela aplicação para finalidade de telemetria, como quantidade de execuções de determinada tarefa, quantidade de erros emitidos e informações semelhantes.

A aplicação que segue o modelo de monolito, é organizada em um único repositório e tem 2 pontos de entrada (entry points). Um para o servidor web, permitindo fácil escalabilidade horizontal e respectivamente designados no diagrama como "Backend - API" e outro para execução de serviços distribuídos em rede de forma assíncrona designado no diagrama como " Backend -Worker".

Uma Single Page Application fará a interface com o usuário para os serviços backend em nuvem, sendo o ponto de entrada de todos os usuários da plataforma, sejam eles vendedores ou clientes. A single page application se conecta exclusivamente com o Backend através de protocolo HTTPS no padrão de API Rest utilizando JSON.

As rotas (urls) de HTTPS da API Backend que necessitam de autenticação estão protegidas por token JWT através do protocolo OpenID. Ele se conecta para troca de tokens e validações direto com o Backend protegido por criptografía SSL.

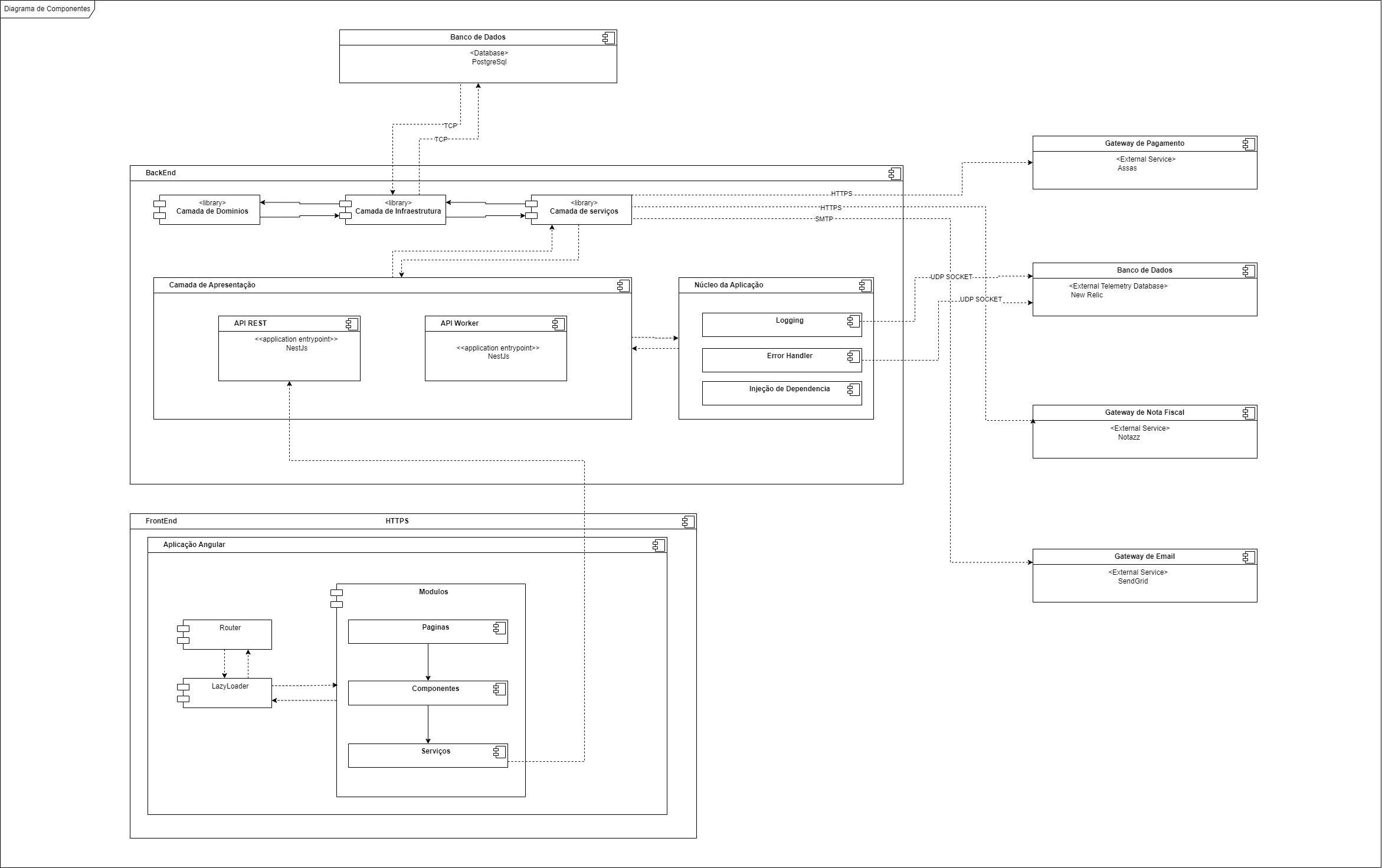
O Gateway de pagamentos da empresa chamada Stripe que será responsável pela comunicação com os bancos e bandeiras de pagamento e se conecta com o Backend REST da aplicação através de API Rest utilizando JSON fornecida por eles e documentada no seguinte site web https://docs.stripe.com.

O Serviço de envio de e-mails denominado Sendgrid processa o envio de notificações para os clientes da plataforma através de protocolo SMTP que pode ser enviado através da API da aplicação ou através de tarefas assíncronas do Worker da aplicação.

Por fim, o serviço de emissão de notas fiscais, Notazz, que receberá exclusivamente através do Worker assíncrono chamadas HTTPS por meio de API Rest fornecida pela plataforma.

## 3.3 Diagrama de Componentes

Apresente o Diagrama de Componentes da aplicação (baseado na UML), indicando os elementos da arquitetura e as interfaces entre eles. Liste os estilos/padrões arquiteturais utilizados e faça uma descrição sucinta dos componentes indicando o papel de cada um deles dentro da arquitetura/estilo/padrão arquitetural. Indique também quais componentes serão reutilizados (navegadores, SGBDs, middlewares, etc), quais componentes serão adquiridos por serem proprietários e quais componentes precisam ser desenvolvidos.



**Figura 3 – Diagrama de Componentes.**

A figura 3 apresenta o diagrama de componentes que representa a arquitetura da aplicação. Os componentes descritos no diagrama podem ser detalhados como:

CAMADA DE DOMÍNIOS - Camada é responsável por definir a lógica de modelagem dos objetos e entidades que representam o domínio da aplicação.

CAMADA DE SERVIÇOS - Camada responsável por implementar a lógica de negócio da aplicação, fornecendo serviços que podem ser utilizados pela Camada de Apresentação e pela Camada de Infraestrutura. Esses serviços podem incluir validação de dados, autenticação de usuários, integração com sistemas externos e outras funcionalidades que não estão relacionadas diretamente com a interação do usuário com a aplicação.

● CAMADA DE INFRAESTRUTURA - A camada de Infraestrutura é responsável por prover serviços e recursos necessários às conexões com bancos de dados.

● API REST - Ponto de entrada da aplicação backend que escuta conexões na porta 80 e 443 (HTTP e HTTPS) desenhada com padrão REST utilizando a biblioteca EXPRESSJS. Utilizada para gerenciar rotas HTTP na camada de apresentação onde o FRONTEND se utilizará para estabelecer a comunicação.

● SERVICE WORKER - Outro ponto de entrada da aplicação backend que executa tarefas em segundo plano de forma assíncrona utilizando padrão computacional de fila através da biblioteca BULLMQ.

● GATEWAY DE PAGAMENTO/STRIPE - Serviço externo acessado pela aplicação utilizado para processamento de pagamentos.

●GATEWAY DE NOTAS FISCAIS (NOTAZZ) - Serviço externo acessado pela aplicação e utilizado para processamento de notas fiscais.

● GATEWAY DE AUTENTICAÇÃO (KEY CLOAK) - Serviço externo responsável pela autenticação e autorização dos usuários na plataforma

● GATEWAY DE NOTIFICAÇÕES/SENDGRID - Serviço responsável pela entrega de notificações por e-mail para a plataforma através do protocolo SMTP.

● MONGODB - Banco de dados "NO-SQL" utilizado para armazenamento dos dados persistentes da aplicação, como dados do usuário, compra, produtos e pagamentos.

● REDIS - Banco de dados de cache em memória, capaz de executar operações de leitura e escrita com baixa latência, se comunica diretamente com o componente "Camada de Infraestrutura" da aplicação backend conforme descreve DIP - Dependency Inversion Principle. E é utilizado para operações de QUEUE e DEQUEUE no padrão de fila gerenciado pela biblioteca BullMQ.

● NÚCLEO DA APLICAÇÃO - Módulo abstrato da aplicação que inclui os componentes básicos de LOGGING, ERROR HANDLER e DEPENDENCY INJECTION necessários às respectivas funções dentro da aplicação.

● NEW RELIC - Banco de Dados de telemetria responsável por coletar e armazenar dados para posterior análise da saúde dos processos da aplicação.

● APLICAÇÃO ANGULAR - Aplicação frontend web responsável pela renderização das páginas web.

● ROUTER - Módulo responsável por organizar e designar as páginas apresentadas na tela do usuário através do navegador.

● LAZY LOADER - Módulo responsável por fazer o carregamento do bundle Javascript gerado pela aplicação react sob demanda evitando gargalos de rede para o usuário.

● GLOBAL STATE - Módulo responsável pelo gerenciamento do estado global da aplicação determinando o uso do padrão de "Gerenciamento de Estado Centralizado".

● COMPONENTES GLOBAIS - Componentes React de uso compartilhado entre diversas páginas da aplicação web, gerando reaproveitamento de código.

● TEMPLATES - Determinam o padrão a ser seguido por determinadas páginas.

● PÁGINAS - Determina o layout e lógica de uma determinada rota na aplicação web.

## Avaliação da Arquitetura (ATAM)

A avaliação da arquitetura desenvolvida neste trabalho é abordada nesta seção visando avaliar se ela atende ao que foi solicitado pelo cliente, segundo o método ATAM.

## Análise das abordagens arquiteturais

Apresente aqui um breve resumo das principais características da proposta arquitetural. Para isto, utilize o método Architecture Tradeoff Analysis Method (ATAM), no qual são utilizados cenários para fazer essa análise.

Exemplo:

|  |  |  |  |
| --- | --- | --- | --- |
| **Atributos de Quailidade** | **Cenários** | **Importância** | **Complexidade** |
| Interoperabilidade | Cenário 1: O sistema deve se comunicar com sistemas de outras tecnologias. | A | M |
| Usabilidade | Cenário 2: O sistema deve prover boa usabilidade. | M | B |
| Manutenibilidade | Cenário 3: O sistema deve ter a manutenção facilitada. | M | M |
| Segurança | Cenário 4: O sistema deve ser seguro e proteger os dados sensíveis | A | A |
| Escalabilidade | Cenário 5: O sistema deve suportar um grande volume de dados e usuários | A | A |
| Disponibilidade | Cenário 6: O sistema deve estar sempre disponível. | A | A |
| Performance | Cenário 7: O sistema deve ser rápido e responsivo em todas as suas funcionalidades | A | A |
| Monitoração | Cenário 8: O sistema deve ser monitorável através de telemetria para que seja possível acompanhar a saúde dos serviços e processos executados | A | M |

## Nota: Para o escopo, as características de escalabilidade, disponibilidade e performance são de alta importância e complexidade, pois a aplicação deve suportar um grande volume de usuários e transações, estar sempre disponível e ter boa performance. A segurança também é de alta importância e complexidade, pois a aplicação deve proteger dados sensíveis dos usuários. A interoperabilidade, usabilidade e manutenibilidade são importantes para garantir que o sistema possa se comunicar com outras tecnologias, oferecer uma boa experiência ao usuário e ser facilmente mantido. A testabilidade é um fator menos relevante em relação às demais, mas ainda é considerada de média importância para garantir a qualidade do sistema.

## 4.2 Cenários

Cenário 1: Interoperabilidade - O sistema deve ser capaz de se comunicar com sistemas externos que utilizam diferentes tecnologias, como REST, SOAP, MQTT, entre outros. O sistema deve ser capaz de converter os dados entre diferentes formatos para garantir a interoperabilidade.

Cenário 2: Usabilidade - O sistema deve ser projetado com uma interface intuitiva, fácil de usar e responsiva. Os usuários devem ser capazes de navegar pelo sistema sem dificuldade e executar as tarefas desejadas de forma rápida e eficiente. O sistema também deve fornecer feedbacks claros e precisos sobre as ações realizadas pelo usuário.

Cenário 3: Manutenibilidade - O sistema deve ser projetado com a manutenção em mente. O código deve ser modular, bem estruturado e fácil de entender. Os desenvolvedores devem ser capazes de fazer alterações no sistema de forma fácil e rápida, sem afetar outras partes do sistema. O sistema também deve ser projetado para facilitar a realização de testes, além de possuir um pipeline CI/CD e código ambiente de desenvolvimento em containers.

Cenário 4: Segurança - O sistema deve ser projetado para proteger os dados sensíveis. O acesso aos dados deve ser restrito apenas aos usuários autorizados e o sistema deve ser capaz de detectar e responder a ameaças de segurança em tempo real. Os dados devem ser criptografados durante o tráfego e em repouso.

Cenário 5: Escalabilidade - O sistema deve ser capaz de lidar com um grande volume de dados e usuários sem comprometer a performance. O sistema deve ser projetado para escalar horizontalmente, adicionando mais servidores conforme necessário. A arquitetura deve ser capaz de distribuir a carga de trabalho de forma eficiente entre os servidores.

Cenário 6: Disponibilidade - O sistema deve estar sempre disponível. O sistema deve ser capaz de lidar com falhas de hardware e software sem interromper o serviço para os usuários. O sistema deve ser capaz de se recuperar automaticamente em caso de falhas e garantir que o serviço esteja sempre disponível.

Cenário 7: Performance - O sistema deve ser rápido e responsivo em todas as suas funcionalidades. As consultas e transações devem ser realizadas em tempo hábil, sem atrasos perceptíveis para os usuários. O sistema deve ser projetado para otimizar o uso dos recursos do sistema e garantir a performance em situações de pico.

Cenário 8: Monitoração - O sistema deve ser monitorado através de dados fornecidos por meio de telemetria a fim de mensurar a saúde dos processos e funções executadas pelo software para rápida identificação de defeitos ou falhas de software.

## 4.3 Evidências da Avaliação

Apresente as medidas registradas na coleta de dados. Para o que não for possível quantificar apresente uma justificativa baseada em evidências qualitativas que suportem o atendimento ao requisito não-funcional.

|  |  |
| --- | --- |
| Atributo de Qualidade: | Interoperabilidade |
| Requisito de Qualidade: | O sistema deve se comunicar com outras tecnologias. |
| Preocupação: | |
| O sistema deve ter como resposta a uma requisição uma saída de fácil leitura por outro componente. | |
| Cenário(s): | |
| Cenário 1 | |
| Ambiente: | |
| Sistema em operação normal | |
| Estímulo: | |
| O sistema de monitoramento envia uma requisição para o serviço REST do módulo de informações gerenciais. | |
| Mecanismo: | |
| Criar um serviço REST para atender às requisições do sistema de monitoramento | |
| Medida de resposta: | |
| Retornar os dados requisitados no formato JSON | |
| Considerações sobre a arquitetura: | |
| Riscos: | Alguma instabilidade na rede pode deixar a conexão lenta ou mesmo a perda de pacotes. |
| Pontos de Sensibilidade: | Não há |
| Tradeoff: | Não há |

Acrescente imagens e descreva os testes realizados, de tal forma que se comprove a realização da avaliação.

Faça isto para todos os cenários apresentados no tópico 6.1.

## Avaliação Crítica dos Resultados

Apresente aqui, de forma resumida, os principais pontos positivos e negativos da arquitetura proposta. Adote uma postura crítica que permita entender as limitações arquiturais, incluindo os prós e contras das tecnologias. Você pode utilizar o formato textual ou produzir um quadro resumo.

Ex. de quadro resumo:

|  |  |
| --- | --- |
| **Ponto avaliado** | **Descrição** |
| Escalabilidade | A arquitetura proposta permite a escalabilidade horizontal e vertical, o que possibilita o aumento de desempenho e processamento de acordo com a demanda do sistema. |
| Modularidade | A arquitetura foi projetada para ser modular, facilitando a manutenção, atualização e a inclusão de novas funcionalidades. |
| Segurança | A utilização de tokens JWT com o protocolo OpenID Connect para autenticação de usuários e a integração com uma API de terceiros para processamento de pagamentos e outras operações financeiras aumentam a segurança do sistema. |
| Complexidade | A adoção de diversas tecnologias diferentes, como Node.js, Angular, PostgreSQL e AWS, Rabbitmq, pode tornar a arquitetura mais complexa e exigir um maior conhecimento técnico da equipe de desenvolvimento. |
| Dependência de API de terceiros | A integração com uma API de terceiros para processamento de pagamentos e outras operações financeiras pode tornar o sistema dependente da disponibilidade e desempenho dessa API. |
| Desempenho | A utilização do PostgreSQL como banco de dados apresenta um bom desempenho em situações de alta demanda, como em sistemas com grande volume de dados. |
| Testabilidade | Com o tamanho do escopo de projeto e a quantidade de componentes de código necessária, a testabilidade automatizada por desenvolvimento de testes unitários torna-se uma tarefa extensa que pode elevar o tempo de desenvolvimento do projeto |
|  |  |

## Conclusão

O sistema desenvolvido neste trabalho demonstrou um desempenho satisfatório no gerenciamento eficiente das transações e operações em cantinas escolares. Utilizando uma arquitetura baseada em APIs Rest seguindo os padrões web, e incorporando características assíncronas, a plataforma foi implementada utilizando tecnologias como Node.js, Express, PostgreSql e Angular, entre outras ferramentas complementares.

A realização da prova de conceito foi essencial para compreender o impacto da priorização da segurança sobre a operacionalidade do sistema, bem como para determinar quais recursos de segurança poderiam ser implementados sem comprometer a facilidade de uso. Foi evidenciado que, embora seja desejável atender a todas as necessidades do sistema, é fundamental priorizar alguns requisitos em detrimento de outros.

A opção por um modelo de serviços assíncronos aumentou a complexidade do sistema, especialmente em termos de manutenção, especialmente em cenários de alta demanda e concorrência de processamento de dados. No entanto, os resultados obtidos demonstraram um software funcional, com uma interface responsiva e adaptável para diferentes dispositivos. A arquitetura assíncrona permitiu uma melhor escalabilidade e facilitou a implementação de novos recursos, superando os desafios enfrentados.

Apesar dos pontos positivos, algumas limitações foram identificadas durante o desenvolvimento do sistema. A implementação de requisitos não funcionais, como a integração com sistemas legados e a garantia de disponibilidade 24/7, demandou mais tempo e esforço do que inicialmente previsto. Além disso, foram encontradas dificuldades na implementação de recursos mais avançados, como a personalização de menus e a gestão de estoque em tempo real.

Em conclusão, a plataforma desenvolvida neste trabalho apresentou uma arquitetura sólida e bem estruturada, com uma interface intuitiva e funcionalidades que atendem às necessidades específicas das cantinas escolares. Embora algumas limitações tenham sido identificadas, os pontos positivos superam os desafios enfrentados durante o processo de desenvolvimento, garantindo uma solução eficaz e eficiente para o gerenciamento das operações nas cantinas escolares.

Referências

● Eric Evans, Domain Driven Design, Edição traduzida - Alta Books, 2016

● Robert C Martin, Arquitetura Limpa, Edição traduzida - Alta Books, 2019

**Manfred Steyer**, Angular Microfrontend - Architecture and Projects with Module Federation (livro). Este livro fornece uma visão abrangente sobre micro frontends com Angular, abordando aspectos de arquitetura e projetos práticos com o conceito de Module Federation.