

**Documentação de um**

**Produto de Software**

**Mooca’s Pizzas**

**Nome dos Alunos: Bruno Godoy Dias, Matheus Henrique Oliveira Chuang, Raí Jóia Miquilino Valencio**

**2025-1**

**ÍNDICE DETALHADO**

**1.** **Introdução 3**

1.1. Tema 3

1.2. Objetivos a serem alcançados 3

1.3. Escopo principal 3

**2.** **Definição do Modelo de Processo 4**

**3.** **Requisitos do Sistema de Software 4**

3.1. Requisitos Funcionais 4

3.2. Requisitos Não-Funcionais 4

**4.** **Projeto 5**

4.1. Arquitetura Lógica 5

4.2. Arquitetura Física 5

**5.** **Protótipo de Interface 6**

**6.** **Critérios de Qualidade de Software 7**

**7. Testes 8**

7.1. Plano de Testes 8

7.2. Roteiro de Testes 8

1. **Introdução**
   1. **Tema**

Mooca's Pizzas, tem 15 anos de história de descendência italiana da Calábria, a especialidade da casa é pizza artesanal com fermentação natural, o mais pedido é a pizza mooca clássica que leva calabresa artesanal, cebola roxa caramelizada e azeitona preta e um toque de orégano fresco, quer automatizar a pizzaria mas sem perder o calor humano, atendimento acolhedor e ambiente familiar, o foco é oferecer uma experiência italiana com um pouco de paulistano.

* 1. **Objetivos a serem alcançados**

Fazer um aplicativo de espera por QR CODE, onde a pessoa irá cadastrar nome e número de pessoas que estão com ela, assim que chegar a sua vez, irá aparecer no celular o número da mesa disponível junto com uma foto de referência para ajudar a encontrar a mesa livre, para o garçom, irá chegar uma notificação no celular informando que uma mesa acabou de chegar informando o nome da pessoa que está na mesa e irá lançar o pedido no celular chegando na cozinha assim que lançado no sistema, o painel da cozinha irá mostrar os pedidos por ordem de chegada e assim que o pedido estiver pronto, a cozinha irá informar pelo aplicativo para que o garçom vá buscar e leve o pedido até a mesa, caso o cliente queira mais um pedido, terá um botão para apertar que irá chamar o garçom de novo.

Para manter a relação entre cliente e garçom, o garçom será o responsável por 5 mesas no máximo, onde somente ele pode atender essas mesas. Para o pagamento, o cliente terá a opção de ler o QR CODE na mesa para realizar o pagamento ou pode ir pagar no caixa do estabelecimento, finalizando seu atendimento

* 1. **Escopo principal**

Este documento de Especificação de Sistema descreve o escopo, funcionalidades e diretrizes para o desenvolvimento do sistema Easy Queue, uma solução digital voltada à modernização do atendimento e da gestão operacional da pizzaria Mooca’s Pizzas, sem comprometer a experiência acolhedora e tradicional que caracteriza o estabelecimento.

O escopo cobre todos os processos relacionados ao atendimento ao cliente — desde a recepção até o pagamento — bem como a comunicação entre salão, cozinha e administração. O sistema afetará positivamente os fluxos de trabalho internos (atendentes, garçons, pizzaiolos e caixa), os dispositivos utilizados (tablets, celulares e computadores), e estará integrado a soluções de pagamento e notificações móveis.

Este sistema deverá operar em múltiplas plataformas (Android, iOS, navegadores modernos), exigindo conexão com a internet, servidores para controle de dados e interface amigável para os usuários internos e clientes. Serão considerados aspectos de segurança, usabilidade e performance para assegurar estabilidade mesmo em horários de pico.

1. **Definição do Modelo de Processo**

Por se tratar de um negócio que funciona diariamente e de que quanto antes for implementado o software mais rápido os resultados serão notados, o processo escolhido foi o SCRUM, por ser um processo ágil baseado em entregas menores, o deployment de versões iniciais do software será uma espécie de estratégia para já verificar quais os pontos de melhora e de detalhamento além dos refinements semanais com o cliente que podem ajudar a melhor direcionar o projeto para evitar gastos desnecessários de ambas as partes. Tudo isso combinado ao fato de não focar tanto em documentações tão extensas uma vez que o stakeholder já admitiu gostar de processos mais simples e humanizados.

1. **Requisitos do Sistema de Software**
   1. **Requisitos Funcionais**

RF01 - O sistema deverá solicitar o nome do cliente e armazenar o mesmo.

RF02 - O sistema deverá notificar o cliente em seu celular quando a mesa estiver pronta.

RF03 - O sistema deverá permitir o acesso à fila por meio de QR Code e exibir uma foto de referência da mesa.

RF04 - O sistema deverá exibir um painel de gerenciamento com o status das mesas (livre, ocupada, aguardando limpeza, etc.).

RF05 - O sistema deverá permitir o cadastro de garçons, limitando a responsabilidade de cada um a no máximo 5 mesas.

RF06 - O sistema deverá permitir que os pedidos sejam feitos por tablet ou celular e enviados diretamente à cozinha.

RF07 - O sistema deverá exibir um painel na cozinha com os pedidos organizados por horário e número da mesa.

RF08 - O sistema deverá disponibilizar um botão na mesa para que o cliente possa chamar o garçom.

RF09 - O sistema deverá gerar a conta automaticamente, com opção de divisão por pessoa.

RF10 - O sistema deverá aceitar os seguintes métodos de pagamento: cartão de crédito/débito, Pix, VR, NFC e QR Code.

RF11 - O sistema deverá gerar relatórios de vendas, tempo de preparo e desempenho dos garçons.

* 1. **Requisitos Não-Funcionais**

RNF01 - O sistema deverá possuir uma interface com poucos elementos, adequada para uso em celulares e tablets. Categoria: Hardware e Software.

RNF02 - O sistema deverá utilizar uma paleta de cores tradicional italiana, incluindo vinho, verde escuro, creme, marrom madeira e grafite. Categoria: Interface.

RNF03 - O sistema deverá garantir maior disponibilidade durante os horários de pico, especialmente no período noturno. Categoria: Performance.

RNF04 - O sistema deverá ser integrado com notificações por celular. Hardware e Software.

RNF05 - O sistema deverá assegurar a proteção dos dados dos usuários e a segurança nas transações de pagamento. Categoria: Segurança.

1. **Projeto**
   1. **Arquitetura Lógica**

A aplicação do sistema da Mooca’s Pizza segue uma arquitetura lógica em camadas, baseada na Clean Architecture, com inspiração nos modelos MVC (Model-View-Controller) e MVVM (Model-View-ViewModel). Essa escolha foi feita com o objetivo de garantir uma separação clara de responsabilidades e facilitar a manutenção, testabilidade e escalabilidade do sistema.

Camadas do Sistema:

* Camada de Apresentação (ViewModel/Controller): Responsável por receber requisições dos usuários (via API REST) e encaminhá-las para os casos de uso correspondentes.
* Camada de Aplicação (Use Cases): Contém a lógica de negócio central da aplicação. Aqui são definidos os fluxos principais do sistema, como criação de pedidos, cadastro de clientes e emissão de relatórios.
* Camada de Domínio (Entidades e Interfaces): Representa os modelos de dados e as regras de negócio. Esta camada é completamente independente de frameworks, bancos de dados ou qualquer tecnologia externa.
* Camada de Infraestrutura (Repositórios e Gateways): Implementa os contratos definidos nas camadas superiores. É onde acontecem as interações reais com o banco de dados e serviços externos.

Tecnologias e Padrões:

* Linguagem: Node.js (JavaScript)
* Frameworks: Nenhum framework utilizado, optando por uma abordagem “bare metal” para reforçar o domínio da arquitetura.
* Banco de dados: PostgreSQL — escolhido por sua confiabilidade, aderência a padrões SQL e boa integração com aplicações Node.js.
* Padrão de projeto: Repository Pattern, aplicado para isolar a lógica de acesso a dados e permitir flexibilidade na troca de tecnologias de persistência.

Componentes externos:

* Gateway de Pagamento (simulado): para processar pagamentos online de forma segura.
* Serviço de Notificações (simulado): envio automático de atualizações por e-mail ou mensagens aos clientes e à equipe interna.
  1. **Arquitetura Física**

A arquitetura física do sistema foi definida para simular uma estrutura moderna de hospedagem em nuvem, com foco em simplicidade, escalabilidade e fácil integração ao pipeline de CI/CD.

Infraestrutura e Ambientes

Hospedagem: Railway, plataforma moderna que suporta deploys contínuos a partir de repositórios GitHub e possui suporte nativo a ambientes distintos.

Ambientes:

* Staging: utilizado para testes internos e validação antes da publicação.
* Produção: ambiente acessado pelos usuários finais, com maior estabilidade.

Componentes da Infraestrutura

Servidor de Aplicação:

* Responsável por hospedar a aplicação Node.js.
* Recursos estimados (simulados): 2 vCPU, 4GB RAM

Banco de Dados PostgreSQL:

* Instância gerenciada na Railway ou integrada via addon.

Serviços Externos:

* Gateway de pagamento e sistema de notificações integrados via APIs.

Monitoramento básico (simulado):

* Para logs e health check da aplicação.

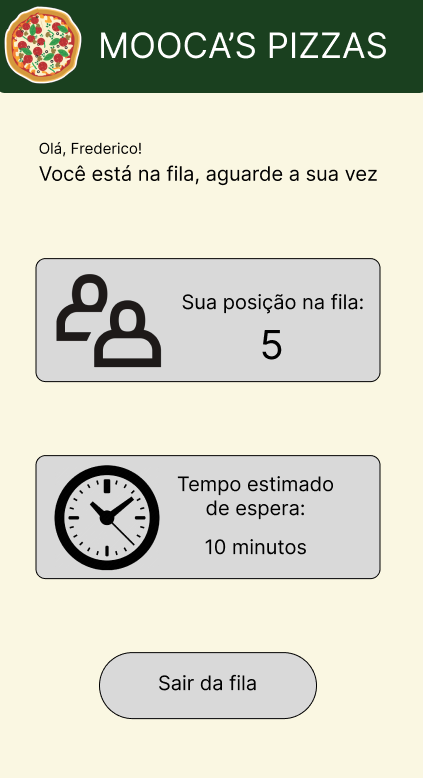
Comunicação e Deploy

A comunicação entre os ambientes é feita via pipeline automatizado (CI/CD) utilizando GitHub Actions.

O deploy automático é acionado a cada push na branch main, sendo direcionado primeiro ao ambiente de staging para validações, e posteriormente à produção.

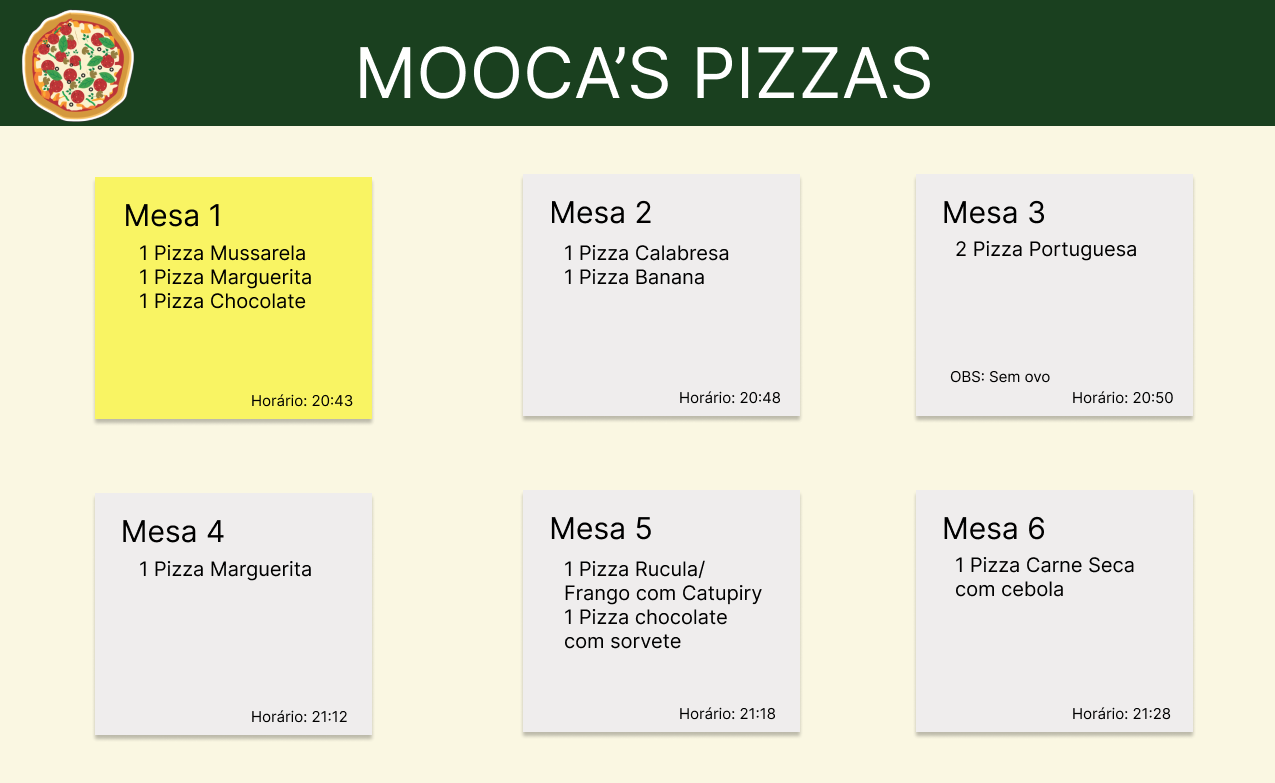
1. **Protótipo de Interface**

Levando em consideração os requisitos levantados com o stakeholder e os clientes finais chegamos em um protótipo que além de atender a simplicidade solicitada com o mínimo de elementos em tela, nos casos mais necessário ainda mantivemos os detalhes e maiores opções de interações para ajudar na produtividade.

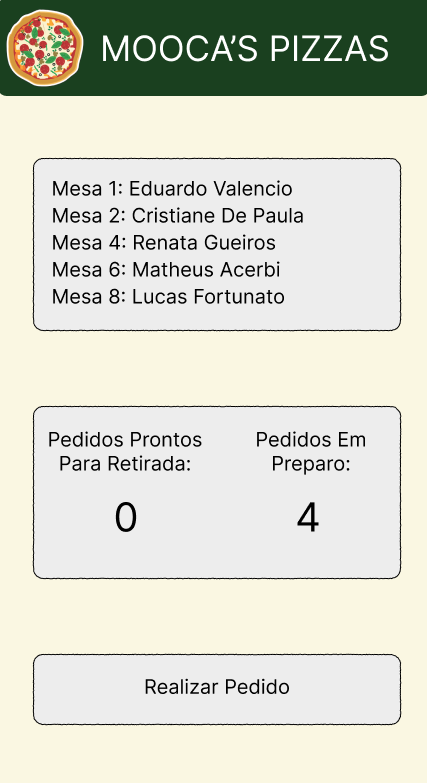
Tela de Espera:  


Tela de Mesa Disponível:  


Tela de Pedido:  


Tela da Cozinha:   


Tela do Garçom:

****

1. **Critérios de Qualidade de Software**

Adotamos como referência os princípios estabelecidos pela norma ISO/IEC 9126, que define um conjunto de características e subcaracterísticas para avaliar a qualidade de produtos de software.

Nos preocupamos em aplicar conceitos de qualidade ao longo do projeto, especialmente por meio da simulação de práticas de DevOps, definição de arquitetura e planejamento das funcionalidades.

Dentre os diversos critérios possíveis, optamos por focar em confiabilidade, eficiência, portabilidade, usabilidade e manutenabilidade, de acordo com o escopo e os objetivos desta etapa do projeto.

**Confiabilidade**

Buscamos garantir que o sistema, represente um fluxo de funcionamento previsível e seguro, com foco em evitar erros lógicos nas etapas do pipeline. A execução automatizada de testes unitários e validações contínuas no fluxo de integração são práticas simuladas que reforçam esse critério.

**Eficiência**

Adotamos o conceito de automação dos processos com uso de CI/CD e deploy automatizado, minimizando o esforço humano nas entregas e reforçando a ideia de eficiência operacional.

**Portabilidade**

A opção por utilizar o Node.js como linguagem principal e o Railway como plataforma de hospedagem foi pensada para simular um sistema leve e facilmente portável, com baixo acoplamento com ambientes específicos. Essa abordagem facilita possíveis migrações ou adaptações futuras.

**Usabilidade**

A usabilidade foi considerada na elaboração de sprints com foco em jornadas simplificadas para o cliente final, simulando funcionalidades como personalização de pedidos, formas de pagamento integradas e comunicação clara através de notificações.

**Manutenabilidade**

A arquitetura lógica baseada em Clean Architecture, separada por responsabilidades bem definidas e sem dependência de frameworks específicos, foi escolhida para permitir a facilidade de manutenção futura. A documentação do pipeline e as práticas DevOps também colaboram para a rastreabilidade e ajustes no sistema.

Critério Não Priorizado:

**Escalabilidade**

Pelo reforço constante por parte do stakeholder de que o restaurante tem uma proposta intimista e não foca no atendimento massificado de clientes, o critério de escalabilidade não foi priorizado nesta etapa. Não foram feitas simulações de alta carga, replicação de dados ou balanceamento de serviços. Esse critério poderá ser considerado em versões futuras com foco em crescimento de uso ou suporte a múltiplas unidades.

**7. Testes**

**7.1. Plano de Testes**

Este plano de testes tem como objetivo garantir que o sistema **Easy Queue** esteja em conformidade com os requisitos estabelecidos no projeto da Mooca’s Pizzas, tanto funcionais quanto não-funcionais. Os testes serão conduzidos em ambiente de staging simulando o uso real do sistema.

# Escopo

Testes manuais e automatizados focando:

* Gestão da fila e mesas
* Integração cozinha-garçom-cliente
* Painel da cozinha
* Sistema de pedidos
* Integrações (pagamento e notificações)

# Objetivos

* Verificar o correto funcionamento dos fluxos
* Garantir usabilidade e performance
* Simular operações reais em ambiente staging

# Requisitos a Serem Testados

* Todos os RFs listados (RF01 a RF11)
* RNFs críticos: RNF01, RNF03, RNF05

# Estratégias, Tipos de Testes e Ferramentas

* **Estratégia:** Testes baseados em casos de uso
* **Tipos de Teste:** Funcional, Regressão, Usabilidade, Integração
* **Ferramentas:** Jest (unit), Postman (API), Cypress (E2E)

# Recursos a Serem Empregados

* Equipe de QA
* Railway (ambientes de staging)
* Mock de APIs de terceiros
* Banco de dados com dados simulados

**7.2. Roteiro de Testes**

Exemplo: Roteiro do Caso de Teste CT01 – Acesso via QR Code

Pré-condição: Cliente possui celular com câmera e acesso à internet.

Passos:

Cliente aponta o celular para QR Code da entrada.

Sistema exibe interface de cadastro.

Cliente preenche nome e número de pessoas.

Confirma o cadastro.

Resultado Esperado: Cliente é adicionado à fila com sucesso.