Trabalho Final - Sistema de Gestão de Restaurante

Ano Letivo 2025-2026

Unidade Curricular: Engenharia de Software

Curso: Engenharia Informática

Ano: 2º

Docente: Pedro Martins

Tipo de Trabalho: Trabalho Final

Trabalho elaborado por:

Gonçalo Caçador Nº230001121

Índice

Trabalho Final - Sistema de Gestão de Restaurante **1**
Índice 2
Introdução 6
Resumo sobre o Sistema 7
Análise de Stakeholders 8
1. Cliente: 8
Descrição: 8
Objetivos: 8
Necessidades: 8
Preocupações: 8
2. Empregado de Mesa: 9
Descrição: 9
Objetivos: 9
Necessidades: 9
Preocupações: 9
3. Cozinheiro 10
Descrição: 10
Objetivos: 10
Necessidades: 10
Preocupações: 10
4. Gerente 11
Descrição: 11
Objetivos: 11
Necessidades: 11
Preocupações: 11
5. Sistema de Pagamento 12
Descrição: 12
Objetivos: 12
Necessidades: 12
Preocupações: 12
Resumo Stakeholders: 12
Requisitos Funcionais 13
Gestão de Reservas 13

RF-01: 13
RF-02: 13
Gestão de Pedidos 13
RF-03: 13
RF-04: 13
RF-05: 13
Estado dos Pedidos 13
RF-06: 13
RF-07: 13
Faturação e Pagamento 14
RF-08: 14
RF-09: 14
Gestão de Stock 14
RF-10: 14
RF-11: 14
Resumo Requisitos Funcionais 14
Requisitos Não Funcionais 15
Desempenho 15
RNF-01: 15
Usabilidade 15
RNF-02: 15
Fiabilidade 15
RNF-03: 15
Segurança 15
RNF-04: 15
Escalabilidade 15
RNF-05: 15
Matriz de Rastreabilidade dos Requisitos 16
Diagrama de Casos de Uso 17
Diagrama de Classes 18
Diagramas de Sequência 19
Criar Pedido (Empregado de Mesa): 19
Atualização do Estado do Pedido (Cozinha): 20
Pagamento da Conta: 21

Reserva de Mesa (Cliente): 22
Atualização de Stock após Venda (Gerente): 23
Diagramas de Atividades 24
Processo de Criação de Pedido: 24
Processo de Preparação do Pedido (Cozinha): 25
Processo de Pagamento: 26
Documento de Arquitetura do Software 27
Visão Geral da Arquitetura: 27
Estilo Arquitetural: 27
Padrões de Desenho Utilizados: 28
Padrão State: 28
Padrão Factory: 28
Padrão Repository: 28
Aplicação dos Princípios SOLID: 29
Liskov Substitution Principle (LSP): 29
Dependency Inversion Principle (DIP): 29
Justificação da Tecnologia Utilizada 30
Gestão do Projeto 31
Product Backlog 31
US-01 – (Alta prioridade – 5 pontos – Implementado): 31
US-02 – (Alta prioridade – 5 pontos – Implementado): 31
US-03 – (Alta prioridade – 3 pontos – Implementado): 32
US-04 – (Média prioridade – 3 pontos – Não implementado): 32
US-05 – (Média prioridade – 2 pontos – Não implementado): 32
Planeamento de Sprints 32
Estrutura do Repositório Git 33
Boas Práticas de Versionamento 33
Documentação de Testes 34
Estratégia de Testes 34
Objetivo dos Testes 34
Níveis de Teste Aplicados 34
Tipos de Teste Utilizados 35
Ferramentas de Teste 35
Ambiente de Teste 35

Documentação dos Casos de Teste 36
TC-01 – Criar Pedido 36
TC-02 – Criar Pedido sem Mesa 36
TC-03 – Adicionar Item Disponível ao Pedido 36
TC-04 – Adicionar Item Indisponível 37
TC-05 – Remover Item do Pedido 37
TC-06 – Criar Pedido com Vários Itens 37
TC-07 – Estado Inicial do Pedido 37
TC-08 – Atualizar Estado para Em Preparação 38
TC-09 – Atualizar Estado para Pronto 38
TC-10 – Atualizar Estado Inválido 38
TC-11 – Calcular Total da Fatura 38
TC-12 – Calcular Fatura sem Itens 39
TC-13 – Registar Pagamento com Sucesso 39
TC-14 – Registar Pagamento Duplicado 39
TC-15 – Criar Reserva Válida 39
TC-16 – Criar Reserva sem Mesas Disponíveis 40
TC-17 – Cancelar Reserva 40
TC-18 – Visualizar Pedidos na Cozinha 40
TC-19 – Atualizar Stock após Venda 40
TC-20 – Stock Insuficiente 41
TC-21 – Adicionar Novo Item ao Menu 41
TC-22 – Remover Item do Menu 41
TC-23 – Acesso Não Autorizado 41
TC-24 – Tempo de Resposta 42
TC-25 – Criar Pedido Simultâneo 42
TC-26 – Integridade dos Dados 42
TC-27 – Quantidade Limite de Itens 42
TC-28 – Quantidade Acima do Limite 43
TC-29 – Sequência Completa de Estados 43
TC-30 – Pedido Finalizado 43
Conclusão 44

Introdução

Este trabalho consiste no desenvolvimento de um Sistema de Gestão de Restaurante, com o objetivo de apoiar e organizar as principais operações de um restaurante. O sistema pretende facilitar a gestão de pedidos, reservas, faturação e controlo de stock, melhorando a eficiência e reduzindo erros no funcionamento diário.

O projeto aplica conceitos de Engenharia de Software, incluindo análise de requisitos, modelação UML, design orientado a objetos, padrões de desenho e testes automatizados, tendo planeamento teórico com implementação prática em Java.

Resumo sobre o Sistema

Este projeto consiste na conceção e implementação de um Sistema de Gestão de Restaurante. O objetivo principal deste sistema é apoiar as operações diárias de um restaurante, fornecendo uma forma organizada e eficiente de gerir reservas, pedidos, faturação e controlo de stock.

Os restaurantes enfrentam vários desafios operacionais, tais como a gestão simultânea de vários pedidos, o acompanhamento do estado dos pedidos entre a sala e a cozinha, a organização das reservas de mesas e a garantia de uma faturação correta. Este sistema pretende reduzir erros, melhorar a comunicação entre os membros da equipa e aumentar a eficiência geral do restaurante.

O sistema foi concebido para apoiar tanto as operações de atendimento ao cliente como as operações internas. Os empregados de mesa utilizam o sistema para registar pedidos e atualizar o seu estado. A equipa da cozinha utiliza o sistema para visualizar e gerir os pedidos em preparação. O gerente utiliza o sistema para gerir o stock, o menu e analisar informações relacionadas com as vendas. Os clientes interagem com o sistema de forma indireta, através dos funcionários do restaurante.

O Sistema de Gestão de Restaurante segue boas práticas de Engenharia de Software, incluindo design orientado a objetos, modelação UML, aplicação de padrões de desenho e testes automatizados. Apenas os componentes mais críticos do sistema são implementados em Java, acompanhados por uma suite de testes JUnit, garantindo um equilíbrio entre o design completo do sistema e a implementação prática.

Análise de Stakeholders

Os stakeholders são todas as pessoas (ou sistemas) que:

- usam o sistema
- ou são afetadas por ele

1. **Cliente:**

Descrição:

- Pessoa que frequenta o restaurante para realizar refeições.

Objetivos:

- Receber o pedido correto
- Ter um serviço rápido e eficiente

Necessidades:

- Pedidos registados sem erros
- Reservas organizadas

Preocupações:

- Atrasos no atendimento
- Erros nos pedidos ou na conta

2. **Empregado de Mesa:**

Descrição:

- Pessoa que frequenta o restaurante para realizar refeições.

Objetivos:

- Receber o pedido correto
- Ter um serviço rápido e eficiente

Necessidades:

- Pedidos registados sem erros
- Reservas organizadas

Preocupações:

- Atrasos no atendimento
- Erros nos pedidos ou na conta

3. **Cozinheiro**

Descrição:

- Funcionário responsável pela preparação dos pedidos.

Objetivos:

- Visualizar pedidos corretamente
- Saber a prioridade de cada pedido

Necessidades:

- Lista clara de pedidos em preparação
- Informação sobre alterações ou cancelamentos

Preocupações:

- Pedidos incompletos ou errados
- Falta de atualização do estado dos pedidos

4. **Gerente**

Descrição:

- Responsável pela gestão geral do restaurante.

Objetivos:

- Controlar o stock
- Analisar vendas
- Garantir o bom funcionamento do restaurante

Necessidades:

- Relatórios de vendas
- Controlo de inventário
- Gestão do menu

Preocupações:

- Falta de controlo de stock
- Perdas financeiras

5. **Sistema de Pagamento**

Descrição:

Sistema externo responsável pelo processamento de pagamentos.

Objetivos:

- Processar pagamentos com segurança

Necessidades:

- Dados corretos da fatura

Preocupações:

- Falhas de comunicação
- Erros nos valores cobrados

Resumo Stakeholders:

O Sistema de Gestão de Restaurante envolve vários stakeholders com diferentes responsabilidades e necessidades.

Os principais stakeholders são os clientes, que esperam um serviço rápido e correto; os empregados de mesa, responsáveis pelo registo e acompanhamento dos pedidos; os cozinheiros, que necessitam de informação clara sobre os pedidos em preparação; e o gerente, que utiliza o sistema para gerir o stock, o menu e analisar as vendas. Adicionalmente, o sistema de pagamento é um stakeholder externo essencial para garantir uma faturação segura e correta. Todos estes intervenientes dependem do sistema para melhorar a eficiência e reduzir erros nas operações do restaurante.

Requisitos Funcionais

São descrições claras e objetivas do que o sistema tem de fazer devem ser específicas e possam ser testadas.

Gestão de Reservas

RF-01:

O sistema deve permitir criar reservas de mesas para uma data e hora específicas.

RF-02:

O sistema deve impedir reservas em horários onde não existam mesas disponíveis.

Gestão de Pedidos

RF-03:

O sistema deve permitir ao empregado de mesa criar um novo pedido para uma mesa.

RF-04:

O sistema deve permitir adicionar e remover itens do menu a um pedido.

RF-05:

O sistema deve associar cada pedido a uma mesa específica.

Estado dos Pedidos

RF-06:

O sistema deve permitir atualizar o estado do pedido (Recebido, Em Preparação, Pronto, Servido).

RF-07:

O sistema deve permitir à cozinha visualizar os pedidos em preparação.

Faturação e Pagamento

RF-08:

O sistema deve calcular automaticamente o valor total da conta.

RF-09:

O sistema deve permitir o registo do pagamento de um pedido.

Gestão de Stock

RF-10:

O sistema deve atualizar o stock com base nos itens vendidos.

RF-11:

O sistema deve impedir a adição de itens indisponíveis a um pedido.

Resumo Requisitos Funcionais

Os requisitos funcionais definem as principais funcionalidades do Sistema de Gestão de Restaurante, incluindo a gestão de reservas, o registo e acompanhamento de pedidos, a atualização do estado dos pedidos, a faturação e o controlo de stock. Estes requisitos garantem o correto funcionamento das operações do restaurante e servem de base para o desenvolvimento do sistema, dos diagramas UML e dos testes.

Requisitos Não Funcionais

Desempenho

RNF-01:

O sistema deve responder às ações do utilizador em menos de 2 segundos.

Usabilidade

RNF-02:

O sistema deve possuir uma interface simples e fácil de utilizar pelos funcionários do restaurante.

Fiabilidade

RNF-03:

O sistema deve garantir que os dados dos pedidos não são perdidos em caso de erro.

Segurança

RNF-04:

O sistema deve permitir o acesso apenas a utilizadores autorizados.

Escalabilidade

RNF-05:

O sistema deve suportar múltiplos pedidos em simultâneo sem perda de desempenho.

Matriz de Rastreabilidade dos Requisitos

ID do Requisito	Descrição	Stakeholder	Referência de Design (Classe)	Casos de Teste
RF-01	Criar reservas de mesas	Cliente	Reserva	TC-01
RF-02	Impedir reservas sem mesas disponíveis	Cliente	Reserva	TC-02
RF-03	Criar pedido para uma mesa	Empregado de Mesa	Pedido	TC-03
RF-04	Adicionar/remover itens do pedido	Empregado de Mesa	Pedido, ItemMenu	TC-04
RF-06	Atualizar estado do pedido	Cozinheiro	Pedido	TC-05
RF-08	Calcular valor total da conta	Gerente	Fatura	TC-06
RF-10	Atualizar stock após venda	Gerente	Stock	TC-07
RFN-01	Tempo de resposta < 2 segundos	Todos	Arquitetura	TC-08
RFN-04	Acesso apenas a utilizadores autorizados	Gerente	Utilizador	TC-09

Diagrama de Casos de Uso

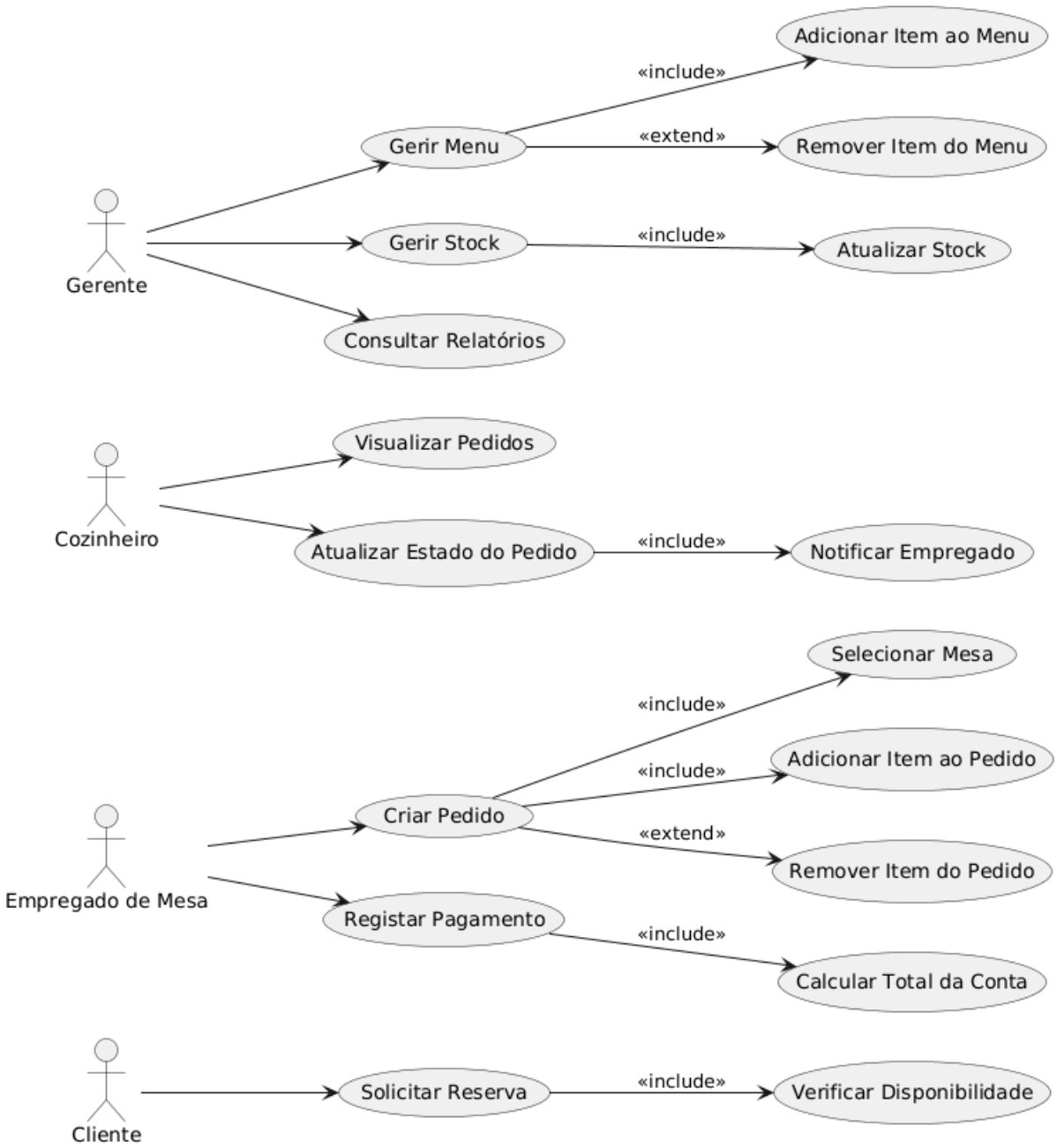
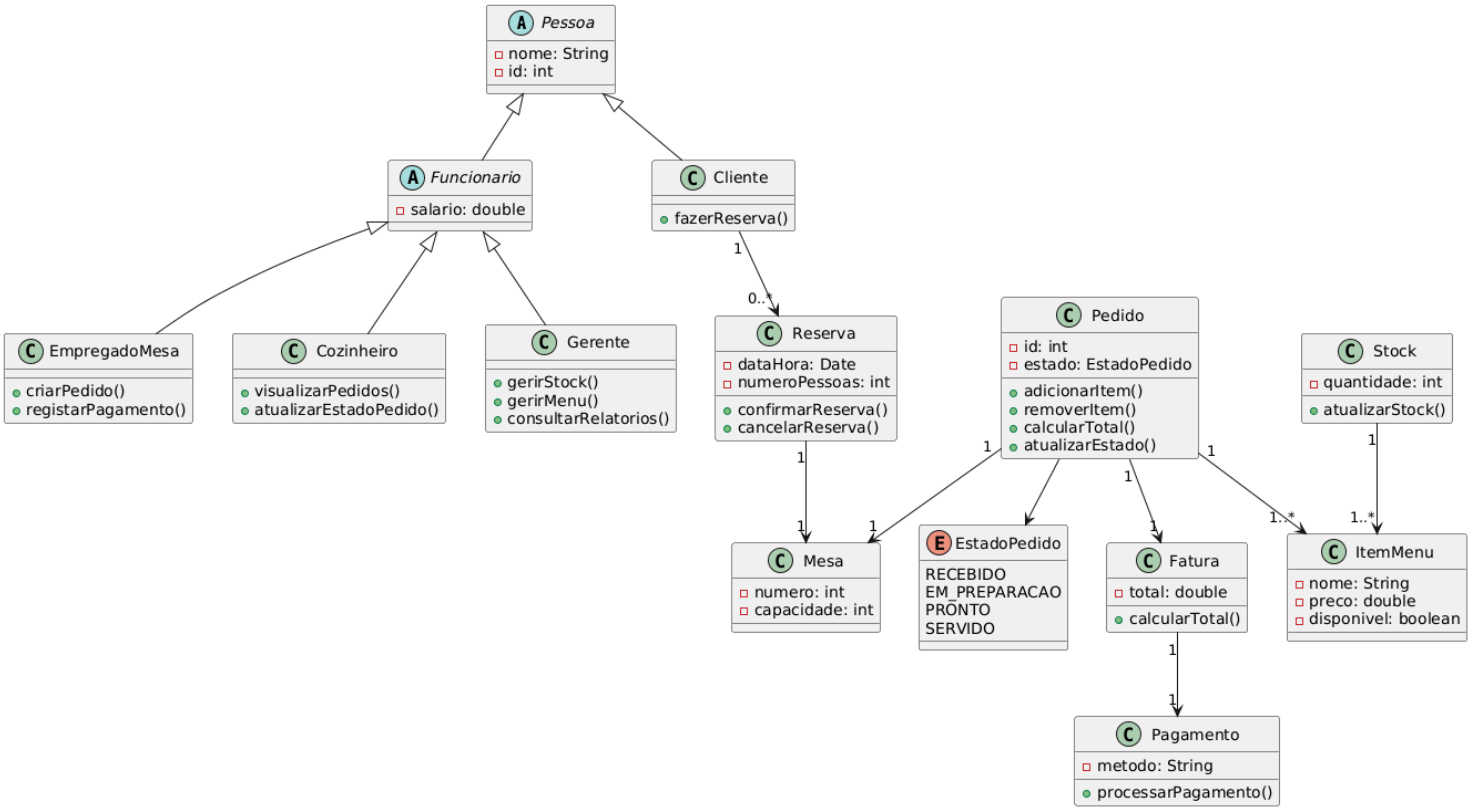
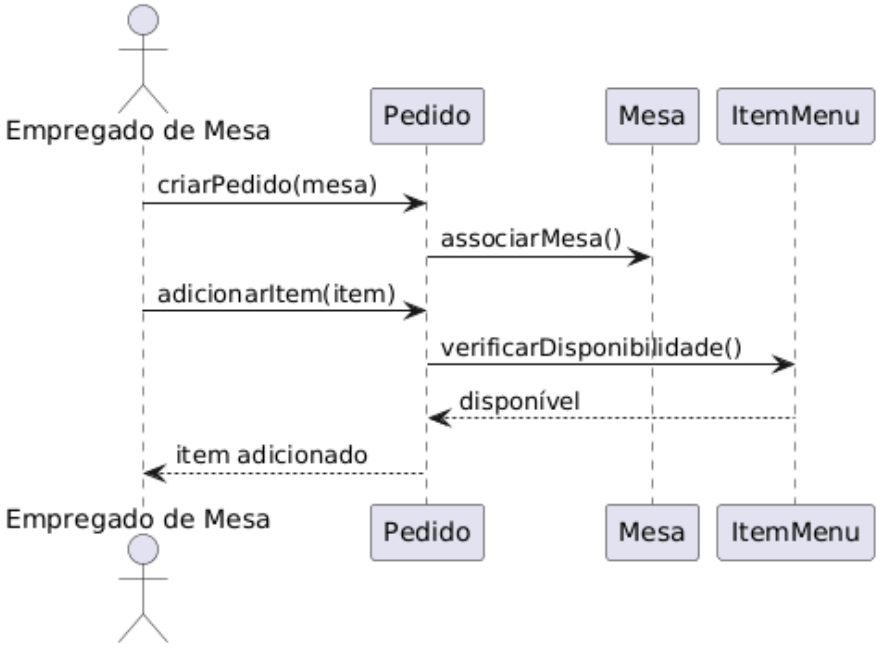


Diagrama de Classes



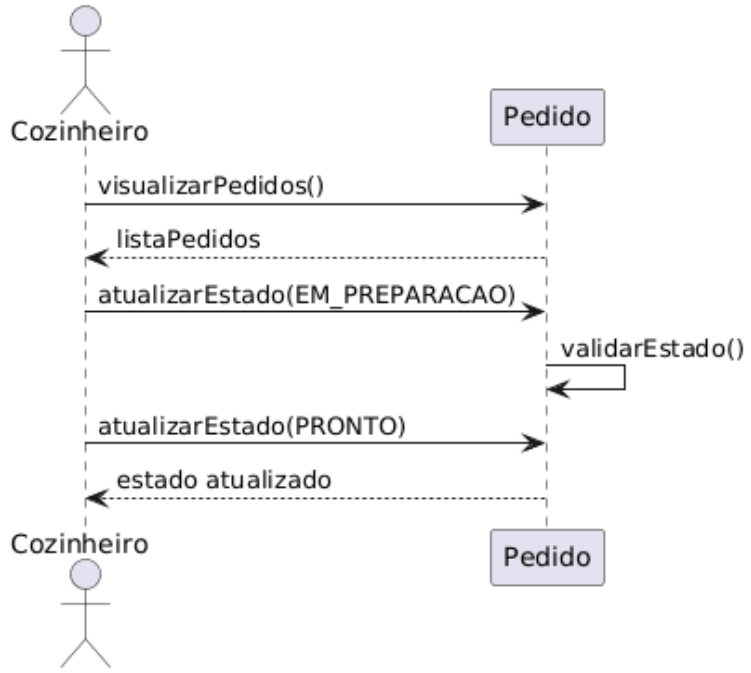
Diagramas de Sequência

Criar Pedido (Empregado de Mesa):



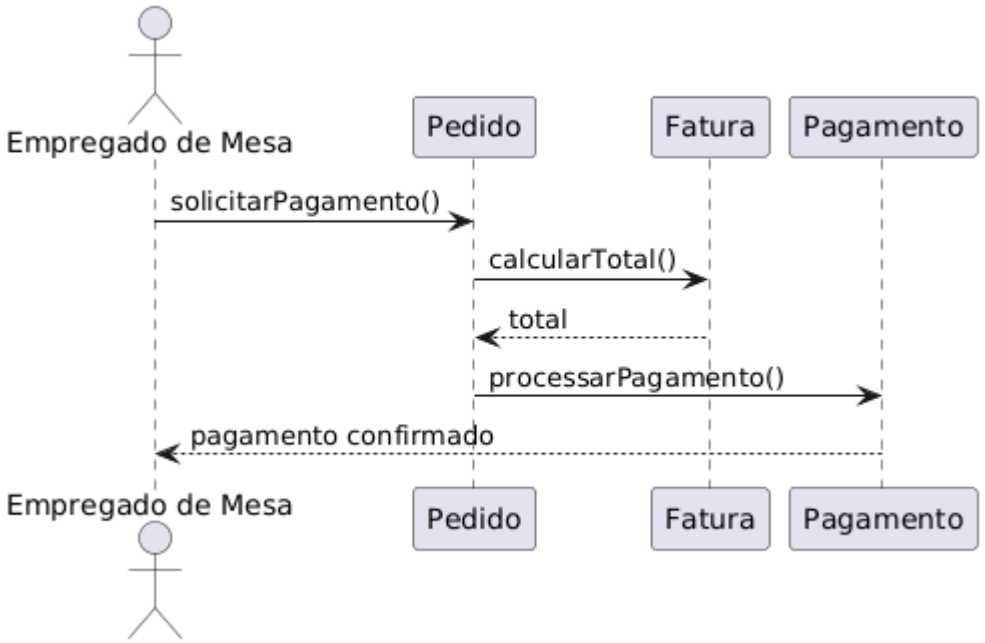
Este diagrama representa o processo de criação de um pedido por parte do empregado de mesa. Mostra a associação do pedido a uma mesa e a verificação da disponibilidade dos itens do menu antes de serem adicionados ao pedido, garantindo que apenas itens disponíveis podem ser registrados.

Atualização do Estado do Pedido (Cozinha):



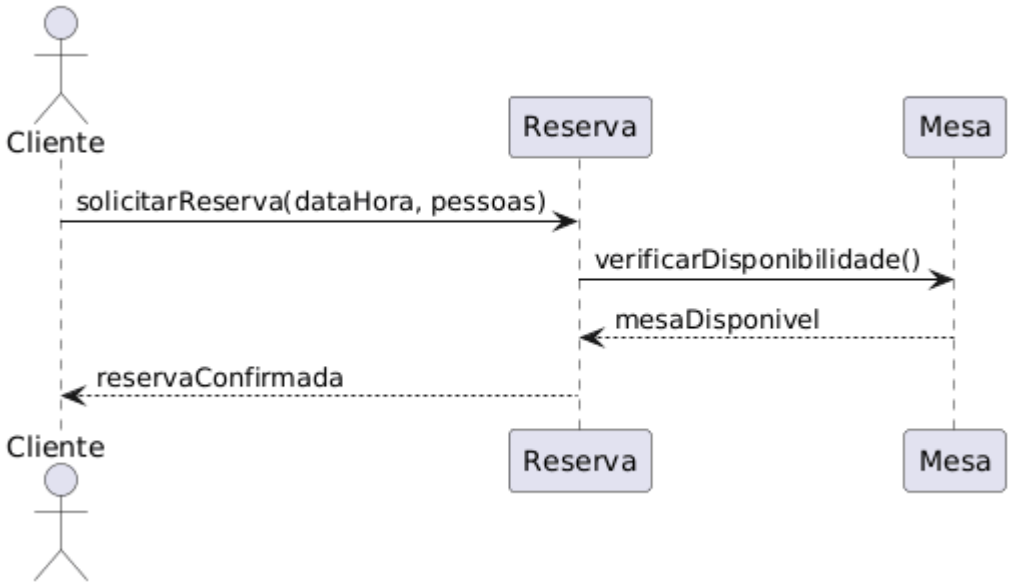
Este diagrama ilustra a interação entre o cozinheiro e o sistema durante a preparação dos pedidos. O cozinheiro visualiza os pedidos existentes e atualiza o seu estado ao longo do processo, permitindo um acompanhamento correto do ciclo de vida do pedido.

Pagamento da Conta:



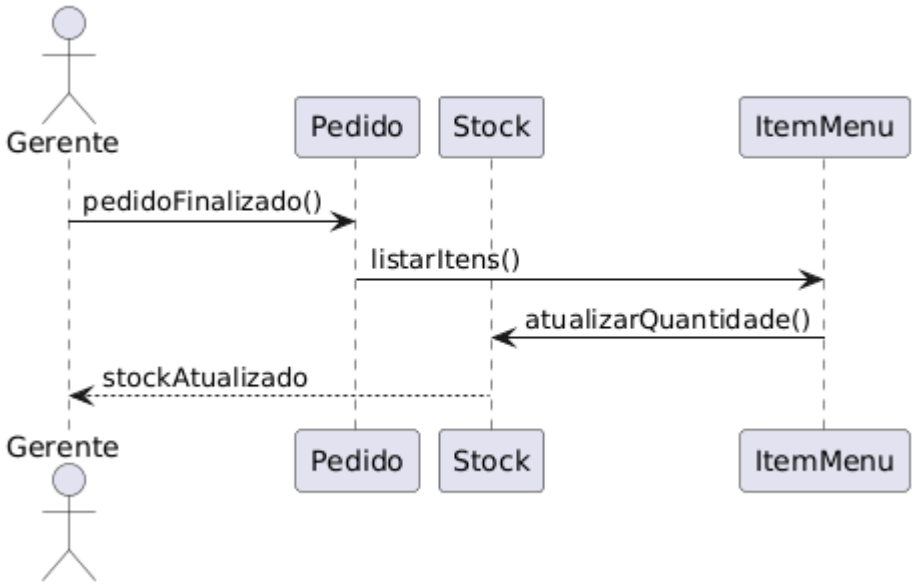
Este diagrama descreve o processo de pagamento de um pedido. O empregado de mesa solicita o pagamento, o sistema calcula o total da fatura e o pagamento é processado, garantindo que a transação é registrada corretamente.

Reserva de Mesa (Cliente):



Este diagrama apresenta o fluxo de uma reserva de mesa efetuada por um cliente. O sistema verifica a disponibilidade das mesas para a data e hora solicitadas e confirma a reserva caso exista disponibilidade.

Atualização de Stock após Venda (Gerente):



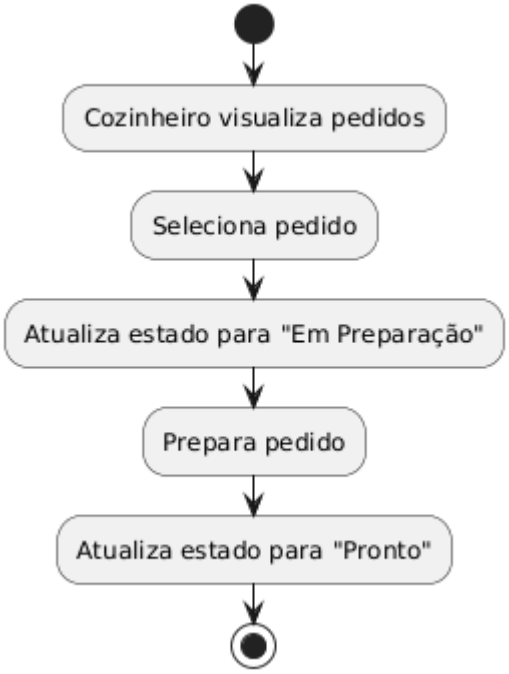
Este diagrama representa o processo de atualização do stock após a finalização de um pedido. Os itens vendidos são identificados e as respetivas quantidades em stock são atualizadas, assegurando um controlo correto do inventário.

Diagramas de Atividades

Processo de Criação de Pedido:

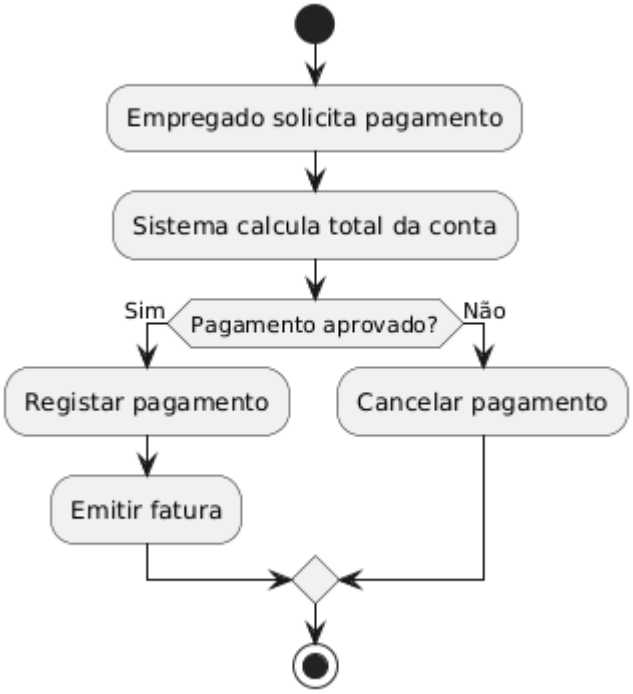
Este diagrama representa o processo de criação de um pedido no restaurante. O empregado de mesa cria o pedido, seleciona a mesa e adiciona itens do menu. O sistema verifica a disponibilidade dos itens antes de os adicionar, garantindo que apenas itens disponíveis são registrados.

Processo de Preparação do Pedido (Cozinha):



Este diagrama descreve o fluxo de atividades realizado pela cozinha durante a preparação de um pedido. O cozinheiro visualiza os pedidos, seleciona um pedido específico e atualiza o seu estado à medida que o processo de preparação evolui.

Processo de Pagamento:



Este diagrama representa o processo de pagamento de um pedido. O sistema calcula o valor total da conta e verifica se o pagamento foi aprovado. Em caso de sucesso, o pagamento é registrado e a fatura é emitida; caso contrário, o pagamento é cancelado.

Documento de Arquitetura do Software

Visão Geral da Arquitetura:

O Sistema de Gestão de Restaurante foi concebido com o objetivo de garantir uma estrutura clara, modular e de fácil manutenção. A arquitetura do sistema separa as responsabilidades principais, permitindo uma melhor organização do código e facilitando futuras alterações ou extensões.

O sistema é composto por vários componentes principais, incluindo a gestão de pedidos, reservas, faturação, stock e utilizadores. Cada componente comunica com os restantes de forma controlada, respeitando os princípios de encapsulamento e baixo acoplamento.

Estilo Arquitetural:

O sistema segue o estilo arquitetural em camadas (Layered Architecture), organizado da seguinte forma:

- **Camada de Apresentação**
  Responsável pela interação com os utilizadores (empregado de mesa, cozinheiro e gerente).
- **Camada de Lógica de Negócio**
  Contém as regras principais do sistema, como a criação de pedidos, atualização de estados, cálculo da fatura e gestão de stock.
- **Camada de Dados**
  Responsável pela gestão e persistência dos dados do sistema (pedidos, reservas, itens do menu, stock).

Este estilo arquitetural facilita a manutenção, a reutilização de código e a realização de testes.

Padrões de Desenho Utilizados:

Padrão State:

- **Problema:** Um pedido pode ter vários estados ao longo do seu ciclo de vida.
- **Aplicação:** Utilizado na gestão do estado do pedido (Recebido, Em Preparação, Pronto, Servido).
- **Benefícios:** Torna a gestão de estados mais organizada e evita estruturas condicionais complexas.

Padrão Factory:

- **Problema:** Criação de diferentes tipos de pagamentos sem dependência direta das classes concretas.
- **Aplicação:** Utilizado para criar objetos do tipo Pagamento conforme o método escolhido.
- **Benefícios:** Facilita a adição de novos métodos de pagamento sem alterar código existente.

Padrão Repository:

- **Problema:** Acesso direto aos dados espalhados pelo sistema.
- **Aplicação:** Centraliza o acesso a dados como pedidos, reservas e itens do menu.
- **Benefícios:** Reduz acoplamento e melhora a organização do código.

Aplicação dos Princípios SOLID:

Single Responsibility Principle (SRP):

Cada classe possui uma única responsabilidade, como Pedido, Reserva ou Fatura.

Open/Closed Principle (OCP):

O sistema permite a extensão de funcionalidades, como novos métodos de pagamento, sem modificar classes existentes.

Liskov Substitution Principle (LSP):

As subclasses de Funcionário podem ser utilizadas de forma transparente no sistema.

Interface Segregation Principle (ISP):

Interfaces são específicas e focadas, evitando métodos desnecessários.

Dependency Inversion Principle (DIP):

As classes dependem de abstrações e não de implementações concretas.

Justificação da Tecnologia Utilizada

A escolha das tecnologias utilizadas no desenvolvimento do Sistema de Gestão de Restaurante teve como principal objetivo garantir robustez, facilidade de manutenção, suporte à programação orientada a objetos e compatibilidade com testes automatizados.

A linguagem Java foi selecionada por ser uma linguagem madura, amplamente utilizada no desenvolvimento de sistemas empresariais e por oferecer um forte suporte à programação orientada a objetos. Java facilita a aplicação de princípios como encapsulamento, herança e polimorfismo, essenciais para a correta implementação dos conceitos de Engenharia de Software abordados neste projeto. Adicionalmente, a sua portabilidade e vasta documentação tornam-na adequada para projetos académicos e profissionais.

Para a implementação dos testes automatizados, foi utilizado o JUnit, uma framework padrão para testes em Java. O JUnit permite a criação de testes unitários claros, repetíveis e automatizados, contribuindo para a deteção precoce de erros e para a garantia da qualidade do software. A utilização do JUnit também facilita a medição da cobertura de código, requisito fundamental do projeto.

O Maven/Gradle foi adotado como ferramenta de gestão de dependências e construção do projeto, permitindo uma organização clara do código, a automatização do processo de compilação e a fácil integração de bibliotecas externas. Esta abordagem aproxima o projeto de práticas profissionais utilizadas na indústria de software.

Para a modelação e documentação do sistema, foi utilizado o StarUML, que permite a criação de diagramas UML normalizados e bem estruturados. A utilização do StarUML, em conjunto com diagramas de classes, casos de uso, sequência e atividades, facilita a comunicação das decisões de design e assegura a consistência entre a documentação e a implementação.

Em conjunto, estas tecnologias garantem uma base sólida para o desenvolvimento de um sistema bem estruturado, testável e alinhado com as boas práticas de Engenharia de Software.

Gestão do Projeto

Product Backlog

US-01 – (Alta prioridade – 5 pontos – Implementado):

Como empregado de mesa, quero criar pedidos para mesas, para registar corretamente os pedidos dos clientes.

Critérios de Aceitação:

- É possível associar um pedido a uma mesa
- O pedido pode conter vários itens
- O pedido tem um estado inicial

US-02 – (Alta prioridade – 5 pontos – Implementado):

Como cozinheiro, quero visualizar e atualizar o estado dos pedidos, para gerir a preparação dos pedidos.

Critérios de Aceitação:

- Pedidos visíveis em tempo real
- Estados atualizáveis (Recebido, Em Preparação, Pronto)

US-03 – (Alta prioridade – 3 pontos – Implementado):

Como empregado de mesa, quero registar pagamentos, para finalizar os pedidos corretamente.

Critérios de Aceitação:

- Cálculo automático do total
- Confirmação do pagamento

US-04 – (Média prioridade – 3 pontos – Não implementado):

Como cliente, quero fazer reservas, para garantir uma mesa disponível.

US-05 – (Média prioridade – 2 pontos – Não implementado):

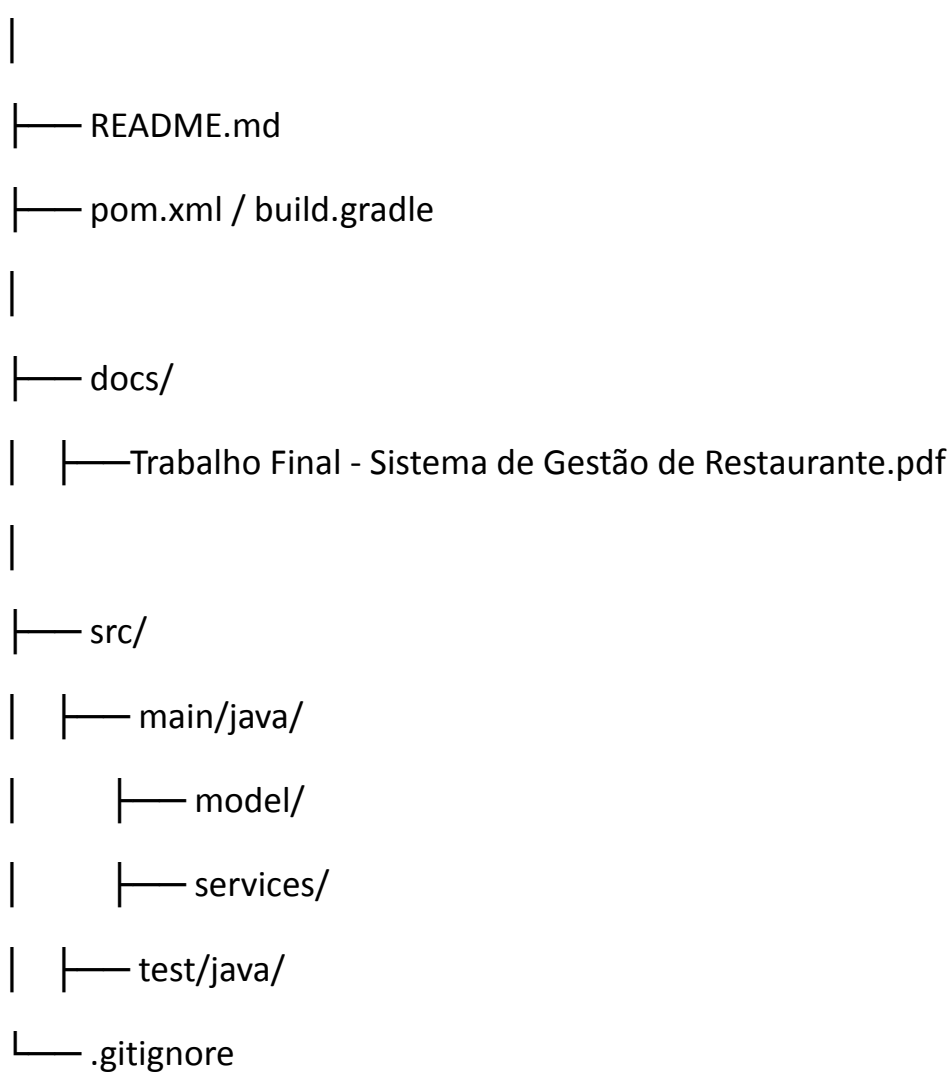
Como gerente, quero gerir o stock, para evitar falta de produtos.

Planeamento de Sprints

Sprint	Foco Principal	Entregáveis
Sprint 1	Análise e Design	Documentação de requisitos, diagramas UML iniciais
Sprint 2	Implementação	Classes Java principais e padrões de desenho
Sprint 3	Testes e Documentação	Testes JUnit, documentação final e apresentação

Estrutura do Repositório Git

project-root/



Repositório público - Restaurante_Sistema_Gestao

Boas Práticas de Versionamento

- Mensagens claras e objetivas
- Commits bem descritos
- Organização por funcionalidades

Documentação de Testes

Estratégia de Testes

Objetivo dos Testes

O objetivo dos testes é garantir que as funcionalidades implementadas do Sistema de Gestão de Restaurante funcionam corretamente, respeitam os requisitos definidos e não introduzem erros durante a execução do sistema.

Níveis de Teste Aplicados

Testes Unitários:

Testam classes e métodos individualmente, como Pedido, Fatura e Reserva.

Testes de Integração:

Verificam a interação entre classes, por exemplo entre Pedido, ItemMenu e Stock.

Testes Funcionais:

Confirmam que os requisitos funcionais são cumpridos, como a criação de pedidos e o cálculo da fatura.

Tipos de Teste Utilizados

- Testes de caixa preta (black-box)
- Testes de caixa branca (white-box)
- Testes de valores limite
- Testes de estados (state transition)

Ferramentas de Teste

- JUnit para testes automatizados
- Maven/Gradle para execução dos testes

Ambiente de Teste

Os testes são executados num ambiente local, utilizando a mesma estrutura de projeto definida no repositório Git, garantindo consistência entre desenvolvimento e testes.

Documentação dos Casos de Teste

TC-01 – Criar Pedido

- **Requisito:** RF-03
- **Tipo:** Caixa preta
- **Pré-condição:** Existe uma mesa registada
- **Resultado Esperado:** Pedido criado com sucesso

TC-02 – Criar Pedido sem Mesa

- **Requisito:** RF-03
- **Tipo:** Caixa preta
- **Resultado Esperado:** Sistema impede a criação do pedido

TC-03 – Adicionar Item Disponível ao Pedido

- **Requisito:** RF-04
- **Tipo:** Caixa preta
- **Pré-condição:** Item disponível no menu
- **Resultado Esperado:** Item adicionado corretamente

TC-04 – Adicionar Item Indisponível

- **Requisito:** RF-11
- **Tipo:** Caixa preta
- **Resultado Esperado:** Sistema impede a adição do item

TC-05 – Remover Item do Pedido

- **Requisito:** RF-04
- **Tipo:** Caixa preta
- **Resultado Esperado:** Item removido do pedido

TC-06 – Criar Pedido com Vários Itens

- **Requisito:** RF-04
- **Tipo:** Caixa preta
- **Resultado Esperado:** Pedido contém todos os itens adicionados

TC-07 – Estado Inicial do Pedido

- **Requisito:** RF-06
- **Tipo:** Caixa branca
- **Resultado Esperado:** Estado inicial é “Recebido”

TC-08 – Atualizar Estado para Em Preparação

- **Requisito:** RF-06
- **Tipo:** Caixa branca
- **Resultado Esperado:** Estado atualizado corretamente

TC-09 – Atualizar Estado para Pronto

- **Requisito:** RF-06

  Tipo: Caixa branca
- **Resultado Esperado:** Estado alterado para “Pronto”

TC-10 – Atualizar Estado Inválido

- **Requisito:** RF-06
- **Tipo:** Caixa branca
- **Resultado Esperado:** Sistema rejeita a transição inválida

TC-11 – Calcular Total da Fatura

- **Requisito:** RF-08
- **Tipo:** Caixa branca
- **Resultado Esperado:** Total calculado corretamente

TC-12 – Calcular Fatura sem Itens

- **Requisito:** RF-08
- **Tipo:** Caixa preta
- **Resultado Esperado:** Total igual a zero

TC-13 – Registar Pagamento com Sucesso

- **Requisito:** RF-09
- **Tipo:** Caixa preta
- **Resultado Esperado:** Pagamento registado

TC-14 – Registar Pagamento Duplicado

- **Requisito:** RF-09
- **Tipo:** Caixa preta
- **Resultado Esperado:** Sistema impede pagamento duplicado

TC-15 – Criar Reserva Válida

- **Requisito:** RF-01
- **Tipo:** Caixa preta
- **Resultado Esperado:** Reserva criada com sucesso

TC-16 – Criar Reserva sem Mesas Disponíveis

- **Requisito:** RF-02
- **Tipo:** Caixa preta
- **Resultado Esperado:** Reserva recusada

TC-17 – Cancelar Reserva

- **Requisito:** RF-01
- **Tipo:** Caixa preta
- **Resultado Esperado:** Reserva cancelada

TC-18 – Visualizar Pedidos na Cozinha

- **Requisito:** RF-07
- **Tipo:** Caixa preta
- **Resultado Esperado:** Lista de pedidos apresentada

TC-19 – Atualizar Stock após Venda

- **Requisito:** RF-10
- **Tipo:** Caixa branca
- **Resultado Esperado:** Quantidade em stock reduzida

TC-20 – Stock Insuficiente

- **Requisito:** RF-11
- **Tipo:** Caixa preta
- **Resultado Esperado:** Pedido não permitido

TC-21 – Adicionar Novo Item ao Menu

- **Requisito:** RF-11
- **Tipo:** Caixa preta
- **Resultado Esperado:** Item adicionado ao menu

TC-22 – Remover Item do Menu

- **Requisito:** RF-11
- **Tipo:** Caixa preta
- **Resultado Esperado:** Item removido

TC-23 – Acesso Não Autorizado

- **Requisito:** RNF-04
- **Tipo:** Caixa preta
- **Resultado Esperado:** Acesso negado

TC-24 – Tempo de Resposta

- **Requisito:** RNF-01
- **Tipo:** Caixa preta
- **Resultado Esperado:** Resposta inferior a 2 segundos

TC-25 – Criar Pedido Simultâneo

- **Requisito:** RNF-05
- **Tipo:** Caixa preta
- **Resultado Esperado:** Sistema suporta múltiplos pedidos

TC-26 – Integridade dos Dados

- **Requisito:** RNF-03
- **Tipo:** Caixa branca
- **Resultado Esperado:** Dados mantidos após erro

TC-27 – Quantidade Limite de Itens

- **Requisito:** RF-04
- **Tipo:** Valores limite
- **Resultado Esperado:** Sistema aceita limite máximo

TC-28 – Quantidade Acima do Limite

- **Requisito:** RF-04
- **Tipo:** Valores limite

  Resultado Esperado: Sistema rejeita valor inválido

TC-29 – Sequência Completa de Estados

- **Requisito:** RF-06
- **Tipo:** Teste de estados
- **Resultado Esperado:** Estados seguem a ordem correta

TC-30 – Pedido Finalizado

- **Requisito:** RF-06
- **Tipo:** Caixa branca
- **Resultado Esperado:** Pedido marcado como concluído

Conclusão

O desenvolvimento do Sistema de Gestão de Restaurante permitiu aplicar, de forma integrada, os principais conceitos abordados na unidade curricular de Engenharia de Software. Ao longo do projeto foi possível realizar uma análise completa do problema, identificar corretamente os stakeholders, definir requisitos funcionais e não funcionais, modelar o sistema através de diagramas UML e implementar parte das funcionalidades em Java, acompanhadas por testes automatizados.

O sistema concebido responde às necessidades essenciais de um restaurante, nomeadamente a gestão de pedidos, o acompanhamento do estado dos pedidos pela cozinha, a faturação e o controlo de stock. A utilização de uma arquitetura em camadas, aliada à aplicação de padrões de desenho como State, Factory e Repository, contribuiu para uma solução modular, extensível e de fácil manutenção. A aplicação dos princípios SOLID reforçou a qualidade do design, promovendo baixo acoplamento e elevada coesão entre os componentes.

Apesar de nem todas as funcionalidades planeadas terem sido totalmente implementadas, como a gestão de reservas e o controlo completo de stock, o sistema apresenta uma base sólida e bem estruturada que permite a sua evolução futura. As funcionalidades implementadas cumprem os requisitos definidos e foram validadas através de uma estratégia de testes abrangente, garantindo a correção do comportamento do sistema e a fiabilidade dos dados.

Em suma, este projeto demonstrou a importância de uma abordagem estruturada no desenvolvimento de software, desde a análise e design até à implementação e testes. Para trabalhos futuros, seria possível expandir o sistema com uma interface gráfica, integração com bases de dados reais, suporte direto ao cliente e relatórios mais avançados de gestão, aproximando ainda mais a solução de um cenário real de utilização.