|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | | | |  | | | |
|  | **escola**  **superior**  **de Tecnologia**  **e gestão**  POLITÉCNICO  DO PORTO | | | | |  | | | |
|  | | | M |  |  | | | | |
| **MESTRADO**  Engenharia Informática | | |  | | |
| **Proposta de Métodos e Técnicas de Suporte ao Desenvolvimento de Software (MTSDS)** | | | José Filipe Ferreira Soares nº 8240047  FoodStream: Repositório de Refeições segundo paradigma Microserviços  Grupo 7:  Luís Henrique dos Santos Soares, nº 8210396Célia Cristina Pereira de Paiva nº 8220954José Filipe Ferreira Soares nº 8240047  André Filipe Nunes de Matos, nº 8240634  **Grupo 7 – Célia Paiva8220954, André Matos8240634, José Soares8240047, Luís Soares8210396**  **2024 / 2025** | | | | |  | |
|
|
|  | | | | | | | | |
|

# **Resumo**

Inclui um resumo do propósito do projeto, os principais desafios e as soluções propostas.

**Abstract**

# **Lista de Figuras**

[Figura 1 – Arquitetura monolítica atual (Mudar desenho importante) 11](#_Toc186297812)

[Figura 2 – Processo de negócio – FoodStream – monolítico 12](#_Toc186297813)

[Figura 3 – Estrutura de microsserviços proposta 12](#_Toc186297814)

[Figura 4 – Processo de negócio – ‘gestao\_utilizadores’ - ALTERAR 46](#_Toc186297815)

[Figura 5 – Processo de negócio – ‘criacao\_menu’ - ALTERAR 47](#_Toc186297816)

[Figura 6 – Processo de negócio – ‘verifica\_stock’ – colocar diagrama BPMN 47](#_Toc186297817)

[Figura 7 – Processo de negócio – ‘repositorio\_entregas’ - colocar diagrama BPMN 47](#_Toc186297818)

[Figura 8 – Processo de negócio – ‘calculo\_rotas’ - colocar diagrama BPMN 47](#_Toc186297819)

[Figura 5 – Casos de uso – ‘gestao\_utilizadores’ – Alterar por motivo notificacoes 49](#_Toc186297820)

[Figura 6 – Domain-Driven Design 55](#_Toc186297821)

[Figura 7 – Modelo de dados ‘gestao\_utilizadores’ 58](#_Toc186297822)

[Figura 8 – Modelo de dados ‘verifica\_stock’ 60](#_Toc186297823)

# **Lista de Tabelas**

[Tabela 1 – Microserviços (MS) e componentes de suporte 14](#_Toc186297824)

[Tabela 2 – Requisitos funcionais – microserviço ‘gestao\_utilizadores’ 34](#_Toc186297825)

[Tabela 3 – Requisito Funcional – Criar Utilizador 35](#_Toc186297826)

[Tabela 4 – Requisito Funcional – Autenticação de Utilizador (Login) 35](#_Toc186297827)

[Tabela 5 – Requisito Funcional – Editar Permissões 36](#_Toc186297828)

[Tabela 6 – Requisito Funcional – Listar Utilizadores 36](#_Toc186297829)

[Tabela 7 – Requisito Funcional – Registar Status de Login 37](#_Toc186297830)

[Tabela 8 – Requisito Funcional – Atualizar Perfil do Utilizador 37](#_Toc186297831)

[Tabela 9 – Requisito Funcional – Enviar Notificações 38](#_Toc186297832)

[Tabela 10 – Requisito Funcional – Validar Token JWT 38](#_Toc186297833)

[Tabela 11 – Requisito Funcional – Excluir Utilizador 39](#_Toc186297834)

[Tabela 12 – Requisito Funcional – Recuperar Informações do Utilizador 39](#_Toc186297835)

[Tabela 14 – Requisito Funcional – CRUD de menus 40](#_Toc186297836)

[Tabela 15 – Requisito Funcional – Listagem por intervalos de tempo 40](#_Toc186297837)

[Tabela 16 – Requisito Funcional – Listar ingredientes e nível de stock 41](#_Toc186297838)

[Tabela 17 – Requisito Funcional – Adicionar Ingredientes ao Stock 41](#_Toc186297839)

[Tabela 18 – Requisito Funcional – Obter Alertas de Reabastecimento 42](#_Toc186297840)

[Tabela 19 – Requisito Funcional – Atualizar Stock Após Consumo 42](#_Toc186297841)

[Tabela 20 – Requisito Funcional – Validar Stock Antes da Criação de Menu 43](#_Toc186297842)

[Tabela 21 – Requisito Funcional – Gerar Relatório de Stock 43](#_Toc186297843)

[Tabela 22 – Requisito Funcional – Listar 44](#_Toc186297844)

[Tabela 23 – Requisito Funcional – Adicio 44](#_Toc186297845)

[Tabela 27 – Requisito Funcional – CRUD de funcionários 45](#_Toc186297846)

[Tabela 28 – Atores do Sistema 48](#_Toc186297847)

[Tabela 29 – Componente de suporte – /eureka-server 63](#_Toc186297848)

[Tabela 29 – Rotas – Serviços subjacentes 64](#_Toc186297849)

[Tabela 31 – Base de Dados dos Microserviços, acessos 66](#_Toc186297850)

**Índice**

[Resumo 2](#_Toc186833123)

[Abstract 3](#_Toc186833124)

[Lista de Figuras 4](#_Toc186833125)

[Lista de Tabelas 5](#_Toc186833126)

[Índice 6](#_Toc186833127)

[1 Introdução ao Projeto 10](#_Toc186833128)

[1.1. Contextualização 10](#_Toc186833129)

[1.2. Trabalho Proposto 11](#_Toc186833130)

[1.2.1 Estrutura Monolítica Atual 11](#_Toc186833131)

[1.2.2 Processo de negócio – monolítico 11](#_Toc186833132)

[1.2.3 Estrutura Proposta de Microserviços 12](#_Toc186833133)

[1.3. Objetivos 14](#_Toc186833134)

[2 Ferramentas, Paradigmas e Tecnologias de Desenvolvimento 16](#_Toc186833135)

[2.1. Ferramentas de Desenvolvimento e IDE (Integrated Development Environment) 17](#_Toc186833136)

[2.2. Paradigma de Arquitetura e Design 18](#_Toc186833137)

[2.3. Linguagem e Ambiente de Execução 21](#_Toc186833138)

[2.4. Ferramentas e Frameworks de Implementação e Construção 22](#_Toc186833139)

[2.5. Ferramentas de Containerização e Orquestração 24](#_Toc186833140)

[2.6. Serviços de Descoberta e Comunicação 26](#_Toc186833141)

[2.7. Gateway e Segurança 29](#_Toc186833142)

[2.8. Ferramentas de Testes e Interação com APIs 30](#_Toc186833143)

[2.9. Bases de Dados 32](#_Toc186833144)

[3 Requisitos Funcionais 34](#_Toc186833145)

[3.1. Microserviço – gestao\_utilizadores 34](#_Toc186833146)

[RF1 – Criar Utilizador 35](#_Toc186833147)

[RF2 – Autenticação de Utilizador (Login) 35](#_Toc186833148)

[RF3 – Editar Permissões 36](#_Toc186833149)

[RF4 – Listar Utilizadores 36](#_Toc186833150)

[RF5 – Registar Status de Login 37](#_Toc186833151)

[RF6 – Atualizar Perfil do Utilizador 37](#_Toc186833152)

[RF7 – Enviar Notificações 38](#_Toc186833153)

[RF8 – Validar Token JWT 38](#_Toc186833154)

[RF9 – Excluir Utilizador 39](#_Toc186833155)

[RF10 – Recuperar Informações do Utilizador 39](#_Toc186833156)

[3.2. Microserviço – criacao\_menu 40](#_Toc186833157)

[3.3. Microserviço – verifica\_stock 41](#_Toc186833158)

[RFX – Listar Ingredientes e nível de stock 41](#_Toc186833159)

[RFX – Adicionar Ingredientes ao Stock 41](#_Toc186833160)

[RFX – Obter Alertas de Reabastecimento 42](#_Toc186833161)

[RFX – Atualizar Stock Após Consumo 42](#_Toc186833162)

[RFX – Validar Stock Antes da Criação de Menu 43](#_Toc186833163)

[RFX – Gerar Relatório de Stock 43](#_Toc186833164)

[3.4. Microserviço – repositorio\_entregas 44](#_Toc186833165)

[3.5. Microserviço – calculo\_rotas 45](#_Toc186833166)

[4 Processos de Negócio 46](#_Toc186833167)

[4.1. Processo – ‘gestao\_utilizadores’ 46](#_Toc186833168)

[4.2. Processo – ‘criacao\_menu’ 47](#_Toc186833169)

[4.3. Processo – ‘verifica\_stock’ 47](#_Toc186833170)

[4.4. Processo – ‘repositorio\_entregas’ 47](#_Toc186833171)

[4.5. Processo – ‘calculo\_rotas’ 47](#_Toc186833172)

[5 Casos de Uso 48](#_Toc186833173)

[5.1 Atores do Sistema 48](#_Toc186833174)

[5.2 Casos de Uso – gestao\_utilizadores 48](#_Toc186833175)

[5.3 Casos de Uso – criacao\_menu 53](#_Toc186833176)

[5.4 Casos de Uso – verifica\_stock 53](#_Toc186833177)

[5.2.1 Caso de Uso 1 – Listar Ingredientes e Respetivos Níveis de Stock 53](#_Toc186833178)

[5.2.2 Caso de Uso 2 – Adicionar Ingredientes ao Stock 54](#_Toc186833179)

[5.2.3 Caso de Uso 3 – Obter Alertas de Reabastecimento 55](#_Toc186833180)

[5.2.4 Caso de Uso 4 – Atualizar Stock Após Consumo 55](#_Toc186833181)

[5.2.5 Caso de Uso 5 – Validar Stock Antes da Criação de Menu 56](#_Toc186833182)

[5.2.6 Caso de Uso 6 – Gerar Relatório de Stock 57](#_Toc186833183)

[5.2.7 Diagrama 57](#_Toc186833184)

[5.5 Casos de Uso – repositorio\_entregas 58](#_Toc186833185)

[5.6 Casos de Uso – calculo\_rotas 58](#_Toc186833186)

[6 Domain-Driven Design 59](#_Toc186833187)

[6.1. gestao\_utilizadores 59](#_Toc186833188)

[6.2. criacao\_menu 61](#_Toc186833189)

[6.3. verifica\_stock 61](#_Toc186833190)

[6.4. repositorio\_entregas 61](#_Toc186833191)

[6.5. calculo\_rotas 61](#_Toc186833192)

[7 Modelos de Dados 62](#_Toc186833193)

[7.1. gestao\_utilizadores 62](#_Toc186833194)

[7.2. criacao\_menu 63](#_Toc186833195)

[7.3. verifica\_stock 64](#_Toc186833196)

[7.4. repositorio\_entregas 65](#_Toc186833197)

[7.5. calculo\_rotas 65](#_Toc186833198)

[8 Arquitetura do Sistema 66](#_Toc186833199)

[9 Componentes de Suporte 67](#_Toc186833200)

[9.1. /eureka-server (Service Discovery) 67](#_Toc186833201)

[9.2. /api-gateway (Spring Cloud Gateway) 68](#_Toc186833202)

[9.2.1 Rotas 68](#_Toc186833203)

[9.2.2 RestTemplateConfig 68](#_Toc186833204)

[9.3. RabbitMQ (Message Broker) 69](#_Toc186833205)

[10 Microserviços 70](#_Toc186833206)

[10.1. Microserviço – gestao\_utilizadores 70](#_Toc186833207)

[Descrição 70](#_Toc186833208)

[Endpoints (API REST) 70](#_Toc186833209)

[Tecnologias e Ferramentas Utilizadas 71](#_Toc186833210)

[Package models 71](#_Toc186833211)

[Package repositories 71](#_Toc186833212)

[Package dto (data transfer object) 72](#_Toc186833213)

[Package services (aplicação) 73](#_Toc186833214)

[Package config (configurações) 74](#_Toc186833215)

[Mensagens com RabbitMQ 74](#_Toc186833216)

[Integração com o Gateway 74](#_Toc186833217)

[Testes 74](#_Toc186833218)

[10.2. criacao\_menu 75](#_Toc186833219)

[Descrição 75](#_Toc186833220)

[Endpoints (API REST) 75](#_Toc186833221)

[Mensagens com RabbitMQ 75](#_Toc186833222)

[Integração com o Gateway 75](#_Toc186833223)

[Testes 75](#_Toc186833224)

[10.3. verifica\_stock 76](#_Toc186833225)

[Descrição 76](#_Toc186833226)

[Endpoints (API REST) 76](#_Toc186833227)

[Mensagens com RabbitMQ 76](#_Toc186833228)

[Integração com o Gateway 76](#_Toc186833229)

[Testes 76](#_Toc186833230)

[10.4. repositorio\_entregas 77](#_Toc186833231)

[Descrição 77](#_Toc186833232)

[Endpoints (API REST) 77](#_Toc186833233)

[Mensagens com RabbitMQ 77](#_Toc186833234)

[Integração com o Gateway 77](#_Toc186833235)

[Testes 77](#_Toc186833236)

[10.5. calculo\_rotas 78](#_Toc186833237)

[Descrição 78](#_Toc186833238)

[Endpoints (API REST) 78](#_Toc186833239)

[Mensagens com RabbitMQ 78](#_Toc186833240)

[Integração com o Gateway 78](#_Toc186833241)

[Testes 78](#_Toc186833242)

[11 Contentorização e Orquestração 79](#_Toc186833243)

[10.1. Docker – Criação de Imagens 79](#_Toc186833244)

[10.2. Kubernetes (minikube – K8s) – Configuração de Clusters 79](#_Toc186833245)

[12 Conclusão 81](#_Toc186833246)

[Referências 82](#_Toc186833247)

# **Introdução** **ao Projeto**

Este projeto insere-se no contexto académico da unidade curricular de Métodos e Técnicas de Suporte ao Desenvolvimento de Software, do Mestrado em Engenharia Informática (MEI) da Escola Superior de Tecnologia e Gestão (ESTG). Será apresentado pelo Grupo 7, composto pelos seguintes membros:

**Grupo 7:**

* Luís Henrique dos Santos Soares, nº 8210396
* Célia Cristina Pereira de Paiva, nº 8220954
* José Filipe Ferreira Soares, nº 8240047
* André Filipe Nunes de Matos, nº 8240634

Por motivos de acesso de alguns membros do grupo e com permissão dos docentes o nosso projeto está alocado no repositório GitHub no endereço:

<https://github.com/luhesas0/mei_mtsds/tree/master> .

**FoodStream: Distribuição de Refeições segundo paradigma Microserviços**, visa conceber uma aplicação suportada por uma arquitetura técnica baseada no paradigma de microserviços, implementando práticas e padrões modernos associados a este modelo arquitetural.

## **Contextualização**

O FoodStream, uma empresa fictícia, enfrenta desafios operacionais relacionados com a gestão de distribuição de refeições para escolas, centros sociais e associações. Atualmente, a solução existente baseia-se numa arquitetura monolítica, que apresenta limitações consideráveis em termos de:

* **Escalabilidade**: Incapacidade de suportar aumento de carga de forma eficiente.
* **Manutenção:** Alterações tornam-se complexas e propensas a erros.
* **Integração de Novas Funcionalidades:** Adicionar novos módulos ou expandir os existentes implica refatorações significativas.

Este projeto propõe uma abordagem escalável e modular, utilizando uma arquitetura baseada em **microserviços** para resolver estas limitações, promovendo uma evolução sustentável.

## **Trabalho Proposto**

### Estrutura Monolítica Atual

Na arquitetura atual, todas as funcionalidades estão integradas num único componente. Este modelo resulta nas seguintes limitações:

* **Escalabilidade limitada:** Um aumento na carga impacta negativamente o desempenho global.
* **Manutenção complexa:** Alterações numa funcionalidade podem introduzir efeitos colaterais noutras áreas.
* **Integração difícil:** Adicionar novas funcionalidades implica alterações significativas no sistema.

Uma imagem com texto, diagrama, file, Desenho técnico

Descrição gerada automaticamente

Figura 1 – Arquitetura monolítica atual (Mudar desenho importante)

### **Processo de negócio – monolítico**

O processo monolítico do FoodStream centraliza todas as funcionalidades em um único sistema. Em BPMN (Business Process Model and Notation), o fluxo de trabalho apresenta etapas sequenciais como criação de menus, verificação de stock e distribuição de ordens de trabalho. Este modelo simplifica a comunicação interna, mas apresenta limitações em termos de escalabilidade e flexibilidade para alterações futuras.

Uma imagem com captura de ecrã, file, diagrama, texto

Descrição gerada automaticamente

Figura 2 – Processo de negócio – FoodStream – monolítico

### **Estrutura Proposta de Microserviços**

Diagrama da arquitetura de microserviços, detalhando os módulos principais.

Apresente o diagrama de microserviços com suas responsabilidades.

A nova arquitetura propõe a divisão do sistema em **microsserviços independentes**, promovendo flexibilidade, escalabilidade e uma evolução modular.

Uma imagem com texto, diagrama, Esquema, Desenho técnico

Descrição gerada automaticamente

Figura 3 – Estrutura de microsserviços proposta

Inicialmente foi decidido implementar o microserviço ‘gestao\_funcionarios’ mas ao analisarmos a abordagem DDD foi verificado que faria mais sentido desenvolver um microserviço de gestao\_utilizadores, autorizados e autenticados. Logo, com o desenvolvimento da implementação desta arquitetura apresentaremos a arquitetura de microserviços final no ponto nº 8.~

Esta arquitetura está dividida em cinco microserviços: **menu**, **stock**, **repositorio\_entregas**, **calculo\_rotas** e **funcionarios.**

Na tabela abaixo, podemos observar em detalhe a função de cada um:

|  |  |
| --- | --- |
| **Microserviços** | **Detalhe** |
| Criação de Menu | Este microserviço trata da criação, listagem, atualização e remoção de menus, conforme um determinado intervalo de tempo seja diário ou semanal, para que um cliente possa escolher um determinado número de refeições dados os menus. |
| Verifica Stock | Este microserviço, perante a exigência de criação dos menus, este gere toda a logística de ingredientes necessários para as receitas. Controla, então, a quantidade de ingredientes e alerta se for necessário encomendar ingredientes (stock baixo). |
| Repositório de Entregas | Este microserviço será um repositório onde as ordens de trabalho irão aparecer e, dependendo da capacidade do seu veículo, um funcionário pode subscrever uma ordem e realizá-la. |
| Cálculo de rotas | Este microserviço, consoante a ordem de trabalho subscrita pelo funcionário, irá calcular a rota de entrega para uma ou mais instituições. |
| Gestão Funcionário | Este microserviço, será responsável pela criação de perfis aos funcionários, atribuição de veículos e registo de entregas feitas pelos funcionários, para futuras avaliações de desempenho |

Para além destes microserviços, outros módulos serão implementados como o **api-gateway, docker, k8s(minikube)** e o **eureka-server.**

## **Objetivos**

A proposta contempla a implementação dos seguintes microsserviços e componentes de suporte: Estrutura de Microserviços proposta, apresenta-se a estrutura da arquitetura de microserviços do FoodStream e os seus diferentes microserviços:

Arquitetura Hexagonal para todos os microserviços (Web api, business Logic e base de dados independente). Domain-Driven Design Estratégico.

Tabela 1 – Microserviços (MS) e componentes de suporte

|  |  |
| --- | --- |
| **Microserviços (MS) e Componentes a Implementar** | **Detalhe** |
| **gestao\_utilizadores** | Este microserviço, será responsável pela criação de perfis de todos os atores, inclusive autenticação e autorização (Login). Editar permissões (role), Listar utilizadores, Registar Status de Login, Atualizar perfil do utilizador.  enviar notificações para funcionário e enviar e receber notificações do cliente, Validar Token JWT, Excluir utilizador, recuperar informações do utilizador, os endpoints para notificações utilizam RabbitMQ configurado no próprio microserviço ‘gestao\_utilizadores’ tal como a avaliação do cliente sobre a entrega do funcionários.  O cliente recebe link para avaliação da entrega, faz a avaliação por níveis de 1=mau até 5=muito bom e envia resposta notificação para o seu perfil cliente e com permissão leitura para administrador e funcionário. |
| **criacao\_menu** | Este microserviço trata da criação, listagem, atualização e remoção de menus, conforme um determinado intervalo de tempo seja diário ou semanal, para que um cliente possa escolher um determinado número de refeições dados os menus. |
| **verifica\_stock** | Este microserviço, perante a exigência de criação dos menus, este gere toda a logística de ingredientes necessários para as receitas. Controla, então, a quantidade de ingredientes e alerta se for necessário encomendar ingredientes (stock baixo). |
| **repositorio\_entregas** | Este microserviço será um repositório onde as ordens de trabalho irão aparecer e, dependendo da capacidade do seu veículo, um funcionário pode subscrever uma ordem e realizá-la. |
| **calculo\_rotas** | Este microserviço, consoante a ordem de trabalho subscrita pelo funcionário, irá calcular a rota de entrega para uma ou mais instituições. |
| **api-gateway** | Gestão de segurança e tráfego entre utilizador e microsserviços. |
| **eureka-server** | Facilitar o registo e descoberta dinâmica dos serviços. |
| **K8s (minikube)** |  |
| **Comunicação de Mensagens (RabbitMQ)** | Comunicação assíncrona e eventos entre microsserviços. |
| **Documentação (Swagger)** | Ferramenta de documentação interativa para os endpoints dos serviços. |

# **Ferramentas, Paradigmas e Tecnologias de Desenvolvimento**

O desenvolvimento deste projeto baseia-se numa arquitetura moderna de microserviços, suportada por um conjunto de ferramentas, paradigmas e tecnologias que garantem escalabilidade, modularidade e eficiência. Este capítulo apresenta as principais soluções utilizadas, agrupadas por categorias:

1. **Ferramentas de Desenvolvimento e IDEs:** O *IntelliJ IDEA*, um ambiente integrado para desenvolvimento em Java, facilita a codificação e depuração. O *GitLab* (gitlab.estg.ipp.pt) é utilizado para gestão de versões, integração contínua e colaboração. Por motivos de acesso de alguns membros do grupo e com permissão dos docentes o nosso projeto está hospedado no GitHub em: <https://github.com/luhesas0/mei_mtsds/tree/master> .
2. **Paradigmas de Arquitetura e Design:** *Domain-Driven Design (DDD)* organiza o domínio em contextos específicos. A arquitetura de *microserviços* promove independência entre serviços. *REST* (Representational State Transfer) e *APIs RESTful* garantem comunicação eficiente e modularidade. O paradigma *CRUD* (Create, Read, Update, Delete) define as operações fundamentais sobre recursos.
3. **Linguagem e Ambiente de Execução:** A linguagem *Java* é utilizada pela sua robustez, portabilidade e integração com frameworks modernos.
4. **Frameworks de Implementação e Construção:** *Spring Boot*, *Maven, Lombok* aceleram o desenvolvimento e garantem consistência.
5. **Containerização e Orquestração:** *Docker, Kubernetes (K8s)* e *Minikube* permitem a gestão eficiente de serviços e ambientes.
6. **Serviços de Descoberta e Comunicação:** *Eureka, RabbitMQ, HTTP* e *AMQP* promovem conectividade robusta e comunicação assíncrona.
7. **Gateway e Segurança:** O *Spring Cloud Gateway* assegura roteamento eficiente e proteção de APIs.
8. **Ferramentas de Teste e Interação:** *Postman*, *JUnit* e *REST Assured* garantem qualidade e validação dos serviços.
9. **Bases de Dados:** *PostgreSQL* e *MySQL* fornecem armazenamento relacional escalável e confiável.

Estes elementos suportam a criação de uma arquitetura orientada a microserviços, detalhada nos tópicos seguintes.

## **Ferramentas de Desenvolvimento e IDE *(Integrated Development Environment)***

* **IntelliJ IDEA**

O IntelliJ IDEA é um Ambiente de Desenvolvimento Integrado (*Integrated Development Environment*, IDE) desenvolvido pela JetBrains, focado no desenvolvimento de software em linguagens baseadas na JVM (*Java Virtual Machine*), como Java, Kotlin e Groovy. Este IDE está disponível em duas edições: a *Community Edition*, licenciada sob a Apache 2.0, que é gratuita e de código aberto, e a *Ultimate Edition*, uma versão comercial com funcionalidades avançadas [1].

Lançado em 2001, o IntelliJ IDEA destacou-se por introduzir funcionalidades pioneiras de navegação e refatoração de código, estabelecendo um padrão para IDEs modernos. Em 2009, a JetBrains tornou o código-fonte do IntelliJ IDEA público, permitindo a criação de uma comunidade ativa em torno da versão aberta do projeto [1].

As funcionalidades principais incluem assistência inteligente ao código, navegação eficiente, suporte para integração contínua (*continuous integration*), ferramentas de controlo de versão como Git, e integração com tecnologias amplamente utilizadas em arquiteturas modernas, como *Docker* e *Kubernetes*. Além disso, a versão comercial oferece suporte adicional para frameworks empresariais e linguagens não baseadas na JVM, como Python e PHP [1].

O IntelliJ IDEA é um IDE amplamente utilizado que oferece ferramentas avançadas para aumentar a produtividade no desenvolvimento de software, sendo essencial para arquiteturas orientadas a microserviços.

* **Git (Gitlab.estg.ipp.pt)**

O GitLab é uma ferramenta de *DevOps* desenvolvida pela GitLab Inc., que integra um conjunto completo de funcionalidades para desenvolvimento, segurança e operações de software. Baseado no controlo de versões distribuído Git, o GitLab oferece recursos como gestão de acesso, rastreamento de erros, gestão de tarefas *(task management)*, wikis e snippets *(fragmentos de código)* para projetos [2].

A plataforma foi criada em 2011 por Dmytro Zaporozhets e Sytse Sijbrandij, tendo evoluído para uma solução robusta amplamente adotada em todo o mundo. A empresa opera num modelo *freemium*, com uma versão comunitária de código aberto (licença MIT) e uma edição comercial com funcionalidades avançadas [2]. O GitLab suporta todo o ciclo de vida do *software*, integrando desenvolvimento, integração contínua (*CI/CD*), controlo de versões, monitorização de segurança e *deployment* em arquiteturas modernas como Kubernetes.

Reconhecido como uma das maiores empresas totalmente remotas, o GitLab promove a colaboração eficiente entre equipas distribuídas. Além disso, contribui para a implementação de arquiteturas orientadas a microserviços, permitindo um desenvolvimento ágil e modular [2].

O GitLab é uma ferramenta essencial para gestão de projetos e desenvolvimento contínuo em ambientes orientados a microserviços, facilitando a integração e o *deployment* de aplicações.

## **Paradigma de Arquitetura e Design**

* **Domain-Driven Design**

O Domain-Driven Design (DDD) é uma abordagem de design e desenvolvimento de software que foca no domínio do negócio como centro das decisões arquiteturais e de design [3]. Introduzido por Eric Evans em 2003, o DDD combina técnicas de design estratégico e tático, proporcionando ferramentas para lidar com a complexidade dos sistemas, ao dividir problemas em contextos bem definidos *(Bounded Contexts)* e promover uma linguagem ubíqua *(Ubiquitous Language)*, que alinha especialistas técnicos e de negócios [3].

O DDD está estruturado em dois pilares principais:

1. **Design Estratégico:** Agrupa padrões e princípios que ajudam a definir limites claros entre os contextos do domínio e os seus relacionamentos. Ferramentas como *Context Maps* e *Bounded Contexts* facilitam a modularização do sistema, promovendo escalabilidade e flexibilidade.
2. **Design Tático:** Foca em padrões de nível médio e baixo para implementar o domínio em código. Elementos como *Entities* (Entidades), *Value Objects* (Objetos de Valor) e *Aggregates* (Agregados) refinam a lógica de negócio e garantem a consistência das operações [3].

Embora seja ideal para sistemas complexos, o DDD pode não ser necessário em domínios simples devido ao esforço requerido para implementar suas práticas.

O Domain-Driven Design organiza sistemas complexos ao alinhar modelagem de domínio e desenvolvimento técnico, sendo uma abordagem eficaz para arquiteturas orientadas a microserviços.

* **Microserviços**

A arquitetura de microserviços é um padrão arquitetural que organiza uma aplicação como uma coleção de serviços fracamente acoplados (*loosely coupled*), com granularidade reduzida, que se comunicam através de protocolos leves, como HTTP ou *Message Queues* (Filas de Mensagens) [4]. Cada serviço é projetado para implementar uma funcionalidade específica do negócio, permitindo que equipas de desenvolvimento os implementem e atualizem de forma independente.

Este tipo de arquitetura reduz a interdependência do código, melhora a modularidade e facilita a escalabilidade, pois serviços individuais podem ser dimensionados conforme necessário, sem impactar a aplicação como um todo [4]. No entanto, a sua adoção requer uma estrutura inicial mais complexa e a manutenção de interfaces claras (*APIs*) para a comunicação entre serviços. Além disso, os microserviços são frequentemente utilizados em sistemas nativos para a *cloud* (*cloud-native*), suportando ambientes distribuídos e soluções de *containerization* como Kubernetes [4].

Em comparação com arquiteturas monolíticas, os microserviços oferecem maior flexibilidade na integração de tecnologias heterogéneas, mas apresentam desafios como latência, teste de sistemas distribuídos e a necessidade de ferramentas avançadas de monitorização [4].

A arquitetura de microserviços organiza aplicações como coleções de serviços independentes, suportando escalabilidade, modularidade e implementações rápidas, sendo ideal para a criação da *Plataforma E-Commerce EducaMarket orientada a microserviços*.

* **REST (Representational State Transfer)**

O REST (Representational State Transfer) é um estilo arquitetural que define padrões para a comunicação entre sistemas na web. Ele facilita a interoperabilidade ao promover a separação de responsabilidades entre utilizador e servidor. A sua principal característica é o *statelessness* (ausência de estado), que permite que utilizador e servidor processem mensagens independentemente de interações anteriores [5].

A comunicação no REST utiliza o protocolo HTTP e baseia-se em operações sobre *resources* (recursos), que representam objetos ou dados acessíveis. Estas operações são mapeadas para os verbos HTTP básicos: **GET** (recuperar recursos), **POST** (criar novos recursos), **PUT** (atualizar recursos existentes) e **DELETE** (remover recursos). Os utilizadors enviam *requests* (pedidos) contendo um *header* (cabeçalho) e, opcionalmente, um corpo *(body)*, enquanto os servidores respondem com *status codes* (códigos de status) que indicam o sucesso ou erro da operação [5].

A arquitetura REST é amplamente utilizada em APIs *(Application Programming Interfaces)* devido à sua simplicidade, escalabilidade e capacidade de se adaptar a diferentes utilizadors, incluindo aplicações web, móveis e sistemas distribuídos. Além disso, REST suporta a troca de dados em formatos padronizados, como JSON ou XML, facilitando a integração entre sistemas heterogéneos.

REST define uma abordagem eficiente e modular para comunicação em arquiteturas orientadas a microserviços, sendo fundamental para a criação de APIs flexíveis e escaláveis.

* **API RESTful**

A API RESTful é um padrão arquitetural baseado na ***Representational State Transfer* (Transferência de Estado Representacional).** Ela define as regras para troca de informações entre sistemas de forma eficiente, segura e confiável. Este estilo é amplamente utilizado para criar serviços web que suportam comunicação entre utilizadors e servidores. APIs RESTful permitem a separação entre o utilizador e o servidor, promovendo flexibilidade e escalabilidade nas interações [6].

APIs RESTful seguem princípios fundamentais, como ausência de estado *(statelessness)*, interface uniforme e capacidade de armazenamento em cache *(caching)*. A ausência de estado garante que cada requisição seja processada de forma independente, sem necessidade de contexto prévio. A interface uniforme possibilita que diferentes aplicações utilizem os mesmos recursos *(resources)* com padrões estabelecidos para identificar, criar, modificar ou apagar dados. Estas interações são realizadas através de métodos HTTP, como **GET**, **POST**, **PUT** e **DELETE** [6]**.**

As APIs RESTful oferecem flexibilidade, permitindo que aplicações utilizador e servidor evoluam de forma independente, e independência tecnológica, permitindo a integração entre sistemas desenvolvidos em diferentes linguagens ou plataformas. Além disso, o suporte ao cache melhora o desempenho e reduz a carga do servidor.

APIs RESTful são indispensáveis para arquiteturas modernas orientadas a microserviços, promovendo comunicação eficiente e adaptabilidade.

* **CRUD (Create, Read, Update, Delete)**

O paradigma **CRUD (Create, Read, Update, Delete)** define as quatro operações fundamentais para gerir recursos em sistemas de armazenamento persistente. Estas operações são essenciais no desenvolvimento de APIs e bases de dados, permitindo a criação, leitura, atualização e eliminação de dados de forma consistente e eficiente. No contexto de sistemas web, o CRUD é frequentemente mapeado para os métodos HTTP: **POST** (Criar), **GET** (Ler), **PUT** (Atualizar) e **DELETE** (Apagar) [7].

No caso de uma plataforma de e-commerce, como a gestão de produtos, o CRUD permite:

* **Criar (Create):** adicionar novos produtos ao catálogo, como nome, descrição, preço e quantidade em stock, através de uma solicitação **POST**.
* **Ler (Read):** listar os produtos disponíveis ou obter detalhes de um produto específico com uma solicitação **GET**.
* **Atualizar (Update):** modificar informações de produtos existentes, como alterar o preço ou atualizar a quantidade em stock, utilizando **PUT**.
* **Apagar (Delete):** remover produtos descontinuados do catálogo por meio de uma solicitação **DELETE**.

Estas operações garantem que os recursos permaneçam organizados e acessíveis. No contexto de APIs RESTful, os códigos de resposta HTTP como **200 (OK)**, **201 (CREATED)** ou **404 (NOT FOUND)** ajudam a monitorizar o estado das operações.

A implementação do CRUD em arquiteturas orientadas a microserviços simplifica a gestão de dados, garantindo flexibilidade e escalabilidade.

## **Linguagem e Ambiente de Execução**

* **Java**

Java é uma linguagem de programação orientada a objetos, desenvolvida pela Sun Microsystems e lançada em 1995. Atualmente mantida pela Oracle Corporation, a linguagem destaca-se pela portabilidade, devido à sua execução numa Máquina Virtual Java *(Java Virtual Machine, JVM)*, que permite que o mesmo código funcione em diferentes plataformas. Este princípio é resumido no lema: “Escreva uma vez, execute em qualquer lugar” (Write Once, Run Anywhere) [8].

A linguagem é amplamente utilizada em sistemas de larga escala, como aplicações web e empresariais, devido à sua segurança, robustez e vasto ecossistema de bibliotecas *(Libraries)* e *frameworks*, como o Spring e o Hibernate. Em plataformas de e-commerce, Java pode ser usada para criar sistemas modulares e escaláveis. Por exemplo, o Spring Framework facilita a construção de microserviços para gerir funcionalidades como catálogos de produtos, pagamentos e análise de dados, garantindo integração com sistemas externos através de APIs RESTful.

Java suporta conceitos como herança, interfaces e programação funcional (desde a versão 8), o que permite flexibilidade no desenvolvimento de software. Além disso, é compatível com ferramentas como NetBeans e IntelliJ IDEA, que simplificam a implementação e manutenção de aplicações.

A versatilidade e a robustez do Java fazem desta linguagem uma escolha ideal para sistemas orientados a microserviços, garantindo desempenho e escalabilidade.

## **Ferramentas e Frameworks de Implementação e Construção**

* **Spring Boot**

O Spring Boot é um *framework* *open-source* desenvolvido a partir do **Spring Framework**. Criado para facilitar o desenvolvimento de aplicações em Java, ele simplifica configurações iniciais e elimina a necessidade de processos manuais extensivos. Oferece funcionalidades como um servidor embutido (como o Tomcat) e suporte nativo a várias dependências, otimizando a criação de aplicações robustas e escaláveis. É amplamente usado no ***backend*** para desenvolver microserviços, devido à sua arquitetura modular e capacidade de integração com outras ferramentas [9].

Um exemplo prático é o desenvolvimento de uma plataforma de e-commerce. Nesta, o Spring Boot pode gerir microserviços separados para o catálogo de produtos, carrinho de compras e sistema de pagamentos. Utilizando APIs RESTful, esses serviços são independentes e desacoplados, garantindo maior resiliência. O Spring Boot inicializa essas aplicações rapidamente e permite que sejam escaladas ou modificadas sem impactar o sistema como um todo.

Com recursos como injeção de dependências, anotações *(annotations)* simplificadas e o suporte ao Spring Initializr, o Spring Boot reduz significativamente o tempo de desenvolvimento. O Spring Actuator, por sua vez, permite monitorizar o desempenho e a saúde da aplicação, essencial em sistemas complexos.

O Spring Boot é uma ferramenta versátil para desenvolvimento de aplicações backend orientadas a microserviços, combinando simplicidade, modularidade e desempenho.

* **Apache Maven**

O Apache Maven é uma ferramenta de automação de compilação e gestão de projetos, amplamente utilizada no desenvolvimento Java, mas também suportando outras linguagens como C# e Scala. Baseado em uma arquitetura modular por *plugins*, o Maven utiliza o conceito de ***POM (Project Object Model)*** para centralizar as configurações do projeto, como dependências, diretórios e fases de compilação. Essa abordagem promove a convenção sobre configuração, reduzindo o esforço manual na gestão do ciclo de vida do software [10].

Na prática, o Maven automatiza tarefas como compilação, execução de testes e empacotamento de aplicações, utilizando comandos simples como *mvn compile* ou *mvn package*. Ele também gere dependências e as baixa de repositórios centralizados, como o Maven Central, garantindo que bibliotecas externas e as suas dependências transitivas estejam disponíveis localmente [10].

No contexto de microserviços desenvolvidos com Spring Boot, o Maven desempenha um papel crucial ao simplificar a gestão de dependências e automatizar a criação de artefactos *(artefacts)*. Por exemplo, em uma plataforma de e-commerce, o Maven pode coordenar a compilação e execução de microserviços para funcionalidades como gestão de produtos e pagamentos, garantindo que cada módulo seja devidamente empacotado e deployado.

No IntelliJ IDEA, o Maven é integrado diretamente, permitindo configurar projetos com dependências como Spring Web e Spring Data JPA no arquivo *pom.xml*, facilitando o desenvolvimento incremental de microserviços.

O Apache Maven é essencial no desenvolvimento de microserviços com Spring Boot, oferecendo automação, gestão eficiente de dependências e integração direta com IDEs como IntelliJ IDEA.

* **Lombok Java**

Lombok é uma biblioteca Java amplamente utilizada que reduz significativamente o código repetitivo *(boilerplate code)*, como *getters, setters,* métodos construtores, e implementações de *equals()*, *hashCode()* e *toString()*. Através de anotações simples, Lombok gera automaticamente esses elementos durante a compilação, sem incluí-los no código-fonte original, promovendo maior legibilidade e manutenção do código [11].

Entre as principais anotações do Lombok estão:

* ***@Getter***e ***@Setter***, que geram métodos de acesso para os campos da classe;
* ***@NoArgsConstructor***, ***@AllArgsConstructor***, e ***@RequiredArgsConstructor***, que criam diferentes tipos de construtores;
* ***@Builder***, para implementar o padrão *builder*;
* ***@Data,*** que combina várias anotações úteis em classes de objetos simples (POJOs);
* ***@Slf4j,*** que adiciona suporte para *logging* integrado.

No contexto de Spring Boot em um ambiente orientado a microserviços, Lombok é frequentemente utilizado para criar rapidamente classes modelo e **DTOs (*Data Transfer Objects*)**, melhorando a produtividade dos programadores. Em uma aplicação de e-commerce, por exemplo, Lombok pode ser usado para definir objetos como Produto ou Utilizador, eliminando a necessidade de codificar manualmente métodos repetitivos.

No IntelliJ IDEA, Lombok integra-se perfeitamente com o Maven, sendo configurado no *pom.xml* para automatizar sua aplicação no projeto. Isso simplifica o desenvolvimento, reduzindo o tempo gasto em tarefas manuais e permitindo foco nas regras de negócio.

Lombok é uma ferramenta indispensável para aplicações Java modernas, otimizando o desenvolvimento em projetos Spring Boot orientados a microserviços, com foco em produtividade e manutenção simplificada.

## **Ferramentas de Containerização e Orquestração**

* **Docker**

Docker é uma plataforma de containerização de aplicações, que utiliza virtualização a nível do sistema operativo para empacotar aplicações e todas as suas dependências em unidades isoladas chamadas containers. Esses containers são leves, portáteis e independentes da infraestrutura subjacente, permitindo que aplicações sejam executadas de forma consistente em qualquer ambiente com o Docker instalado [12].

A arquitetura do Docker baseia-se em um modelo utilizador-servidor, onde o Docker Engine (o servidor) é responsável pela criação e gestão de containers, imagens (images), redes e volumes. Os Dockerfiles definem o processo de construção de imagens, que podem ser armazenadas no Docker Hub, um repositório na nuvem usado para partilhar imagens [12].

No contexto de microserviços, como em um projeto de e-commerce Spring Boot no IntelliJ IDEA, o Docker desempenha um papel crucial. Ele facilita o isolamento de serviços, a escalabilidade e a gestão de dependências. Por exemplo, em uma plataforma de e-commerce, diferentes containers podem ser usados para o serviço de autenticação, gestão de produtos e processamento de pagamentos, garantindo independência e flexibilidade.

Além disso, o Docker permite a criação de ambientes consistentes para desenvolvimento, testes e produção, eliminando problemas como *"funciona no meu computador"*. Essa consistência é essencial para projetos distribuídos e sistemas complexos baseados em microserviços.

Docker é uma ferramenta essencial para a gestão eficiente de ambientes de desenvolvimento e produção, sendo fundamental para a implementação de microserviços em sistemas modernos.

* **Kubernetes (K8s)**

O Kubernetes (K8s) é uma plataforma de orquestração de containers open source, projetada para automatizar a implementação, gestão e escalabilidade de aplicações containerizadas. Originalmente desenvolvido pelo Google e agora mantido pela Cloud Native Computing Foundation (CNCF), o Kubernetes simplifica a gestão de infraestruturas baseadas em containers, organizando-os em grupos chamados *pods* [13].

O Kubernetes opera em um modelo de *cluster*, composto pelo *control plane*, responsável por gerir o estado desejado do cluster, e pelos nós, que executam os containers. Essa arquitetura permite o balanceamento de carga, alocação eficiente de recursos e substituição automática de *pods* em caso de falhas. Além disso, suporta diferentes infraestruturas, como servidores bare-metal, máquinas virtuais e nuvens públicas ou híbridas [13].

No desenvolvimento de microserviços, como em um projeto e-commerce Spring Boot no IntelliJ IDEA, o Kubernetes é crucial. Ele permite a orquestração de múltiplos serviços independentes, como autenticação, carrinho de compras e gestão de produtos, garantindo escalabilidade e alta disponibilidade. Por exemplo, o Kubernetes pode redistribuir a carga de trabalho entre os *pods*, assegurando desempenho contínuo durante picos de tráfego.

Kubernetes é uma ferramenta indispensável para gerir aplicações containerizadas em escala, sendo essencial para arquiteturas modernas baseadas em microserviços.

* **Minikube**

O Minikube é uma ferramenta *open source* que permite criar um *cluster local* do Kubernetes (K8s) em ambientes de desenvolvimento. Destina-se a simplificar a aprendizagem e a experimentação com Kubernetes, permitindo que programadores configurem um ambiente Kubernetes completo em apenas um comando: *minikube start* [14].

O Minikube é executado em sistemas operativos compatíveis, como Windows, macOS e Linux, e requer um gestor de containers (como Docker) ou máquinas virtuais (VMs) para funcionar. Ele suporta múltiplos *drivers* de virtualização, como VirtualBox e Hyper-V, além de oferecer compatibilidade com ferramentas como kubectl para gerir e monitorizar o *cluster*. Durante a execução, o Minikube também disponibiliza o Kubernetes Dashboard, uma interface gráfica para visualização e gestão do ambiente [14].

Na implementação de aplicações baseadas em microserviços, como em um projeto Spring Boot e-commerce no IntelliJ IDEA, o Minikube permite simular *clusters* Kubernetes locais. Isso é crucial para testar funcionalidades como escalabilidade, balanceamento de carga e comunicação entre serviços. Por exemplo, num e-commerce, microserviços para carrinho de compras, gestão de pagamentos e catálogo de produtos podem ser testados de forma integrada antes da implantação em um ambiente de produção.

O Minikube é uma ferramenta essencial para simular ambientes Kubernetes localmente, facilitando o desenvolvimento, teste e aprendizagem de arquiteturas orientadas a microserviços.

## **Serviços de Descoberta e Comunicação**

* **Eureka**

O Eureka, parte do Spring Cloud Netflix, é uma solução de descoberta de serviços *client-side* que permite que diferentes microserviços localizem e comuniquem-se entre si sem depender de configurações estáticas de *host* ou porta. A arquitetura de descoberta baseia-se num registo centralizado de serviços, conhecido como Eureka Server, onde cada microserviço utilizador *(Eureka Client)* se regista e envia sinais de vida *(heartbeats)* para manter sua disponibilidade atualizada [15].

O Eureka Server é configurado como uma aplicação Spring Boot anotada com *@EnableEurekaServer* e fornece um painel de monitorização para verificar os serviços registados. Já o Eureka Client, configurado com *@EnableEurekaClient*, comunica-se com o registo para obter informações sobre outros serviços, permitindo balanceamento de carga e alta disponibilidade através de algoritmos integrados [15].

Em arquiteturas de microserviços orientadas a e-commerce, como em projetos Spring Boot no IntelliJ IDEA, o Eureka desempenha um papel essencial. Por exemplo, em uma plataforma de e-commerce, microserviços para gestão de inventário, processamento de pagamentos e notificações podem ser dinamicamente localizados e escalados conforme necessário, otimizando o desempenho e a resiliência do sistema.

O Eureka é um componente central na orquestração e comunicação de microserviços, garantindo escalabilidade e gestão eficiente em ambientes distribuídos.

* **RabbitMQ**

O RabbitMQ é um *message broker* (sistema ou protocolo de mensagens) que facilita a comunicação assíncrona entre sistemas, atuando como intermediário para a troca de mensagens em uma arquitetura distribuída [13]. Utilizado em plataformas de e-commerce, permite que diferentes aplicações, como módulos de venda e gestão de inventário, se comuniquem de forma escalável e sem estarem diretamente ligadas [14]. Esta comunicação desacoplada é especialmente útil para sistemas que precisam processar diversas tarefas, como processamento de compras e atualizações de inventário, mas que não exigem resposta imediata [13].

Dentro do RabbitMQ, o Producer (Produtor) é responsável por criar e enviar mensagens, enquanto o Consumer (Consumidor) as recebe e processa. As mensagens transitam por uma Queue (Fila) e são distribuídas por um Exchange, que determina o destino com base em uma Routing Key (Chave de Roteamento) [14]. Esse fluxo reduz a sobrecarga em sistemas e melhora a resiliência, já que mensagens ficam em espera até que o Consumer esteja disponível para processá-las [13]. Em resumo, RabbitMQ é um sistema de mensagens que permite comunicação assíncrona, facilitando a troca de informações de forma confiável e organizada [14].

* **HTTP (Hyper Text Transfer Protocol)**

O HTTP (Hypertext Transfer Protocol), ou Protocolo de Transferência de Hipertexto, é o principal protocolo utilizado na comunicação entre utilizadores e servidores na web. Ele segue um modelo de requisição e resposta, onde o utilizador (geralmente um navegador) solicita recursos, como páginas HTML, imagens ou dados, e o servidor responde com o conteúdo solicitado e informações no cabeçalho da mensagem. Desde sua criação por Tim Berners-Lee em 1989, o HTTP evoluiu significativamente, culminando em versões como HTTP/2 e HTTP/3, que aprimoram eficiência, segurança e velocidade [18].

O HTTP utiliza métodos, como **GET**, **POST** e **DELETE**, para definir ações em recursos, enquanto os cabeçalhos contêm informações adicionais, como tipo de conteúdo e controle de cache. A simplicidade do protocolo, por não manter estado entre requisições, é complementada por mecanismos como *cookies* e *sessões* para atender a aplicações que necessitam de persistência.

Em projetos de e-commerce baseados em microserviços, como no Spring Boot em um ambiente IntelliJ IDEA, o HTTP facilita a integração entre APIs *(Application Programming Interfaces –* Interfaces de Programação de Aplicações) e *frontends*, permitindo que módulos como carrinho de compras, sistemas de pagamento e gestão de inventário comuniquem-se de forma confiável e escalável.

O HTTP é um protocolo essencial para a comunicação na web, garantindo transferência eficiente de dados e integração entre microserviços em arquiteturas distribuídas.

* **AMQP (Advanced Message Queuing Protocol)**

O AMQP (Advanced Message Queuing Protocol), ou Protocolo Avançado de Fila de Mensagens, é um protocolo de comunicação independente de plataforma, projetado para sistemas de *middleware* orientados a mensagens. Ele padroniza a troca de mensagens entre aplicações, permitindo comunicação assíncrona e interoperabilidade. Amplamente utilizado em soluções baseadas em microserviços, o AMQP fornece uma interface robusta para sistemas distribuídos, oferecendo entrega confiável, enfileiramento e roteamento de mensagens [19].

Uma das implementações mais conhecidas do AMQP é o **RabbitMQ**, que funciona como um *message broker* (corretor de mensagens) para gerir o envio e recebimento de mensagens entre serviços. O Spring Boot integra-se facilmente ao RabbitMQ, utilizando o **Spring AMQP**, que aplica conceitos do *framework* Spring para simplificar o desenvolvimento de soluções baseadas em mensagens. Os programadores podem configurar propriedades, como o *host* e credenciais do RabbitMQ, para criar filas, enviar e processar mensagens diretamente com o uso de anotações, como *@RabbitListener* [19]*.*

Em plataformas de e-commerce que utilizam Spring Boot com arquiteturas de microserviços, o AMQP desempenha um papel fundamental na conexão entre componentes, como serviços de inventário, processamento de pagamentos e notificações. Além disso, proporciona escalabilidade, confiabilidade e desacoplamento entre serviços, características indispensáveis para aplicações modernas e distribuídas.

O AMQP é um protocolo fundamental para comunicação eficiente em sistemas distribuídos, especialmente na orquestração de microserviços, contribuindo para maior escalabilidade e integração em plataformas e-commerce.

## **Gateway e Segurança**

* **Spring Cloud Gateway**

O Spring Cloud Gateway é uma solução moderna para atuar como um **API Gateway**, implementado com base no Spring Ecosystem (Spring Boot 3, Spring Framework 6 e Project Reactor) [20]. Este componente desempenha um papel crucial em arquiteturas de microserviços, oferecendo funcionalidades como roteamento para APIs e preocupações transversais, como segurança, monitorização e resiliência. Utiliza o modelo de programação reativa do Spring WebFlux, permitindo lidar com grandes volumes de tráfego e garantindo eficiência.

O principal objetivo do Spring Cloud Gateway é gerir rotas, que são as unidades básicas de sua configuração. Cada rota é definida por:

* Predicates (Predicados): Condições para corresponder pedidos (por exemplo, cabeçalhos ou parâmetros de URL).
* Filters (Filtros): Permitem modificar pedidos e respostas antes ou depois de serem processados [20].

A configuração pode ser feita através de arquivos **YAML** ou programaticamente. Um exemplo prático para uma aplicação de e-commerce envolve a configuração de rotas que redirecionam pedidos para microserviços como catálogo, carrinho ou pagamento, com validações de segurança integradas por meio de filtros.

O Spring Cloud Gateway atua no *backend*, mas é essencial para a comunicação entre microserviços e utilizadores. Em projetos de e-commerce baseados no Spring Boot e com arquiteturas de microserviços, ele assegura que os pedidos sejam roteados de forma eficiente e segura, utilizando filtros como *Circuit Breakers* para evitar falhas em cascata.

Numa plataforma de e-commerce, um pedido de inventário pode ser redirecionado ao serviço apropriado usando uma rota configurada com um predicado que valida o caminho (por exemplo, /stocks/\*\*) e um filtro que garante a autenticação [20].

O Spring Cloud Gateway é um componente essencial para a gestão de comunicação em sistemas distribuídos, garantindo segurança, escalabilidade e controle robusto de tráfego.

## **Ferramentas de Testes e Interação com APIs**

* **Postman**

O Postman é uma ferramenta independente amplamente utilizada para testar APIs Web, permitindo realizar requisições HTTP sem a necessidade de implementar código de infraestrutura [21]. Ele é essencial em ambientes de desenvolvimento orientados a microserviços, como projetos de e-commerce baseados em Spring Boot, por facilitar a validação e depuração de interações entre serviços.

O Postman oferece uma interface gráfica que suporta a criação de coleções *(collections)*, que agrupam um conjunto de requisições configuráveis. Estas podem incluir métodos como **GET**, **POST, PUT** e **DELETE**, com suporte a personalizações como cabeçalhos, parâmetros e corpos de mensagem. As respostas retornadas podem ser avaliadas automaticamente através de *scripts* em JavaScript, utilizando o módulo integrado *pm*, que permite criar testes e verificar a conformidade dos dados [21].

No contexto de microserviços, o Postman opera como uma ferramenta de apoio ao *backend*, desempenhando um papel crítico na verificação da comunicação entre serviços. Por exemplo, num projeto de e-commerce desenvolvido em IntelliJ IDEA, o Postman pode ser usado para testar endpoints REST, como o envio de pedidos de inventário ou a verificação de status de pagamentos.

O Postman acelera o ciclo de desenvolvimento e aumenta a confiabilidade ao permitir testes rápidos e precisos. Além disso, com a execução de coleções automatizadas, é possível simular fluxos completos de aplicações, como a criação de pedidos, verificação de estados e validações de erro.

O Postman é uma ferramenta essencial para testar APIs e validar integrações entre microserviços, garantindo eficiência e confiabilidade no desenvolvimento de sistemas distribuídos.

* **Junit**

O JUnit é uma framework de testes para aplicações Java que permite a criação e execução de testes automatizados, sendo amplamente utilizada para garantir a qualidade de software, especialmente em arquiteturas baseadas em Spring Boot e microserviços [22]. Ele suporta tanto *unit tests* (testes unitários), que verificam componentes isolados, quanto *integration tests* (testes de integração), que analisam a interação entre diferentes camadas da aplicação.

Características e Funcionalidades:

* Propósito: Permitir a validação automática de comportamentos esperados em classes e métodos, reduzindo erros humanos durante o processo de desenvolvimento.
* Backend ou Frontend: Atua no backend, sendo frequentemente usado em combinações com frameworks de persistência e serviços REST.
* Métodos de Teste: Suporte a anotações como *@Test* (definir um teste), *@BeforeEach* (configurações antes de cada teste) e *@SpringBootTest* (para inicializar o contexto do Spring).

Em projetos de e-commerce, é possível testar o comportamento de serviços como o cálculo do preço final, verificando a lógica de integração entre camadas (e.g., serviços e controladores REST).

O JUnit é crucial para validar que as alterações em uma parte do código não causam regressões em outros módulos. Em projetos baseados em microserviços, isso assegura a confiabilidade na comunicação e no processamento de dados entre os serviços.

O JUnit é uma ferramenta indispensável para criar testes automatizados em projetos *backend*, garantindo a robustez e a integridade em sistemas distribuídos como os baseados em microserviços.

* **REST Assured**

O REST Assured é uma biblioteca Java destinada ao teste de APIs RESTful, permitindo verificar a conformidade de respostas HTTP de forma automatizada e eficiente. Integrando-se perfeitamente com *frameworks* de teste como o JUnit, fornece uma linguagem específica de domínio (DSL) para criar testes fluentes e fáceis de manter [23].

É usado para testes de APIs RESTful permitindo especificar expectativas como o código de estado HTTP status (código de estado), tipo de conteúdo content-type (tipo de conteúdo), e campos no corpo body (corpo) da resposta simplifica a interação com endpoints de API através de métodos como *get()* para GET e *post()* para POST integra-se com sistemas de teste populares, como o JUnit e executa verificações detalhadas de respostas HTTP.

Em sistemas de microserviços orientados a *backend*, como um projeto e-commerce Spring Boot, o REST Assured assegura que os endpoints estão a retornar as respostas esperadas, mesmo durante alterações ou expansões do sistema. É essencial para validar operações como registo de utilizadores ou cálculo de preços.

REST Assured é uma ferramenta fundamental para testes automatizados de APIs RESTful, garantindo a confiabilidade e a qualidade em arquiteturas orientadas a microserviços.

## **Bases de Dados**

* **PostgreSQL**

O PostgreSQL é um Sistema de Gestão de Base de Dados Relacional (RDBMS) robusto, escalável e de código aberto. Reconhecido pela sua versatilidade e conformidade com os padrões SQL, é amplamente utilizado em sistemas de microserviços, sendo ideal para aplicações de e-commerce devido ao seu suporte avançado para transações, integridade referencial e extensibilidade [24].

PostgreSQL é multiplataforma possui suporte a transações ACID integra-se facilmente com *frameworks* como o Spring Boot suporta extensões como JSON (JavaScript Object Notation), tornando-o adequado para aplicações modernas orientadas a dados permite a configuração de sequências automáticas e índices personalizados inclui ferramentas para segurança, como controle granular de acessos e autenticação.

No desenvolvimento de um sistema de e-commerce baseado em Spring Boot com arquitetura de microserviços, o PostgreSQL destaca-se pela confiabilidade na gestão de dados críticos, como informações de utilizadores, inventários e transações. A integração com Spring Data JPA simplifica a manipulação dos dados no backend, enquanto o suporte a índices e consultas complexas otimiza a performance de operações frequentes.

PostgreSQL é um componente essencial para sistemas orientados a microserviços, fornecendo armazenamento robusto e escalável, indispensável para aplicações modernas como e-commerce.

* **MySQL**

O MySQL é um Sistema de Gestão de Bases de Dados Relacional (RDBMS) de código aberto, amplamente utilizado para armazenar, gerir e recuperar dados estruturados. Desenvolvido pela Oracle Corporation, utiliza SQL (Structured Query Language – Linguagem de Consulta Estruturada) para realizar operações de base de dados. Destaca-se pela fiabilidade, velocidade e facilidade de uso, sendo utilizado tanto por pequenas aplicações como por sistemas empresariais de grande escala [25].

MySQL é gratuito e open source (código aberto) adota o modelo relacional que organiza dados em tabelas com linhas e colunas oferece alta performance em transações volumosas e grandes conjuntos de dados é escalável para atender a cargas crescentes de utilizadores e dados suporta linguagens como Java, Python e PHP inclui funcionalidades de segurança, como encriptação e controlos de acesso suporta integrações com serviços cloud (nuvem), otimizando escalabilidade e backup automáticos.

Em sistemas e-commerce baseados em Spring Boot com microserviços, o MySQL é essencial para gerir dados de produtos, utilizadores e transações. A sua compatibilidade com frameworks como o Spring Data JPA facilita a criação de serviços backend robustos no IntelliJ IDEA. A escalabilidade e suporte a transações ACID tornam-no adequado para aplicações críticas.

O MySQL é um RDBMS versátil e confiável, essencial para o desenvolvimento de sistemas orientados a microserviços, como plataformas de e-commerce, devido à sua alta performance, escalabilidade e integração com diversas ferramentas.

# **Requisitos Funcionais**

Os requisitos funcionais definem as funcionalidades essenciais para o sistema, determinando "o quê" o projeto **FoodStream** deve realizar. Cada microserviço implementa um conjunto específico de funções alinhadas ao domínio do negócio, garantindo modularidade e coesão. A seguir, são descritas as responsabilidades dos microserviços no sistema.

## **Microserviço – gestao\_utilizadores**

O **microserviço gestao\_utilizadores** é responsável pela gestão de perfis de utilizadores, autenticação e autorização, suportando operações como criação, atualização e eliminação de contas para clientes, funcionários e administradores. A sua arquitetura modular, baseada na abordagem DDD, assegura escalabilidade e separação de responsabilidades. Este microserviço integra endpoints para autenticação, permissões e notificações, sendo um componente central no ecossistema FoodStream.

Tabela 2 – Requisitos funcionais – microserviço ‘gestao\_utilizadores’

|  |  |
| --- | --- |
| **RF** | **Descrição do RF** |
| **RF1** | Criar Utilizador |
| **RF2** | Autenticação de Utilizador (Login) |
| **RF3** | Editar Permissões |
| **RF4** | Listar Utilizadores |
| **RF5** | Registar Status de Login |
| **RF6** | Atualizar Perfil do Utilizador |
| **RF7** | Enviar Notificações |
| **RF8** | Validar Token JWT |
| **RF9** | Excluir Utilizador |
| **RF10** | Recuperar Informações do Utilizador |

### **RF1 – Criar Utilizador**

Tabela 3 – Requisito Funcional – Criar Utilizador

|  |  |
| --- | --- |
| **RF1** | **Criar Utilizador** |
| **Ator:** | Administrador |
| **Ator Responsável:** | Administrador |
| **Microserviço:** | gestao\_utilizadores |
| **Descrição:** | O sistema deve permitir a criação de um utilizador com nome, email, password e status. O administrador deve associar um ou mais papéis (roles) ao utilizador. |
| **Objetivo:** | Registar novos utilizadores no sistema. |
| **Base de Dados:** | Tabela: utilizadores  Tabela: utilizadores\_roles |
| **Verificações:** | O sistema deve verificar se:   * 1. Todos os campos obrigatórios são preenchidos.   2. O email do utilizador deve ser único.   3. A password deve ser criptografada.   4. O papel associado deve existir na tabela roles. |
| **HTTP Codes:** | 201 (Created),  400 (Bad Request),  409 (Conflict). |
| **Restrições:** | Apenas administradores podem criar utilizadores. |
| **Endpoints:** | **POST /api/utilizadores** |

### **RF2 – Autenticação de Utilizador (Login)**

Tabela 4 – Requisito Funcional – Autenticação de Utilizador (Login)

|  |  |
| --- | --- |
| **RF2** | **Autenticação de Utilizador (Login)** |
| **Ator:** | Todos os utilizadores (Administrador, Funcionário, Cliente) |
| **Ator Responsável:** | Utilizador solicitante |
| **Microserviço:** | gestao\_utilizadores |
| **Descrição:** | O sistema deve autenticar o utilizador com email e password, gerando um token JWT para acesso aos serviços. |
| **Objetivo:** | Permitir que os utilizadores façam login no sistema |
| **Base de Dados:** | Tabela: utilizadores  Tabela: registo\_logins |
| **Verificações:** | O sistema deve:   1. O utilizador deve existir na base de dados. 2. A password fornecida deve corresponder à armazenada. 3. Registar o status do login (sucesso ou falha) na tabela registo logins. |
| **HTTP Codes:** | 200 (OK)  401 (Unauthorized)  404 (Not Found) |
| **Restrições:** | A password deve ser criptografada na base de dados. |
| **Endpoints:** | **POST /api/auth/login** |

### **RF3 – Editar Permissões**

Tabela 5 – Requisito Funcional – Editar Permissões

|  |  |
| --- | --- |
| **RF3** | **Editar Permissões** |
| **Ator:** | Administrador |
| **Ator Responsável:** | Administrador |
| **Microserviço:** | gestao\_utilizadores |
| **Descrição:** | Permitir que o administrador edite os papéis associados a um utilizador.  Existem 3 níveis:  Nível 1 – Administrador  Nível 2 – Funcionário  Nível 3 – Cliente (padrão).  Os níveis quando são criados são todos nível 3. |
| **Objetivo:** | Alterar os papéis de acesso dos utilizadores. |
| **Base de Dados:** | Tabela: utilizadores\_roles |
| **Verificações:** | O sistema deve verificar se:   * 1. O utilizador deve existir na base de dados.   2. O papel a ser associado ou removido deve ser válido. |
| **HTTP Codes:** | 200 (OK),  400 (Bad Request)  404 (Not Found). |
| **Restrições:** | Apenas administradores podem editar permissões. |
| **Endpoints:** | **PUT /api/utilizadores/{id}/roles** |

### **RF4 – Listar Utilizadores**

Tabela 6 – Requisito Funcional – Listar Utilizadores

|  |  |
| --- | --- |
| **RF4** | **Listar Utilizadores** |
| **Ator:** | Administrador |
| **Ator Responsável:** | Administrador |
| **Microserviço:** | gestao\_utilizadores |
| **Descrição:** | Listar todos os utilizadores registados no sistema. |
| **Objetivo:** | Exibir uma lista de utilizadores com seus detalhes (nome, email, status e papéis). |
| **Base de Dados:** | Tabela: utilizadores  Tabela: utilizadores\_roles  Tabela: roles |
| **Verificações:** | O sistema deve verificar se:   * 1. O utilizador solicitante deve ter papel de administrador. |
| **HTTP Codes:** | 200 (OK)  403 (Forbidden) |
| **Restrições:** | Apenas administradores podem listar todos os utilizadores. |
| **Endpoints:** | **GET /api/utilizadores** |

### **RF5 – Registar Status de Login**

Tabela 7 – Requisito Funcional – Registar Status de Login

|  |  |
| --- | --- |
| **RF5** | **Registar Status de Login** |
| **Ator:** | Sistema |
| **Ator Responsável:** | Sistema |
| **Microserviço:** | gestao\_utilizadores |
| **Descrição:** | Registar o status (sucesso ou falha) de cada tentativa de login na tabela registo\_logins. |
| **Objetivo:** | Rastrear logins no sistema. |
| **Base de Dados:** | Tabela: registo\_logins |
| **Verificações:** | O sistema deve verificar se:   * 1. O id do utilizador deve ser válido. |
| **HTTP Codes:** | 201 (Created) |
| **Restrições:** | Nenhuma. |
| **Endpoints:** | **POST /api/auth/login**  (registado automaticamente como parte do fluxo de autenticação) |

### **RF6 – Atualizar Perfil do Utilizador**

Tabela 8 – Requisito Funcional – Atualizar Perfil do Utilizador

|  |  |
| --- | --- |
| **RF6** | **Atualizar Perfil do Utilizador** |
| **Ator:** | Utilizador |
| **Ator Responsável:** | Utilizador autenticado |
| **Microserviço:** | gestao\_utilizadores |
| **Descrição:** | Permitir que os utilizadores atualizem suas informações pessoais (nome, email, password). |
| **Objetivo:** | Manter os dados pessoais atualizados. |
| **Base de Dados:** | Tabela: utilizadores |
| **Verificações:** | O sistema deve verificar se:   * 1. O utilizador deve estar autenticado.   2. O email atualizado deve ser único. |
| **HTTP Codes:** | 200 (OK)  400 (Bad Request)  401 (Unauthorized)  409 (Conflict) |
| **Restrições:** | Apenas o utilizador dono do perfil pode editá-lo. |
| **Endpoints:** | **PUT /api/utilizadores/{id}** |

**RF7 – Enviar Notificações**

Tabela 9 – Requisito Funcional – Enviar Notificações

|  |  |
| --- | --- |
| **RF7** | **Enviar Notificações** |
| **Ator:** | Sistema (via RabbitMQ) |
| **Ator Responsável:** | Sistema (via RabbitMQ) |
| **Microserviço:** | gestao\_utilizadores |
| **Descrição:** | O sistema deve enviar notificações para os utilizadores por meio do RabbitMQ, como atribuições de entregas ou avaliações pendentes. |
| **Objetivo:** | Garantir que os utilizadores sejam notificados sobre eventos importantes. |
| **Base de Dados:** | Tabela: **notificacoes** |
| **Verificações:** | O sistema deve verificar se:   1. O id do utilizador é válido. 2. A mensagem é gerada corretamente. |
| **HTTP Codes:** | 200 (OK)  500 (Internal Server Error) |
| **Restrições:** | Nenhuma. |
| **Endpoints:** | **POST /api/notificacoes** |

### **RF8 – Validar Token JWT**

Tabela 10 – Requisito Funcional – Validar Token JWT

|  |  |
| --- | --- |
| **RF8** | **Validar Token JWT** |
| **Ator:** | API Gateway, Outros Microserviços |
| **Ator Responsável:** | Sistema |
| **Microserviço:** | gestao\_utilizadores |
| **Descrição:** | Permitir que o sistema valide tokens JWT enviados por outros serviços ou o API Gateway. |
| **Objetivo:** | Garantir a segurança dos endpoints protegidos por autenticação. |
| **Base de Dados:** | Nenhuma. |
| **Verificações:** | O sistema deve verificar se:   1. O token é válido e não expirou. 2. As informações do token correspondem a um utilizador existente. |
| **HTTP Codes:** | 200 (OK)  401 (Unauthorized) |
| **Restrições:** | Nenhuma. |
| **Endpoints:** | **GET /api/auth/validate** |

### **RF9 – Excluir Utilizador**

Tabela 11 – Requisito Funcional – Excluir Utilizador

|  |  |
| --- | --- |
| **RF9** | **Excluir Utilizador** |
| **Ator:** | Administrador |
| **Ator Responsável:** | Administrador |
| **Microserviço:** | gestao\_utilizadores |
| **Descrição:** | Permitir que o administrador exclua um utilizador do sistema. A exclusão deve remover todas as associações (exemplo: permissões e notificações). |
| **Objetivo:** | Remover utilizadores inativos ou indesejados. |
| **Base de Dados:** | Tabela: **utilizadores**  Tabela: **utilizadores\_roles**  Tabela: **notificacoes** |
| **Verificações:** | O sistema deve verificar se:  1. O id do utilizador é válido.  2. O utilizador a ser excluído não possui dependências que impeçam a exclusão. |
| **HTTP Codes:** | 200 (OK)  404 (Not Found) |
| **Restrições:** | Apenas administradores podem excluir utilizadores. |
| **Endpoints:** | **DELETE /api/utilizadores/{id}** |

### **RF10 – Recuperar Informações do Utilizador**

Tabela 12 – Requisito Funcional – Recuperar Informações do Utilizador

|  |  |
| --- | --- |
| **RF10** | **Recuperar Informações do Utilizador** |
| **Ator:** | Utilizador Autenticado |
| **Ator Responsável:** | Utilizador Autenticado |
| **Microserviço:** | gestao\_utilizadores |
| **Descrição:** | Permitir que um utilizador autenticado recupere suas informações pessoais e permissões. |
| **Objetivo:** | Exibir detalhes do utilizador logado. |
| **Base de Dados:** | Tabela: **utilizadores**  Tabela: **utilizadores\_roles**  Tabela: **roles** |
| **Verificações:** | O sistema deve verificar se:  1. O utilizador está autenticado. |
| **HTTP Codes:** | 200 (OK)  401 (Unauthorized) |
| **Restrições:** | Apenas o utilizador autenticado pode aceder suas informações. |
| **Endpoints:** | **GET /api/utilizadores/me** |

## **Microserviço – criacao\_menu**

Este microserviço trata da criação, listagem, atualização e remoção de menus, seja diários ou semanais. Permite que clientes selecionem refeições de acordo com os menus disponíveis, otimizando a oferta de opções para consumo. Está integrado com o microserviço de stock para garantir a viabilidade das receitas.

Tabela 14 – Requisito Funcional – CRUD de menus

|  |  |
| --- | --- |
| **RF01** | **Criar novo menu** |
| **Ator:** |  |
| **Ator Responsável:** |  |
| **Microserviço:** |  |
| **Descrição:** |  |
| **Objetivo:** |  |
| **Localização de Dados:** |  |
| **Verificações:** |  |
| **HTTP Codes:** |  |
| **Restrições:** |  |
| **Endpoints:** | POST/menus |

Tabela 15 – Requisito Funcional – Listagem por intervalos de tempo

|  |  |
| --- | --- |
| **RF02** | **Atualizar menu existente** |
| **Ator:** |  |
| **Ator Responsável:** |  |
| **Microserviço:** |  |
| **Descrição:** |  |
| **Objetivo:** |  |
| **Localização de Dados:** |  |
| **Verificações:** |  |
| **HTTP Codes:** |  |
| **Restrições:** |  |
| **Endpoints:** |  |
|  |  |
|  |  |

## **Microserviço – verifica\_stock**

Responsável por gerir a logística de ingredientes necessários para a produção de menus, este microserviço controla as quantidades em stock e emite alertas para reposição quando os níveis são críticos. Desta forma, assegura que as receitas podem ser preparadas sem interrupções operacionais.

### **RFX – Listar Ingredientes e nível de stock**

Tabela 16 – Requisito Funcional – Listar ingredientes e nível de stock

|  |  |
| --- | --- |
| **RF** | **Listar ingredientes e nível de stock** |
| **Ator:** | Administrador |
| **Ator Responsável:** | Sistema |
| **Microserviço:** | verifica\_stock |
| **Descrição:** | Permite visualizar todos os ingredientes disponíveis e respetivos níveis de stock em tempo real. |
| **Objetivo:** | Garantir que o administrador possa monitorizar o stock e tomar decisões de reabastecimento. |
| **Base de Dados:** | Tabela de **produtos**, incluindo ID, nome, categoria, quantidade total, tamanhos e disponibilidade. |
| **Verificações:** | Confirmar que a base de dados está acessível e correta. |
| **HTTP Codes:** | 200 OK,  404 Not Found,  500 Internal Server Error |
| **Restrições:** | Apenas administradores autenticados podem consultar os dados. |
| **Endpoints:** | **GET /stock** |

### **RFX – Adicionar Ingredientes ao Stock**

Tabela 17 – Requisito Funcional – Adicionar Ingredientes ao Stock

|  |  |
| --- | --- |
| **RF** | **Adicionar Ingredientes ao Stock** |
| **Ator:** | Administrador / Funcionário |
| **Ator Responsável:** | Sistema |
| **Microserviço:** | verifica\_stock |
| **Descrição:** | Permite registar a entrada de novos ingredientes no sistema. |
| **Objetivo:** | Atualizar o inventário após a aquisição de novos ingredientes. |
| **Base de Dados:** | Tabela de **categoria**, informações sobre o produto (nome, categoria, quantidade inicial, tamanhos e preços). |
| **Verificações:** | Validar os dados do ingrediente e atualizar o inventário corretamente. |
| **HTTP Codes:** | 201 Created,  400 Bad Request,  500 Internal Server Error |
| **Restrições:** | Apenas administradores autenticados podem registar ingredientes. |
| **Endpoints:** | **POST /ingredients** |

### **RFX – Obter Alertas de Reabastecimento**

Tabela 18 – Requisito Funcional – Obter Alertas de Reabastecimento

|  |  |
| --- | --- |
| **RF** | **Obter Alertas de Reabastecimento** |
| **Ator:** | Administrador |
| **Ator Responsável:** | Sistema |
| **Microserviço:** | verifica\_stock |
| **Descrição:** | Fornece alertas automáticos para ingredientes com níveis de stock baixos. |
| **Objetivo:** | Evitar ruturas no fornecimento devido à falta de ingredientes. |
| **Base de Dados:** | Ingredientes com níveis de stock abaixo do limite mínimo configurado. |
| **Verificações:** | Identificar níveis de stock abaixo do limite mínimo e gerar notificações. |
| **HTTP Codes:** | 200 OK,  204 No Content,  500 Internal Server Error |
| **Restrições:** | Apenas administradores autenticados podem aceder aos alertas. |
| **Endpoints:** | **GET /stock/alerts** |

### **RFX – Atualizar Stock Após Consumo**

Tabela 19 – Requisito Funcional – Atualizar Stock Após Consumo

|  |  |
| --- | --- |
| **RF** | **Atualizar Stock Após Consumo** |
| **Ator:** | Sistema |
| **Ator Responsável:** | API de integração com microserviços |
| **Microserviço:** | verifica\_stock |
| **Descrição:** | Deduz automaticamente o consumo de ingredientes ao preparar e entregar menus. |
| **Objetivo:** | Atualizar o inventário em tempo real, refletindo o consumo diário. |
| **Base de Dados:** | Tabela de movimentação de **stock.** |
| **Verificações:** | Validar os dados do consumo e atualizar os níveis de stock corretamente. |
| **HTTP Codes:** | 200 OK,  400 Bad Request,  500 Internal Server Error |
| **Restrições:** | Apenas microserviços autorizados podem realizar atualizações. |
| **Endpoints:** | **PUT /stock/{ingredientId}** |

### **RFX – Validar Stock Antes da Criação de Menu**

Tabela 20 – Requisito Funcional – Validar Stock Antes da Criação de Menu

|  |  |
| --- | --- |
| **RF** | **Validar Stock Antes da Criação de Menu** |
| **Ator:** | Sistema |
| **Ator Responsável:** | API de integração com microserviços |
| **Microserviço:** | verifica\_stock |
| **Descrição:** | Verifica a disponibilidade de stock suficiente antes de permitir a criação de um menu. |
| **Objetivo:** | Garantir que menus não sejam criados sem stock suficiente. |
| **Base de Dados:** | Requisição do menu e lista de ingredientes necessários. |
| **Verificações:** | Validar a disponibilidade de cada ingrediente necessário para o menu. |
| **HTTP Codes:** | 200 OK,  400 Bad Request,  500 Internal Server Error |
| **Restrições:** | Apenas serviços autorizados podem realizar a validação. |
| **Endpoints:** | **GET /stock/validate?menuId={id}** |

### **RFX – Gerar Relatório de Stock**

Tabela 21 – Requisito Funcional – Gerar Relatório de Stock

|  |  |
| --- | --- |
| **RF** | **Gerar Relatório de Stock** |
| **Ator:** | Administrador |
| **Ator Responsável:** | Sistema |
| **Microserviço:** | verifica\_stock |
| **Descrição:** | Gera um relatório detalhado do histórico de movimentação de stock. |
| **Objetivo:** | Fornecer uma visão clara do fluxo de ingredientes ao longo do tempo. |
| **Base de Dados:** | Consolidação de movimentações em formato legível (CSV ou PDF). |
| **Verificações:** | Consolida dados de movimentação num intervalo de tempo especificado. |
| **HTTP Codes:** | 200 OK,  500 Internal Server Error |
| **Restrições:** | Apenas administradores autenticados podem gerar relatórios. |
| **Endpoints:** | GET /stock/report?startDate={date}&endDate={date} |

## **Microserviço – repositorio\_entregas**

Este microserviço é o núcleo do domínio do negócio, funcionando como um repositório central para ordens de trabalho. Funcionários podem subscrever tarefas de entrega com base na capacidade dos veículos disponíveis. Este serviço assegura a atribuição eficiente de entregas, rastreando o progresso e garantindo a conclusão das ordens de trabalho.

Tabela 22 – Requisito Funcional – Listar

|  |  |
| --- | --- |
| **RF05** | **Listar** |
| **Ator:** |  |
| **Ator Responsável:** |  |
| **Microserviço:** |  |
| **Descrição:** |  |
| **Objetivo:** |  |
| **Localização de Dados:** |  |
| **Verificações:** |  |
| **HTTP Codes:** |  |
| **Restrições:** |  |
| **Endpoints:** | GET/stock |

Tabela 23 – Requisito Funcional – Adicio

|  |  |
| --- | --- |
| **RF06** | **Adicionar** |
| **Ator:** |  |
| **Ator Responsável:** |  |
| **Microserviço:** |  |
| **Descrição:** |  |
| **Objetivo:** |  |
| **Localização de Dados:** |  |
| **Verificações:** |  |
| **HTTP Codes:** |  |
| **Restrições:** |  |
| **Endpoints:** | POST /ingredients |

## **Microserviço – calculo\_rotas**

Consoante as ordens de trabalho atribuídas, este microserviço calcula as rotas de entrega mais eficientes, considerando a localização das instituições a serem atendidas. Suporta otimização para múltiplos destinos, melhorando a gestão do tempo e recursos logísticos.

Tabela 27 – Requisito Funcional – CRUD de funcionários

|  |  |
| --- | --- |
| **RF01** | **CRUD de Funcionários** |
| **Ator:** |  |
| **Ator Responsável:** |  |
| **Microserviço:** |  |
| **Descrição:** |  |
| **Objetivo:** |  |
| **Localização de Dados:** |  |
| **Verificações:** |  |
| **HTTP Codes:** |  |
| **Restrições:** |  |
| **Endpoints:** |  |

# **Processos de Negócio**

Os processos de negócio representam o funcionamento interno do sistema, descrevendo **"como"** as funcionalidades são realizadas. A modelação é feita em **BPMN** (Business Process Model and Notation), uma notação padrão para representar processos de forma visual e estruturada. Estes fluxos detalham os principais processos operacionais dos microserviços implementados no projeto **FoodStream**.

## **Processo – ‘gestao\_utilizadores’**

O processo de **gestão de utilizadores** organiza as operações relacionadas à criação, autenticação e autorização de perfis. Este processo garante que utilizadores, incluindo funcionários e clientes, sejam geridos de forma segura e eficiente. Além disso, abrange a atribuição de permissões através da relação entre as tabelas **Utilizadores**, **Roles** e **UtilizadoresRoles**. A monitorização de acessos e envio de notificações é suportada pelas tabelas **RegistoLogins** e **Notificacoes**, assegurando rastreabilidade e comunicação eficazes. O processo promove integridade e modularidade, facilitando a escalabilidade e manutenção do sistema.

Uma imagem com texto, captura de ecrã, software, Software de multimédia

Descrição gerada automaticamente

Figura 4 – Processo de negócio – ‘gestao\_utilizadores’ - ALTERAR

## **Processo – ‘criacao\_menu’**

Este processo define o fluxo para criar, atualizar, listar ou remover menus. Envolve verificações de disponibilidade de ingredientes no stock e gera as opções de refeições diárias ou semanais para os clientes, considerando restrições de tempo e preferências.

Figura 5 – Processo de negócio – ‘criacao\_menu’ - ALTERAR

## **Processo – ‘verifica\_stock’**

O processo de verificação de stock monitoriza continuamente os níveis de ingredientes necessários. Inclui a análise de consumo projetado com base nos menus criados, alertando para encomendas em caso de stock baixo, garantindo a continuidade das operações de produção.

Figura 6 – Processo de negócio – ‘verifica\_stock’ – colocar diagrama BPMN

## **Processo – ‘repositorio\_entregas’**

O processo central do negócio, repositorio\_entregas, gere ordens de trabalho. Funcionários selecionam tarefas de entrega de acordo com a capacidade dos veículos e o local das instituições. Este fluxo assegura o acompanhamento das entregas, priorizando eficiência e rastreabilidade.

Figura 7 – Processo de negócio – ‘repositorio\_entregas’ - colocar diagrama BPMN

## **Processo – ‘calculo\_rotas’**

O processo de cálculo de rotas otimiza as entregas, considerando múltiplos destinos. Com base nas ordens de trabalho atribuídas, calcula o percurso mais eficiente, reduzindo custos operacionais e tempos de transporte, garantindo a qualidade do serviço prestado.

Figura 8 – Processo de negócio – ‘calculo\_rotas’ - colocar diagrama BPMN

# **Casos de Uso**

**Casos de Uso**: Estrutura interações.

## **Atores do Sistema**

Tabela 28 – Atores do Sistema

|  |  |
| --- | --- |
| **Ator** | **Descrição** |
| **Administrador** | Gere os utilizadores, define permissões e realiza listagem. |
| **Funcionário** | Acede funcionalidades específicas atribuídas pelos administradores. |
| **Cliente** | Cria e gere o seu próprio perfil, acessando serviços. |
| **Sistema** | Atua automaticamente para registar tentativas de login. |

## **Casos de Uso – gestao\_utilizadores**

**Casos de Uso para o Microserviço gestao\_utilizadores**

A seguir, os casos de uso identificados com base nos requisitos funcionais, incluindo **extend** e **include** onde aplicável:

**1. Criar Utilizador**

**Descrição:** Permite que o administrador crie novos utilizadores com dados básicos e permissões.

**Ator Principal:** Administrador.

**Fluxo Principal:**

O administrador preenche os dados obrigatórios para criação do utilizador.

O sistema valida os dados e verifica unicidade do e-mail.

A password é criptografada e os dados são salvos no banco.

Retorna sucesso.

**Inclui:**

Verificar unicidade do e-mail.

Criptografar password.

**Restrição:** Apenas administradores podem acessar este caso de uso.

**2. Autenticação de Utilizador (Login)**

**Descrição:** Permite que qualquer utilizador (cliente, funcionário ou administrador) faça login no sistema.

**Ator Principal:** Utilizador.

**Fluxo Principal:**

O utilizador fornece e-mail e password.

O sistema valida as credenciais e gera um token JWT.

Registra o status do login.

Retorna o token para o utilizador.

**Inclui:**

Validar credenciais.

Gerar token JWT.

Registrar status do login.

**Extende:** Registrar Status de Login.

**Restrição:** A password deve ser criptografada e comparada com a base de dados.

**3. Editar Permissões**

**Descrição:** Permite que o administrador altere os papéis associados a um utilizador.

**Ator Principal:** Administrador.

**Fluxo Principal:**

O administrador seleciona um utilizador e define as novas permissões.

O sistema valida se o utilizador existe e os papéis são válidos.

As permissões são atualizadas.

Retorna sucesso.

**Restrição:** Apenas administradores podem acessar este caso de uso.

**4. Listar Utilizadores**

**Descrição:** Permite que o administrador liste todos os utilizadores registrados.

**Ator Principal:** Administrador.

**Fluxo Principal:**

O administrador solicita a lista de utilizadores.

O sistema retorna uma lista com os dados e permissões.

**Restrição:** Apenas administradores podem acessar este caso de uso.

**5. Registrar Status de Login**

**Descrição:** Rastreia o status de cada tentativa de login no sistema.

**Ator Principal:** Sistema.

**Fluxo Principal:**

O sistema registra a tentativa de login (sucesso ou falha) após validação de credenciais.

Os dados são salvos na tabela de registros.

**Extende:** Autenticação de Utilizador.

**Restrição:** Nenhuma.

**6. Atualizar Perfil do Utilizador**

**Descrição:** Permite que um utilizador autenticado atualize suas informações pessoais.

**Ator Principal:** Utilizador Autenticado.

**Fluxo Principal:**

O utilizador solicita a atualização de dados (nome, email, password).

O sistema valida a autenticidade do utilizador e a unicidade do email.

Os dados atualizados são salvos.

Retorna sucesso.

**Inclui:**

Validar unicidade do email.

Criptografar nova password (se alterada).

**Restrição:** Apenas o próprio utilizador pode atualizar seus dados.

**7. Enviar Notificações**

**Descrição:** Permite que o sistema envie notificações para utilizadores via RabbitMQ.

**Ator Principal:** Sistema.

**Fluxo Principal:**

Um evento dispara a geração de uma notificação.

O sistema valida o utilizador e a mensagem.

Envia a notificação via RabbitMQ.

**Inclui:**

Validar id do utilizador.

Enviar mensagem via RabbitMQ.

**Restrição:** Nenhuma.

**8. Validar Token JWT**

**Descrição:** Valida tokens JWT enviados por outros serviços ou pelo API Gateway.

**Ator Principal:** API Gateway, Outros Microserviços.

**Fluxo Principal:**

O sistema verifica se o token é válido e não expirou.

Valida se o token pertence a um utilizador existente.

Retorna sucesso ou erro de autenticação.

**Inclui:**

Decodificar e validar token JWT.

**Restrição:** Nenhuma.

**9. Excluir Utilizador**

**Descrição:** Permite que o administrador exclua um utilizador do sistema, removendo dependências.

**Ator Principal:** Administrador.

**Fluxo Principal:**

O administrador solicita a exclusão de um utilizador.

O sistema verifica se o id do utilizador é válido.

Remove os dados do utilizador e suas dependências.

Retorna sucesso.

**Restrição:** Apenas administradores podem acessar este caso de uso.

**10. Recuperar Informações do Utilizador**

**Descrição:** Permite que o utilizador autenticado recupere suas informações pessoais e permissões.

**Ator Principal:** Utilizador Autenticado.

**Fluxo Principal:**

O utilizador solicita suas informações.

O sistema valida a autenticidade do utilizador.

Retorna os dados do utilizador logado.

**Inclui:**

Validar autenticidade do utilizador.

**Restrição:** Apenas o utilizador autenticado pode acessar suas informações.

**Diagrama de Casos de Uso**

**Inclui (Include):**

Autenticação inclui:

Registrar status do login.

Gerar token JWT.

Atualizar perfil inclui:

Validar unicidade do e-mail.

Criptografar password.

Enviar notificações inclui:

Validar id do utilizador.

**Extende (Extend):**

Registrar status do login estende:

Autenticação de utilizador.

<https://whimsical.com/use-case-diagram-for-user-management-AVXghfGG39cePP3VQ4S8T1>

Uma imagem com texto, captura de ecrã, círculo, lua

Descrição gerada automaticamente

Figura 5 – Casos de uso – ‘gestao\_utilizadores’ – Alterar por motivo notificacoes

## **Casos de Uso – criacao\_menu**

Caso de Uso – Criar novo menu

Caso de Uso – Atualizar menu existente

## **Casos de Uso – verifica\_stock**

### **C**aso de Uso 1 – Listar Ingredientes e Respetivos Níveis de Stock

| **Campo** | **Descrição** |
| --- | --- |
| **Nome** | Listar Ingredientes e Respetivos Níveis de Stock |
| **Ator Principal** | Gestor de Operações |
| **Ator Secundário** | Sistema Automatizado |
| **Objetivo** | Permitir que o gestor visualize os níveis de stock em tempo real. |
| **Resumo** | O sistema exibe uma lista de todos os ingredientes disponíveis com as respetivas quantidades. |
| **Pré-condições** | O utilizador deve estar autenticado com permissões adequadas. |
| **Fluxo Principal** | 1. O gestor acede ao sistema.  2. Seleciona a funcionalidade de monitorização de stock.  3. O sistema apresenta os dados de stock em tempo real. |
| **Fluxo Alternativo** | - Caso o utilizador não tenha permissões adequadas, é exibida uma mensagem de erro. |
| **Pós-condições** | Os níveis de stock ficam visíveis para o gestor. |
| **Restrições** | Apenas utilizadores autorizados podem visualizar os dados. |

### **Caso de Uso 2 – Adicionar Ingredientes ao Stock**

| **Campo** | **Descrição** |
| --- | --- |
| **Nome** | Adicionar Ingredientes ao Stock |
| **Ator Principal** | Funcionário do Armazém |
| **Ator Secundário** | Sistema Automatizado |
| **Objetivo** | Permitir o registo de novos ingredientes no inventário. |
| **Resumo** | O sistema regista a entrada de novos ingredientes e atualiza o inventário. |
| **Pré-condições** | O utilizador deve estar autenticado e autorizado para adicionar ingredientes. |
| **Fluxo Principal** | 1. O funcionário acede à funcionalidade de adição de ingredientes.  2. Insere os dados do ingrediente (nome, quantidade, etc.).  3. O sistema valida os dados e atualiza o inventário. |
| **Fluxo Alternativo** | - Caso os dados do ingrediente sejam inválidos, o sistema exibe uma mensagem de erro. |
| **Pós-condições** | O inventário é atualizado com os novos ingredientes. |
| **Restrições** | Apenas utilizadores autorizados podem registar ingredientes. |

### **Caso de Uso 3 – Obter Alertas de Reabastecimento**

| **Campo** | **Descrição** |
| --- | --- |
| **Nome** | Obter Alertas de Reabastecimento |
| **Ator Principal** | Gestor de Operações |
| **Ator Secundário** | Sistema Automatizado |
| **Objetivo** | Notificar automaticamente o gestor quando o stock de algum ingrediente atinge níveis críticos. |
| **Resumo** | O sistema verifica periodicamente o stock e gera alertas para itens com quantidade abaixo do limite mínimo. |
| **Pré-condições** | O sistema deve estar configurado com limites mínimos para cada ingrediente. |
| **Fluxo Principal** | 1. O sistema executa uma verificação automática dos níveis de stock.  2. Identifica ingredientes abaixo do limite.  3. Gera uma notificação para o gestor. |
| **Fluxo Alternativo** | - Caso não existam ingredientes com stock baixo, nenhuma ação é tomada. |
| **Pós-condições** | O gestor recebe os alertas e pode tomar medidas corretivas. |
| **Restrições** | Apenas utilizadores com permissões podem aceder aos alertas. |

### **Caso de Uso 4 – Atualizar Stock Após Consumo**

| **Campo** | **Descrição** |
| --- | --- |
| **Nome** | Atualizar Stock Após Consumo |
| **Ator Principal** | Sistema Automatizado |
| **Ator Secundário** | API de Integração com Microserviços |
| **Objetivo** | Deduzir automaticamente o nível de stock conforme os ingredientes são utilizados. |
| **Resumo** | Sempre que um menu é preparado, o sistema deduz a quantidade correspondente do stock. |
| **Pré-condições** | A integração entre os microserviços deve estar configurada e funcional. |
| **Fluxo Principal** | 1. O sistema regista um menu preparado.  2. Deduz as quantidades de ingredientes utilizados.  3. Atualiza os dados no inventário. |
| **Fluxo Alternativo** | - Caso haja falha na comunicação com a base de dados, o sistema exibe um erro e não realiza a atualização. |
| **Pós-condições** | Os dados do inventário refletem o consumo atual em tempo real. |
| **Restrições** | Apenas microserviços autorizados podem realizar atualizações no stock. |

### **Caso de Uso 5 – Validar Stock Antes da Criação de Menu**

| **Campo** | **Descrição** |
| --- | --- |
| **Nome** | Validar Stock Antes da Criação de Menu |
| **Ator Principal** | API de Integração com Microserviços |
| **Ator Secundário** | Sistema Automatizado |
| **Objetivo** | Garantir que os ingredientes necessários para um menu estão disponíveis antes de criá-lo. |
| **Resumo** | O sistema valida os níveis de stock necessários antes de autorizar a criação de novos menus. |
| **Pré-condições** | O menu deve estar devidamente configurado com as quantidades necessárias de cada ingrediente. |
| **Fluxo Principal** | 1. O sistema recebe uma solicitação para criação de menu.  2. Verifica os níveis de stock para todos os ingredientes necessários.  3. Autoriza ou rejeita a criação do menu. |
| **Fluxo Alternativo** | - Caso o stock seja insuficiente, o sistema rejeita a criação do menu e emite um alerta. |
| **Pós-condições** | Apenas menus viáveis são criados. |
| **Restrições** | Apenas serviços autorizados podem realizar a validação. |

### **Caso de Uso 6 – Gerar Relatório de Stock**

| **Campo** | **Descrição** |
| --- | --- |
| **Nome** | Gerar Relatório de Stock |
| **Ator Principal** | Gestor de Operações |
| **Ator Secundário** | Sistema Automatizado |
| **Objetivo** | Gerar relatórios detalhados do fluxo de ingredientes para análise. |
| **Resumo** | O sistema permite ao gestor gerar relatórios consolidados sobre as movimentações de stock. |
| **Pré-condições** | O gestor deve estar autenticado com permissões adequadas. |
| **Fluxo Principal** | 1. O gestor seleciona o intervalo de datas desejado.  2. O sistema consolida os dados de movimentação.  3. Gera o relatório em formato CSV ou PDF. |
| **Fluxo Alternativo** | - Caso não haja dados para o período solicitado, o sistema exibe uma mensagem informativa. |
| **Pós-condições** | O gestor tem acesso ao relatório para análise e planeamento. |
| **Restrições** | Apenas utilizadores autorizados podem gerar relatórios. |

### Diagrama

## **Casos de Uso – repositorio\_entregas**

Caso de Uso – Registar ordem de trabalho

Caso de Uso – Subscrever funcionário a uma ordem

## **Casos de Uso – calculo\_rotas**

Caso de Uso – Calcular rota para entrega única

Caso de Uso – Calcular rota para múltiplas entregas

# **Domain-Driven Design**

* + **Business Logic:** Para regras de negócio.
  + **Infraestrutura:** Para suporte técnico e persistência.
  + **Bounded Context:** Para separação lógica do domínio.

Uma imagem com texto, diagrama, captura de ecrã, file

Descrição gerada automaticamente

Figura 6 – Domain-Driven Design

## **gestao\_utilizadores**

|  |  |  |
| --- | --- | --- |
| **Categoria** | **Descrição** | **Subtítulo** |
| **Core Domain** |  |  |
| **Web API** |  |  |
| **Business Logic** |  |  |
| **Infraestrutura** |  |  |
| **Linguagem Ubíqua** |  |  |

**Bounded Context (Contextos Delimitados)**

|  |  |
| --- | --- |
| **Contexto** | **Descrição** |
| **Core Domain** |  |
| **Agregados** |  |
|  |  |
|  |  |
| **Entidades** |  |
| **Web API** |  |
| **Controllers** |  |
| **DTOs** |  |
| **Linguagem Ubíqua** |  |

**Modelação de Entidades e Agregados**

|  |  |
| --- | --- |
| **Elemento** | **Descrição** |
| **Infraestrutura** |  |
| **Repositories** |  |
| **Configurations** |  |
| **Auditing** |  |
| **Models** |  |
| **Util** |  |
| **Listeners** |  |
| **Messages** |  |
| **Data** |  |

**Business Logic (Serviços do Domínio)**

|  |  |  |
| --- | --- | --- |
| **Elemento** | **Descrição** | |
| **Domínio** |  | |
| **Serviços do Domínio** |  | |
|  |  | |
| **Aplicação** |  | |
| **Serviços de Aplicação** |  | |
| **Exceptions** |  | |
|  |  |

## **criacao\_menu**

[**https://medium.com/cwi-software/domain-driven-design-do-in%C3%ADcio-ao-c%C3%B3digo-569b23cb3d47**](https://medium.com/cwi-software/domain-driven-design-do-in%C3%ADcio-ao-c%C3%B3digo-569b23cb3d47)

**Domain-Driven Design (DDD)**: Estrutura o domínio em entidades e serviços.

1. Bounded Context (Contextos Delimitados)
2. Modelação de Entidades e Agregados
3. Business Logic (Serviços do Domínio)

## **verifica\_stock**

1. Bounded Context (Contextos Delimitados)
2. Modelação de Entidades e Agregados
3. Business Logic (Serviços do Domínio)

## **repositorio\_entregas**

1. Bounded Context (Contextos Delimitados)
2. Modelação de Entidades e Agregados
3. Business Logic (Serviços do Domínio)

## **calculo\_rotas**

1. Bounded Context (Contextos Delimitados)
2. Modelação de Entidades e Agregados
3. Business Logic (Serviços do Domínio)

# **Modelos de Dados**

Os modelos de dados organizam e estruturam as informações do projeto. Em microserviços, cada serviço tem um modelo de dados independente, adaptado às suas necessidades específicas. Esta abordagem promove consistência, escalabilidade e modularidade, utilizando bases de dados relacionais ou não relacionais. O capítulo apresenta os esquemas que suportam as funcionalidades do projeto **FoodStream**.

## **gestao\_utilizadores**

O modelo de dados do microserviço **gestao\_utilizadores** é implementado em PostgreSQL, e utiliza relações **one-to-many** e **many-to-many** para suportar a gestão de utilizadores, autenticação e notificações. As tabelas principais, como **Utilizadores**, **Roles**, e **UtilizadoresRoles**, promovem modularidade e integridade referencial. Relações **many-to-many**, como entre utilizadores e roles, são essenciais para a flexibilidade na atribuição de permissões em arquiteturas de microserviços.

Uma imagem com texto, software, Página web, Ícone de computador

Descrição gerada automaticamente

Figura 7 – Modelo de dados ‘gestao\_utilizadores’

## **criacao\_menu**

## **verifica\_stock**

Uma imagem com texto, software, Ícone de computador, Sistema operativo

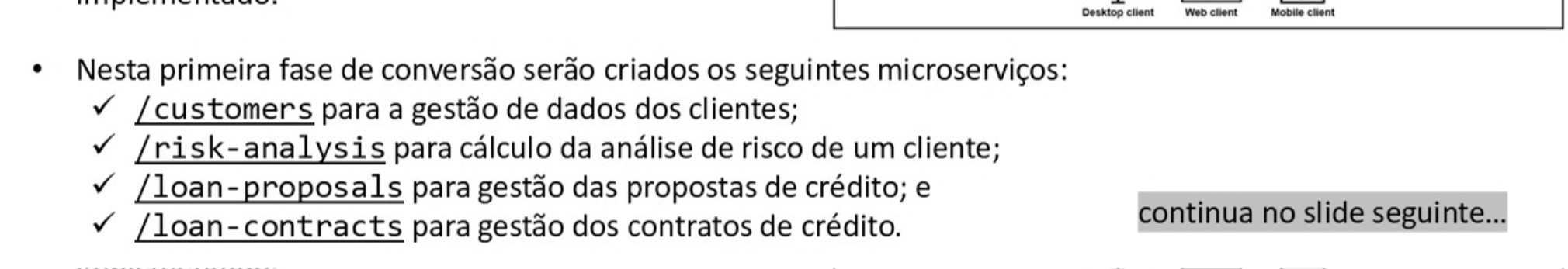
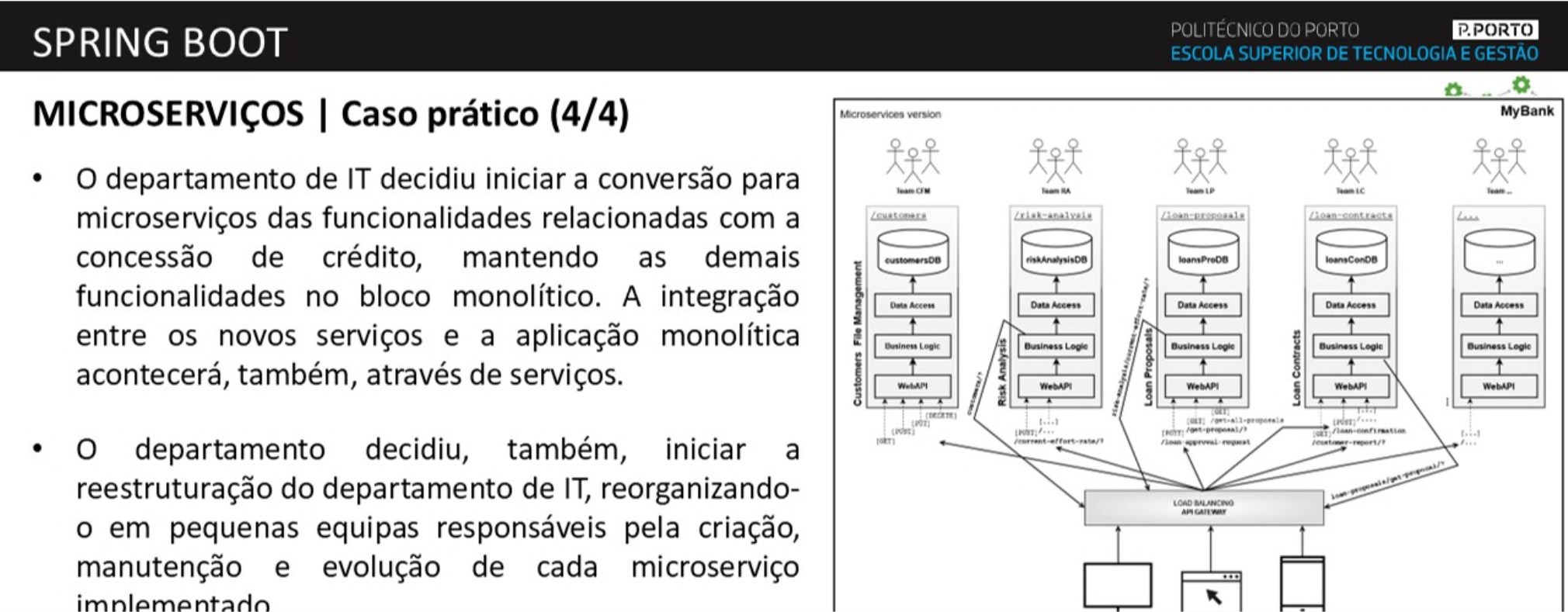
Descrição gerada automaticamente

Figura 8 – Modelo de dados ‘verifica\_stock’

## **repositorio\_entregas**

## **calculo\_rotas**

# **Arquitetura do Sistema**



# **Componentes de Suporte**

**Componentes de Suporte**: Eureka, Gateway, RabbitMQ.

## **/eureka-server (Service Discovery)**

O Eureka Server é um serviço de registo e descoberta de serviços utilizado em arquiteturas de microserviços. Ele possibilita que os serviços **menu**, **stock**, **repositorio\_entregas**, **calculo\_rotas** e **funcionarios** se registem de forma dinâmica e descubram outros serviços durante o tempo de execução, sem necessidade de configurações estáticas.

O Eureka Server é implementado com o framework Spring Boot, utilizando a dependência **spring-cloud-starter-netflix-eureka-server**. Os microserviços clientes configuram o endereço do servidor para registo automático e consulta dos serviços disponíveis.

O objetivo principal é facilitar a comunicação entre microserviços numa arquitetura distribuída, garantindo:

* Alta disponibilidade;
* Balanceamento de carga;
* Resiliência, por meio da monitorização do estado dos serviços registados.

Sem um serviço de descoberta, seria necessário configurar manualmente os endereços IP e as portas entre os serviços, o que é pouco escalável e sujeito a erros. O Eureka Server automatiza este processo, melhorando a manutenção e a escalabilidade da aplicação.

A gestão do Eureka Server é pela **equipa de infraestrutura e suporte do sistema FoodStream**, funcionando como um **hub central** para todos os microserviços da arquitetura.

Tabela 29 – Componente de suporte – /eureka-server

|  |  |
| --- | --- |
|  | **/eureka-server** |
| **Ator:** | Serviços de backend (menu, stock, repositorio\_entregas, calculo\_rotas e funcionarios). |
| **Ator Responsável:** | Equipa de infraestrutura FoodStream. |
| **Microserviço:** | /eureka-server |
| **Descrição:** | Registo e descoberta de serviços, monitorização do estado dos serviços. |
| **Objetivo:** | Facilitar a comunicação dinâmica e escalável entre microserviços. |
| **Verificações:** | - Registo automático de serviços;  - Atualização dinâmica de registos;  - Estado "UP" (em funcionamento) dos serviços. |
| **HTTP Codes:** | - 200 OK: Requisição bem-sucedida;  - 404 NOT FOUND: Serviço não encontrado;  - 500 INTERNAL SERVER ERROR: Falha no servidor. |
| **Restrições:** | - Alta disponibilidade para evitar falhas em cascata;  - Configurações adequadas de rede e autenticação. |

O papel do Eureka Server no sistema, atua como um **catálogo de serviços** centralizado na arquitetura de microserviços do FoodStream. Ele permite que cada serviço conheça e se conecte a outros dinamicamente, eliminando dependências de endereços fixos ou configurações rígidas.

## **/api-gateway (Spring Cloud Gateway)**

### **Rotas**

* route("nome\_logico", ...):
* nome\_logico → Nome da rota que será registado internamente no Gateway.
* path("/<endpoint>/\*\*"):
* Especifica o caminho de entrada para aceder o microserviço.
* uri("http://localhost:<porta>"):
* Redireciona as requisições ao microserviço que está na porta definida.

Tabela 29 – Rotas – Serviços subjacentes

|  |  |
| --- | --- |
| **Rota (api-gateway) localhost:8070** | **Serviços Subjacentes** |
| [utilizadores-route] Path=/utilizadores/\*\* | localhost:8071/utilizadores |
| [menu-route] Path=/menu/\*\* | localhost:8072/menu |
| [stock-route] Path=/stock/\*\* | localhost:8073/stock |
| [repositorio-route] Path=/repositorio/\*\* | localhost:8074/repositorio |
| [rotas-route] Path=/rotas/\*\* | localhost:8075/rotas |

### **RestTemplateConfig**

 **@Configuration**:

* Define que esta classe contém configurações de beans para o Spring Context.

 **@Bean**:

* Cria uma instância única (singleton) do **RestTemplate**, que é Gereda pelo container do Spring.

 **RestTemplate**:

* Uma classe do Spring que facilita a realização de requisições HTTP para outros serviços REST.

## **RabbitMQ (Message Broker)**

# **Microserviços**

**Microserviços**: Implementação e lógica de negócios.

Tabela 31 – Base de Dados dos Microserviços, acessos

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Serviço BD** | **Microserviço** | **Base de Dados** | **Username** | **Password** |
| PostgreSQL | **gestao\_utilizadores** | utilizadores | utilizadores | utilizador |
|  | **criacao\_menu** |  |  |  |
|  | **verifica\_stock** |  |  |  |
|  | **repositorio\_entregas** |  |  |  |
|  | **calculo\_rotas** |  |  |  |

## **Microserviço – gestao\_utilizadores**

### **Descrição**

O microserviço **gestao\_utilizadores** gere perfis de utilizadores, autenticação e autorização, assegurando a criação, atualização e eliminação de contas para clientes, funcionários e administradores. A sua arquitetura modular, seguindo a abordagem DDD, promove escalabilidade e separação de responsabilidades. É um componente central na gestão de utilizadores, permissões e notificações, integrando endpoints para autenticação e rastreamento de atividades.

### **Endpoints (API REST)**

Tabela 28 – Endpoints – gestao\_utilizadores

|  |  |  |
| --- | --- | --- |
| **RF** | **Descrição do RF** | **Endpoints** |
| **RF1** | Criar Utilizador | POST /api/utilizadores |
| **RF2** | Autenticação de Utilizador (Login) | POST /api/auth/login |
| **RF3** | Editar Permissões | PUT /api/utilizadores/{id}/roles |
| **RF4** | Listar Utilizadores | GET /api/utilizadores |
| **RF5** | Registar Status de Login | POST /api/auth/login (automático) |
| **RF6** | Atualizar Perfil do Utilizador | PUT /api/utilizadores/{id} |
| **RF7** | Enviar Notificações | POST /api/notificacoes |
| **RF8** | Validar Token JWT | GET /api/auth/validate |
| **RF9** | Excluir Utilizador | DELETE /api/utilizadores/{id} |
| **RF10** | Recuperar Informações do Utilizador | GET /api/utilizadores/me |

### **Tecnologias e Ferramentas Utilizadas**

### **Package models**

O package **models** implementa as entidades do domínio utilizando JPA. Cada classe reflete as regras de negócio, como Utilizador para autenticação, Role para permissões e RegistoLogins para auditoria. As relações @ManyToOne e @ManyToMany estruturam a lógica de associação entre utilizadores e permissões, enquanto enums como UtilizadorRule definem papéis fixos no sistema.

* **Utilizador.java**: Representa utilizadores, integrando autenticação com o Spring Security e relações com permissões e logins.
* **Role**: Define papéis (Administrador, Funcionário e Cliente) para autorização, alinhada aos requisitos de gestão de permissões.
* **UtilizadoresRoles:** Mapeia relações muitos-para-muitos entre utilizadores e papéis, assegurando flexibilidade.
* **RegistoLogins:** Audita logins, registando status e timestamps.
* **Notificacoes:** Garante rastreabilidade no envio de mensagens associadas a utilizadores.
* **Enum UtilizadorRule:** Uniformiza papéis no sistema, simplificando lógica de validação e autorização.

### **Package repositories**

Para implementar os repositórios no microserviço gestao\_utilizadores, é necessário criar interfaces do tipo public interface no pacote repository. Essas interfaces irão herdar de JpaRepository, fornecendo acesso às operações CRUD e integração com o JPA. O **package repository** desempenha um papel crucial na abordagem DDD ao separar a lógica de persistência de dados do restante do sistema. Ele encapsula as interações com a base de dados, promovendo modularidade e garantindo que as regras de negócio permaneçam isoladas na camada de domínio.

O package repository organiza as interfaces para acesso aos dados, seguindo a abordagem DDD:

* **UtilizadorRepository**: Gestão de dados de utilizadores.
* **RoleRepository**: Gestão de papéis no sistema.
* **UtilizadoresRolesRepository**: Relacionamento entre utilizadores e papéis.
* **RegistoLoginsRepository**: Registo de acessos ao sistema.
* **NotificacoesRepository**: Gestão de notificações enviadas.

### **Package dto (data transfer object)**

O package dto é fundamental na arquitetura DDD, separando as entidades de domínio das transferências de dados entre camadas. Ele protege informações sensíveis, adapta os dados às necessidades dos endpoints e garante modularidade e eficiência na comunicação.

Os DTOs implementados, como:

* **LoginDto:** Permite autenticação de utilizadores com email e password, validando as credenciais para gerar token JWT (RF2).
* **UtilizadorDto:** Alinhado aos RF4, RF10, fornece dados essenciais do utilizador (excluindo password) para listagem e exibição das informações pessoais.
* **CriarUtilizadorDto**: Permite criação de utilizadores, garantindo que os campos obrigatórios sejam preenchidos e suportando associação de papéis (roles) – RF1.
* **NotificacaoDto:** Fornece dados sobre notificações enviadas, como mensagem e timestamp, para integração com RabbitMQ e exibição aos utilizadores (RF7).
* **RoleDto**: Garante manipulação de papéis (roles), fornecendo informações como nome e identificador para atribuição de permissões (RF3).
* **LoginResponseDto:** Responde ao endpoint de autenticação retornando um token JWT (RF2).
* **AtualizarPerfilDto:** Facilita atualização de informações pessoais (nome, email, password), validando as entradas (RF6).
* **RegistoLoginDto:** Representa dados do status e timestamp de logins, ajudando a auditar tentativas de login (RF5).

Esta estrutura promove segurança e assegura a conformidade com os requisitos funcionais, protege dados sensíveis e promove uma comunicação eficiente no sistema.

### **Package services (aplicação)**

Na arquitetura DDD, o package service centraliza a lógica de negócio, separando as regras de domínio das operações técnicas e garantindo a coesão. Este pacote atua como intermediário entre os repositórios e os controladores, encapsulando funcionalidades como autenticação, gestão de utilizadores e permissões. Cada classe de serviço é projetada para alinhar-se aos requisitos funcionais do sistema.

O package service implementa serviços essenciais, incluindo:

* **UtilizadorService:** Faz a gestão das operações de criação, atualização, listagem e exclusão de utilizadores e valida unicidade de email e utiliza DTOs para encapsular dados (RF1, RF4, RF6).
* **RoleService:** Facilita a busca e manipulação de papéis no sistema (RF3).
* **RegistoLoginsService:** Regista tentativas de login e audita eventos de autenticação (RF5).
* **NotificacaoService:** Envia notificações aos utilizadores e lista mensagens associadas (RF7).
* **TokenService:** Faz a gestão e valida tokens JWT, essencial para autenticação segura (RF8).
* **AuthService:** Realiza autenticação de utilizadores, verificando credenciais e retornando tokens JWT (RF2).

Esta estrutura assegura a modularidade, uma vez que cada serviço encapsula responsabilidades específicas, promovendo manutenção e escalabilidade. Além disso, as funcionalidades implementadas atendem integralmente os requisitos funcionais definidos, garantindo aderência aos objetivos do sistema. Com o uso de injeção de dependência, ModelMapper para conversão de DTOs e a anotação @Service para modularidade, a implementação segue boas práticas de desenvolvimento.

### **Package config (configurações)**

### **Mensagens com RabbitMQ**

### **Integração com o Gateway**

### **Testes**

**Testes**: Garantir qualidade e funcionalidade.

#### **Testes Unitários com Junit**

#### **Testes de Integração com Postman**

#### **Testes de Comunicação entre Microserviços**

## **criacao\_menu**

Para cada microserviço:

### **Descrição**

### **Endpoints (API REST)**

### **Mensagens com RabbitMQ**

### **Integração com o Gateway**

### **Testes**

**Testes**: Garantir qualidade e funcionalidade.

#### **Testes Unitários com Junit**

#### **Testes de Integração com Postman**

#### **Testes de Comunicação entre Microserviços**

## **verifica\_stock**

Para cada microserviço:

### **Descrição**

### **Endpoints (API REST)**

### **Mensagens com RabbitMQ**

### **Integração com o Gateway**

### **Testes**

**Testes**: Garantir qualidade e funcionalidade.

#### **Testes Unitários com Junit**

#### **Testes de Integração com Postman**

#### **Testes de Comunicação entre Microserviços**

## **repositorio\_entregas**

Para cada microserviço:

### **Descrição**

### **Endpoints (API REST)**

### **Mensagens com RabbitMQ**

### **Integração com o Gateway**

### **Testes**

**Testes**: Garantir qualidade e funcionalidade.

#### **Testes Unitários com Junit**

#### **Testes de Integração com Postman**

#### **Testes de Comunicação entre Microserviços**

## **calculo\_rotas**

Para cada microserviço:

### **Descrição**

### **Endpoints (API REST)**

### **Mensagens com RabbitMQ**

### **Integração com o Gateway**

### **Testes**

**Testes**: Garantir qualidade e funcionalidade.

#### **Testes Unitários com Junit**

#### **Testes de Integração com Postman**

#### **Testes de Comunicação entre Microserviços**

# **Contentorização e Orquestração**

**Docker e Kubernetes**: Contentorização e orquestração.

## **Docker – Criação de Imagens**

10.1. Docker: Criação de Imagens

## **Kubernetes (minikube – K8s) – Configuração de Clusters**

10.2. Kubernetes (Minikube): Configuração de Clusters

# **Conclusão**

A proposta apresentada oferece uma solução modular e escalável para os desafios enfrentados pela arquitetura monolítica existente, utilizando as melhores práticas do paradigma de microserviços. A implementação desta arquitetura garante flexibilidade, facilidade de manutenção e adaptação às necessidades futuras.

Resumo dos Resultados

Benefícios da Arquitetura de Microserviços

# **Referências**