Gestão de Clientes

**Relatório de Desenvolvimento de Software**

ISMT – Engenharia de Software 2024-25

Joaquim Barreiros - 141593 - <quim@mist.pt>

Carlos Manuel - 12321 - <carlao@mist.pt>

Índice

[Controlo de Versões 2](#_Toc146485285)

[1. Introdução 3](#_Toc146485286)

[2. Análise de Contexto 4](#_Toc146485287)

[3. Requisitos 5](#_Toc146485288)

[4. Diagramas de Casos de Uso 6](#_Toc146485289)

[5. User Stories 7](#_Toc146485290)

[6. Análise de Domínio / Modelo Entidade-Relação 8](#_Toc146485291)

[7. Análise da Estrutura do Projeto 9](#_Toc146485292)

[8. Apresentação do Projeto Final: "Screenshots" da UI 10](#_Toc146485293)

[9. Conclusão Geral / Reflexão final 11](#_Toc146485294)

# 1. Introdução

Este relatório documenta o desenvolvimento de uma aplicação web baseada em Spring Boot e Vaadin, que inclui funcionalidades de autenticação, visualização de utilizadores e administração de dados. O projeto foi organizado de forma modular e segue boas práticas de engenharia de software.

Este relatório inclui as seguintes secções:

* **Análise de Contexto:** define o problema e a motivação do projeto.
* **Requisitos:** enumera os requisitos funcionais e não-funcionais.
* **Casos de Uso:** diagrama das principais interações do utilizador com o sistema.
* **User Stories:** exemplos realistas de funcionalidades.
* **Análise de Domínio:** explicação das entidades e relações do sistema.
* **Estrutura do Projeto:** arquitetura e divisão modular do código.
* **Screenshots:** imagens ilustrativas da interface gráfica.
* **Conclusão:** reflexão sobre o trabalho desenvolvido.

# 2. Análise de Contexto

O projeto simula um sistema de gestão para uma pequena empresa ou instituição que precisa de gerir dados de utilizadores com perfis distintos (admin, user). A aplicação pode ser usada por uma organização para gerir colaboradores, dados pessoais e acessos administrativos.

A motivação prende-se com a necessidade de um sistema simples, acessível via navegador, com interface moderna e que permita gerir dados com segurança e facilidade de utilização.

# 3. Requisitos

**Requisitos Funcionais**  
Requisitos que definem o *comportamento* esperado do sistema.

Tabela 1. Requisitos Funcionais

|  |  |  |
| --- | --- | --- |
| # | Descrição | Prioridade |
| RF01 | O utilizador pode fazer login com email e password | Alta |
| RF02 | O utilizador autenticado pode visualizar dados pessoais | Alta |
| RF03 | O administrador pode gerir utilizadores (CRUD) | Alta |
| RF04 | O sistema deve permitir filtros nos dados apresentados | Média |

**Requisitos Não-funcionais**  
Requisitos que definem *características de qualidade* do sistema.

|  |
| --- |

Tabela 2. Requisitos Não-funcionais

|  |  |  |
| --- | --- | --- |
| # | Descrição | Prioridade |
| RNF01 | A aplicação deve estar disponível em navegadores modernos | Alta |
| RNF02 | O sistema deve garantir autenticação segura | Alta |
| RNF03 | O tempo de resposta deve ser inferior a 2 segundos | Média |

# 

# 4. Diagramas UML

**4.1. Diagrama de Casos de Uso**

Uma imagem com texto, diagrama, file, Tipo de letra

Os conteúdos gerados por IA poderão estar incorretos.

**🛠️ Ferramenta recomendada: Visual Paradigm Online**

* Link direto: https://online.visual-paradigm.com/diagrams.jsp
* É gratuito (sem login obrigatório) e permite fazer diagramas de Casos de Uso de forma visual.
* **Como usar:**
  1. Escolher “Use Case Diagram”.
  2. Arrastar o ator (figura humana) para o lado esquerdo.
  3. Arrastar ovais para representar os casos de uso.
  4. Ligar com setas simples.

**💡 Exemplo para este projeto:**

**Atores:**

* Utilizador
* Administrador

**Casos de uso:**

* Fazer Login
* Visualizar Dados
* Editar Perfil
* Gerir Utilizadores (CRUD)

**⚡ Alternativa ultra-rápida: usar scripts com PlantUML**

Se os alunos preferirem **escrever texto e gerar o diagrama automaticamente**, podem usar este site:

🔗 Gerador online: <https://www.plantuml.com/plantuml/uml/>

**Exemplo de script para copiar e colar:**

Ir a <https://www.plantuml.com/plantuml/uml/>

(se necessário, clicar em "Refresh" para gerar o diagrama automaticamente)

@startuml

left to right direction

skinparam packageStyle rectangle

actor "Utilizador" as U

actor "Administrador" as A

A --|> U ' Admin herda de Utilizador

package "Sistema de Gestão" {

usecase "Fazer Login" as UC1

usecase "Visualizar Dados" as UC2

usecase "Editar Perfil" as UC3

usecase "Gerir Utilizadores" as UC4

}

U --> UC1

U --> UC2

U --> UC3

A --> UC4

@enduml

**4.2. Diagrama de Classes:**

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

Os conteúdos gerados por IA poderão estar incorretos.

O Spring Boot promove uma “POO assistida”, ajudando-nos a estruturar o código com clareza e boas práticas — como a separação de responsabilidades e o baixo acoplamento — sem recorrer à herança na lógica central, o que torna o sistema mais flexível, modular e fácil de manter.

🔧 **Sobre o uso de relações de herança, realça-se que:**

* A herança está a ser usada apenas para **fins técnicos e reutilização estrutural**, e **não para organizar a lógica de negócio**.
* As classes de serviço **não herdam umas das outras**, e isso é muito bom: promove **composição sobre herança**, um dos princípios mais modernos e recomendados da POO.

Na página seguinte está o código que permite “extrair” as relações nucleares do projeto, e fazer um diagrama de classes a partir dessas relações.

O site usado foi o mesmo (é só copiar e colocar lá: pode ser preciso fazer “refresh”)  
<https://www.plantuml.com/plantuml/uml/>

@startuml

skinparam classAttributeIconSize 0

class Application {

+main(String[]): void

}

class User {

-id: Long

-username: String

-name: String

-email: String

-role: Role

-passwordHash: String

}

class SamplePerson {

-id: Long

-firstName: String

-lastName: String

-email: String

}

interface UserRepository

interface SamplePersonRepository

class UserService {

-userRepository: UserRepository

+getAllUsers(): List<User>

+saveUser(User): void

}

class SamplePersonService {

-samplePersonRepository: SamplePersonRepository

+getAllSamplePersons(): List<SamplePerson>

}

Application --> UserService

Application --> SamplePersonService

UserService --> UserRepository

SamplePersonService --> SamplePersonRepository

UserRepository ..|> interface

SamplePersonRepository ..|> interface

UserService --> User

SamplePersonService --> SamplePerson

@enduml

# 5. User Stories

Como **Utilizador**, pretendo **fazer login** de modo a **aceder à minha área pessoal**.

Como **Administrador**, pretendo **visualizar e editar utilizadores** de modo a **gerir o sistema eficazmente**.

Como **Utilizador**, pretendo **ver os meus dados num quadro com filtros** de modo a **encontrar facilmente a informação**.

# 6. Análise de Domínio / Modelo Entidade-Relação

O domínio principal inclui duas entidades:

* **User**: com atributos id, username, name, email, role, passwordHash.
* **SamplePerson**: entidade de exemplo para demonstração, com dados fictícios.

Estas entidades estão armazenadas em tabelas relacionais, persistidas via JpaRepository. A existência da entidade Role como enum garante consistência nos tipos de utilizador (e.g. ADMIN, USER).

# 7. Análise da Estrutura do Projeto

O projeto segue a arquitetura típica Spring Boot:

* application: contém a classe principal Application.java.
* data: entidades (User, SamplePerson) e repositórios.
* security: gestão de autenticação e configuração de segurança.
* services: lógica de negócio (ex: UserService, SamplePersonService).
* views: interface gráfica em Vaadin com páginas como LoginView, GridwithFiltersView, VisaoAdminView.
* resources: ficheiros de configuração e dados iniciais (data.sql).

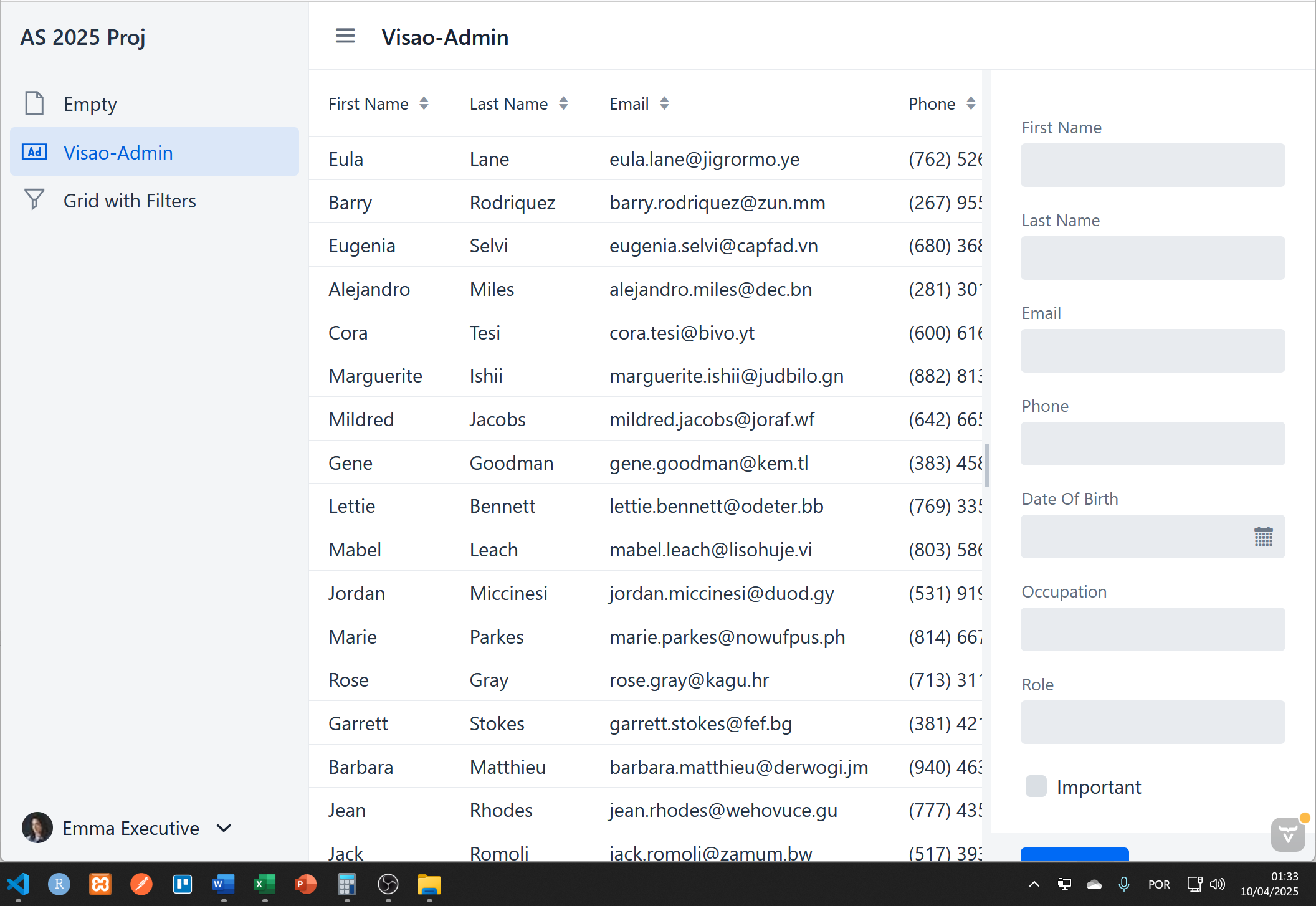
# 8. Apresentação do Projeto Final: "Screenshots" da UI

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Os conteúdos gerados por IA poderão estar incorretos.

Uma imagem com texto, captura de ecrã, software, Sistema operativo

Os conteúdos gerados por IA poderão estar incorretos.



Uma imagem com texto, captura de ecrã, software, Ícone de computador

Os conteúdos gerados por IA poderão estar incorretos.

# 9. Conclusão Geral / Reflexão final

Este projeto permitiu consolidar conhecimentos de desenvolvimento web com Java, integrando frontend e backend de forma eficiente com Vaadin. A segurança via Spring Security foi uma das maiores aprendizagens, especialmente na gestão de autenticação e perfis.

Aprendemos também a importância de estruturar bem o código e dividir responsabilidades em camadas bem definidas. A utilização de frameworks modernas e boas práticas (como injeção de dependências e MVC) torna o projeto facilmente extensível.

Foi uma excelente oportunidade para aplicar técnicas de engenharia de software num contexto prático, envolvendo análise de requisitos, modelação de dados e implementação funcional.