EnergySales

Martim Parente

Orientador: Nuno Leite

Relatório Inicial realizado no âmbito de Projeto e Seminário Licenciatura em Engenharia de Informática e de Computadores

Junho de 2024

Resumo

Este projeto tem como objetivo desenvolver uma aplicação web para a gestão de equipas de vendas dentro de uma empresa. Neste projeto em concreto o produto / serviço serão contratos de energia elétrica, mas é considerada a possibilidade de ser adaptada para ter qualquer produto / serviço dependendo das necessidades

A aplicação permitirá que os administradores gerenciem equipas de comerciais e os preços que serão praticados no mercado. Cada comercial poderá registar e fazer propostas aos seus clientes.

Incluirá funcionalidades para filtrar produtos e aplicar promoções conforme as necessidades e estratégias de venda. Tanto os administradores como os comerciais poderão utilizar estas ferramentas para aumentar a eficiência operacional e as vendas assim como fornecer aos clientes a possibilidade de fazer simulações e fazer comparativos dos custos atuais com os propostos.

A aplicação será desenvolvida com uma interface *web* de fácil utilização, assegurando que todos os utilizadores (administradores, comerciais e clientes) possam utilizar as funcionalidades que lhes compete.

Palavras-Chave: Aplicação Web, Gestão, Equipas, Comerciais, Clientes, Contratos, Preços, Propostas.

Lista de Acrónimos

API Application Programming Interface

BD Base de Dados

DAO Data Access Object

HTTP Hypertext Transfer Protocol

JSON JavaScript Object Notation

REST Representational State Transfer

UI User Interface

URL Uniform Resource Locator

WWW World Wide Web

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

Com o avanço constante da digitalização e a crescente necessidade de eficiência nas operações empresariais, torna-se imperativo adotar soluções tecnológicas que facilitem a gestão e organização dessas mesmas operações. Este projeto insere-se neste contexto, apresentando o desenvolvimento de uma *Single Page Application* (SPA) suportada por um *backend* constituído por uma *Representational State Transfer* (REST) *Application Programming Interface* (API) destinada a otimizar a gestão de vendas de uma empresa.

A aplicação permite que administradores ou empresas criem e administrem equipas de comerciais. Cada comercial terá a capacidade de registar e acompanhar os seus próprios clientes, promovendo um acompanhamento mais próximo e personalizado. Esta funcionalidade é essencial para garantir que cada cliente recebe a atenção necessária, contribuindo para a fidelização e satisfação do mesmo.

Adicionalmente, tanto os administradores como os comerciais podem filtrar produtos e ajustar preços de acordo com as suas estratégias de venda. Este sistema de filtragem e ajuste de preços visa maximizar os lucros, permitindo uma maior flexibilidade e adaptabilidade às condições do mercado.

A implementação de uma interface intuitiva e de fácil utilização assegura que todos os utilizadores possam navegar e utilizar as funcionalidades sem dificuldades. A API de *backend* robusta garante que a aplicação seja rápida, segura e escalável, preparada para atender às exigências do mercado.

O presente relatório encontra-se organizado em cinco capítulos. Nos dois primeiros, é dado a conhecer o problema que este projeto pretende resolver, as aplicações existentes no mercado que partilham a mesma finalidade e os requisitos funcionais da aplicação. No terceiro capítulo, são descritas as tecnologias usadas, a arquitetura da aplicação e das classes. No quarto capítulo é abordada a implementação das funcionalidades desenvolvidas. Por fim, no último capítulo, descreve-se a discussão de resultados, conclusões do trabalho e são indicados pontos de futuras melhorias.

Capítulo 2

Formulação do Problema

Conteúdo

2.1 Estado de Arte

No mercado existem diversas aplicações que têm como finalidade a gestão de equipas de vendas, clientes e produtos. Estas aplicações intituladas de *Customer Relationship Management* (CRM), permitem às empresas serem mais organizadas e eficientes.

Algumas aplicações presentes no mercado são:

- *Salesforce* - Plataforma de CRM que oferece funcionalidades para gestão de vendas, atendimento ao cliente, marketing e análise de dados. Inclui personalização extensiva, automação de processos, gestão de clientes e contactos, e criação de relatórios detalhados. Suporta integração com diversas aplicações de terceiros, permitindo uma visão holística das operações empresariais.

- *HubSpot* - Plataforma que integra ferramentas para marketing, vendas e atendimento ao cliente. Permite a automatização de campanhas de *marketing*, monitorização de e-mails, gestão de contactos e acompanhamento de clientes. Oferece funcionalidades para criação de relatórios detalhados e análise de desempenho.

- *Zoho CRM* - Outra aplicação que facilita a gestão de vendas, marketing e suporte ao cliente. Inclui automação de processos de vendas, gestão de contactos, análise de desempenho e painéis personalizáveis. Integra-se nativamente com outras ferramentas da *Zoho* e suporta integração com aplicações de terceiros, proporcionando uma visão unificada das operações empresariais.

O mercado de CRM está altamente saturado, com várias plataformas estabelecidas como as referidas acima, entre dezenas de outras, oferecendo um conjunto abrangente de funcionalidades. Estas soluções já permitem a gestão eficiente de vendas, clientes e produtos, automatização de processos, análise de dados e integração com outras ferramentas. Dada esta saturação, as funcionalidades adicionais propostas por esta aplicação não são necessariamente diferenciadoras, uma vez que estas plataformas já cobrem de forma extensa as necessidades das empresas no que toca à gestão de vendas e relacionamento com clientes.

2.2 Funcionalidades

O sistema será usado por vários tipos de utilizador, nomeadamente administradores, comerciais e clientes. Cada um terá acesso a um conjunto de funcionalidades.

- Administradores:
  - Administrar equipas de vendas e respetivos comerciais
  - Criar serviços / produtos energéticos
  - Estipular preços
  - Atribuir promoções por equipa ou localidade
- Comerciais:
  - Adicionar os seus clientes
  - Criar propostas personalizadas para os clientes
- Clientes:
  - Fazer simulações para verificar custos e poupanças
  - Comparativos com contrato atual

Capítulo 3

Abordagem

Conteúdo

Neste capítulo, abordaremos a arquitetura do sistema, o modelo de dados e as tecnologias utilizadas no desenvolvimento do projeto.

3.1 Arquitetura

A arquitetura do sistema foi construída para responder às funcionalidades necessárias que foram descritas no capítulo anterior. Para tal, o primeiro passo foi construir um diagrama de blocos que representasse componentes lógicos e funcionais do sistema. Para além disso, definir como cada bloco comunicaria com os restantes blocos, de modo a ter uma arquitetura estável e escalável. Por fim, para cada componente determinou-se a tecnologia que satisfaria o funcionamento proposto e chegou-se ao diagrama ilustrado na 3.1.

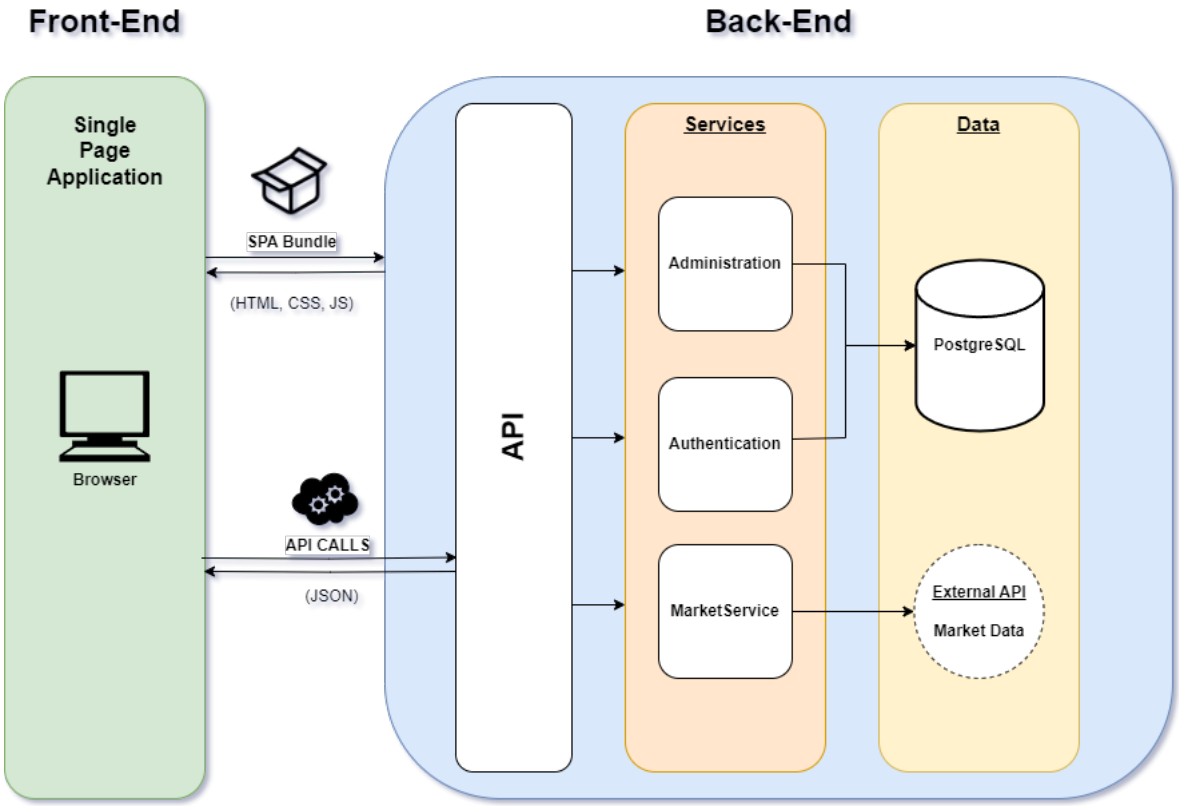

Figura 3.1: Arquitetura da solução

Do lado esquerdo da figura, vemos os dispositivos cliente, que interagem com o sistema através da *Single Page Application* (SPA) desenvolvida em *React*. A aplicação *web* interage diretamente com a *Application Programming Interface* (API) desenvolvida em *Ktor* para obter os dados e fazer ações. A API, componente central do sistema onde se en-

contra toda a lógica de negócio, comunica com todos os outros componentes. Uma dessas interações é com a base de dados *PostgreSQL*, onde são armazenados todos os dados do sistema de forma persistente. A API utiliza o *framework Exposed* para facilitar as operações de leitura e escrita na base de dados. A autenticação e autorização são gerenciadas com recurso a *JSON Web Tokens* (JWT). Este componente verifica a validade dos *tokens* e controla o acesso aos diferentes recursos da aplicação, garantindo segurança nas operações. Para manter a arquitetura modular e facilitar futuras manutenções e substituições de componentes, cada módulo é desenvolvido e implantado de forma independente.

3.2 Modelo de Dados

O modelo entidade-relacionamento inclui as entidades essenciais para representar algumas das funcionalidades apresentadas no Apêndice A.

3.3 Tecnologias

Para o desenvolvimento do projeto, optou-se por uma solução que utiliza uma API desenvolvida em Ktor, uma *framework* para criação de servidores e clientes *Hypertext Transfer Protocol* (HTTP) assíncronos, tirando proveito das corrotinas existentes na linguagem *Kotlin*. A arquitetura modular de *Ktor* facilita a adição de funcionalidades através de *plugins*, como autenticação, sessões, *routing*, monitorização, entre outros. Ktor oferece suporte a múltiplos protocolos, incluindo HTTP e *WebSockets*, e integra-se facilmente com bibliotecas de *Object Relational Mapper* (ORM) para gestão de dados. Para essa gestão de dados foi usado o *Exposed*, uma *framework Structured Query Language* (SQL) construida em cima de *Java Database Connectivity* (JDBC) *driver* para *Kotlin*. A nível de *User Interface* (UI), a aplicação foi construída utilizando a biblioteca *React*, com o suporte das bibliotecas *Mantine* para componentes UI e *Refine* para aplicar algumas soluções para a autenticação.

3.3.1 Frontend

React

React é uma biblioteca *open-source* em *JavaScript* para construir interfaces de utilizador. É mantida pelo *Facebook*, outras empresas e por uma comunidade de desenvolvedores individuais. O *React* pode ser usado para o desenvolvimento de SPA ou aplicações móveis nativas através da *framework React Native*. Esta biblioteca é baseada em compo-

nentes, permitindo que a aplicação seja construída a partir de pequenas peças de código reutilizáveis. Além disso, *React* é declarativo, o que facilita a descrição da interface de utilizador sem a necessidade de especificar como ela deve ser atualizada.

Mantine

Mantine é uma biblioteca *open-source* de componentes *React*, baseada em *TypeScript*, desenvolvida e mantida por Vitaly Rtishchev e mais de 300 contribuidores. Novas funcionalidades e componentes/*hooks* são frequentemente adicionados com base no *feedback* da comunidade.

Oferece uma vasta gama de componentes prontos para uso, altamente personalizáveis através de propriedades e temas. Suporta design responsivo nativamente, assegurando que as interfaces se adaptem a diferentes tamanhos de ecrã. Além dos componentes de UI, inclui utilitários para gestão de estado, manipulação de formulários, notificações e outras funcionalidades essenciais para aplicações web. A documentação abrangente e a comunidade ativa fazem do *Mantine* uma escolha robusta para desenvolvedores *React*.

Refine

Refine é um framework open-source baseada em React, concebida para simplificar o desenvolvimento de ferramentas internas, painéis de administração e dashboards. Integra-se facilmente com vários fornecedores de dados como REST e GraphQL, suporta autenticação e autorização de utilizadores, e inclui componentes UI pré-desenhados e personalizáveis. O Refine também gere o *routing* do lado do cliente com React Router e suporta internacionalização, tornando-o numa solução abrangente para construir aplicações web complexas de forma eficiente.

Vite

Vite é uma ferramenta moderna de construção (*build tool*) e um servidor de desenvolvimento voltado para projetos web. Foi criada por Evan You, o mesmo criador do Vue.js. Através dela proporciona uma experiência de desenvolvimento rápida, permitindo atualizações instantâneas sem necessidade de recompilação completa. Na produção, otimiza os arquivos para melhor desempenho ao processar os módulos com dependências e gerando no final ficheiros estáticos. Embora inicialmente focada no Vue.js, Vite é compatível com vários *frameworks* como React e Svelte.

3.3.2 Backend

Ktor

Ktor é um *framework* assíncrono para a criação de aplicações web e API em Kotlin. Desenvolvido pela JetBrains, Ktor é projetado para ser flexível e modular, permitindo que os desenvolvedores escolham apenas os componentes necessários para o seu projeto. Com

o Ktor, é possível criar servidores e clientes HTTP de forma eficiente, tomando proveito das corrotinas existentes na linguagem *Kotlin* para efetuar operações assíncronas. Oferece uma ampla gama de recursos, incluindo *routing*, autenticação, sessões, acesso a dados, monitorização e suporte a *WebSockets*, entre outros.

Arrow

Arrow é uma biblioteca funcional para Kotlin que oferece uma série de ferramentas para programação funcional como erros tipificados, gerenciamente de corotinas, funções e abstrações. Desenvolvida para ser altamente modular e extensível, Arrow permite que os programadores escrevam código mais conciso, seguro e fácil de manter. A biblioteca inclui suporte para *monads*, *functors*, *applicatives* e muitas outras estruturas de programação funcional. Com uma documentação abrangente e uma comunidade ativa, Arrow é uma escolha robusta para quem deseja aplicar princípios de programação funcional em projetos Kotlin.

Exposed

Exposed é uma biblioteca SQL para Kotlin desenvolvida pela JetBrains, que facilita a interação com bases de dados de forma idiomática e segura. A biblioteca oferece duas API principais: uma *Domain Specific Language* (DSL) e uma *Data Access Object* (DAO), permitindo flexibilidade na escolha do estilo de acesso a dados. A API DSL permite a construção de consultas SQL de forma programática e tipificada, enquanto a API DAO proporciona uma abordagem orientada a objetos para a manipulação de dados. Suporta várias bases de dados, incluindo *PostgreSQL, MySQL, SQLite, Oracle, MariaDB, Microsoft SQL Server e H2*.

PostgresSQL

PostgreSQL é um sistema de gestão de bases de dados relacional de código aberto. Suporta uma ampla gama de tipos de dados, índices avançados, transações *Atomicity, Consistency, Isolation, and Durability* (ACID), replicação e processamento de consultas complexas. É extensível, permitindo a adição de novos tipos de dados, funções e operadores.

Com suporte para *JavaScript Object Notation* (JSON), o *PostgreSQL* facilita a integração com aplicações *NoSQL*. A documentação detalhada e a comunidade ativa tornam o *PostgreSQL* uma escolha fiável para programadores e administradores de bases de dados que necessitam de uma solução versátil.

Capítulo 4

Implementação

Conteúdo

4.1 Ferramentas Utilizadas

Para o desenvolvimento do projeto, foram utilizadas duas ferramentas principais: *IntelliJ IDEA* para o desenvolvimento da *Application Programming Interface* (API) com a *framework* Ktor e o *IntelliJ WebStorm* para a *Single Page Application* (SPA) em React, ambos os programas criados pela empresa *JetBrains*.

4.2 Modelo de Base de Dados

Os dados da aplicação são armazenados de forma persistente utilizando a biblioteca *Exposed* em conjunto com a base de dados *PostgreSQL*. A arquitetura da biblioteca *Exposed* possui três componentes principais:

- Entidades: Representam as tabelas da base de dados e definem o esquema das tabelas. Cada classe de entidade corresponde a uma tabela no PostgreSQL.
- *Data Access Object* (DAO): Fornecem métodos para consultar, atualizar, inserir e apagar dados na base de dados. Eles encapsulam as operações *Create, Read, Update, Delete* (CRUD) e oferecem uma interface de acesso aos dados.
- Transações: Garantem a integridade e consistência dos dados durante as operações de leitura e escrita. Todas as operações na base de dados são realizadas dentro de transações, assegurando que as alterações sejam atómicas e consistentes.

4.3 Autenticação de Utilizador

A autenticação de utilizador no sistema é implementada utilizando *JSON Web Tokens* (JWT) (JSON Web Tokens). Neste sistema, existem apenas administradores e comerciais. As contas de administradores são criadas diretamente no software, enquanto os comerciais têm as suas contas criadas e geridas pelos administradores na aplicação web. Os dados de autenticação são fornecidos diretamente aos respetivos utilizadores, pelo que não existe uma página de registo.

O processo de autenticação e autorização é descrito a seguir.

4.3.1 Processo de Autenticação

- Login: O utilizador (administrador ou comercial) envia as suas credenciais (nome de utilizador e palavra-passe) para a API através de um pedido *Hypertext Transfer*

Protocol (HTTP).

- Validação: A API valida as credenciais contra os dados armazenados na base de dados *PostgreSQL*. Caso as credenciais sejam válidas, é gerado um JWT.
- Geração do JWT: O JWT inclui informações sobre o utilizador autenticado, como o seu identificador e privilégios, sendo assinado digitalmente para garantir a sua integridade e autenticidade.
- Envio do JWT: O JWT é retornado ao utilizador como resposta ao pedido de login.
- Utilização do JWT: O utilizador inclui o JWT no cabeçalho de autorização dos pedidos HTTP subsequentes à API.
- Verificação do JWT: A API verifica a validade e integridade do JWT em cada pedido. Se o JWT for válido, o pedido é processado; caso contrário, o acesso é negado.

4.3.2 Gestão de Sessões e Segurança

Para garantir a segurança e integridade dos dados, são implementadas as seguintes medidas:

Expiração do JWT: O JWT possui um tempo de expiração, após o qual deixa de ser válido, reduzindo o risco de utilização indevida em caso de comprometimento do token.

Renovação de JWT: Antes do JWT expirar, o utilizador pode solicitar a renovação do *token* enviando um pedido apropriado à API.

Revogação de JWT: Em caso de suspeita de comprometimento ou quando um utilizador é desativado, os administradores podem revogar o JWT, invalidando-o imediatamente.

Armazenamento Seguro de Credenciais: As palavras-passe dos utilizadores são armazenadas de forma segura, utilizando o algoritmo de *hashing* SHA-256 para criar uma *hash* + *salt* e após guardar na *Base de Dados* (BD), para evitar compromissos em caso de acesso não autorizado à base de dados.

4.4 Testes unitários

Capítulo 5

Conclusão e Trabalho Futuro

Conteúdo

5.1 Conclusão

5.2 Trabalho futuro

Apêndice A

Modelo Entidade-Associação

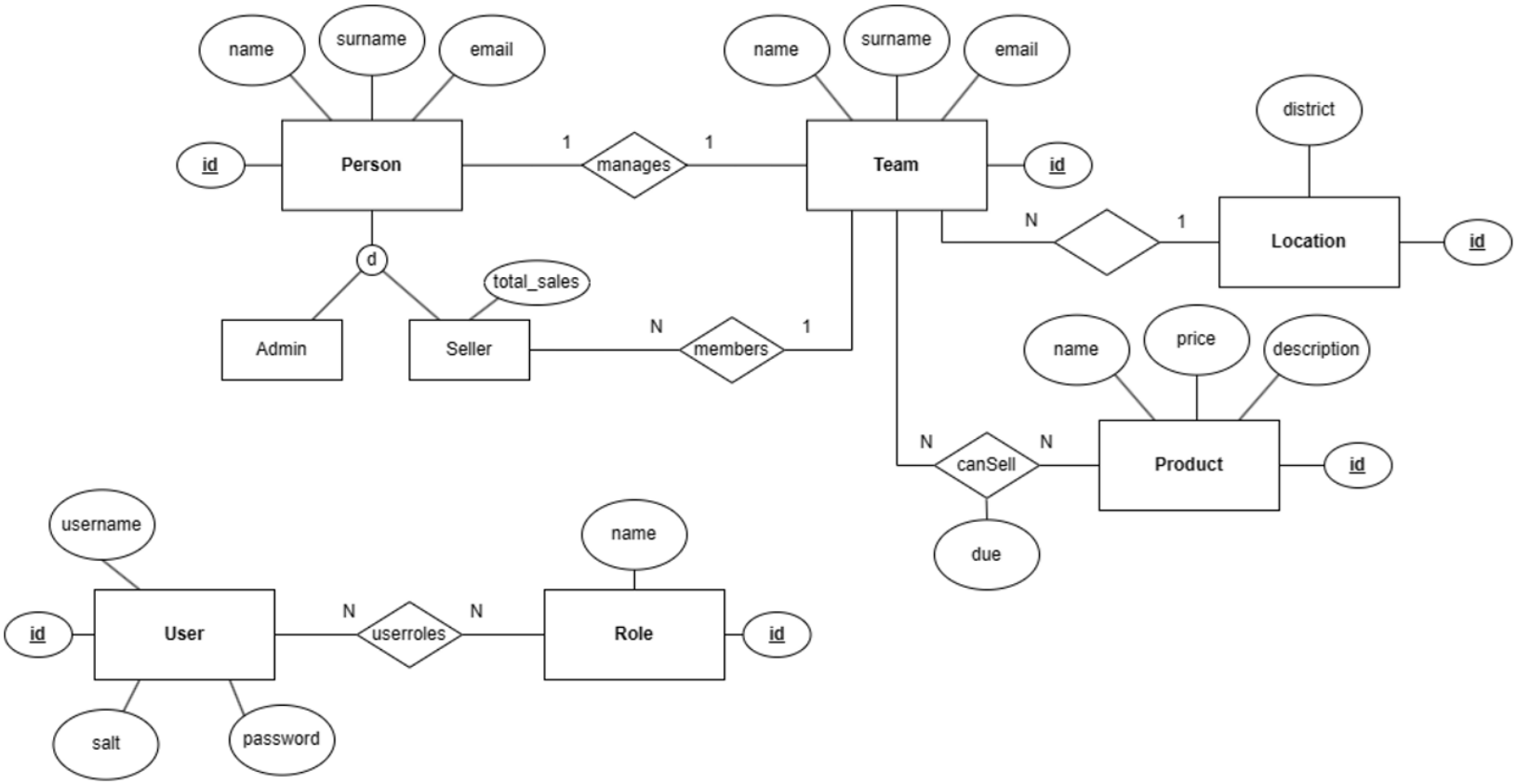

Figura A.1: Modelo Entidade-Associação

Apêndice B

Modelo Relacional

Referências