System Design Document – Gamelist

1. Visão Geral do Sistema

Nome: Gamelist

Descrição: Gamelist é uma API REST que permite a listagem de jogos com a opção de reordenação por posição. O objetivo é oferecer uma interface simples e eficiente para organização de coleções de jogos. Este projeto foi desenvolvido para composição de portfólio.

URL da API em produção: Gamelist Railway

2. Tecnologias Utilizadas

Linguagem: Java 17

Gerenciamento de Dependências: Maven (pom.xml)

Framework: Spring Boot 3.4.2

Módulos do Spring:

- Spring Starter Web
- Spring Data JPA
- Spring Starter Test
- Lombok
- Dev Tools

Banco de Dados:

- **Para testes (test):** H2 (banco em memória, ideal para testes rápidos)
- **Para homologação (dev) e produção (prod):** PostgreSQL 14 via Docker (com PgAdmin para administração)

3. Arquitetura e Estrutura do Projeto

A aplicação segue uma arquitetura baseada em camadas, organizadas nos seguintes pacotes:

- **config:** Contém configurações gerais da aplicação
    - WebConfig: Configuração de CORS, permitindo requisições de origens específicas
- **entities:** Representa as entidades do banco de dados
    - Game
    - GameList
    - Belonging (relação entre Game e GameList)
    - BelongingPK (chave composta da relação Game-GameList)
- **dto:** Define objetos de transferência de dados (Data Transfer Objects)
    - GameDTO
    - GameListDTO
    - GameMinDTO
- **repositories:** Camada de persistência e acesso ao banco
    - GameRepository
    - GameListRepository
- **services:** Contém a lógica de negócios, intermediando a comunicação entre os controllers e os repositories
    - GameService
    - GameListService
- **controllers:** Define os endpoints da API e gerencia requisições
    - GameController
    - GameListController

4. Modelo de Dados

As principais entidades e seus relacionamentos são:

- **Game** (Tabela de jogos, contendo informações como título, ano, URL da imagem, descrição curta, etc.);
- **GameList** (Listas de jogos criadas pelo usuário, permitindo a organização personalizada dos games);
- **Belonging** (Tabela intermediária que define a relação entre Game e GameList, armazenando a posição do jogo na lista);
- **BelongingPK** (Chave composta para a relação entre Game e GameList, garantindo unicidade na associação).

5. Fluxo de Dados e Operações

1. O usuário faz uma requisição para listar jogos.
2. A API recupera os dados do banco de dados via **GameRepository**.
3. Os dados são processados pelo **GameService**, convertidos para DTOs para otimização do retorno.
4. O **GameController** retorna os dados formatados como JSON.
5. Para a reordenação dos jogos, a API atualiza a posição dos jogos na entidade **Belonging**.

6. Perfis de Ambiente (application.properties)

A aplicação possui três perfis de execução:

- **test:** Banco de dados H2 em memória para testes rápidos sem necessidade de configuração externa.
- **dev:** PostgreSQL via Docker, usado para desenvolvimento e homologação.
- **prod:** PostgreSQL em um ambiente de produção real, hospedado na Railway.

Cada perfil tem sua configuração separada no application.properties, garantindo flexibilidade no desenvolvimento e implantação.

7. Configuração de Segurança e CORS

Segurança:

No futuro, pode ser integrado com **Spring Security** para autenticação e autorização.

CORS:

Configurado na classe **WebConfig**, permitindo requisições apenas de origens específicas definidas no application.properties.

8. Endpoints da API

Todos os endpoints são **GET**:

- /games - Lista todos os games
- /games/{id} - Busca um game por ID
- /lists - Lista todas as listas
- /lists/{id} - Busca uma lista por ID
- /lists/{id}/games - Busca todos os games em determinada lista (exemplo: retorna todos os games que estão na lista 1)

9. Implantação

O projeto está implantado na plataforma Railway, facilitando a hospedagem e escalabilidade da API.