Aplicação com Spring Boot

Programação Web II

Exercícios

Exercício 1: Construir uma API REST em Spring Boot seguindo MVC, que consome a PokeAPI v2 e persiste/cacheia dados no H2

Objetivo

Implementar um microserviço que:

1. Busca Pokémon na PokeAPI v2
2. Persiste dados essenciais no H2 (cache local).
3. Expõe endpoints REST para consulta/gerenciamento desses dados.
4. Segue a arquitetura MVC: controller (REST), service (regra/integração), repository (JPA), model (entidades/DTOs).

Requisitos Funcionais

1. Endpoint de importação/cache
   - POST /api/pokemon/cache/{nameOrId}

   - Comportamento:
     - Consulta a PokeAPI (/pokemon/{id|name}), mapeia e salva/atualiza no H2.
     - Retorna o registro persistido.

   - Campos mínimos a persistir:
     - **idPokeApi** (int), **name** (String), **height** (int), **weight** (int),
     - **primeira ability** (String), **lista de types** (String CSV ou relação simples),
     - **cachedAt** (timestamp).

   - Se já existir, **atualiza** os campos e cachedAt.

Requisitos Funcionais

1. Endpoint de importação/cache
   - POST /api/pokemon/cache/{nameOrId}

   - Comportamento:
     - Consulta a PokeAPI (/pokemon/{id|name}), mapeia e salva/atualiza no H2.
     - Retorna o registro persistido.

   - Campos mínimos a persistir:
     - **idPokeApi** (int), **name** (String), **height** (int), **weight** (int),
     - **primeira ability** (String), **lista de types** (String CSV ou relação simples),
     - **cachedAt** (timestamp).

   - Se já existir, **atualiza** os campos e cachedAt.

Requisitos Funcionais

2. Listagem paginada

- GET /api/pokemon?page=0&size=10
- Retorna página com: idLocal, idPokeApi, name, types, cachedAt.

3. Detalhe

- GET /api/pokemon/{idLocal}
- Retorna o registro completo do H2.

4. Busca por tipo

- GET /api/pokemon/search?type={typeName}
- Filtra Pokémon cujo types contenha {typeName} (case-insensitive).

Requisitos Funcionais

5. Favoritar/nota

- PATCH /api/pokemon/{idLocal}/favorite
  - Body: { "favorite": true, "note": "opcional" }
- Atualiza campos favorite (boolean) e note (String).

6. Saúde

- GET /actuator/health (opcional via starter actuator) — OK se preferir ignorar para caber em 2h.

Requisitos Não Funcionais / Técnicos

- Spring Boot (3.x), MVC, Spring Data JPA, H2 (console habilitado), RestTemplate ou WebClient para integrar com PokeAPI.

- **Pacotes sugeridos:**
  - controller/ (REST)
  - service/ (regras e integração PokeAPI)
  - repository/ (JPA)
  - model/ (entities JPA) e dto/ (PokeAPI → DTO)
  - config/ (H2, WebClient/RestTemplate)

- H2 em memória com spring.jpa.hibernate.ddl-auto=update e console (/h2-console) habilitado.

- Mapeamento mínimo dos campos da PokeAPI (não precisa cobrir todas as estruturas).

Requisitos Não Funcionais / Técnicos

- Validações e erros:
  - Pokémon não encontrado na PokeAPI → HTTP 404.
  - idLocal inexistente → HTTP 404.

- Documentação breve no README (como rodar, exemplos de curl).

Critérios de Aceite

- POST /api/pokemon/cache/{nameOrId} consulta PokeAPI e persiste/atualiza no H2.
- GET /api/pokemon lista paginada.
- GET /api/pokemon/{idLocal} detalha registro.
- GET /api/pokemon/search?type=... filtra por tipo.
- PATCH /api/pokemon/{idLocal}/favorite atualiza favorite e note.
- Erros 404 adequados para não encontrados.
- Projeto compila e roda com ./mvnw spring-boot:run ou ./gradlew bootRun.
- README curto (como rodar + exemplos de curl).

Obrigada