Dokumentace EAR

15. prosince 2025

Obsah

1 Popis aplikace

Backend pro správu půjčovny deskových her. Umožňuje správu katalogu her, evidenci fyzických kusů, proces výpůjček a psaní recenzí.

- **Architektura:** Vrstvená (Controller → Service → Repository → Model).
- **Technologie:** Spring Boot, Hibernate (JPA), PostgreSQL.

2 Návod na spuštění

Aplikace se připojuje k externí databázi nastavené v `application.properties`.

2.1 Varianta A: Maven

Spuštění přímo v terminálu v kořenovém adresáři projektu:

1. `mvn clean install` (Sestavení a stažení závislostí)
2. `mvn spring-boot:run` (Spuštění serveru)

nebo

1. `záložka Maven v Intellij Idea` Lifecycle a clean
2. `start aplikace` (spuštění konfigurace BoardGameRentalApplication)

Server běží na: `http://localhost:8080`

2.2 Varianta B: Docker

Pro spuštění v kontejneru stačí vygenerovat jeden image:

1. Vytvoření image:
   `docker build -t ear-app .`
2. Spuštění kontejneru:
   `docker run -p 8080:8080 ear-app`

3 Postman, testovací data

3.1 Jak spustit?

1. Připojit se do workspace odkaz pro připojení
2. Spustit collection runner

3. Vybrat, které API otestovat
4. Vybrat soubor s testovacími daty, je vložen do workspace files s názvem testData.csv
5. spustit collection runner

3.2 Jak vyhodnotit výsledek?

Kvůli způsobu, jakým je Postman runner nakonfigurován, vzniká velké množství iterací requestů, pro která nejsou definována testovací data. Tyto requesty jsou označeny za skipped a přesměrovány na Postman echo, aby nijak nezatěžovaly server.

Každý request má předem definovanou očekávanou hodnotu HTTP status odpovědi a je doplněn o krátký komentář, který popisuje, co by měl tento request otestovat.

3.3 Testovací data

Každý enpoint má v souboru testData.csv své sloupce, které dodržují jmennou konvenci názevControllerMetody_názevParametru . Pro přidání nového testu stačí vyplnit nový řádek ve všech potřebných sloupcích definující request.

3.4 Testovací config v IDE

Databáze je naplněná testovacími daty, které odpovídají testovací sadě na postmanovi, tedy script na něm ví co má očekávat. Defaultně je v `application.properties` nastavený profile na `prod`, pokud budete chtít test na postmanovi spustit znovu, je nutné přepnout `spring.jpa.hibernate.ddl` na `create-drop`, jinak vzniknou konflikty a je nutné `spring.profiles.active=prod` nastavit na `dev`. Vše je popsané v komentářích.

4 Použití persistentní vrstvy

Aplikace splňuje požadavky na pokročilé mapování v JPA:

1. **Pojmenované dotazy (@NamedQuery):** Definovány na entitě `BoardGameItem`.
2. **Ordering (@OrderBy):** V BoardGame třídě na uspořádání ratings podle data.
3. **Kaskádní operace (Cascade):** Nastaveny pro propagaci změn mezi entitami - typicky u mapování kompozice.

5 Zkušenosti, zhodnocení

5.1 Filip Kopecký

- **Zabezpečení (Security): Vlastní JWT Implementace**
  - Bylo implementováno **vlastní řešení security**, zahrnující **vlastní Filter Chain** a autorizaci pomocí **JWT** tokenů.

- **Srovnání s Basic/Digest:** Oproti jednodušším metodám umožňuje tato volba **mnohem komplexnější konfiguraci** a nastavení granulárního přístupu k endpointům, což s sebou neslo nutnost složitější implementace.
- Bylo klíčové nastavit **zachytávání výjimek** generovaných Spring Security.
- **P**řekvapivě, ladění (**debugging**) security logiky bylo často efektivnější přímo v prohlížeči, což zjednodušilo identifikaci problémů.

- **Návrh REST Rozhraní a Práce s DTO**

  - Osvojování **resource-oriented "pattern"** u RESTful rozhraní představovalo zpočátku výzvu.
  - Používání **DTO objektů** pro striktní oddělení datové vrstvy. To je kritické pro zamezení úniku dat (data leakage) a zajištění, že se posílají pouze informace **relevantní pro request**, nikoli celá entita.
  - Implementace **globálního exception handleru** a správné handlování událostí pro konzistentní komunikaci s klientem.

5.2 Ondřej Feix

- **Postman testovací prostředí**

  Vytvořené prostředí v Postman runner výrazně usnadňuje testování. Na jedno kliknutí proběhne hromadně několik testů a není potřeba manuálně upravovat request pro každý test. Zároveň není potřeba znát jak request vypadá, ale stačí pouze přidat další řádek do tabulky testovacích dat.

- **Spring JSON parser**

  Měl jsem problém s tím, jak Jackson knihovna pasuje chyby v request body. Chtěl jsem vracet všechny validační chyby najednou, ale kvůli JSON parseru, který se zastaví na první chybě a dál nepokračuje, nešel tento problém vyřešit. Proto naše responses vrací pouze první validační chybu a ne všechny.