JPA / Hibernate Opdracht

Beer Brewery Management System

Niveau: Gevorderd (JPA zonder Spring)

Domein: Beheer van bieren, brouwers en categorieën

Doel: Leren werken met JPA/Hibernate, EntityManager, relaties, repositories en service-architectuur.

1. Beschrijving van het domein

Een bierbrouwerij wil een klein beheersysteem dat bieren, brouwers en biercategorieën bijhoudt. De cursisten krijgen *drie reeds aangemaakte entity-klassen*:

- **Beer**
- **Brewer**
- **Category**

Ze moeten een werkende CRUD-toepassing bouwen in **Java + JPA/Hibernate** (zonder Spring), met ondersteuning voor relaties en een duidelijke lagenstructuur.

2. Entiteiten

✓ Beer

- id (PK)
- name
- alcoholPercentage
- price
- brewer (ManyToOne)
- category (ManyToOne)

✓ Brewer

- id (PK)
- name
- location
- beers (OneToMany)

✓ Category

- id (PK)
- name
- description
- beers (OneToMany)

Alle relaties moeten correct geannoteerd zijn met JPA-annotaties.

3. Te bouwen projectstructuur

De cursist moet een project maken met exact deze structuur:

```
Src/main/java/

    config/
        JpaConfig.java

    model/
        Beer.java
        Brewer.java
        Category.java

    repository/
        BeerRepository.java
        BrewerRepository.java
        CategoryRepository.java

    service/
```

```
        BeerService.java
        BrewerService.java
        CategoryService.java

    app/
        MainApp.java
```

? 4. Repository-laag (verplicht)

Elke repository moet volgende methodes bevatten:

CRUD

- `create(T entity)`
- `findById(Long id)`
- `findAll()`
- `update(T entity)`
- `delete(Long id)`

Bijzondere queries

- **BeerRepository**
  - `findBeersByCategory(long categoryId)`
  - `findBeersByBrewer(long brewerId)`
  - `findBeersCheaperThan(double maxPrice)`
- **BrewerRepository**
  - `findBrewerByName(String name)`
  - `findAllBrewersWithBeerCount()`
- **CategoryRepository**
  - `findCategoryByName(String name)`

Technische eisen

- Gebruik **EntityManager** (van JpaConfig)
- Gebruik **TypedQuery**
- Sluit EntityManager correct
- Gebruik transacties voor create/update/delete

5. Service-laag (verplicht)

Elke service moet:

✓ CRUD opnieuw aanbieden

(but with validation and checks)

✓ Validatie toepassen

Bijvoorbeeld:

- naam mag niet leeg zijn
- prijs > 0
- alcoholpercentage ≥ 0
- brewer en category moeten bestaan vóór je Beer maakt

✓ Business logica bevatten

Voorbeelden:

- **BeerService**
  - prevent duplicatie (zelfde naam + brewer)
  - bier mag geen negatieve prijs hebben
- **BrewerService**
  - controleer of bestaande brouwer geen null-naam heeft
- **CategoryService**
  - categorie mag niet verwijderd worden zolang ze bieren bevat *(extra challenge)*

6. MainApp – Console applicatie

Cursist bouwt een menu-gestuurde applicatie:

Hoofdmenu

```
1. Manage Brewers
2. Manage Categories
3. Manage Beers
```

```
0. Exit
```

Bijv. Beer Submenu

```
1. Add Beer
2. View All Beers
3. Find Beer by ID
4. Update Beer
5. Delete Beer
6. Find Beers by Category
7. Find Beers by Brewer
8. Find Beers cheaper than X
0. Back
```

Gebruik:

- Scanner
- Netjes afhandelen van user input
- Exceptions opvangen
- Services correct aanroepen

7. Optionele uitbreidingen

- H2 in-memory database voor testen
- Abstracte `BaseRepository<T>` bouwen
- Lazy-loading elegant oplossen
- `@NamedQuery` gebruiken
- Command pattern toepassen in MainApp
- Logging (Log4j/SLF4J) toevoegen
- JSON import/export voor bieren