Übung W5: Spring Data Repositories & Facade Pattern

1) Spring Data Repositories

Sie haben im Unterricht gesehen wie einfach die Repositories mit Hilfe von Spring Data formuliert werden können. Die Interfaces, die ich ihnen bei Übung W2 im JPA Projekt abgegeben habe, entsprechen von den Schnittstellen her bereits den Spring Data Repositories, daher genügt es, wenn Sie Ihre Implementierungen wegwerfen ☹ und das Repository-Interface vom entsprechenden Spring-Interface ableiten.

Bsp: Das `MovieRepository`, das Sie implementiert haben,

```
public interface MovieRepository extends Repository<Movie, Long> {
	List<Movie> findByTitle(String title);
}
```

wird neu zu

```
public interface MovieRepository extends JpaRepository<Movie, Long> {
	List<Movie> findByTitle(String title);
}
```

Das Interface `ch.fhnw.edu.rental.persistence.Repository` können Sie nach dieser Migration ebenfalls entsorgen.

2) DTOs

In der Vorlesung haben Sie gesehen, dass die Service-Schnittstellen, so wie sie vorgegeben sind, nicht optimal sind. Die Service-Methoden geben als Resultate Domänen-Objekte zurück (welche vom Entity-Manager abgekoppelt sind):

```
public interface UserService {
	public User getUserById(Long id);
	public User save(User user);
	...
}
```

Mit diesem Ansatz haben wir folgende Probleme:

a) Falls auf diesen Domänen-Objekten (ausserhalb der Transaktionsgrenzen) auf LAZY deklarierte Assoziationen zugegriffen wird, so erhält man eine `LazyInitializationException` (ausser man verwendet open-session-in-view was bei Spring-Boot per Default (leider) der Fall ist).

b) Falls alle Assoziationen mit dem Fetch-Typ EAGER definiert sind, dann sieht man zwar keine `LazyInitializationExceptions` mehr, aber es werden potentiell zu viele Daten geladen. Mit den Benutzern werden auch alle Rental-Objekte geladen, und mit diesen auch alle referenzierten Filme, und mit diesen vielleicht auch alle User, welche diesen Film reserviert haben, etc., etc.

c) Bei der Methode `save` ist nicht klar, ob nur die im User gespeicherten Daten übernommen werden sollen, oder auch die im User gespeicherten Ausleihen in der Datenbank übernommen werden sollen (d.h. neue Ausleihobjekte in der DB gespeichert und nicht mehr enthaltene Ausleihobjekte gelöscht werden sollen).

Aufgabe:
Definieren Sie neue Service-Interfaces, in welchen anstelle der Domänenobjekte nur noch DTOs zurückgegeben werden. Für das Beispiel des Movie-Services könnte das Interface wie folgt aussehen:

```
public interface DtoMovieService {
    List<MovieDto> getAllMovies() throws RentalServiceException;
    MovieDto getMovieById(Long id) throws RentalServiceException;
    Long saveOrUpdateMovie(MovieDto movie) throws RentalServiceException;
    void deleteMovie(Long id) throws RentalServiceException;
}
```

Die DTO Objekte selber enthalten keine Referenzen auf Domänen Objekte sondern allenfalls Primärschlüssel der referenzierten Objekte oder Referenzen auf weitere DTO-Objekte.

Bemerkungen:

- Die neuen Service-Interfaces müssen implementiert werden. Sie können dazu eine (oder drei) Klassen definieren, welche die existierenden Services verwenden. Die oben definierte Methode `getMovieById` sieht dann z.B. wie folgt aus:

  ```
  public MovieDto getMovieById(Long id) throws RentalServiceException {
      return convertMovie(movieService.getMovieById(id));
  }
  ```

- Für die Erzeugung der DTOs in der Implementierung dieser Interfaces auf Serverseite (oben in die Methode `convertMovie` ausgelagert) stehen verschiedene Methoden zur Verfügung:
  - Java Code
  - Mapper Framework (z.B. MapStruct)
  - JPA Query Ausdrücke (SELECT NEW XY(…) …)

- Die Service-Implementierungen müssen im Kontext einer Transaktion ausgeführt werden, damit innerhalb der Implementierung auf den Entity-Manager zugegriffen werden kann. Sie müssen diese daher ebenfalls mit einer `@Transactional`-Annotation versehen.

- Die Controller-Klassen müssen angepasst werden, so dass diese Ihre neuen Services verwenden. Allenfalls müssen auf ihren DTOs noch JSON Annotationen angebracht werden.

- Auf Klientenseite müssen die Zugriffe angepasst werden. Der Klient wird dabei jedoch *nicht* mehr von den Entity-Klassen abhängen sondern nur noch von den DTO-Klassen. Alternativ können Sie auch mit Postman direkt auf die REST Schnittstelle zugreifen um das Verhalten zu testen.

Ausgabe: 18.10.2021
Abgabe: 01.11.2021 (per Pull-Request)