295 - Backend

Auftrag Fallstudie 4 - Tag

|  |  |
| --- | --- |
| Richtzeit  2.5 Stunden | Sozialform  Einzelarbeit |

## Beschreibung

Zu Beginn der Fallstudie haben wir bereits die Datenbank angeschaut und gesehen, dass wir neben dem Item und der Person noch Tags haben werden. In diesem Auftrag werden wir nun die Entität Tag umsetzen. Der Aufbau der Entität und der dazugehörigen Klassen wird analog zum Auftrag Fallstudie 3 sein. Das Tag wird direkt mit Service und DTOs realisiert.

Hier noch einmal die Datenbankstruktur:

Diagram

Description automatically generated

Erstelle alle folgenden Klassen im Package ch.bbcag.backend.todolist.tag.

## Graphical user interface, text, application, chat or text message Description automatically generatedTag

Die Entität Tag soll nach nebenstehendem Klassendiagramm erstellt werden. Wichtig ist dabei die Beziehung zum Item. Wende dafür die Annotationen der DataJPA – Assoziationen an. Wie du im ERD weiter oben ablesen kannst, handelt es sich um eine **ManyToMany**-Beziehung. Vergiss dabei nicht die Beziehung auch im Item einzutragen. Ausserdem soll das name-Feld **unique** sein. Schau dafür am besten im Cheatsheet zu den Annotationen nach.

Lass dir die Getter und Setter generieren und überschreibe die equals- und hashCode-Methode. Achte darauf, dass nur die Id verglichen und gehashed wird.

## TagRepository

Sobald die Entity Tag erstellt ist, können wir das dazugehörige Repository erstellen. Dieses kannst du analog zum Item machen. Erstelle dabei auch die Methode findByName(String name), damit ein Tag beim Namen gesucht werden kann.

## TagService

Graphical user interface, application

Description automatically generatedDa nun die Persistenzschicht des Tags gemacht ist kommt als Nächstes die Serviceschicht. Dazu gehört der Service selbst, die DTOs und ein Mapper.

### DTOs

Erstelle auf Grund des nebenstehenden Diagramms die beiden DTOs.

Auch hier wird das Set von Items aus der Entität umgewandelt in eine Liste mit ItemIds. Ob du dies von Hand oder mit einem Stream machst, kannst du selbst entscheiden.

### TagMapper

Mit den DTOs und der Entität erstellt kann nun der TagMapper geschrieben werden. Nachfolgend siehst du die Signaturen der beiden Methoden. Es werden wieder die Felder, die im DTO vorhanden sind in das Item geschrieben und umgekehrt.

public static TagResponseDTO toResponseDTO(Tag tag)

public static Tag fromRequestDTO(TagRequestDTO tagRequestDTO)

### TagService

Als letzter Teil der Serviceschicht kommt noch der TagService selbst. Im Service sollen alle Methoden des Repositories zur Verfügung gestellt werden inkl. der findByName. Auch hier werden analog zum Item RequestDTOs entgegengenommen und ResponseDTOs zurückgegeben. Für die Update-Funktionalität wird wieder eine merge-Methode benötigt. Lies dafür im letzten Auftrag nach, wie diese funktioniert. Beim Tag kann nur der Name aktualisiert werden.

## TagController

Zum Abschluss wird noch die Präsentationsschicht umgesetzt. Dafür erstellen wir den TagController, welcher die Requests behandelt.

Achte darauf, dass die Funktionen unter folgenden **Pfaden** und **HTTP-Methoden** erreichbar sind:

|  |  |  |  |
| --- | --- | --- | --- |
| Funktion | Pfad | Methode | Repository-Funktion |
| void delete(Integer id) | /tags/{id} | DELETE | deleteById(id) |
| Tag findById(Integer id) | /tags/{id} | GET | findById(id) |
| Tag insert(Tag tag) | /tags | POST | save(item) |
| Tag update(Tag tag) | /tags/{id} | PATCH | update(tag, tagId) |
| List<Tag> findTags(String name) | /tags?name={name} | GET | findByName(name)  findAll() |

Der name bei der findTags-Funktion soll optional sein, wenn nichts angegeben wird, sollen alle zurückgegeben werden.

Überprüf deine Arbeit mit Postman/SwaggerUI.

## Erweiterung Item

### ItemResponseDTO, ItemMapper und ItemService

Die Tags wollen wir neu auch im ItemResponseDTO mitgeben. Pass dieses also so an, dass die Tags in TagIds umgewandelt werden und in der Antwort drin sind. Entsprechend braucht es im DTO ein neues Feld, eine Liste mit Integers namens TagIds und im Mapper muss das Set mit Tags in die Liste mit deren Ids umgewandelt werden. Nimm die TagIds auch in die equals() und hashCode() Methoden auf

Auch im ItemService muss die merge-Methode angepasst werden, da neu auch Tags enthalten sind.

### Weitere Funktionen

Da das Tag nun vorhanden und mit dem Item verbunden ist können wir die Funktionalitäten des Items noch erweitern. Dafür erstellen wir im ItemRepository zwei neue Methoden, ein findByTagName und ein findByNameAndTagName. Verwende dafür jeweils ein SQL-join auf das Tag. Um die Funktionen auch verfügbar zu machen, müssen die beiden Methoden im Service ergänzt und die findItems-Methode im ItemController entsprechend angepasst werden, dass sie neben dem Item Namen auch ein Tag Namen entgegennehmen kann. Als Beispiel hier die findByTagName-Methode.

@Query("SELECT i FROM Item i " +  
 "JOIN i.linkedTags t " +  
 "WHERE t.name LIKE CONCAT('%', :tagName, '%')")  
List<Item> findByTagName(@Param("tagName") String tagName);

Überprüf auch hier die sowohl die vorherige als auch die neue Funktionalität mit Postman/SwaggerUI.