Belegaufgabe 4

songsRX mit Datenbankanbindung und mit neuem Endpoint songLists

Abgabetermin: Ihre Übung in KW 3, die Woche vom 14.01.2019.

Wir werden in diesem Projekt im Backend eine richtige Datenbank (☺) und Hibernate als JPA-Implementierung benutzen. Sie können die DB-Server auf dem studi-Server nutzen:
https://studi.f4.htw-berlin.de/www/services/mysql/ oder https://studi.f4.htw-berlin.de/www/services/pgsql/.
Legen Sie sich eine Datenbank an, z.B., 'SongsDB', mit entsprechenden Tabellen 'User', 'Song' und 'SongList' (zu SongLists später mehr). Tragen Sie die 10 Songs in die Song-Tabelle ein. Ihr Service kennt zwei User: {"id":1, "userId":"mmuster", "lastName":"Muster", "firstName":"Maxime"} und {"id":2, "userId":"eschuler", "lastName":"Schuler", "firstName":"Elena"}. Sie können sich natürlich auch andere und mehrere Nutzer anlegen. Tragen Sie Ihre Nutzer in die User-Tabelle ein. In diesem Beleg "gehören" die Songs allen Nutzern: **Nutzer können neue Songs anlegen und updaten, aber Nutzer dürfen keine Songs mehr löschen.**

Aufgaben:

1. Der Endpoint /songsRX/rest/songs muss entsprechend umgeschrieben werden und sollte die Songs aus der bzw in die Song-Tabelle lesen/schreiben, unter Verwendung von Hibernate und Dependency Injection.
2. Der Endpoint /songsRX/rest/auth muss umgeschrieben werden und sollte die User aus der User-DB-Tabelle entnehmen, unter Verwendung von Hibernate und Dependency Injection.
3. Die Unit-Tests für "PUT /songs" und "GET /auth?userid=" müssen angepasst werden. In beiden Testklassen **muss** eine TestDB (z.B. InMemoryTestDB) verwendet werden.
4. Bitte erst nachdem Sie 1., 2. und 3. erledigt haben, mit Aufgabe 5. anfangen:
5. Für Ihr letztes Projekt in diesem Semester werden wir unserem songsRX Service einen weiteren Endpoint hinzufügen:

   /songsRX/rest/**songLists**/<SONG_LIST_ID> bzw. /songsRX/rest/**songLists**?userId=**SOME_USERID**

Zusätzlich zu unserer Songs-Collection legen wir eine Sammlung von Songlisten (play lists) und ensprechende DB-Tabelle an. Eine Songliste hat:

- eine Id
- einen Owner (der ein User des Service ist)
- ist "private" oder "public"
- eine Liste von Songs

Zum Verständnis der Anforderungen, nehmen wir an, dass **der User "mmuster" mit 'GET /songsRX/rest/auth?userId= mmuster' von Ihrem Service authorisiert wurde und den Access-Token TOKEN bekommen hat.** Alle folgenden Anfragen von mmuster an Ihren Service müssen mit dem Header "Authorization: TOKEN" versehen werden (Beleg 3). Diesen Header werde ich übersichtshalber im Folgenden weglassen.

Anforderungen an den "songLists"-Endpoint:

a) GET /songsRX/rest/songLists?userId=SOME_USERID

Beispiel: '**GET /songsRX/rest/songLists?userId=mmuster'** soll alle Songlisten des Users 'mmuster' an den User 'mmuster' zurückschicken.

Beispiel: '**GET /songsRX/rest/songLists?userId=eschuler'** soll nur die 'public' Songlisten des Users 'eschuler' an den User 'mmuster' zurückschicken.

b) GET /songsRX/rest/songLists/<SONG_LIST_ID>

Beispiel: '**GET /songsRX/rest/songLists/22'** soll die Songliste 22 an den User 'mmuster' zurückschicken. Eine Songliste 22 muss existieren. Wenn die Liste mmuster gehört, dann kann sie an mmuster zurückgeschickt werden. Wenn die Liste einem anderen User gehört, dann kann die Liste nur zurückgeschickt werden, wenn sie "public" ist, ansonsten einen HTTP-Status 403 (FORBIDDEN) schicken.

c) **POST /songsRX/rest/songLists** mit XML oder JSON-Payload legt eine neue Songlist für den User ‘mmuster’ an und schickt die URL mit neuer Id für die neue Songliste im Location-Header zurück. Alle Songs in der Payload der Anfrage müssen in der DB existieren. Falls einer der Songs nicht existiert, können Sie der einfachheithalber eine 400 (BAD_REQUEST) zurückschicken.

d) DELETE /songsRX/rest/songLists/<SONG_LIST_ID>

User können nur eigene Songlisten löschen, nicht die der anderen User, auch keine fremden, public Songlisten.

Beispiel: ‘**DELETE /songsRX/rest/songLists/22’**. Löscht die Songliste 22, die dem User ‘mmuster’ gehört.

Beispiel: Ein ‘**DELETE /songsRX/rest/songLists/33’** vom User ‘mmuster’ requested, aber 33 gehört eschuler, soll von Ihrem Service mit HTTP-StatusCode 403 (FORBIDDEN) abgewiesen werden.

e) PUT-Funktionalität muss **nicht** implementiert werden.

Songlisten können Sie sich selbst zusammenstellen und in die entsprechende Tabelle eintragen. Zum Testen von POST für den endpoint “songLists” sollten Sie sich eigene Payloads erstellen.

Beide Nutzer sollen in Ihrer DB ein paar private und public Songlisten besitzen.

Hinweise:

1. Unit-Tests, wenn man sie parallel zum Entwickeln schreibt (sprich: code a little, test a little), helfen bei der Entwicklung. Deshalb können Sie für den neuen Enpoint welche schreiben, aber sie werden dieses Mal **ausnahmsweise** nicht bewertet (d.h. für Unit-Tests für den songList-endpoint gibt es keine Punkte).
2. Dependency Injection nutzen.
3. Hibernate nutzen.
4. Übersichtliche package-Struktur
5. packages von Klassen und ihren Testklassen müssen übereinstimmen.
6. Klassennamen müssen großgeschrieben werden, Methodennamen immer klein.