Backend 1st RE-Exam Fall 2024 - Item Booking Application

PDF Version

- Item Booking Application - pdf

Exercise Guidelines

- Allowed resources: written materials, personal computers, laptops, extra monitor, and internet resources. Headphones, and listening to music.
- Prohibited: communication with anyone other than the examiner, censor, and proctor. So no use of social media, forums, emails, sms, chatrooms, etc.
- Do not store solutions on external networks or drives/hosts like Facebook, OneDrive, Google Drive, etc. And don't share your code on Github until the end of the exam.
- Duration: 4 hours. Restroom breaks only. No smoking.

Consider your problem solving strategy (important)

1. Read the entire exercise before starting.
2. Sometimes you need to interpret the tasks. If you are unsure, make a decision and document it by adding a comment in the code or the `README.md` file.
3. If you get stuck on a task, move on to the next one.
4. Focus on demonstrating your approach to solving the tasks.
5. You will be asked to create entities and JPA DAOs from the beginning. If you get totally stuck with JPA, and that makes it difficult to continue, you can create a mock DAO instead. The mock DAO should implement the same interface as the JPA DAO and have the data hardcoded in the class in a list or map. BUT, if most of your JPA code is working, you should continue with JPA. Most likely, you will not need to implement a mock DAO today. Consider it the last resort.

Percentage distribution of the tasks

This is a breakdown of the distribution for each task.

Task	Topic	%
2	JPA and DAOs	25%
3	Building a REST Service Provider with Javalin	25%
4	REST Error Handling	5%
5	Streams	10%
6	Getting additional data from API	15%
7	Testing REST Endpoints	15%
8	Security	5%

Task	Topic	%
	Total	100%

Hand in on Wiseflow

1. A zip file containing your whole project, including the `README.md` file with answers to the theoretical questions.
2. A link to your GitHub repository. Don't push your solutions until the very end of the exam. Do not copy the clone link from GitHub, but grab the link from the browser address bar and paste it into Wiseflow.

Introduction

Build a backend system for an educational institution to manage and track items like cameras, microphones etc, in the medialab and makerlab. Tasks include managing students and their borrowed items. Theoretical questions are part of the exercise.

Domain Description

The application manages items and their borrowing status.

1. **Item**:
    - Attributes: `id`, `name`, `purchasePrice`, `category`, `acquisitionDate`, `description`.
    - Category: Enum (`VIDEO`, `VR`, `SOUND`, `PRINT`, `TOOL`).
2. **Student**:
    - Attributes: `id`, `name`, `email`, `enrollmentDate`, `phone`, `itemList`.
    - A student can borrow multiple items, but each item can belong to only one student.

Task 1: Setup

1.1 Create a new Java Project for Javalin and JPA.

1.2 Document your work in a `README.md` file.

Task 2: JPA and DAOs (25%)

2.1 Establish a **HibernateConfig** class with a method that returns an **EntityManagerFactory**.

2.2 Implement an **Item** entity class with these properties: `id`, `name`, `purchasePrice`, `category`, `acquisitionDate`, `description`. Use an enum for the category.

2.3 Implement a **Student** entity class with these properties: `id`, `name`, `email`, `enrollmentDate`, `phone`, and a **OneToMany** relationship to `items`.

2.4 Implement the DAOs for **Item** and **Student** as described below:

2.4.1 Create **ItemDTO** and **StudentDTO** classes.

2.4.2 Define a generic interface **IDAO** with CRUD operations.

2.4.3 Implement **ItemDAO** and **StudentDAO** classes using JPA, which implement the **IDAO** interface. You will need to implement the CRUD operations for **ItemDAO**, but you should only implement the CRUD operations for **StudentDAO** that you need for this exercise. So wait and see what you need.

2.4.4 Extend **ItemDAO** with these additional methods:

- `void addItemToStudent(int itemId, int studentId)`
- `Set<ItemDTO> getItemsByStudent(int studentId)`

2.5 Create a **Populator** class to populate the database with items and students.

Task 3: Building a REST Service Provider with Javalin (25%)

3.1 Develop a REST API with Javalin for items as described below with `routes`, a `controller`, and a `http-file`:

3.2 Create an **ItemController** using the **ItemDAO**. Use DTOs for data transfer.

3.3 Create an **ItemRoutes** file to define the API endpoints:

Method	Route	Description
GET	/items	Get all items
GET	/items/{id}	Get an item by ID
POST	/items	Add a new item
PUT	/items/{id}	Update an item
DELETE	/items/{id}	Delete an item
PUT	/items/{itemId}/students/{studentId}	Assign an item to a student

Method	Route	Description
POST	/items/populate	Populate the database

3.3.1 Manually test the endpoints using a **dev.http** file and document outputs in the `README.md`.

3.3.2 Include the student details when fetching an item by ID (for `/items/{id}`)

3.3.3 Theoretical question: Why is PUT used for assigning an `item` to a student instead of `POST`? Document the answer in the `README.md`.

Task 4: REST Error Handling (5%)

4.1 Return `JSON` exceptions for at least:

- Fetching an item by ID if it does not exist.
- Deleting an item that does not exist.

Task 5: Streams and Queries (10%)

5.1 Implement a method in **ItemController** to filter items by category, and add a corresponding route to test it from the `http-file`.

5.2 Add a route to summarize the total purchase price of items borrowed by each student:

```
[
  {
    "studentId": 1,
    "totalPurchasePrice": 5000
  },
  {
    "studentId": 2,
    "totalPurchasePrice": 3000
  }
]
```

Task 6: Getting Additional Data from API (15%)

6.1 Fetch shop data from an external API:

- Endpoint: `https://shopapi.cphbusinessapps.dk/shops`
- JSON response format:

```
{"shops":[
    {
      "id": 1,
      "name": "Tech World",
```

```
        "url": "https://techworld.com",
        "categories": ["VIDEO", "VR"]
      },
    ...
    ]
  }
```

6.2 In **ItemController**, fetch relevant shops for an item category. (`https://shopapi.cphbusinessapps.dk/shops/{category}`).

6.3 Include the shops with details in the response when fetching an item by ID. Depending on the item category, include the shops that sell items in that category.

6.4 Add an endpoint to get a list of all shops for a given category.

Task 7: Testing REST Endpoints (15%)

7.1 Create a test class for REST endpoints in **ItemRoutes**.

7.2 Use `@BeforeAll` to set up the Javalin server and dependencies.

7.3 Use `@BeforeEach` to prepare test objects.

7.4 Test each endpoint thoroughly.

7.5 Verify that shop data is included when fetching an item by ID.

Task 8: Security (5%)

8.1 Implement JWT-based authentication for the REST API.

8.2 Protect endpoints with roles.

8.3 Describe how to fix failing tests caused by role-based security or implement fixes in code.
