LI Databases & Web Programming

Week 8 - lab sheet - REST API Design & JDL Modelling

Search

Search this course

Week 8 - lab sheet - REST API Design & JDL Modelling

In this lab, we will apply the concepts from the lecture to design RESTful APIs and modify/model applications using JHipster JDL.

Part 1: REST API Design Conventions

This part contains 4 questions based on the REST API design conventions from the lecture slides 15-23.

Question 1:

A colleague suggests creating an API endpoint for retrieving all users with the following design:

GET /getAllUsers

Based on the lecture , why is this a poor design? What should the correct HTTP method and URL endpoint be?

Question 2:

Another colleague suggests an endpoint to publish a blog post:

GET /item/publish?id=123

Based on the lecture, name two REST conventions this endpoint violates and propose the correct HTTP method and URL path to achieve this.

Question 3:

You have a User entity and a Post entity, where a user can have many posts. How would you design a REST API endpoint to get the 10 most recent posts (i.e., page 1, with a limit of 10) for a specific user with ID 42? What is the name of this REST convention?

Question 4:

Your team is releasing version 2 of your API. Give an endpoint for your mobile app to get all available products, sorted by price (descending) and then by name (ascending).

Part 2: JHipster JDL Modelling

This part contains 5 questions. You will be given partially complete JDL specifications. For each question, update the JDL as requested, generate a new application, and answer the follow-up question.

Question 5: Entities and Validation

Task: Take the following JDL and add a model field that is required, a vin (Vehicle ID Number) that must be unique and have a minimum length of 17, and a required modelYear field with a minimum value of 1900.

```
entity Car {

make String

}
```

Follow-up: what happens if we use the field name year rather than modelYear? (if you try this using jhipster jdl car.jdl you must delete the folder target or generate a new application.)

Follow-up: After generating the code, what file in the src/main/java/.../domain/ directory defines this entity in a Java class? What do you notice about this class and the fields compared to the Java from 1st year?

Follow-up: What curl command and REST API endpoint would you use to get a list of all cars? What did you need to add in order for the command to work? How could you modify this command to get a single car?

Follow-up: What curl command and REST API endpoint would you use to create a Car entity? Run the command with valid and invalid data, to confirm the validation is working.

Question 6: Enums and Relationships

Task: You are given the following entity.

```
entity Employee {

name String required

}
```

Add a new enum called Department with values HR, ENGINEERING, and SALES. Then, create a new entity called Team with a name field (required, unique). Finally, add a ManyToOne relationship so that an Employee belongs to one Team, and a Team can have many Employees.

Follow-up: After generating, identify which backend folder (e.g., src/main/java/.../) would you find the Java Enum for Department?

Follow-up: After generating, which backend folder and method (e.g., src/main/java/.../web/rest/) would you look in to find the REST API endpoints that handle GET and POST requests for the Employee entity?

Follow-up: According to REST design conventions, what is the curl command for getting just the name of the first employee in the database? Try to run the command, note down the status code, and try to explain the result.

Question 7: Extending the User Entity

Task: The built-in User entity in JHipster is not meant to be modified directly. Using the pattern from the lecture, create a new UserProfile entity with a bio (TextBlob) and a profilePicture (ImageBlob). Then, add the correct OneToOne relationship to link UserProfile to the built-in User entity.

Follow-up: Why is it necessary to create a separate UserProfile entity instead of just adding bio and profilePicture fields to the User entity in the JDL?

Follow-up: Is there any overlap between the User and UserProfile Entities? How do we avoid this overlap?

Follow-up: What other steps/modifications are needed so that UserProfile and User are linked to each other?

Question 8: JDL Generation Options

Task: Given the "Project Task Management" JDL below, apply the following JDL options to all entities:

Fix the issue with the User relationship.

Paginate all entities with infinite-scroll.

Create a service layer for all entities using serviceClass.

Generate DTOs (Data Transfer Objects) for all entities using mapstruct.

/**

---

Example 2: Project Task Management

This models projects which contain tasks. Users can comment on tasks.

```
*/
enum TaskStatus {
TODO, IN_PROGRESS, DONE
}
entity Project {
name String required unique,
description String
}
entity Task {
title String required,
dueDate LocalDate,
status TaskStatus
}
entity Comment {
text String required,
postedAt Instant required
}
/** A Project can have many Tasks. */
relationship OneToMany {
Project{task} to Task{project(name)}
}
/** A Task can have many Comments. */
relationship OneToMany {
```

Task{comment} to Comment{task(title)}

}

/** A Comment is written by one User. */

relationship ManyToOne {

Comment{author(login)} to User

}

Follow-up: According to the lecture, what is the main security benefit of using DTOs?

Follow-up: According to the REST conventions from the lecture, what curl command is needed to get all comments made by a specific User? Try to run the command, note down the status code, and try to explain the result.

Follow-up: Looking at the JDL documentation - what alternative short-hand notation do we have for marking the pagination of Entities? e.g. if we wanted to have different pagination types for each entity.

Question 9: Application Regeneration

Task: You have an existing, generated application. You now need to add a priority (Integer) field to the Task entity from the previous question. What two different jhipster commands could you use from your terminal to apply this change?.

Follow-up: If you had already written custom code in your backend files, what key would you press at the interactive prompt to see a "diff" of the changes before deciding whether to overwrite?

Part 3: JDL Design Scenarios

This part contains 3 written scenarios. For each scenario, write a complete JDL file (including entities, relationships, and basic validation) that models the domain. Generate each application.

Question 10: Simple Library System

Scenario: Design a JDL for a simple library. The system needs to track Books, Authors, and Genres.

Requirements:

Book: Needs a title (required), isbn (required, unique), and description (TextBlob).

Author: Needs a name (required).

Genre: Needs a name (required, unique).

Relationships:

A Book has exactly one Author. An Author can have many Books.

A Book can have many Genres. A Genre can be applied to many Books.

Follow-up: Write the curl command to find the list of all Authors.

Question 11: Medical Clinic

Scenario: Design a JDL for a medical clinic appointment system.

Requirements:

Patient: Needs name (required) and dateOfBirth (Instant).

Doctor: Needs name (required) and specialty (String).

Appointment: Needs dateTime (Instant, required) and a reason (String).

Relationships:

An Appointment must be for exactly one Patient. A Patient can have many Appointments.

An Appointment must be with exactly one Doctor. A Doctor can have many Appointments.

Follow-up: Write the curl command to create a new Patient. How do we get the ID of the newly created Patient?

Question 12: Music Streaming Service

Scenario: Design a JDL for a music streaming service.

Requirements:

Artist: Needs name (required, unique).

Album: Needs title (required) and releaseDate (Instant).

Track: Needs title (required) and durationInSeconds (Integer, required, must be min(1)).

Relationships:

An Artist can have many Albums.

An Album must belong to exactly one Artist.

An Album can have many Tracks.

A Track must belong to exactly one Album.

Options: Apply dto with mapstruct, service with serviceClass, and paginate with pagination to all entities.

Follow-up: Write the curl command to find a specfic Album by database ID?