REST API part 1

Creating a REST API with Spring Data JPA and a RestController

Learning objectives

- Create a Spring Boot project using Spring Initializr
- Understand the role of the main application class and annotations
- Define a JPA entity and map it to a relational database
- Use Spring Data JPA to perform CRUD operations
- Build and expose RESTful endpoints with @RestController
- Configure and use the H2 in-memory database for persistence and testing

Introduction to the Demo Project – Smoothie Bar

- Scenario: Manage drinks in the resort’s Smoothie Bar
- Entity: Drink
- Fields: id, name, size, price
- Endpoints:
  - GET /drinks – List all drinks
  - POST /drinks – Add a new drink
  - PUT /drinks - Update en existing drink
  - DELETE /drinks - Delete a drink

Technology Highlights

- Spring Boot auto-configuration
- Spring Data JPA repositories
- H2 database integration
- RESTful architecture

Project Setup

- Create a new project using Spring Initializr
  - Dependencies: Spring Web, Spring Data JPA, H2 Database
- Folder structure generated automatically:
  - entity, repository, controller, resources
- Main class annotated with @SpringBootApplication
  - Enables component scanning
  - Configures default application context

Creating a JPA Entity

- Create Drink.java in entity package
- Annotate with @Entity to map it to a database table
- Annotate the primary key with @Id and @GeneratedValue
- Add other fields: name, size, price
- Include standard getters and setters
- Spring Boot automatically generates the schema based on entity fields

Using Spring Data JPA

- Create DrinkRepository interface
- Spring Data JPA automatically implements standard CRUD operations
  - For example: findAll(), findById(), save(), deleteById()
- Eliminates the need for writing boilerplate data access code
- The repository is injected into the controller to access data layer

Creating the REST Controller

- Create DrinkController annotated with @RestController
- Map endpoints using @RequestMapping("/drinks")
- Define methods for GET, POST, PUT, DELETE
- The controller directly interacts with the repository
- JSON serialization is handled automatically by Spring Boot

Configuring the H2 Database

- Use H2 for fast, in-memory persistence
- Add configuration in application.properties:
- Access the H2 console at /h2 in the browser
- View and verify the automatically created DRINK table
- Perfect for learning and prototyping

Testing the API

- Run the application
- Test using Postman or browser
- Example requests:
  - GET /drinks → Returns list of drinks
  - POST /drinks with body: { "name": "Mango Smoothie", "size": "Large", "price": 6.50 }
- Verify persistence by re-fetching all drinks
- Observe auto-generated JSON responses

Lab Project – Beach Activity Service

- Use the starter project: beach-activity-starter
- Task 1 - Create methods in the Controller
- Task 2 - Add the correct annotations to the Activity class to make it into a JPA entity
- Task 3 - Run the application, and test endpoints with Postman!

Summary of Technical Learnings

- Spring Boot simplifies REST service creation through auto-configuration
- @Entity and Spring Data JPA remove boilerplate persistence code
- The repository layer abstracts database operations cleanly
- @RestController easily exposes REST endpoints as JSON APIs
- H2 in-memory database provides quick setup for learning and testing