Integrating Data Sources

Overview

In this lab you'll enhance your "Online Retailer" app so that users can suggest new products that the company might like to sell in future. For each product suggestion, users specify the following information:

- Product description
- Recommended price
- Estimated sales per year

Later on in the course, you'll define a repository class to persist product suggestions data to a relational database. You'll also define a REST API to enable users to insert product suggestions, query them, modify the recommended price and estimated sales per year, etc.

IntelliJ starter project

If you're happy to continue where you left off in the previous lab, use the following project:

- **`student\student-online-retailer`**

If you'd prefer a fresh start, use the solution project from the previous lab instead:

- **`solutions\solution-spring-boot-techniques`**

IntelliJ solution project

The solution project for this lab is located here:

- **`solutions\solution-integrating-data-sources`**

Roadmap

There are 5 exercises in this lab, of which the last exercise is "if time permits". Here is a brief summary of the tasks you will perform in each exercise; more detailed instructions follow later:

1. Adding database support in the pom file
2. Defining application properties for persistence
3. Defining a **`ProductSuggestion`** entity class
4. Verifying the database schema is created correctly
5. (If time permits) Seeding the database with sample data

Exercise 1: Adding database support in the pom file

Add the following dependencies to your pom file, to support JPA persistence to an in-memory H2 database (see the demo application for details, if you need a reminder):

- The Spring Boot Starter for JPA
- The H2 database dependency

Exercise 2: Defining application properties for persistence

Spring Boot auto-configuration helps a lot when it comes to configuring persistence properties. For example, when it sees the H2 dependency in your pom file, Spring Boot automatically configures the following properties for you (i.e., you don't need to set these yourself):

```
spring.datasource.url=jdbc:h2:mem:example

spring.datasource.username=sa

spring.datasource.password=

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
```

Spring Boot auto-configuration doesn't always do everything for you though. You still need to set some application properties for JPA support. We recommend the following properties:

```
spring.jpa.hibernate.ddl-auto=create-drop

spring.jpa.properties.hibernate.show_sql=true

spring.jpa.properties.hibernate.use_sql_comments=true

spring.jpa.properties.hibernate.format_sql=true
```

Exercise 3: Defining a `ProductSuggestion` entity class

Define a JPA entity class named **`ProductSuggestion`** with the following properties:

- **`id`** (auto-generated primary key)
- **`productDescription`**
- **`recommendedPrice`**
- **`estimatedAnnualSales`**

JPA will automatically create/drop a table based on this entity class at the start/end of your application (because you set the **`spring.jpa.hibernate.ddl-auto`** application property to **`create-drop`**).

Exercise 4: Verifying the database schema is created correctly

Run the application, and verify that a *create table* statement is displayed in the console window, indicating the database table has been created. Note that H2 uses snake-case for table names and column names, i.e., the table is named **`product_suggestion`** and the columns are named:

- **`id`**
- **`product_description`**
- **`recommended_price`**
- **`estimated_annual_sales`**

Now enhance your application so that you can view the database schema via the H2 Console. You'll need to make the following changes (see the PowerPoint chapter if you need a reminder):

- In the pom file, locate the **`spring-boot-starter`** dependency and change it to **`spring-boot-starter-web`**.
- In the application properties file, add properties to enable the H2 Console.

Run the application again; you'll see a message that indicates the JDBC connection string for the database. Open a browser window and browse to the H2 Console, and connect to the database via the connection string. Verify that the database has a **`product_suggestion`** table with the correct schema.

Exercise 5 (If time permits): Seeding the database with sample data

Define a class named **`SeedDb.java`**. Write some code to insert some sample entries in the **`product_suggestion`** table. Then run the application again and use the H2 Console to verify that the data has been inserted successfully into the database.