Enterprise Java with Spring
Spring REST API
Lab 3

1 Lab setup

Make sure you have the following items installed

- Latest LTS JDK version (at this point: JDK 21)
- Spring Tool Suite (STS) or IntelliJ IDEA
- Latest version of Maven (at this point: Maven 3.9.9)
- A free account at Postman and installed the Postman app
- A suitable text editor (Notepad ++)
- A utility to extract zip files (7-zip)

In each of the main lab folders, there are two subfolders: `changes` and `final`. The `changes` subfolder just holds the source code files for the lab, while the `final` subfolder holds the complete Eclipse project starting from its project root folder. We will use the code from the `changes` subfolder

to build up our applications from scratch and you can always fall back on the complete Eclipse project if you encounter any errors while building up the application.

2 Designing REST API

In this lab, we will design a REST API from scratch. The first thing we need to do is to create the underlying classes that model the business domain and then determine the REST API endpoints that access and manipulate a collection of objects from these classes.

The REST service will maintain a list of developers with the following schema:

Developer
`id: Integer`
`name: String`
`age: Integer`
`languages: Array[String]`
`married: Boolean`

It will expose the following API endpoints for consumption by a front end service:

Method	**Endpoint**	**Description**
GET	`/api/developers`	Get the list of developers
GET	`/api/developers/{id}`	Retrieve the developer with the specified id
GET	`/api/developers?language=XXX`	Retrieve the developers with capability in the specified language
GET	`/api/developers?married=XXX`	Retrieve the developers with the specified marital status
GET	`/api/developers?married=XXX &language=YYY`	Combination of the two previous conditions
GET	`/api/developers?age>XX|age<XX`	Retrieve the developers whose age is more or less than XX
GET	`/api/developers?limit=X&start=Y`	Pagination functionality to retrieve the next X developers starting from the first Y developers
POST	`/api/developers`	Add a new developer to the list of developers
PUT	`/api/developers/{id}`	Make a modification to a developer with the specified id
DELETE	`/api/developers/{id}`	Delete a developer with the specified id

We have left out the PATCH operation at the moment as this requires JSON PATCH or specialized logic to perform correctly.

3 Creating the application package structure

The source code for this lab is found in `Dev-Spring-Rest/changes` folder.

Start up STS. Ensure you are in the Java EE perspective.

Go to File -> New -> Other -> Spring Boot -> Spring Starter Project. Complete it with the following details:

Name: `DevSpringRest`
Group: `com.workshop.rest`
Artifact: `DevSpringRest`
Version: `0.0.1-SNAPSHOT`
Description: `Simple REST API for working with a group of devs`
Package: `com.workshop.rest`

Click Next.

Add the following dependencies:
`Web -> Spring Web`
`Developer Tools -> Spring Boot Dev Tools`

The Spring Boot DevTool dependency will automatically restart the Spring Boot application whenever it detects changes in your compiled application code – this will facilitate the development of REST API service as you will not need to start / restart the service every time you change your source code and save. If you wish to leave out this dependency, then make sure you manually start / restart your app after source code changes in all the subsequent lab sessions.

We will use the Data Access Object (DAO) design pattern in our implementation in order to separate low level data access logic from higher-level business logic abstraction through an abstract API. This allows us to change the underlying implementation for the data access logic (to an in-memory list or a relational database table) without changing the business logic code.

In `src/main/java`, create the following packages which are going to have the following purposes:

Package name	Purpose
`com.workshop.rest.model`	Holds all the classes for the business domain model
`com.workshop.rest.dao`	Holds the interfaces for the DAO pattern
`com.workshop.rest.service`	Holds the implementations for the DAO interfaces
`com.workshop.rest.controller`	Holds all the @RestController classes and any @ControllerAdvice classes that performs exception handling
`com.workshop.rest.exception`	Holds all the user-defined exceptions as well as any custom error messages that we wish to return to the client

The main @SpringBootApplication class will reside in the top level package: `com.workshop.rest` so that Spring will scan all its subpackages to locate @Component classes for DI if necessary.

4 Create model, interface, implementation and controller

In `com.workshop.rest.model`, place:

`Developer`

In `com.workshop.rest.dao`, place:

`DeveloperDAO`

In `com.workshop.rest.service`, place:

`DeveloperService`

We hardcode in some values for the list of developers in DeveloperService. In a real life implementation, this would typically be read from a file or a database table instead.

In `com.workshop.rest.controller`, place:

`DeveloperController`

We have the skeleton now for the main controller class that will provide the logic for mapping all the API endpoints. This will be fleshed out step by step.

Notice that it has a field of the interface type DeveloperDAO that is marked with @Autowired. The Spring framework will perform DI for this using the DeveloperService implementation since that is marked with @Component and is in a subpackage of the @SpringBootApplication class.

Start the app in the usual manner.
Right click on the project entry in the Project explorer pane and then select Run As -> Spring Boot App, or right click on the project entry in Spring Boot dashboard (Window -> Show View -> Other -> look in the Other folder -> Boot Dashboard) and run from there

5 Implement GET /api/developers

In `com.workshop.rest.controller`, make the change:

`DeveloperController-v2`

In `com.workshop.rest.service`, make the change:

`DeveloperService-v2`

Start up Postman and create a collection to group requests for this app: `Developer REST requests`

Make a GET request to:

`localhost:8080/api/developers`

and verify that the initial list of hardcoded developers in the constructor of `DeveloperController` is returned.

6 Implement POST /api/developers

In `com.workshop.rest.controller`, make the change:

`DeveloperController-v3`

In `com.workshop.rest.service`, make the change:

`DeveloperService-v3`

In `com.workshop.rest.exception`, copy the following files from `changes`

`CustomErrorMessage`
`IncorrectJSONFormatException`

In `com.workshop.rest.controller`, copy the following file from `changes`

`DeveloperControllerExceptionHandler`

Make a POST request to:

`localhost:8080/api/developers`

with the following raw JSON content in the body:

```
{
  "name": "Ryan",
  "age": 42,
  "languages": [
      "JavaScript",
      "Python",
      "Java"
  ],
  "married": true
}
```

Check that a status 201 Created is returned with the following URL in the Location header:
http://localhost:8080/api/developers/6
NOTE: You will not be able to make a GET request to this API endpoint yet, as we do not yet have the implementation for it.

Verify that this new developer has been added to the list of existing developers by now making a GET request to:

`localhost:8080/api/developers`

Introduce an error into the POST submission by leaving out an important property (either `name`, `age` or `languages`). For e.g. sent a POST to the same URL with this content which is intentionally missing the `languages` property:

```
{
  "name": "Aaron",
  "age": 42,
  "married": true
}
```

Verify that the correct exception handling method is invoked server side and an appropriate error message is returned.

Make another GET request to the same URL to verify that this information was not added as a new developer.

Introduce an error by submitting invalid JSON using a POST for e.g.

```
{
  "name": "Debbie"
  "age": 29
  "languages": [
      "JavaScript",
      "Python",
      "Java"
  ]
  "married": false
}
```

Verify that the correct exception handling method is invoked server side and an appropriate error message is returned.
Make another GET request to the same URL to verify that this information was not added as a new developer.

7 Implement PUT /api/developers/{id}

In `com.workshop.rest.controller`, make the change:

```
DeveloperController-v4
DeveloperControllerExceptionHandler-v4
```

In `com.workshop.rest.service`, make the change:

```
DeveloperService-v4
```

In `com.workshop.rest.exception`, copy the following file from `changes`

`DeveloperNotFoundException`

Notice that we are using the IncorrectJSONFormatException to signal different types of errors that can occur in the request. Ideally, we should create separate custom exception types to cater for different types of errors (just as we have done for DeveloperNotFoundException), but to keep it simple here we will just reuse IncorrectJSONFormatException.

Make a PUT request to:

`localhost:8080/api/developers/3`

with the following valid raw JSON content in the body:

```
{
  "name": "Edwin",
  "age": 21,
  "languages": [
      "Python",
      "C++"
  ],
  "married": false
}
```

Verify that this returns with status 200 OK. Next make a GET request to:

`localhost:8080/api/developers`

and verify that the developer with id 3 has had the details changed accordingly.

Let's introduce some potential errors in the request.

Make a PUT request to:

`localhost:8080/api/developers/3`

with the following raw JSON content in the body which is intentionally missing the `name` property:

```
{
  "age": 21,
  "languages": [
      "Python",
      "C++"
  ],
  "married": false
}
```

Verify that the error message returned points out that all fields in the developer record needs to be specified.

Make a PUT request to:

`localhost:8080/api/developers/3sw`

Verify that the error message returned points out that the developer id needs to be specified correctly as a number.

Make a PUT request to:

`localhost:8080/api/developers/888`

with valid raw JSON content in the body:

```
{
  "name": "Edwin",
  "age": 21,
  "languages": [
      "Python",
      "C++"
  ],
  "married": false
}
```

Verify that the error message returned points out that no developer with such id exists.

8 Implement DELETE /api/developers/{id}

In `com.workshop.rest.controller`, make the change:

`DeveloperController-v5`

In `com.workshop.rest.service`, make the change:

`DeveloperService-v5`

Make a DELETE request to:

`localhost:8080/api/developers/3`

Verify that this returns with status 200 OK. Next make a GET request to:

`localhost:8080/api/developers`

and verify that the developer with id 3 has been removed

We can check for similar errors as in the case of the PUT operation.
Make a DELETE request to:

```
localhost:8080/api/developers/3sw
```

Verify that the error message returned points out that the developer id needs to be specified correctly as a number.

Make a DELETE request to:

```
localhost:8080/api/developers/888
```

Verify that the error message returned points out that no developer with such id exists.

9 Implementing GET /api/developers/{id}

In `com.workshop.rest.controller`, make the change:

```
DeveloperController-v6
```

In `com.workshop.rest.service`, make the change:

```
DeveloperService-v6
```

This time stop and restart the app, so that all the previous live changes that you have made to the Developer list in DeveloperService are lost and it is reconstructed with the original 5 hardcoded Developer object values.

Make a GET request to:

```
localhost:8080/api/developers/x
```

where x is valid developer ID (which at the start should be from 1 to 5). Verify that the correct developer is returned.

We can check for similar errors as in the case of the PUT operation.
Make a GET request to:

```
localhost:8080/api/developers/3sw
```

Verify that the error message returned points out that the developer id needs to be specified correctly as a number.

Make a GET request to:

```
localhost:8080/api/developers/888
```

Verify that the error message returned points out that no developer with such id exists.

10 Implementing GET /api/developers with query parameters

In `com.workshop.rest.controller`, make the change:

`DeveloperController-v7`

In `com.workshop.rest.service`, make the change:

`DeveloperService-v7`

In `com.workshop.rest.dao`, make the change:

`DeveloperDAO-v7`

Since there are many query parameters that can be passed via a GET to `/api/developers`, we will refactor the existing code base to streamline our implementation by changing the way we perform a `get` in the DAO and the way it is implemented in the service and invoked from the controller.

We make distinction between a GET request without any query parameters (which should return the list of all developers) and a GET request with some query parameters (which we initially list and subsequently process one at a time).

Make a GET request to:

`localhost:8080/api/developers`

and verify that the initial list of hardcoded developers in the constructor of `DeveloperController` is returned.

Make a GET request to the same URL but with some random query parameters, for e.g:

`localhost:8080/api/developers?hero=ironman&age=33`

and verify that nothing is returned now (even though status remains as 200 OK) and the key value pairs are logged on the server side

10.1 Query parameter: language=XXX

In `com.workshop.rest.service`, make the change:

`DeveloperService-v8`

We implement the filtering functionality of this query parameter in a separate private function, which returns a smaller filtered list then can subsequently undergo further filtering if necessary.
We also define all the query keys as String constants at the top of the class to minimize typos and to facilitate change if we ever decide to change the name of the keys.

Make multiple GET requests to:

`localhost:8080/api/developers ?language=XXXX`

where `XXX` can be any of the valid languages associated with the existing developers: e.g. Python, Java, Javascript, etc. Note that we use a lowercase function on the strings to be compared so the value of `XXX` is case-insensitive for comparison purposes.

Verify that the correct subset of developers is returned with a status 200 OK.

Also verify that making a GET request to:

```
localhost:8080/api/developers
```

still returns the initial list of hardcoded developers in the constructor of `DeveloperController`.

10.2 Query parameter: married=XXX

In `com.workshop.rest.service`, make the change:

```
DeveloperService-v9
```

We implement the filtering functionality of this query parameter in a separate private function, which returns a smaller filtered list then can subsequently undergo further filtering if necessary.

Make GET requests to:

```
localhost:8080/api/developers?married=false
```

and

```
localhost:8080/api/developers?married=true
```

Verify that the correct subset of developers is returned with a status 200 OK.

Make a GET request with an invalid value for the married key, for e.g.

```
localhost:8080/api/developers?married=sdf
```

Verify that an appropriate error message is returned with a status 400 BAD REQUEST.

Also verify that making a GET request to:

```
localhost:8080/api/developers
```

still returns the initial list of hardcoded developers in the constructor of `DeveloperController`.

10.3 Combine query parameters: language=XXX&marriage=YYY

The current implementation for the filtering functionality is designed to cascade the filtering effect by passing the results from one specific filtering function as the input list to the next filtering function.

Make a GET request to:

`localhost:8080/api/developers?married=true&language=java`

Verify that the correct subset of developers is returned with a status 200 OK.

Notice that changing the order of appearance of the parameters does not affect the result:

`localhost:8080/api/developers?language=java&married=true`

Experiment around with a few other combinations for the `married` and `language` query string parameters and verify that the correct subset of developers is returned with a status 200 OK.

10.4 Query parameter: age>XX or age<XX

Add the following dependencies to the existing `pom.xml` in order to utilize the GraalVM Javascript engine in order to perform evaluations of conditional expressions in string format (for e.g. `age>XX` or `age<XX`)

```
<!--		Dependencies for GraalVM Javascript engine to support -->
<!--		JavaScript evaluation-->

		<dependency>
			<groupId>org.graalvm.js</groupId>
			<artifactId>js</artifactId>
			<version>23.0.3</version>
		</dependency>

		<dependency>
			<groupId>org.graalvm.js</groupId>
			<artifactId>js-scriptengine</artifactId>
			<version>23.0.3</version>
		</dependency>
```

In `com.workshop.rest.service`, make the change:

`DeveloperService-v10`

Note that the format: `age>XX` or `age<XX` is counted as a key without a value as the = sign is used to separate keys and values. We need to take this into account when checking query parameters. We also use the ScriptEngine and ScriptEngineManager to perform evaluation of a String conditional expression.

Make a GET request to:

`localhost:8080/api/developers?age<35`

Verify that the correct subset of developers is returned with a status 200 OK.

Make a GET request to:

`localhost:8080/api/developers?age>50`

Verify that the correct subset of developers is returned with a status 200 OK.

Notice that we can also combine the filtering here with the previous parameters, for e.g:

```
localhost:8080/api/developers?age<40&language=java
```

Verify that the correct subset of developers is returned with a status 200 OK.

Play around with different combinations of the various query parameters we have implemented so far and verify that the correct subset of developers is returned with a status 200 OK.

If we attempt to use a conditional expression involving > or < with a parameter other than age, the server should return an error.
Make a GET request to:

```
localhost:8080/api/developers?name>40
```

Verify that an appropriate error message is returned.

10.5 Query parameters: limit=X&start=Y

In `com.workshop.rest.service`, make the change:

```
DeveloperService-v11
```

We need to ensure that both `limit` and `start` parameters are supplied together, as both are needed in combination to perform this filtering functionality.

Make a GET request to:

```
localhost:8080/api/developers?start=2&limit=3
```

Verify that the correct subset of developers is returned with a status 200 OK.

Make a GET request to:

```
localhost:8080/api/developers?start=3&limit=3
```

Verify that the correct subset of developers is returned with a status 200 OK.

Make a GET request to:

```
localhost:8080/api/developers?start=3&limit=3
```

Verify that the correct subset of developers is returned with a status 200 OK.

Make a GET request to:

```
localhost:8080/api/developers?start=3&limit=10
```

Notice that even thought the limit specified exceeds the total remaining number of developers in the list, no error is flagged and all remaining developers in the list are returned instead.

Errors will be flagged if:

- Non-numeric values are specified for either the start or limit parameters
- The start value is more than the number of items in the list
- The start or limit parameter is supplied individually without the other (both are required for the filtering functionality to work)

Make GET requests to these URLs to verify that all of these errors are flagged with appropriate error message responses:

```
localhost:8080/api/developers?start=xxx&limit=10

localhost:8080/api/developers?start=2&limit=yyy

localhost:8080/api/developers?start=10&limit=2

localhost:8080/api/developers?start=2

localhost:8080/api/developers?limit=2
```

10.6 Further filtering / sorting functionality

Query parameters used with GET requests are ideal for performing a variety of filtering and sorting functionality. For e.g. we could sort the list of developers returned on ascending or descending value of a particular field (such as age or name). The query parameter might look like this:

```
GET /api/developers?sort=-age
```

which might mean to sort on descending order of the age field of the developers.

```
GET /api/developers?sort=+name
```

which might mean to sort on ascending order of the name field of the developers.

```
GET /api/developers?sort=-age,+name
```

which might mean to sort on descending order of the age field of the developers first (primary sort), and then sort on ascending order of the name field for the case of developers who have the same age (secondary sort).

See whether you can implement this sorting functionality as an exercise on your own.

https://www.javatpoint.com/how-to-sort-arraylist-in-java
https://beginnersbook.com/2013/12/how-to-sort-arraylist-in-java/