**German International University**
**Faculty of Engineering**
**Department of Media Engineering and Technology**

**Architecture of Massively Scalable Applications**, Spring 2025
**Lab Manual 1**

Welcome to your first lab! In this lab, we will be working hands-on on the content we discussed in the labs. To follow along with this lab, you need to install the required tools mentioned in the installation guide uploaded to the CMS.

# 1 Initializing your project

a) Visit `https://start.spring.io/` and generate a new project with the following dependencies:

1. Spring Web
2. Spring Boot DevTools
3. Rest Repositories

b) Open the project in IntelliJ

c) Configure your SDK to use jdk23: File > Project Structure > Platform Setting > SDKs > Add JDK 23 (`https://www.jetbrains.com/help/idea/sdk.html`)

d) You may encounter a problem in the latest IntelliJ setups if you are using macOS or Linux, the error is when maven fails to read the pom.xml file. To fix this you need to go to Settings/Preferences > Build, Execution, Deployment > Build Tools > Maven, and change the Maven home path from **Use Maven Wrapper** to **Bundled**

e) To enable the hot-reload feature, go to Settings > Advanced Settings > Tick: Allow auto-run to start ...

f) Run the project and visit `localhost:8080` to make sure all is running correctly.

# 2 Structure your project

Create packages with the following namings

a) controllers

b) services

c) repositories

d) models

e) utils

# 3 Create your first controller

In the `controllers` folder create `UserController` with URL `/users`, then create a method called `testController` which returns the text `Hello from user controller` with URL `/hello` , do not forget to use the following annotations:

a) @RestController

b) @RequestMapping

c) @GetMapping

# 4 Create User Model

Create a User class in the models folder with the following attributes

a) **id**: String (to be used as `UUID.randomUUID();`)

b) **name**: String

c) **age**: Integer

d) **email**: String

for the time being, we will note be annotating our model with any annotation, until we get into the database, we will use the @Entity annotation.

Then we create getters and setters for the above attributes and create an empty constructor, a constructor with fields except id, and a constructor with all fields for the class

# 5 Modify User Controller

In this section, we need to create 3 API request handlers as follows:

a) `GET /users` => `getUsers()`: This method should return a list containing a dummy User created on the spot.

b) `GET /users/{id}` => `getUserById(...)`: This method should take as input the id in the URL using `@PathVariable`.

c) `POST /users` => `createUser(...)`: This method takes the user content from the body using `@RequestBody` in the parameters and returns the same user with a generated id.

# 6 Create User Repository

To complete this section, you can use the JSON file containing dummy users from `https://gist.github.com/mmedhat1910/f38d112e3353165817c84ff295b24a0a`, and place it inside your `resources` folder.

Start by creating `UserRepository` class and annotate it with @Repository

Create the following methods

a) `findAll()`: read the JSON file and return all users. You can use the block below to read the JSON file.

```
ObjectMapper objectMapper = new ObjectMapper();
String FILE_PATH = "src/main/resources/users.json";
File file = new ClassPathResource("users.json").getFile();
List<User> users = objectMapper.readValue(file, new TypeReference<List<User>>() {});
```

make sure to handle the IOException thrown when trying to open the file, if the file does not exist you should throw the following exception

```
throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR,
     "Unable to read users.json");
```

b) `findById(String id)`: this method should return the user (if exists) by the id you can use the below code block to perform this task

```
return findAll().stream()
.filter(user -> user.getId().equals(id))
.findFirst();
```

take care, this function returns `Optional<User>` type, which indicates that the output could be null, so to check if it is null we can user `.isEmpty()` method on it and use `.get()` on it to get the actual user value.

c) `save(User user)`: this is a dummy method to simulate saving the user to db. This method should return the same user with the id set to `UUID.randomUUID().toString();`

# 7 User Service

Now let's create our `UserService` class annotated with @Service. Instead of having our functionalities inside the controller, we will move it to the service and use it via dependency injection.

We need to first inject the `UserRepository` using the @Autowired annotation.

Finally, the class should have the following methods

a) `getUsers()`: this returns list of users using findAll

b) `getUserById(String id)`: this method should use the findById method in the repository, but make sure to check if the user is null throw `ResponseStatusException` with status code 404 using `HttpStatus.NOT_FOUND`, otherwise return the user.

c) `createUser(User user)`: this method calls the dummy save method in the repository

Finally, test your controllers by visiting the `/users` and checking it returns the list of users, visit `/users/cf82b98f-3438-4696-8316-c8f6e4010f36` you should find a user with the name *Martha Downing*, and if you visit `/users/dummy` this should return 404 error. You can send a POST request to `/users` with a user created with your data and it should return the same user with a generated ID, to perform the last one you will need to use any REST client mentioned in the installation guide.

# 8 Testing

Write a single unit test to test that the user created in the `UserService` returns a user with the same name. Your test case should follow the convention discussed in the lab, `functionality_input_expectedOutput`.

a) Inject the UserService in the ApplicationTests class (without constructor)

b) Create a test case called `createUser_withValidInput_shouldReturnSameNameAndEmail`

c) Follow the Arrange, Act, and Assert technique.

You can find the full example solution on `https://github.com/Scalable2025/Lab1`