Lab 10 Hometask: Implementation of HTTP with Java/Spring Boot

Examples in the following labs will be shown using Java and Spring Boot, but you are allowed to use a different programming language or framework to implement your web application server (for example, Node.js with Express, Python with Flask/Django, Go with Gin, etc.). The important part is to understand the HTTP concepts and implement the required functionality. Java/Spring Boot is used here as a reference example.

Recommendation: Before you start coding, think about the architecture and purpose of your server/application. A simple starting point is to design a layered structure:
- Database layer: persistence of user and application data.
- Controllers/endpoints: handle HTTP requests and responses (will implement it on the next lessons)
- Services: contain business logic.
- Repositories: interact with the database.

Since the future assignments will cover database, authentication/authorization, access control, and session management, here are several project types that fit well and are simple enough to start with: task management app, simple blog platform, notes app, event planner, book tracker, basic e-commerce prototype, contact manager.

Before completing the tasks make sure you understand the HTTP basics

Know what requests, responses, methods, status codes and headers are so your code maps correctly to the protocol.

- HTTP is *request → response*. The client sends a request (method + path + headers + optional body). The server returns a response (status code + headers + optional body).
- Common methods: GET (read), POST (create), PUT/PATCH (update), DELETE (delete), OPTIONS (capabilities).
- Status codes: 2xx success, 3xx redirects, 4xx client error, 5xx server error. Choose codes carefully (e.g., 200 OK, 201 Created, 204 No Content, 400 Bad Request, 401 Unauthorized, 404 Not Found, 500 Internal Server Error).
- Headers carry extra info (content type, caching, auth tokens). Content-Type tells how to parse a body (for example, application/json).
- Stateless: unless you add sessions, each request is independent.

Every handler you implement must parse request details and form the correct HTTP response. Spring maps HTTP concepts onto Java constructs (controllers, annotations, request/response objects).

Tasks:

1. Set up your Spring Boot project

Create a runnable Java /Spring Boot application with the dependencies you'll need: **Spring Web, Spring Security, Spring Data JPA, Flyway for migration**, Lombok (optional).

- Use https://start.spring.io/ or your IDE (IntelliJ/Eclipse) to create a project.
- Choose: **Java**, build tool **Maven** or **Gradle**, Spring Boot version (stable), and add these dependencies:
    - Spring Web (required - builds HTTP endpoints)

- Spring Boot DevTools (optional: auto reload while developing)
- Spring Data JPA + a database driver (SQLite ,or Postgersql, or any other of your choice) to persist data
- Validation (spring-boot-starter-validation) for input validation
- Spring Security to secure endpoints
- Spring Cache if you later add caching (optional)

After generating:

- Build and run: ./mvnw spring-boot:run or ./gradlew bootRun.

spring-boot-starter-web includes an embedded Tomcat server so you can run your HTTP server with one command.

Tip: For SQLite you may also need to add dependencies (choose any recent Final version):

implementation 'org.xerial:sqlite-jdbc:3.47.1.0'
implementation 'org.hibernate.orm:hibernate-community-dialects:6.6.3.Final'

If you prefer to use different datadase – chose appropriate dependency.

2. Set up version control system

Add **.gitignore** file. It should include **.env** file or any other secret/irrelevant for the project file (for example, ***.db** file).

Initialise your repository with **git**.

3. Set up environment variables with datasource properies for database

Create a **.env** file with credentials for accessing database. Add **.env.example** and write in the list of names of environment variables used for the project.

Create database for your project. If it's SQLite it can be a file **database.db** in the root folder of your project (make sure you won't commit it).

Add datasource properies for your database to application.properties file. Use placeholders: **${name_of_the_variable}.**

To read **.env** file add this to **application.properties**:
spring.config.import=optional:file:.env[.properties]

Tip: If you are using SQLite, you may need to add these as well:
spring.datasource.driver-class-name=org.sqlite.JDBC
spring.jpa.properties.hibernate.dialect=org.hibernate.community.dialect.SQLiteDialect

Write Flyway migration (use dialect of the dtabase of your choice) to create table **users**. Table must have at least id (primary key), username, email, password columns. Email should be unique value. Other columns could be added by your choice.

Migration file should be in the folder **src/main/resources/db/migration**. Name of a migration file should start with **V1__**

Run the application to make sure that it works.

4. Structure your code (keep it simple and layered)

Create the basic structure of your project (packages - **model, repository, service**).

Write code for **User** entity that represents the table users in your database.

Write code for **UserRepository** interface and **UserService** class. User service must have a method to create a user and method to authenticate (login) a user. You have to write your own logic to authenticate a user.

Common layout () – this is **only an example**, you can have different structure:

```
src/main/java/com/example/demo
  ├─ Application.java       // main class (SpringApplication.run)
  ├─ controller/
  │   └─ UserController.java
  ├─ service/
  │   └─ UserService.java
  ├─ repository/
  │   └─ UserRepository.java    // optional: JPA interface
  └─ model/
      ├─ User.java            // entity / domain model
      └─ dto/
          └─ CreateUserRequest.java
```

Controllers handle HTTP concerns, services contain business logic, repositories handle persistence. This separation keeps your HTTP handling clean.

5. Create a simple GET endpoint (basic mapping)

Objective: See the full request → response flow with minimal code. Create a **simple GET controller returning "OK" (REST) or Thymeleaf "hello" page (MVC).**

Explanation:

- A @Controller/@RestController class exposes routes.
- @GetMapping("/hello") maps HTTP GET /hello to a Java method.
- Return type can be a string, an object (auto-serialized to JSON if you use @RestContriller), or a ResponseEntity<T> (gives explicit control over status & headers).

Example:

```
@RestController
public class HelloController {
  @GetMapping("/hello")
  public String hello() {
```

```
    return "Hello, user!";
  }
}
```

If you return an object:

```
@GetMapping("/user")
public User getUser() {
  return new User(1, "Alice"); // auto JSON -> {"id":1,"name":"Alice"}
}
```

This demonstrates Spring translating Java return values into HTTP responses with correct headers (for example, Content-Type: application/json).

6. Create GitHub/GitLab repo and add README describing project setup.

7. Before the next lesson read about controllers and Spring Security (pay attention to SecurityContextHolder, SecurityFilterChain, UserDetailsService), Authorization header, session management.

Sources:

Annotated Controllers, https://docs.spring.io/spring-framework/reference/web/webmvc/mvc-controller.html

What is the difference between @Controller vs @RestController in Spring Boot? https://symflower.com/en/company/blog/2024/controller-restcontroller-spring-boot/

Building REST services with Spring, https://spring.io/guides/tutorials/rest

Spring Security, https://docs.spring.io/spring-security/reference/index.html

What is CRUD? https://www.codecademy.com/article/what-is-crud

Session Management Cheat Sheet , https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html

Authentication Persistence and Session Management, https://docs.spring.io/spring-security/reference/servlet/authentication/session-management.html

Guide to Spring Session, https://www.baeldung.com/spring-session

https://medium.com/@ZiaurrahmanAthaya/how-to-create-session-authentication-using-spring-boot-801320adcd26