**Learning Platform**

2nd year 1st semester

Object Oriented Programming Project

Balogh Flóra

Table of Contents

[1.General description 3](#_Toc156650746)

[2.Technical Specifications 3](#_Toc156650747)

[2.1. Spring Web MVC Framework 3](#_Toc156650748)

[2.2. MVC Architecture 4](#_Toc156650749)

[2.3. Other Specifications 5](#_Toc156650750)

[3.Database Schema 5](#_Toc156650751)

[4.Presenting the application 6](#_Toc156650752)

[4.1. Running the application 6](#_Toc156650753)

[4.2. Users and authentication 6](#_Toc156650754)

[4.3. Home Page 9](#_Toc156650755)

[4.4. Detail Page 13](#_Toc156650756)

[4.5. The comment section 14](#_Toc156650757)

[5.Flow of Execution 15](#_Toc156650758)

[6.Conclusions 16](#_Toc156650759)

# 1.General description

„**Learning Platform**” is a website for sharing knowledge and learning.

When registrating, there are two type of users, „**Teacher**” and „**Student**”. Teachers can create, delete and edit **courses**, add **lectures** to their courses. Students can view and **browse** courses, and view lecture. Students and Teachers can communicate with eachother via **comments**.

The project was developed using **Spring Web MVC framework**, a widely used framework for creating web applications in Java.

From structural point of view, it follows an **N-tier design pattern**.

# 2.Technical Specifications

## 2.1. Spring Web MVC Framework

**[[1]](#footnote-1)Spring MVC Framework**follows the **Model-View-Controller**architectural design pattern which works around the **Front Controller**i.e. the **Dispatcher Servlet**. The **Dispatcher Servlet** manages and dispatches all the incoming HTTP requests to the appropriate controller. It uses **@Controller** and **@RequestMapping** as default request handlers. The **@Controller** annotation defines that a particular class is a controller. **@RequestMapping** annotation maps web requests to Spring

Controller methods. The Front Controller stands first, that is why its name is

like this. After the requests come into this, the dispatcher servlet accepts the requests and decides which controller will be suitable to handle these requests. Then it dispatches the HTTP requests to a specific controller.

## 2.2. MVC Architecture

The **MVC** design pattern separates the project into three main logical components: **Model**, **View** and **Controller**.

**The mode**l handles data logic and interacts with the database, in this project’s case, here are the entities defined: users, courses, lectures and comments.

**The view** is responsible for representing the data, here are implemented the interfaces the user interracts with.

**The controller** handles requests sent by the users, it does not interract with the data.

My project is implemented according to the **N-tier architecture**, besides the above mantioned there were several more „layers” implemented.

**The Service** interfaces encapsulate the logic needed for separarting the controller from the data access. The functions defined in them are implemented in the **Service Implementation** java classes, they interract with the repository and provide services to controllers.

**The Repository** is responsible for the interaction with the underlying database. This package extends **Spring Data JPA** repositories, which is utilized for easy data access and manipulation.

**DTO (Data Transfer Objects)** help transfering data between layers and „**Mappers**” are responsible for mapping entities between DTOs and models.

The view are implemented as **HTML templates** the utilize Thyleaf expressions.

## 2.3. Other Specifications

The project was initialized as a Spring Boot application using **Spring Initializr**. **Maven** was used as the build automation tool to manage dependencies and to simplify configuration.

**Thymeleaf** is integrated as the HTML templating engine, which helps process content. It simplifies the creation of dynamic web pages.

There is a designed template integrated in the project from [[2]](#footnote-2)Start Bootstrap.

# 3.Database Schema

The application is connected to a PosgreSQL database and its schema is represented in the following diagram:

**A diagram of a computer

Description automatically generated**

*Database diagram*

As it is shown, there are **six tables**, four of these are **representing models** in from project: users, courses, lectures and comments. The other two tables are for determinig a user’s roles. In „roles” table are the actual roles hardcoded and in the „users-roles” table are stored which roles do users have.

The tables are interconnected, courses and lectures store the id of the user who created them. In the „comments” table it is also stored, besides the user, the course below which it has been written on.

The connection between the database and the project is created in the **„application-properties”** file, defining the database, the username and the password.

In the project for creating the relations between the tables from the database the tags @OneToMany, @ManyToOne and @ManyToMany were used.

# 

# 4.Presenting the application

## 4.1. Running the application

This project was developed in **Java 17** and **IntelliJ** was used as IDE. Its **dependencies** can be found in the “pom.xml” file, and any additional dependency can be added.

To open the project right click on the “web” file and choose the “Open as an IntelliJ Project” option.

Make sure to **change the fields username and password** in the „application-properties” filebefore running the application.

After running the project, its UI can be accessed by typing into the browser “localhost:8080/register” which will connect the user to the “register” page.

## 4.2. Users and authentication

As a dependency, **Spring Boot Security** has been added, which simplifies the process of securing the application and includes features like key, role based authentication, authorization etc.

The **UserEntity** is defined as the following:

A screen shot of a computer program

Description automatically generated

*User model*

And its DTO:

A computer screen with text

Description automatically generated

*UserDTO*

As mentioned, the first page the user interacts with is the registration form:

A screenshot of a computer

Description automatically generated

*Registration page*

**Email, username, password** and **role** are required for registration, if any of the fields are left empty or the provided emial address is not a valid one, the page gives a **warning message**, which are implemented as it can be seen in the RegistrationDTO class:



*Example of a warning message*

The registration view’s implementation can be found in the templates file and its logic is implemented in the „AuthController” class.

After registering, a message appears, saying that the registration was successfull and the page require to **log in**:

A screenshot of a computer screen

Description automatically generated

*Login page*

The same features were implemeted on login as they are on registration.

There is also a „**Log Out**” button, which returns the login page and tells the user that they have been logged out.

In the User Service interface can be found the methods which are needed to communicate with the data: findByEmail, findByUser, saveUser.

## 4.3. Home Page

The Home Page’s view will vary based on what kind of user just logged in.

If the user is a **teacher**, they will see the **courses they created**:

A blue background with white text

Description automatically generated

*Home page*

Thymeleaf was used for displaying for specific users specific courses, it navigates through the courses using the expression “th:each”, and in this case it checks if the user is the one who created the course: th:if="${user.id} == ${course.createdBy.id}". Only teachers can see the **“Edit”** button and the “**Create Course”** button:

A blue square with white dots

Description automatically generated

This button navigates the user to a new page with a form, where they can add the new course’s title and description:

A white line with blue lines

Description automatically generated with medium confidence

After pressing the “Create” button, with all the fields filled, the new course will appear:

A screenshot of a web page

Description automatically generated

The Course model was implemented in the Model package, where its fields are defined. The photo upload has not yet been implemented, but for further improvements it can be.

If the user wants to update the course’s title/description, it can be done with the “Edit” button:

A screenshot of a computer

Description automatically generated

And after changing the fields, the updated version will appear at the Home Page:

A white background with black text

Description automatically generated

If the user is a **student**, on the homepage they can see **all the courses uploaded** by teachers:

A screenshot of a computer

Description automatically generated

For the students only the “view course” button is available and a **search bar**, where students can search. The logic behind this is implemented in such way that it can search in the course’s title:

A screenshot of a computer

Description automatically generated

*Example for searching for “Literature”*

For students the “Create course” button is also unavailable.

## 4.4. Detail Page

For a **teacher**, the following view is available:

A screenshot of a computer screen

Description automatically generated

*View of the detail page for teachers*

Here are **listed** all the **lectures** added, which can be edited and deleted, and they can also add new lectures or delete the course.

When **adding a new lecture**, the following form will appear:

A close-up of a computer screen

Description automatically generated

*Create lecture form*

Pressing the “**Choose File**” button a window will appear which lets the user browse their file. However, the logic here couldn’t be implemented, the application stores the name of the file, but it can’t store the actual file and create a link for students to open it.

The “**Delete this course**” button deletes the entire course.

The “**Edit**” and “**Delete**” options work the same way it was presented for course entity.

For **students**, only the “**Open file**” button is available:

A screen shot of a computer screen

Description automatically generated

*View for students for the detail page*

## 4.5. The comment section

**The comments section** is implemented in the course’s detail page, they are available for both **students and teachers**.

There is a form at the end of each course:

A white background with blue and white squares

Description automatically generated

Pressing the “Submit Comment” button, the new comment will appear on the page:

A white text box with black text

Description automatically generated

There can be seen that the **“Edit”** and **“Delete”** buttons are only available for the user who wrote the comments.

The “Edit” button navigates to a new page where the text of the comment is loaded and the user can update it:

A blue rectangle with white lines

Description automatically generated

The “Delete” button deletes the comment.

# 5.Flow of Execution

All the entities have a model, a controller, a service, a repository, a DTO and a mapper.

The http requests are received by a controller, which delegates tasks to the service layer. The services implement the “Service Implementations” where the methods are defined, which process request and interact with the repositories. The repository accesses the data from the database. Mappers handle data transfer from DTO to the effective entity.

All the presented functionalities of the application are implemented along this principle.

# 6.Conclusions

This application is created for teachers and students, to share information, to learn from one another and to communicate. The views and the actions available differ based on what type of user is currently logged in.

All **CRUD** (Create, read, update, delete) operations were implemented on entities course, lecture and comment. These are the functionalities presented in the “Presenting the application” section.

**Further improvements** can be added, such as creating an Admin user or managing the file upload and open operations.

1. https://www.geeksforgeeks.org/spring-mvc-framework/ [↑](#footnote-ref-1)
2. https://startbootstrap.com/templates [↑](#footnote-ref-2)