

COMP30860 Laboratory: Implementing Server Side Logic

This laboratory focuses on getting practice at implementing Server Side Logic for the Notelt web application. As a starting point for this lab, you are expected to use my versions of the static HTML pages you produced in lab 2. These pages are part of the template project. This template will also contain an outline Spring Boot MVC project to get you started.

Summary of Tasks to be completed:

1. Create a Data Model for the application.
2. Implement Object Persistence using JPA.
3. Implement a “/create” endpoint to handle the form submission.
4. Implement a “/browse” endpoint to display notes.
5. Implement a “/view” endpoint to display a note.
6. Ensure all hyperlinks are passed to the controller (no direct links to static HTML pages)
7. Configure application to work with MySQL
8. Deploy your solution using docker-compose.

Marking Scheme

Your grade will depend on the number of the above tasks that you complete. Some additional criteria / advice can be found in the notes below.

Grade	Criteria
D	Complete any 2 tasks
C	Complete any 4 tasks
B	Complete any 6 tasks
A	Complete all 8 tasks

You should download the template project from: <https://gitlab.com/comp30860/labs/lab3>

NOTES:

- I advise you to use the H2 in memory database while developing your app. You should then switch to MySQL for your final deployment.
- In the template, the sample HTML pages are in the “resources/static” folder. There should be no HTML content in this folder for the final version.
- Your app should run if you type in <http://localhost:8080>. I have explained how to do this in the class notes.

Task 1: Create Data Model

The only element to be modelled in this application is a note. Don't try to model for later versions of Notelt, do *just enough* to implement the current version.

- (a) Notes have versions, but v0.1 of the app only allows notes to be created and viewed (all notes will be version 1).
- (b) Because there is only one version of each note, the created data and last updated dates will be the same.
- (c) Each note needs a unique field for a primary key – the title is not a good fit. Also - numbers are often better than strings.

Task 2: Implement Object Persistence using JPA

This is relatively straight forwards for a basic scenario like Notelt. If you write more than 10 lines of Java code, you are doing too much.

- (a) You will need to configure a database to get JPA working...

Task 3: Implement a “/create” endpoint


This endpoint should be used to handle form submission. As per the lectures, I recommend that you do this with a form that uses the POST method (so the form data is part of the HTTP Request body).

- (a) The method should store the data in the database.
- (b) After storing the form data, you should be redirected to the “index” page.

Task 4: Implement a “/browse” endpoint

This endpoint should use a templated version of the “browse.html” page. You will need to retrieve the current notes from the database and pass them to the template for display.

Task 5: Implement a “/view” endpoint

**Notelt**
For all your note-taking needs

View Note

Back :B1

Title: 2018 Christmas Shopping List

Date Created: 01-Nov 2018

Note:

Wife: Perfume
Coral: Fireman Sam Toy
Tanya: XBox

P4: View Note Page

- (a) This page should present details of a single view
- (b) The layout should be similar to the “Create Note” page
- (c) The back button should return the user to the “Notes List Page”

Task 6: Ensure all links are passed via the controller

The final version of the app should have no static HTML pages. This means that every page should be accessible through the controller.

- (a) A GET request to “/” should load the index page
- (b) A GET request to “/create” should load the “Create Note” page
- (c) A GET request to “/browse” should load the “Notes List” page
- (d) A GET request to “/view?id={id}” should load the “View Note” page

All navigation buttons (the big buttons on the index page, and the “back” buttons) should result in a HTTP request to an appropriate end point (from the above list).

Task 7: Configure the app to use a MySQL database

All of this is covered in the notes. You will need to use docker to deploy a mysql instance.

Task 8: Deploy the app using docker-compose

Docker-compose lets you define “docker applications” which consist of multiple docker images. You should create a single `docker-compose.yml` file that deploys two images: a mysql image and a spring-boot image. The second image should deploy the application, with port 8080 mapped.