Software Design and Engineering

Lab Document

High Level Purpose Statement:	The purpose of this lab is to use the Spring Boot framework. My goal is to utilize Spring Boot in my final project, which involves creating a web app that generates fast, efficient course schedules for students. **Addendum (Part II):** For the second part of this lab, I am going to add some features to enhance the current web application that I have. I plan to explore smooth deployability and adding pages to my current web app.
Experimental Design:	This lab will be divided among the following components: • Spring Boot Framework configuration (REST Controller, Service classes, DAO classes) • Backend Configuration • Frontend Configuration • Compatibility (ensuring each component can communicate with each other and work together efficiently) **Addendum (Part II):** I am going to try to get Docker to containerize the database to reduce the number of steps required by users to run the app. I also plan on adding the schedule configuration page and an empty schedule generation page.
Resources Available:	**Websites (Tutorials/Documentation):** • https://www.baeldung.com/jpa-entities • https://www.baeldung.com/jsf-spring-boot-controller-service-dao • https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/jdbc/core/JdbcTemplate.html **YouTube:** • https://www.youtube.com/watch?v=MHdYN4cqsDs **Postman** (https://www.postman.com/downloads/) I also plan on utilizing Copilot to assist with the frontend configuration. JavaScript is pretty scary.
Time Estimate:	I am estimating this project to take between 12-20 hours. *(I was soooooo wrong. 45-50 hours later, here I stand.*) **Addendum (Part II):**

	Because I did *so* much of the work for the last lab, I am only expecting to spend 3-6 hours on this lab.
Experiment Notes:	**Spring Boot Framework configuration (REST Controller, Service classes, DAO classes)** This was a learning curve for sure! Although implementing Spring Boot into my project was simple, learning each HTTP request and configuring handlers was a challenge. I installed Postman, which made this process *so* much easier for me. By trial and error, I was able to slowly build an understanding of REST calls. I already had a decent understanding of PostgreSQL and creating an application that interacts with the database (hence DAO-wrapped objects), but man, trying to deploy the database was *extremely* challenging, for some reason. More on this later. **Backend Configuration** To configure the backend, I implemented a Node Express framework to assist with routing REST calls. This was not *too* difficult to learn and implement. The main challenge was configuring CORs *and* knowing it was an issue. All of my HTTP requests were being blocked and returning errors. I probably spent about 2-3 hours banging my head against a wall because of CORs. It had a simple fix, but I did not realize how important it is to manage its configuration. **Frontend Configuration** This was a frustrating process, mostly because of the confusing syntax of JavaScript. During the process of configuring the frontend, I had to make adjustments in the service classes and DAO classes. It was slightly difficult to get the frontend to properly display the courses and categories. **Compatibility** I am at a loss for words when I think about how frustrating this component was. I had to try and fail at so many different things to get the different components to be compatible with each other. This was not a linear project. I had to work on the different modules simultaneously, make adjustments, and run tests the entire time. When one component finally functioned the way I wanted to, there was something else wrong with another. I had to work diligently to get the components to not be so tightly coupled. **Addendum (Part II):** **Employing Docker** I spent a few days trying to get Docker to containerize the database. Even after reaching out to multiple sources, watching countless videos, and reading articles, I could not get it to work successfully. I hope I am able to get this figured out at some point. **Schedule Configuration and Generation Pages**

	These were relatively easy to set up. For now, I decided to substitute the database logic with confirmation messages on the page. I will employ the logic once the degree catalogue is implemented.
Results:	The current features are: • Individually listed categories for degree program • Remaining required credit hours in each category • Interactive courses from which users can toggle completion Although I originally wanted to generate at least a *basic* schedule by the end of this project, I think it's for the best that I stop at this point. Building the full-stack web app was much more challenging than I anticipated, so I did not have as much time to add more features. I will say that I am very disappointed in the deployment of this web app. I originally wanted to use Docker to containerize the app, but I was having issues with connecting to the database. I scratched that idea, and then once I thought the project was finally finished, I submitted it to a guinea pig, and the web app would not even build on his system. So, I tried to utilize Docker again… five hours later, I was still having the same issue with web app connecting to the JDBC driver via a Docker container. After giving up on Docker, I cloned the repo to a separate device and used it to test new modifications to the web app. I hope that I can figure out how to use Docker correctly eventually… **Addendum (Part II):** The current features are: • Individually listed categories for degree program • Remaining required credit hours in each category • Interactive courses from which users can toggle completion • Prompt for user to select of semesters to make schedules for • Prompt for user to enter time constraints The mission to employ Docker failed again… although I'm very disappointed (again), I know that I should probably explore other options. Maybe I should try uninstalling and re-installing Postgres and Docker.
Consequences for the Future:	I think that next time I build a full-stack web app, focusing on the deployment should be an early step. Instead of spending so much time trying to add features, I should have ensured everything was compatible on someone else's computer first. Though, I could argue that I would have probably wasted a lot of time trying to

	containerize the app (and failed) and run out of time on adding features. Interesting thought. **Addendum (Part II):** I followed the advice I gave myself from the last lab to explore deployability at the early stage. I think that was an excellent decision, although I did not get the results I was hoping for. I did most of the configuration in the last deadline, so I believe that is my biggest takeaway as far as Part II goes.