SWE 573 - Application Development Progress Report

Nilay Aydin

2022719204

Building Application Locally:

To build and run the application locally, ensure you have Java Development Kit (JDK) 17 or higher, Apache Maven, and optionally Docker installed on your system. Begin by cloning the repository from GitHub using the command *git clone https://github.com/niloaydin/SWE-573-HW.git* and navigating into the cloned directory. Use Maven to build the application by running *mvn clean package*, which compiles the source code, runs tests, and packages the application into a JAR file in the target directory. To run the application, execute *java -jar target/community-application-0.0.1-SNAPSHOT.jar*, and access it at *http://localhost:8080* in your web browser. Alternatively, you can run the application in a Docker container by building the Docker image with *docker build -t community-application* , and running a container based on that image with *docker run -p 8080:8080 community-application*. Once the application is running, you can interact with it through your web browser. For testing purposes, I have added both postman collection url.

Deployment url = https://swe-573-hw.onrender.com
Repository url = https://github.com/niloaydin/SWE-573-HW
Issues = https://github.com/niloaydin/SWE-573-HW/issues
Wiki pages = https://github.com/niloaydin/SWE-573-HW/wiki
Postman Collection url =
https://documenter.getpostman.com/view/20862524/2sA3BuXUdF

Report:

This report contains the information regarding the process of developing a community-specific information application required for SWE 573. Initially, I would like to explain my development choices. Firstly, we were provided with 2 options for developing language and framework: python-django and java-spring boot. I decided to choose java-spring boot on the grounds of learning a new framework and enhancing a programming language that I am not familiar with. I also decided to use Postgresql for the same reason. Another decision point was to create the backend first, then proceed to the frontend. Designing and creating a backend is harder for me, so I wanted to focus my main energy for the backend. I decided to separate the backend and the frontend; thus, not using server-side rendering. I will use the React framework for the frontend.

First and one of the most challenging steps was to create Design points for me. The first

step towards designing the application was to create requirements for the application. Requirements really helped me to design the application system. After creating requirements, I also created diagrams that would visualize the functionality of the program. Creating the ER diagram was the hardest part of the design points. After researching relational databases, I created the abstract for the ER diagram and continue to polish it.
Before starting to write the application, I researched how Spring Boot works, and understanding this took quite some time. As none of the frameworks I had used before resembled it, the details were quite challenging. To progress the project in a healthy manner, I set a roadmap using GitHub issues. However, to be honest, I sometimes deviated from the planned path. Since I hadn't used GitHub for project management before, it took me a while to get used to it. When determining the issues, I tried to break them down into as small pieces as possible, both to make them more manageable and to progress the project with a sense of accomplishment. At the beginning of the project, I decided to prioritize implementing authentication. As there were many resources available online for authentication and it wasn't project-specific, I implemented authentication in my project based on the resources I found on the internet.
During the authentication implementation, I gained a better understanding of both the framework and the project. Since Spring Security is a challenging and extensive topic in itself, I tried to research and understand it as much as possible during authentication, but I can't say I was very successful. After authentication, I slowly started to build the project structure. While designing the structure, I researched the design patterns within the framework and chose the repository pattern. When implementing the functionalities, I tried to implement the most basic features first. The implementation process was both challenging and extremely enjoyable. With each implemented functionality, I gained confidence and continued to develop the application with even more motivation. The progress of my classmates during the lessons also reassured me. Seeing that others were experiencing the same confusion as me reduced the pressure I put on myself. Additionally, discussing our progress regularly during the lessons and examining the details of the application also helped me manage the application process.
Throughout this process, I paid special attention to writing POST and GET requests. Focusing on these requests helped establish the basic structure of the project and its functionalities, making it easier to write UPDATE and DELETE requests later on. Among the fundamental features, I implemented functionalities for creating communities, templates, and posts, along with allowing users to join communities. One of the most challenging aspects was establishing and implementing relationships between relational databases. Since I was accustomed to NoSQL databases, adapting to relational databases while considering their

features was quite challenging. However, as the project progressed and I conducted necessary research, I gained a better understanding of relational databases.

Another challenge I encountered was setting up many-to-many relationships. JPA and Hibernate handle entities and relationships internally, making it difficult to understand their workings. Instead of using annotations to set up many-to-many relationships, I opted for manual configuration. Although this decision worked for certain functionalities, I continued to face similar challenges, particularly in the process of creating posts. Due to my lack of familiarity with relational databases, creating community-specific posts and post templates took almost days. Nevertheless, despite these difficulties, I successfully managed to create community-specific post templates and enable post creation within the scope of the MVP.

Moving forward, remaining functionalities include commenting and voting on posts. Additionally, I'll focus on organizing the code and implementing exception handling. After organizing the backend code, the next step is to develop the frontend of the application as I have discussed with my instructor, Suzan Uskudarli. Given that I've outlined all endpoints and logic in the backend, I don't anticipate encountering significant challenges while developing the frontend. My prior experience with frontend development leads me to believe that after defining the logic in the backend, I'll be able to easily develop the frontend.

Regarding deployment, since I do not have a frontend, I added a Swagger-ui, so people can test the endpoints without needing to use Postman. I will add my deployment link; however, it’s important to keep in mind that my Security Config only allows some endpoints to be accessed without authentication. I configured my Swagger-ui settings to create JWT so authorized endpoints can be tested. My swagger-ui endpoint can be reached through this link. There are some test endpoints, please disregard that. Moreover, some of the request body examples also include the entries from the relationships of the database entities. Therefore, I have added a postman collection. Postman collection can be reached through this link. As I mentioned before, since I have not implemented exceptions, all exceptions return HTTP 403 responses. There are three main reasons for exceptions: JWT token might be expired, there are duplicated entries on the database, or some of the fields are wrongly filled. I also would like to state that, I wanted to add exception handling; however, creating a post functionality took me days to figure out. All of these missing functionalities will be implemented before completing the application development. Lastly, as a reminder, when first sending a request to a deployed url, it can take a while because when the url is not used, it will spin down; thus, restarting the server might take up to 50 seconds for the first request.