Change Impact Report

ECE651 Project Evolution 3

Team 5

Haolou Sun, Haoyuan Geng, Hiep Nguyen, Hunter Shen, Yuyu Hsieh

1. Changes in data storage

a. Added some functions for Course Management

In order to make the Course Management app more functional, more intuitive, we have expanded some functions in SectionDao, CourseDao, such as getAll() for the backend of the app to extract the data from the database, then show them to users.

b. Utilized Hibernate

We've adopted the Hibernate framework for interacting with the database. In contrast to our previous approach, we've discontinued the use of custom-designed DAOs. Instead, each of our DAOs now simply extends BaseDao<T>, with the type T determined by its own entity type. For example, StudentDao extends BaseDao<Student>. This significantly reduces the manual effort required for designing DAOs compared to before.

2. Web App Interface for Student/Faculty

a. Impact on data model

To accommodate the many-to-many relationship in Spring Boot ORM, we've changed the courseSections field in the Professor model from **Set<String>** (where String represents the ID of a course section) to **Set<Section>**.

b. Impact on server

— Used the Spring Boot framework..
— Utilized HTTP requests for communication with frontend instead of sockets.
— Used Java Persistence API, replacing the previously designed DAO.
— JWT Authentication tokens are employed to verify user identities. (implemented by using spring security)

c. Impact on client code

— Used the Spring Boot framework(MVC).
— The application has been upgraded from a terminal-based interaction program relying on plain text to an HTML-based web interaction application.
— Utilized HTTP requests for communication with backend instead of sockets.

— We've designed several controllers to handle interaction with the backend, linking their methods to corresponding URLs using GetMapping and PostMapping. When accessing a webpage via a URL, the system automatically invokes the controller methods to connect to the backend, retrieve information, and then pass it to HTML for rendering.
— We've created numerous HTML templates for rendering pages and designed corresponding CSS styles for them.

3. JavaFX for Admin app

a. Impact on frontend

The frontend of this app has undergone significant changes. The interface has transitioned from a text-based Command Line Interface to a graphical JavaFX application. While the design structure and the sequence of functions remain intact, a CSS file shared between the Admin app and the Course Management app has been introduced to enhance the app's appeal and uniformity.

With GUI, the application utilizes the benefits of having a graphic interface by showing more information in rows and columns, and more ways to receive responses from users (through input text, button, click,...). This significantly improves Users Experience and productivity.

b. Impact on backend

Several backend functions have been modified to accommodate JavaFX's user interaction methods, although the core algorithms and logic are preserved. Instead of using the ReadLine function from BufferedReader for input from the terminal, inputs are now processed from the JavaFX UI elements such as window stages, buttons, text fields, or password fields. A wrapper main class (Launcher.java) has been incorporated to package the app, concealing all JavaFX dependencies within the main class.

4. JavaFX for CourseManagement app

a. Impact on the original frontend

The frontend of the Course Management app has been completely redesigned to enhance user interaction and leverage the capabilities of a desktop application with graphical interfaces, as opposed to the previous text-only terminal interface. The introduction of a shared CSS file between the Admin app and the Course Management app helps achieve a more attractive and consistent appearance.

With GUI, the application utilizes the benefits of having a graphic interface by showing more information in rows and columns, and more ways to receive responses from users (through input text, button, click,...). This significantly improves Users Experience and productivity.

b. Impact on the backend

Mirroring the changes in the Admin app, some backend functions in the Course Management app have been restructured to align with the JavaFX interface. The app's structure was modified to support a more intuitive frontend interface. New classes have been developed to manage the creation of sections and lectures independently. A wrapper main class (Launcher.java) has been implemented to package the app, effectively hiding all JavaFX dependencies in the main class.

5. Applying Spring Framework

For evolution 3, we switched back to spring for both server and client in the attendance app. We also get rid of flag in the requests, since now we can utilize spring to have different url end point for each requests.

6. Multiple users using at the same time

For spring application, every time an HTTP request arrives at a web service, tomcat creates a thread to process the request. By default, the max number of threads are 200, which means 200 requests can be handled at the same time.

7. Password encryption

For evolution3, we utilized BCrypt library, which might be secured than sha-256.
We also do not need to store salt separately since now the salt will be included as part of the hashed password.