Daniel Fiala

Meeting Planner Protocol

The goal of this project was to make a functional desktop application for the purpose of creating and managing meetings and the purpose of those meetings. The application was developed using **Java** and the **JavaFX** framework, as well as other tools like **Maven**, **Spring Boot**, **PostgreSQL** and **Docker**. Some of the core functionalities include, like mentioned above, creating, managing and deleting meetings and notes, searching for meetings using a search bar and the generation of a report for meetings in the form of a PDF file. The development process followed layer-based architecture and the MVVM design-pattern.

The core structure of the application was divided into 4 different layers, all serving a different purpose and only communicating with one another through a top-down flow.

- **UI Layer**
  The UI Layer presents the user with everything they can actually interact with. It contains the Controller and ViewModel classes and communicates with the FXML files through the MVVM pattern. The View the user sees is a connection of smaller components like the PlannerView and MeetingNotesView among others, which communicate with the respective Models through the Controllers.

- **Business Layer**
  The Business Layer contains the core logic of the application. It consists of the two service classes MeetingService and ReportService. The ReportService contains the logic to generate the PDF files, while the MeetingService handles the requests handed to it by the UI Layer.

- **Data Access Layer**
  The Data Access Layer is the one actually responsible for the communication with the PostgreSQL database. This is handled using JPA interfaces using the Spring framework, which implements all the functionality and operations necessary to make queries to the database.

- **Model Layer**
  The Model Layer contains the actual entities of the application, Meeting and MeetingNote. The application uses these models to create and manage tables in the database. Each of them have different configurations like @OneToMany or @ManyToOne, based on the requirements.

The following Design Patterns were used during the development of this application:

- **MVVM**
  The MVVM pattern builds the main architecture of the entire project. Like briefly mentioned above, it’s main purpose is to split the FXML files from the ViewModel and other parts of the application.

- **Dependency Injection**
  The Dependency Injection design pattern was a big part of the development process, as its use can be found through many parts of the application, and it’s handled by the Spring framework. Dependencies are not created by each class itself, but declared in the constructor and injected by Spring to make sure the components are only loosely coupled.

- **Singleton Pattern**
  By making use of Spring annotations like @Component or @Service, the application makes sure that only one instance of each part exists to avoid complications.

- **Facade Pattern**
  The application follows the design of the Facade pattern, by providing a simple interface to the UI Layer, while hiding all of the logic and necessary steps in the other, loosely-coupled layers. By separating the UI Layer from the DAL, for example, the UI Layer wouldn’t have to be changed because of changes made to the DAL, but rather the BL, for example.

The **Unit tests** were written for the Business Layer of the application, since it’s a critical part of the application because of the core logic it handles. Complications in this layer would be more worrying than complications in the UI layer.
10 tests were written for the service, covering functionalities like saving, managing and deleting meetings and/or their notes to see if the functionalities work as intended. Some tests were created to make sure that database functionalities like “cascade” and “orphanRemoval” also work as intended for big parts of the Notes functionality.
The application also makes use of two tests implemented using TestFX, to simulate actual user input by making it click buttons and typing in fields in the UI to verify that the integration also works as intended.

Link to GitHub Repository: https://github.com/dayfx/MeetingPlanner.git