FH University of Applied Sciences TECHNIKUM WIEN

Tour Planner

Software Engineering 2 SS2023

Lorenz Duelli, Jasmin Duvivié

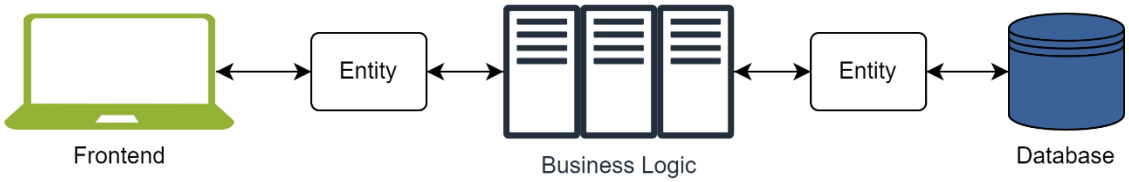
www.technikum-wien.at

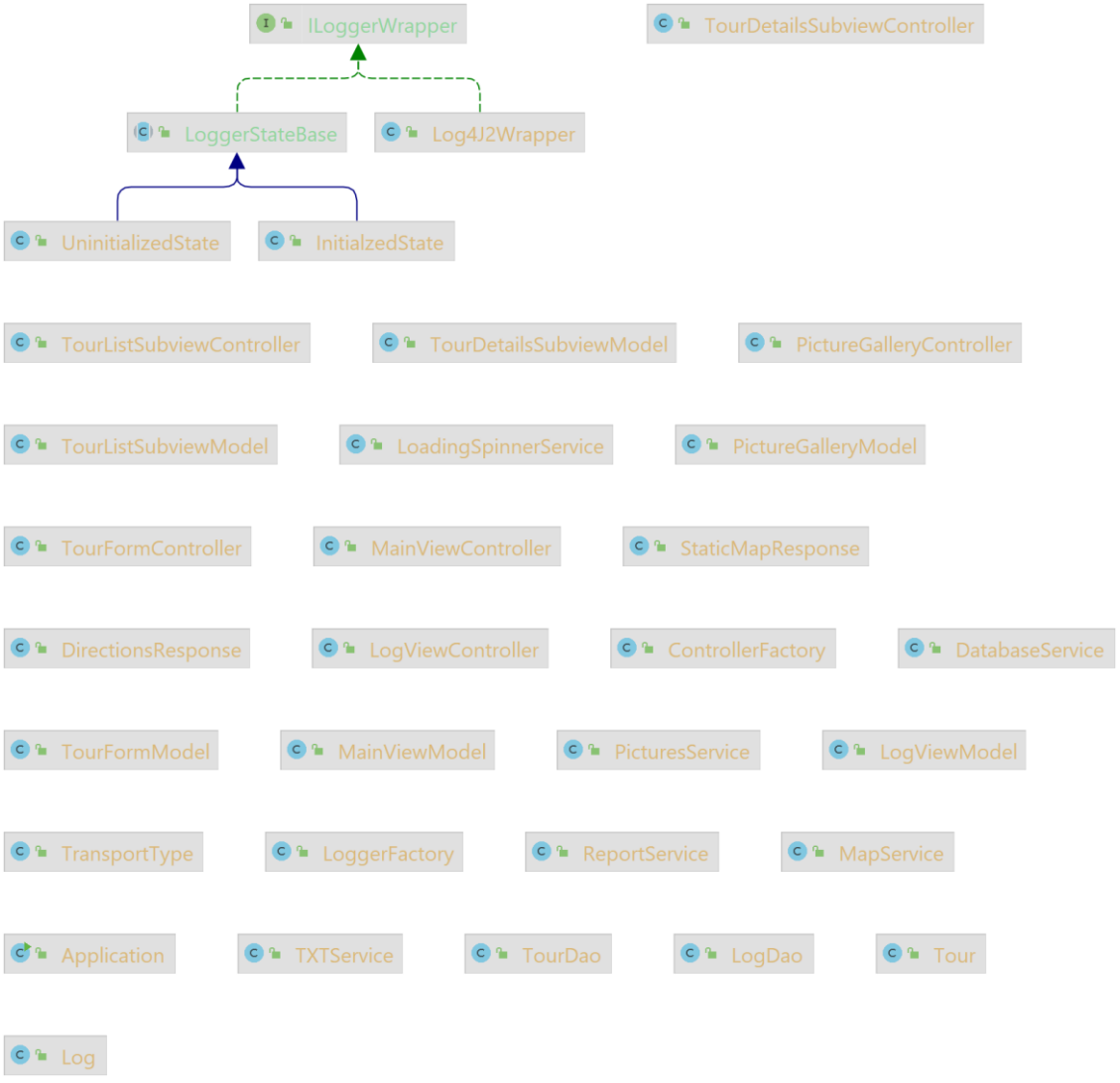
Architecture

We implemented a layered architecture for the Tour Planner as instructed in the SWEN course. The structure encompasses these layers:

- Frontend Layer
- Business Logic Layer
- Database Layer

Each layer only invokes methods of either their own layer or the layers beneath them. The data is passed on from layer to layer as annotated entities. These entities are mapped to the database with the Hibernate Object-Relational-Mapper and the Jakarta Persistence API.



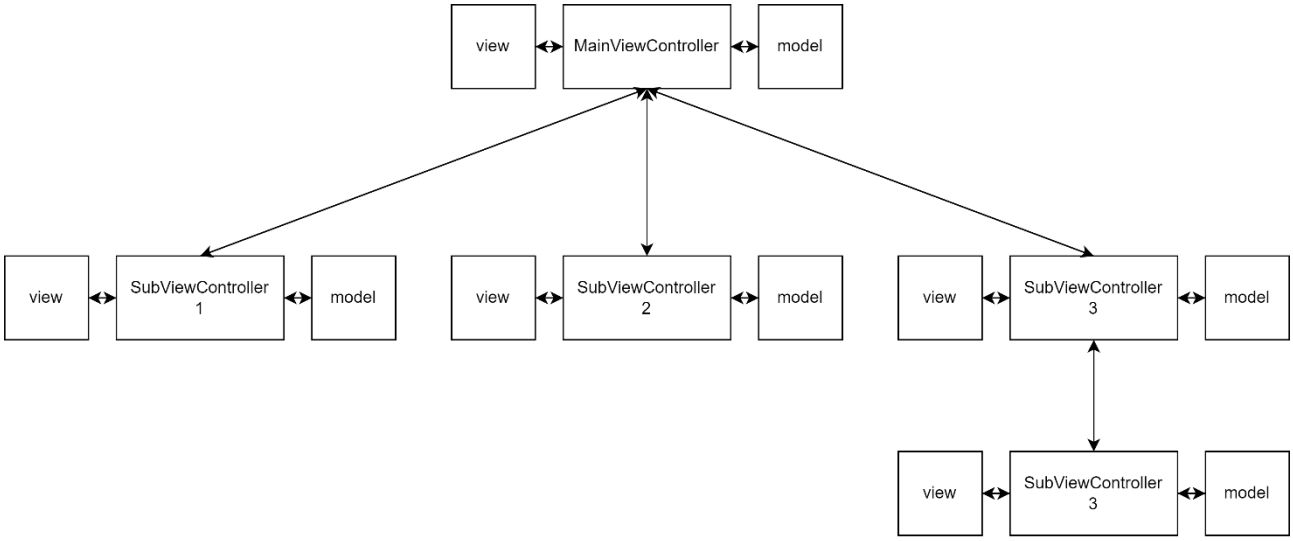


Frontend Layer

The Frontend Layer is structured according to the MVVM-Principle:

- the user sees and interacts with the contents of the *.fxml files (views)
- controllers (viewmodel) bind data between the views and their models
- the models (model) invoke the services of the business logic layer

Each controller binds between one *.fxml and one model only. The mainview includes the other reusable UI components (subviews). The MainViewController acts as a kind of parent. It gets notified by the controllers when something has been updated and distributes the information to the other controllers, so that each controller only invokes their own model and is responsible only for their own tasks. Other views can also include subview components if applicable.

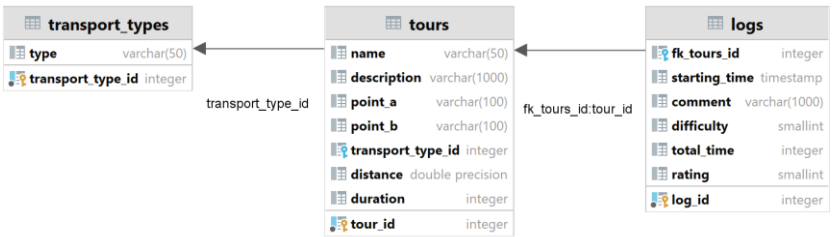


Business Logic Layer

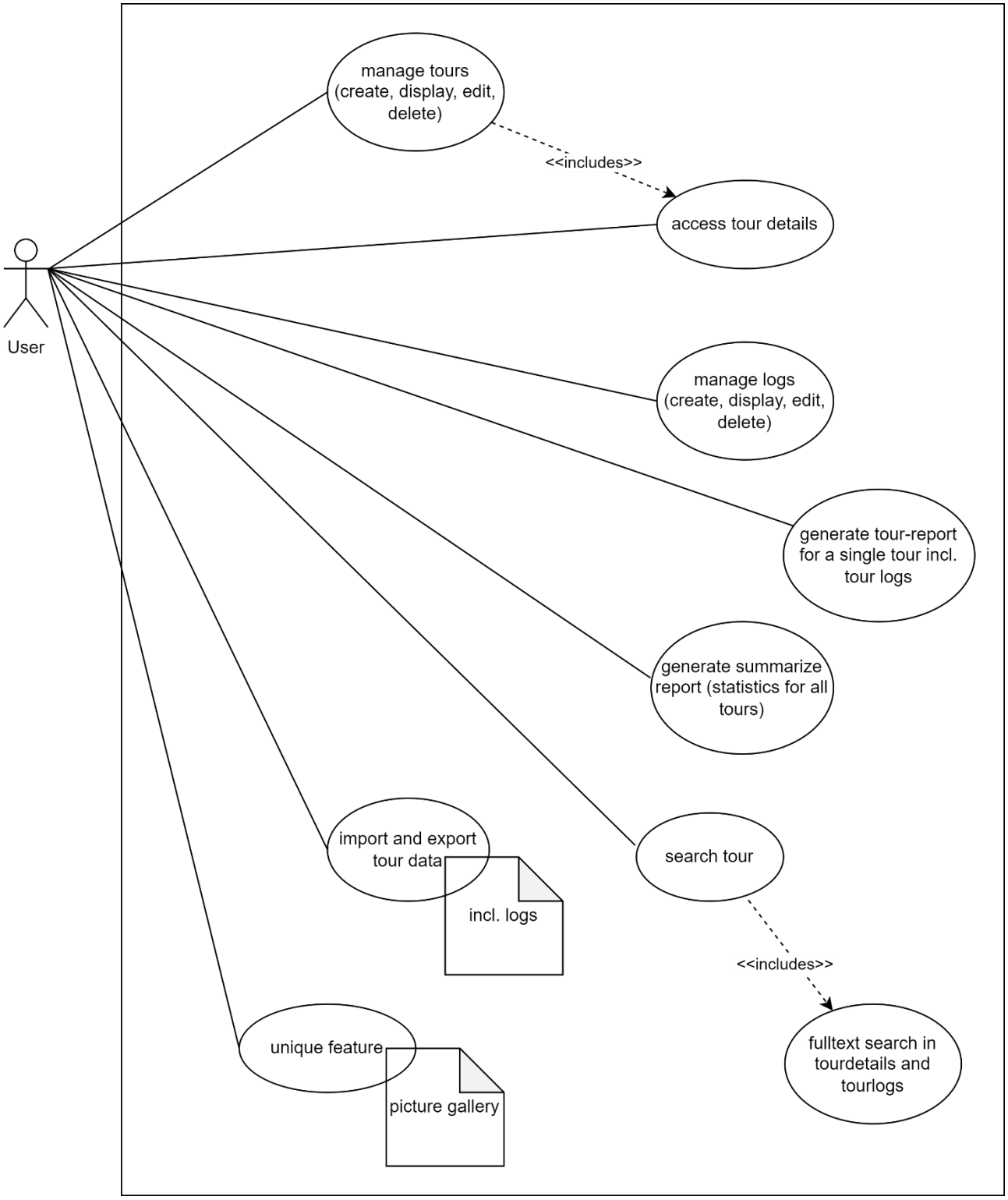
This layer carries out the functionality of the Tour Planner. In our application this layer is organized by services. These encompass requesting directions and images from the MapQuest API, handling image uploads, generating reports as well as importing and exporting tour data and invoking the Database layer when needed.

Database Layer

This layer is responsible for interacting with the database through DAOs (Data Access Objects). The database schema looks as follows:



Use Cases

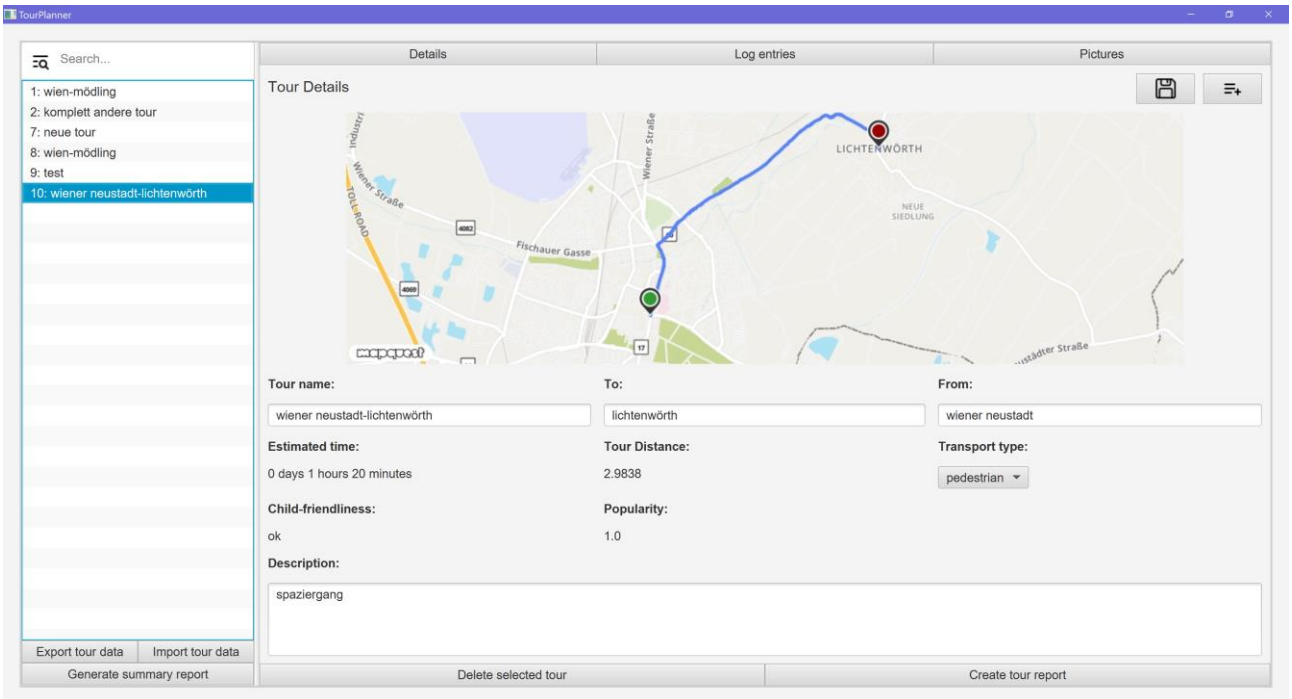

manage tours (create, display, edit, delete)
<<includes>>
access tour details
User
manage logs (create, display, edit, delete)
generate tour-report for a single tour incl. tour logs
generate summarize report (statistics for all tours)
import and export tour data
incl. logs
search tour
<<includes>>
fulltext search in tourdetails and tourlogs
unique feature
picture gallery

User Experience

We designed our application in a simplistic and classic looking way for the following reasons:

- we do not want the user to be overwhelmed by too much information. Thus, our information architecture is simple. Rather than bombarding the user with words, we use simple images such as s little disk icon for save and a little trash can for delete.

- we do not want the user to be overwhelmed by flashy colours and effects. Thus, our design does not go far beyond simple and plain javafx.

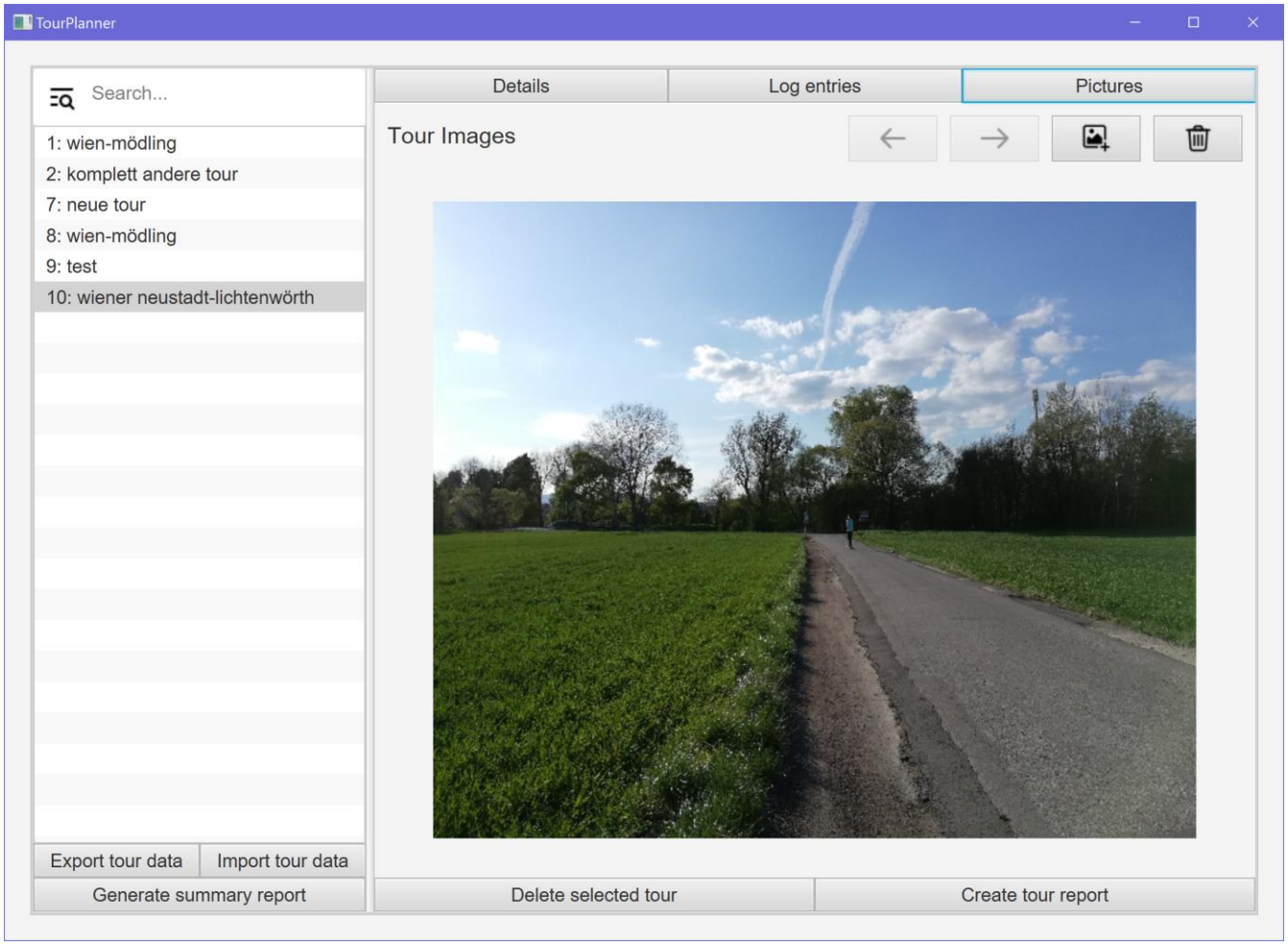


- None of us are particularly good designers so we could not be bothered to wrap our heads around fancy gimmicks and tricks that go beyond a spinner.

Loading...

Unique Feature

As a unique feature we have implemented the functionality to upload pictures, which are associated with a tour. We know this kind of feature from e.g., Google Maps, where it helps people to get a concrete idea about the places they are going. Similarly, it is possible in our application to upload as many pictures as one likes to a specific tour to further support the usefulness of the Tour Planner.



Design Patterns

The following design patterns have been implemented:

- Factory Pattern (Controller Factory and Logger Factory)
- Publisher Subscriber Pattern (when a controller makes modifications which affect the state of the system, it publishes its changes. The MainViewController subscribes to these changes and invokes every other controller so that they carry out their necessary actions.

Unit Tests

We mostly test the models and services, where applicable. The controllers simply bind between the *fxml. files and the models and invoke methods of the models, so there is no need to test the controllers. The models have been tested where necessary i.e., where they carry out calculations to display the data which they receive from the business logic as they need it for the frontend. Most of the times however, the models only invoke services, in which case in makes sense to test the services but not the models. Thus, the services and models are most heavily tested.

Tracked Time

We have met 10 times to work on the project together. On average, these meetings lasted 4 hours, which sums up to 40 hours per person and 80 hours for the team. Additionally, we have invested approximately 5 hours each to prepare for each meeting, which sums up to 50 hours per person and 100 hours for the team. Thus, we have spent roughly 90 hours per person and 180 hours in total working on the project.

Lessons Learned

There were some aspects in this project which highly frustrated us at first, such as setting up Hibernate and including dependencies generally. Last semester, everything which was included in the pom.xml seemed to work instantly, but this semester, we very often had problems at start-up because e.g., lines were missing in the module-info.java file, that were not placed there by the IDE but which were also completely new to us. Once it was up and running however, the Hibernate ORM was a very useful tool.

We also feel like we have learned a lot about how git and a proper git workflow works. At first, we often had merge conflicts or were simply scared of merging because we did not have much experience with it and were afraid to somehow lose our progress. With some iterations though, we got used to branching and merging a lot and feel like this was a very valuable experience.

Finally, we learned a lot about the layered architecture and the MVVM principles. They weren't easy to grasp at first, but after talking about it again and again and trying out our own implementation, it finally makes sense and seems like a very good and robust design choice.

Link to git

https://github.com/smoothjass/swen2-project

The repository is private; however the lecturer has been invited to collaborate and anyone interested is welcome to email us a request to: if21b145@technikum-wien.at or if21b144@technikum-wien.at

Additionally, the repository will be public between the date of the hand-in and the recording of the grades.