Raza Ghulam, Felix Piber

TourPlanner Documentation

Structure

The App is structured in 3 layers: Presentation Layer, Business Layer and Data Layer

Presentation Layer

Responsible for handling the user interface and user interactions. It consists of multiple parts.

- View
  The View displays the data to the user interface, reacts to user interactions and sends them to the ViewModel.
- ViewModel
  Acts as a bridge between the View and the Business Layer. It exposes the data and behavior that the View needs.

Business Layer

Contains the core business logic. It provides the data and communicates with the Data Layer.

- Model
  There are 3 Models: RouteModel, TourModel and TourLogModel. These Models contain the data, that is fetched from the database or supposed to be inserted into the DB. They are used in the viewmodel, to display the data.
- Service
  The services are responsible for communicating with various APIs and the Data Layer. They create Maps, Routes, PDFs and interact with the datalayer to insert and get data from and into the DB.

Data Layer

Deals with the data storage, such as the retrieval and insertion of data.

- Entity
  Classes that contain the data for OR Mapping
- Repository

Libraries

- Jackson – provides easy de/serialization for JSON Objects
- ControlsFX – additional controls for JavaFX
- Log4J – Easily maintainable configuration, granularity of log levels helps with debugging and error tracking
- IText – Simplifies the process of creating pdfs
- Lombok – Very useful for readability, eliminating repetitive code

Design Pattern

Observer Pattern

Notify multiple objects about any changes to the object that is being observed, making it possible for other objects to adapt to said objects, whether it is to display something specific or carry out an event.

Factory Pattern

Defines an interface for creating objects but lets subclasses alter the type of objects that will be created.

MVVM

Separates the presentation logic (ViewModel) from the user interface (View) and application data (Model)

Unit Testing

For the **Business Layer**, various services are tested, as these services are critical for the core functions of the application.

Image Service: It is necessary to ensure that images are saved properly.

Report Service: Ensure that PDFs are generated and saved.

TourLog Service: It is important that the tour model is properly parsed into a tour object for data persistence.

Tour Service: The same goes for parsing a tour object into a tour model object.

JsonConverter: For the import/export of tours, Objects need to be converted from and to JSON properly.

For the **Data Layer**, the Repository is tested to make sure that data is inserted into the database.

For the **Presentation Layer**, tests are carried out to ensure valid data presentation in the UI.

SearchBar: It is tested whether a proper search works, if it also searches the computed attributes and tourlogs and if the result is empty, if nothing is found.

Tour Detail: Test if popularity is calculated properly.

TourLog: Ensure that every tourlog is displayed for a tour.

Lessons Learned

Keeping UI related logic separated from the UI and data management really helped with the code management. It took quite some time at first to really understand, how it should be separated. But after implementing a base structure, the code overview and new implementations were much easier.

Binding elements was another aspect that was new and very helpful. It simplifies keeping the UI in sync.

Another valuable lesson was the use of layers. It allowed for a more structured code base, facilitated collaboration and made the code more maintainable.

The biggest lesson to take away is to really think about the base structure of a project. This guarantees a much smoother coding experience.

Time

All in all, it took about 2 weeks to complete this project with an effort of around 8 hours daily.

Github Link

https://github.com/FancyFelicious/TourPlanner

Unique Feature

Translation: Translate the UI in one click to English or German