Tourplanner - Protocol

Author: Konstantin Lampalzer

Design Choices

General

The software is split into a **server-client architecture**, where the backend is responsible for business logic and storage, contrary to the frontend which only does the presentation to the user. The communication between server and client is done using a **REST-API**.

Frontend

The fronted is implemented in Java using JavaFX and FXML. Furthermore, the MVVM pattern is used to follow best-practices in building a JavaFX application. No data is stored on the frontend.

Backend

The backend is responsible for all business logic and data storage. Furthermore, it connects to **min.io** to store the images. **PostgreSQL** is used as the database to store all logs and tours. The **Spring Framework** is used in the backend allowing Inversion of Control and Dependency Injection.

Failures / Lessons learned

- Communication between two views can be implemented using events. This makes it possible to decouple the two views.
- Don't implement a REST API yourself. Frameworks are easier and cleaner.

Tests

repository

Both repository-classes responsible for persistence are tested using integration tests. During each test a embedded PostgreSQL database is started, simulating a real environment. These tests are really important, as then SQL statements are written by hand, which can quickly result in human errors.

search-service

As this service has a large amount of logic behind it, it's critical to make sure everything behaves as expected. The dependecies on the persistence-layer have

been broken by using **Mockito** to mock the LogRepository and TourRepository.

tour / log-service

These two services handle all logic of inserting, updating and deleting tours and logs. Therefore, testing is a requirement.

Unique Feature

The TourPlanner support Lucene query syntax for searching. This syntax is described in detail at https://www.elastic.co/guide/en/kibana/current/lucene-query.html

Data Model

Tour

- id
- name
- description
- distance
- image

Log

- id
- tourId
- startTime
- endTime
- startLocation
- endLocation
- rating
- meansOfTransport
- distance
- notes
- moneySpent

Time spent

- **28.04.:** Initial Commit
- **29.04.:** Setup and start on backend
- **28.04.:** Database setup, client for mapQuest
- **06.05.:** Tour detail panel
- **20.05.:** Client code refactoring & cleanup

- **16.04.:** Log details panel, Import and export
- **01.06.:** Switch backend to Spring Framework
- **07.06.:** Finish import & export, Add reports
- **10.06.:** Tour searching, Config properties
- **21.06.:** Unit tests, cleanup, protocol

On average one day was about 3-4 hours of programming. This results in ~50 Hours spent in total on the project.