TourPlanner Protocol
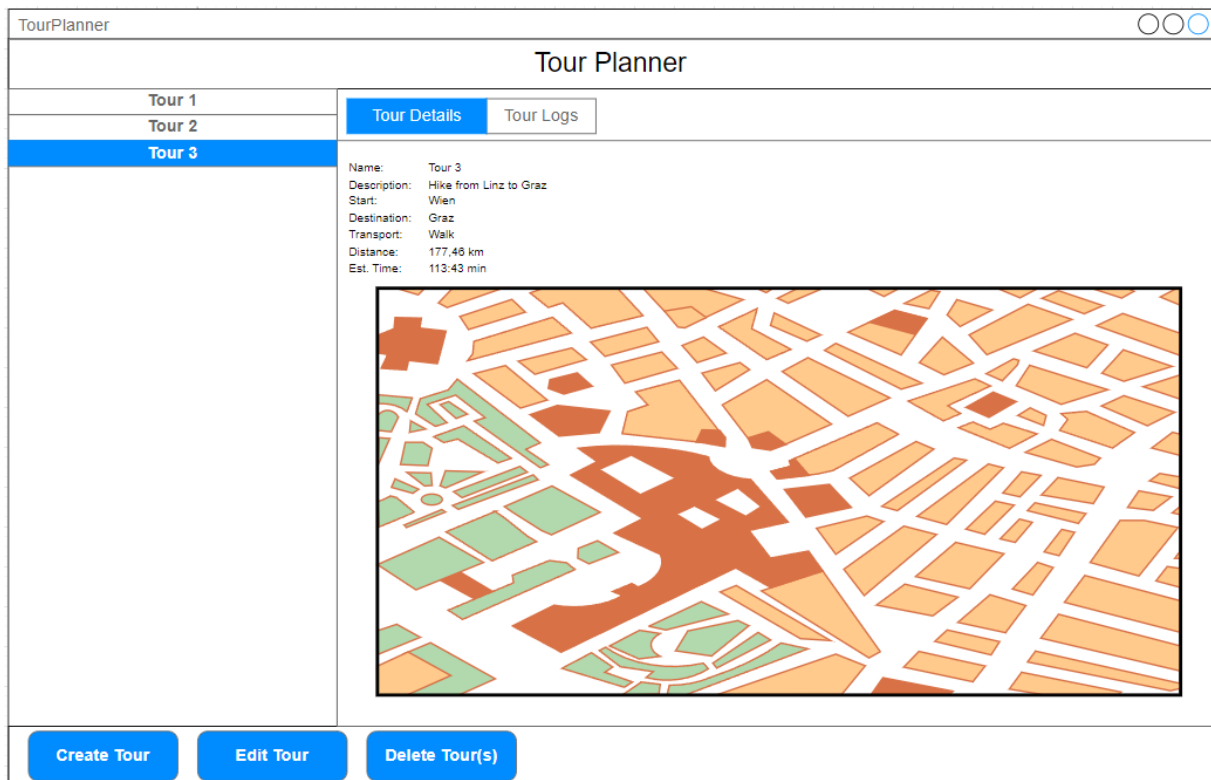
Link to git
https://github.com/Marcel19985/TourPlanner

Design and Architecture

Wireframes

On the left hand side, all available tours are listed. By selecting a tour, the details and map are shown on the right hand side of the stage.

By using the TabPane at the top, it can be switched between Tour Details and Tour Logs. After selecting Tour Logs, all Tour Logs belonging to the currently selected Tour are listed. Moreover, the Create, Edit and Delete Buttons are now connected to Tour Logs.

When selecting a Tour Log, the details of the Log are shown on the right hand side.

After clicking on Create (or Edit), a separate window should pop up where users can input data.

Layering

The functionalities of the application is logically grouped into layers. It is separated into presentation layer, business layer and data layer as follows:

- Presentation Layer

- o Controllers: are a bridge between the view (FXML) and the logical classes (as Services and Models)
- o Mediators: a ButtonSelectionMediator is used which reacts on the selected list elements (Tours and TourLogs)
- o ViewModels: are used to separate models from the view
- BusinessLayer
  - o Models: Tour and TourLog which enable the creation of instances for the added Tour(log)s
  - o Services: offer functionalities such as Import/Export and report generation and communicate with the Repositories from the DataLayer
- DataLayer: TourLogRepository and TourRepository enable data access of the database

Architecture

Reusable components: The same window is used for Create and Edit (for both TourLogs and Tours). In Edit, the non-editable properties are displayed but cannot be changed („greyed out").

The application emphasizes separation of concerns, maintainability and testability. The primary architectural pattern is the Model-ViewModel (MVVM) pattern. Moreover, several key design patterns are integrated.

MVVM

The application is divided into three main components:

1. Model: It is responsible for handling data operations such as persisting and processing data
   a. Tour
   b. TourLog
2. View: UI Layer is built with JavaFX. It provides the interface with which users interact.
3. ViewModel: Acts as a bridge between the view and the model.

Design Pattern

Singleton Pattern: The database class utilizes the singleton patter to manage a single instance of the database connection. This ensures that there is only one active connection, improving resource usage.

Mediator Pattern: The ButtonSelectionMediator<T> encapsulates the logic needed to manage the state of the UI buttons (create, edit and delete) based on the selection in a ListView. This pattern decouples the UI components, making the code more modular and easier to maintain.

Observer Pattern: Property bindings and observable collections (ObservableList used in models) are an implementation of the observer pattern. Changes to data are automatically propagated to the UI, ensuring that views always reflect the current state of the underlying data without requiring manual refreshes.

Repository Pattern: TourRepository and TourLogRepository abstract the data access and separate it from the service classes

Dependency Injection: @Controller and @Service beans are managed by Spring and injected via @Autowired

Class Diagram

The current version of the class diagram can be found in the „Dokumente" folder.

PDF Report Generation

For generating PDF reports, we evaluated **iText7** and **OpenPDF**. We chose **iText7** due to its extensive functionality and clear documentation, despite challenges with Java modules.

JSON import/export

Jackson was chosen for JSON handling because of its robust serialization/deserialization capabilities and ease of integration with Spring Boot.

Issue: Circular Reference during JSON Export

While exporting Tour objects with associated TourLog entries to JSON, we encountered a circular reference problem (Tour → TourLog → Tour, etc.) resulting in infinite recursion.

Solution:

We resolved this by applying Jackson annotations:

- @JsonManagedReference on the List<TourLog> in the Tour class.

- @JsonBackReference on the Tour reference within TourLog class.

Logging Framework

Log4j2 was integrated to enhance the application's maintainability. This framework was selected for its asynchronous logging capability, detailed configuration options, and strong community support.
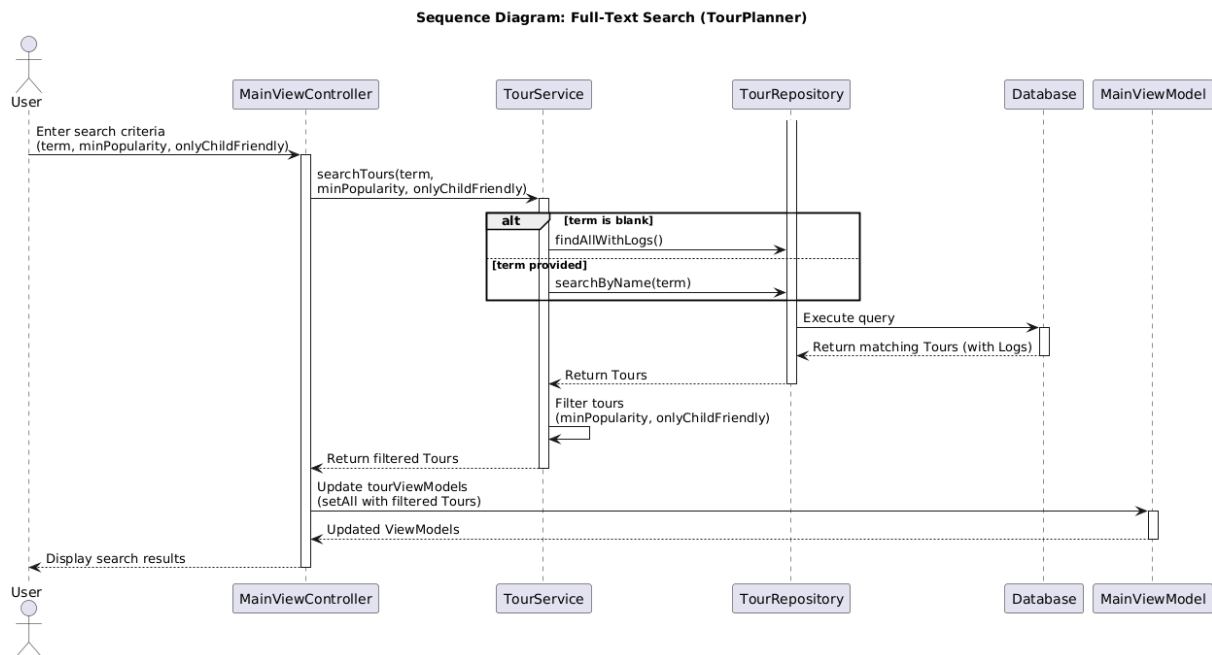
Issue: Module System Conflicts with iText7

Using Java's module system (JPMS) together with iText7 led to conflicts, as iText7 is not explicitly modularized.

Usecase Diagram
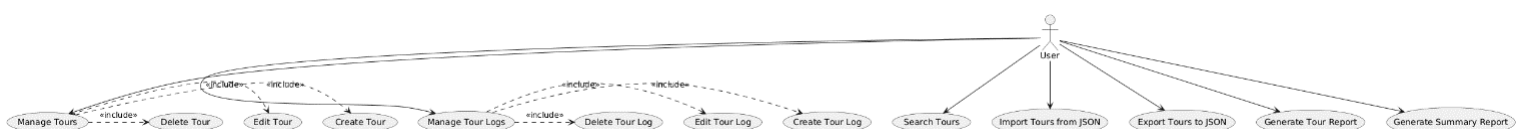
Sequence Diagram searchfunction

Sequence Diagram: Full-Text Search (TourPlanner)

Unit Tests

A total number of 47 Unit Tests were implemented across various classes to test methods in isolation. We skipped unit tests for the Controllers because their logic is so tightly coupled to JavaFX UI components that they're better verified via integration or end-to-end tests rather than isolated unit tests.

- Model Tests
  - TourLogTest
    - Verifies that the TourLog entity's constructors, getters/setters, and default state behave correctly
    - Why critical? Logic like persisting logs, calculating summaries for Tours etc. assumes the entity fields are initialized and mutable as expected
  - TourTest
    - Exercises getPopularity() and isChildFriendly() in edge and typical cases
    - Why critical? Tours are the foundation of the application, they can be filtered based on rating and child friendliness and if those computations do not work as expected, those filters will break
- Service Tests
  - ImportExportServiceTest
    - Checks JSON import/export (with and without duplicates)
    - Why critical? Backup and restore must not lose or duplicate Tours
  - OpenRouteServiceClientTest
    - Tests JSON parsing and simulates a request to OpenRouteService using Mocks
    - Why critical? External API Integration is crucial for creating Tours; routing lookup and paring is tested
  - ReportServiceTest
    - Confirms that generating tour and summary PDFs actually creates non-empty files

- Why crucial? Failures in report generation can be hard to notice as one has to compare the data from the application to the report PDF.
  - o TourLogServiceTest and TourServiceTest
    - CRUD operations are tested against mocked Spring Data repositories
    - Why crucial? Direct communication with the Repositories from the data layer; errors can corrupt the data flow
- Presentation Layer Tests
  - o InputValidatorTest
    - Ensures UI validators reject Tours/Tourlogs with missing data values
    - Why crucial? Prevents incomplete data from entering the business or data layer
  - o MainViewModelTest and TourViewModelTest
    - Test whether the ViewModels who are responsible for data shown in the UI are in a consistent state after add/delete/update
    - Why critical? Any miss synchronisation would break the two way data binding
- Integration Test
  - o DatabaseTest
    - Verifies that one can open and close a connection to PostgreSQL
    - Why crucial? Data needs to be persisted throughout the use of the application
- UI Tests
  - o Until the intermediate hand in, UI tests (TestFX) were used. However, since the implementation of Spring, various compilation errors occurred that could not be resolved

Unique Features

Flexible Map Screenshot

When creating a tour, users have the possibility to choose the region of interest (by drag and drop + zoom) of the tour map that should be saved:
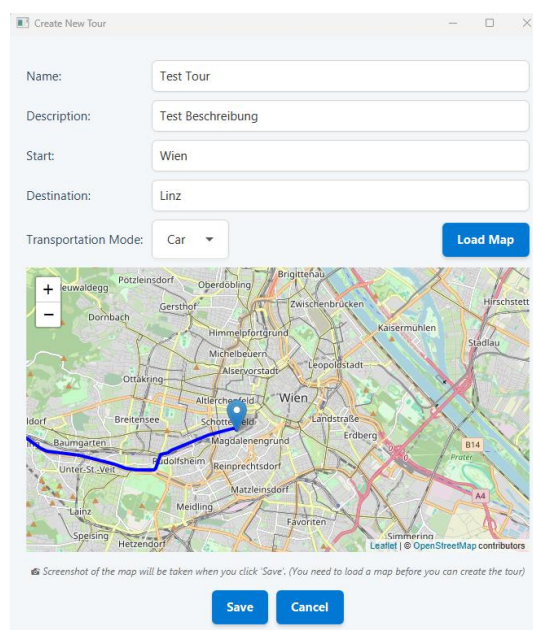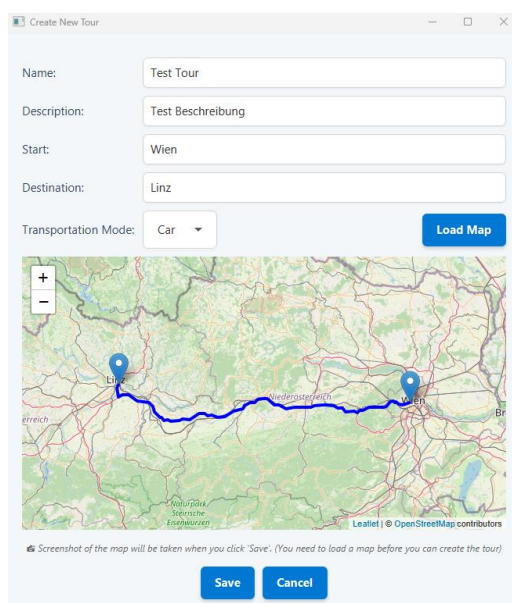
Image Upload for TourLogs

Users have the possibility to upload an optional picture documenting the TourLog:

Time Spent

Toggl was used for time tracking. It is possible to time the invested time and add descriptions for the tracked hours.

Intermediate Hand in

Until the intermediate hand in, approximately 52 hours were invested (combination of both team members).
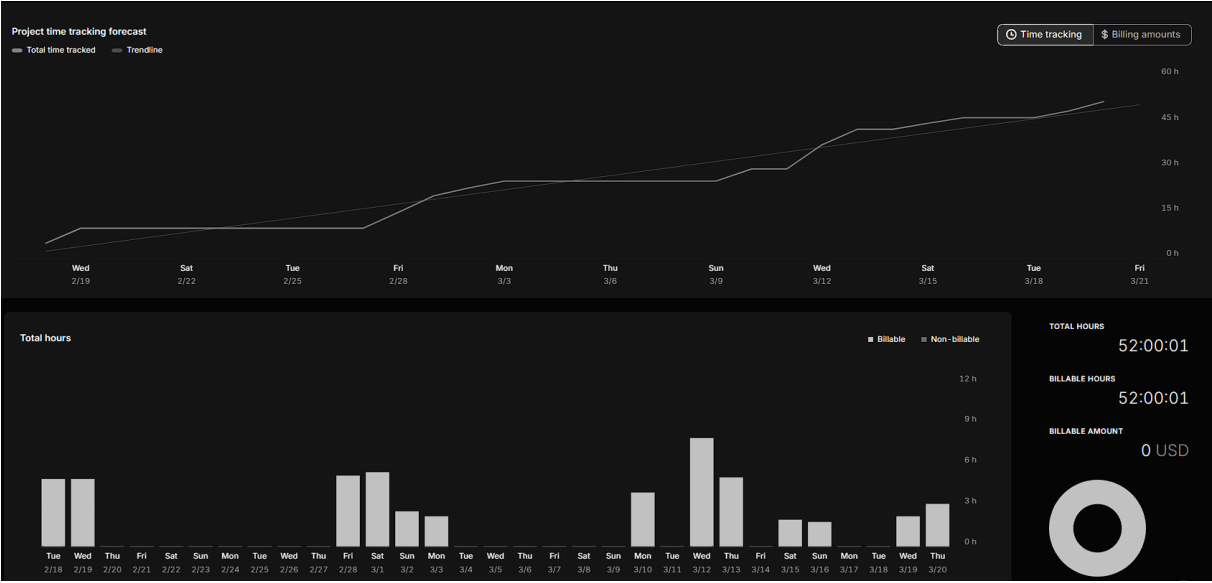
Description	Duration
Description	**Duration**
Besprechung Marcel Dominik	02:15:00
create/edit/delete	05:00:00
Debugging Anzeige TourLogDetails	01:15:00
Debugging: TourDetails nach edit aktualisiert, Kommentare hinzugefügt	01:30:00
Doku, Kommentare, Code Struktur	03:15:00
Edit and new in one file	02:00:00
Edit und Delete bei Tour Logs hinzugefügt, Mediator für Tour Logs angepasst	02:30:00
Grundstruktur überarbeiten, Klassenaufteilung	05:15:00
Keyboard Shortcuts	01:45:00
OpenrouteService API verknüpft	05:30:00
TabPane implementiert, Backend angepasst	06:00:00
TourLogs	06:00:00
Tourplanner	05:00:00
Unit Tests, UI Tests	02:15:00
validator for tourlogs	01:00:00
Wireframes	02:15:00
Summe	**52:45:00**

Complete time

Dominik
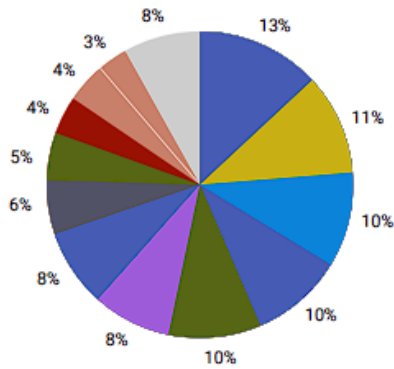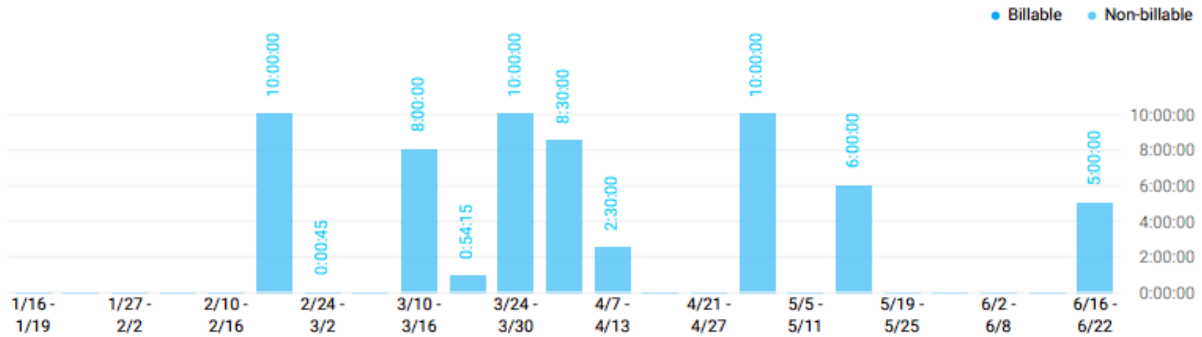
TIME ENTRY	DURATION
Leaflet added	8:00:00
report generation	6:30:00
TourLogs	6:00:15
Logging added	6:00:00
database	6:00:00
Tourplanner	5:00:00
create/edit/delete	5:00:00
Json export/import	3:30:00
Added unit test	3:00:00
Fixed a bug, where time was always 00:00 on edit	2:30:00
Service class	2:30:00
Edit and new in one file	2:00:00
Other time entries	4:54:45

Marcel

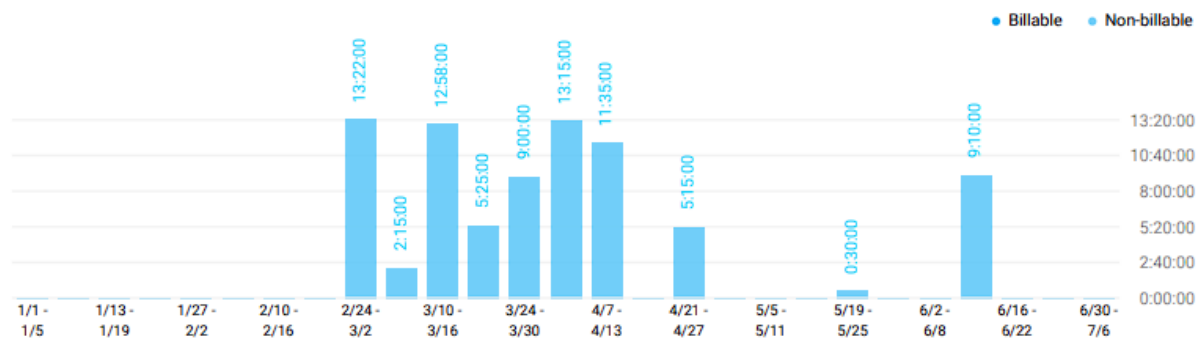| TIME ENTRY	DURATION
Datenbankanbinding, Hibernate, Boot Spring	10:45:00
Layer ueberarbeitet, kommentare hinzugefuegt, ConfigLoader ausgelagert zum Abfragen der Daten im config file	6:15:00
TabPane implementiert, Backend angepasst	6:01:00
Data Binding bei Tours anpassen, MVVM pattern mehr berücksichtigen	6:00:00
OpenrouteService API verknüpft	5:30:00
Suche für Tour und Tour Logs	5:15:00
Grundstruktur überarbeiten, Klassenaufteilung	5:15:00
Aufräumen, Kommentare	3:15:00
Doku, Kommentare, Code Struktur	3:10:00
OpenRouteServiceClient.getRouteDetails refactored: statt double mit Koordinaten wird jetzt Tour Objekt zwischen Methoden uebergeben	3:10:00
Data Binding bei Tourlogs anpassen, MVVM pattern mehr berücksichtigen	3:00:00
Edit und Delete bei Tour Logs hinzugefügt, Mediator für Tour Logs angepasst	2:37:00
Besprechung Marcel Dominik	2:15:00
Unit Tests, UI Tests	2:15:00
Wireframes	2:15:00
Bildupload für TourLogs hinzugefügt	2:10:00
Kommentare hinzugefuegt, redundanten Code in fxml entfernt, MainViewController etwas aufgeräumt	2:10:00
Refactor	2:05:00
Keyboard Shortcuts	1:50:00
Tests fuer OpenRouteServiceClient hinzugefuegt	1:40:00
Debugging: TourDetails nach edit aktualisiert, Kommentare hinzugefügt	1:37:00
Other time entries	4:15:00

A total of 143h was invested in total (both team members) for the implementation of the TourPlanner. The most time consuming implementations were spring and leaflet.

Lessons Learned

1. **Good Architecture Saves Time**

   o Using **MVVM** made the code cleaner and easier to manage.

   o Adding multiple design patterns (Singleton, Mediator, Observer) helped, but also made things more complex to understand in the beginning.

2. **JavaFX UI Can Be Tricky**

   o Keeping the UI responsive and consistent required extra effort.

   o The **Mediator Pattern** helped manage buttons and UI behavior better.

3. **APIs Are Not Always Simple**

   o Integrating **OpenRouteService** worked, but handling API limits and errors was challenging.

   o Using a **wrapper class (OpenRouteServiceClient)** made it easier to manage API calls.

4. **GitHub and Teamwork**

- o **Version control** was crucial, but merging FXML files caused conflicts.

- o **Clear commit messages** and regular **code reviews** helped avoid issues.

5. **Testing JavaFX Is Harder Than Expected**

   - o Business logic was easy to test, but testing **UI controllers** was difficult.

   - o Scene Builder sometimes needed manual tweaks to fix layout issues.

   - o We focused on testing the business logic extensively with unit tests, while relying on manual testing for the UI components. This strategic compromise provided thorough testing coverage without excessive complexity.

6. **Time Tracking Helped**

   - o Using **Toggl** showed how much time tasks actually took.

   - o Some things (especially UI work) took longer than expected.

7. **What We Can Improve**

   - o **Better UI design** and smoother error handling.

   - o **Automated UI testing** to catch problems early.