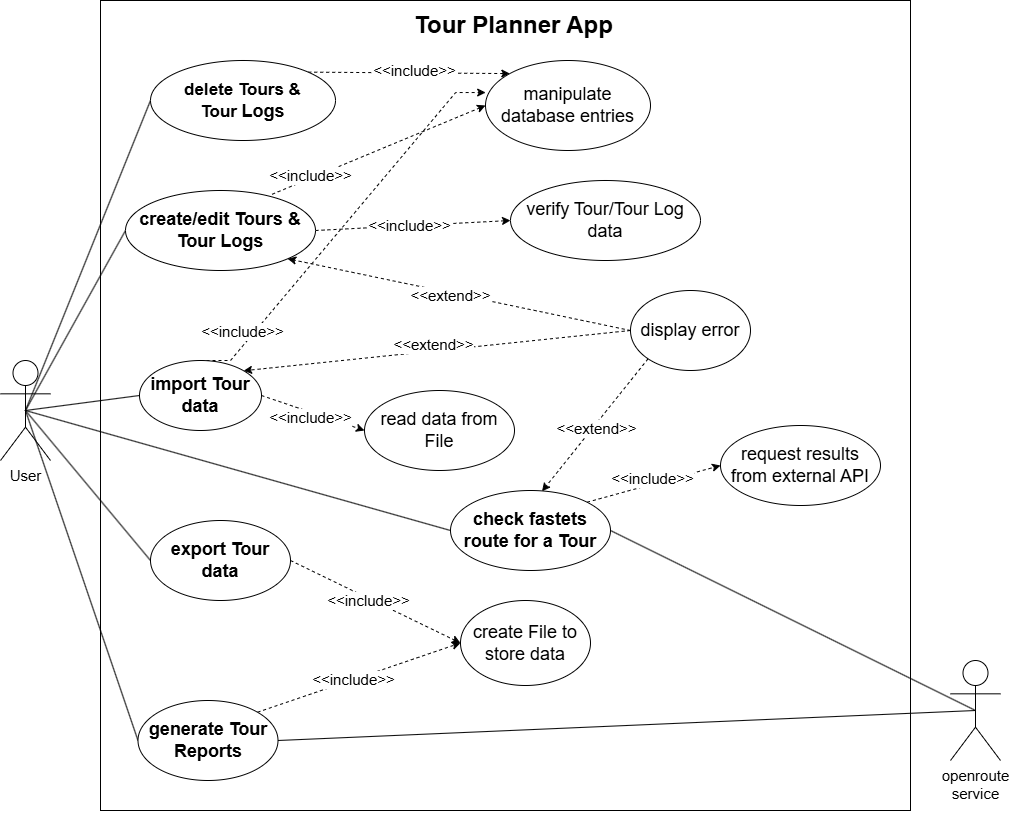
# **SWEN2 Tour Planner – Protocol**

## **App Architecture**

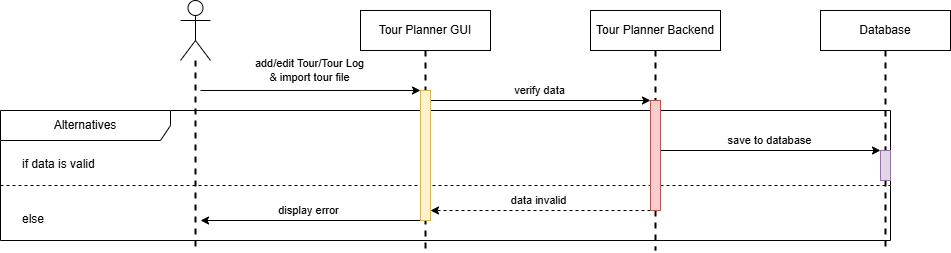
## **Use Case description (Diagrams)**

### Use Case Diagram

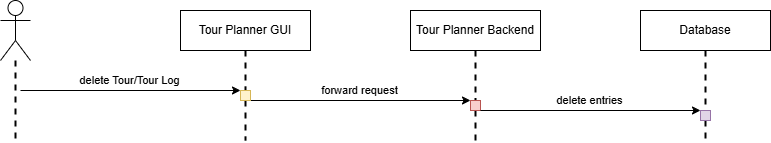


### Sequence Diagrams

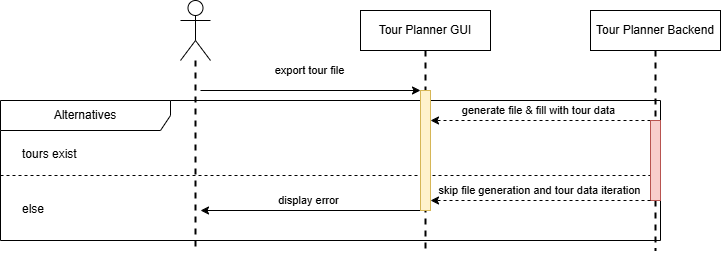
#### Add & Edit Tour data / import Tour file



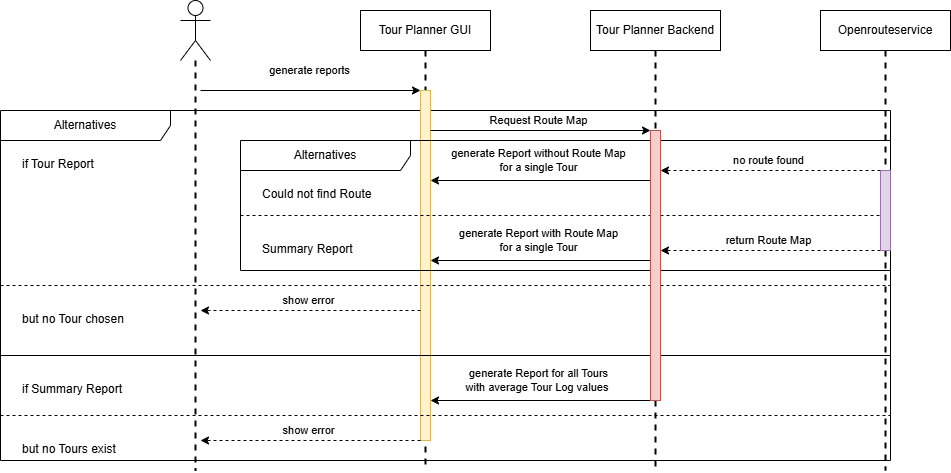
#### Delete Tour data



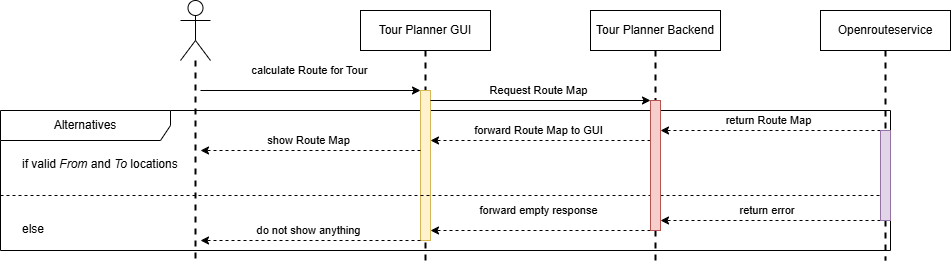
#### Export Tour File



#### Generate Reports



#### Calculate Route for Tour



## **UX-Design & Wireframes (Done)**

### Select Tour

Ein Bild, das Text, Screenshot, Software, Computersymbol enthält.

KI-generierte Inhalte können fehlerhaft sein.



When a Tour is selected, its information is shown to its right.

### Add Tour

Ein Bild, das Text, Screenshot, Software, Display enthält.

KI-generierte Inhalte können fehlerhaft sein.



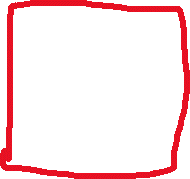
On clicking the “Add” button, the user gets write permission on Tour Fields.

“Cancel” button shows up and “Add” button gets renamed to “Confirm”.

### Edit Tour

Ein Bild, das Text, Screenshot, Software, Computersymbol enthält.

KI-generierte Inhalte können fehlerhaft sein.



On clicking the “Edit” button, user gets write permission on “Description”, “From:”, “To:”, “Transport Type:“ fields.

“Cancel” button shows up and “Edit” button gets renamed to “Apply”.

### Delete Tour



### Ein Bild, das Text, Screenshot, Software, Computersymbol enthält. KI-generierte Inhalte können fehlerhaft sein.



When pressing the “Delete” button the currently selected tour will be deleted. (For this example, we will not delete)

### Individual Tours and Tour Logs

Ein Bild, das Text, Screenshot, Software, Zahl enthält.

KI-generierte Inhalte können fehlerhaft sein.Ein Bild, das Text, Screenshot, Software, Display enthält.

KI-generierte Inhalte können fehlerhaft sein.



Each Tour has its own information and own Tour Logs.

### Add Tour Log

Ein Bild, das Text, Screenshot, Software, Computersymbol enthält.

KI-generierte Inhalte können fehlerhaft sein.



On clicking the “Add Log” button, a new Tour Log is shown. User must fill in sensible information and confirm to add the Tour Log.

“Cancel” button shows up and “Add Log” button gets renamed to “Confirm”.

### IMG 6 – Edit Tour Log

Ein Bild, das Text, Screenshot, Software, Computersymbol enthält.

KI-generierte Inhalte können fehlerhaft sein.

On clicking the “Edit Log” button, User gets write permission over the selected Tour Log.

“Cancel” button shows up and “Edit Log” button gets renamed to “Confirm”.

### Delete Tour Log

Ein Bild, das Text, Screenshot, Software, Computersymbol enthält.

KI-generierte Inhalte können fehlerhaft sein.



When pressing the “Delete Log” button the currently selected Tour Log will be deleted.

### Validations

Ein Bild, das Text, Screenshot, Display, Software enthält.

KI-generierte Inhalte können fehlerhaft sein.

Ein Bild, das Text, Screenshot, Software, Computersymbol enthält.

KI-generierte Inhalte können fehlerhaft sein.

On confirming/applying, validations are included as mentioned in the project requirements. (the examples shown are only a small part of all validations implemented)

## **Library Decisions**

### Frontend

* **JavaFX** for UI development.
* **Ikonli** for modern vector-based icons.
* **Jackson** for JSON serialization.
* **iText PDF** for PDF generation.
* **Log4j** for logging.
* **JUnit 5** for unit testing.

### Backend

* **Spring Boot** to streamline configuration and REST API setup.
* **Spring Data JPA** with **Hibernate** for ORM/database access.
* **Jackson** for JSON handling.
* **PostgreSQL JDBC Driver** for DB connectivity.
* **HikariCP** for database connection pooling.

### Lessons learned

We had conflicts between **Spring Boot** and **Log4j** since **Spring Boot** is already logging the backend. We chose to implement **Log4j** only in the Frontend.

We learned that using libraries like **Ikonli** is a much easier way of implementing icons in an application than using images from the web.

## **Design Pattern (Observer Pattern)**

We implemented the Observer pattern through JavaFX components and data structures. The pattern is used to ensure that changes in the application's model or data are immediately applied in the UI as well.

One example is the use of an ObservableList to store tour logs, which is bound to the UI component tourLogs. When a tour log is added, edited, or deleted through the model (e.g. model.getTourLogs().remove(...) or model.editTourLog()), the UI is automatically updated without manual refresh logic.

Similarly, the use of a FilteredList<TourLog> in the onLogSearch(...) method applies dynamic filtering to the observable tour log list. The FilteredList observes changes in the original list and updates the view accordingly.

We used the Observer pattern in a similar way for Tours as well.

## **Unit Testing**

### Testing Strategy

For our unit testing, we decided to **focus entirely on the MainModel class**, as it contains most of the application logic. Testing this class allows us to verify the most important business rules without relying on UI components.

### Why MainModel?

* Core application logic
* Bridge between JavaFX UI and service layer
* Validates input
* Implements derived logic (popularity, child-friendliness)

### What we tested

* **addTour** – *Scenario*: success, missing name, duplicate name
* **addTourLog** – *Scenario*: success, missing comment
* **editTour** – *Scenario*: success, invalid input
* **editTourLog** – *Scenario*: success, failure
* **deleteTour** – Scenario: success, failure
* **deleteTourLog** – *Scenario*: success, failure
* **setTourPopularity** – *Scenario*: success
* **setTourChildFriendliness** – *Scenario*: success, no Logs
* **addTourLogPreCheck** – *Scenario*: success, no Tour selected
* **importTourJson** – *Scenario*: success, invalid content, invalid Json

### Mocking and Isolation

* We used Mockito to mock the TourService and TourLogService classes
* We used reflection to set private fields (currentTourLog)

### Coverage

* 21 Unit Tests
* Covered nearly all public methods of the MainModel class
* Covered success and failure paths

## **Unique Feature**

### Description

Our application integrates an **interactive map view** into the JavaFX UI. This allows users to visualize tour routes in detail (**zooming**, **panning**, and **live interaction**) all directly within the Application.

### How it works

* We use a WebView component in JavaFX to **embed an HTML file**.
* This HTML is dynamically generated using a **GeoJSON response** from a routing API (like OpenRouteService).
* The Java code reads a template HTML file (map\_template.html) that contains Leaflet.js map logic.
* The placeholder \_\_GEOJSON\_\_ is replaced with the real route data, creating a complete, working HTML map.

### Technologies Involved

* **JavaFX** WebView
* **Leaflet.js** for map rendering
* **GeoJSON** route data
* Custom **HTML** **template** with JavaScript injection
* **Temporary file I/O** for dynamic HTML content

### Why it is unique

As our assignment suggests, we only need to show a picture of the map. What we instead did was create an immersive and interactive map experience within our JavaFX view.

## **Tracked Time**

Plachetzky Nicolai:

Anzola Fabio:

## **Git-Link (Done)**

https://github.com/fabio-anzola/TourPlanner.git