TourPlanner Protocol

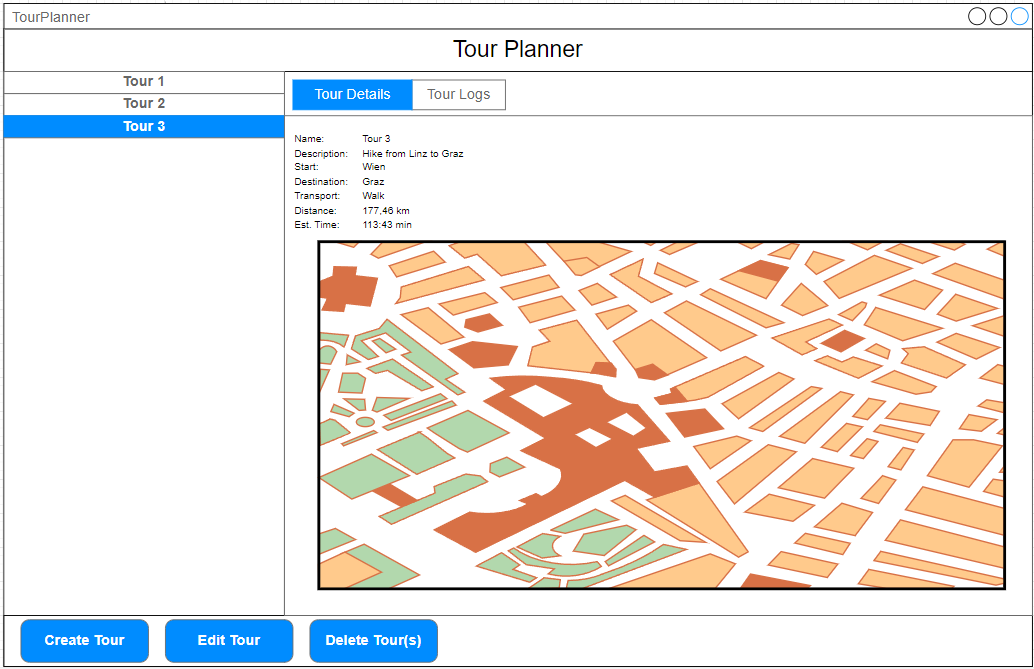
# Link to git

<https://github.com/Marcel19985/TourPlanner>

# Design and Architecture

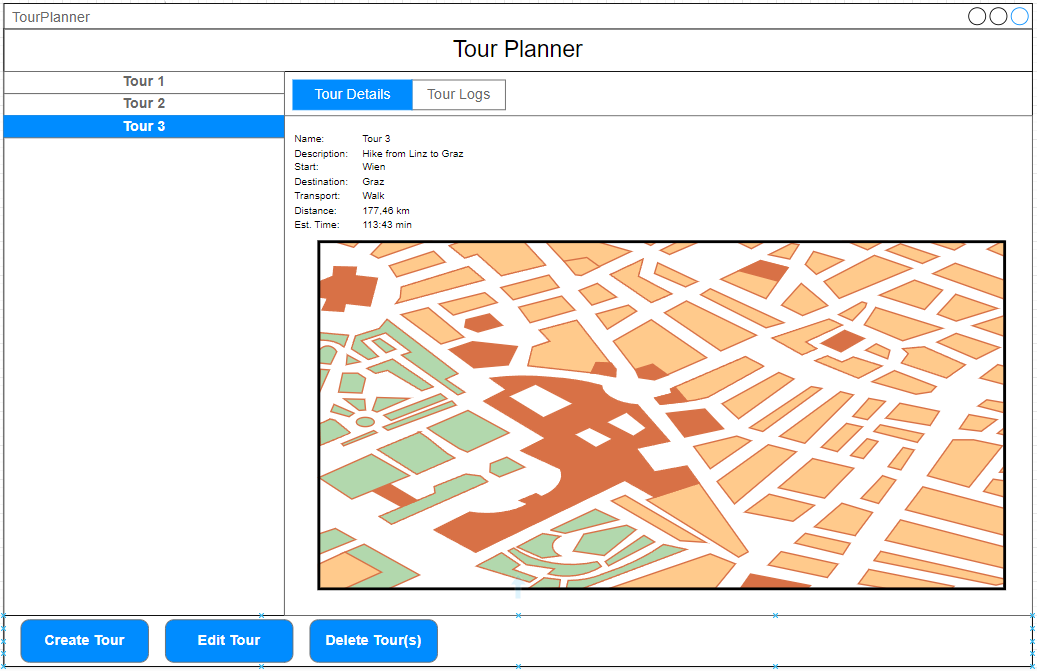
## Wireframes

On the left hand side, all available tours are listet. By selecting a tour, the details and map are shown on the right hand side of the stage. Please note that the map is currently only a placeholder.



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 oft he Log are shown on the right hand side.



## 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 seperation of concerns, maintainability and testability. The primary architectural pattern ist he Model-ViewModel (MVVM) pattern. Moreover, several key design patterns are integrated.

### MVVM

The application is divided into three main compinents:

1. Model: It is responsible for handling data operations such as persisting and processing data
   1. Tour
   2. TourLog
   3. OpenRouteServiceCleint
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 nto manage a single instance oft he database connection. This ensures that there is only one active connection, improving resource usage. Please note that the database is not used so far.

**Mediator Pattern**: The ButtonSelectionMediator<T> encapsulates the logic needed to manage the state oft he 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 oft he underlying data without requiring manual refreshes.

## Class Diagram

The current version of the class diagram can be found in the „Dokumentation“ folder.

## Time Spent

Toggl was used for time tracking. It is possible to time the invested time and add descriptiions for the tracked hours. Until the intermediate hand in, approximately 52 hours were invested (combination of both team members).

## 

## 

## Lessons Learned

* Architectural Patterns, especially MVVM
* Design Patterns as mentioned above
* Working with UI Tests
* Using git within a team project
* JavaFX
* CSS experience gained