TourPlanner Protocol

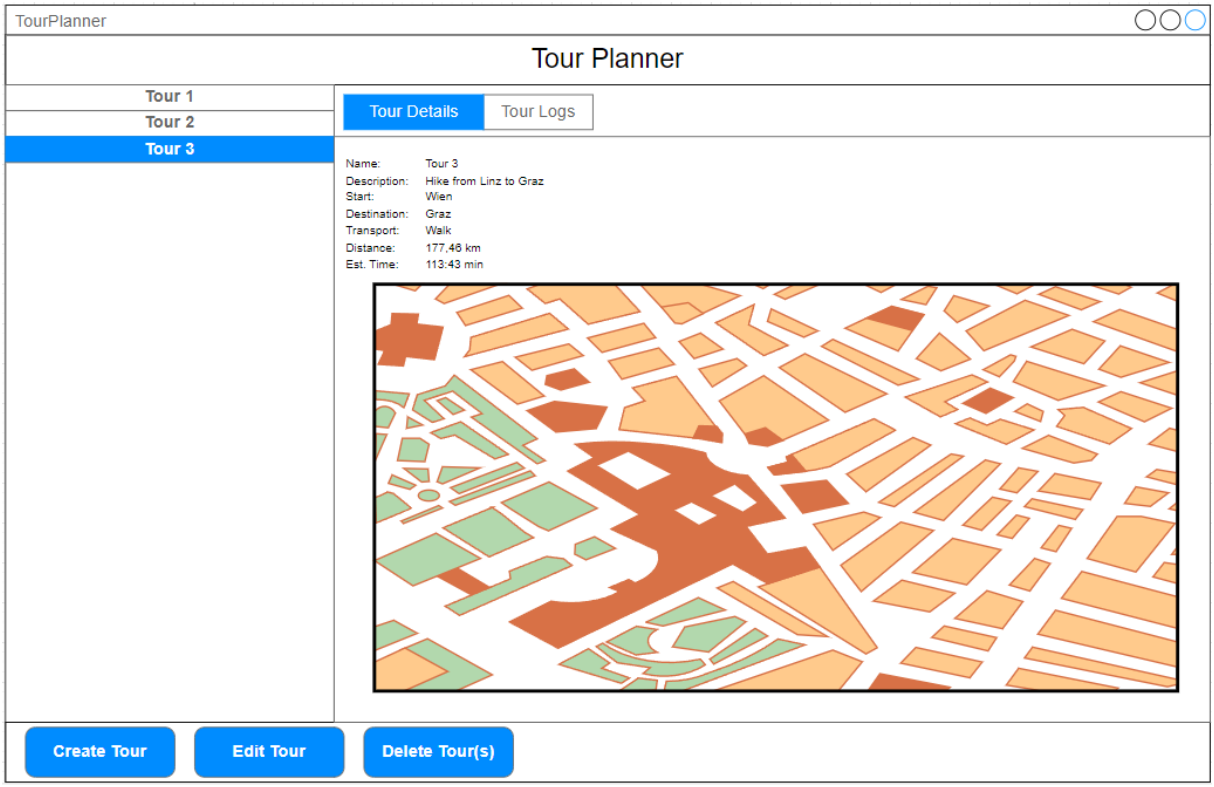
Link to git

https://github.com/Marcel19985/TourPlanner

Design and Architecture

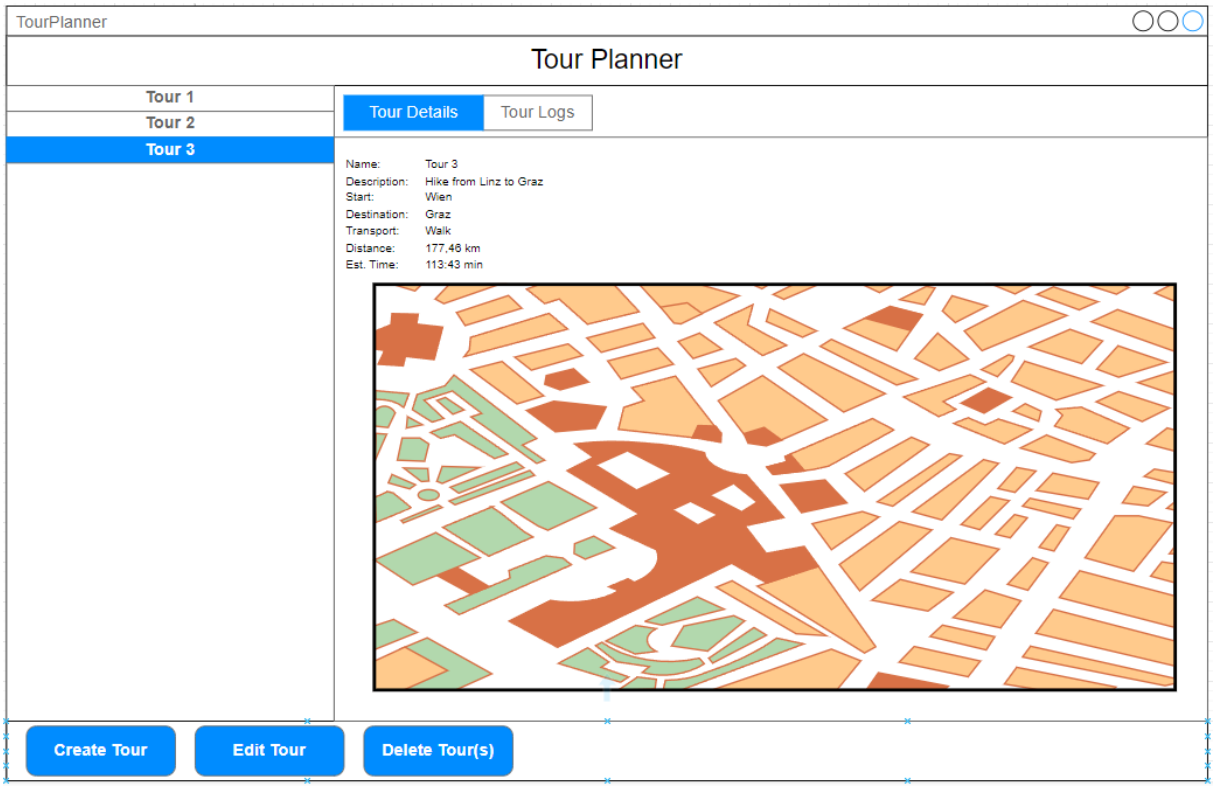
Wireframes

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
    a. Tour
    b. TourLog
    c. 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,

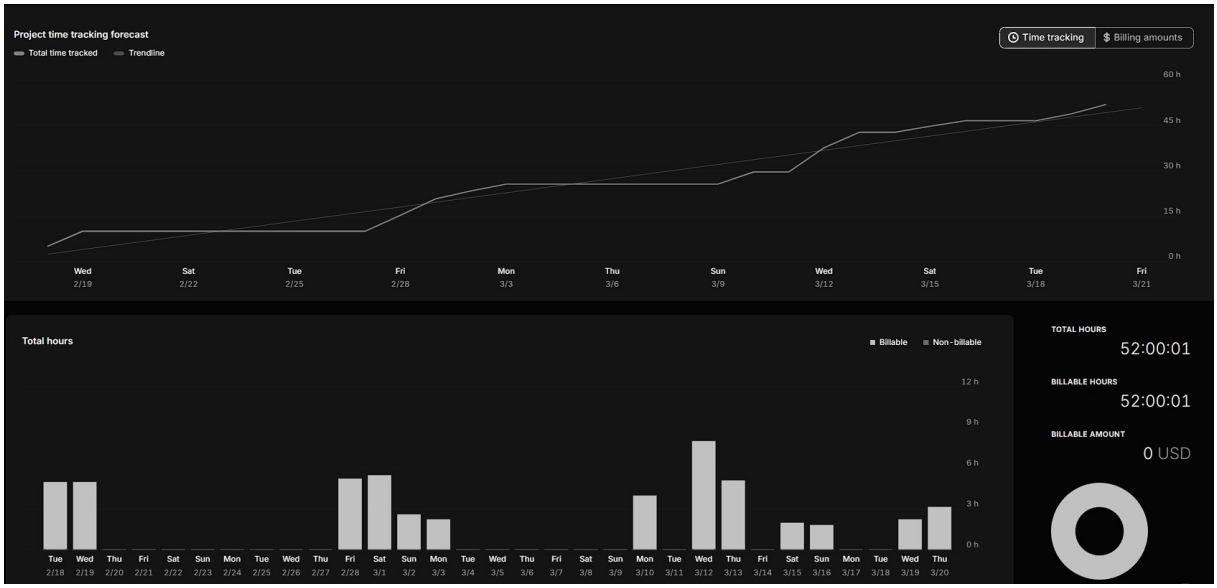
ensuring that views always reflect the current state oft he underlying data without requiring manual refreshes.

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).



	A	B
	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**

Lessons Learned

1. **Good Architecture Saves Time**

- Using **MVVM** made the code cleaner and easier to manage.
- Adding multiple design patterns (Singleton, Mediator, Observer) helped, but also made things more complex.

2. **JavaFX UI Can Be Tricky**
    - Keeping the UI responsive and consistent required extra effort.
    - The **Mediator Pattern** helped manage buttons and UI behavior better.
3. **APIs Are Not Always Simple**
    - Integrating **OpenRouteService** worked, but handling API limits and errors was challenging.
    - Using a **wrapper class (OpenRouteServiceClient)** made it easier to manage API calls.
4. **GitHub and Teamwork**
    - **Version control** was crucial, but merging FXML files caused conflicts.
    - **Clear commit messages** and regular **code reviews** helped avoid issues.
5. **Testing JavaFX Is Harder Than Expected**
    - Business logic was easy to test, but testing **UI controllers** was difficult.
    - Scene Builder sometimes needed manual tweaks to fix layout issues.
6. **Time Tracking Helped**
    - Using **Toggl** showed how much time tasks actually took.
    - Some things (especially UI work) took longer than expected.
7. **What We Can Improve**
    - **Better UI design** and smoother error handling.
    - **Database integration** to save tours and logs.
    - **Automated UI testing** to catch problems early.