Contents

Christoph Leopold, Philipp Wudernitz

1 Design and Architecture

1.1 General Information

When designing the tourplanner, we took the example in the moodle course as a guideline for our own design. With emphasis on a smooth user experience, the tourplanner offers easy oversight of all UI elements, a clear path to follow and a nice and smart combination of colors to make navigating through the project as friendly as possible.

1.2 Features

Tours

- Creating a tour by specifying parameters like name, start location, end location, …
- Deleting a tour
- Viewing the previously specified tour details in a summary window

Tour Logs

- Creating a tour log by specifying parameters like duration, distance, rating, …
- Deleting a tour log

Map

- Figure of the map of the tour with a route to follow
- It is possible to zoom in and out in the map

Report

- Generating a report for a tour including information of both the tour and the logs

Import / Export of tours

- Export the tours in a .json file for later use.
- Import the tours from an existing .json file.

Search

- The user can search for most attributes using the search bar.

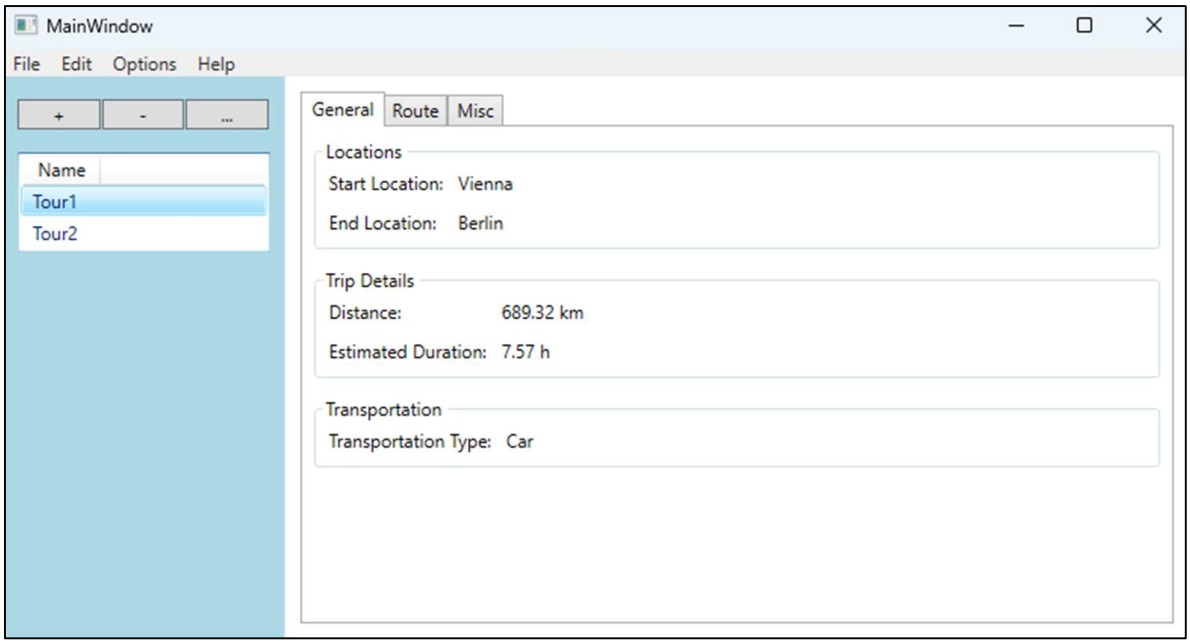

Figure 1: View details after entering a new tour

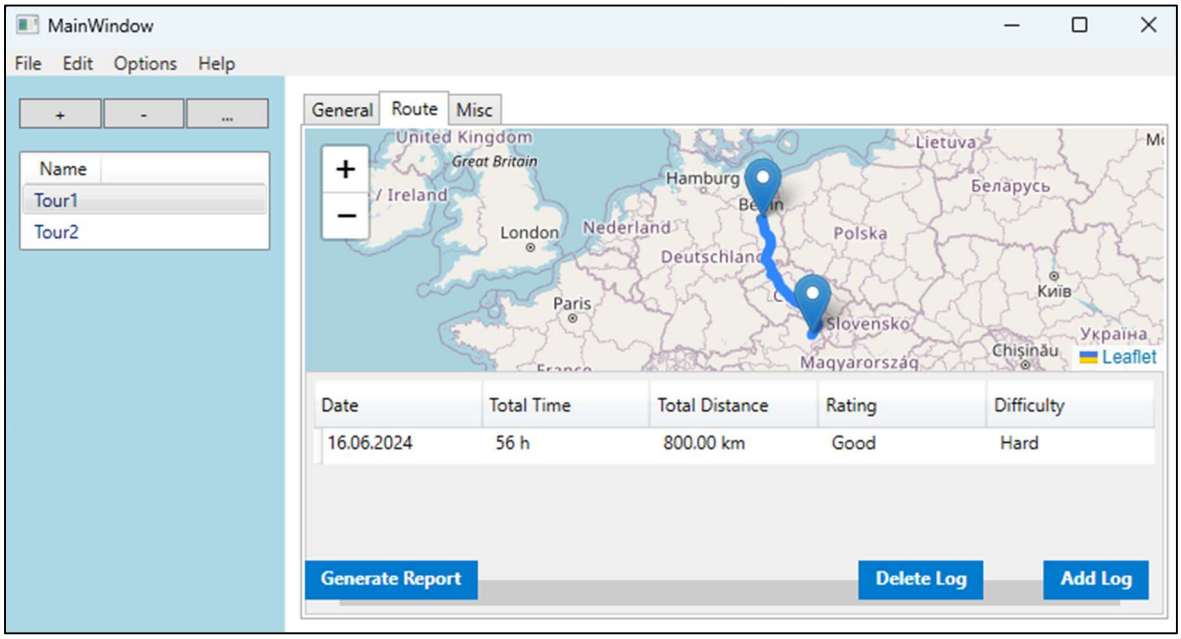

Figure 2: Map of the route and a tour log entry

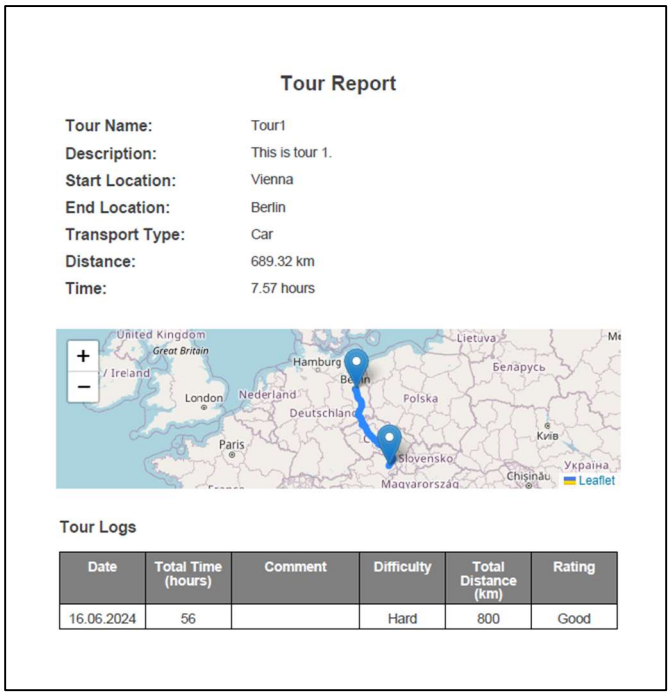

Figure 3: Tour Report

1.3 Layers

The way we structured our project is a bit untraditional, but we still implemented a layer-based architecture.

- DAL: The Data Access Layer provides a Context class which allows for the program to easily store data to the database or retrieve data from the database using the OR-Mapper.
- BL: In the Business Layer, computations of the data are conducted and processed. Here data is being prepared for displaying in the UI.
- RESTServices: This is used to establish a connection to OpenRouteServices and retrieve route information.
- MapServices: MapServices allows to program to prepare the map for displaying.
- Helpers: Important helper classes, e.g. for the report.

1.4 Library Decisions

- Microsoft.EntityFrameworkCore
  - Provides an OR-Mapper to map the model used in the program to the database structure. This allows for easy access / manipulation of data. We used this because it is by far the most popular framework for these kinds of operations.
- System.Net.Http
  - Allows to access the OpenRouteService API using an http client. Since the client is already preconfigured and just needs the API key and URL, it was a very easy choice to use this.
- iTextSharp.text.pdf
  - This library is a very popular one for creating pdf files.
- Microsoft.Web.WebView2
  - Provides a map where the route can be specified later.
- Log4net
  - An easy to use and very popular library for C# to log stuff.

1.5 Design Pattern

- Command Pattern: This pattern is intrinsic to the MVVM pattern and is used to encapsulate a request as an object. It is already demonstrated in the example above with the *RelayCommand*.
- Observer Pattern: The Observer pattern is fundamental to data binding in WPF. It allows an object to maintain a list of its dependents, called observers, and notify them of any state changes, usually by calling one of their methods. The *INotifyPropertyChanged* interface used in MVVM is a form of the Observer pattern.

2 Lessons Learned

What has been learnt from this project?

- Implementation and usage of MVVM
- Creation of a C# WPF application
- Implementation of an OR-Mapper
- Git Collaboration

During the project following things turned out to be more challenging than expected

- Dependency Injection: After numerous hours of debugging, we decided to not use it and stick to more traditional ways of coding
- Postgres Docker: We could not authorize in our project when trying to connect to a docker container running postgres, so we just used postgres (without any containerization)

3 Unit Test Design

When designing the unit tests, following aspects were taken into consideration

- Are the unit tests easy to understand for an outside developer?
- Are the unit tests independent from any preconfigured / preexisting data collection like a database?
- Does one unit test only test one ‘module’ in the code?
- Are both passing and failed cases tested in most unit tests?

4 Unique Feature

As the unique feature, we implemented the dark mode. This allows users to switch from light mode to a more soothing UI at night.

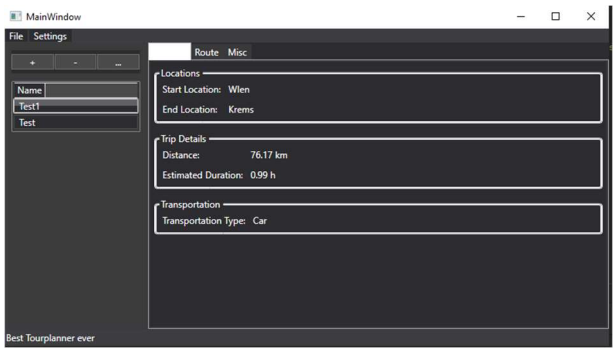

Figure 4: Dark Mode 1

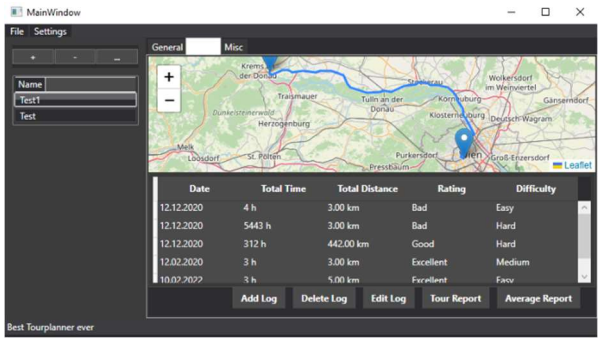

Figure 5: Dark Mode 2

5 Time spent

Leopold

- 70h
- Focus on design (including designing the whole tourplanner, MVVM concept and design patterns, postgres setup, db design and connection)

Wudernitz

- 60h
- Focus on functionality (including OpenRouteService connection / processing of the retrieved data, displaying of the map, unit tests, protocol)

6 Git

https://github.com/XayosAT/TourPlanner