Tourplanner Protocol

The project can be found at https://github.com/ezveee/SS2024-TourPlanner

1 App Architecture

1.1 Layers, Contents and Functionality

UI

- Windows Application
- Project references: None
- Responsible for user interactions
- Uses WPF
- Split into Model, View and Viewmodel
- Users can create tours and view them in a list view
- Users can create tourlogs for tours and view them in a list view
- Serializes data to JSON and sends it to the BE the HTTP Server + receives backend data from the HTTP Server

HTTP Server

- Console Application
- Project references: Business
- Interface between FE and BE
- Uses ASP.net Core
- Receives Http requests from UI and sends responses
- Controllers for Tours and TourLogs
    - CRUD operations
- HttpHelperClasses
    - Follow a similar pattern to BLs Services and DALs Repositories
    - Called by Controllers

Business Layer

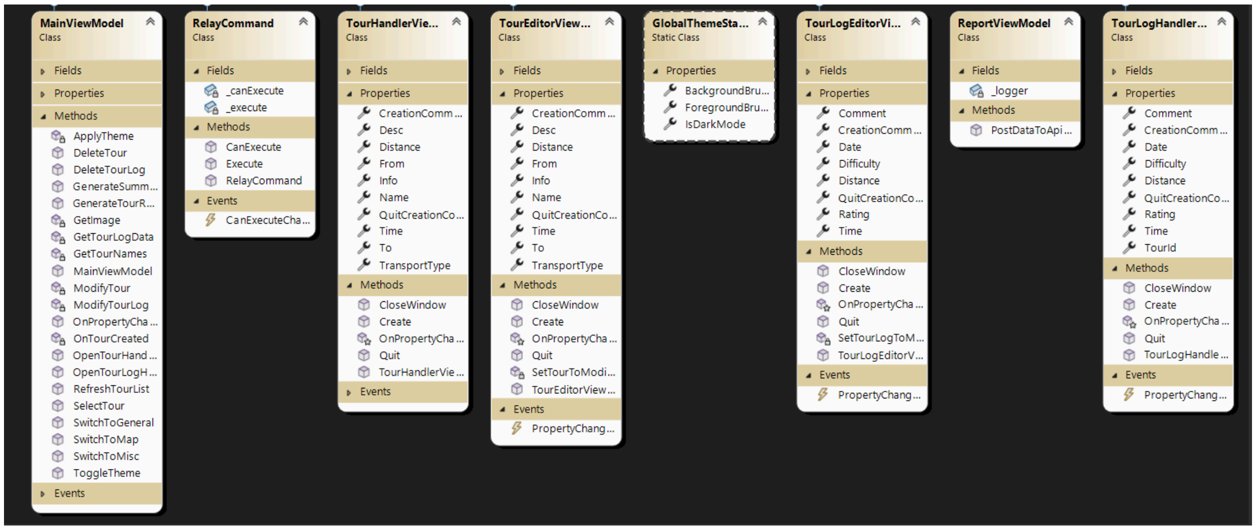
- Class Library
- Project references: DataAccess
- Contains Backend Logic
- Contains API calls to OpenStreetMap and OpenRouteService
- Contains logic for report generation
- Receives JSON data from FE via the HTTP server, serializes data into JSON and sends it to FE via server
- Passes data to DAL, communication with DAL via Interfaces

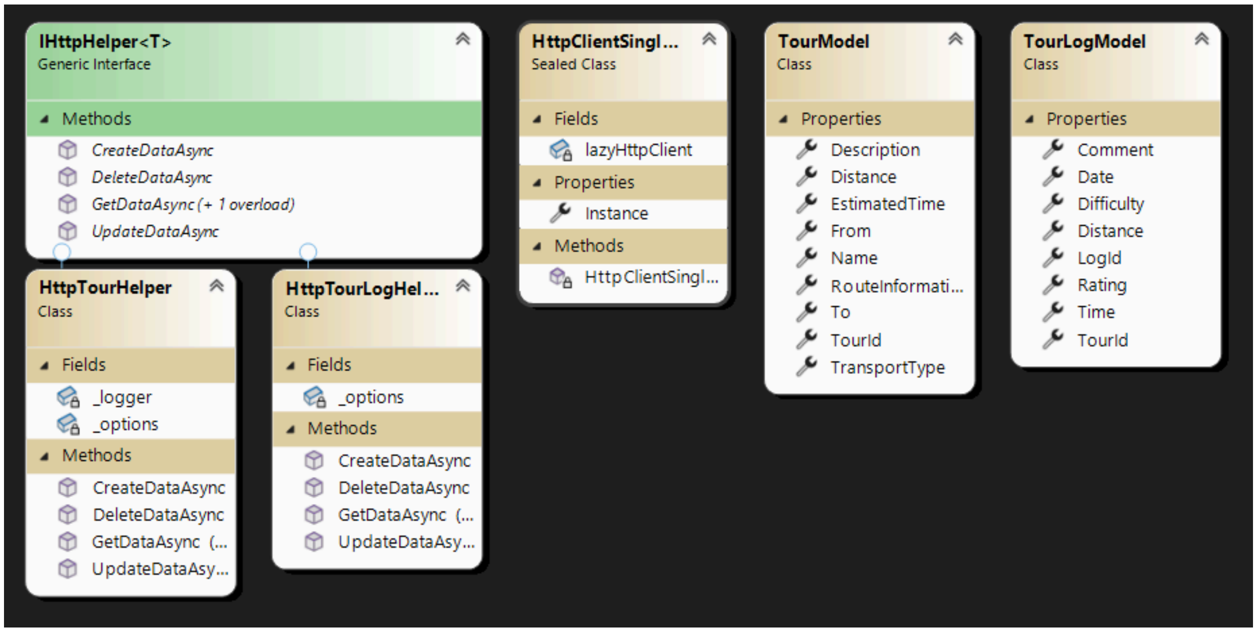
Data Access Layer

- Class Library
- Project references: None
- Leverages Entity Framework Core
- Code First (Migrations)
- Contains Database code
- Follows the Repository Pattern
- Receives data from the BL via Interfaces and retrieves data from the database to send back to BL

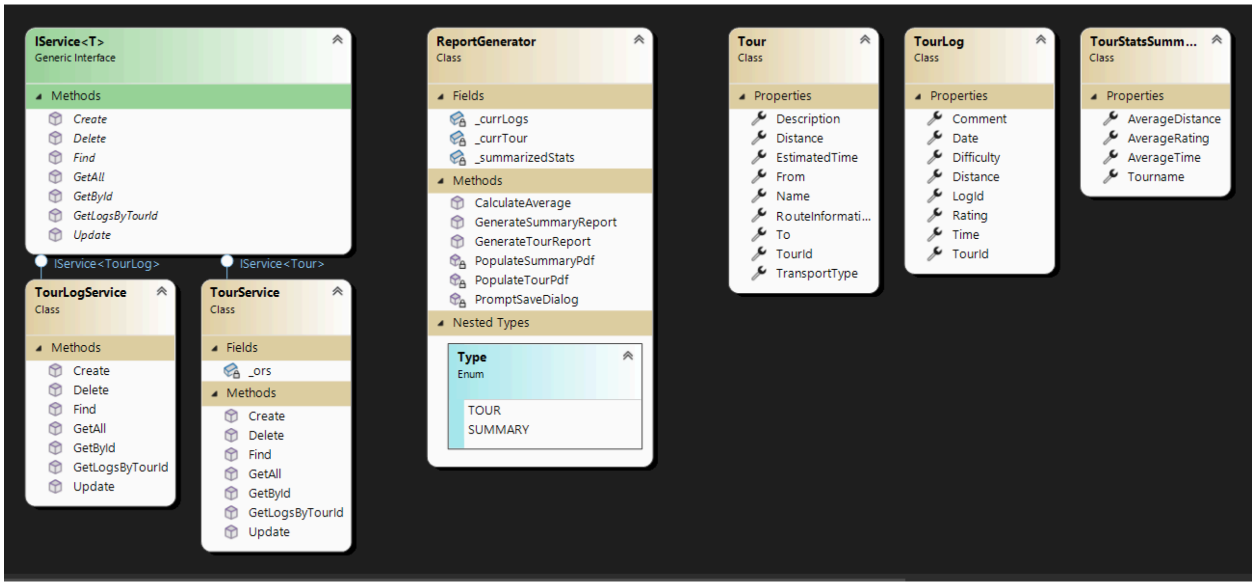
1.2 Class Diagrams

UI Layer

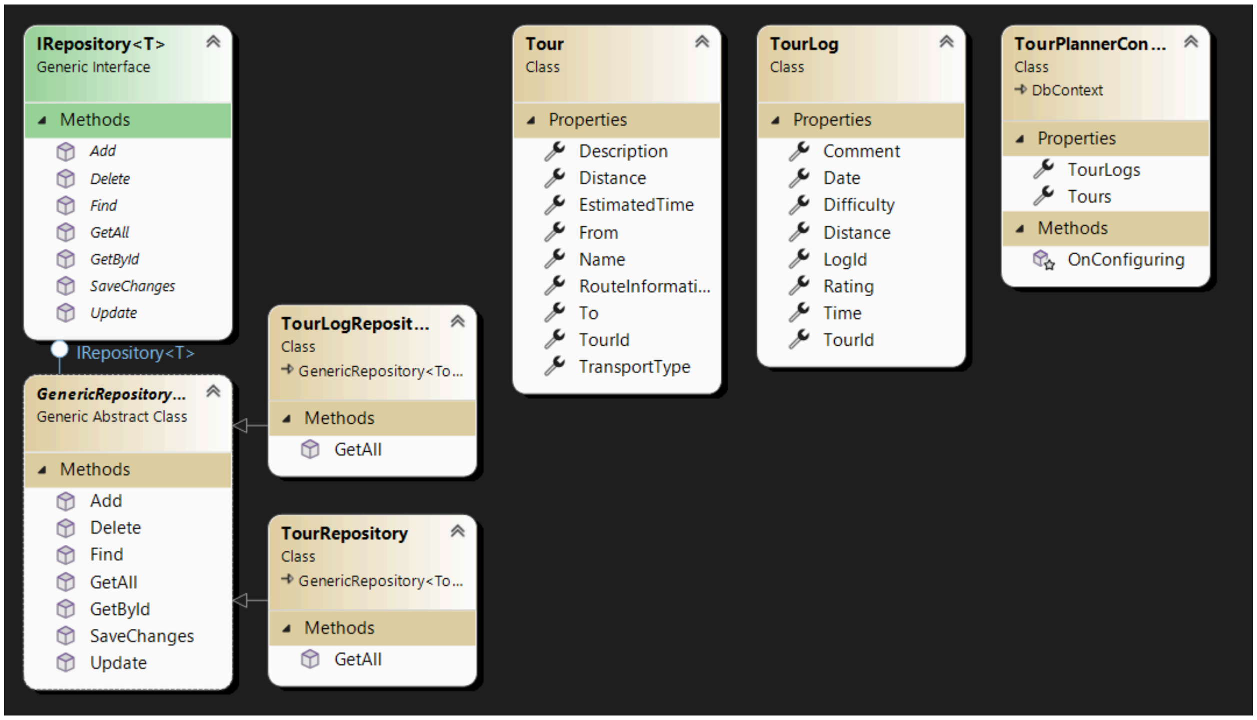




Business Layer



Data Access Layer



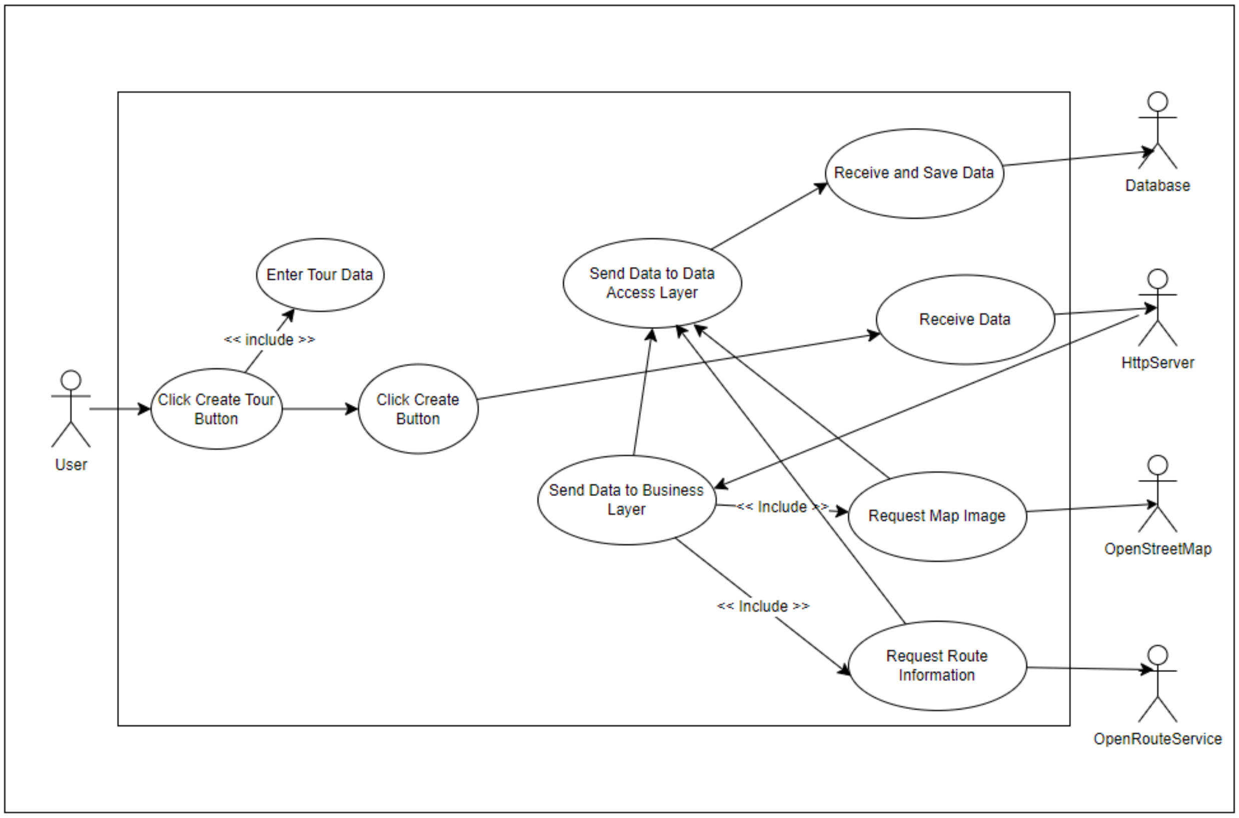
2 Use Cases

The Tourplanner is for users that want to plan a tour. Users can create their own tours or look at tours by other users. Users can also edit and delete tours. Each tour has a tourlog section which once again users can create, edit and delete.

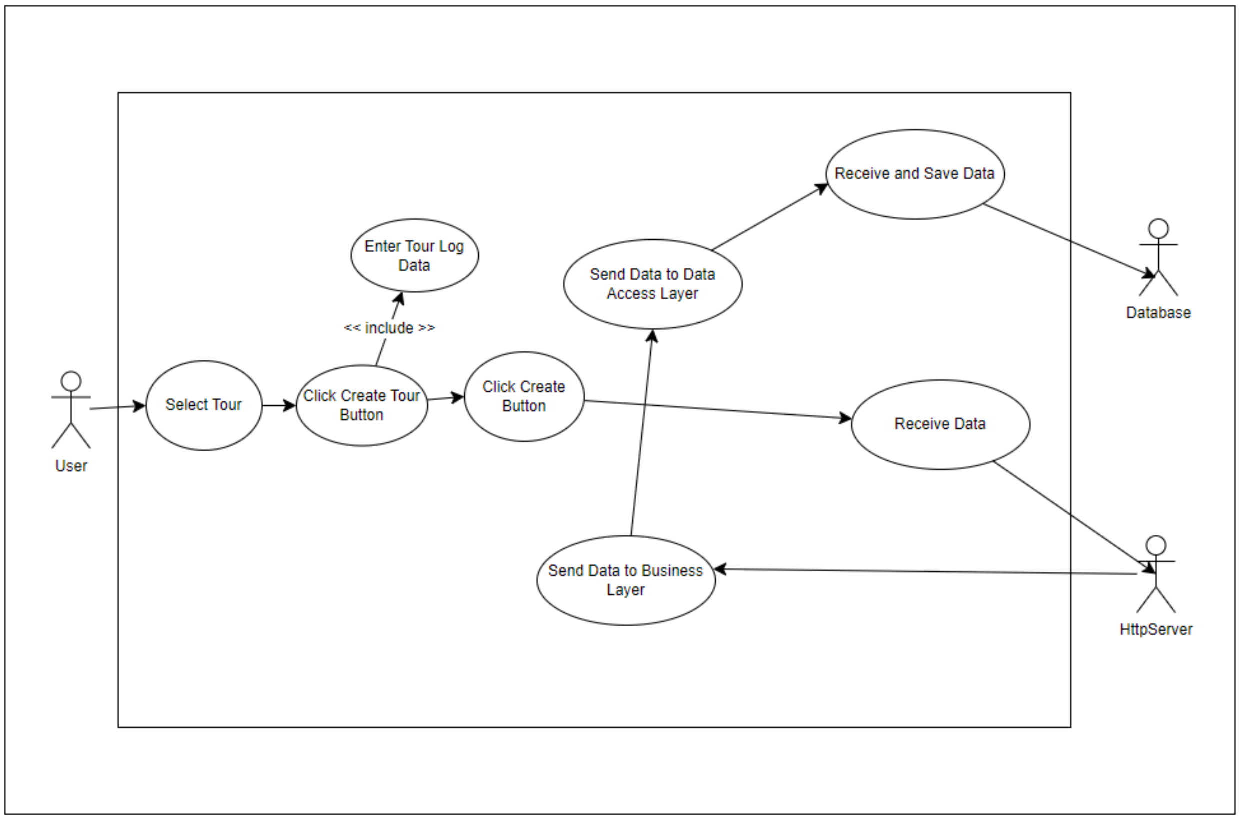
Certain tour data, such as distance, time and an image of the map, are retrieved via external APIs.
Users can also generate PDF reports of tours and their corresponding logs, as well as the tour attributed and the map image.

2.1 Use-Case Diagrams

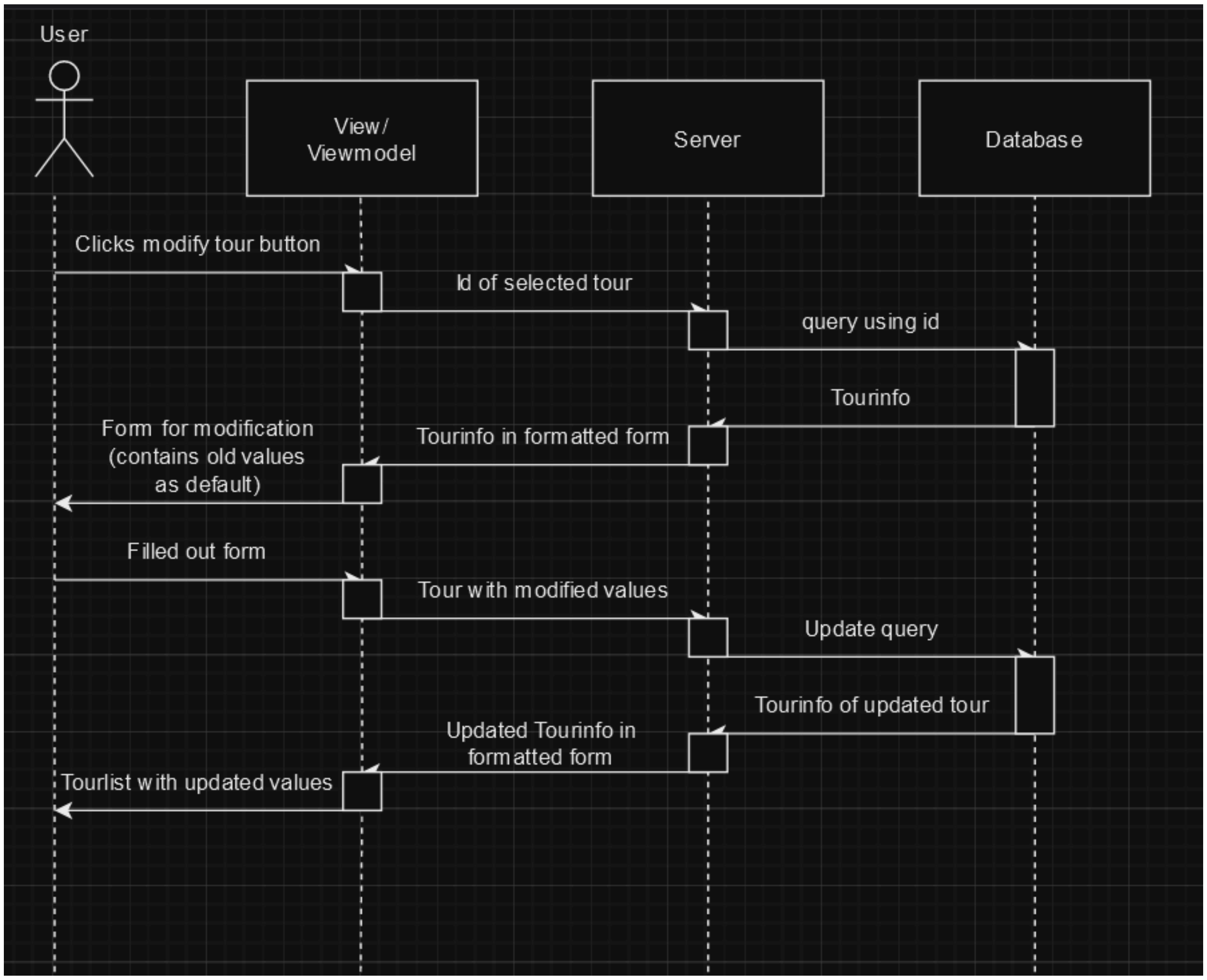
2.1.1 Use Case 1: Create a Tour



2.1.2 Use Case 2: Create a Tour Log



2.2 Sequence Diagrams


User
View / Viewmodel
Server
Database
Clicks modify tour button
Id of selected tour
query using id
Tourinfo
Form for modification (contains old values as default)
Tourinfo in formatted form
Filled out form
Tour with modified values
Update query
Tourinfo of updated tour
Updated Tourinfo in formatted form
Tourlist with updated values

3 Library Decisions

PDF Generation:

- iText7 -> easy to use pdf generation library/framework was also recommended by course which is why we used the nuGet package (+dependencies) for our report generation (located in Business layer)
- Windows forms -> makes a save file dialog window possible for the user to select where to save the generated report to (also located in Business layer)

Config:

- Microsoft.Extensions.Configuration -> enables configuration using a .json file, used to store the db connection string (in Data Access layer)

Logging:

- Log4Net -> also recommended by course, used in UI layer mainly to log events into an external file (found in UI layer in the logOutput folder)

BE:
- Entity Framework Core -> database ORM
- Newtonsoft -> JSON Serialization

Tests:
- NUnit -> Unit testing
- Moq -> Mocking of Services and Controllers

4 Lessons Learned

- SOLID principles
- Development of an application using layered architecture
- Usage of WPF to create a UI
- Entity Framework Core
- Dependency Injection
- ASP.net Core

5 Implemented Design Patterns

- Layered Architecture
  - Mandatory
- Repository Pattern
  - Used so BL doesn't directly access EF functions
- Guard Pattern
  - Minimization of ‘else’ usage
- Singleton Pattern (UI > HttpHelpers > HttpClientSingleton)
  - One HttpClient connection per session
- Dependency Injection
  - To fully make use of interfaces (goes hand in hand with repository pattern)
- Extension Object (Business > Extensions)
  - Made for better code structure

6 Unit Testing Decisions

Tests were created mostly for the Services (BL), Controllers (HttpServer) and Viewmodels (UI). NUnit and Moq were used for the tests.

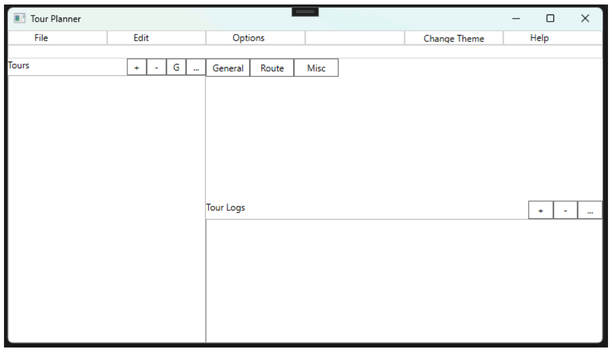
Backend:
- The tests were designed to check if each of the CRUD operations from the layer below the tested one were being called properly.

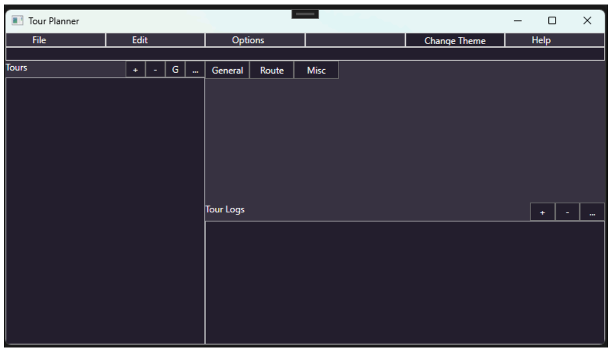
- In the Service tests, the repositories from the DAL were mocked to ensure proper calling of them.
- The controllers followed a similar pattern, with the Services being mocked to ensure they were being called upon properly.

Frontend:
- Tests designed to check if the Tour Detail Category Views switch correctly (= if “General” is the current view when called, etc.)
- Tests to check if Tour Creator and Tour Log Creator Windows open correctly
- Tests to see if changes in the UI are correctly registered.

7 Unique Feature

A toggleable Dark Mode, on the right side of the Navbar is a button that functions as a toggle to switch between Light Mode and Dark Mode.





8 Time Tracked

The individual spreadsheets containing the tracked time can be found in the Hand-In.