# TourPlanner – Protocol

Inhaltsverzeichnis

[TourPlanner – Protocol 1](#_Toc168065829)

[App Architecture 2](#_Toc168065830)

[Use-Cases 3](#_Toc168065831)

[Use-Case-Diagram 3](#_Toc168065832)

[Sequence-Diagram 3](#_Toc168065833)

[UX 4](#_Toc168065834)

[Library Decisions / Lessons Learned 5](#_Toc168065835)

[Implemented Design Pattern 5](#_Toc168065836)

[Unit-Testing Decisions 5](#_Toc168065837)

[Unique Feature 6](#_Toc168065838)

[Tracked Time 7](#_Toc168065839)

[Link to GIT 8](#_Toc168065840)

## App Architecture

The TourPlanner application is structured around a **layered architecture model**, ensuring a clear separation of concerns and promoting maintainability. The **Presentation Layer** encompasses all user interface components, primarily implemented using XAML files such as MainWindow.xaml, AddTours.xaml, and AddTourLogs.xaml. These files, along with their respective code-behind files, handle all user interactions and manage the visual elements displayed to the user.

The **Business Logic Layer** (BL) is the core of the application, containing essential services and API interactions. Key components within this layer include TourService.cs, RouteService.cs, and the GeoLocationAPI folder. This layer is responsible for managing operations, performing calculations, and facilitating communication between the Presentation and Data Access layers.

Data persistence is handled by the **Data Access Layer** (DAL), which includes files like TourContext.cs, TourRepository.cs, and ITourRepository.cs. This layer interacts directly with the database to perform CRUD operations, thereby abstracting the underlying data storage mechanisms from the rest of the application.

The **Models** layer defines the data structures used throughout the application, with Tour.cs and TourLog.cs being the primary model classes. These models represent the entities and their relationships within the TourPlanner system.

Configuration and resource management are handled by the Configuration and Resources components, which include files such as appsettings.json and log4net.config. These files store configuration settings and resources used by different parts of the application. The files in the Resources folder are needed for the embedding of the Leaflet-Map.

Finally, the application also includes a Unit Testing setup to ensure functionality and reliability. The tests validate various aspects of the application to ensure it operates correctly and efficiently.

## Use-Cases

### Use-Case-Diagram

The App supports several use cases, each enhancing the user's experience with the application. The primary use cases include adding, viewing, editing, and deleting tours and tourlogs, as well as generating reports.

For instance, the "Add Tour" use case begins when a user initiates the process through the interface provided by AddTours.xaml. This action triggers a request handled by TourPlannerVM, which then calls the TourService to perform the necessary operations. The TourService interacts with TourRepository to save the new tour to the database, and a confirmation is sent back through the TourPlannerVM to the user interface.

### Sequence-Diagram

Sequence diagrams provide a detailed view of the interactions between different components during these use cases. For the "Add Tour" sequence, the diagram illustrates the flow from the user's initial action, through the various service calls, and back to the user with a confirmation of the added tour.

## UX

The user experience design of TourPlanner is documented with wireframes that outline the application's layout and user interaction flow. These wireframes serve as blueprints for the interface design, ensuring a consistent and intuitive user experience:

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

Automatisch generierte Beschreibung

## Library Decisions / Lessons Learned

Throughout the development of TourPlanner, careful consideration was given to the choice of libraries to enhance functionality and maintainability. For instance, log4net is utilized for logging application events, as configured in log4net.config. This provides robust logging capabilities essential for monitoring and debugging. Additionally, JSON.NET is employed for JSON parsing and serialization, as configured in appsettings.json, facilitating seamless handling of JSON data within the application.

## Implemented Design Pattern

The architecture of TourPlanner is built on several design patterns to ensure a clean and efficient codebase. The MVVM (Model-View-ViewModel) pattern is extensively used, separating the user interface from the business logic and enhancing the testability and maintainability of the code. This is evident in the interaction between the TourPlannerVM ViewModel and the corresponding XAML views. Additionally, the Repository Pattern is implemented to abstract the data access logic, providing a clean API for data operations. This pattern is exemplified in TourRepository.cs, which implements the ITourRepository interface.

## Unit-Testing Decisions

## Unique Feature

CSV? idk

## Tracked Time

Puh schwierig

## Link to GIT

<https://github.com/FloberPoP/Swen2-Tour_Planner.git>