Tour Planner Protocol

1. Project Overview

The Tour Planner application is designed to allow users to create, manage, and log various types of tours. Users can view and modify tours, add logs, and generate detailed reports. The application employs C# and WPF for the graphical user interface, adhering to the MVVM pattern, and uses PostgreSQL for data storage.

Team Members:

- Yannick NWANKWO
- Donya MOHARAM

2. Design and Architecture

Application Architecture

The application follows a layered architecture consisting of:

- UI Layer: Manages user interactions and displays data (WPF, XAML).
- Business Layer (BL): Contains the core functionality and business logic.
- Data Access Layer (DAL): Manages data storage and retrieval using Entity Framework and PostgreSQL.

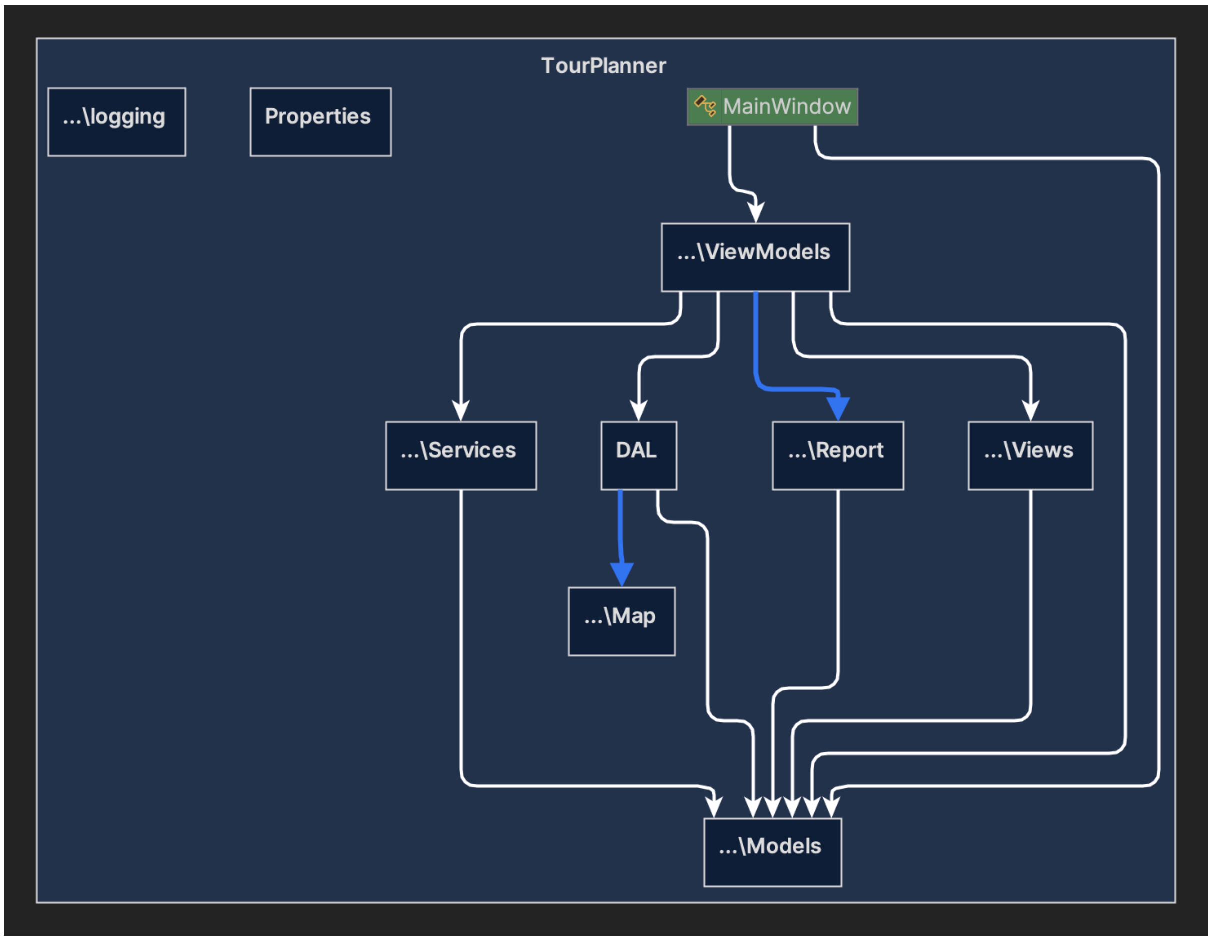
2.1. UI Layer:
Handles the graphical user interface and user interactions.
Utilizes WPF and XAML for the front-end design.
Implements data binding to connect the UI with the ViewModel.

2.2. Business Layer:
Contains the application logic and rules.
Handles the communication between the UI Layer and the DAL.
Implements business processes like CRUD operations, data validation, and computations.

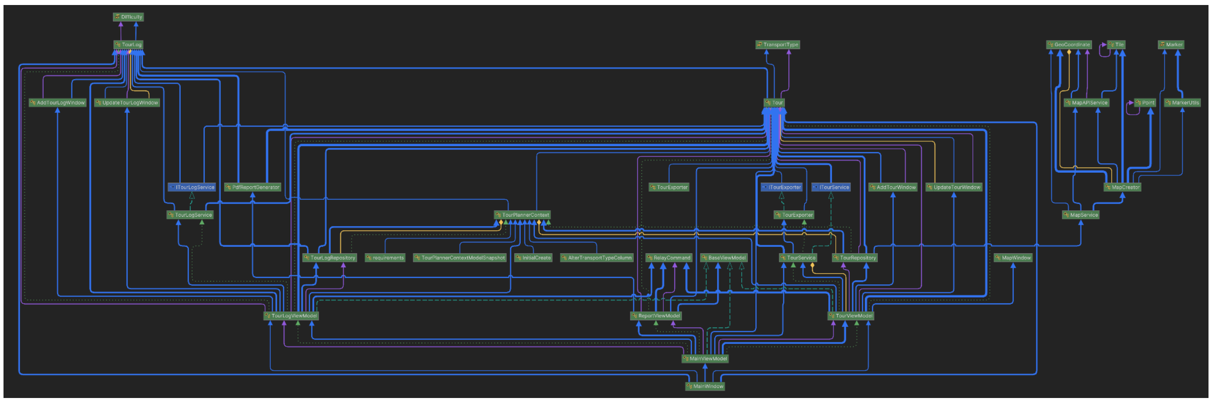
2.3. Data Access Layer:
Manages database operations using Entity Framework.
Handles data persistence and retrieval from PostgreSQL.
Implements a repository pattern for clean data access.

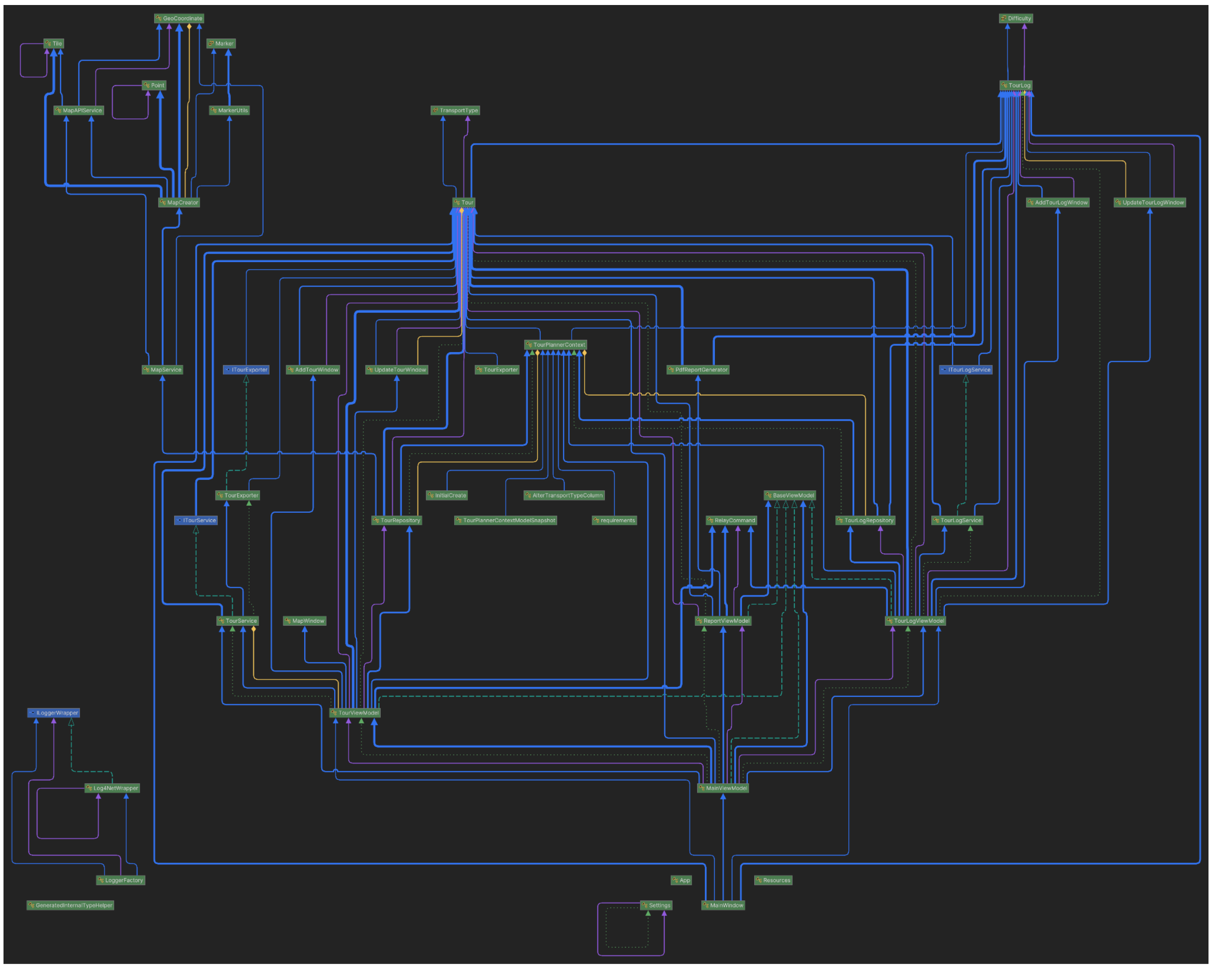
2.4. Diagrams

2.4.1. Project Structure Diagram:



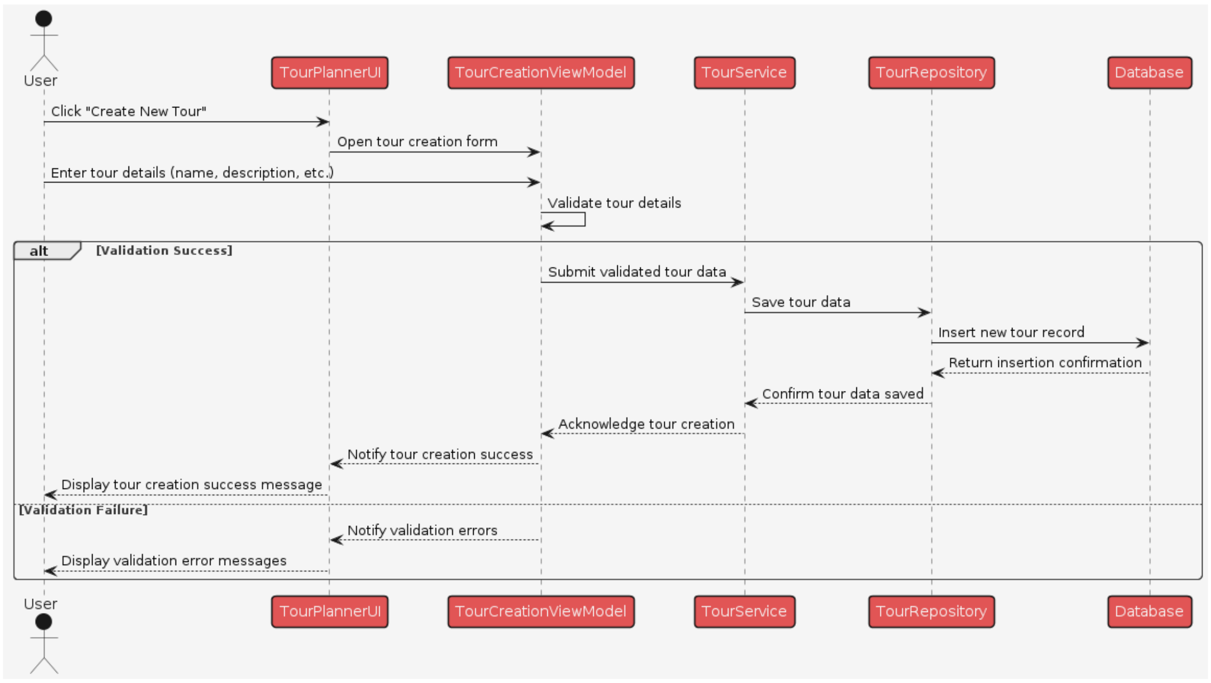
2.4.2. Class Diagram:





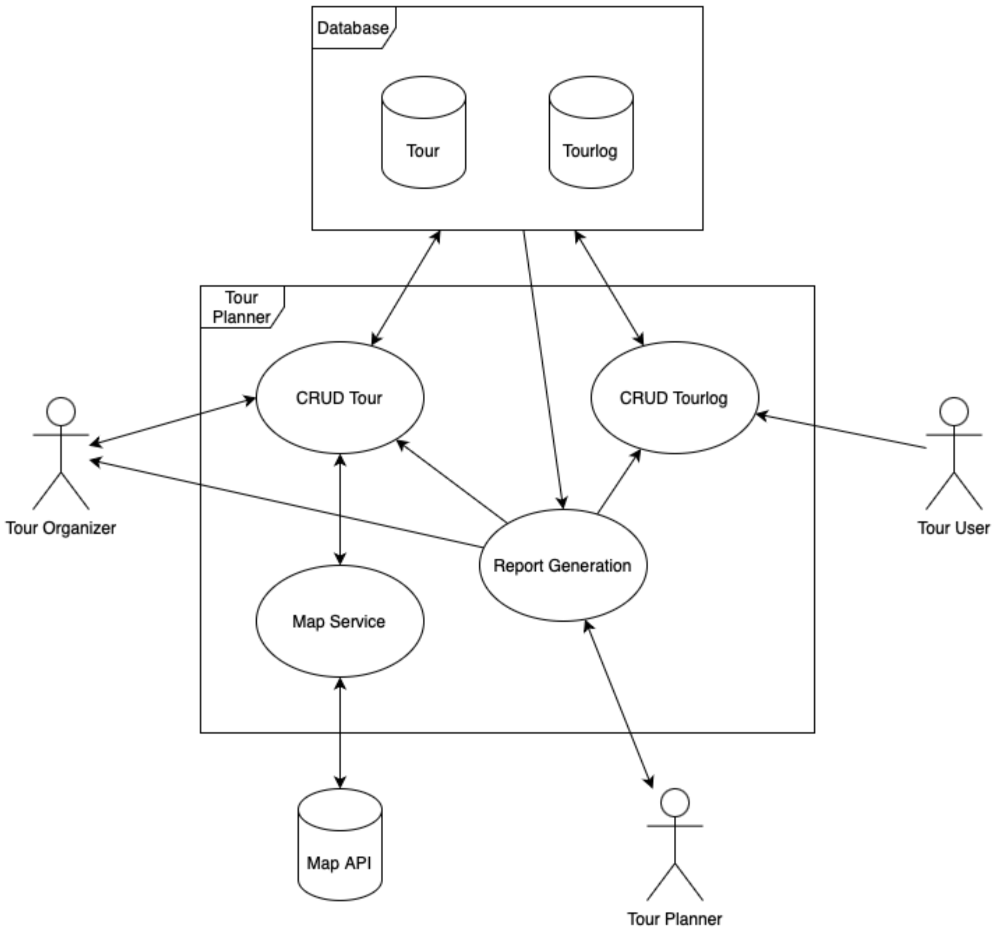
2.4.3. Sequence diagram:

[Plant UML Sequence Diagram]



2.4.4. Use-Case Diagram:

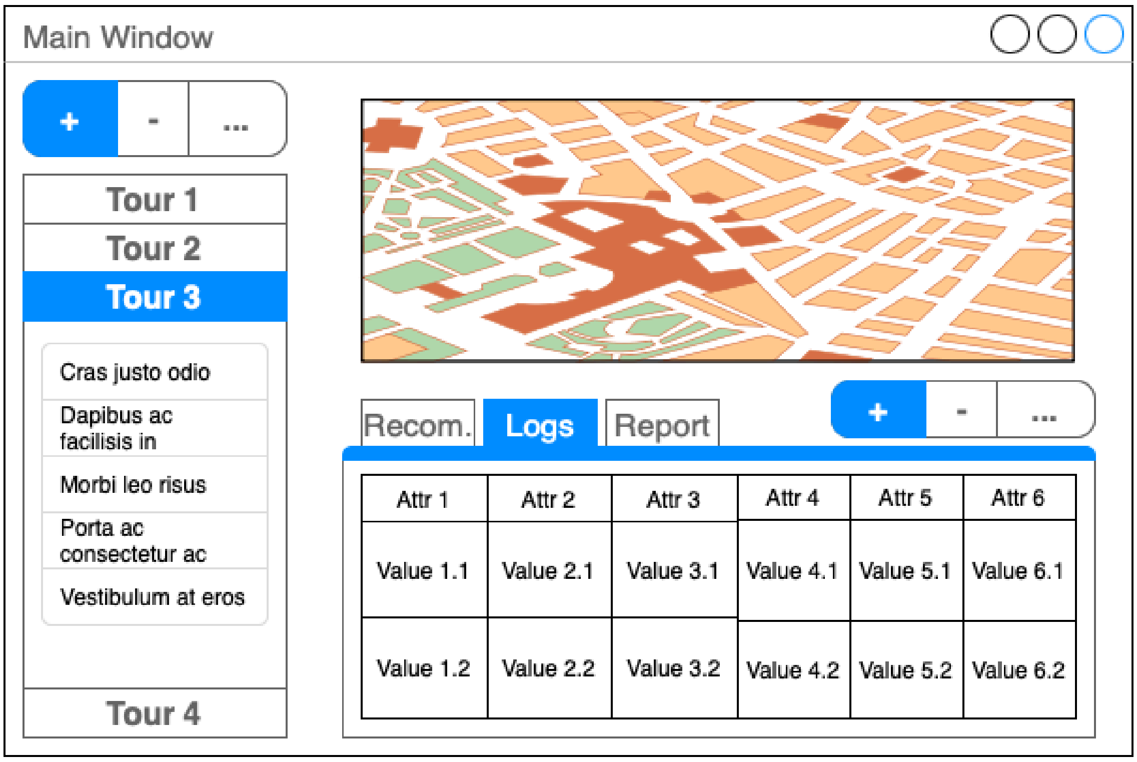
[Tour-Planner.drawio]



3. UI Flow / Wireframe

3.1. Main Window

Tour-Planner_Wireframe_Main-Window.drawio



Attr 1	Attr 2	Attr 3	Attr 4	Attr 5	Attr 6
Value 1.1	Value 2.1	Value 3.1	Value 4.1	Value 5.1	Value 6.1
Value 1.2	Value 2.2	Value 3.2	Value 4.2	Value 5.2	Value 6.2

4. Technical Decisions and Steps

4.1. Design Patterns

We implemented the following design patterns:
MVVM (Model-View-ViewModel): This pattern was used to separate the UI from the business logic.
Repository Pattern: For data access, allowing the business logic to interact with a clean and decoupled data access layer.
Factory Pattern: To manage the creation of complex objects.

4.2. MVVM Implementation

Model: Represents the tour and log data.
View: XAML files that define the UI.
ViewModel: Handles the interaction between the Model and View, processing user input and updating the UI.

4.3. Layer-Based Architecture

UI Layer: Manages user interactions and displays data (WPF, XAML).
Business Layer (BL): Contains the core functionality and business logic.
Data Access Layer (DAL): Manages data storage and retrieval using Entity Framework and PostgreSQL.

4.4. Configuration File:

Used for storing DB connection string and API key.

4.5. External Libraries:

NUnit
QuestPDF
Log4net
Entity Framework
MOQ
Newtonsoft.Json
Npgsql

5. Unit Tests

The tested code is crucial because it ensures the core functionalities of the Tour Planner application work correctly. This includes adding, updating and deleting tours and tour logs, as well as importing and exporting data.

TourLogServiceTests:

1. AddTourLog_WhenValidTourLogProvided_ShouldAddTourLogToTour
   - Purpose: Ensures that a valid TourLog is successfully added to a Tour.
2. UpdateTourLog_WhenExistingTourLogProvided_ShouldUpdateTourLogInTour
   - Purpose: Checks if an existing Tour Log is properly updated in a Tour
   - Importance: Updating Tour Logs is crucial for maintaining up-to-date information.
3. DeleteTourLog_WhenExistingTourLogProvided_ShouldRemoveTourLogFrom Tour
   - Purpose: verifies that an existing Tour Log is correctly removed from a Tour

TourServiceTests:

1. AddTour_AddsTourToCollection

- Ensures that a new Tour is added to the collection

2. UpdateTour_UpdatesTourInCollection
    - Verifies that a Tour is properly updated in a collection
3. DeleteTour_RemovesTourFromCollection
    - Checks if a Tour is correctly removed from the collection
4. ExportToursToJson_ExportToursToFile
    - Ensures that tours are correctly exported to a JSON file
5. ImportToursFromJson_ImportToursFromFile
    - Ensures that tours are correctly imported from a JSON file

6. Unique Feature

The unique feature we implemented is a Top Tours tab that acts as a recommendation. Based on the popularity of the tours, the top three tours get recommended to the user.

7. Time Tracking

Week 1-3: Requirements gathering, initial design, and setup.
Week 4-6: Implementation of core features (CRUD operations, MVVM pattern).
Week 5: Intermediate Hand-In.
Week 7-8: Database connection with entity framework.
Week 9-11: Integration of OpenRouteService API, logging, and report generation.
Week 12-14: Unit testing, bug fixing, and documentation.
Week 15-16: Final presentation preparation and protocol completion.
~100h

8. Lessons Learned

Understanding MVVM and Layer-Based Architecture
Implementing the MVVM pattern and a layer-based architecture was initially challenging but proved beneficial for maintainability and scalability. Understanding the separation of concerns between the UI, business logic, and data access layers enhanced our code organization.

Integration of External APIs
Integrating the OpenRouteservice.org and OpenStreetMap APIs required thorough documentation review and testing. This integration added significant value by automating route information and tile retrieval.

Error Handling and Logging

Incorporating log4net for logging was instrumental in monitoring application behavior and debugging. Proper error handling and logging practices allowed us to quickly identify and resolve issues, leading to a more robust application.

Unit Testing
Developing comprehensive unit tests with NUnit ensured that our application functions as intended and helped catch bugs early. Writing these tests required careful planning to cover critical parts of the code, particularly around data management and user interactions.

Report Generation
Implementing a PDF generation library for report creation presented learning opportunities regarding third-party library integration and document formatting. This feature provided users with valuable insights and summaries, enhancing the application's utility.

Challenges and Solutions
API Integration and Map generation: Initial difficulties with the Map creation were overcome by thorough testing and in class discussions and communication with peers.
Data Persistence: Ensuring efficient data storage and retrieval with PostgreSQL required testing/optimizing queries of the entity framework and understanding the ORM tool's capabilities.

9. Git Repository Link

https://github.com/Yannick-Nw/SWEN2-Tour-Planner.git