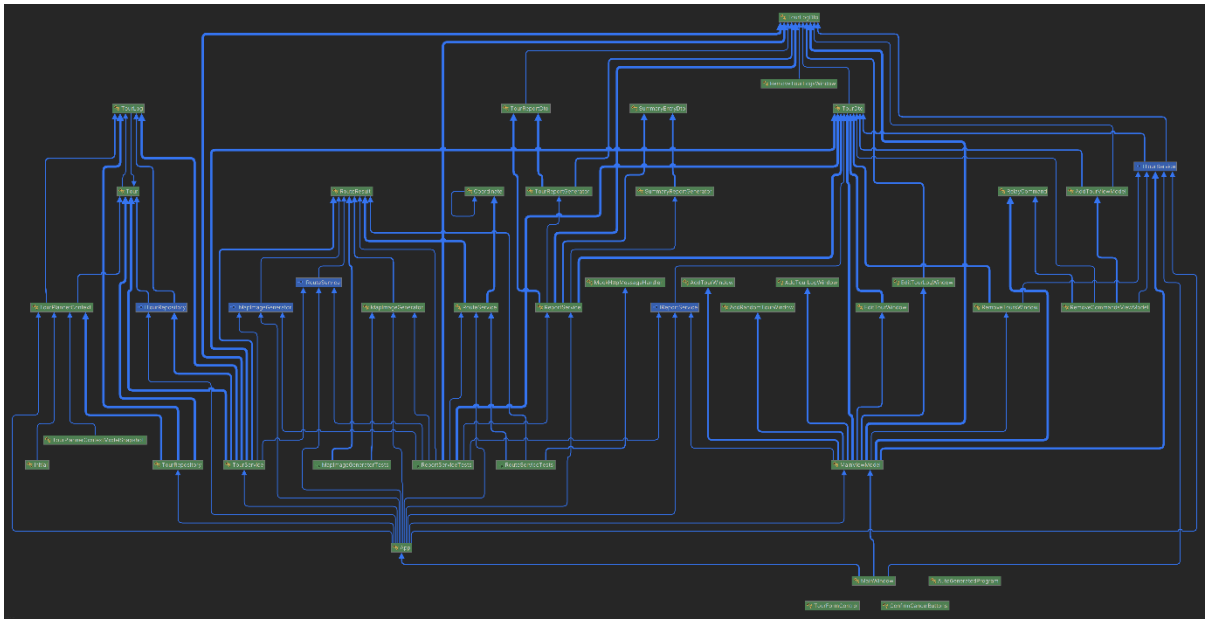
1 Class Diagram



1.1 UI (User Interface)

Purpose: Handles user interaction and display logic.

Functionality: Presents data to the user and sends user input to the Business Logic layer.

1.2 BL (Business Logic)

Purpose: Core application logic and data processing.

Functionality:

- DTOs: Defines data structures passed between layers.
- External: Handles third-party integrations (e.g., routing services).
- Reports: Generates PDF reports using QuestPDF.
- Services: Application services implementing core business rules.
- TourImage: Generates map images (likely using Puppeteer and Leaflet).

1.3 DAL (Data Access Layer)

Purpose: Manages database interaction.

Functionality:

Migrations: Contains Entity Framework migrations.

Models: Database entity definitions.

Repositories: Abstracted data access logic.

TourPlannerContext.cs: EF DbContext for managing database connection.

2 Libraries

2.1 NUnit

Used for writing and running unit tests to ensure individual components of the application behave correctly and reliably.

2.2 Puppeteer

Used to automate a headless browser, allowing you to simulate user interactions and take screenshots of the map generated by Leaflet.

2.3 log4net

Used for logging application events, errors, and debugging information to help monitor and troubleshoot the application's behavior.

2.4 QuestPDF

Used to generate PDF reports, including tour details and summaries, with precise layout and styling.

2.5 Leaflet

A JavaScript library used to generate interactive maps, allowing you to visually present the tours with routes and markers

2.6 Entity Framework

Used for database access and to generate and apply migrations, making it easier to evolve the database schema alongside the code.

2.7 Moq

Used in unit tests to mock dependencies, enabling isolation of the components being tested and simulating different scenarios.

3 Design Patterns

3.1 Single Responsibility Principle

Each class or service has one clear responsibility f.e.:

- TourService: Business logic for managing tours.
- TourRepository: Data access for Tour and TourLog.
- MapImageGenerator: Generating map images using Leaflet and Puppeteer.

3.2 Open/Closed Principle

Your services and repositories are open for extension, closed for modification. Interfaces like ITourService, ITourRepository, IMapImageGenerator allow replacing or extending implementations without modifying consumers.

4 Unit Tests

We decided to focus our testing efforts on the Report Management and Image Generation components because they provided immediate and visible feedback, making it straightforward to verify whether the output met our expectations. Since these components generate visual artifacts (such as map images or report files), it was easy to validate their correctness and formatting by simply running the unit tests and examining the resulting files.

In addition, we also tested the TourService and RouteService extensively, as they represent core parts of the application's business logic. These services handle critical operations such as creating and managing tours, calculating routes, and integrating external data (like distance and estimated time). Ensuring their correctness was essential for the overall functionality and reliability of the application.

By testing both visual outputs and core logic, we were able to verify that the application behaves correctly both in terms of user-facing results and internal processing.

5 Unique Feature

This feature lets users quickly create a tour by entering only a starting location, description, and transport type. A random European city is chosen as the destination. The system then generates a tour with a fitting name and description and saves it using the TourService.

6 Tracked time

	Tracked time
David	19 hours
Lukas	21 hours

7 Git-link

https://github.com/SwipSwup/Swen2-TourPlanner