SWEN2

Tour-Planner Protocol

BIF/4B

Alen Asanov, Michael Reichenberger

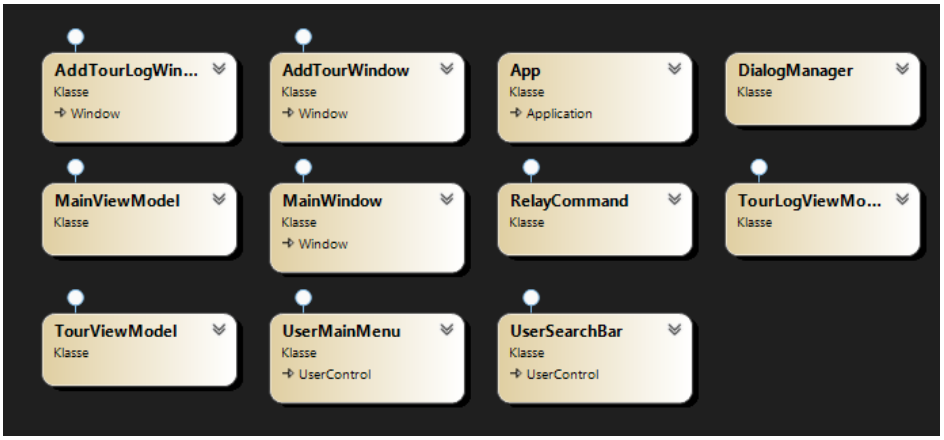
1) App Architecture:

1. TourPlanner (View + Viewmodel)

- **Resources**
  - directions.js: JavaScript file, for handling map directions or related functionality.
  - leaflet.html: HTML file that isused for displaying the map or other web content within the application.
  - Styles.xaml: XAML file for styling the application (e.g., defining themes, templates).

- **ViewModels**
  - DialogManager.cs: Manages dialogs, handling interactions and state for various dialogs in the application.
  - MainViewModel.cs: The main ViewModel, coordinating between various sub-ViewModels and the main view.
  - RelayCommand.cs: Implementation of ICommand, used for binding commands in MVVM.
  - TourLogViewModel.cs: ViewModel for handling the tour log functionality.
  - TourViewModel.cs: ViewModel for handling the tour details and interactions.

- **Views**
  - SubViews: Contains subviews or secondary windows used within the application.
  - AddTourLogWindow.xaml: XAML file for the window or user control to add tour logs.
  - AddTourWindow.xaml: XAML file for the window or user control to add new tours.
  - MainWindow.xaml: The main window of the application, acting as the primary interface for the user.

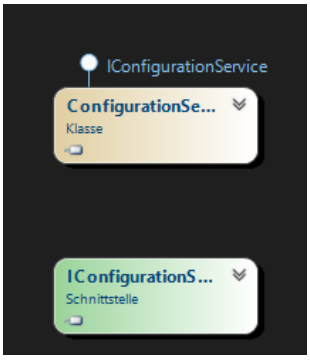


2. TourPlannerBusinessLayer (Business Layer)

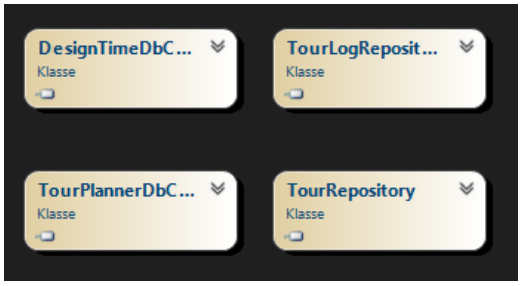
- **Managers:** Contains classes responsible for managing different aspects of the application logic.
  - FileTransferManager.cs: Handles file transfer operations.
  - ReportManager.cs: Manages report generation.
  - RouteDataManager.cs: Handles route data management.

- **Service:** Contains service classes that provide various functionalities.
  - DirectionService.cs: Manages directions-api-related operations.
  - GeocodeService.cs: Handles geocoding-api functionalities.
  - TourLogService.cs: Provides core tourLog-related services.
  - TourService.cs: Provides core tour-related services.

3. TourPlannerConfig (Configuration, Logging)
ConfigurationService.cs: Manages application configuration settings.
IConfigurationService.cs: Interface for the configuration service.

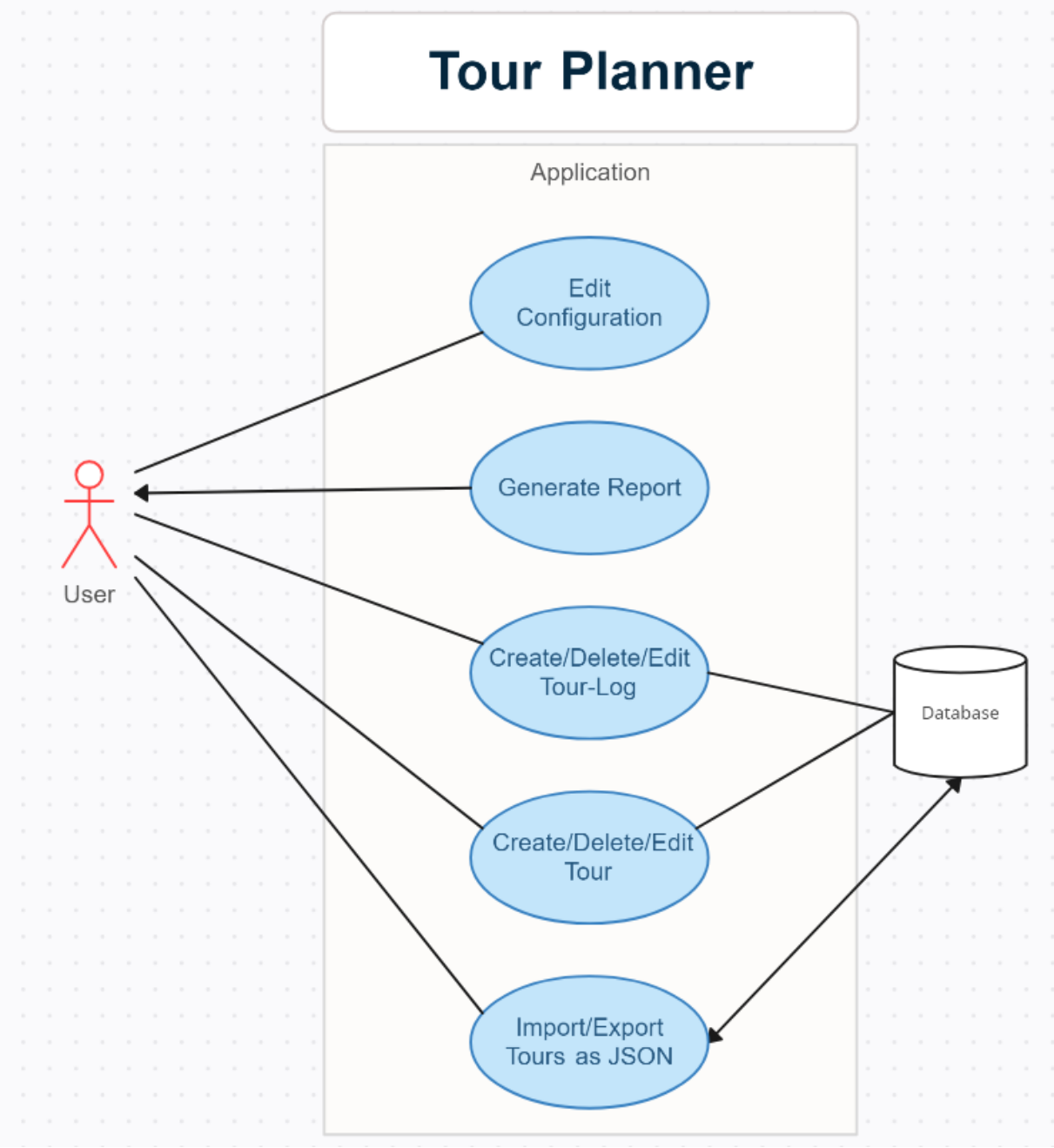


4. TourPlannerDAL (Data Access Layer)
Migrations: Contains Entity Framework migrations for database schema changes.
Repos: Contains repository classes for data access.
DesignTimeDbContextFactory.cs: Factory for creating DbContext instances at design time.
TourPlannerDbContext.cs:
DbContext class managing the database connection and entity sets.



5. TourPlannerLogging (Configuration, Logging)
ILoggerWrapper.cs: Interface or base class for logging functionality.
Log4NetWrapper.cs: Implementation of logging using log4net.
log4net.config: Configuration file for log4net logging framework.

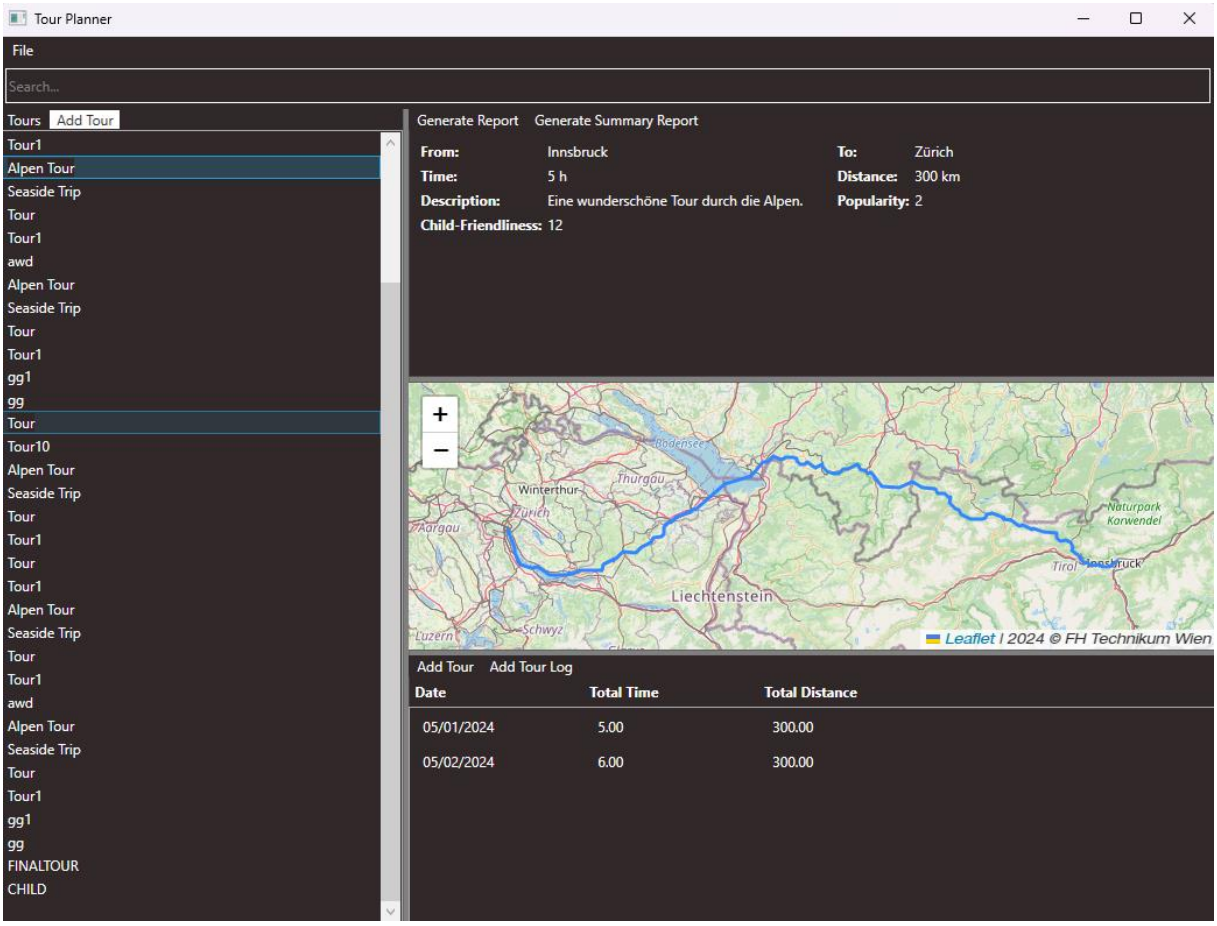
2) Use Cases:



Explanation:

User:
Can create tours: The user can add new tours to the system.
Can view tours: The user can view details of existing tours.
Can edit tours: The user can modify details of existing tours.
Can delete tours: The user can remove tours from the system.
Can add tour logs: The user can add logs or notes to specific tours.
Can view tour logs: The user can view logs associated with tours.
Can edit tour logs: The user can modify logs or notes of tours.
Can delete tour logs: The user can remove logs or notes from tours.

Can generate reports: The user can create reports based on tour data and import or export them as PDF-files.
Can import and export tours with tourlogs as JSON.
Can Change the configuration of the app (api-key, db connections string)



3) UX:

Main Components:

- **Menu Bar (Top)**

Contains File option

- **Search Bar (Below Menu Bar)**

Should allows users to search for specific tours (**logic not implemented**).

- **Tabs (Left Side)**

Tours: Displays a list of tours.
Add Tour: Provides an option to add a new tour.

- **Tours List (Left Pane)**

Displays the names of the tours (e.g., Alpen Tour, Seaside Trip, Tour, Tour1).

- **Tour Details (Right Pane)**

Displays detailed information about the selected tour.
Generate Report and Generate Summary Report buttons.
From: Starting location.
To: Destination.
Time: Duration of the tour.
Distance: Length of the tour.
Description: A brief description of the tour.
Child-Friendliness: A numeric rating.
Popularity: A numeric rating derived from the number of tourlogs.

- **Map (Center-Right)**

Shows the route of the selected tour on a map.
Includes zoom in (+) and zoom out (-) buttons.

- **Tour Log (Bottom)**

Allows users to log details of the tours.
Add Tour and Add Tour Log buttons.
Date: Date of the tour entry.
Total Time: Time taken for the tour entry.
Total Distance: Distance covered in the tour entry.
!!!MORE INFORMATION IS SHOWN WHEN CLICKING ON A TOURLOG

UI Design:
The layout is straightforward, with a dark theme and clear sections for different functionalities.

4) Libraries:

iText7:
Function: A PDF library for creating and manipulating PDF documents.
Usage: Utilized for generating tour reports, ensuring they are formatted and can be easily shared or printed.

Entity Framework Core:
Function: An Object-Relational Mapper (ORM) for .NET.
Usage: Used for data access and management, allowing us to interact with the PostgreSQL database using high-level, object-oriented code instead of raw SQL.

log4net:
Function: A logging framework for .NET.
Usage: Used for logging application events, errors, and performance metrics, which is crucial for debugging, monitoring, and maintaining the application.

WebView2:
Function: A control for embedding web content in .NET applications.
Usage: Integrated to display dynamic web content within the application, in this case interactive maps from OpenStreetMap, enhancing the user experience with rich, embedded web features.

5) Leason Learned:

- **MVVM Pattern:**
  Learned to implement the Model-View-ViewModel (MVVM) pattern for better separation of concerns, making the codebase more maintainable and testable.
- **Layered Architecture:**
  Developed skills in creating a UI Layer, Business Layer (BL), and Data Access Layer (DAL) to enhance code maintainability and scalability.
- **Design Patterns:**
  Applied design patterns like Singleton, Factory, and Command to improve code reusability, flexibility and looser coupling.
- **Reusable UI Components:**
  Designed reusable UI components, reducing duplication and improving efficiency.

- **Database Integration:**
  Gained experience with ORM tools like Entity Framework/Hibernate for managing PostgreSQL database operations.
- **API Integration:**
  Integrated external APIs (OpenRouteService.org, OpenStreetMap) to retrieve and use external data effectively.
- **Logging:**
  Implemented logging with frameworks like log4net/log4j for debugging and monitoring application performance.
- **Unit Testing:**
  Created unit tests using NUnit, improving code reliability and supporting continuous integration.
- **Configuration Management:**
  Managed application settings through external files like appsettings.json for flexibility and security.
- **Documentation and UML:**
  Documented the application using UML diagrams to effectively communicate architecture and design.
- **Report Generation:**
  Implemented report generation using appropriate libraries to compile and present data.
- **Import/Export Functionality:**
  Developed features for data import/export.
- **Team Collaboration:**
  Enhanced teamwork skills, managing code merges and resolving conflicts with Git.
- **Time Management:**
  Improved project planning and time management, setting realistic goals and tracking progress effectively.

Conclusion:
This project provided a comprehensive learning experience, enhancing technical skills in software development and highlighting the importance of effective project management and collaboration.

6) Used Design Pattern:

Command Pattern:
It's a behavioral design pattern where requests are encapsulated as objects, allowing parameterization of clients with queues, requests, and operations. It decouples sender and receiver, providing flexibility and extensibility in handling requests.
Primarly used in:
TourPlanner/Viewmodels/RelayCommand.cs

Dependency Injection:
It's a design pattern for removing hard-coded dependencies from an application by injecting them at runtime. It improves flexibility, maintainability, and testability by allowing components to be loosely coupled and easily replaceable.
Primarly used in:
TourPlanner/App.xaml/App.xaml.cs

7) Testing Decisions:

DirectionServiceTests: This set of tests verifies the functionality of the DirectionService class, which is responsible for retrieving directions and route information from an external service. It ensures that valid requests return the expected results, handles invalid input gracefully, and properly handles unsuccessful responses from the external service.

ReportManagerTests: These tests focus on the ReportManager class, which generates reports based on tour data. It checks whether reports are correctly generated in various scenarios, such as for individual tours and summary reports, and ensures that the generated PDF documents contain the expected content.

RouteDataManagerTests: This test suite evaluates the functionality of the RouteDataManager, which manages route-related data processing. It verifies that tour data is retrieved correctly, handles exceptions appropriately, saves directions to a file as expected, and retrieves distance and duration information accurately.

TourServiceTests: These tests assess the functionality of the TourService class, which interacts with the database to perform CRUD operations on tour entities. It confirms that tours can be added, retrieved, updated, and deleted correctly from the database.

TourViewModelTests: This set of tests focuses on the TourViewModel, which serves as the view model for the tour-related views in the application. It verifies that the view model correctly handles operations such as starting editing, saving tours (both new and existing), and deleting tours, ensuring that the underlying service and data manager interactions are handled appropriately.

8) Time Spent:

The time spent on the TourPlanner project was carefully tracked throughout its development. The breakdown of time allocation is as follows:
Design and Planning: 25 hours (Asanov: 10 hours, Reichenberger:15 hours)
Included architectural decisions, database schema design and outlining the overall project structure.
Implementation: 80 hours (Asanov: 35 hours, Reichenberger: 45 hours)
Actual coding and development of the graphical user interface, ORM, API, etc. .
Testing and Debugging: 20 hours (Asanov: 10 hours, Reichenberger: 10 hours)
Dedicated time to write unit tests, testing GUI and debugging.
Documentation: 5 hours (Asanov: 4 hours, Reichenberger: 1 hours)
Time spent on creating this protocol and other necessary documentation.
Total Time: 130 hours (Asanov: 59 hours, Reichenberger: 71 hours)

9) Link to Git Repository:

The Git repository for TourPlanner project is available at the following link:
https://github.com/Alino22Git/TourPlanner