Design Document for .NET MAUI Desktop App - Weather and Public Transport Integration

Table of Contents

1. Introduction

This design document outlines the structure and components of a .NET MAUI desktop application that integrates data from three different REST APIs: positionstack, weather forecast and public transport. The primary goal of the application is to provide users with weather information when departing and arriving at their destinations, allowing them to find public transport routes from place A to B while considering configurable weather data.

2. Application overview

The application will consist of the following major features:

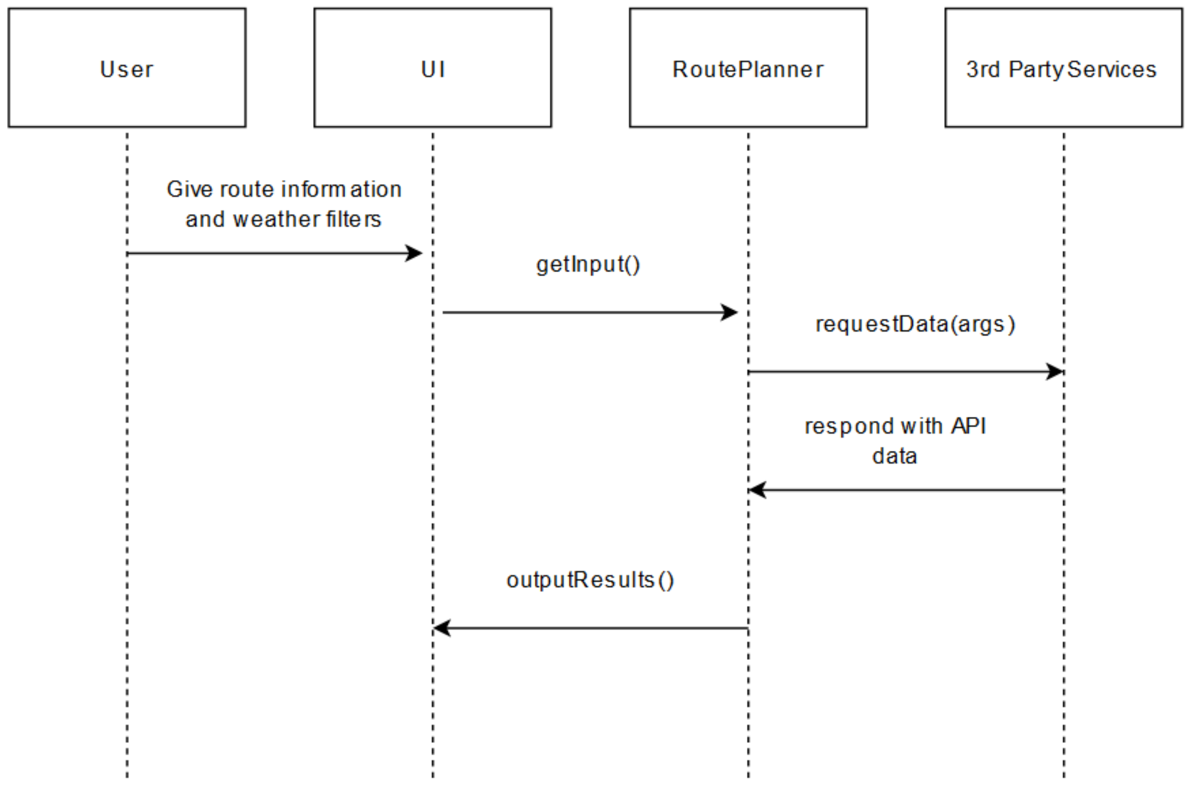
- Will convert street addresses to latitude and longitude coordinates.
- Weather information display for the departure and arrival locations. Mostly useful when there is walking on the journey.
- Public transport route planning between two user-defined locations.
- Configurable weather data to influence route selection.
- User-friendly graphical user interface (GUI).

3. System Architecture

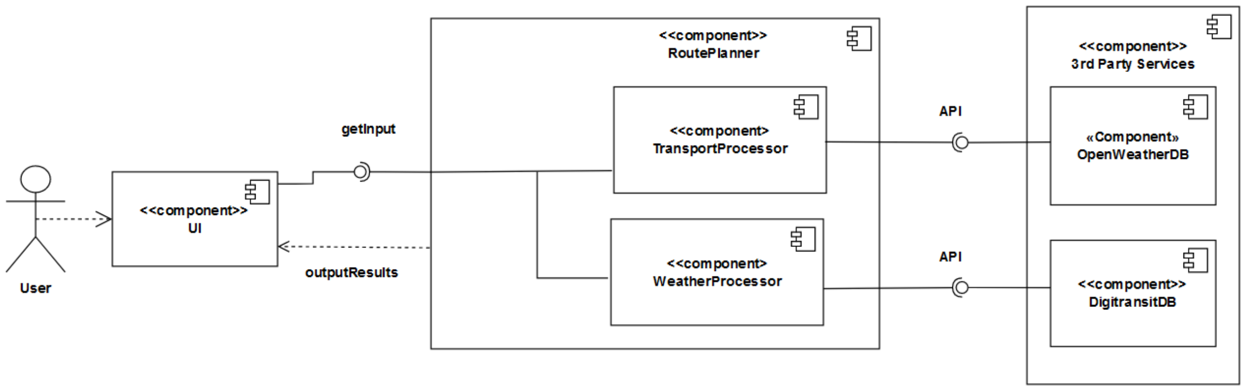
The proposed system architecture follows a client-server model, where the client application communicates third party weather and public transport APIs to retrieve data. The key architectural components include:

- Client Application: The .NET MAUI desktop application responsible for presenting the user interface and handling user interactions.
- API Integration Layer: This layer manages communication with the external REST APIs. It includes modules to retrieve weather forecasts and public transport route information.
- Data Processing Layer: Responsible for processing and formatting data received from APIs for presentation in the user interface.
- User Interface Layer: The graphical user interface, which displays weather information and allows users to plan public transport routes. It interacts with the API integration layer and the data processing layer.
- There is no other backend on this project. All data is fetched from third party APIs by the client application.

.



4. Key Components and Interfaces



Client Application

Main window: User can interact with the GUI and input their departure and arrival locations.

Settings Panel: User can optionally configure weather-related filters (temperature, chance of rain etc.) to influence what are the most optimal routes.

Data Processing Layer

Route-planner processor: combines the data received from the weather and public transport API.

Weather Data Processor: Responsible for parsing and formatting weather data received from OpenWeathers API.

DigiTransit Data Processor: Processes and organizes public transport route data.

API Integration Layer

OpenWeather API: An interface defining methods for retrieving weather forecast data.

DigiTransit API: An interface defining methods for obtaining public transport route information.

Geocoding API: In interface converting street addresses to latitude and longitude coordinates.

5. API Integration

We plan to use the following APIs:

OpenWeatherMap: We will be using OpenWeatherMap for weather forecasting. It's easy to use REST-API and suits well for our project's needs. It gives accurate enough data for our use case. It also offers geocoding which we will be utilizing on the digitransit API side.

Digitransit API: We will be using digitransit API for fetching routes from place A to B inside Finland. This API offers detailed view of different routes.

Geocoding API: We will be using geocoding API to convert street addresses to latitude and longitude coordinates. This is used to fetch data from openweathermap and digitransit APIs.

6. User Interface

The user interface will be designed to be intuitive and user-friendly. We will employ modern design principles for a responsive and visually appealing desktop application. Visual components will include:

- From-to screen which contains autocompleted input fields and submit button along with modern visualization.
- Travel settings which contain sliders for user input. User can change different variables of the weather configuration.
- Result view which contains search results of the trip. Result view is scrollable list which contains results of the search. Most matching results are shown at the top.
- Detailed result view containing visual presentation of the data, including graphs.

7. Configuration and Customization

Users will be able to customize their weather-related preferences, such as temperature ranges, precipitation thresholds, and preferred weather conditions for route planning. These configurations will influence the suggested routes. User will also have possibility to save and load previous searches.

8. Design Patterns and Approaches

While the choice of specific design patterns and architectures may evolve during development, some preliminary approaches include:

- MVVM (Model-View-ViewModel): Utilize MVVM architecture to separate the presentation logic from XAML UI. This is the main purpose of using MVVM, since the application doesn't have a backend, no model approach is required.
- Dependency Injection: Implement dependency injection to promote loose coupling between components and facilitate unit testing.
- RESTful API Consumption: Adopt RESTful principles for communicating with external APIs, ensuring a standardized and scalable approach.
- Caching: Implement caching mechanisms to reduce API calls and improve application responsiveness.

9. Usage of Artificial intelligence

10.1 ChatGPT for Ideation and Planning

ChatGPT, powered by OpenAI's GPT-3.5 architecture, plays a pivotal role in shaping the conceptualization and planning of our project. Here's how we leverage ChatGPT:

- Idea Generation: We initially used ChatGPT to brainstorm project ideas. It assisted us in exploring various concepts, provided insights, and helped refine our initial vision.
- Design Assistance: ChatGPT has been instrumental in drafting documents, including this design document. It assists in structuring content, suggesting relevant sections, and refining the language for better clarity.
- Problem Solving: Throughout the project, ChatGPT serves as a valuable resource for troubleshooting and problem-solving. It aids in understanding and resolving technical challenges.
- Prototyping Guidance: ChatGPT assists in creating rough prototypes and mockups, allowing us to visualize and iterate on our ideas more effectively.

10.2 GitHub Copilot for Efficient Coding

GitHub Copilot is a code completion tool that utilizes AI to assist developers in writing code more efficiently. We employ GitHub Copilot in our coding phase for various tasks:

- Code Generation: Copilot generates code snippets, functions, and even entire classes based on the context of our coding project. This accelerates development by automating repetitive coding tasks.
- Code Suggestions: Copilot provides real-time code suggestions as we type, enhancing code quality and adherence to best practices. It offers insights into coding patterns and standards.
- Documentation Assistance: GitHub Copilot aids in creating code documentation, making it easier for team members to understand and maintain the codebase.
- Error Handling: It assists in identifying and rectifying errors, reducing debugging time and ensuring a more stable codebase.

10. Conclusion

This design document outlines the key components, interfaces, and architectural considerations for the development of a .NET MAUI desktop application that integrates weather forecast and public transport information. As development progresses, we will refine these designs and adapt to emerging requirements and best practices in software design.