WorldTravels Application on .NET with Azure

Anna Ostrowska

January 24, 2025

Contents

1 Motivation

In today's world, planning a trip can be overwhelming for some people due to the large number of available options. The WorldTravels application addresses this problem with the aim of simplifying the process of choosing a destination by providing users with personalized travel recommendations. By tailoring suggestions based on budget, preferred activities and surroundings, regions, and weather conditions, the application helps users to discover their ideal destinations with ease.
This app was also created to encourage people to see new places, which are often forgotten when considering travel destinations. The big cities which are popular among tourists can be worth visiting, but so can be other, not so popular locations, This app has a database with hundreds of destinations, which allows users to discover places there would never think of visiting.
This project was created for *Introduction to .NET Software Development on Azure* course at Johannes Kepler University Linz. I highly recommend visiting WorldTravels GitHub repository.

2 Introduction

The *WorldTravels* application is a Blazor Web App that provides users with personalized recommendations of travel destinations based on their preferences such as budget, activities, regions, and weather. It was created in .NET using Visual Studio 2022. Several other sources and services were used for the project, such as datasets from Kaggle, SQL Server Management Studio 20, Python in Pycharm and Azure services: Azure Maps, Azure Weather API, Azure App Service and Azure SQL Database. This report outlines the application architecture, classes and components of the code, usage, and deployment structure on Microsoft Azure.

2.1 Application features

Home page provides a map with all possible recommendations destinations showed.

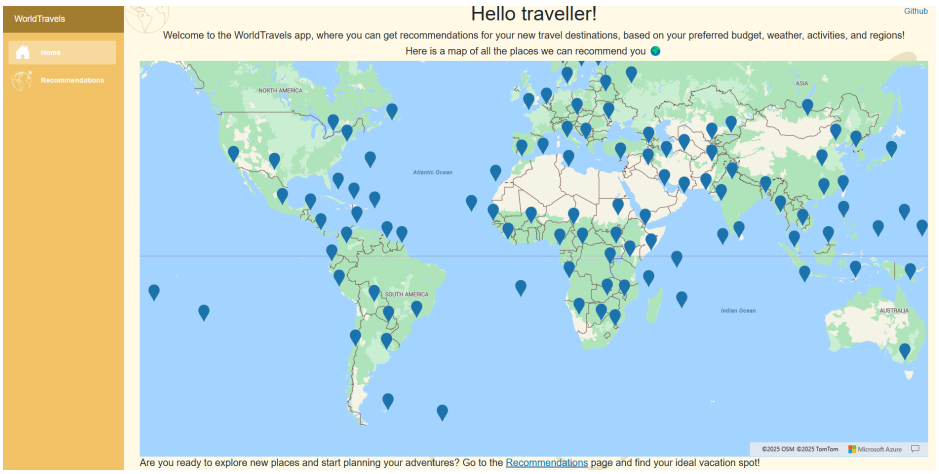

Figure 1: Home page

Recommendations page allows user to mark their preferences, such as budget, start and end date of the trip, preferred activities, regions and temperatures.

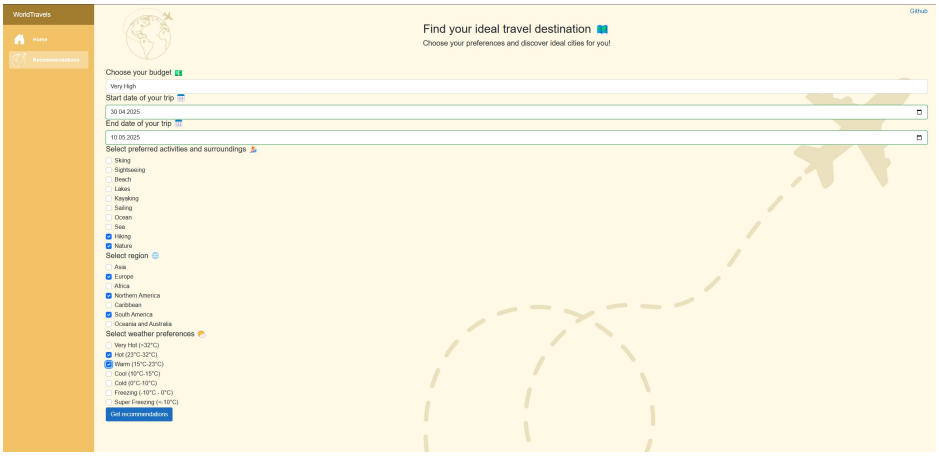

Figure 2: Recommendations page - user preferences

Then after clicking *Get recommendations* button, the app displays filtered recommendation and their features and presents them on the world map.

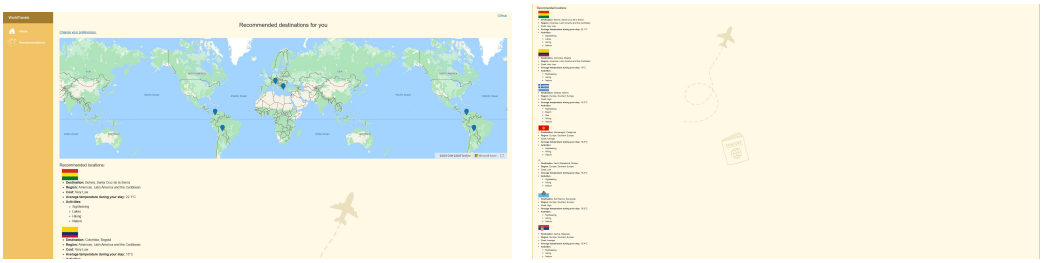

Figure 3: Page with recommendations based on user's preferences.

Application handles exceptions, such as no available recommendations, wrong dates or no budget selected by displaying suitable messages on the screen.

3 Architecture

The development process of the application consists of five main stages:

1. **Idea Design**: The initial concept and user experience were designed using Canva to create a clear and intuitive interface.

2. **Data Collection and Integration**: Relevant data about locations and their attributes were sourced from Kaggle. The data was then processed and stored in an SQL database using Python.

3. **Data Enhancement**: Missing information, such as travel costs and activities, was filled using SQL queries to ensure the database was comprehensive and accurate.

4. **Weather Data Integration**: Historical weather records were retrieved using the Azure Maps API and stored in the SQL database. This step was implemented in C# using the .NET framework.

5. **Application Development (code)**: The final application was developed in C# using Blazor WebApp, resulting in a dynamic and interactive user interface.

Additionally, Azure was used to deploy both the database and the application in the cloud, hosted on the domain worldtravels.azurewebsites.net.

3.1 Idea Design

The app was designed with a focus on a user-friendly and straightforward interface. Initially, the design was created in Canva to clearly define the goals and requirements. While the final version differs a bit visually, the main elements remain consistent, ensuring the app delivers the same services and user experience as originally intended.

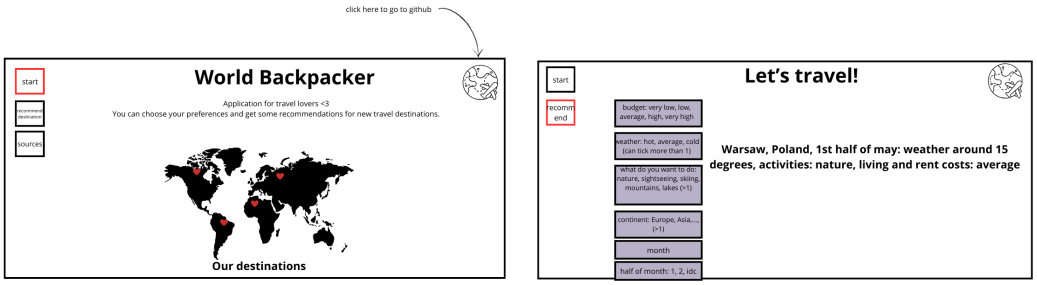

Figure 4: Prototype of the web application.

3.2 Data Collection and Integration

In order to get essential, initial data to create the database of cities to recommend in the application, I downloaded some databases from Kaggle and imported them to app's database.
Data about cities, their locations (countries, longitudes and latitudes) ad population was taken from *World cities database*[1].
Data about countries, their regions and sub-regions and URLs of their flags was taken from *Countries ISO Codes |Continent |Flags URL*[2].
Information about cost of living and rent (with indexes) was taken from *Cost of Living Index by Country*[3] from Kaggle, but scraped from Numbeo[4].
Then these 3 databases were uploaded to the database with Python.

3.3 Data Enhancement

Missing data was added manually with SQL queries, by checking information about ex. cost of living on the Internet. Activities available in each city (skiing, sightseeing, beach, lakes, kayaking, sailing, ocean, sea, hiking, nature) were also added manually, based on my own experience and research.

Available countries dataset was filtered, taking one city with the largest population from every country, adding 70 other cities with the biggest population from all over the World and adding some other locations by manually (which I though are worth visiting).

3.4 Weather Data Integration

Historical temperature data for every location was retrieved using Azure Maps API - Weather. Average temperature was calculated in C#, taking the average temperature in each location for every week in every month.

3.5 Application Development

3.5.1 Architecture

The application follows a modular design, combining backend (Services), frontend , and shared data classes to ensure a clean separation of concerns and tasks. Below is a detailed description of the architecture components.

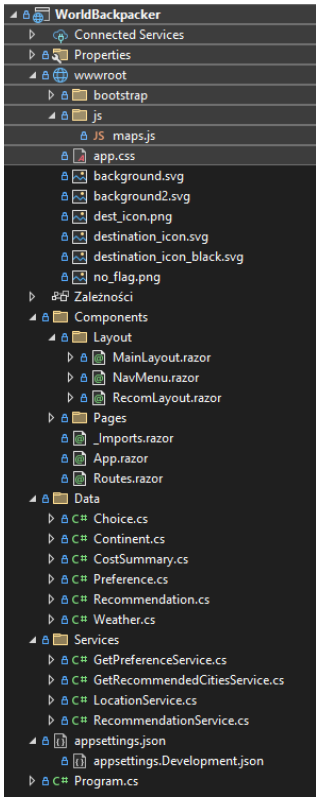

Figure 5: Architecture of the code

3.5.2 Backend Services

The backend consists of multiple services, each responsible for a specific functionality:

- **LocationService**: Fetches geographic coordinates of recommended destinations.
- **GetRecommendedCitiesService**: Queries the database to retrieve cities based on user preferences.
- **GetPreferenceService**: Processes user inputs to create a structured preference object used by recommendation logic.
- **RecommendationService**: Manages and stores recommendations fetched from the backend, ensuring data consistency across pages.

3.5.3 Data Classes

The `Data` folder contains the data model classes used to structure and handle application data. Key classes include:

- **Choice.cs**: Represents user-selected options during the recommendation process.
- **Continent.cs**: Enum defining regions.
- **CostSummary.cs**: Enum defining a level of travel costs (very high, high, average, low, very low).
- **Preference.cs**: Represents user preferences like budget, activities, dates, region and weather requirements.
- **Recommendation.cs**: Represents a recommended destination and its associated attributes.
- **Weather.cs**: Enum representing weather types (ex. SuperHot).

3.5.4 Frontend

The frontend is built using Blazor Web App. Key folders include:

- **Components/Layout**: Contains layout components like `MainLayout.razor`, `NavMenu.razor`, and `RecomLayout.razor`.
- **Components/Pages**: Includes application pages such as `App.razor` and `Routes.razor`, responsible for defining navigation and core user interactions.

3.5.5 Map Integration

The application incorporates interactive map functionality using the `maps.js` script, which is located in the `wwwroot/js` folder. The script uses Azure Maps to provide a dynamic map interface with marked locations.

3.5.6 Configuration and Resources

The application configuration is managed via:

- `appsettings.json`
- `Program.cs`: For initializing the application.

Static assets such as icons and backgrounds are stored in the `wwwroot` folder.

4 Azure

The application is hosted on Azure:

- **Azure App Service**: Hosts the Blazor Web App.
- **Azure SQL Database**: Stores the application data with security measures.
- **Azure Key Vault**: Secures sensitive configuration data such as database connection strings.

References

[1] SimpleMaps.com. *World cities database*. 2024. DOI: 10.34740/KAGGLE/DSV/7903661. URL: https://www.kaggle.com/dsv/7903661.

[2] AndresHG. *Countries ISO Codes — Continent — Flags URL*. URL: https://www.kaggle.com/datasets/andreshg/countries-iso-codes-continent-flags-url.

[3] Numbeo. *Cost of Living Index by Country - Kaggle*. 2024. URL: https://www.kaggle.com/datasets/myrios/cost-of-living-index-by-country-by-number-2024.

[4] Numbeo. *Cost of Living Index by Country - Numbeo*. URL: https://www.numbeo.com/cost-of-living/rankings.jsp.