# Instructions to build and run the application

1. Ensure you have Visual Studio 2022 installed on your system.
2. Open Visual Studio: Launch Visual Studio 2022.
3. Download the source code as a ZIP file from the email attachment.
4. Open Visual Studio. Go to "File" > "Open" > "Project/Solution." Select the project folder from the extracted ZIP download.
5. In Visual Studio, go to "Build" > "Build Solution" to compile the application.
6. After a successful build, press F5 run the application.
7. A web browser will open, and you can access the application.
8. Enter an address in the search box and click "Search" to retrieve location results.
9. Click on a location result to view the 10 closest locations on the map.
10. The 10 closest locations will be displayed on Google Maps below the search results.

# Documentation of the Locator App

## High-level Requirement

1. Write an application that allows the user to search the attached Data.csv file on the Address field. It does not need to search the City, State or Zip fields.
2. Display the results to the user.
3. When a user selects a row in the results
4. Display another set of results that shows the 10 closest locations in Data.csv based upon the selected row in the first set of results.

## Solution Overview

The solution is a web application that enables users to search for and display location data based on address and proximity. It incorporates the following key components and design principles:

**Key Components:**

1. **Data Import**: The application loads location data from a CSV file into memory.
2. **Address-Based Search**: Users can search for locations based on an address query.
3. **Proximity-Based Search**: When a user selects a location from the search results, the application displays the 10 closest locations to the selected one.
4. **Web Interface**: The solution is built as an ASP.NET Core web application with Razor Pages, providing a user-friendly web interface with Google Map JavaScript.

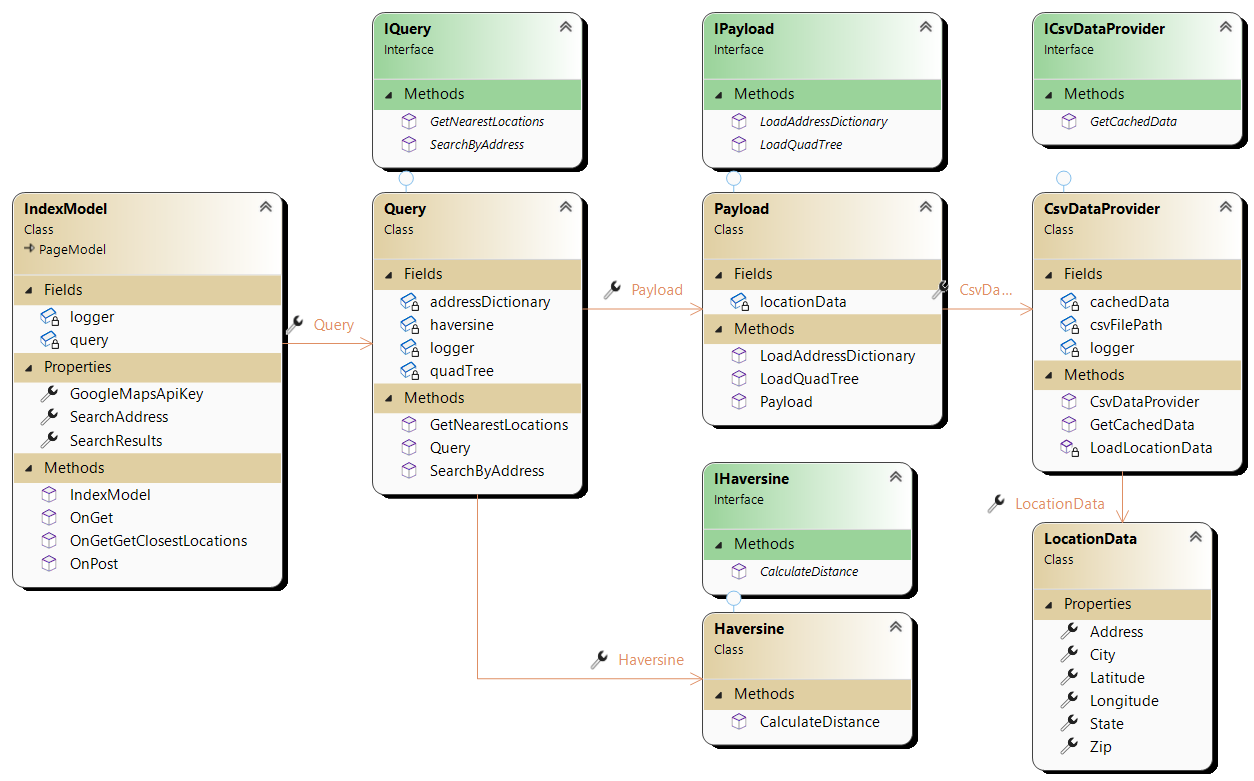
**Design Principles Adopted:**

1. **Separation of Concerns**: The solution separates functionality into distinct components for data loading, address-based search, and proximity-based search.
2. **Dependency Injection:** Utilizes dependency injection to manage services and promote modularity.
3. **Use of Spatial Index:** Employs Quad-Tree for efficient proximity-based searches.
4. **Efficient Distance Calculation:** Implements the Haversine formula to calculate distances between latitude and longitude coordinates.
5. **SOLID Principles:** Adheres to SOLID design principles for code maintainability and extensibility.
6. **Efficient Data Structures**: Chooses appropriate data structures (e.g., dictionaries and Quad-Trees) for specific tasks.

Technology Stack and library

ASP.NET Core 6 Razor Page, NetTopologySuite

## Class Diagram



## Class Diagram Definition

|  |  |
| --- | --- |
| Class | Comment |
| **IndexModel (Razor Page)** | * This is the main Razor Page for the application. * The page allows users to search for locations based on address and displays the results. * When a user selects a location, it calls the OnGetGetClosestLocations handler to find and display the 10 closest locations. |
| **Query** | * This class handles the logic for searching by address and finding the nearest locations. * It uses a Quad-Tree data structure for spatial indexing. * The GetNearestLocations method finds the 10 closest locations using the Haversine formula for distance calculation. * The SearchByAddress method performs a case-insensitive partial match on the address to find matching locations. |
| **Payload** | * This class is responsible for loading data and creating data structures. * It loads data from a CSV file and creates a dictionary of addresses with associated location data. * It also creates a Quad-Tree for spatial indexing. |
| **Haversine** | * This class provides the Haversine formula for calculating the distance between two sets of coordinates. * It's used by the Query class to calculate distances for proximity searches. |
| **CsvDataProvider** | * This class loads location data from a CSV file. * It uses the CsvHelper library to read data from the CSV file. |
| **Program.cs** | * The entry point of the application. * It configures services, such as the CsvDataProvider, Payload, Haversine, and Query. * It sets up the HTTP request pipeline and handles errors. |

## Data Structure: Performance and Memory management decision

**Data Structure Choice:**

Dictionary vs. List: The choice of a Dictionary for address-based search and an Quad-Tree for proximity-based search was made to optimize performance and memory management.

**Performance Justification:**

Dictionary vs. List: While a List could be used for address-based search, it would have a higher time complexity for search operations (O(n)), where n is the number of records. This can result in slower search times, especially with a large dataset. In contrast, a Dictionary's key-value lookup has an average time complexity of O(1), ensuring near-instantaneous search times.

Tree vs. List: For proximity-based search, a Tree was chosen over a List to optimize performance. With a List, finding the 10 closest locations would require calculating distances between the target point and all locations, resulting in a time complexity of O(n), which could be prohibitively slow for large datasets. The Tree's query operation, with a time complexity of O(log n), significantly reduces the number of distance calculations and speeds up proximity-based searches.

**Memory Management:**

Dictionary vs. List: The Dictionary efficiently manages memory for address-based search, as it stores data by address key, avoiding redundancy and conserving memory. A List would consume more memory, as it stores duplicate data entries.

**Load Once and Dependency Injection:**

Singleton Lifetime and DI vs. Recreating Data: The choice to load data once, cache it, and use dependency injection with a singleton lifetime is a best practice to optimize performance and memory. Without this approach, recreating data structures for every request would consume more memory and reduce performance.