Hiran Amarasinghe

hiransamamarasinghe@gmail.com

Abstract

This document detailed the design principles, tech stack and future improvements

system design document

Food Truck Service

Contents

[Introduction 2](#_Toc103867476)

[Technology stack 2](#_Toc103867477)

[Design Considerations 2](#_Toc103867478)

[Application Configuration 2](#_Toc103867479)

[Architectural Design 4](#_Toc103867480)

[FoodTruck Api 5](#_Toc103867481)

[Design concepts 5](#_Toc103867482)

[Unit testing 5](#_Toc103867483)

[System Constraints 5](#_Toc103867484)

[Api documentation 5](#_Toc103867485)

[Client-side technology stack 6](#_Toc103867486)

[Improvements for the next phase 6](#_Toc103867487)

# Introduction

Food truck service api is designed to provide closest food trucks for a given coordinates within a geographical area. The selected geographical area could be a city, building, street, etc. The Food truck service api is designed to read data from different types of sources such as CSV file, RESTful service, or database. For demonstration purpose an Angular based SPA application was developed to access the Food truck api. Angular application fetch user current location (latitude and longitude) details to find the closest food trucks. In order to find the distance between two geographical locations, the Haversine formula is used. Moreover, user can search for food trucks by preferred food items. Hence, the search will first filter the trucks by food items, its status and then by its location. The document has outlined the technology stack and suggestions to improve the system.

# Technology stack

* Food truck Finder Api is implemented using Asp.net core 6 minimal api. Minimal api is the latest the approach to develop RESTful microservices with minimal dependencies.
* Angular 13 as single page application (SPA)
* Food Truck Source data are stored as a csv file in api project. This source is cached in-memory for fast retrieval. CSVHelper nuget package is used to read CSV file and convert it into in-memory data collection.

## Design Considerations

* The API performs higher CPU operations to calculate distance between two coordinates. Hence, Nodejs is not a suitable operation as it a single threaded application and it is designed if your application involves asynchronous activities. Hence, Asp.net core was chosen as it supports multi-threaded interaction for better performance
* LINQ is used to perform querying the CSV data source. However, the application can be configured to consume data from REST endpoints/Data base and therefore LINQ could be used to either translate to SQL queries which results minimal code changes.
* CSV Datasource is cache in memory for faster retrieval of data
* Middleware were used to handle all errors in the application and convert those errors into standard HTTP codes

# Application Configuration

FoodTruck api configurations are stored in settings.json file and those used to control the dynamic configuration of the application. Since, application does not use any secrets or keys, there is no requirement to use a secret vault or environment variables.

**MedianLocation - This setting contains city coordinates and the max distance between two coordinates in the city. For instance**

**"MedianLocation": {**

**"Latitude": 37.7749,**

**"Longitude": -122.4194,**

**"DistanceInKms": 15**

**},**

**AllowedDomains - Governs the CORS policy. This allow which domains could request data**

**MinRecordsToView - Governs minimum search records count to return from a query**

**"DataSource": {**

**"Name": "CSV",**

**"Path": "Mobile\_Food\_Facility\_Permit.csv"**

**}**

# Architectural Design

**Middleware pipeline**

**(**Error Handling Middleware**)**

**FoodTruck Api**

**(**REST endpoint**)**

InMemory Data

CSV Data

**FoodTruck Service**

**(Business Logic layer)**

**Unit Tests**

**FoodTruck Domain**

**(Models, Entities, Contracts)**

## FoodTruck Api

|  |  |  |
| --- | --- | --- |
| Endpoint | Description | Parameters |
| Get – Http GET | Search Food truck | ​/FoodTruck​/{latitude}​/{longitude}​/{foodsearch} |
| GetByAddress – Http GET | Search Food truck by address | /FoodTruck/{location} |

## Design concepts

Distance between two geographical location is calculated using Haversine formula. This formula calculates the great-circle distance between two points on a sphere. All the distance are calculated using Kilo meters (km)

# Unit testing

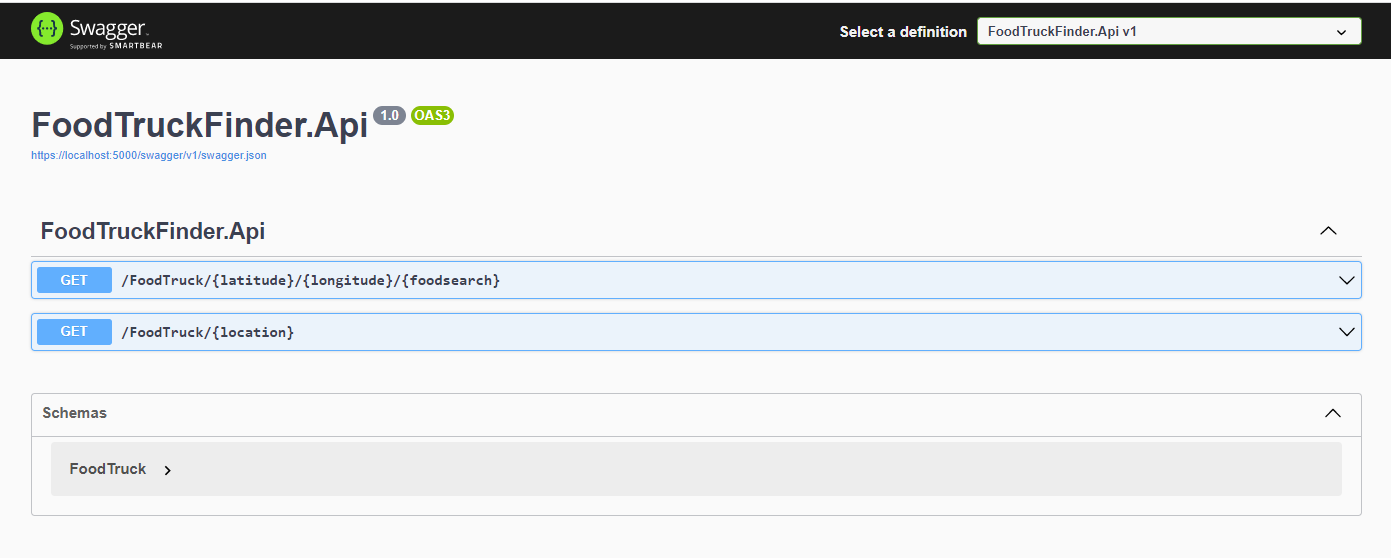
Api methods are unit tested using XUnit. In-memory provider for CSV data source is configured for unit testing

# System Constraints

Food truck api is designed to search closest food trucks for the given coordinates using **Haversine formula**. This calculation is CPU intensive operation rather IO intensive operation. Hence, to minimize unnecessary calculation the API calculates if the provided coordinates are within the city premises. To provide a generic solution, the api is configured with city coordinates and the maximum distance within two locations in the city. For instance, we could use Google maps to find city coordinates and maximum distance between two coordinates within the city. If the distance between user and city is more than the configured distance, it is assumed that the user is outside the city premises and hence calculation is not performed.

# Api documentation

Swash documentation is used to document api endpoints. The documentation describes endpoint name, Type (GET, POST, etc), parameters, response codes (Http 200, 400, etc)



# Client-side technology stack

* Angular 13
* google maps package
* Angular Core UI Admin template
* Bootstrap

# Improvements for the next phase

Following are suggestions to improve the system in terms of deployments, features, performance, scalability, and security.

* Provide social media sign-in option to authenticate users using Oath 2.0 and retrieve user information such as users name and profile picture. This information could be used to rate the food trucks and feedback. The rating information could be utilized to search food truck.
* Api will be restricted only authenticated requests such as authenticated users, third part service access via API key
* Cache search results for faster retrieval based on geographical location.
* Containerization of Api and Client application for deployments, portability, and scalability.
* Devops pipeline configuration to build and deploy api and client to a cloud infrastructure. This approach is ideal as it does not require maintenance of infrastructure and security.