Redfin Work Sample

Introduction

The food truck console application is written in C# using Visual Studio 2019 and .NET Core 3.0. Its purpose is to print a list of food trucks that are open at the time that the application is run. The results are printed alphabetically by name, in batches of 10, until no additional open food trucks are available.

Food Truck Console Application

The application's repository layer retrieves an array of San Francisco-based food truck information in JSON format via Socrata API. This information includes the food trucks' names, addresses, and opening hours. The raw JSON is deserialized into an enumerable of mapped C# objects, namely, in the form of the FoodTruck.cs class. Beyond this one-to-one attribute mapping, the FoodTruck class contains TimeSpan members that are used for date and time comparisons.

The service layer filters the result-set by those food trucks that are open at the time at which the program is run, and then sorts ascendingly by name alphabetically. This ordered and reduced result-set is cached in memory for 30 minutes to minimize bulky calls to the third-party API, and prevent redundant data manipulations.

The main driver of the application invokes the service-tier to query for food trucks in the result-set from memory. Each call maintains a batch size that is limited to 10, and pagination (or offset) is incremented in response to the user hitting 'Enter' in the console to retrieve additional results. The results continue to be printed until no additional food trucks are available to be shown.

Scaling for a Web Application

To support a fully-featured web application, I could follow a similar architecture to that of the console application with one notable change.

Assuming the Socrata API remains the data provider, I would not be able to leverage in-house database procedures to perform the fundamental querying, filtering, and sorting (potentially a significant optimization in itself). Additionally, storing a result set in-memory on the server would be inappropriate to service millions of concurrent users due primarily to problematic space considerations. On this point, I believe a distributed cache like Redis would be best for memorization and shared caching. Further, Redis can be used to store subsets of sorted sets which is useful for API pagination.