RedFin Assessment Explanation

How to run this JAR in local

1. To run this code
    a. Dependancies :
        i. jackson-annotations-2.9.0
        ii. jackson-core-2.9.6
        iii. jackson-databind-2.9.6
        iv. java-sdk 8
        v. socrata API endpoint

        In order to build this application, run the app. Get the JAR.
        And then use **java -jar <name of JAR>.jar**
        To run this application.

        NOTE: This code does not compile on the IDE Provided due to some IDE issues, but then I was able to run the application in my local.
        And this is JAR I have obtained from running the app. [refer to the email]

        Java -jar RedfinTakeHome.jar

This will execute the application. When the application executes, it will display the first ten food trucks that are open. The user will have three options:

1. Type 'q' to quit the application
2. Type 'n' to grab the next 10 food trucks
3. Type 'p' to grab the 10 previous food trucks

Part 2:

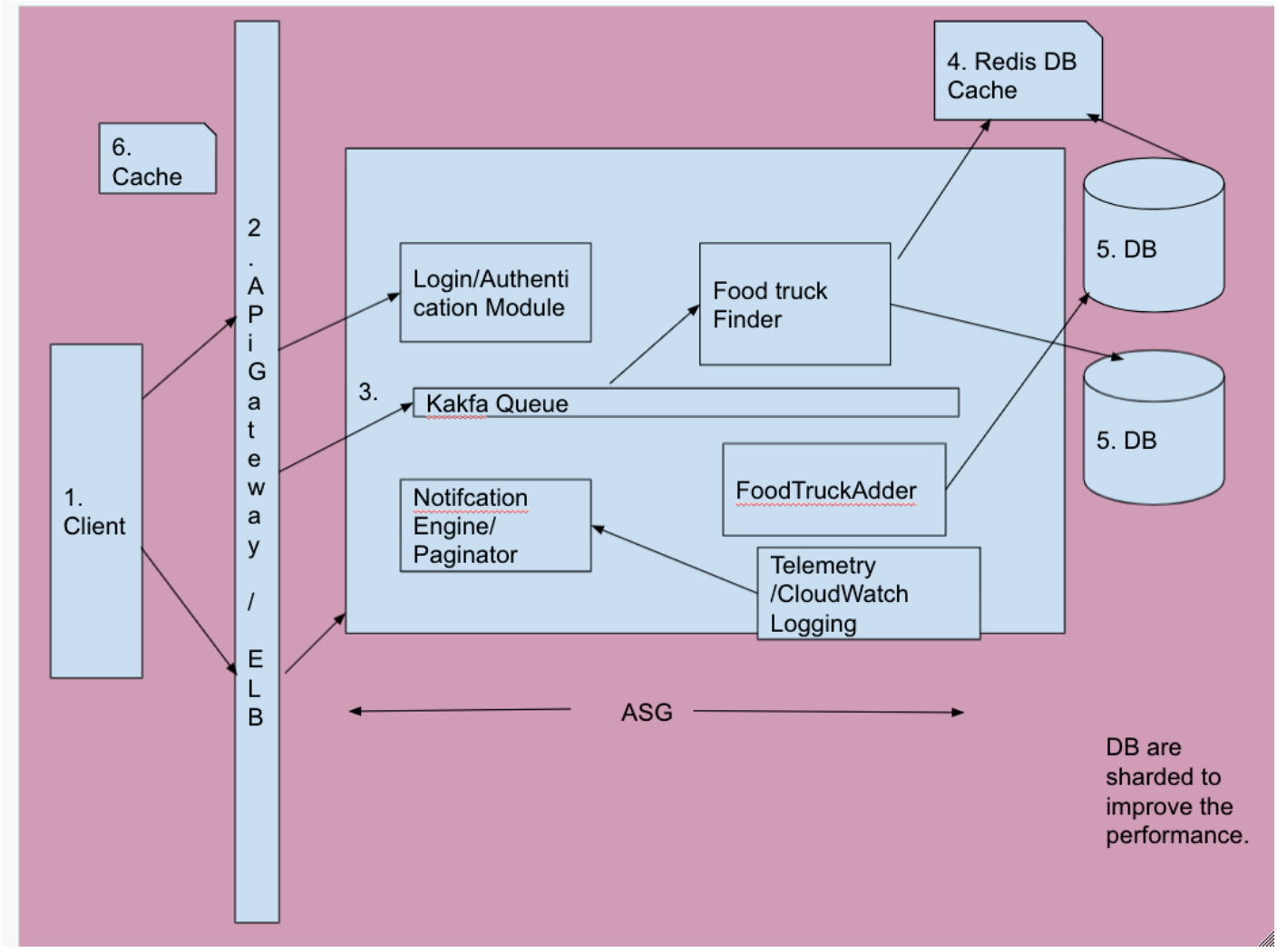
Here are the things I would do to make this application highly scalable and concurrent.
This is the architecture diagram I will design along with the explanation for each module.

System Requirments:

1. Understand the use cases and how much users will be hitting the end points.

2. If the application is used by millions of people from all over the words, we can employ auto-scaling of the instances and make sure during the peak times the app scales up and scales down to serve the request.
3.

Here is my proposed architecture:
I used google draw to capture this image and I have explained in detail about the various components.
(Link: https://docs.google.com/drawings/d/1p1KKcYVakw8f8LbwxqnByLU5r_Oin-z1whbNNaDPFeE/edit)



Explanation of each component :

In order to make this application highly available and scalable to serve multiple requests, we need to make this asynchronous. The following is a brief description of each component and their choice

Assumptions:

1. Clients access applications using mobile/desktop app on HTTPS traffic.
2. They have to be logged in to access our REST APIs.

Component	Description	AWS Specific Application
1. Client	Here the client could be any user trying to use a mobile/desktop to access the native app.	Could be simple iOS/Android app or Desktop application
2. API gateway	1. The main function of the API is to route the traffic to the corresponding server based on their location. 2. Its main purpose is to route the traffic to the nearest server based on their location(to minimize latency) 3. Perform SSL off-loading aka convert HTTPS traffic to HTTP traffic in order ti reduce the load of the application/instances inside the architecture.	Could be implemented using Application Load Balance + Route 53 for geo-location-based routing.
3. Cloud Hosted Application (It could be a Virtual Private Cloud)	1. **Login Module:** This is a separate module for authenticating the user and makes sure a valid session is established before the user access endpoints. A token is returned when the user has been successfully validated, if not the request is rejected. 2. **Queue**: The purpose of the queue is to provide asynchronous communication and save the incoming client request. 3. **Food Truck Finder:** This picks the request from kafka, and is implemented using a quad-tree server. The idea behind using a quad-tree is to quickly and efficiently retrieve all near by food-trucks based	VPC. It is also replicated across regions say for example we can have regions-based replication across US-EAST or one in ASIA /EUROPE depending on our user base. For Queue: I have chosen Kaka since its highly available and can accept a large stream of requests and is fault-tolerant. QuadTree server: backed is using Dynamo DB since it provides a Key-value structure to store data and has DAX for in-memory fast cache access.

	on zipcodes or coordinates that the user has entered. This saves data in a Hierarchical database we can cache the “hot” responses using a redis or in-memory DB 4. These instances are implemented using an auto-scaling groups that can scale up/down real quick based on traffic 5. Notifcation Engine/Paginator: this gets the response from the FoodTruck Finder, and sends the response to client based on the client capacity aka it slows down the response to the client and sends only 10-specified number of data to client	Redis for in-memory Database We can leverage AWS’s auto-scaling groups of instances to quickly support for several loads.
Auditing Tools	We can implement cloud watch for logging and monitoring purposes. We could also send the logs to Splunk where we can persist and save the logs based on indexing and later can be used for querying.	Splunk, Cloud watch (AWS), Hot-Hot deployment (Blue/Green Deployment) AWS Shield (to protect against DOS and DDOS attacks)