**Software Design Document**

**for**

**StreetFoodLove**

**Version 0.16 approved**

**Prepared by Silvia Casaburi, Makoto Emura, Wenhuan Tan, and Colin Zhou**

**Instructor: Sara Farag, Ph.D**

**Course: CS 482**

**Date: 05/30/2022**

Table of Contents

[1. Revision History 3](#_Toc104991107)

[2. INTRODUCTION 5](#_Toc104991108)

[2.1 Purpose 5](#_Toc104991109)

[2.2 Scope 5](#_Toc104991110)

[2.3 Overview 6](#_Toc104991111)

[2.4 Reference Material 6](#_Toc104991112)

[2.5 Definitions and Acronyms 7](#_Toc104991113)

[3. SYSTEM OVERVIEW 7](#_Toc104991114)

[4. SYSTEM ARCHITECTURE 9](#_Toc104991115)

[4.1 Architectural Design 9](#_Toc104991116)

[4.2 Decomposition Description 10](#_Toc104991117)

[API documentation 10](#_Toc104991118)

[Getting and modifying resources 10](#_Toc104991119)

[Authentication flow 11](#_Toc104991120)

[Sign-in with Google flow 12](#_Toc104991121)

[How to get a page and its pictures 13](#_Toc104991122)

[How to post a review with pictures 14](#_Toc104991123)

[Discount exchange scenario 15](#_Toc104991124)

[Leaflet map overlays 15](#_Toc104991125)

[AWS OpenSearch 16](#_Toc104991126)

[4.3 Design Rationale 20](#_Toc104991127)

[5. DATA DESIGN 21](#_Toc104991128)

[5.1 Data Description 21](#_Toc104991129)

[5.2 Data Dictionary and EER Model 21](#_Toc104991130)

[6. COMPONENT DESIGN 32](#_Toc104991131)

[7. HUMAN INTERFACE DESIGN 33](#_Toc104991132)

[7.1 Overview of User Interface 33](#_Toc104991133)

[7.2 Screen Images 35](#_Toc104991134)

[8. REQUIREMENTS TRACEABILITY MATRIX 47](#_Toc104991135)

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Primary Authors** | Date | Reason For Changes | Version |
| Silvia Casaburi, Makoto Emura, Wenhuan Tan, and Colin Zhou | 11/16/2021 | This is the initial version | 0.1 |
| Makoto Emura | 1/18/2022 | Add StarRating column to Reviews table | 0.2 |
| Makoto Emura | 1/24/2022 | Add ReplyTo column to Reviews table; change ID types to char | 0.3 |
| Makoto Emura  Colin Zhou | 1/31/2022 | Add mockup for review reply  Vendor’s application will not include multiple applications, just one step. | 0.4 |
| Silvia Casaburi | 1/31/2022 | Add Favorite Table and to the Data Dictionary and update EER Model | 0.5 |
| Makoto Emura | 2/13/2022 | Add description for sign-in with Google flow | 0.6 |
| Makoto Emura | 2/23/2022 | Change Photo ID type | 0.7 |
| Makoto Emura | 2/25/2022 | Add Stars schema | 0.8 |
| Makoto Emura | 3/6/2022 | Revise architecture diagram | 0.9 |
| Makoto Emura | 4/13/2022 | Add screenshot for New Reviews page | 0.10 |
| Makoto Emura | 4/18/2022 | Add LastReviewSeen column | 0.11 |
| Makoto Emura | 4/20/2022 | Add discount exchange scenario diagram | 0.12 |
| Makoto Emra | 5/12/2022 | Add Location Service dependency | 0.13 |
| Makoto Emura | 5/15/2022 | Add LastLocationUpdate column | 0.14 |
| Makoto Emura | 5/22/2022 | Update data dictionary to match source code | 0.15 |
| Silvia Casaburi | 5/30/2022 | Updated Graphs | 0.16 |

# INTRODUCTION

## 2.1 Purpose

This document aims to present a detailed description of the designs of the StreetFoodLove application. The developers also intend to use this document for designs and guidelines to implement the project. This document is also for the team’s instructor, Sara Farag, Ph.D., as it fulfills one of the requirements of the capstone class. Also, this document could be used by designers who try to upgrade or modify the existing design of the system.

## 2.2 Scope

This document contains a detailed description of the StreetFoodLove application. It specifies the software system architecture, design of the user interfaces, and database. The basic architecture of StreetFoodLove is an Amazon EC2 machine and a Go-based backend with a TypeScript-based web application front-end for clients. The database is planned to be a relational database from MySQL, paired to a TypeScript web application that utilizes HTML, CSS, and React to implement UI systems. Planned features include, but are not limited to, the ability to allow street food vendors to build their vendor page, allow customers to leave reviews and rate the food vendors, display the street food vendors on the map, allow users to search/filter/sort street food vendors and keep track of the location of mobile food vendors.

## 2.3 Overview

This document provides a system overview diagram, a layer architecture diagram, a class diagram, the authentication flow diagram, a diagram to show how to get a vendor page and its pictures, a diagram to present how to post a review with pictures, the overview of the user interface, and screen images. Coding examples are also provided to give us a brief description of how to get and modify resources in the API documentation and how to use React Leaflet library to interact with maps. In addition, we also talk about 4 design rationales in this document: Web App, 3-tier architecture, Rest architecture, and JSON Web Token. Meanwhile, a data dictionary that provides a list of data components and examples is provided in this document. In a word, this document provides an overview of our system design, user interface, and data design for the StreetFoodLove application.

## 2.4 Reference Material

* AWS SDK for JavaScript Doc - <https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/>
* AWS SDK for Go Doc - <https://docs.aws.amazon.com/sdk-for-go/api/>
* React Leaflet - <https://react-leaflet.js.org/docs/>
* “Requesting temporary security credentials” from AWS Identity and Access Management - <https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_temp_request.html>
* Designing Better Online Review Systems - [Harvard Business Review](https://www.google.com/search?q=video+la+migliore+pizza+di+napoli+con+i+bocconcini+scattata+fuori+vista+di+napoli&tbm=isch&ved=2ahUKEwi58artjZL0AhVWFzQIHYmsDqEQ2-cCegQIABAA&oq=video+la+migliore+pizza+di+napoli+con+i+bocconcini+scattata+fuori+vista+di+napoli&gs_lcp=CgNpbWcQAzoHCCMQ7wMQJ1CnGljXImC3JGgAcAB4AIABWYgBrwOSAQE3mAEAoAEBqgELZ3dzLXdpei1pbWfAAQE&sclient=img&ei=yP2NYbntFtau0PEPidm6iAo&bih=625&biw=1366)
* How to Make Sense of SQL Server Geography Data Type - <https://codingsight.com/how-to-make-sense-of-sql-server-geography-data-type/>
* Spatial Indexes Overview - <https://docs.microsoft.com/en-us/sql/relational-databases/spatial/spatial-indexes-overview?view=sql-server-ver15>
* UUID - <https://en.wikipedia.org/wiki/Universally_unique_identifier>
* OpenAPI documentation - <https://oai.github.io/Documentation/>
* AWS documentation - <https://docs.aws.amazon.com/>
* Redux Fundamentals, Part 2: Concepts and Data Flow - <https://redux.js.org/tutorials/fundamentals/part-2-concepts-data-flow\>

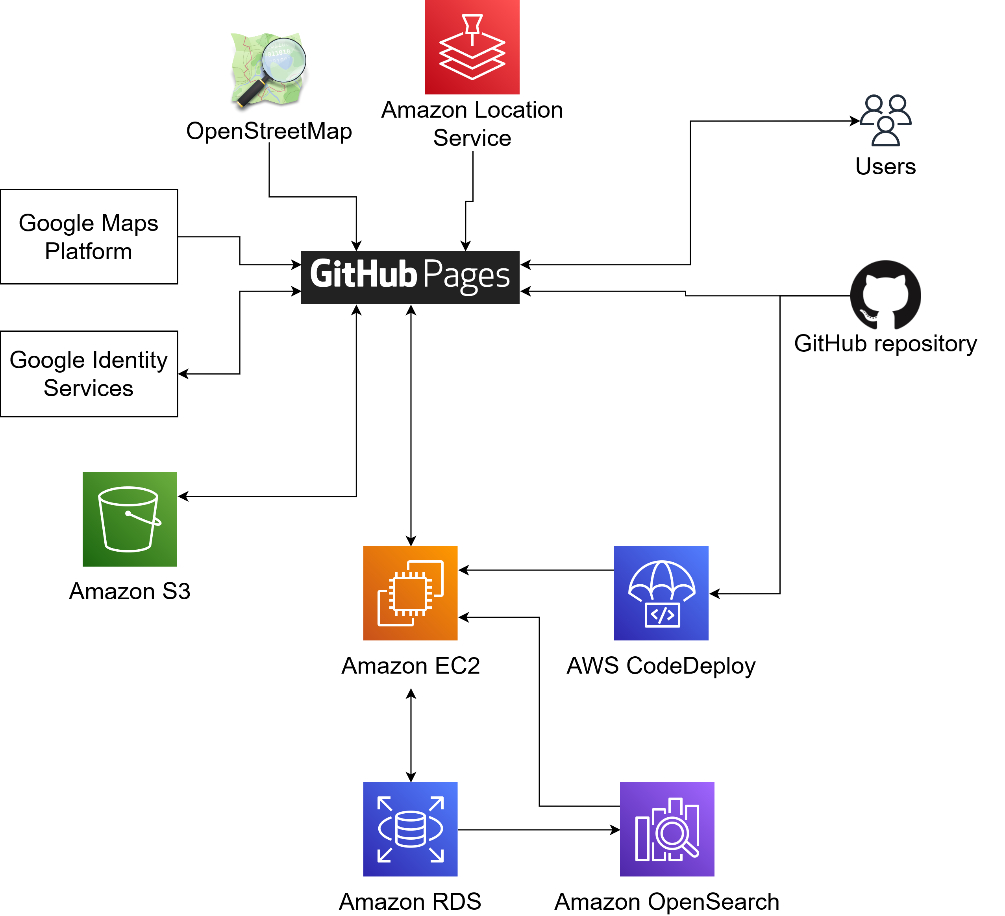
## 2.5 Definitions and Acronyms

* API (application programming interface): An interface that two software components use to communicate
* AWS (Amazon Web Services): The cloud computing platform that StreetFoodLove will use.
* Frontend, backend: The client and server, respectively, in the client-server model.
* Geocoding: The process of converting addresses into geo-coordinates. Reverse geocoding converts coordinates into addresses.
* JWT (JSON Web Token): A digital signing standard that is typically used for user authentication. Defined in RFC 7519.
* MVC (model-view-controller): A design pattern for user interfaces.
* MVP (minimum viable product): The usable product with the minimum number of features
* REST (representational state transfer): An architectural style for client-server APIs.
* SPA (single page application): A web app consisting of one HTML file that mimics a static website with multiple pages
* User: A user in Street Food Love may be further classified into a customer or a vendor.
* UUID (universally unique identifier): A standardized ID that has a property of uniqueness across all resources at any point in time.
* XSS (cross-site scripting): A code injection vulnerability that can affect web apps.

# SYSTEM OVERVIEW

StreetFoodLove is a web app that has a 3-tier architecture (frontend, backend, database). The TypeScript-based SPA will use the React framework and will be hosted on GitHub Pages. The Go-based backend will be hosted on Amazon EC2. Various other AWS and Google services will be used for object storage and database management. The MySQL database will be hosted on Amazon RDS.

This diagram shows how various services will interact with each other.

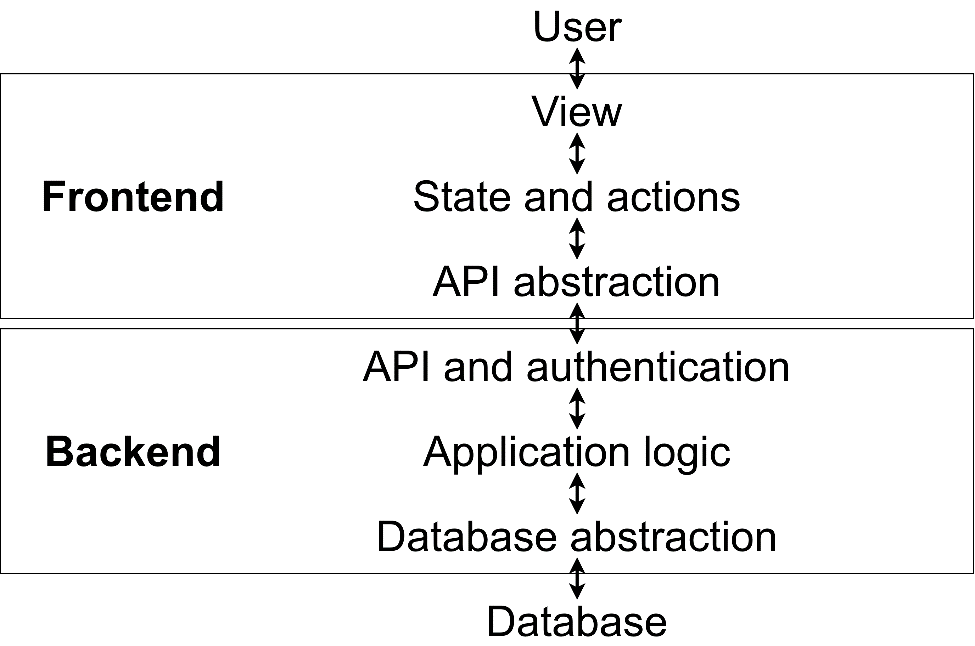


* **GitHub** will be used to store the source code and run CI/CD workflows
* On deployment, the client app will be compiled and copied to the **GitHub Pages** static host
* The server code will be compiled and deployed to an **EC2** instance using **CodeDeploy**
* The client interacts with the backend API
* The client requests map data from **OpenStreetMap** and **Google Maps Platform**
* The client can get geocoding services from Amazon Location Service
* The client uses **Google Identity Services** for the sign-in with Google authentication flow
* The client can view and upload images to an **S3** bucket
* The backend will store content data on an **RDS** database

# SYSTEM ARCHITECTURE

## 4.1 Architectural Design

This diagram illustrates the layered architecture for our app.

****

|  |  |
| --- | --- |
| **Layer** | **Responsibility of layer** |
| View | * The view component of the MVC pattern * Shows a graphical interface to represent the application state * User input is passed down to the layer below as an action |
| State and actions | * The state is the model component of the MVC * Actions, which are the controller, interact with the state and API |
| API abstraction | * Hides the API details |
| API and authentication | * Presents a standardized interface for the frontend to communicate with the backend * Handles authentication |
| Application logic | * Makes changes to content data based on API calls * Enforces access control |
| Database abstraction | * Contains all SQL queries and hides the database details |
| Database | * Stores content data and processes queries |

## 4.2 Decomposition Description

### API documentation

The API for the StreetFoodLove backend is accessed through HTTP. The API documentation can be viewed at <https://app.swaggerhub.com/apis-docs/foodapp/FoodApp/0.0.1>.

### Getting and modifying resources

To read a resource without modifying it, the GET method is used. For example, to get information about a user, the client calls GET /users/userID (In this section, “\_\_\_\_ID” is used as a placeholder for all IDs). The API returns a User object, and an example is shown below.

{

"ID": "userID",

"Username": "AlfredoLinguini",

"Picture": "pictureID",

"UserType": "user"

}

When the client wants to modify a resource, PUT or POST is used. If Alfredo wishes to change his username, he sends a PUT /users/userID/protected request with the following request body.

{

"ID": "userID",

"Username": "Chef Linguini",

"Picture": "pictureID",

"UserType": "user",

"Email": "icancook@gusteaus.com",

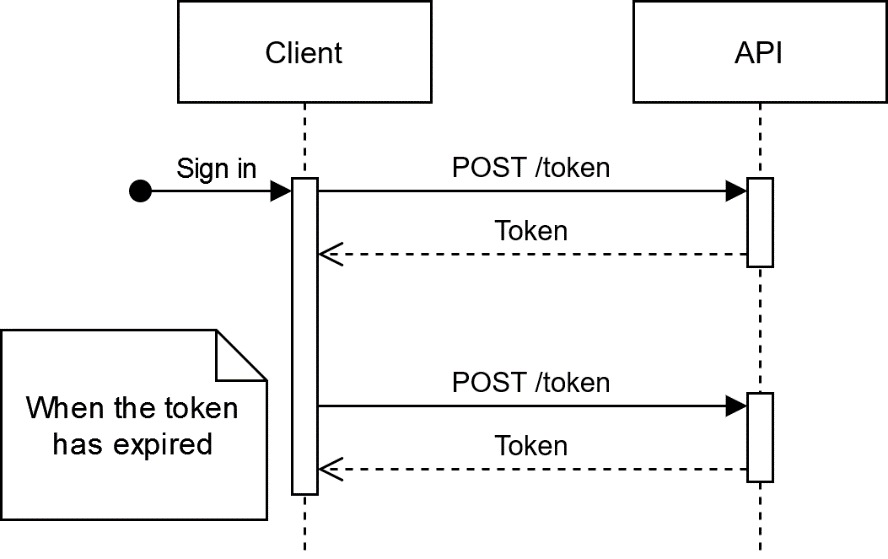
"FirstName": "Alfredo",

"LastName": "Linguini",

}

Modifying resources requires an authentication token, and this is included in the Authorization header. The section below contains more details.

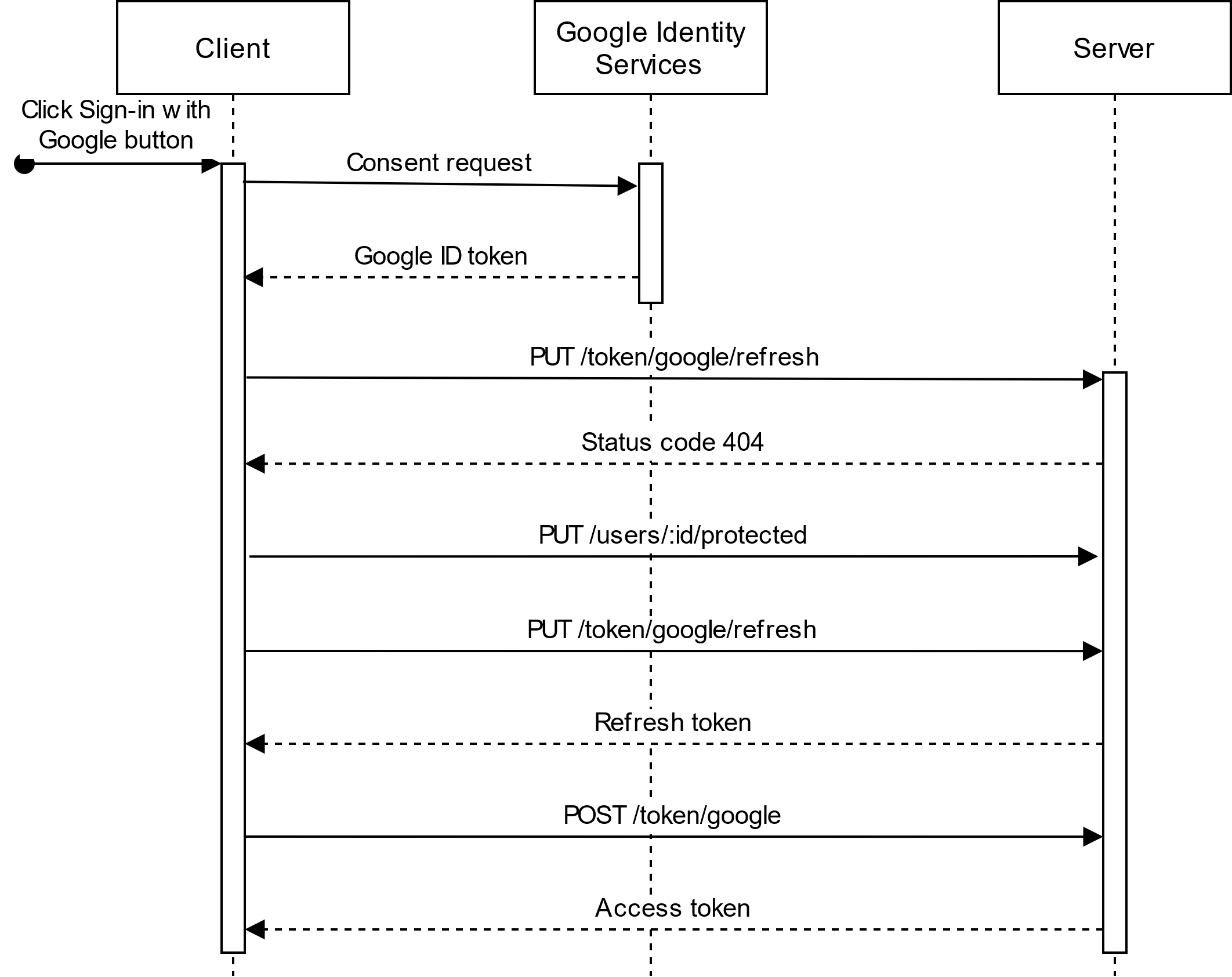
### Authentication flow



This is the sequence of steps taken if the user account was created using the create account process and not through sign-in with Google. When the user signs in by submitting their username and password, the client requests POST /token with the username and password in the request body, and an access token is returned. The client must include the token in the authorization header with the Bearer scheme whenever it wants to access a password-protected resource.

The token is a JWT which includes the user ID and expiration time. It expires in 10 minutes, after which the client makes another request to POST /token to get a new token. Because this request needs the user password, the client should store the password from the first request.

### Sign-in with Google flow



SSO with Google is enabled through their sign-in with Google feature. After the OAuth exchange happens, Google generates a Google ID token which contains an ID that identifies the Google Account and is used to associate an account to StreetFoodLove to a Google Account. From the first sign in, the sign-in with Google flow follows the following procedure, which is shown in the sequence diagram above.

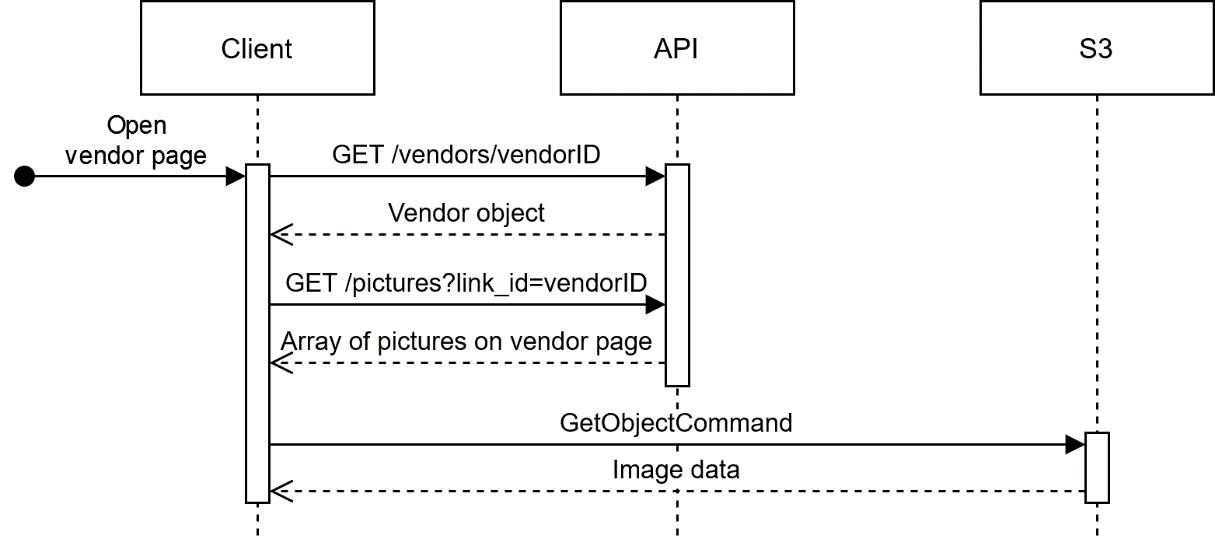
1. Client clicks the sign-in with Google button which opens the consent screen and completes the OAuth process.
2. Client gets Google ID token which contains their Google ID, email address, and name.
3. Client tries to get a refresh token with POST /token/google/refresh with the Google ID token in the body. It responds with a 404 since user does not exist.
4. Client creates account with PUT /users/:id/protected, sending the Google ID, email, name, and generated username and password.
5. Client tries PUT /token/google/refresh again, and this time it succeeds as their account exists. It returns a refresh token that contains the Google ID. The refresh token is stored in localStorage.
6. Client passes the refresh token to POST /token/google, which returns an access token. The access token is used to access resources.

The next time the user opens the app, the client uses the stored refresh token to get an access token. The client sends the refresh token to POST /token/google, to which the server responds with an access token.

The following are the steps for signing out and signing back in.

1. On sign out, the access token and refresh token are removed from localStorage.
2. To sign in, the user clicks on sign-in with Google button, and the client receives the Google ID token.
3. The client sends the Google ID token to PUT /token/google/refresh to get a refresh token
4. Client passes the refresh token to POST /token/google to get an access token.

### How to get a page and its pictures

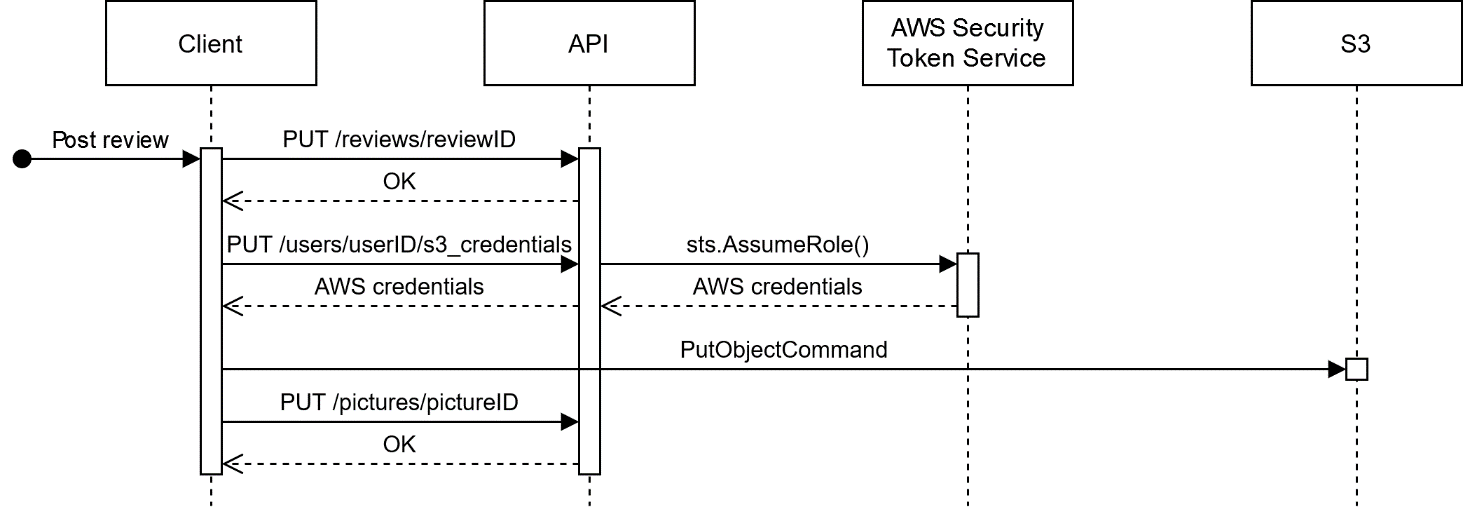


All pictures are stored on an Amazon S3 bucket. Many pages have multiple pictures in their content, such as vendor pages and guides. To explain how pictures are retrieved, this example uses a vendor page.

When the user opens a vendor page, the client requests GET /vendors/vendorID and the API returns a Vendor object which contains information about the vendor but does not include the pictures. To get the pictures for the vendor page, the client requests GET /pictures?link-id=vendorID. The API returns an array of all photos that are associated with the given vendor ID.

These picture IDs correspond with their object names in the S3 bucket. The client iterates through each picture ID to get the image files from S3 to display on the page.

### How to post a review with pictures



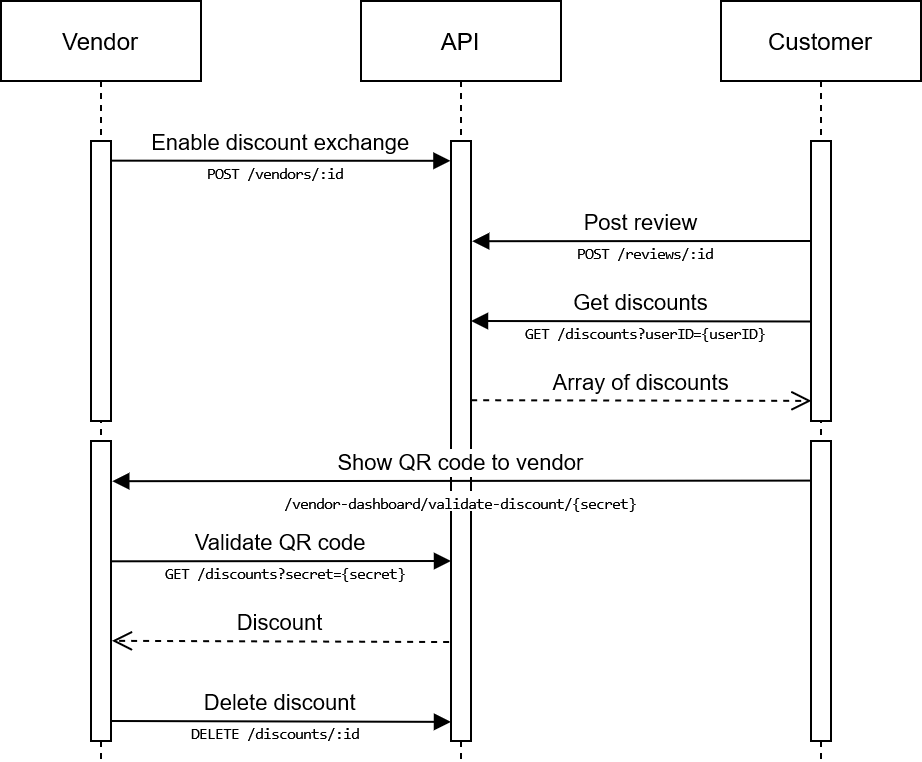
Some features consist of adding pictures to content, like posting a review that contains pictures.

When the user posts a review, the client makes a PUT /reviews/reviewID call with all necessary information about the review. The ID must be a UUIDv4.

The client prepares to upload pictures to S3 by requesting credentials with write access to the S3 bucket. This is done by calling PUT /users/userID/s3\_credentials. The backend has an IAM role to generate temporary credentials from AWS Security Token Service, and these credentials are forwarded to the client. The client uses the credentials to issue PutObjectCommand to S3 for each review picture. Objects in S3 must have a key in the format <UUIDv4>.<file extension>, for example, c093d5a4-6780-41c6-9a86-9cd55bdc131e.jpg.

After all the pictures are uploaded, the client notifies the backend to attach the pictures to the review by calling PUT /pictures/pictureID with the object key as the picture ID and "reviewID" in the LinkID field. This is done for all pictures to link to the review.

### Discount exchange scenario



This sequence diagram visualizes the steps for distributing and claiming discounts. The first half involves the vendor enabling discount exchange and a customer posting a review. When a review is posted, a discount record is inserted. In the second half, the QR code containing a URL to validate the discount is scanned by the vendor. The discount record is deleted so that it cannot be used again.

Note that scanning the QR code using a generic reader is a security vulnerability since the URL in the QR code cannot be trusted. In a later version, the app should contain a QR code reader so that the QR code cannot direct the vendor to a malicious site.

### Leaflet map overlays

The Leaflet library, for interactive maps, provides a variety of methods to add overlay elements on the map. For our app, we need to be able to add markers and popups with content. The examples below show how this is done. As we are using the React Leaflet library, the syntax is in JSX.

Maps are created with

<MapContainer>

<TileLayer url="[tile provider URL]" />

</MapContainer>

To add a marker, you simply add <Marker position={[0.0, 0.0]} /> inside the MapContainer.

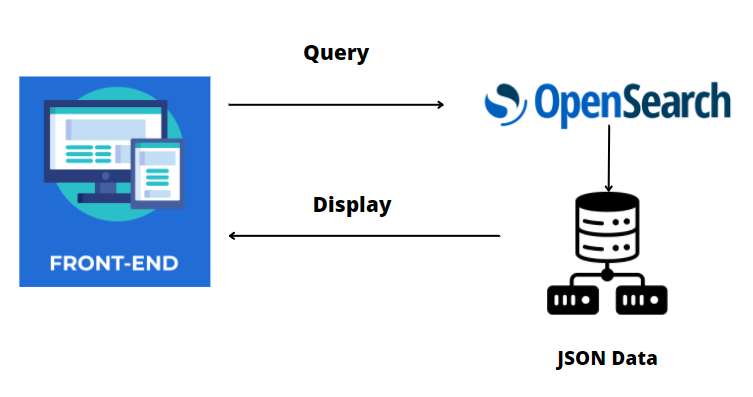
To create a popup with content, the Popup component can be used, and child components can be added to it.

<Popup>

<p>Content here</p>

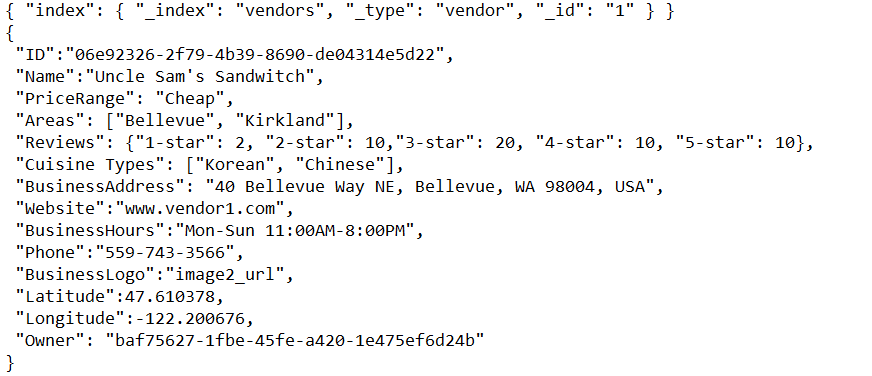
</Popup>

### AWS OpenSearch



AWS OpenSearch service is used to provide a searching function for our App. Steps to implement AWS OpenSearch service are as follows:

* Create a domain in AWS OpenSearch service
* Upload JSON format data to AWS OpenSearch



(Sample Data)

* Add queries to OpenSearch Console

1. Search for anything matches the query input in frontend

GET \_search

{

"query": {

"match\_all": {}

}

}

2. Search for certain areas

GET \_search

{

"query": {

"multi\_match": {

"query": "Bellevue",

"fields": ["Areas", "Name"]

}

}

}

3. Search for certain cuisine types

GET \_search

{

"query": {

"multi\_match": {

"query": "Korean",

"fields": ["Cuisine Types"]

}

}

}

4. Search by price range

GET \_search

{

"query": {

"multi\_match": {

"query": "Cheap",

"fields": ["PriceRange"]

}

}

}

* Implement query handler function in the frontend to link to AWS OpenSearch, send queries from Users to the AWS OpenSearch dataset, and pull out of data to display in the frontend.
* Filtering:

Two filters: Cuisine Type and PriceRange

1. Sample Query for filtering by Cuisine Type

GET \_search

{

"query": {

"bool": {

"must": [

{

"match": {

"text\_entry": "Bellevue"

}

}

],

"should": [

{

"match": {

"text\_entry": "Korean"

}

}

],

"minimum\_should\_match": 1,

"must\_not": [

{

"match": {

"PriceRange": "Gourmet"

}

],

"filter": {

"term": {

"Cuisine Type": "Korean"

}

}

}

}

2. Sample Query for filtering by PriceRange

GET \_search

{

"query": {

"bool": {

"must": [

{

"match": {

"Name": "Tasty"

}

},

{

"multi\_match": {

"query": "Bellevue",

"fields": ["Areas"]

}

}

],

"filter": {

"term": {

"PriceRange": "Cheap"

}

}

}

}

}

* Sorting

1. Sample Query

GET \_search

{

"query": {

"match": {

"Areas": "Redmond"

}

},

"sort": [

{

"Reviews.5-star": {

"order": "desc"

}

}

]

}

**Search History**

* The most recent vendor will be listed at the top
* Up to 10 vendors are listed in the search history

## 4.3 Design Rationale

**Web app**: We choose to make a browser-based web app instead of a mobile app for a few reasons. Two team members already have experience in writing web apps with JavaScript. Additionally, we believe it is a lot easier to deploy a web app with GitHub Pages than it is to bundle, sign, and upload to Google Play.

**3-tier architecture**: Web services commonly use the 3-tier architecture because it includes the client, server, and database which are needed to run a web app. We took the 3-tier architecture and subdivided the presentation and logic layers into smaller responsibilities. By dividing responsibilities, it is easier to subdivide complex features and isolate bugs.

**REST architecture**: We choose the REST architecture as it is commonly used in other web APIs, which means that we can look at other APIs as examples. The JSON format allows us to easily read requests and responses, and manually write requests for testing.

We did consider more complex protocols like GraphQL and Protobuf which can be advantageous for enterprise solutions. However, we wanted something very easy to learn and use so that all team members would be able to understand how everything works.

**JWT**: The JWT format is also commonly used in web apps. There is a simple-to-use Go library for generating JWT tokens, and it satisfies all of our authentication and access control needs.

In our very basic authentication flow, the password will be stored in localStorage, making it a potential target for XSS attacks. We are using React which greatly reduces this risk. As an additional measure, we may implement refresh tokens or use an OAuth service in the future. For now, we wanted to choose the simplest option for the MVP.

# DATA DESIGN

## 5.1 Data Description

The data is stored in a relational database using MySQL and Amazon RDS data are stored in tables and its fields, from and to the database, are listed in the data dictionary list. Photos will be stored in Amazon S3 and can be retrieved with LinkID from the Photo table.

## 5.2 Data Dictionary and EER Model

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Data Format** | | **Field Size** | | **Description** | **Example** |
| **Guide** | | | | | | | |
| ID | Char | UUID | 36 | | ID, PK | | 123e4567-e89b-12d3-a456-426614174000 |
| Title | text | Alpha-numeric | 500 | | Title of guide | | This is the Title |
| Guide | text | Alpha-numeric | 5000 | | Article text | | This is the article text |
| DatePosted | datetime | MM/DD/YYYY HH:MM:SS |  | | Date of the article post | | 11/11/2021 00:00:00 |
| ArticleAuthor | varchar | Alpha-numeric | 500 | | Person that wrote the article | | Jane Doe |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Data Format** | | **Field Size** | | **Description** | **Example** |
| **Link** | | | | | | | |
| ID | Char | UUID | 36 | | ID, PK | | 123e4567-e89b-12d3-a456-426614174000 |
| Title | Varchar | Alpha-numeric | 45 | | Link Title | | How to create a Business |
| URL | Varchar | Alpha-numeric | 255 | | URL | | www.url.com |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Data Format** | | **Field Size** | | **Description** | **Example** |
| **Photos** | | | | | | | |
| ID | varchar | UUID+extention | 50 | | ID, PK  S3 object key for image data | | 123e4567-e89b-12d3-a456-426614174000.jpg |
| DatePosted | datetime | MM/DD/YYYY HH:MM:SS |  | | Date the photo was posted | | 11/11/2021 00:00:00 |
| Text | text | Alpha-numeric | 500 | | Field for text to describe Photo | | This is what real Pizza looks like, Yum! |
| LinkID | Char | UUID | 36 | | Foreign key to review or vendor | | 123e4567-e89b-12d3-a456-426614174000 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Data Format** | **Field Size** | **Description** | **Example** |
| **Reviews** | | | | | |
| ID | Char | UUID | 36 | ID, PK | 123e4567-e89b-12d3-a456-426614174000 |
| Text | text | Alpha-numeric | 500 | Field for free text review | Pizzeria Margherita has the best Pizza in town! |
| VendorID | Char | UUID | 36 | References the vendor that the review was posted to | 123e4567-e89b-12d3-a456-426614174000 |
| DatePosted | datetime | MM/DD/YYYY HH:MM:SS |  | Date the Review was posted | 11-11-2021 00:00:00 |
| StarRating | integer | Numeric |  | Number of stars given to review (1 to 5) | 5 |
| ReplyTo | Char | UUID | 36 | ID of the parent review or NULL if this is a root review | 123e4567-e89b-12d3-a456-426614174000 |
| VendorFavorite | bool | 0 or 1 |  | Indicates if vendor favorited this review | 1 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Data Format** | **Field Size** | **Description** | **Example** |
| **User** | | | | | |
| ID | Char | UUID | 36 | ID, PK | 123e4567-e89b-12d3-a456-426614174000 |
| Email | varchar | Alpha-numeric | 100 | User email | [1user@StreetFoodLove.com](mailto:1user@StreetFoodLove.com) |
| Username | varchar | Alpha-numeric | 100 | Unique string identifier of user | Foody! |
| FirstName | varchar | Alpha-numeric | 100 | First name of user | Jane |
| LastName | varchar | Alpha-numeric | 100 | Last name of user | Smith |
| SignUpDate | datetime | Numeric |  | Date that this user was created | 11/10/2021 00:00:00 |
| LoginPassword | binary | Hash | 32 | SHA256 hash of password | e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855 |
| UserType | tinyint | Numeric | 1 | Type of user: customer is 0 and vendor is 1 | 0 |
| Photo | varchar | filename | 50 | S3 object for this person’s avatar | 123e4567-e89b-12d3-a456-426614174000.png |
| GoogleID | Varchar | Alpha-numeric | 50 | If this user used Sign-in with Google, this is the Google account identifier | 0123456789 |
| LastReviewSeen | Char | UUID | 36 | The last review that the vendor has seen. It is null if the vendor page has no reviews or the user is a customer. | 123e4567-e89b-12d3-a456-426614174000 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Data Format** | **Field Size** | **Description** | **Example** |
| **Vendor Table** | | | | | |
| ID | Char | UUID | 36 | ID, PK | 123e4567-e89b-12d3-a456-426614174000 |
| Name | varchar | Alpha-numeric | 100 | Business Name | Pizzeria Margherita |
| BusinessAddress | varchar | Alpha-numeric | 500 | Vendor’s Address | Pizzeria Margherita  1234 Candycane Lane  Seattle, WA 98112 |
| Website | varchar | Alpha-numeric | 500 | Business Website | example.com |
| BusinessHours | varchar | Alpha-numeric | 500 | Business Hours | 08:00 AM – 05:00 PM |
| Phone | varchar | Alpha-numeric | 50 | Phone Number | 206-654-7895 |
| BusinessLogo | varchar | filename | 50 | S3 object for vendor logo | 123e4567-e89b-12d3-a456-426614174000.png |
| Latitude | float | Numeric |  | Business geolocation latitude |  |
| Longitude | float | Numeric |  | Business geolocation longitude |  |
| LastLocationUpdate | datetime | numeric |  | When the BusinessHours, Lattitude, and Longitude fields were last updated | 2022-05-15 9:36 AM |
| Description | varchar | Alpha-numeric | 1500 | “About Us” box | Animals are friends, not food |
| SocialMediaLink | varchar | Alpha-numeric | 500 | Social media URL | twitter.com/example |
| Owner | Char | UUID | 36 | References vendor user that created this record | 123e4567-e89b-12d3-a456-426614174054 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Data Format** | **Field Size** | **Description** | **Example** |
| **Favorite** | | | | | |
| ID | Char | UUID | 36 | ID, PK | 123e4567-e89b-12d3-a456-426614174054 |
| DatePosted | datetime | numeric |  | When this record was created | 11/10/2021 00:00:00 |
| VendorID | Char | UUID | 36 | References vendor that the user favorited | 123e2867-e89b-12d3-a456-426614174054 |
| UserID | Char | UUID | 36 | References the user who created this record | 123e2867-e89b-12d3-a456-426614174459 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Data Format** | **Field Size** | **Description** | **Example** |
| **Stars** | | | | | |
| UserID | Char | UUID | 36 | User who created the star | 123e4567-e89b-12d3-a456-426614174054 |
| VendorID | Char | UUID | 36 | Vendor that the user starred | 123e4567-e89b-12d3-a456-426614174054 |

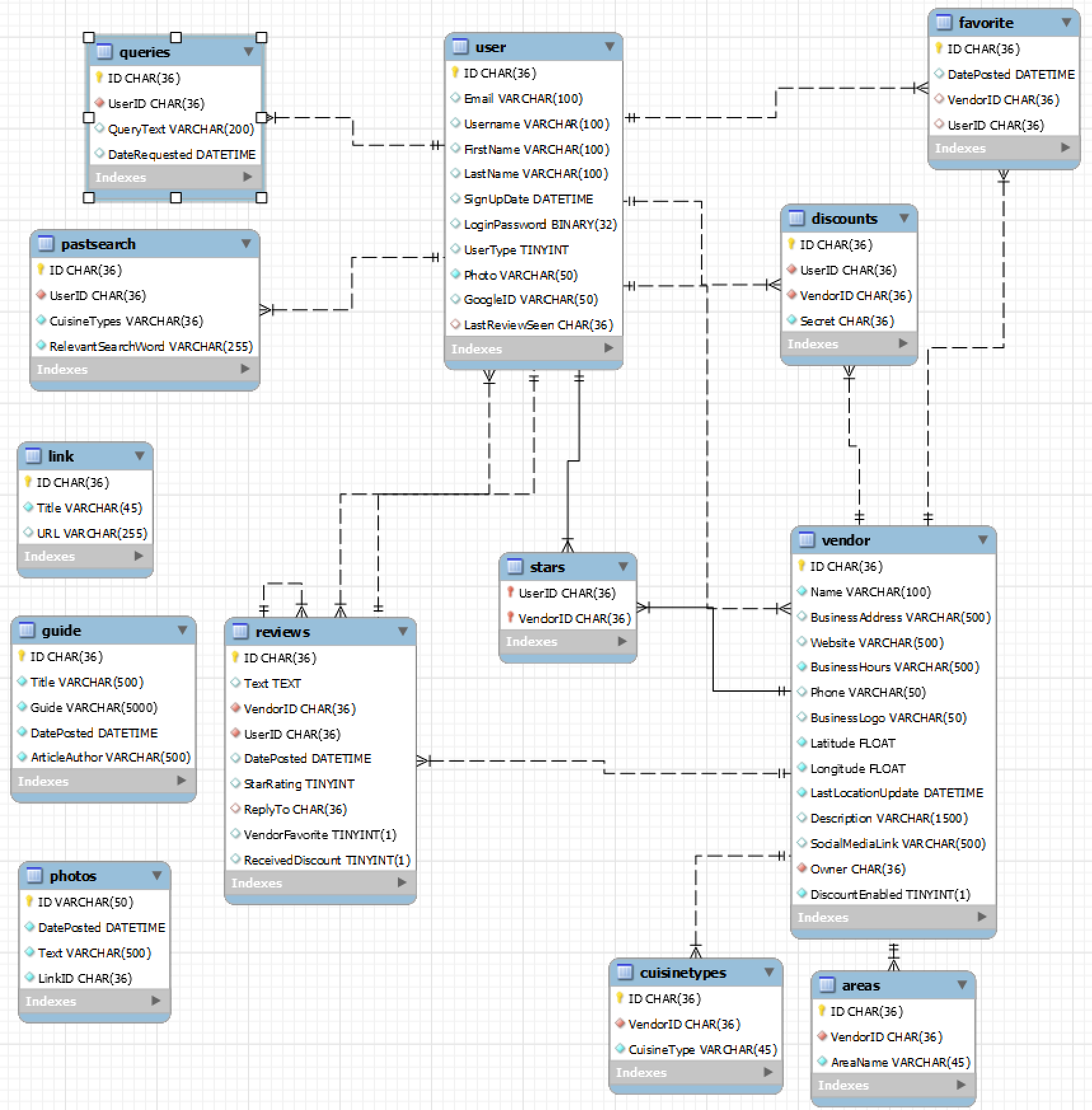
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Data Format** | **Field Size** | **Description** | **Example** |
| **Areas** | | | | | |
| ID | Char | UUID | 36 | ID, PK | 123e4567-e89b-12d3-a456-426614174054 |
| VendorID | Char | UUID | 36 | References vendor | 123e4567-e89b-12d3-a456-426614174054 |
| AreaName | Varchar | Alpha-numeric | 45 | The area that the vendor is located | Bellevue |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Data Format** | **Field Size** | **Description** | **Example** |
| **CuisineTypes** | | | | | |
| ID | Char | UUID | 36 | ID, PK | 123e4567-e89b-12d3-a456-426614174054 |
| VendorID | Char | UUID | 36 | References vendor | 123e4567-e89b-12d3-a456-426614174054 |
| CuisineType | Varchar | Alpha-numeric | 45 | Vendor cuisine type | Japanese |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Data Format** | **Field Size** | **Description** | **Example** |
| **Queries** | | | | | |
| ID | Char | UUID | 36 | ID, PK | 123e4567-e89b-12d3-a456-426614174054 |
| UserID | Char | UUID | 36 | User who made the query | 123e4567-e89b-12d3-a456-426614174054 |
| QueryText | Varchar | Alpha-numeric | 200 | Text of query | bread |
| DateRequested | datetime | numeric |  | When this record was created | 11/10/2021 00:00:00 |

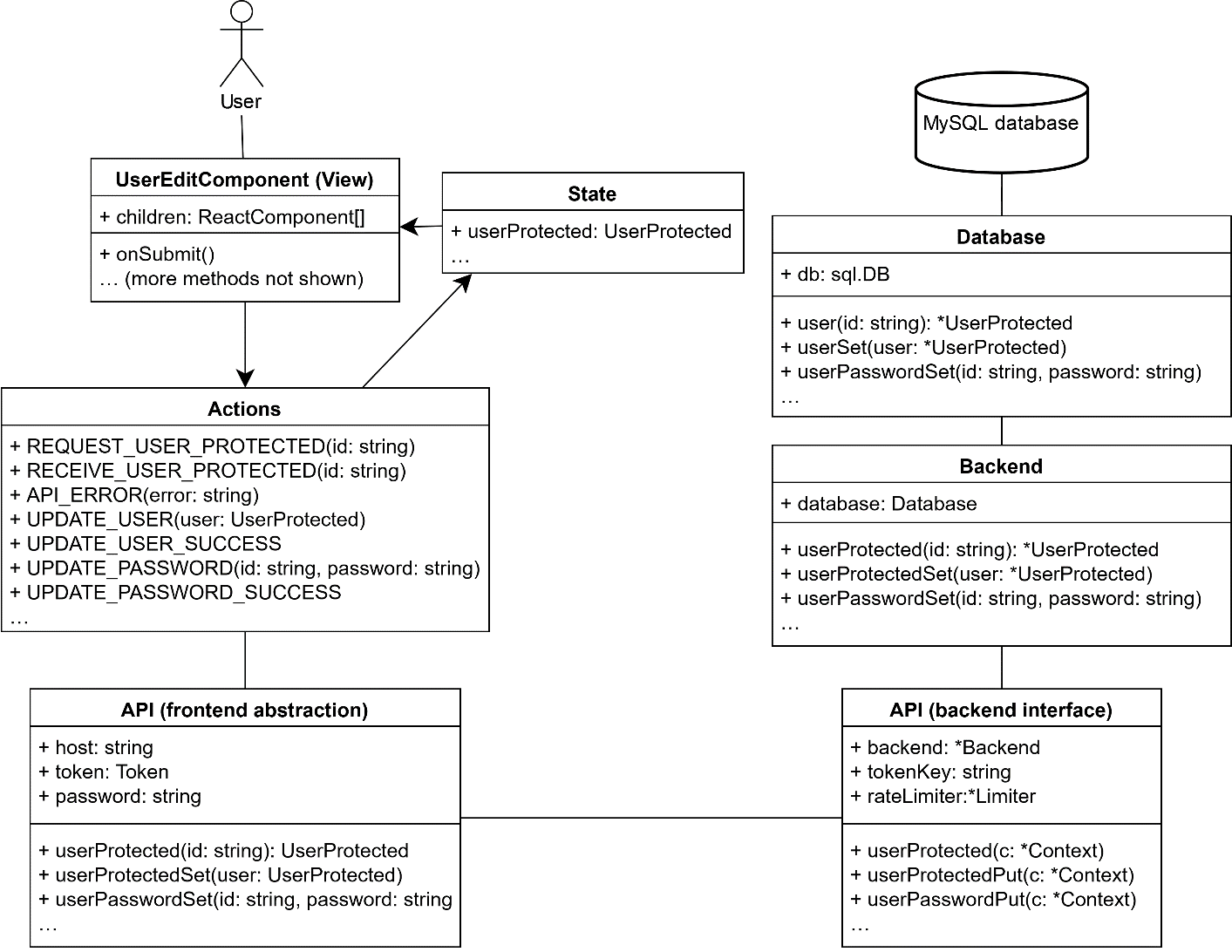
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Data Format** | **Field Size** | **Description** | **Example** |
| **PastSearch** | | | | | |
| ID | Char | UUID | 36 | ID, PK | 123e4567-e89b-12d3-a456-426614174054 |
| UserID | Char | UUID | 36 | User who made this search query | 123e4567-e89b-12d3-a456-426614174054 |
| CuisineTypes | Varchar | Alpha-numeric | 36 | The cuisine type filters | Japanese |
| RelevantSearchWord | Varchar | Alpha-numeric | 255 | Search string used in this search | string |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Data Format** | **Field Size** | **Description** | **Example** |
| **Discounts** | | | | | |
| ID | Char | UUID | 36 | ID, PK | 123e4567-e89b-12d3-a456-426614174054 |
| UserID | Char | UUID | 36 | User that this discount belongs to | 123e4567-e89b-12d3-a456-426614174054 |
| VendorID | Char | UUID | 36 | The vendor that this discount can be used on | 123e4567-e89b-12d3-a456-426614174054 |
| Secret | Char | Alpha-numeric | 36 | Secret key that is needed to claim this discount | 123e4567-e89b-12d3-a456-426614174054 |



# COMPONENT DESIGN

This class diagram gives a breakdown of the 3-tier architecture from [4.1 Architectural Design](#_4.1_Architectural_Design). Only the functions used to view and update user info are shown. The term “protected” in object names differentiates public fields from those that are password-protected.



# HUMAN INTERFACE DESIGN

## 7.1 Overview of User Interface

Once the user opens the app, they will be greeted by a big map layout, which by default, will show the food trucks near their current physical location (home). If the user decides to enter another destination, they can type in their location in the search bar provided. The searchbar will keep track of the recent search history of the user. The results from that search will populate the left sidebar of the map, as well as show on the actual map (with a detailed popup). The filter selection in the left sidebar allows the user to filter the search results accordingly. The user will be able to click on the popups on the map which take them to the specific pages of the vendors. In the left sidebar will be a toggle that when toggled, will show recommended vendors based off previous search queries.

On the page of the specific vendor, the user will be able to look at a gallery of photos that showcase the foods of the vendor. It contains contact information and other important details of the food truck. Customers will be able to leave reviews if they wish, which other customers and vendors can reply to. If a vendor likes a specific review of their business, they can “heart” that review, which will show up as a heart icon on the user side. The map option that will be displayed will take the users to Google Maps, which will give directions to the location of that specific food truck. Users can also share their favorite vendors to social media via the social media icons at the top of the specific vendor page, and will also be able to click the star icon at the top of the page, to star the vendor (this will show up in a chart).

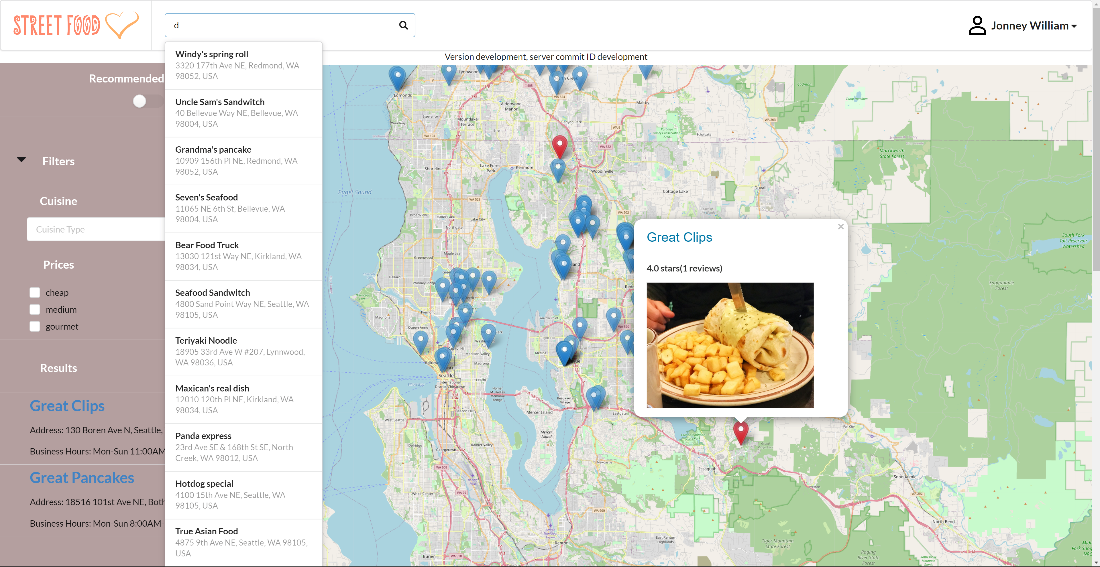
If the user wants to sign up for an account, then they can click the sign-up button on the top right-hand corner, which will direct them to the page that will make the user choose what type of account setup they want. If the user just wants to set up a customer account for general users, they can click the user option that will lead them to the specific signup form. If they want to start a vendor account, then they click the vendor option, which will lead them to the initial vendor signup form, which is very similar to the one for the typical user. The vendor's application includes details about their business.

For vendor owners, they will have access to all the same functionalities as the regular user, but they will get a personalized dashboard. The dashboard contains links to various pages. The customize page lets the vendor customize their vendor page. The trends page will show all the important metrics of their business. The New Reviews page shows new reviews that have been posted on their vendor page. The street food business guides open the business guide index. The upload photos page will direct the user to a page to upload new photos to their vendor page.

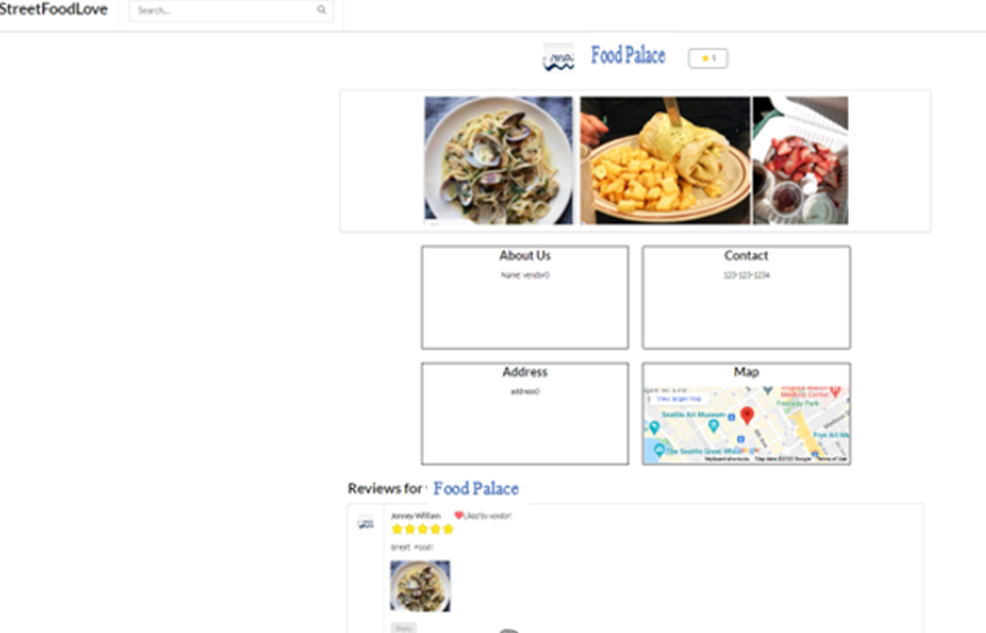
There will also be an ability for users to see which vendors are near their specific location. In the user dropdown menu on the top right of the page, there is an option called “Vendors Near You”, which will show nearby vendors based on your location.

The Profile Settings page will be visible after the user selects the “Profile Settings” option in the user dropdown menu. In the Account Settings tab, users will be able to change their registered first and last name, as well as upload a new profile picture. In the Starred Vendors, users will be able to see the vendors that they starred from the specific vendor page. In the Discounts page, users will be able to see the discounts that they received from leaving a review for a particular vendor. Discounts are enabled by vendors in the edit vendor page, through a checkbox at the very bottom of the form. This will allow both a generated QR code for users to claim on a mobile device, and also the ability for users to receive discounts when leaving a review.

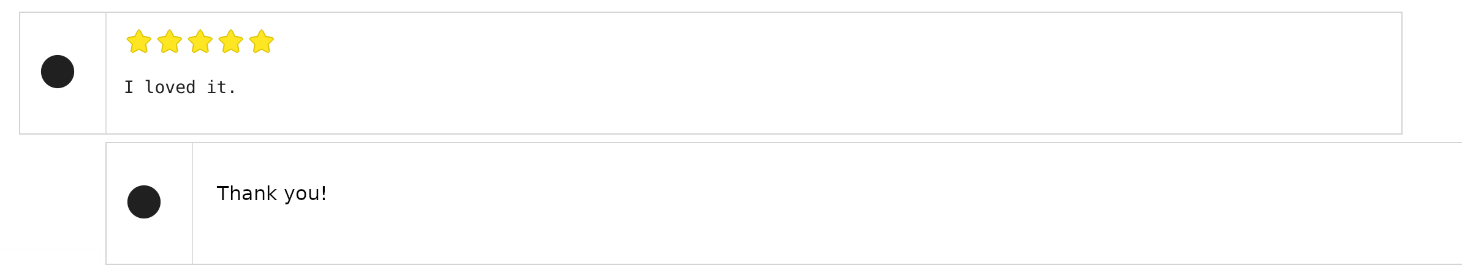
## 7.2 Screen Images



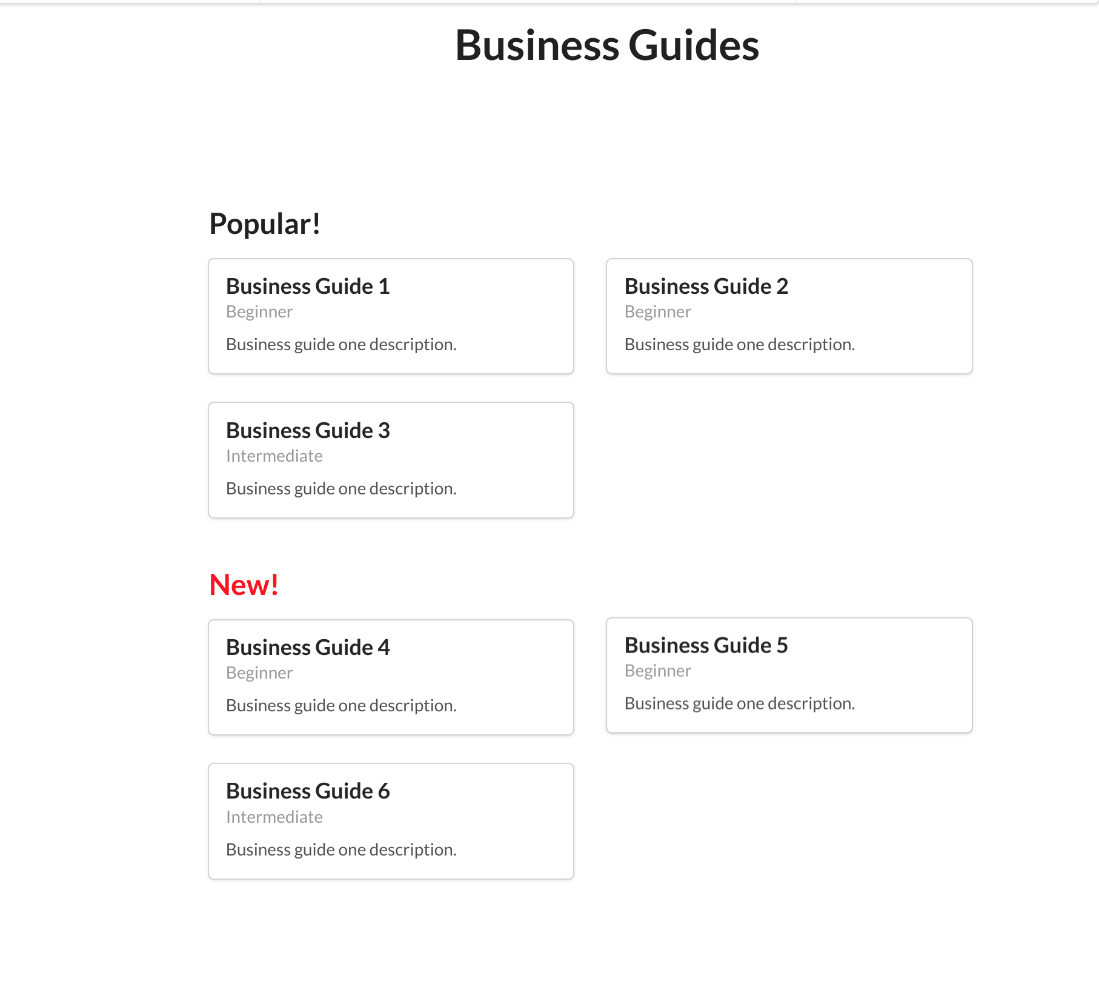
Screen Image 1



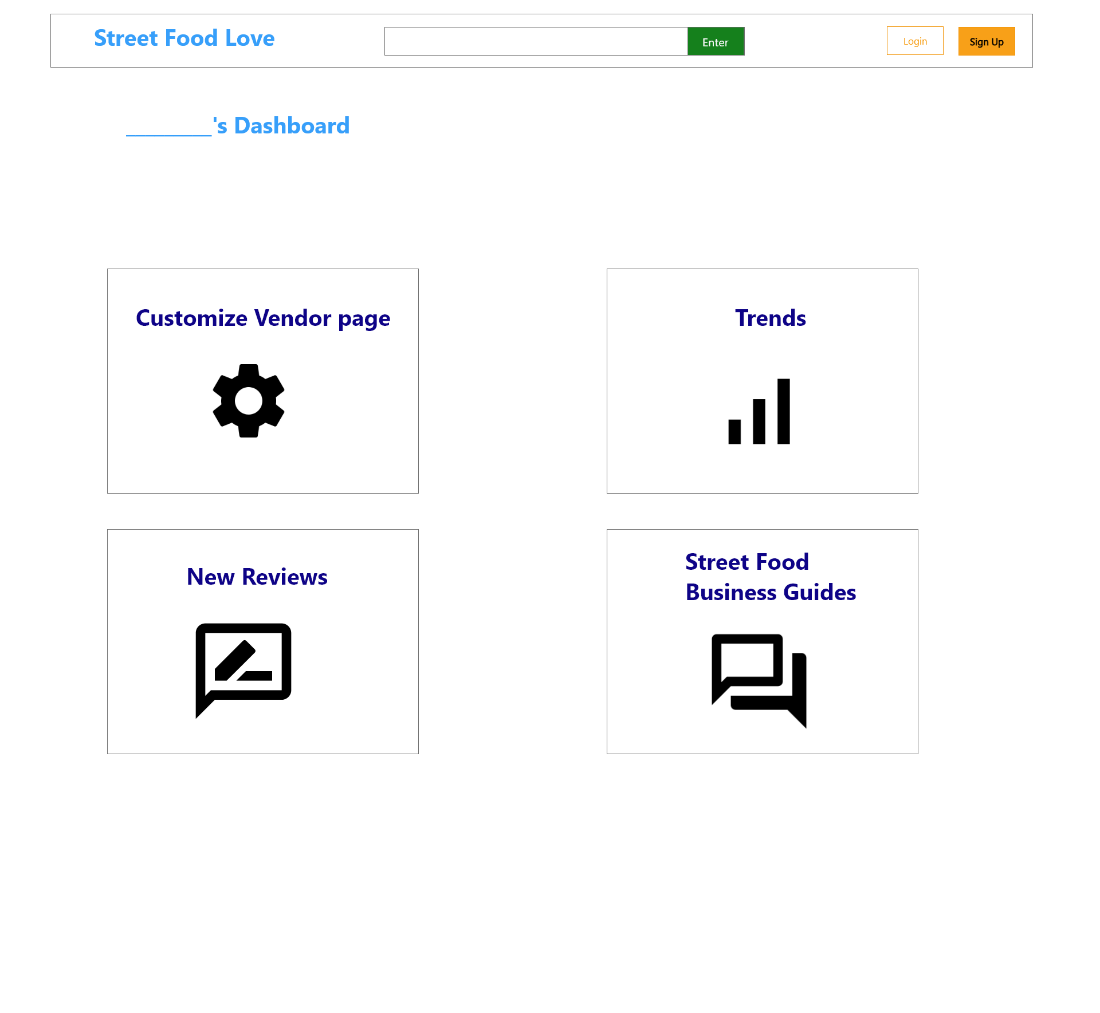
Screen Image 2



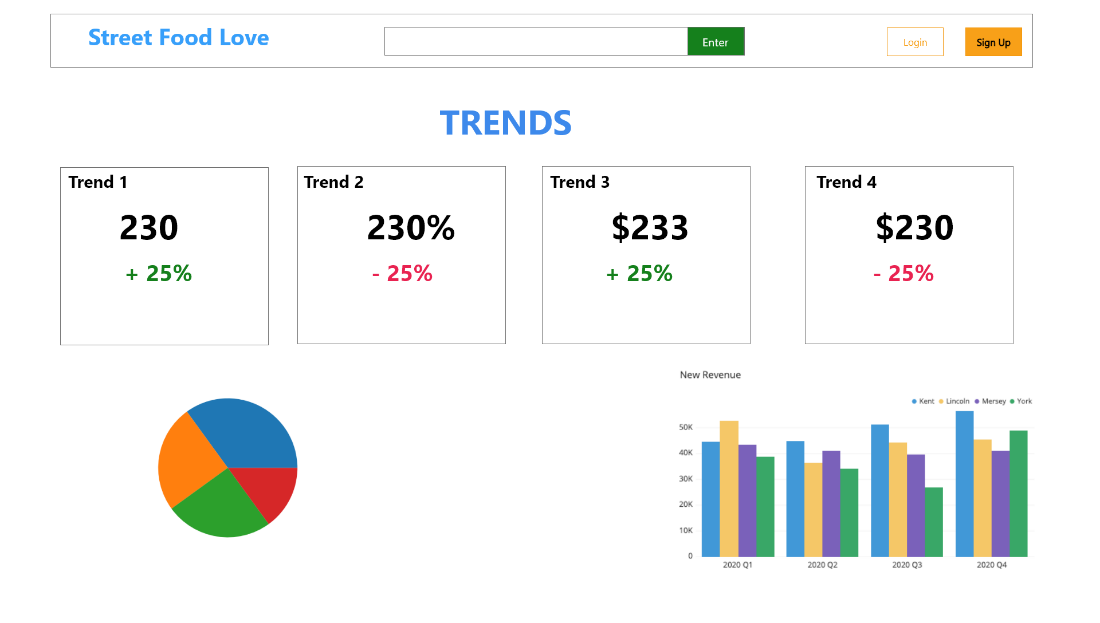
Screen Image 3

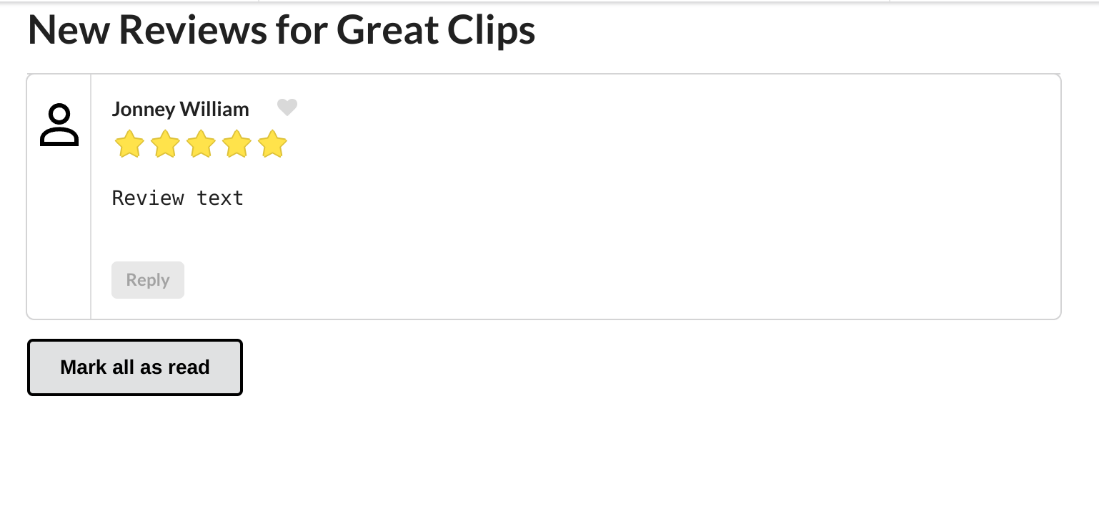


Screen Image 4

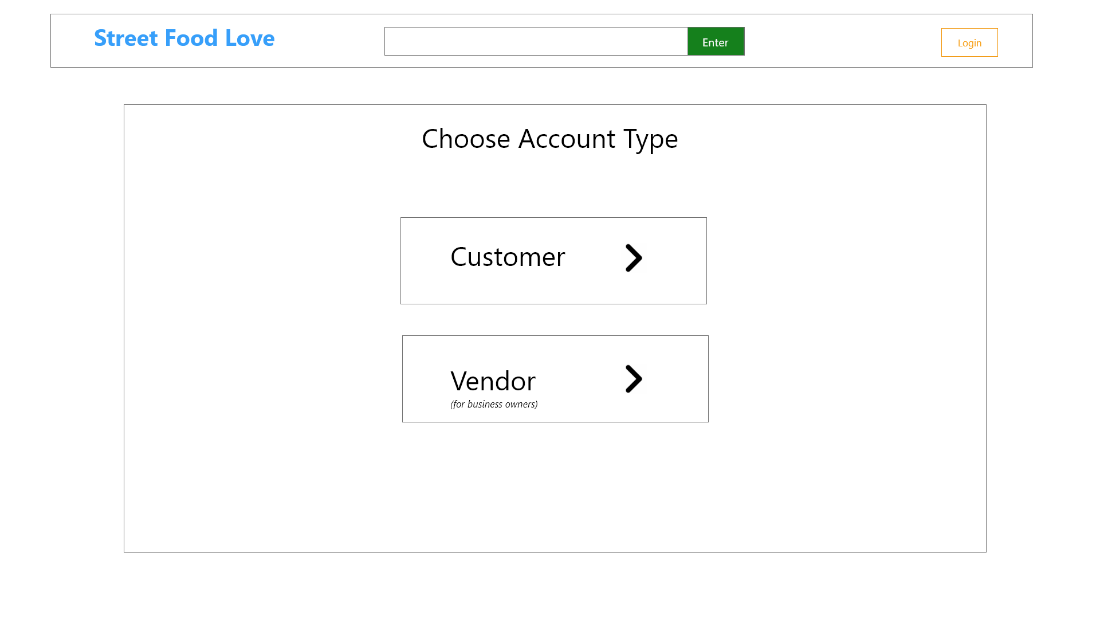


Screen Image 5

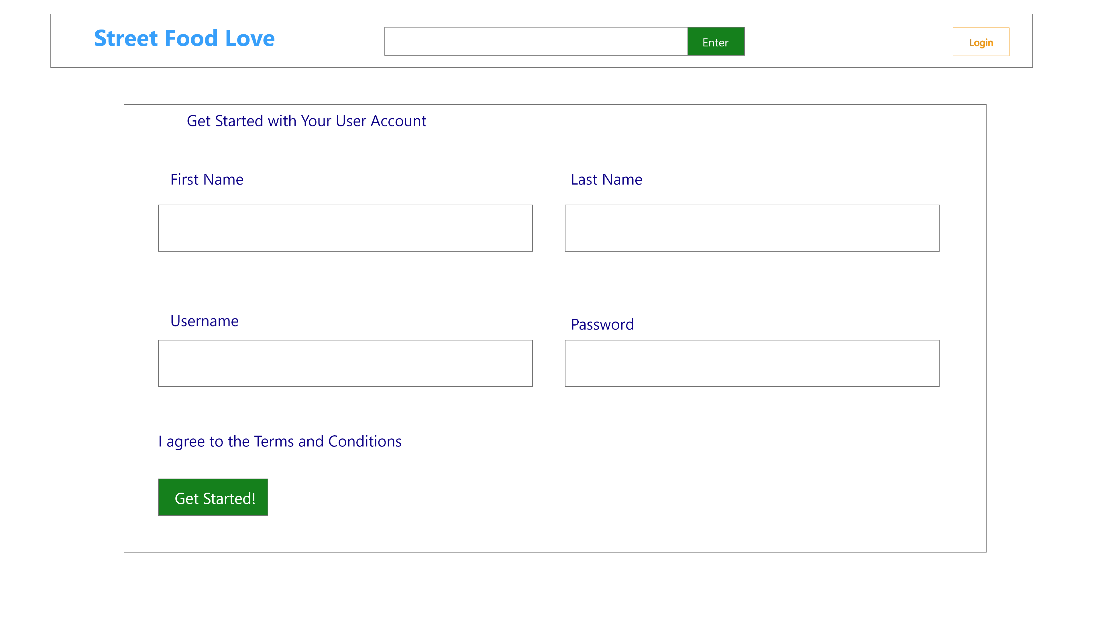




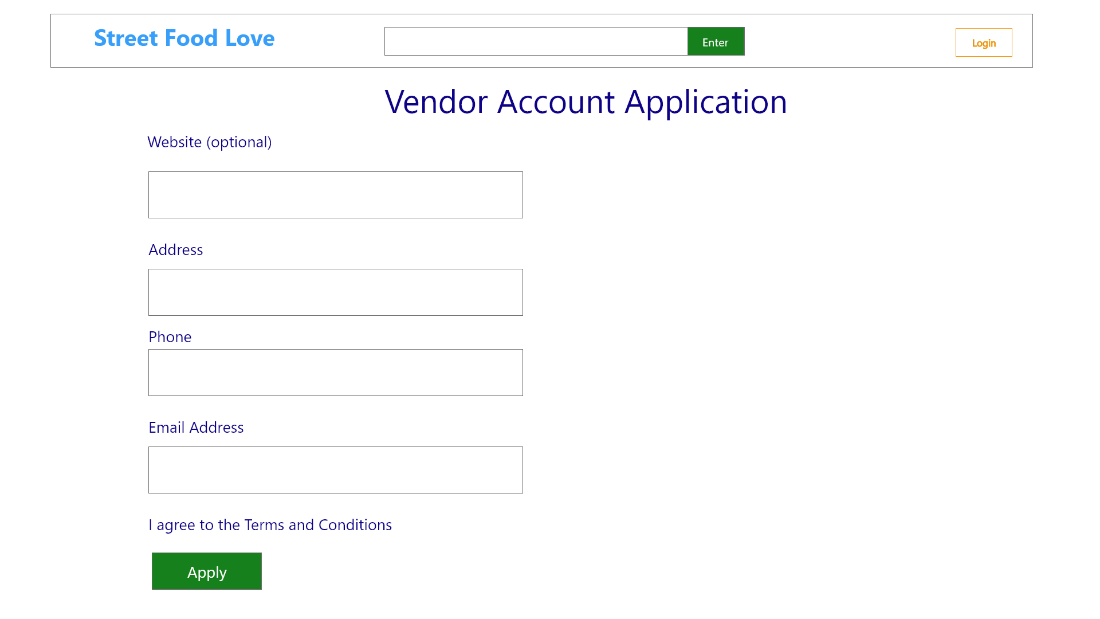
Screen Image 6



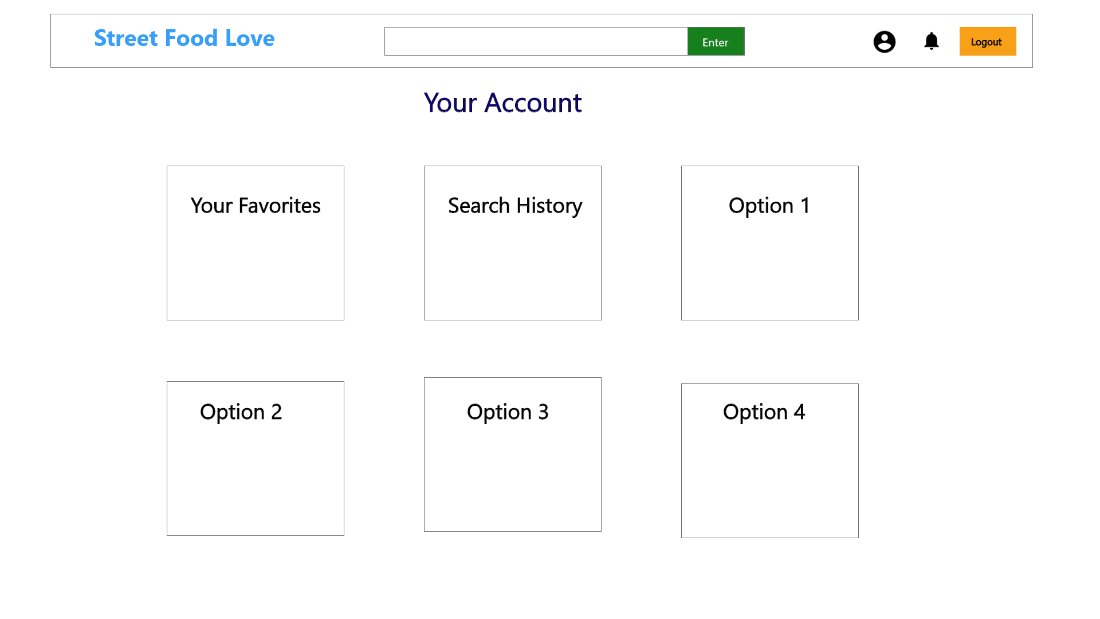
Screen Image 7



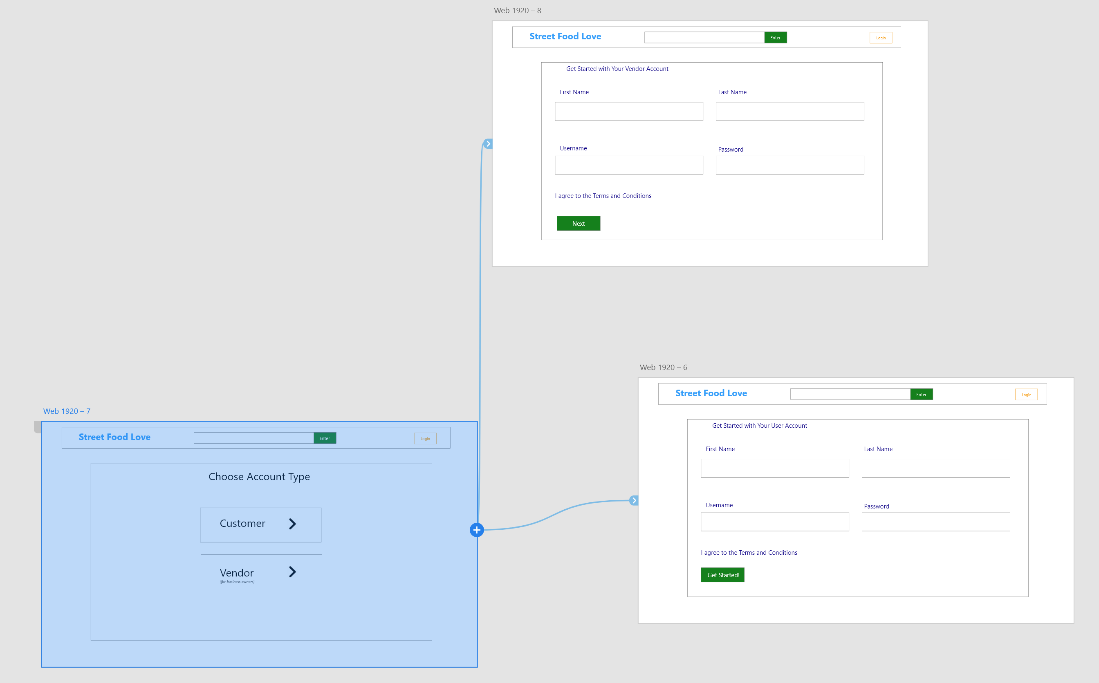
Screen Image 8

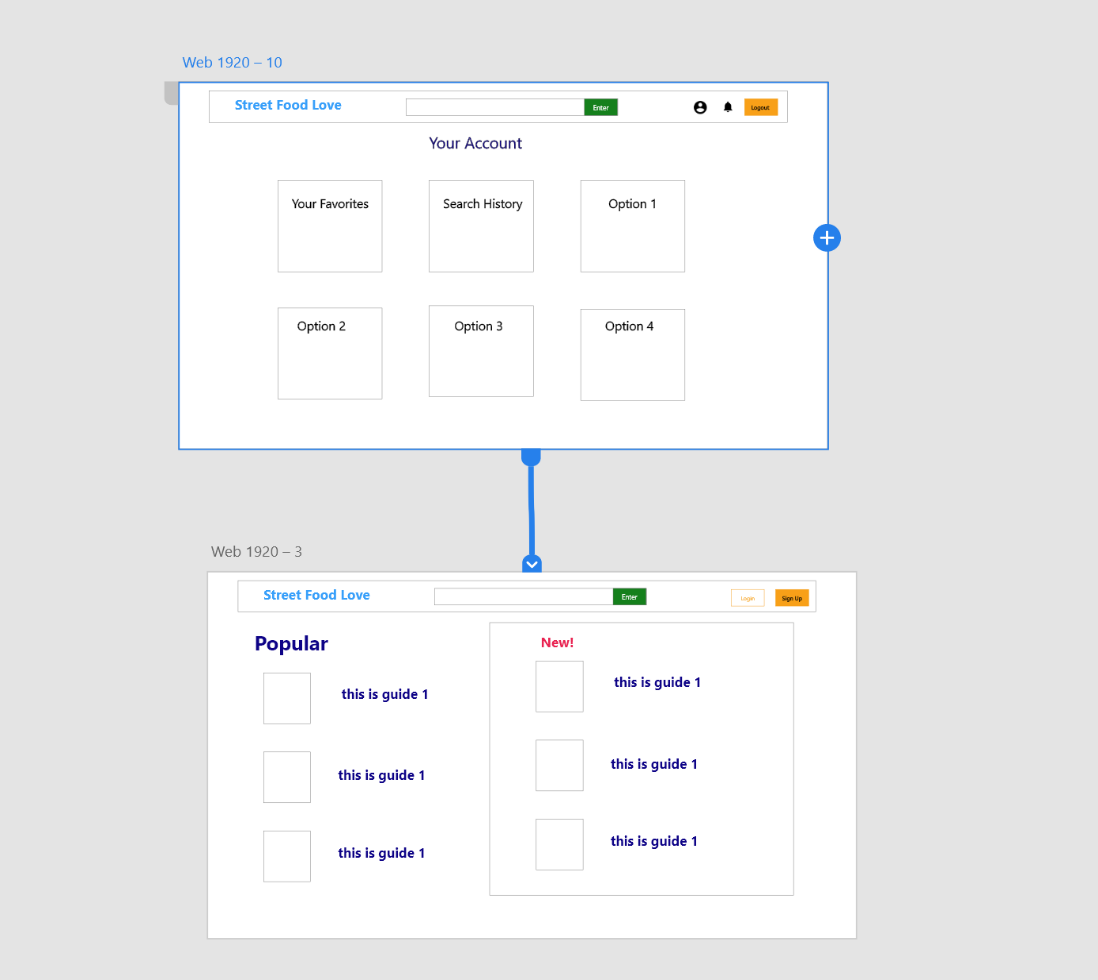


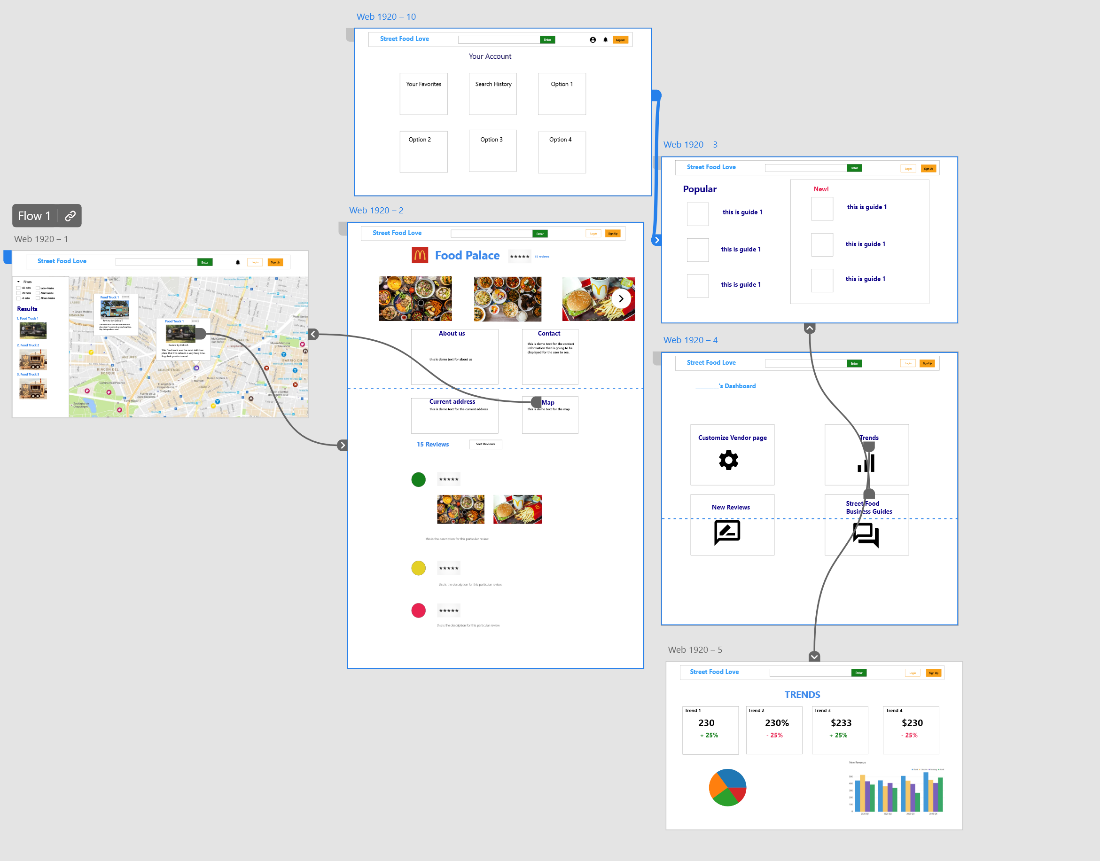
Screen Image 9



Screen Image 10

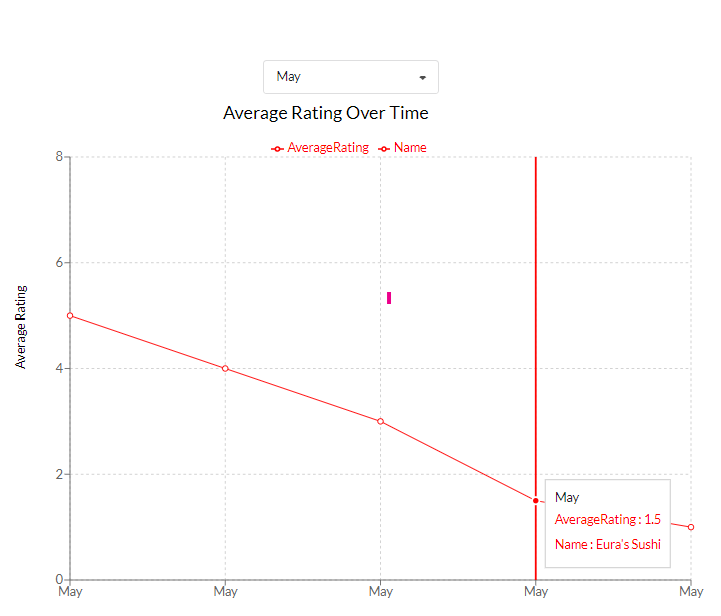






7.3 Dashboard Graph Designs:

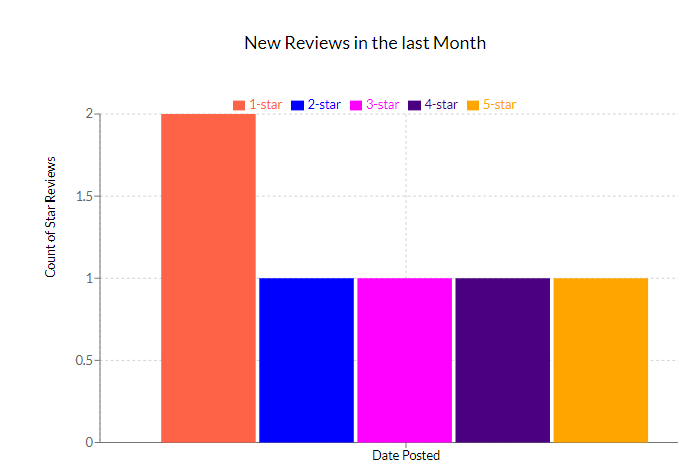
* Graph 1: Average Rating over Time



This graph shows the average Star Rating over time and can be filtered by month. One can see the month, average Star Rating, and the Vendor Name as a box pop-up by hovering over the points.

**Data source:** StarRating, average(StarRating), DatePosted, and VendorID from Reviews and Vendor tables

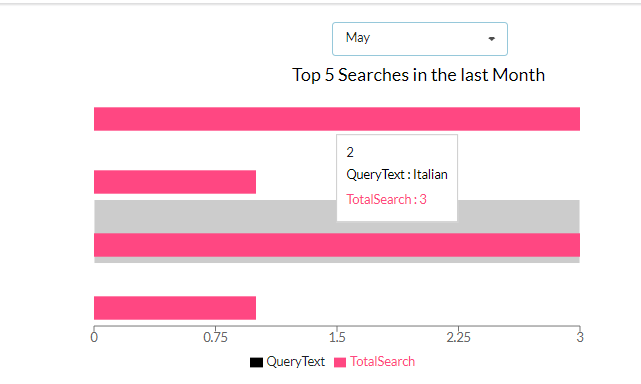
* Graph 2: New Reviews in the last month



This chart will display New Reviews submitted last month and broken down by individual stars. For example, from the graph above, we can see that two 1-star, one 2-star, one 3-star, one 4-star, and one 5-star review have been submitted in the last month.

**Data source**: StarRating, count(StarRating), DatePosted from Reviews table

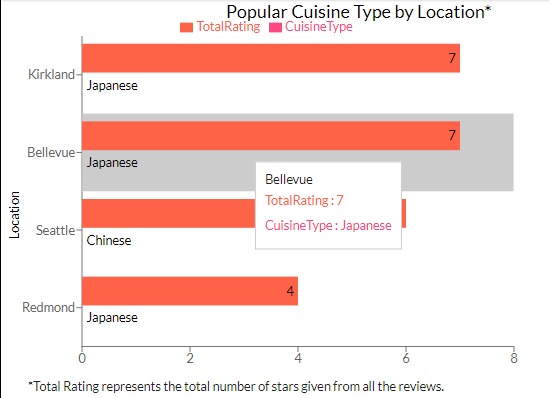
* Graph 3: Popular Searches by Month



The Popular Searches by month graph pulls searches submitted on the website and displays the most popular searches, which can be filtered by month.

**Data Source:** QueryText, DateRequested, count(QueryText) from Queries table

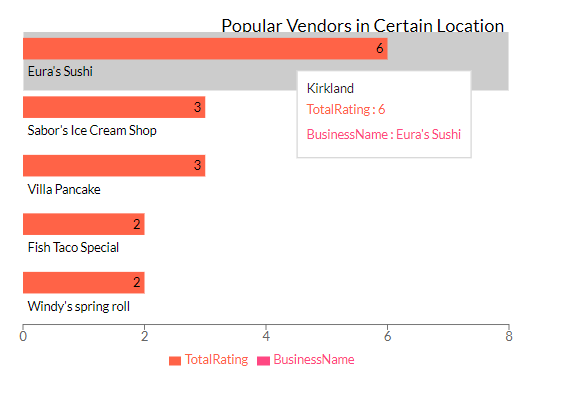
* Graph 4: Popular Cuisine Types by Location



The Popular Cuisine Type by Location graph displays the most popular cuisine types by location, and the popularity is based on the Rating. A box will pop up with the area, the total Rating, and the Cuisine type associated with the bar by hovering over the bar. The numbers on the end of each bar represent the TotalRating.

**Data Source:** AreaName, CuisineType, count(StarRating) from Areas, CuisineTypes, and Reviews tables with inner join on VendorID.

* Graph 5: Popular Vendors in a certain Location



The Popular Vendors by Location graph displays the most popular vendor by location, and the popularity is based on the Rating. A box will pop up with the area, the total Rating, and the Business Name associated with the bar by hovering over the bar. The numbers on the end of each bar represent the TotalRating.

**Data Source:** VendorName, AreaName, count(StarRating) from table Vendor, Areas, and Reviews with inner join on VendorID.

# REQUIREMENTS TRACEABILITY MATRIX

The following requirement lists don’t include any functions specific to Mobile Apps. At the moment, some features do not have a design as they will be added incrementally.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Requirement-ID | Requirement Description | Design  Component | Data Design Component | Interface  Design  Component |
| 1 | Vendor Page | 4.2 Getting and modifying resources  4.2 How to get a page and its pictures | Vendor table | Screen Image #2 |
| 2 | Reviews and Star Rating | 4.2 How to post a review with pictures | Review table | Screen Image #2 |
| 3 | Photo/Video Library | 4.2 How to get a page and its pictures | Photo table and S3 | Screen Image #2 |
| 4 | Sign Up and Log In | 4.2 Authentication flow | User table | Screen Image #1 |
| 5 | Map | 4.2 Leaflet map overlays | OpenStreetMap API | Screen Image #1 |
| 6 | Search | 4.2 AWS OpenSearch | AWS OpenSearch | Screen Image #1 |
| 7 | Filter/Sorting Items | 4.2 AWS OpenSearch | Vendor Table | Screen Image 1 |
| 9 | Location Tracking | 4.2 Leaflet map flow | OpenStreetMap API | Screen Image 1 |
| 10 | Vendor star | 4.2 Getting and modifying resources | Star table | Screen Image 2 |
| 11 | Notification | 4.2 Getting and modifying resources | Vendor table | Screen Image 1 |
| 12 | Food Recommendation | 4.2 Getting and modifying resources | PastSearch table | Screen Image 1 |
| 13 | Settings | 4.2 Getting and modifying resources | User table | Screen Image 3 |
| 14 | Single Sign-On | 4.2 Sign-in with Google flow | User table | Screen Image 7 |
| 15 | Analytics | 7.3 Dashboard Graph Designs | All tables + OpenSearch | Screen Image 4&5 |
| 16 | Dashboard on vendor page | N/A | N/A | Screen Image 5 |
| 17 | Business Starting-Up instruction | 4.2 Getting and modifying resources | Guides table | Screen Image 3 |