Team 15 - Design Document

Frederick Li, Shane Li, Shai Mohan, Joseph Singer, David Tong, Andrew Tully

Table Of Contents

PURPOSE

When ordering food online, costly delivery fees and steep minimum order amounts often discourage potential customers from completing their order. If these delivery costs were reduced, customers would be more likely to place an order, and businesses would benefit as well. Our project will offer a solution to this issue by connecting nearby customers to form large group purchases, reducing the delivery costs for each person that joins the group, or "cluster".

As a guest, I would like to be able to:

- view existing clusters.
- create an account using Facebook

As a logged-in user, I would like to be able to:

- view my profile.
- view and join existing clusters.
- sort clusters by newest, distance, or alphabetically by restaurant name.
- view information about a cluster and its members.
- request to start a cluster.
  i) specify where I would like to pickup the order.
  ii) choose a restaurant to order from.
  iii) set minimum cluster size.
  iv) set maximum cluster size.
  v) set a duration time for the cluster.
- log out.

As a cluster leader, I would like to be able to:

- *have all the capabilities of a cluster member.*
- approve/reject users joining cluster.
- edit information about my cluster.

As a cluster member, I would like to be able to:

- *have all the capabilities of a logged-in user.*
- view restaurant information about my cluster.
- view pickup/meeting location for my cluster
- view information about all other members of my cluster.
- leave ratings for other members of my cluster.
- leave a cluster.

As an administrator, I would like to be able to:

- disband clusters.
- review order history.
- review cluster history.
- view number of clusters active.
- view number of users online.
- view total number of orders completed.
- view total number of orders not completed.
- view user information.
- ban users (if time permits)

Design Outline

High Level Overview

Cluster will use a client-server model, where the server will communicate with the client using the RESTful API protocol. The client will be responsible for making requests to and parsing responses from the server.

Client

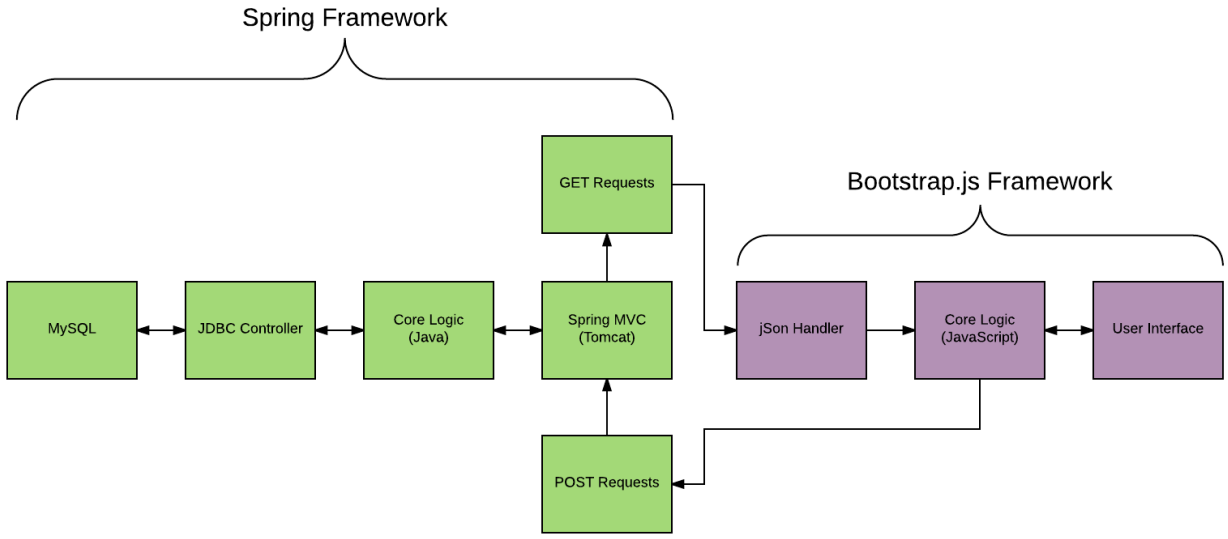
The client will be implemented in JavaScript using a Bootstrap.js framework. It will be responsible for sending requests to the server through GET and POST requests and for parsing JSON responses from the server and generating a view for users.

Server

The server will be implemented in Java using a Spring framework and will be responsible for receiving requests, sending responses to the client, sending queries to update or request information from the database, validating requests sent from the client, and processing data to be interpreted by both the client and the database.

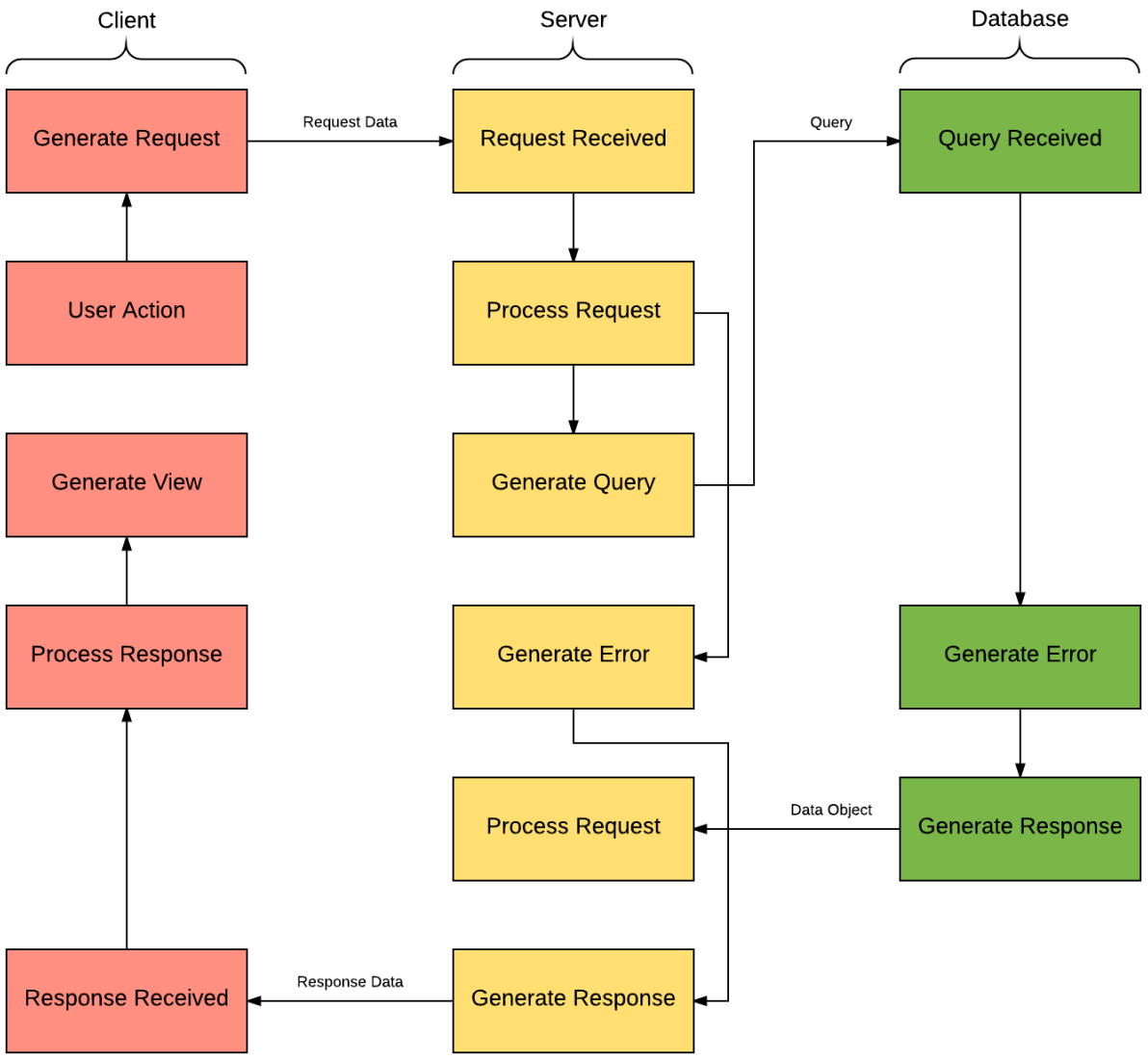
Database

The database will be implemented using MySQL and will store user data, cluster data, restaurant data, etc.



Activity Diagram

This activity diagram designates a user action as the starting point. The user action will be interpreted by the client and a request will be sent to the server using an HTTP request. If the request is invalid, an error will be returned immediately to the client, which will process the error appropriately. If the request is valid, the request is processed normally. If a database query is required, the server will query the database through the database controller. The database will then process the query and return an appropriate response back to the server. After the response is received by the server, it will process the data and create a response for the client. Once the client receives the response, it will process the response and generate a view for the user.



Design Issues

Issue: Which platform should we use?

- **Option 1: Web application**
- Option 2: Mobile (iOS/Android)

When considering a platform to use, reaching a large user base and having low overhead were prioritized. Because of this, a web application was chosen. A mobile-friendly web application is able to target both the desktop and mobile user bases without requiring two separate applications.

Issue: What type of architecture should we use?

- **Option 1: Client-Server architecture**
- Option 2: Unified architecture

The Client-Server Architecture separates the application into front end and back end, allowing our team to work on different parts of the project concurrently without risk of compatibility issues.

Issue: What back end framework should we use?

- **Option 1: Spring**
- Option 2: JSF

Spring is a request based framework while JSF is component based. This means that rather than create components that only fit our current view of the app, we parse HTML requests and generate HTML responses when needed. This allows us to develop custom components for concepts with out of the box functionality. Also, JSF attempts to decrease the separation of client and server making communication between the two very slow and unreliable.

Issue: What database software should we use?

- **Option 1: MySQL**
- Option 2: MongoDB
- Option 3: PostgreSQL

MySQL is a relational database, as opposed to the document-oriented MongoDB and object-relational PostgreSQL. In addition, MySQL offers easy integration with Spring framework. Members of our team have experience with using MySQL, so selecting MySQL for our project will reduce time spent learning and potentially reduce mistakes as well.

Issue: What front end framework should we use?

- Angular.js
- **Option 2: Bootstrap.js**
- Option 3: React.js

Bootstrap.js has an excellent standardized platform with templates, themes, and plugins options and ensures compatibility with all major browsers. The user interface is responsive, helping us accommodate mobile devices. Additionally, Bootstrap.js offers quality documentation and community support.

Issue: What type of design scheme should we use?

- Option 1: Adaptive web design
- **Option 2: Responsive web design**

A responsive web design creates a consistent user experience regardless of the device used to view the website. Creating another version of our application for different devices would create more work for our front end team for insignif.

Issue: Do users need to login to use our service?

- **Option 1: Have users login through Facebook**
- Option 2: Allow guest access
- Option 3: Allow account creation through email and password

Having users login through Facebook avoids potential security issues. Facebook can handle the majority of user information. We decided to go with option 1 to have users login through Facebook. This has Facebook handle all user information as well as allow users to check out the people before they order with them. This allows users to feel safer when meeting up with random people.

Issue: When should clusters be removed from the dashboard?

- Option 1: Set a certain time such as 15 minutes
- **Option 2: Allow users to set a time**
- Option 3: Set a time where it stays up until nobody joins for that amount of time

We decided to go with option 2 which allows users to set the time it stays up. We will have a default time but they can change this if they would like to. We can also use a hybrid with option 3 if it would be useful in the application.

Issue: In what order should we display the clusters on the dashboard?

- Option 1: Order alphabetically by restaurant name
- Option 2: Order cluster creation time
- Option 3: Order how long is left on the cluster
- Option 4: Order by average rating of users in a cluster
- **Option 5: Allow users to select filter**

Allowing the user to sort the dashboard according to their priorities is the ideal implementation. Letting the user decide increases flexibility and can make the application more maneuverable and intuitive for the user. If time allows, we will implement a search bar to target specific restaurants or clusters.

Issue: How should cluster members communicate with each other?

- **Option 1: Internal chat server**
- **Option 2: Provide contact information**

A chat server would allow users to communicate without sharing potentially sensitive contact information, but this feature will only be implemented if time allows due to it being low priority overall. In the event that time constraints restrict us from creating a chat server, we will require users to share Facebook contact information but allow them to choose to share other contact information.

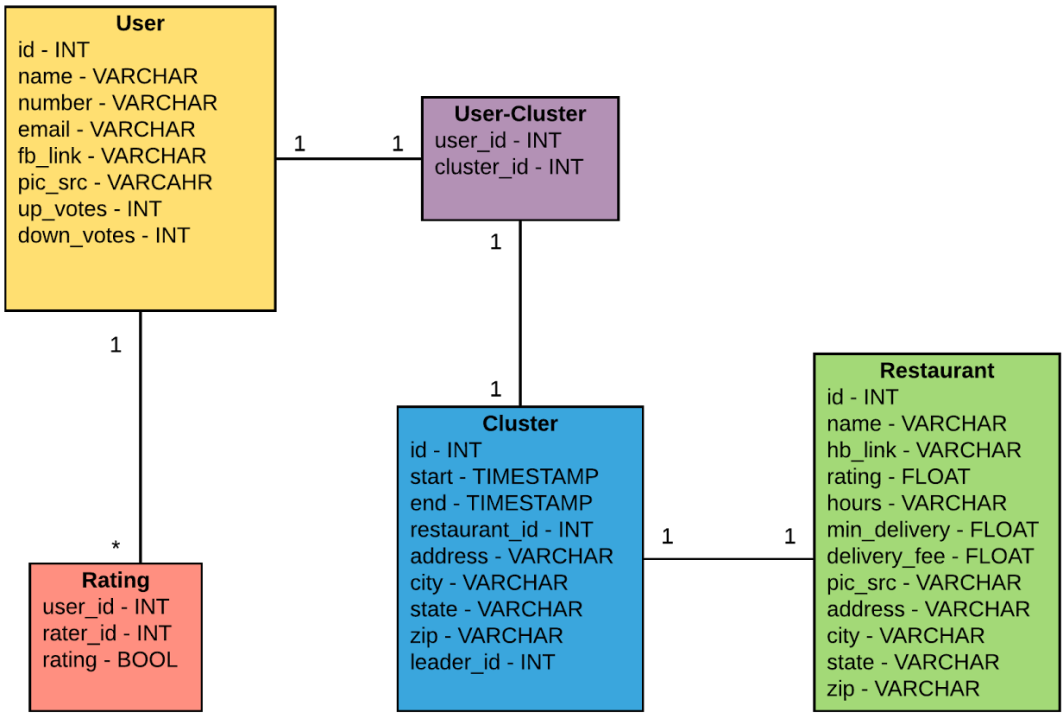
Issue: Once a group has been formed, how should the order be placed?

- Option 1: Place order through the web application
- Option 2: Route to restaurant's website
- **Option 3: Route to third party delivery website**

By routing users to a third party's delivery website, we avoid managing sensitive user information and monetary transactions. In addition, a third party delivery website may have more flexible delivery hours compared to the restaurant, hence our selection of option 3 over option 2.

Design Details

Database Schema



Description of Database Tables

Each of the boxes in the database schema represent a table in our mySQL database. The columns of each table are listed below the table's name, with their data types listed to the right.

User: Rows represent users

- *id* - unique INT assigned upon the user's account creation
- *name* - name of the user stored as a VARCHAR
- *number* - phone number of the user stored as a VARCHAR
- *email* - email of the user stored as a VARCHAR
- *fb_link* - url of the user's FaceBook profile stored as a VARCHAR
- *pic_src* - url of the profile photo of the user
- *up_votes* - number of upvotes that a user has received stored as an INT
- *down_votes* - number of downvotes that a user has received stored as an INT

Rating: rows represent individual ratings

- *user_id* - INT id of the user who received the rating
- *rater_id* - INT id of the person who submitted the rating
- *rating* - BOOL field indicating whether the rating was an upvote (TRUE) or downvote (FALSE)

User-Cluster: Used to correlate users to clusters, rows represent correlations

- *user_id* - unique INT id of a user
- *cluster_id* - corresponding INT id of the cluster a user has joined

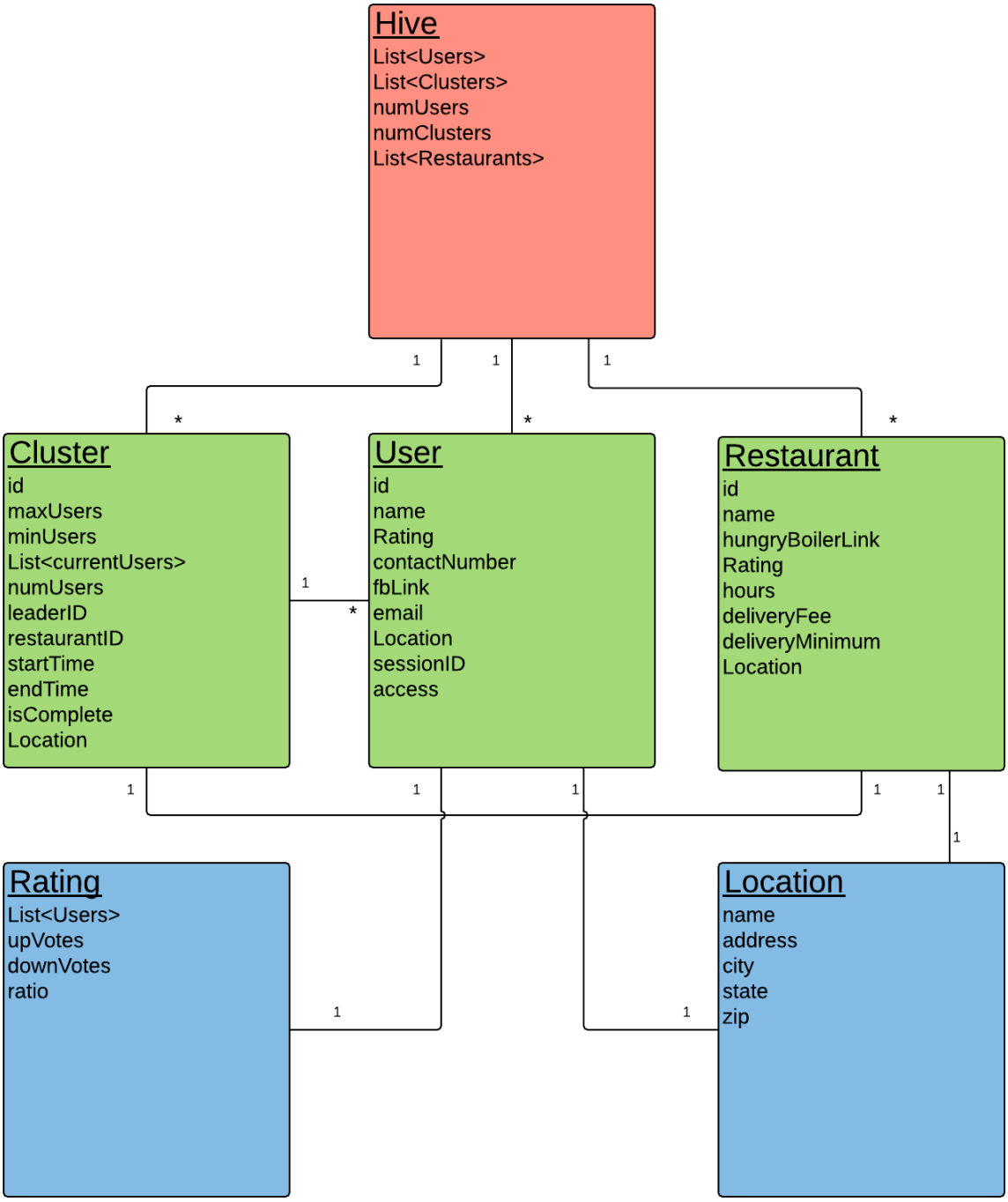
Cluster: Rows represent clusters

- *id* - unique INT id assigned upon the cluster's creation
- *start* - TIMESTAMP representing the time the cluster was created
- *end* - TIMESTAMP representing the time the cluster's order was completed
- *restaurant_id* - INT representing the id of the restaurant the cluster will be ordering from
- *address* - name of the street address that the order will be delivered to the cluster members stored as a VARCHAR
- *city* - city that the order will be delivered in stored as a VARCHAR
- *state* - state that the order will be delivered in stored as a VARCHAR
- *zip* - the zip code that the order will be delivered in stored as a VARCHAR
- *leader_id* - the user_id of the leader of the cluster

Restaurant: Rows represent restaurants

- *id* - unique INT id of the restaurant
- *name* - name of the restaurant stored as a VARCHAR
- *hb_link* - url link to the restaurant on HungryBoiler stored as a VARCHAR
- *rating* - FLOAT value of the rating this restaurant has on HungryBoiler
- *hours* - the hours of operation that the restaurant is open stored as a VARCHAR
- *min_delivery* - FLOAT value representing the minimum purchase order needed for delivery
- *delivery_fee* - FLOAT value representing the delivery fee
- *pic_src* - url link to the src for the restaurant's photo stored as a VARCHAR
- *address* - street address of the restaurant stored as a VARCHAR
- *city* - city that the restaurant is located in stored as a VARCHAR
- *state* - state that the restaurant is located in stored as a VARCHAR
- *zip* - zip code that the restaurant is located in stored as a VARCHAR

Class Structure



Description of Classes and their Interactions

Hive

- Highest level of the hierarchy and can only be accessed by the admin
- It contains all of the restaurants users can order from, the users currently online, and the clusters that are currently active
- Will be the first class created
- Used to populate the dashboard

Cluster

- Represents a group being formed for the group order
- End goal of any user is to join an instance of this class
- Created by a user and tied to a specific restaurant
- Includes information about the restaurant, delivery prices, and users in the cluster
- A one-to-many relationship exists between the cluster and its users, as each user can only be in one cluster at a time

User

- Represents a single, logged in user
- Contains personal information about the user such as their name, phone number, and location
- Provides a link to the user's FaceBook page for group chats to be formed

Restaurant

- Represents a real restaurant
- Contains information about the restaurant such as its hours of operation, its delivery fee, and its location
- Clusters and restaurants will be paired in a one-to-one relationship

Rating

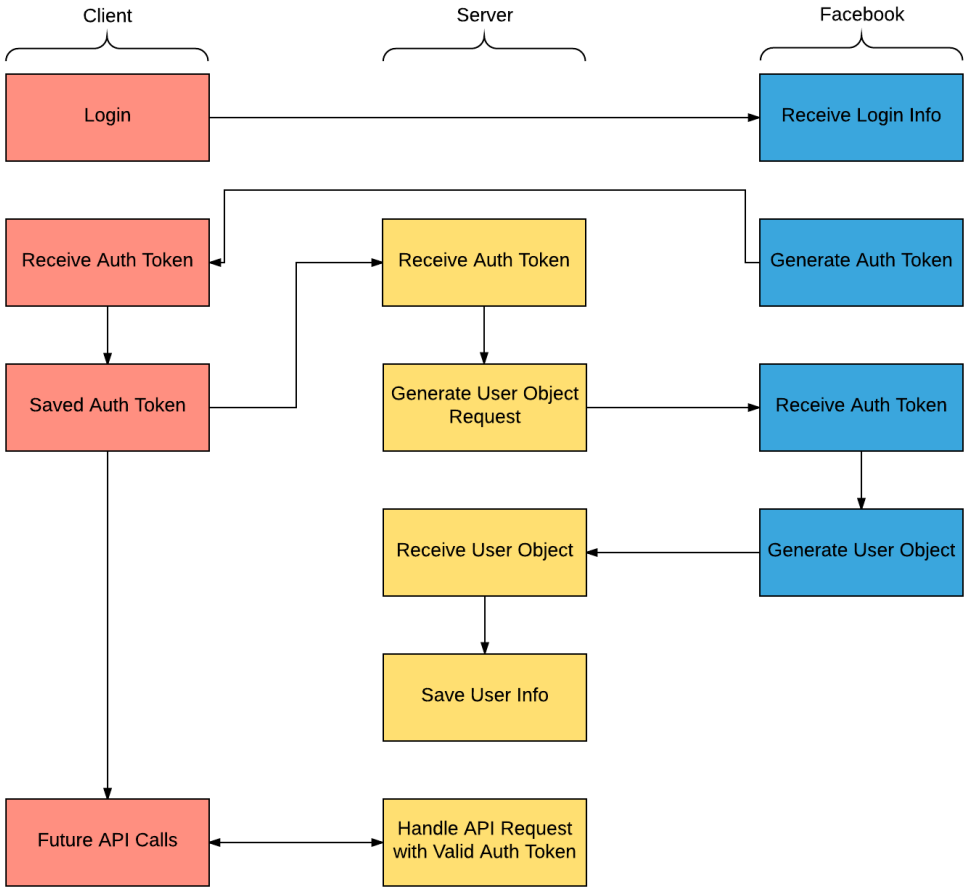
- Represents the reliability of a user
- Uses an upvote/downvote system
- Reports upvote percentage

Location

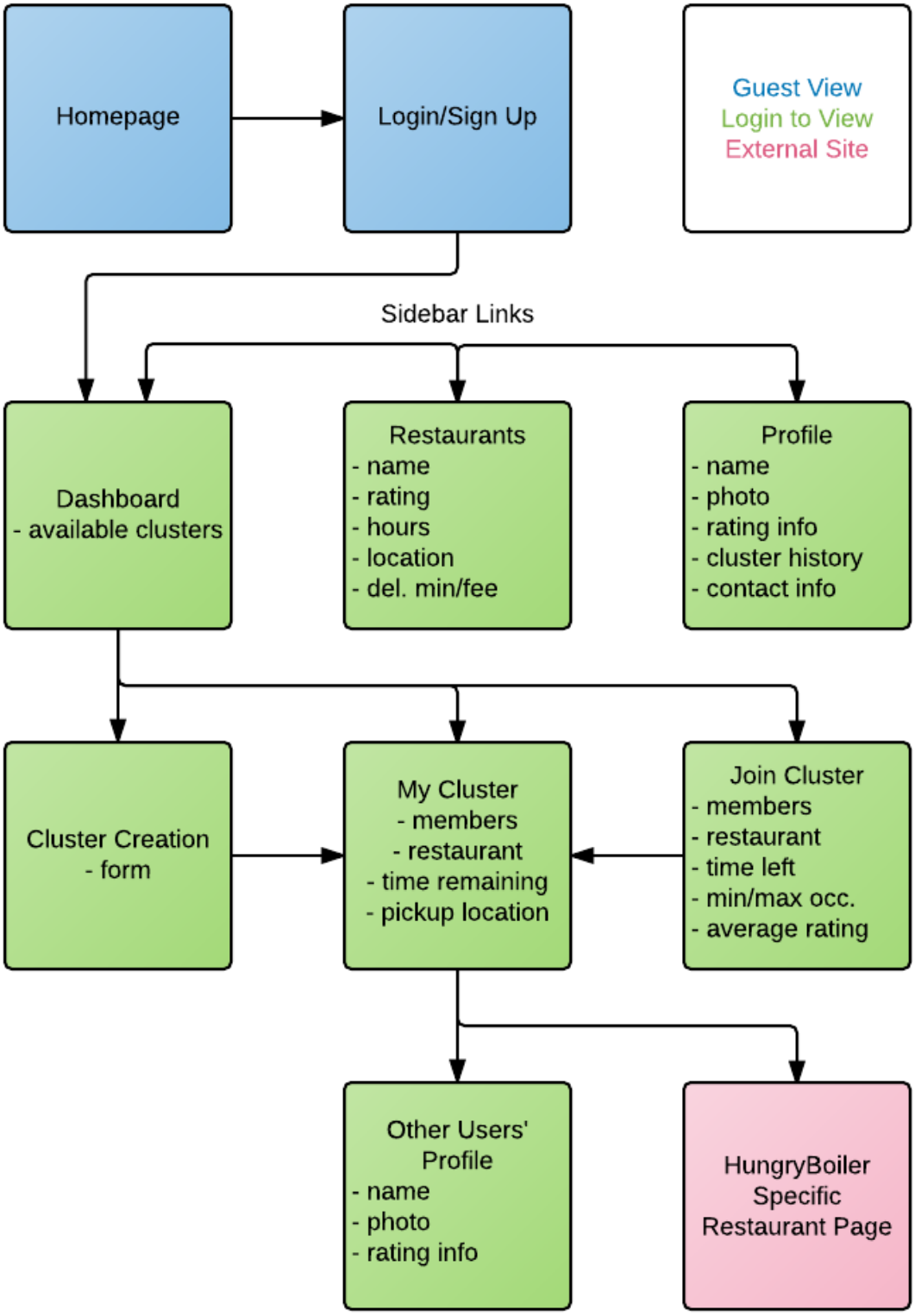
- Represents a physical postal address
- Used to specify users' locations, restaurants' locations, and clusters' pickup locations
- Includes the details of the postal address

Facebook Authentication

Cluster will be using facebook authentication to handle login authorization. Users will be prompted to login at the home page into their facebook accounts. When they click the login button, they will be redirected to facebook servers. Once they have logged in, the facebook servers will return an access_token to the client. The client will then return the access_token to the server, and the server will authenticate the token with facebook, and facebook will return a user object. Now, the server has the necessary information from facebook, and the client has the access_token to make API calls.



User Interface Flow Chart



User Interface Page Description

Homepage: The homepage will provide a brief description of our project and its purpose, as well as a link to redirect the user to the login/sign up page.

Login/Sign Up: The login/sign up page will allow for users to login view Facebook and input contact information.

Dashboard: The dashboard displays the currently active clusters which you can click to be redirected to the join cluster page.

Restaurants: The restaurants page displays a list of all restaurants available for delivery, as well as their hours, delivery minimum, address, contact information, and available clusters for that restaurant.

Profile: The profile page displays the user's profile photo, contact information, past clusters, rating, and an edit profile option.

Cluster Creation: The cluster creation page contains a form to fill out for the cluster. It will request name, pickup location, restaurant, maximum and minimum amount of members, and time until cluster disband.

My Cluster: The my cluster page will display members currently in the cluster, the restaurant, time remaining, pickup location, and member ratings. Users who applied to join the cluster will be displayed to the cluster leader to accept or reject.

Join Cluster: The join cluster page will display the cluster's restaurant, time left, members, member ratings, and maximum and minimum members for the cluster.

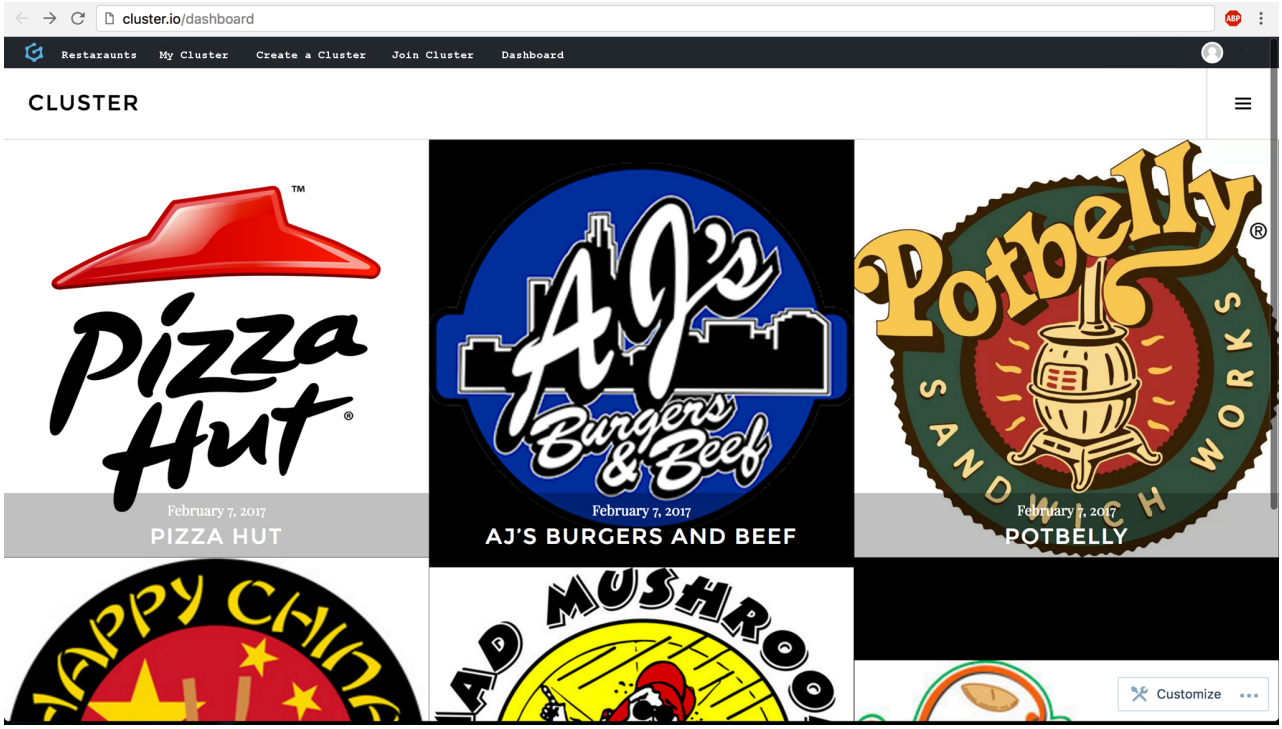
Other Users' Profile: The user profile page will display their profile photo, select contact information, and rating.

HungryBoiler Specific Restaurant Page: Once there are enough members in the cluster, the leader can choose to be redirected to the chosen restaurant's HungryBoiler ordering page.

User Interface Mockups

Homepage

Dashboard



User Profile Page