fooDrop - Food sharing webapp

PART C
URL: github.com/preethamrn/fooDrop
YouTube video: https://youtu.be/bweVMIYLL2s
Team Name: fooDrop Inc.

Preetham Narayanareddy: 604630101
Leslie Liang: 204625818
Farhan Alam: 804595663
Karthik Ramesh: 104687637
Udayan Sahai: 504646937

12/2/2018

1 Motivation

In our current era, food waste is an ever-growing problem. From having expired food in the refrigerator to making too much food and having leftovers, we seem to waste food quite frequently. For example, one of our team members mom would make a large portion of food for a family gathering only to have half of the food go to waste because half of the family members decided not to show up. Our proposed product aims to solve this very problem. We are creating a web application called fooDrop. Our web app will help users save money by being able to sell their leftovers immediately. We do not want the sellers to wait too long to sell the food because the food may go bad once the buyers have purchased it. Our web app not only helps people save money but it is also beneficial towards the environment because we would be discarding less food into the dumpsters.

Our vision is to develop a feature-rich web application with an intuitive user interface to help our sellers sell their leftovers to other nearby buyers. This benefits the seller because they wouldnt need to risk having their leftovers go bad in the refrigerator. On the other hand, this can also benefit the buyer because the food will be sold at a cheaper price than food at a restaurant. To limit any liability issues that fooDrop might face, we support the ability to filter food by allergens, thereby creating a safe platform for buyers and sellers of food to leverage and minimize food wastage.

As developers, we are aware of the health concerns and risks of buying expired food for our buyers. If we decide to deploy this application to the public, we will require the sellers to abide by the Cottage Food Bill, AB1616 by written consent. The Cottage Food Bill allows certain low-risk food products to be prepare and package in private home kitchens in California. This ensures that the sellers are aware of the limitations of some of the food they can provide; this also aims to increase our buyers' shopping confidence.

2 User Benefits

2.1 Scenario: A successful attempt at purchasing food using fooDrop

1. Thomas presses the "SIGN IN WITH FACEBOOK" button
2. fooDrop prompts Thomas to enter his Facebook credentials to log him in
3. Thomas enters his Facebook credentials
4. fooDrop displays a homepage with a map overlay in the center of the app and a navbar on the upper right side of the app
5. Thomas presses the "SEARCH" button

6. fooDrop displays the input field for **Name**, dropdown menu for **Dietary Restrictions**, input field for **Ingredients**, and slider bars for **Price** and **Radius**

7. Thomas indicates the dish name is BigMac, no dietary restrictions, ingredients include eggs, tomatoes and sausage, price is $4 and radius is 1 mile.

8. fooDrop displays a list of dishes that matches satisfies Thomas' inputs

9. Thomas presses the "VIEW DISH" button next to the second option in the listings

10. fooDrop opens up more details about the dish. For example **Description**, **Dietary Restrictions**, **Price** and **Quantity** are shown. On the bottom, there are three buttons: **CLOSE**, **CONTACT SELLER** and **BUY**

11. Thomas scrolls through the pop-up to see more details about the dish. Thomas then presses the **BUY** button.

12. fooDrop redirects the transaction to PayPal

13. PayPal creates the transaction then prompts Thomas to enter his PayPal credentials

14. Thomas enters his PayPal credentials

15. PayPal executes the transaction and returns control back to fooDrop

16. Thomas is redirected back to the homepage of fooDrop with the map overlay centered at the middle of the page

2.2 Scenario: Buyer with nut allergies buying food from fooDrop

1. Kim presses the "SIGN IN WITH FACEBOOK" button

2. fooDrop prompts Kim to enter her Facebook credentials to log her in

3. Kim enters her Facebook credentials

4. fooDrop displays a homepage with a map overlay in the center of the app and a navbar on the upper right side of the app

5. Kim presses on the "PROFILE" button

6. fooDrop displays Kim's user profile along with the her transaction history

7. On dietary restrictions, Kim inputs **nut free**. Kim now presses the "HOME" button

8. fooDrop filters all the dishes that satisfies Kim's user profile and displays a list of dishes that matches Kim's user profile settings

9. Kim selects a dish on the map
10. fooDrop opens up more details about the dish. **Description**, **Dietary Restrictions**, **Price** and **Quantity** are all shown. And on the bottom, there are three buttons: **CLOSE**, **CONTACT SELLER** and **BUY**
11. Kim scrolls through the pop-up to see more information about the dish. She notices that the dish has a field called **Dietary Restrictions** and under it says nut free. After she has confirmed that the dish has no nuts, she presses the "BUY" button
12. fooDrop redirects the transaction to PayPal
13. PayPal creates the transaction then prompts Kim to enter her PayPal credentials
14. Kim enters her PayPal credentials
15. PayPal executes the transaction and returns control back to fooDrop
16. Kim is redirected back to the homepage of fooDrop with the map overlay centered at the middle of the page

2.3 Scenario: Seller has extra food that he would like to sell using fooDrop

1. Carey presses the "SIGN IN WITH FACEBOOK" button
2. fooDrop prompts Carey to enter his Facebook credentials to log him in
3. Carey enters his Facebook credentials
4. fooDrop displays a homepage with a map overlay in the center of the app and a navbar on the upper right side of the app
5. Carey presses on the "SELL" button (**previously the button was "NEW" **)
6. fooDrop displays the input field for **Image URL**, input field for **Name**, input field for **Description**, dropdown menu for **Dietary Restrictions**, input field for **Ingredients**, input field for **Price** and and input field for **Quantity**
7. Carey enters the **Image URL**, **name** of the dish, **description** of the dish. From the dropdown menu, Carey selects **nuts** as one of the ingredients. After that, Carey presses the "POST" button
8. fooDrop will query the backend to upload Carey's data to the database
9. Backend responds with "SUCCESS" meaning data has been successfully stored in MongoDB

10. fooDrop reponds with an alert message “Success!” to inform Carey that his dish has been successfully created

3 Feature and Requirements Description

3.1 Feature Description

Our app consists of various features: homepage dish listings, homepage dish details, homepage Google Maps, seller dish creation page, dish search, seller and buyer chat page, and a user profile page.

3.1.1 Log in Page (with Facebook Oauth)

This log in page allows a user to sign in using their Facebook account in order to access their profile. Pressing the 'log in with facebook' button prompts a pop-up with Facebook log in. If a user enters their proper credentials, they are redirected to the dish listings home page once they have successfully logged in. If a user fails to provide proper authentication, they are prompted to log in with proper credentials. Once a user presses the log out button, they are redirected to this log in page.

We have kept our implementation the same from parts A & B.

3.1.2 Homepage Dish Listings

The user can view the available dishes that match their profile and filter dishes as per their preferences on this page. The radius, dietary restrictions and price range the user sets from the user profile page will automatically filter dishes to match the user's specifications and needs. The user can also search for dishes with more filters from the search page. The user can also see the distance from their current location to the dish on a link in each dish listing. They can click on the link to be redirected to a new tab from Google that gives them directions from their current location to the dish.

We have kept this part the same from parts A & B.

3.1.3 Homepage Dish Details

The dish listings page contains information about the corresponding dish. It has fields for description, location, dietary restrictions, ingredients, and price. Furthermore, the user can select a quantity of the dish they would like to buy before pressing the 'buy' button. The user also has the ability to contact the seller about the dish by pressing the 'contact seller' button. This will lead them to a chat page that enables them to directly chat with the seller for that dish.

We have added on to our implementation from parts A & B by creating a chat page for the buyer and seller to personally talk about dishes.

3.1.4 Homepage Google Maps

The Google Maps overlay on the home page contains markers for all the dishes listed below from the Dish Listings. Each of the dish listings below the map will be shown as markers on this map overlay. By clicking on a marker, the user can see the dish details, and choose to either buy the dish, or contact the seller for more information via a chat page.

We have added to our implementation from part B by implementing the Google Maps API that we specified in part A.

3.1.5 Chat Page

A user can access the chat page by clicking on 'contact seller' on the dish details page. This enables a user and seller to communicate through our application. A buyer and seller will have their own unique chat page for each unique dish.

We have added to our implementation from part B by implementing this chat page that we specified in part A.

3.1.6 Sell Dish Page (with Google Maps)

A user who wants to sell their dish can route to this page by clicking on the 'sell' button from the header. This sell dish page provides a form that lets the seller details various dish information, such as the quantity, price, ingredients, location, and dietary restrictions that the dish entails. The location is selected by using the Google Maps overlay feature, similar to the one on the home page. This time, this seller can click to place a marker on the map that shows where they would like to sell their dish. Once the seller presses the 'submit' button, an alert from the browser will let the user know if their listing has been successfully saved to the database or not. This makes things more convenient for the user, since they are now working with a visual feature for the location they will be selling the dish from.

We have added to our implementation from part B by implementing this page with the Google Maps API that we specified in part A.

3.1.7 User Profile Page

This feature allows a user to modify various settings relating to their profile. If logged in through Facebook using OAuth, the user will receive an authentication token and a userId which is linked to their profile. A user would reach this menu by selecting the 'profile' button in the page header. Once on the profile page, the user can see a list of previous transactions

that they've completed as well as some settings that they can modify. Clicking on any one of these transactions brings up the dish description page. The modifiable settings are the dietary restrictions of the user and the default radius, which will automatically filter dishes from the listings and map on the homepage. The PayPal ID is linked to the account and used for payment when buying a food item.

We have added to our implementation from part B by implementing PayPal and the chat page we specified in part A.

3.1.8 Search Page

When the user clicks on the 'search' button in the page header, it will lead them to the search page which allows the user to filter by certain categories. The categories include name, ingredients, dietary restriction, price range, and distance radius. The name category allows the user to type a name and filter the searches by a certain dish name. The user can also specify certain ingredients and/or dietary restrictions they want in their dish. For example, the ingredients can be carrot, onion, and cheese, and the app will search for dishes with all those ingredients in them. We will also have a drop down menu with many dietary restrictions that the user can select from. The results of the search will be filtered against these restrictions. Finally, the user can input a radius in miles that denotes the distance in radius between their home and the pickup location for the food. When the user clicks 'submit', they will be redirected to the homepage with all dishes that satisfy their filters.

We have maintained our implementation from parts A & B.

3.1.9 Header

The user can see the header on every page once they log in. The feature contains links to all the main pages in the application, such as the homepage, user profile, logout, sell dishes, and search dishes that are buttons titled with 'home', 'profile', 'logout', 'sell', and 'search', respectively. This feature enables a user to access many parts of our application relatively easily. The header is not found on the 'login' page because a user should not be able to access any of the application's other features unless they are logged in.

We have maintained our implementation of the header base from parts A & B.

3.2 Non-Functional Requirements

- Having 100 concurrent fooDrop users, 80% of searches should be satisfied within 2 seconds
- A user of fooDrop should not be able to login to a different user's account
- Having 50 concurrent chats open, 80% of chat messages should be sent within 3 seconds

4 Design

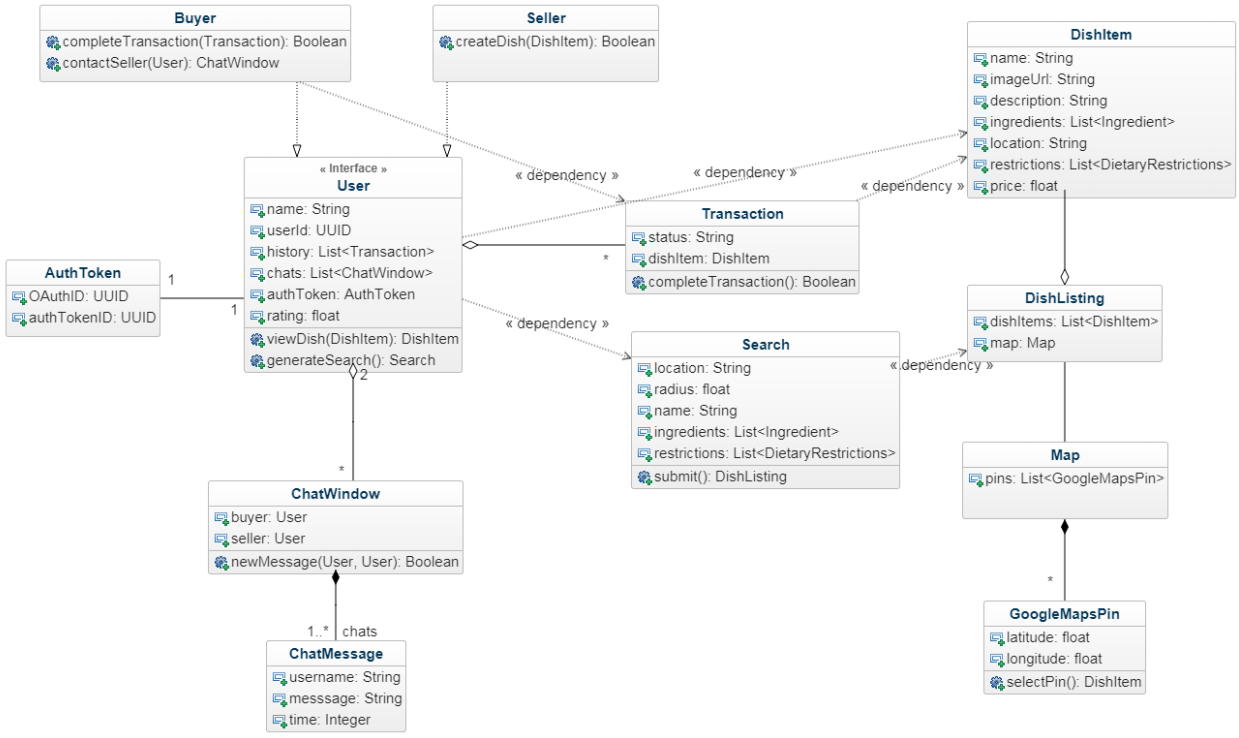

Figure 1: Old class diagram for our application.

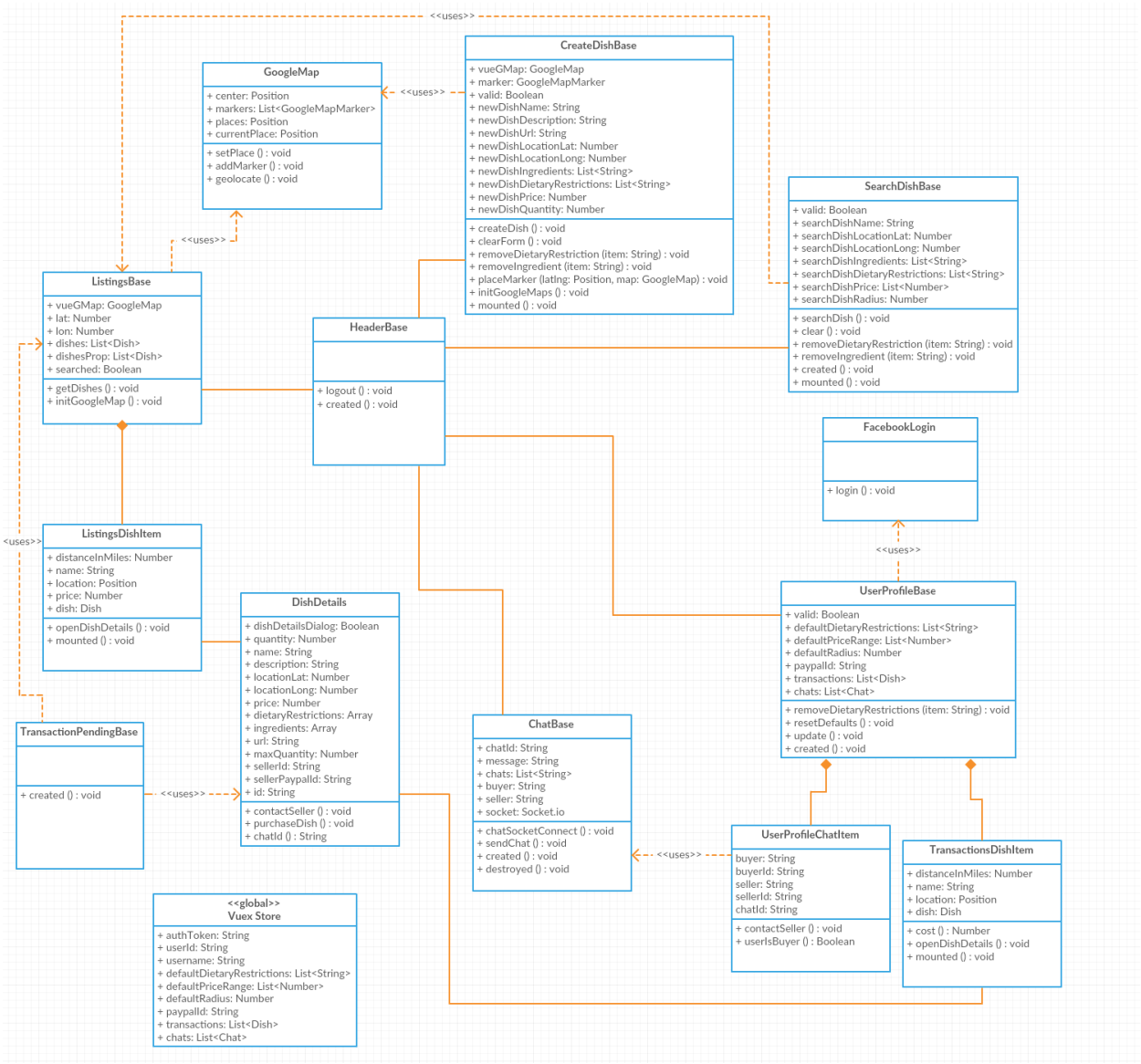

Figure 2: Updated front-end class diagram for our application.

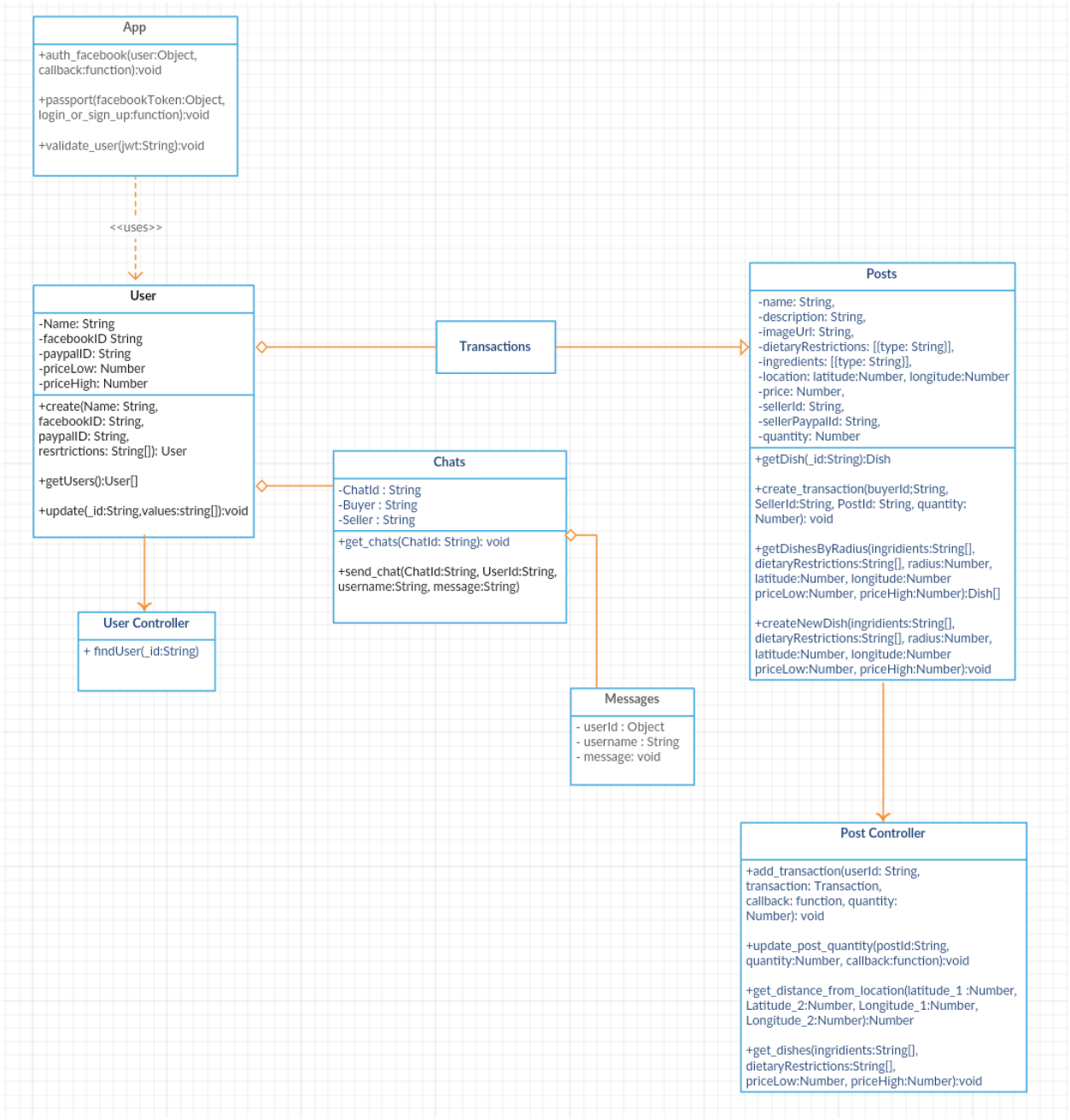

Figure 3: Updated back-end class diagram for our application.

Our old class diagram had assumed a unified front-end and back-end application. This turned out to not be the case as our back-end and front-end work as independent units and thus require separate class diagrams.

Our back-end class diagram is a simpler version of the front-end class diagram and attempts to model the same data as the front-end one. It does so without the added com-

plexity of having to deal with Map overlays, Listings, Facebook authentication etc. Our back-end class diagram is only responsible for keeping track of and providing information to the front-end. The front-end can manipulate and store this information appropriately in a more complex and intricate structure.

Hence, we will only be comparing our front-end class diagram with our old class diagram as it is an accurate indicator of the changes we have implemented since Part A. While our old class diagram is well-suited to an Object oriented programming language such as Java, a callback-based web development language such as JavaScript requires us to make some changes to our class structure.

On the user's side we decided to collapse the inheritance hierarchy of having a buyer and a seller subclass extending the user base class into a a single user class. This not only makes it significantly easier for us to store such an object in Mongoose, but also allows the buyer's and seller's to freely interchange their roles to form a fluid marketplace.

The chat implementation in the UserProfileChatItem is similar to the old chat class as it uses the same fields and the same basic principles as the old class diagram. However, it uses a new socket.io API through the ChatBase class. We still continue to make use of the Transactions class, but we embed this inside the User class so that each user can keep track of their own transaction information which can be displayed to them.

We do not make any major implementation and structural changes in the DishListings and the DishItem class.
However, we have added a new DishDetails class to avoid the problem of primitive obsession which is caused by adding too many primitive type variables.
Our DishListings class in the new version has an aggregation of Google Maps objects.
This is similar to the old version but we have decided to get rid of the Map pins objects as we found those to be redundant.

For the purpose of this project we have used the following libraries and technologies:

Node.js - This is an asynchronous JavaScript based library which is usually run on the back-end of a web application to serve a client. We use Node.js to retrieve information from the database and handle any http requests from the client.

MongoDB - MongoDB is a NoSQL document based databases that used JSON objects for its schema. We used MongoDB to store our persistent data such as our User and Post information.

Google Maps Platform - The Google Maps platform allows us to build custom Maps with specific locations indicated by pins. We have used this API to create a map overlay on

our front end to help the users get a visual idea of the food options available near them.

Paypal REST API - The Paypal payments REST API allows easy and secure online and mobile payments. We use the Paypal payments API to allow the user to effortlessly pay the seller for a dish that they wish to purchase.

Vue - Vue is a well developed open source framework for frontend web development. It's single file components allows for easy development and documentation. There are also many modules and plugins that support Vue such as Vuex for state management and Vue-Router for single page application routing.

5 Description of your APIs

Link to the Front-End API Documentation:

`http://preethamrn.github.io/fooDrop/client/docs`

Link to the Back-End API Documentation:

`http://preethamrn.github.io/fooDrop/server/docs`

6 User Interface Snapshots

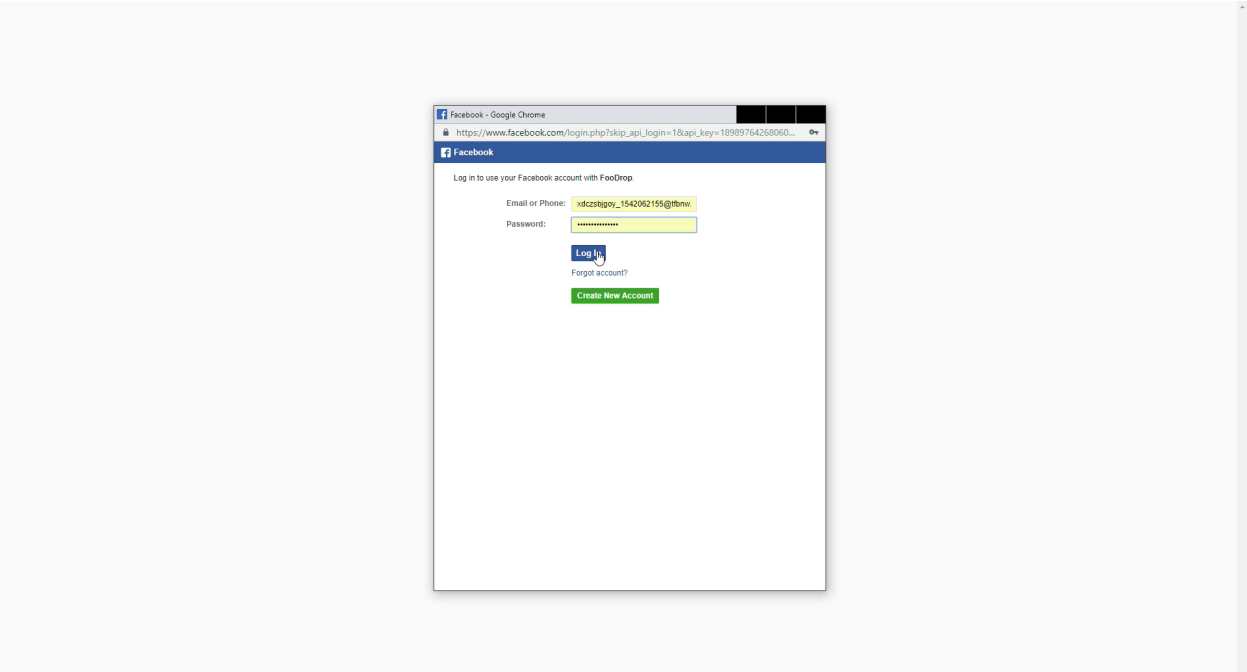

Figure 4: Login securely with Facebook.

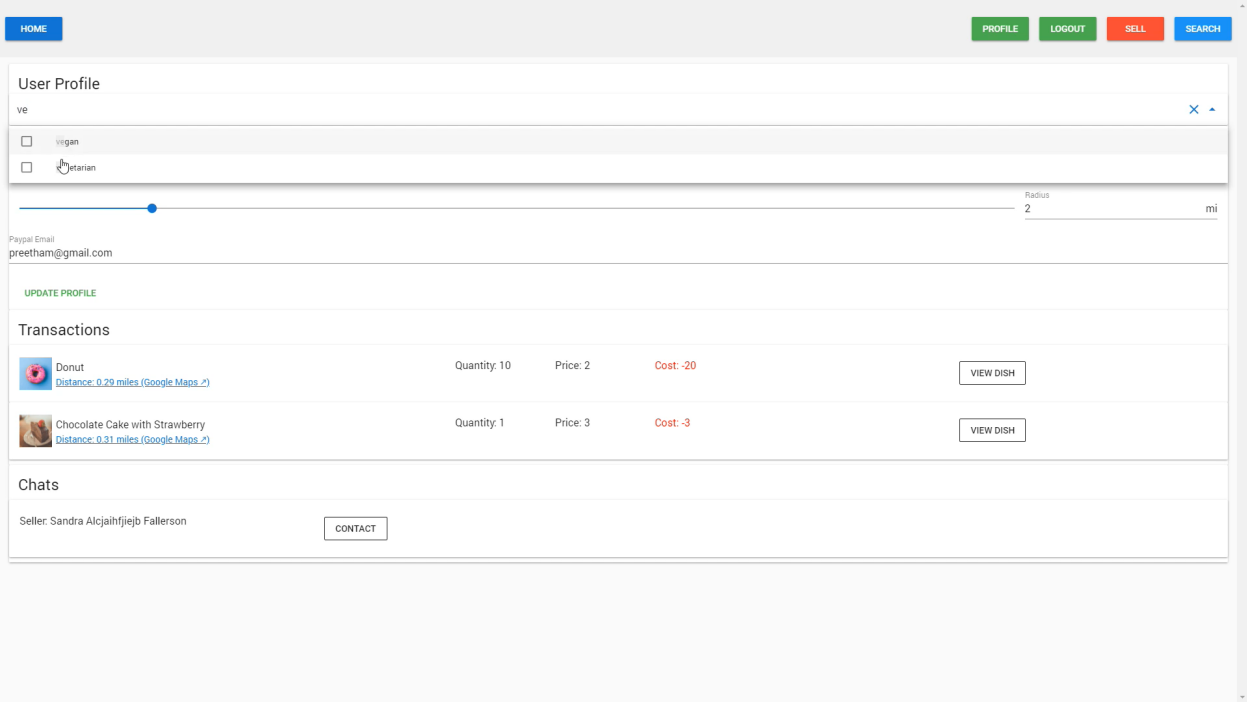

Figure 5: Let sellers know your dietary restrictions so you can eat with ease.

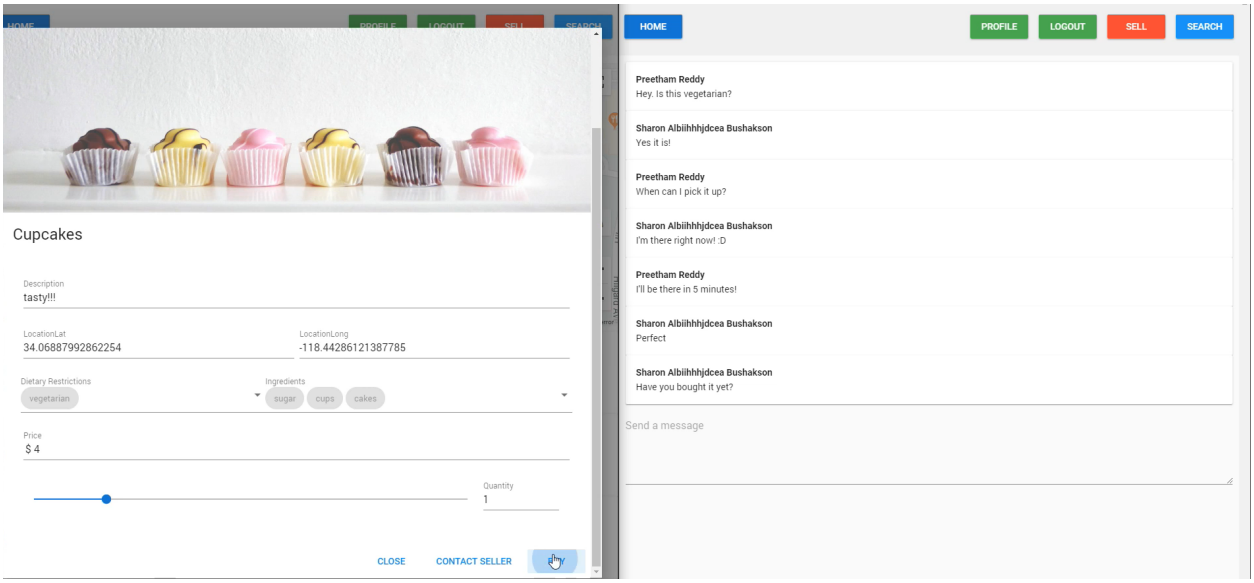

Figure 6: Find your food, chat with the seller and buy it instantly

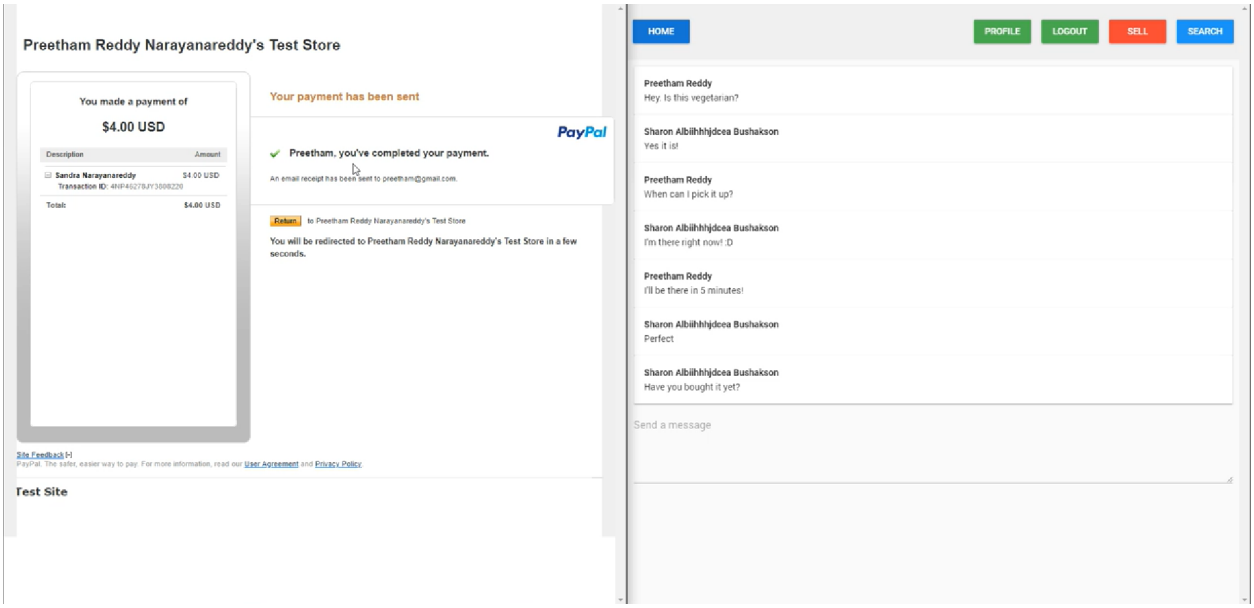

Figure 7: Pay securely using PayPal

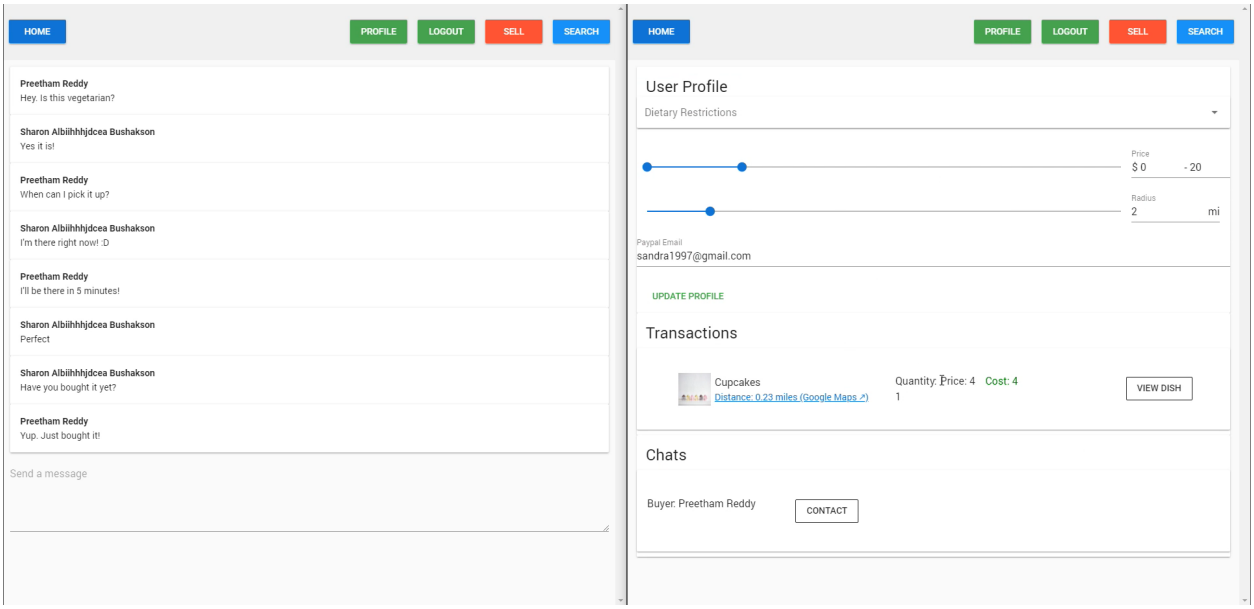

Figure 8: Sellers receive their money instantly

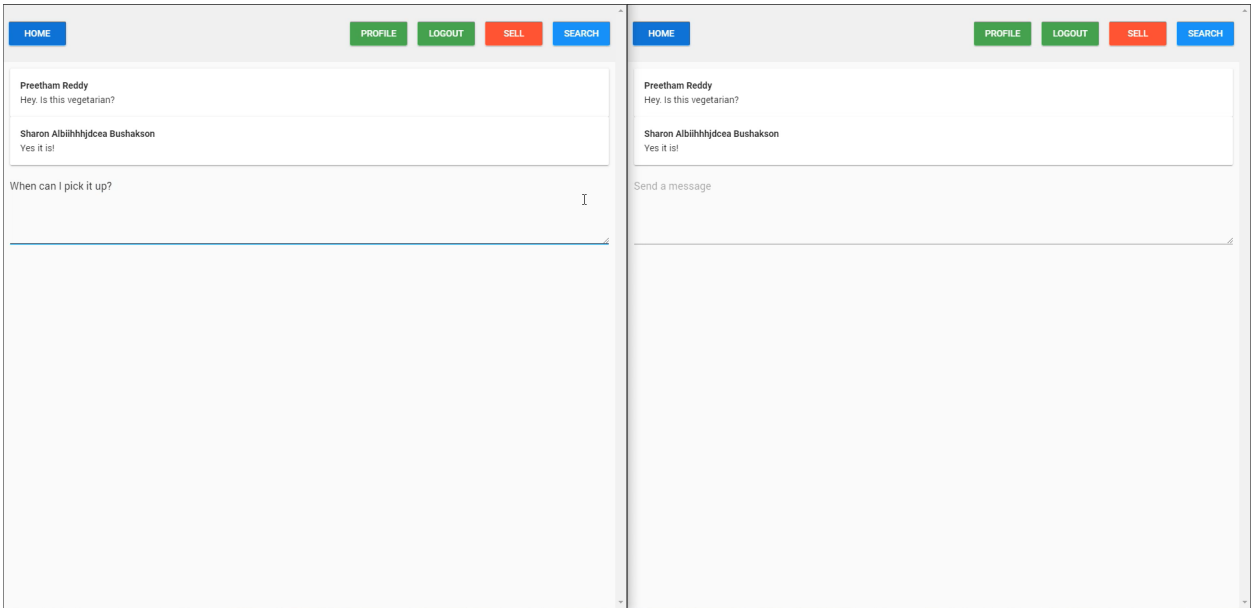

Figure 9: Chat with your sellers with ease

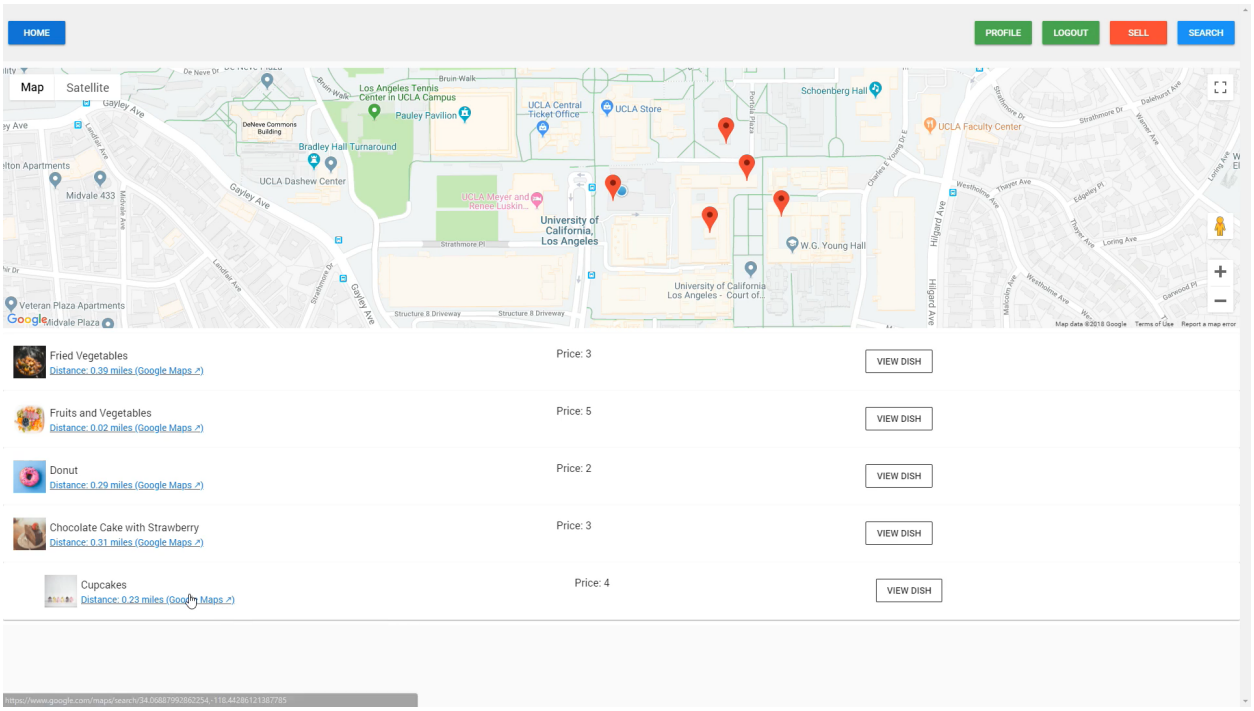

Figure 10: Know exactly how far food is

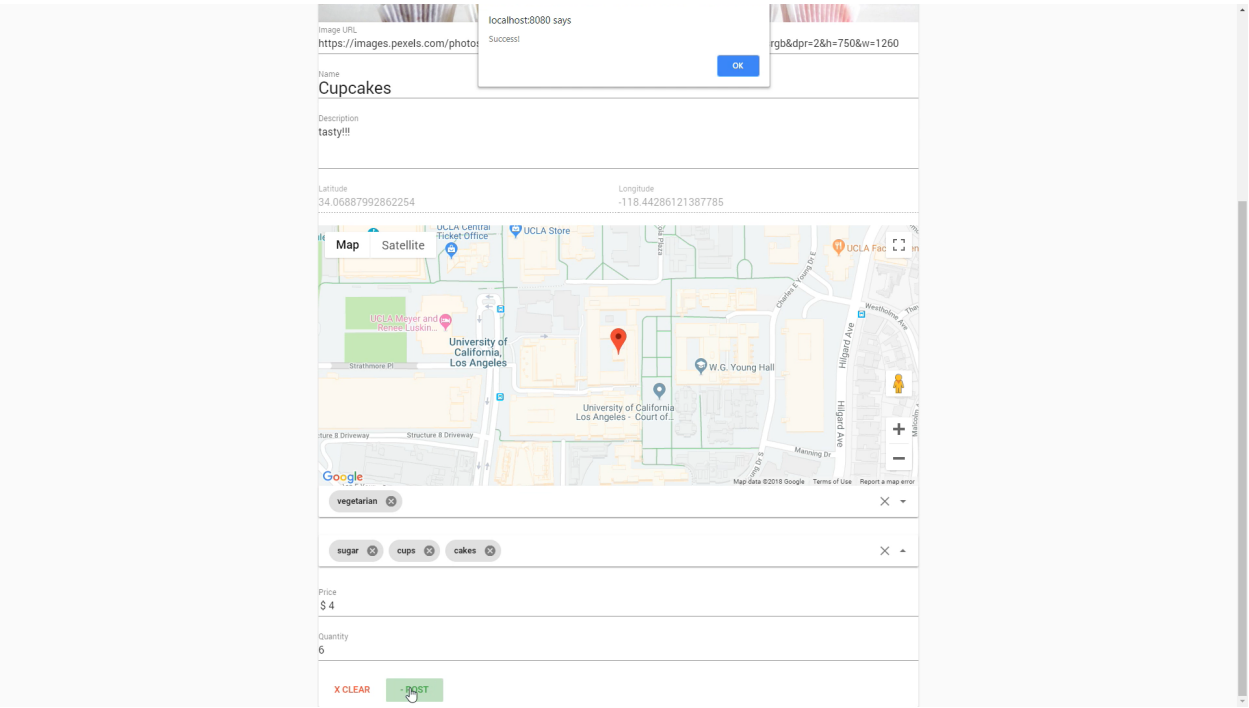

Figure 11: As a seller let the buyer know the ingredient's, price and where you are

7 Test Scenarios and Test Cases in [Jest] and [Selenium]

7.1 Backend

Here is the link to the unit and integration tests for the backend:

`https://github.com/preethamrn/fooDrop/tree/master/server/test`

Instructions: To run tests, do

```
mocha *.js
```

7.2 Frontend

Here is the link to see all of the unit and integration tests for the frontend:

`https://github.com/preethamrn/fooDrop/tree/master/client/tests`

7.2.1 Unit Tests

We used the jest framework with vue test-utils for unit testing on our front end.
When using the jest framework, we had to create mock versions of Axios so that we do not continually send HTTPS requests whenever we make a new test.
Additionally, we had to create a mock router with jest to reroute mocked components to a new endpoint.
From vue test-utils, we used mount, shallowMount, and createLocalVue.
We used shallowMount when we wanted to mount isolated components, wherein we do not want/need to render its child components.
When we are using a component that will interact with other components and would need us to render its child components, we would use mount.
When we test a generic component suite that may get shared across different apps, it was better to test the components in a more isolated setup, without polluting the global Vue constructor, so we used createLocalVue to create a local instance of the the global Vue.
All of these unit tests suites are located in the tests/unit folder in the client, with the '.spec.js' file types.
You can run the tests by going into the client directory and entering: 'npm run test:unit'. This will run all of the 'spec.js' test files in the 'tests/unit' folder, and tell you how many of the test suites passed and failed. We have 6 test suites, and 14 tests altogether.

7.2.2 Integration Tests

In order to perform integration tests on the application, we used the testing framework, Selenium.
To streamline the process of creating tests, we used the Selenium IDE, which is a browser plugin.
Selenium IDE allowed us to view the elements and components and write tests around them, as well as record our actions in order to replicate them.
Selenium IDE also has the feature of showing exactly which command breaks the test, thus allowing for easy debugging.
All the integration tests are stored in a file called Integrate Tests.side.
In order to run the test script, one must first install the Selenium IDE on the browser.
It is recommended that Chrome browser is used as it allows the use of http to get location data.
When Selenium is installed, launch it through the browser and open Integration Tests.side.
Once the tests have been loaded, locate the test suite tab and select interface to view all tests in order and the details of each test.
Selenium IDE allowed us to quickly create integration tests that mimicked users, however there are some downsides to using it.
The main downside is that Selenium IDE does not allow use to create a precondition environment before executing the tests.
This means in order to run the tests, the collections in foodrop database (MongoDB) need

to be dropped beforehand.
Another issue is that if the logout test in the end fails, we needed to manually logout of Facebook. The following is a rundown of 3 test from the list of 15 integration tests. To view the rest of the tests please go the link:

`https://github.com/preethamrn/fooDrop/blob/master/client/tests/integration/Integration%20Tests.side`

Test#1: CreateDish

In this test Selenium simulates a button press on the Search button in order to redirect to the search page. Once on the search page, Selenium looks for the appropriate input elements, such as Name, quantity, price and fills them out with valid data. Selenium then places a marker on the Google Map feature in a location within 2 miles of the Tester's current location. Once all parameters are filled out, the post is submitted and Selenium asserts for a 'Success' alert. The purpose of this test is to see the interaction between the Vue components on the Search page, Google Maps and the calls to the create_dish endpoint.

Test#2: UpdateUserDistance

In this test Selenium simulates a button press on the Profile button. It then checks if we were redirected to the user's profile page. If the proper elements for the user profile is found, Selenium changes the user's default radius to 15 miles. It then simulates a button press on the 'home' button to redirect us back to the home page. Once the home page is loaded, Selenium makes an assertion on whether a second entry is now listed with a larger radius or not. The purpose of the test is to check if the Header Vue component can redirect the user between the home and profile page, the profile page can make successful calls to the user_update endpoint, and if the home page can make successful calls to the get_dishes_by_radius endpoint and properly display the data.

Test#3: Login

In this test Selenium opens up the login page and clicks the 'Sign In With Facebook' button at the center of the page. This opens up another window prompting the user for Facebook credentials. Selenium inputs mock user information provided by Facebook in order to log in. If successful the window should close and the user is redirected to the home page. From here Selenium makes an assertion on the Header component to determine if the user is at the home page. This test is important as it tests the entire authentication flow from the frontend and the backend. The initial button press checks if the Facebook javascript SDK is properly loaded and if the Vue component can call the Facebook login function. Submitting the credentials tests the /auth/facebook endpoint in the backend and whether the Vue components can properly handle the user information.

8 Contributions

Karthik Ramesh: He worked with Farhan and Preetham on the Google Maps API. He also continued front end unit testing from part B with Preetham to test more features the team added into the application in part C using the jest framework, and vue-test-utils. Finally, Karthik worked on feature/requirement description, and UI screenshots for the report.

Preetham Narayanareddy: He worked on integrating PayPal with the transaction endpoint and building the chat feature of the application. He also worked on unit testing the application with Karthik, and fixing bugs to help make the application production ready.

Udayan Sahai: He worked with Leslie on Paypal integration and was responsible for coding JavaScript implementation to create and complete a transaction between two parties.

Leslie Liang: Leslie worked with Udayan on PayPal integration. He explored the PayPal documentation and decided with Udayan that the Adaptive Payments API should be the approach. He also wrote a backend integration test for the chat endpoint using mocha, chai and chai-http. Finally, Leslie also worked on revising the usage scenarios for the Part C report.

Farhan Alam: Farhan worked with Karthik and Preetham to implement Google Maps API and to uncover any bugs in the frontend. He also worked on utilizing Selenium IDE to create thorough integration tests.