Home Foods Delivery App

Prepared By

Group 2: Krish Bavana, Soham Patel, Hari Seelam, Ish Soni

CS 440 at the

University of Illinois Chicago

Spring 2022

Table of Contents

List of Figures

List of Tables

There are no tables in this report.

Project Description

Project Overview

The Home Food Delivery mobile application will provide a way to deliver fresh homemade food to one's doorstep (like Uber or Door Dash). Cooks will prepare meals from their personal kitchen for customers who desire home-cooked meals, and drivers will deliver the meals to the customer's home.

Project Domain

For the Home Foods Delivery App, we have implemented a general ordering structure for the 3 main users of the app, the cook, customer, and driver using Flutter as our codebase and SQL and AWS for our database backend.

Cook

The cook can set up shop and can add items to their menu and be linked to a marker on google maps where customers select to see their menu to begin ordering. After customers have placed an order to their restaurant, the cooks will be able to see a list of orders placed and other details regarding the customer (notes for special dietary accommodations, time requested, etc.) and begin to fulfill the requests.

Customer

The customer can view a list of cooks in the nearby area in a list view or on google maps as markers and select a cook to view their menu and place an order. Moreover, the customer also has the option to add or delete items after adding to their cart, and even placing a note for special dietary accommodations before placing their order.

Driver

When the driver is active and ready to deliver orders, they can toggle an on/off button to show a list of available orders ready to deliver. Moreover, the driver also has access to the google maps and can view all the available cooks in the area. Finally, the driver also has access to the current weather in their location through the weather button on the navigation bar.

APIs

We have incorporated the google maps and open weather APIs in this app as well. The google maps serves as a visual way for customers and drivers to view cooks and the weather enables drivers to take safety precautions before proceeding with deliveries.

What this project has not yet implemented:

We have not yet implemented the Gas Price API due to a lack of one available for the United States, as well as the payments system for cooks, customers, and drivers.

Relationship to Other Documents

The project was completely inspired based on the "Home Foods Delivery" documentation project from Group 26 of CS 440 Fall 2021 semester. This document helps explain all the important components we have developed from group 26. Our document provides information on what changes we have made to the original idea and how we adapted it.

Naming Conventions and Definitions

Definitions of Key Terms

Customer – A person who orders and pays for food from the cook and receives it from the driver.

Cook – The business owner. This person receives orders from the customer and cooks it before handing it off to the driver.

Driver – The person who delivers food to the customer from the cook. They will have their own mode of transportation.

Menu – A set of food items that a cook has available for purchase. Each cook will have their own distinct menu.

Order – A set of food items that a customer purchases from a cook.

Purchase – Paying for the food. An order is considered purchased when the customer clicks on the "Place Order" button on the final order page.

UML and Other Notation Used in This Document

We used an auto generated UML diagram for our database by MySQL. There are a few distinct symbols that are important to note. The key icon indicates a primary key in a table. The blue diamond is a normal column field. The orange diamond signifies a foreign key. If the diamond is filled in, then the data field cannot be null. If the diamond is not filled in, it can be null. The arrows connecting the tables signifies a foreign key relationship.

Data Dictionary for Any Included Models

Cook

- Menu
- Menu item
- Order
- State ID

- Business name
- Average rating

Customer

- Customer ID
- Order ID

Driver

- Driver ID
- License ID
- License Plate

Orders

- Order ID
- Menu ID
- Item
- Quantity
- Order Progress
- Cook ID
- Driver ID
- Customer ID
- Delivery Date
- Order Date
- Special Request

Project Deliverables

First Release

This first scenario focused on user (customer or cook) account creation and login. Customers will be able to view the catalog of cooks available in their area and their

menus. The customer will then be able to select menu items without any regards to purchasing. When the cooks sign up, they will be able to add their availability and menu which will be uploaded into our database.

We also implemented the database structure that was used for this release as well as future features (such as ordering). The database contains information regarding the users and their personal information. If they are customers, we will be able to fetch data from the cook's database and display the menu items available at said cook's "restaurant".

We also completed the skeletons for most of the classes that we needed for the app and UI features such as the task bar and drawer menu.

Figure 1- Login screen developed during Release I

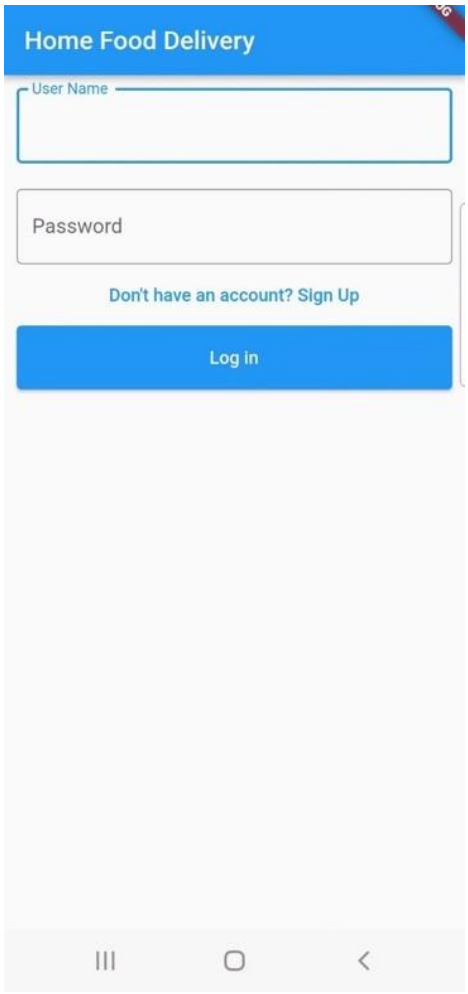
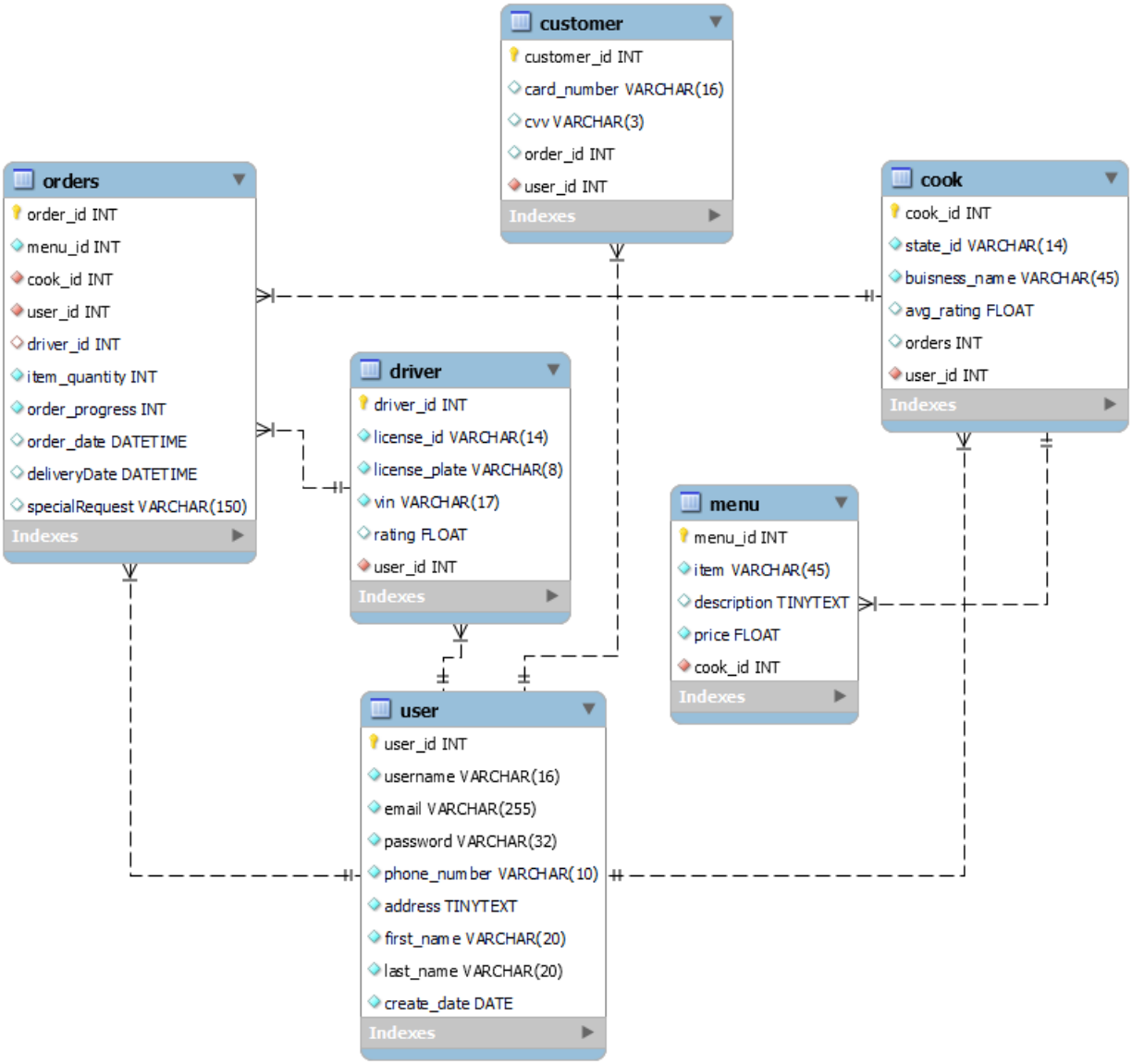


Figure 2 – UML Diagram for the Database Tables, Autogenerated by MySQL Workbench



Second Release

The second release was focused on Google Maps API, Weather API, and Ordering System. One of the biggest challenges we faced was designing the ordering page UI because it contains many details like item name, the quantity of the item, descriptions, prices, the option to increase the quantity and decrease the quantity. All the information on the cooks' page is dynamically set based on the cooks' options and data is fetched from the database. We have used shared preferences to send the data of the customer's

choice from the cook's order page to the final order page. Even on the final order page, the customer can change the item quantity and delete the item.

We have also implemented two APIs in this release, one of them is Google Maps API. In the google maps API, we were able to add the pins to the cook's location and if the customer clicks on them we were able to redirect the maps onto the cook order page. The other API we have developed was the weather API where we were able to fetch the real time weather of the driver's location.

Apart from this we were able to add minute features like toggle button for the driver's page and the template for the gas prices.

Figure 3 – Google Maps API & Cooks Menu Page developed during Release II

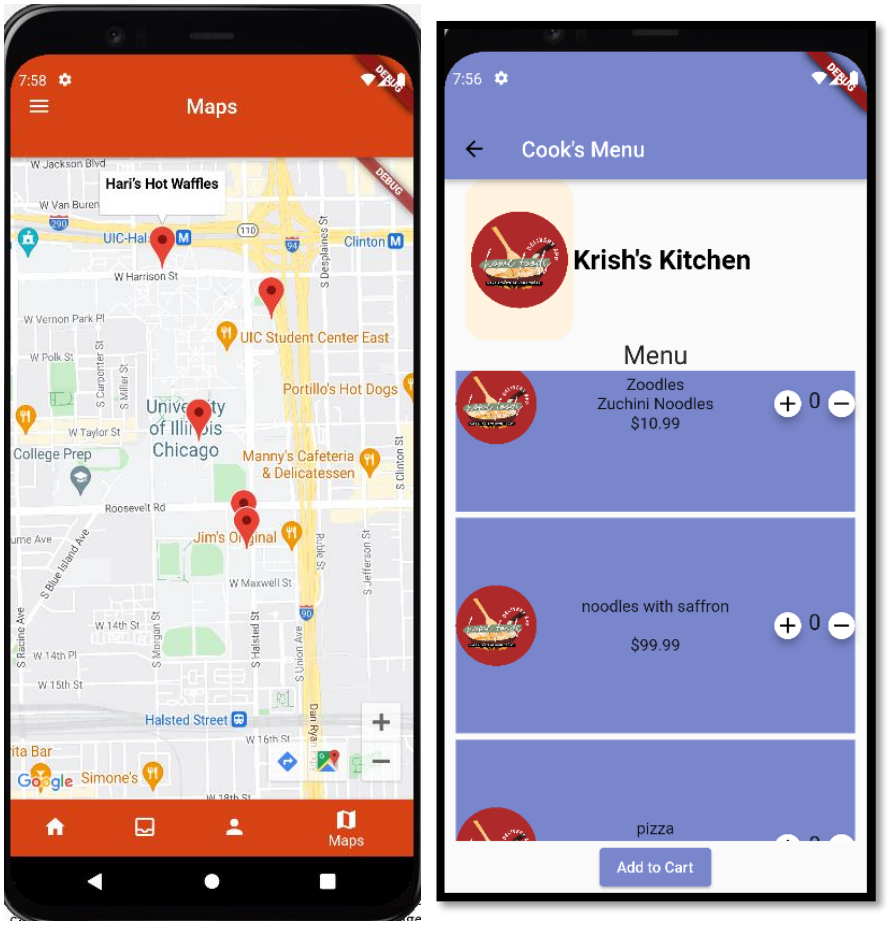
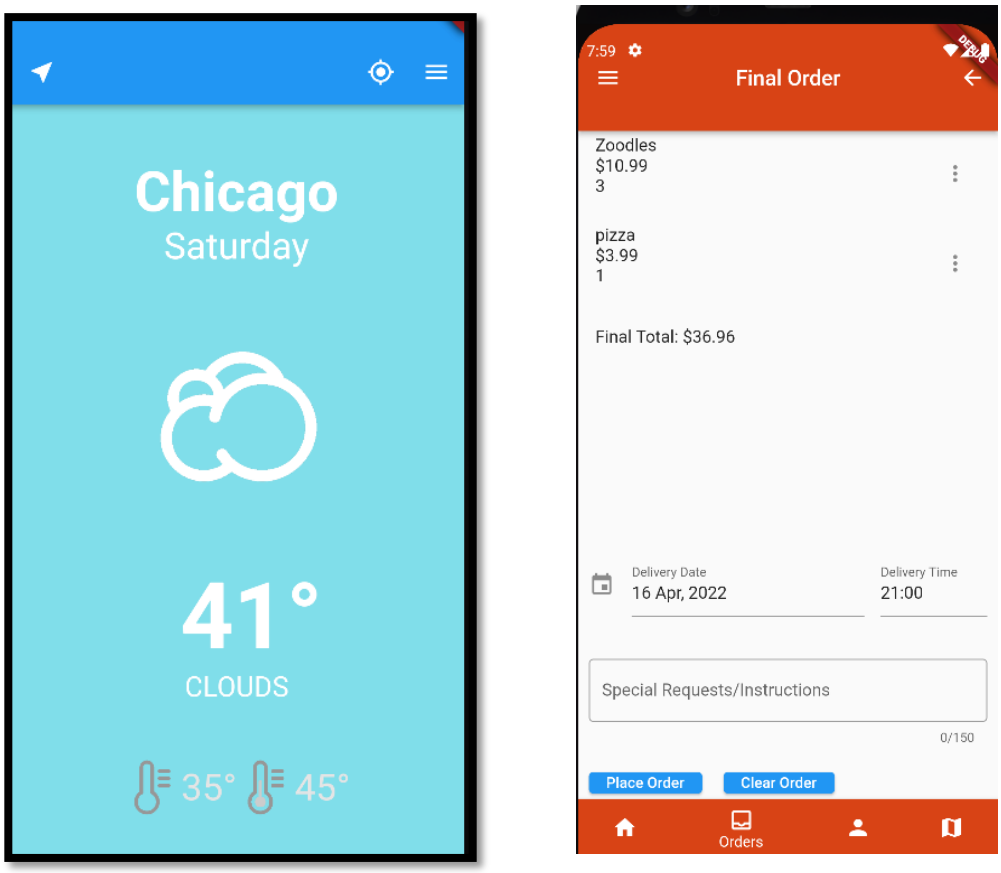


Figure 4 – Open Weather API & Customers Final Order Page developed during Release II



Comparison with Original Project Design Document

Our project was inspired by Group 26 of CS 440 Fall 2021 semester. There are many similarities and many differences in comparison to their prototype. I will be referencing section numbers from their final report throughout this analysis.

We kept their idea intact, as we have three distinct users of the app: customers, cooks, and drivers. All of these users have a similar UI structure when we compared our design to the final block diagram in section 25f of the report. There are slight differences in layout, but the structure is similar. A customer can order from a cook, and the cook will be able to view the order on their account. The customer is also able to view cooks using the Google Maps API. Additionally, the driver can view the weather of their current city using a weather API. All of this is reflected on the Scenario Diagram in section 4a. For database, we agreed with using SQL as suggested. Finally, the cooks are able to use the application offline as mentioned in section 17 since the app does not require Wi-Fi connection.

The major difference regarding the prototype and our project is the ideas for solution in section 35. Group 26 suggested we use Swift and XCode for iOS development and

React Native for android development. Instead, we used Dart with Flutter framework. This supports both iOS and Android, while writing on just one code base. This gives the app a consistent look no matter what device or emulator we use. We all used Visual Studio Code and Android Studio for development. Additionally, there were a couple of ideas proposed that were not developed. First is messaging between the driver and the customer, which is in section 6a. The second is the EIA Open Data API regarding gas prices for the driver in section 6c. We attempted to implement this; however, we were unable to due to limited options for this API in the United States. Finally, we were unable to incorporate password encryption as suggested in section 15b. We wanted to implement this; however, we were unable to allocate time to this since we had a limited timeline to create the app. This would be a priority before releasing the app due to security concerns regarding password data.

Testing

Items to be Tested

- *Cook Order Page & Final Order Page*
- *Google Maps API*
- *Weather API*
- *Login Page / Sign Up Page*
- *Driver Mode*

Test Specifications

1 – Cook Order Page & Final Order Page

Description: Checking if an order by the Customer can be reflected on the Cook UI as well as a separate final orders page on the customer's side.

Items covered by this test: Placing Order page**,** Final Order Page, Cooks Current Orders Page.

Requirements addressed by this test: Placed Order Test on Page 29.

Environmental needs: N/A

Intercase Dependencies: Testing if the user can add items to their cart without hitting place order button.

Test Procedures: The customer logs in and clicks on a cook. They add items by using the plus button next to each menu item they want to order. When they are ready, they hit add to cart. They will then be redirected to their final orders page which shows each menu item they ordered and the quantity. The customer can

adjust their order by using the buttons on the right side of each menu item. Once they are done, they hit place order. The cook now hits the refresh button on their account. They will see a list view item that has the customer's name. They click on it and will see the items the customer ordered.

Input Specification: Items added to the cart for a cook by the customer.

Output Specifications: The Customer can view all their menu items purchased as well as a subtotal in the final orders page. The cook can view this order on their current orders page once the customer hits the place order button on the final orders page.

Pass/Fail Criteria: The cook can view the exact customers order on their view.

2 - Google Maps API

Description: Testing if the functionality of the Google Maps page on the driver page accurately shows cook markers.

Items covered by this test: Google Maps API

Requirements addressed by this test: Page 47, Driver Routing Test. Also mentioned on page 13 and 16 that a Google Maps API should be implemented.

Environmental needs: N/A

Intercase Dependencies: N/A

Test Procedures:

1) Log in as a user that is also signed up as a cook.
2) Go to the map and check you can see the cook markers.
3) When you click on them you will get an info box that gives the business name, if you click on the box it will bring you to the Cook Menu Page.

Input Specification: A list of cook addresses.

Output Specifications: A marker of each cook provided. Each marker also contains the cook's menu as well.

Pass/Fail Criteria: The Google Maps should be populated with markers of all the cook's addresses and their menu information.

3 – Weather API

Description: The available orders showed only when the toggle button is on.

Items covered by this test: Open Weather API

Requirements addressed by this test: Page 16 and 52, which mentions the Weather API. As mentioned on page 16, the open weather API "will be needed for current location weather conditions to compute travel challenges."

Environmental needs: N/A

Intercase Dependencies: N/A

Test Procedures:

1) Please check if the initial weather UI is working properly and if the search for the weather is accurate as well.
2) Press the City button and check to see if the weather UI updates properly to show the weather for the city that you typed in.
3) Press the arrow on the top left corner. The UI should update with the weather of the current city you are in.

Input Specification: Coordinates of current location or city name. Also, location permission must be enabled.

Output Specifications: The current weather in Fahrenheit for the selected city. You will also be able to view the high temperature and low temperature for the current day.

Pass/Fail Criteria: You can view the weather for the current city. You can also change the city by searching for a city or by hitting the current location button. In either case, the UI should change to show the weather for the reflected city.

4 – Login Page / Signup Page

Description: Testing login and signup functionality.

Items covered by this test: Login Page, Signup Page

Requirements addressed by this test: Being able to sign up and login to the app is integral to the functionality of the app.

Environmental needs: N/A

Intercase Dependencies: N/A

Test Procedures:

1) Sign up using Signup page.
2) Log in with the account you just made
3) Try signing up with the EXACT SAME username as the account you made earlier
4) Try logging in with a random username
5) Try logging in with correct username but incorrect password

Input Specification: Valid and Invalid login credentials, valid signup credentials, use a duplicate username

Output Specifications: Signup: Valid insertion of data into the database, Login: Successful login and customer view shown

Pass/Fail Criteria: Signup: Database now includes the user data. User can now login with the credentials they signed up with. Login: Successful login and customer view shown

5 - Driver Page

Description: The available orders showed only when the toggle button is on.

Items covered by this test: Position result of the toggle button and the size of the available orders.

Requirements addressed by this test: The major requirements are to check the action of toggle button on driver button.

Environmental needs: N/A

Intercase Dependencies: In the current version, there are no intercase dependencies but in the coming tests, the list should only present completed orders by the cooks

Test Procedures: Test Based on the toggle button by turning on and off. . .

Input Specification: Toggle Button

Output Specifications: Show Available Orders

Pass/Fail Criteria: To pass the driver's test, the toggle button has to work and activate the list.

Test Results

1 – Cook Order Page & Final Order Page

Date(s) of Execution: March 31st

Staff conducting tests: Krish

Expected Results: The customer can place an order and the cook can view it on their login when refreshing the page. The customer can also look at the final orders page when they hit the add to cart button.

Actual Results: Expected Results

Test Status: Pass

2 - Google Maps API

Date(s) of Execution: April 1st 9:00am

Staff conducting tests: Krish Bavana

Expected Results: The Map with the Pins to kitchens available nearby.

Actual Results: The Map with the Pins to kitchens available nearby.

Test Status: Passed all the tests

3 – Weather API

Date(s) of Execution: April 1st 12:15am

Staff conducting tests: Ish

Expected Results: Accurately shows the weather (high, low temperature) in the city they are currently located in or searched for.

Actual Results: Expected Results

Test Status: Pass

4 – Login Page / Signup Page

Date(s) of Execution: Feb 8th 3:00pm, 6:00,pm 10:00pm

Staff conducting tests: Hari, Krish, Soham

Expected Results: User can successfully create an account and login to the app.

Actual Results: Expected Results

Test Status: Pass

5 - Driver Page

Date(s) of Execution: March 31st 9:00pm

Staff conducting tests: Krish

Expected Results: We expect the toggle button functionality to work.

Actual Results: The toggle button functionality worked as expected.

Test Status: Pass

Regression Testing

N/A

Inspection

Items to be Inspected

- *Cook Order Page & Final Order Page*
- *Google Maps API*
- *Weather API*
- *Login Page / Sign Up Page*
- *Driver Mode*

Inspection Procedures

Cook Order Page & Final Order Page

1) Login
2) Click search on search cooks and click on a "restaurant"
3) Add items to cart and click add to cart
4) Finalize cart and click place order. You will see the page clear of items and you will get 2 toaster notifications (Placing Order... and Order Placed!)
5) Check database to see if items were added.
6) Add another order and make sure clear order button clears the screen. Also check that clicking Place Order when there are no items in the cart will not do anything.

Google Maps API

1) Log in as a user that is also signed up as a cook.
2) Go to the map and check you can see the 3 cook markers (TBH, Naperville, Aurora is where they're located).
3) When you click on them you will get a info box that gives the business name, if you click on the box it will bring you to the Cook Menu Page.

Weather API

1) Please check if the initial weather UI is working properly and if the search for the weather is accurate as well.
2) Press the City button and check to see if the weather UI updates properly to show the weather for the city that you typed in.

3) Press the arrow on the top left corner. The UI should update with the weather of the current city you are in.

Login Page/Sign up Page

1) Sign up using Signup page.
   a) Also check it shows up in MySQL.
2) Log in with the account you just made (you will end up at the customer page)
3) Try signing up with the EXACT SAME username as the account you made earlier (Should get popup saying username already exists)
4) Try logging in with a random username (Should get popup saying username or password is incorrect)
5) Try logging in with correct username but incorrect password (Should get popup saying username or password is incorrect)

Driver Mode

1) Login and go to the driver view.
2) You should see a toggle on the top of the screen that it set to off.
3) Click on. You should see a list view pop up underneath**.**
4) Click off. The list view should disappear.

Inspection Results

Cook Order Page & Final Order Page

- Code Reviewer: Krish
- Time and Date: March 31st 9:00 pm
- Results: Passed all tests, no modifications needed

Google Maps API

- Code Reviewer: Hari
- Time and Date: March 9st 1:43am
- Results: Passed test. When tested with iOS, she could not zoom out to view all the markers. Not a bug but can be improved at a later time.

Weather API

- Code Reviewer: Ish
- Time and Date: April 1st 12:15am

- Results: When you searched for a city, the weather shows up properly, but it is displayed in Celsius. Change it to Fahrenheit, and the rest is perfect!

Login Page/Sign up Page

- Code Reviewer: Hari, Krish, Soham
- Time and Date: Feb 8th throughout the day
- Results: Passed Test, no modifications needed

Driver Mode

- Code Reviewer: Soham
- Time and Date: March 31st 8:11am
- Results: Passed Test, no modifications needed

Recommendations and Conclusions

All the tests mentioned above have passed their testing inspection and process. Further testing and implementation is not needed.

Project Issues

Open Issues

As of now, we are still unsure of how to implement an up-to-date gas price indicator for the drivers as there is no Gas Price API currently on the market in the United States based on our research. While this doesn't hinder the development of The Home Foods Delivery App, it was going to be incorporated into the earnings portion for the driver.

Currently, our solution is to just place a Gas Price template onto one of the pages while we try to find a solution.

Waiting Room

An idea that we can implement later is having the cook set hours of operation for their business. As of right now, the cook can receive orders at any time of day. This is not realistic of how a business run. We would have the cook set their business hours, and if the current time is within that time, then the customer would be able to view that cook when they are looking at the available cooks. We implemented a similar idea with the driver where they can toggle a button on or off to signify that they are available to deliver orders. This has a low cost of implantation with high benefits since development of this idea is not difficult for our development team.

Another idea is to have the cook be able to edit menu items. The reason a cook would want to do this is to either change the price, or to edit how the food item is being prepared. This would be a very low cost of implementation since it is one simple button that would run an update statement in the database, with medium benefits as menu items may not be edited often.

Another idea that we could implement later is a popup notification on the customer's device regarding the status of their order. There will be a notification regarding the order being cooked, out for deliver, and being delivered. On the driver side, they can get a notification when there is an order ready to be delivered within one mile of their current location. They can also get notifications if they forget to mark an order as delivered. On the cook side, they can get notifications every time an order is placed to them as well as whenever they get any feedback from the customer. Implementing all of these notifications will have a high cost since we need to incorporate the Google Maps API with the driver notification. This will have a lot of benefit to the users since popup notifications on their device is a very easy way to communicate with the customer.

Finally, we can develop logic so that the customer can track the driver when the food they ordered is out for delivery. In this release, the customer already knows a delivery time for their order since they had to specify a delivery time before placing the order. However, if the customer wants to know the status of the delivery, then they can go to the maps tab from the taskbar and view the route of the driver. This is a high cost of implantation since we need to use a GPS API and integrate it with our current Google Maps API. This will take the development team a longer time compared to other items in the waiting room. This will have the highest benefit since we anticipate most customers to view the driver status at least once during the delivery.

Ideas for Solutions

Some of the other suggestions are changing the system design of the users. The user class can contain all the important fields rather than each one of the classes containing the same exact field.

To make the application smoother and same on all the devices, using dart with flutter framework is a better implementation. With the help of flutter, we don’t have to maintain three different code bases for each type of the device.

For the Gas Prices, it is better to use Guide API or Collect API because they give accurate results for the regions in Northern American.

For the Database, it is a good idea to stick on to SQL because all the data which is collected during the activity is structured data, so tables are better option compared to JSON or XML.

Project Retrospective

Having two branches for the git worked well (master and development), but it would have been better if we each had our own branch to work on and then merge to master rather than all of us using the development branch to do our work. Doing that caused problems with some pulls from the repository would overwrite some of our work so we had to do more work to coordinate the files we were working on and our pull requests to the master branch.

We could have put more research into the packages that are available in Flutter so certain aspects of the app would have been easier to develop. For example, we spent time creating a login screen only to find out there is a package that would have streamlined the process. Had we have known that we could have spent more time on developing other features.

The weekly Jira and Scrum meetings helped us organize the work that had to be done for the week as well as the work distribution throughout the team. The meetings also helped us avoid communication issues and keep the team on the same page.

Glossary

Android Studio: An integrated development environment for Google's Android operating system

API: Application Programming Interface, allows two separate programs to interact with each

other

AWS: Amazon Web Services, public cloud offering

Dart: A programming language designed for client development, such as for the web and mobile apps

EIA Open Data: Data provided by the U.S Energy Information Administration in the form of an API. This data is free and open sourced.

Flutter: An open-source UI software development kit created by Google. It is used to develop cross platform applications for Android, iOS, Linux, macOS, Windows, Google Fuchsia, and the web from a single codebase

Google Maps API: API for management of locations and routes

MySQL: open-source relational database management system

Open Weather API: API for live weather conditions in specific locations

Public Cloud: Computing Infrastructures managed by a third-party provider

Private Cloud: A cloud computing consisting of hardware and software dedicated to one single organization

VS Code: Source-Code editor made by Microsoft for Windows, Linux and macOS

XCode: IDE for MacOS and iOS development

References / Bibliography

[1] H. Habubullah, H. Patel, J. Garg, "Home Foods Delivery" Software Engineering (CS 440) UIC Group 26 Development Project

Index