CSCI 401: Project Harvest

Complete Documentation

Team Members:

Zixuan Li

Lihan Zhu

Denny Shen

Jiayang Li

Table Of Contents

Project Description

Here at Harvest, we believe that farm fresh should be for everyone. Harvest aims to deliver the best quality produce right to your door no matter the weather, location or time. We work directly with Farmers' Markets and local farmers to provide Angelenos a convenient way to order fresh and local produce as easily as any take-out meal.

Harvest is an iOS app that provides farm produce delivery service, similar to other popular apps such as "Postmates" and "DoorDash." Harvest was founded by 4 Iovine and Young Academy students at USC. Over six months, the team has been conducting research, connecting with business partners, testing the product, and preparing clickable prototypes.

As the Harvest development team, we, 4 Viterbi students, designed and developed two applications for Harvest--customer side and business side--where the business version provides service to the farm vendors and the drivers.

The major features of Harvest include:

- Users can browse different markets and choose from available vendors, browse selection and edit quantity, add to cart and pay for product and delivery fee.
- Users can schedule delivery date/time if markets are not open.
- Users can receive updates on delivery, message and tip drivers.
- Drivers can receive the orders nearest to their current position and accept/decline orders based on their needs.
- Drivers can check the active order list and order details when picking up orders in markets.
- Drivers' total driving time and total earnings for the current session is available in Homepage.
- Delivery driver account with integrated map navigation and linking direct payment to their bank account.
- An order is automatically proceeded to the next step when drivers arrive at the designated position.

Link to wireframes:
https://www.figma.com/file/sJg3uPg0LziiKxPTS9M2LS/Updated-Mocks?node-id=0%3A1

Application Overview

The entire produce delivery service follows the flow of placing orders, accepting orders, and delivering orders. In order to access the data more efficiently, we have 6 core collections: customers, farms, markets, drivers, chats, orders. Since there are two versions of the app, the customer version, and the business version, we decided to use separate collections for customers, vendors(farmers), and drivers. As the farmers would sell their produce in markets, each market would have a list of farms that were located in it. Every time an order was placed, an order object would be created in the order collection with a status tag to wait to be accepted and delivered by a driver. Chats collection is responsible for the messaging functionality between a customer and a driver connected through an accepted order.

Sign up/sign in is required in both applications to access all the features, where the users will provide their email address, phone number, password, and agreement to the policies. Though as an optional field in the sign-up section, both customer and business side users will need a valid card to pay and get paid. The same applies to the user location, though optional, both versions of apps require access to user location to be fully functional (mostly for the driver side for real-time location updates).

The core functionality of the customer app will be placing orders. After adding a valid address, the major flow would be browsing and choosing a market, browsing and choosing multiple farms within the market, adding produce from these farms to the cart, checking out, and then waiting for a driver to complete the delivery.

The core functionality of the business app will be delivering orders. After the driver chooses to go online, he/she will be actively searching for currently available orders. The orders will be retrieved with shorter distances prioritized. It is possible for one driver to accept multiple orders from the same market but different customers (only orders from the same market will be retrieved together). The major flow would be accepting orders, driving to the market, purchasing items on the shopping list, driving to the customer to complete the order.

Data Structure

Firebase Firestore is used as the database across the customer and business apps. In terms of data structure, our application uses a combination of nested data in documents, subcollections, and root-level collections. More detail about each type of structure: https://firebase.google.com/docs/firestore/manage-data/structure-data.

The root-level collections are customers, drivers, farms, orders, markets, and chats. Each collection consists of documents in the same format, as shown in the following [field_type: field_name] manner:

Customer document:

- string email
- string first_name
- string last_name
- string phone_num
- string image_url
- string stripe_customer_id
- string stripe_setup_secret
- address active_address
- [address] saved_addresses
- collection orders
    - timestamp date
    - ref market_id
    - number num_of_items
    - number total_cost
- collection payments
    - string order_id
    - StripeObj payment
    - string transfer_group
- collection payment_method
    - StripeObj payment_method

Driver document:

- string email
- string first_name
- string last_name

- string phone_num
- number lat
- number lon
- string current_session
- collection delivery_sessions
    - string market_id
    - address market_address
    - [string] order_ids
    - number step // delivery status
    - collection orders
        - [[string:any]] order
            - string farm
            - [string: any] shopping_list
                - string produce_name
                - number num
                - number unit_price
                - string image_url
        - string customer_first_name
        - string customer_image_url
        - string customer_last_name
        - string customer_phone_num
        - address customer_active_address

Farm document:

- string name
- string category
- string email
- string first_name
- string last_name
- string phone_num
- string market

Order document:

- string customer
- string delivery_status
- string market
- timestamp order_date

- string phone_num
- number total_cost
- number total_num
- [string: any] farm_total_cost
    - number cost
    - string farm
- [[string:any]] order
    - string farm
    - [string: any] shopping_list
        - string produce_name
        - number num
        - number unit_price
        - string image_url

Market document:

- address address
- [string] day_open // All week days open
- string name
- string description
- string image_url
- string marketID
- string time_close
- string time_open
- collection farms
    - string category
    - string farmid
    - string image_url
    - string name

Chat document:

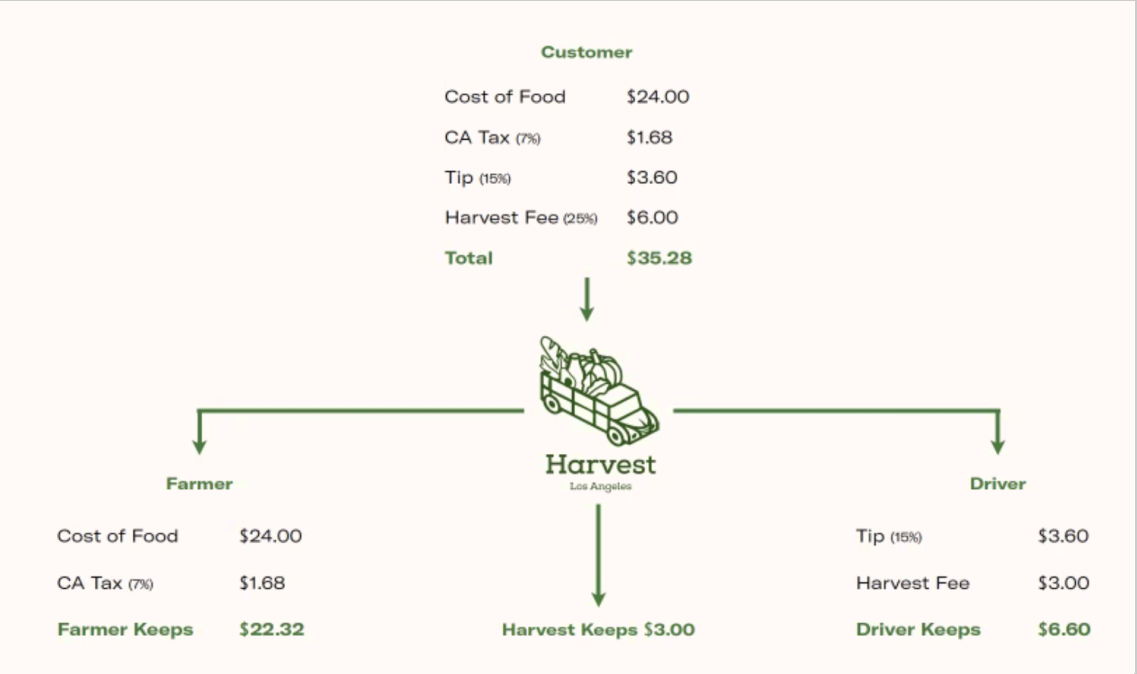
- [string] user // list of the 2 users' id
- collection thread
    - string content
    - timestamp created
    - string id
    - string senderID
    - string senderName

Address data structure:

- number lat
- number lon
- string title
- string subtitle
- string apt // customer only
- string building // customer only
- string instruction // customer only

*Please refer to the Firebase Firestore for the most updated data structures!

Harvest Economics



For each order placed by customers, the fee will be divided in the following way:

- A fixed 25% of the cost of food excluding the tax will be added as the Harvest service fee which will further be distributed to Harvest and the driver.
- A fixed 15% tip will be added to the total cost of the order which directly goes to the driver after the order has been successfully delivered.
- Harvest and the driver will each get 50% of the Harvest service fee.
- The farmer (vendor) will keep the cost of the food along with the tax paid by the customer.

Technology Specification

Hardware

iOS: any version of iPhone

Software

Xcode with Swift

Firestore

Firebase Authentication

Firebase Storage

Stripe

Google SignIn

MessageKit

Figma

MapKit

Technical Design

Order Processing

When a customer successfully places an order, an “order” object will be created and initialized with given parameters in the orders collection with a delivery_status tag marked as “created”.

When a driver goes online and searches for orders, only markets in the markets collection that are currently open will be selected and sorted in order of distance to filter out orders that are either too far or not in the operation time of its market. The first two orders found in the selected markets with delivery_status tag of “created” will be retrieved to the driver, and their delivery_status tag then be updated as “retrieved”.

When a driver accepts an order, the delivery_status tag of that order will be updated as “accepted”. Otherwise, the tag would be reset to “created” so the next driver can retrieve this order and choose to accept it or not. A “delivery session” object would be automatically created under driver collection by the ‘addDeliverySessionDetails’ cloud function, saving all necessary information about the orders and customers.

The accepted orders would be presented as a list in the Active List page. When the driver arrives at the market, he needs to mark all the products in the list as checked to complete picking up.

An integrated navigation system would lead the driver to each destination and s/he can click “complete” after arriving at the customers’ places and finishing dropping off.

After completing all the active orders, the driver could go back to the Driver Homepage, clicks “go online” and looks for new orders to deliver.

Payment

The payment system is implemented with Firebase Cloud Functions and Stripe API. Because Harvest serves as a platform between customers, drivers and vendors, our platform uses Stripe Connect to manage charging and paying out.

Before charging a customer or attaching payment methods to a customer, a Stripe Customer object has to be created to uniquely represent the customer. Upon the creation of a customer document, a Stripe Customer object is created through the ‘createStripeCustomer’ function and its id is stored in this customer document’s ‘stripe_customer_id’ field.

In order to create payments, each customer must have a payment method. Each payment method is created using Stripe Setup Intents API. Whenever a customer document is created, a new SetupIntent object will be created through the 'createStripeCustomer' function and its client secret will be stored in this customer document's 'stripe_setup_secret' field for future reuse. After the SetupIntent is confirmed on the front-end, a new payment document will be added to the 'payments' subcollection under the customer document. The 'addPaymentMethodDetails' function will be triggered to retrieve and fill in the PaymentMethod object. In the same function, a new SetupIntent object will be created and its client secret will be written to the customer's 'stripe_setup_secret' field. When a payment method is deleted, the 'deletePaymentMethod' function will be called to clean up Stripe's PaymentMethod object.

Now that there are Stripe Customer and PaymentMethod, a customer can be charged using Stripe PaymentIntent API. Whenever an order document is created, the 'createStripePayment' function will be triggered to create a Stripe PaymentIntent object with the customer's default payment method and write this PaymentIntent object into this customer's payments subcollection. In cases where cards require authentication, the front-end will check the status of the payment document and perform authentication if needed. On success or failure of an authentication, the payment document's 'status' field will be updated. The 'confirmStripePayment' function will then be triggered to perform manual confirmation.

Stripe Connect allows drivers and vendors to receive payouts from Harvest. Based on the business model of our platform, Stripe Express accounts are the choice of account. When vendors or drivers sign up, they must complete account onboarding through a link retrieved using the 'retrieveStripeOnboardingLink' function. To check if the user has completed an onboarding process, the 'handleStripeConnectWebhooks' function listens to the 'account.updated' webhook events.

More examples integrating Firebase Cloud Functions and Stripe can be found here.

Sign-in Persistence

Both the customer and business app are able to persist the sign-in state when users exit the app. This is done by Firebase Authentication. Each time a user opens the app, the app will check if there is a current user. If there is a current user, the user will be redirected to the home page. Otherwise, the sign-in/sign-up pages are shown.

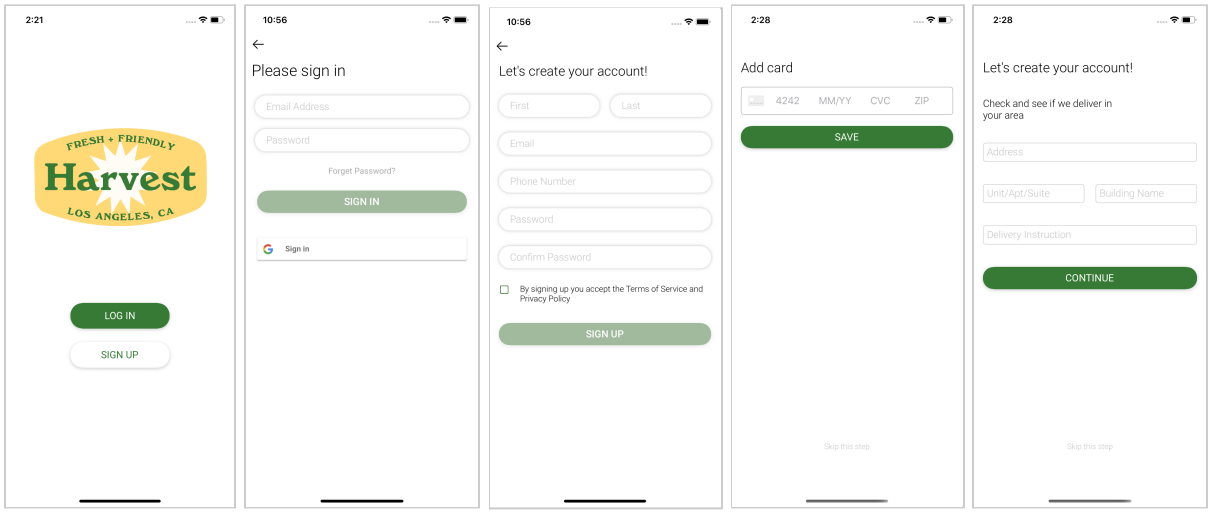
Chat

Chat UI between a customer and a driver is achieved through MessageKit. The logic behind this is to generate a chat session in the database for the customer and the driver when the order is accepted. Then, insert the message to the chat session whenever a message is sent by either the customer or the driver. In order to have real-time updated chat functionality, a listenTo firebase method is utilized to detect instant change within that chat session in the database so that the customer could retrieve the message and have a real-time chat.

Application Page Flow

Customer Version

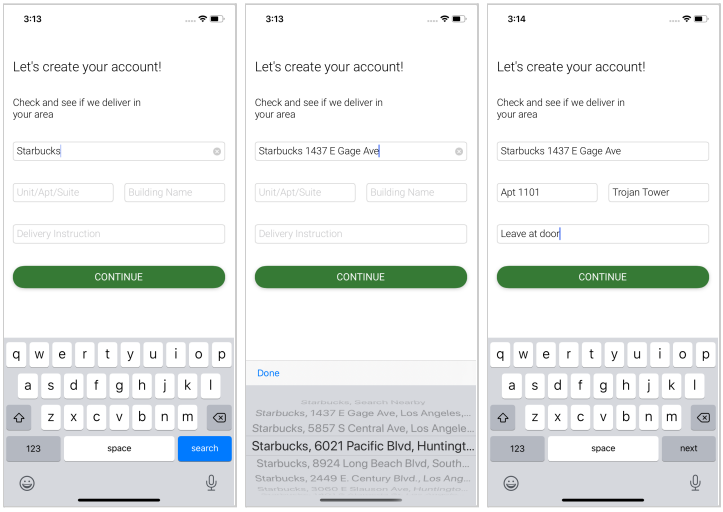
Sign-Up/Sign-In



Upon entering both the customer version of the application, the user is greeted with a simple sign-up/sign-in page, which will provide the options to sign in, sign up, reset the password, or log in with Google. The app would detect an invalid form of email and password, but the account username and password are validated with the Firebase Authentication.

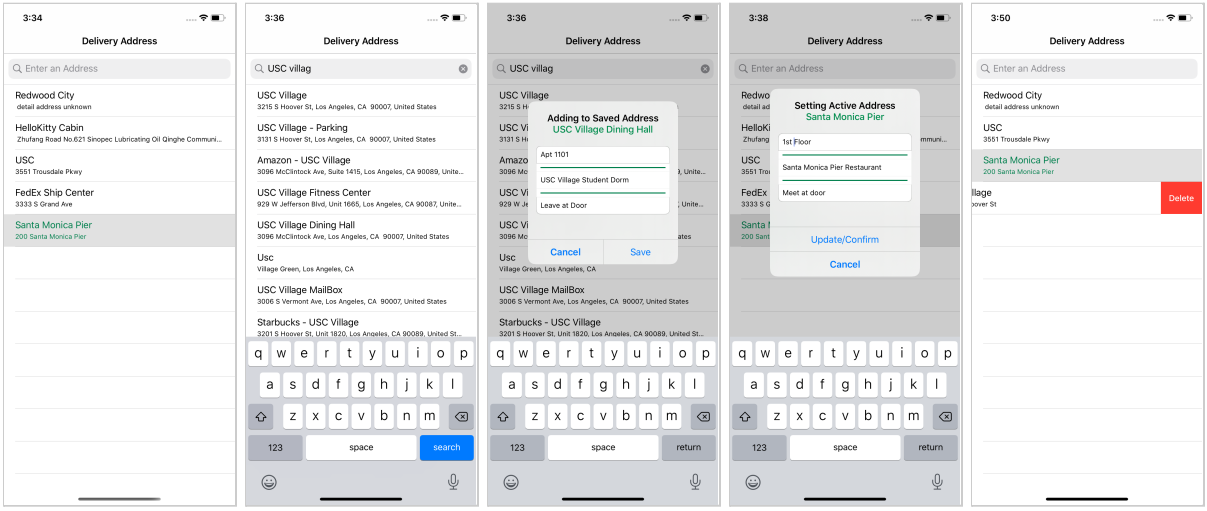
To sign up as a customer, the user is required to provide his/her first and last name, email address, phone number, and password. By checking the agreement to the terms and policies, the customer account will be successfully created in the database, and further, the user could optionally update his/her address information, as well as the card information. The card will be validated with Stripe.

Create Delivery Address in Sign Up



If the customer would like to update his/her address information when creating a new account, type in the address search bar and hit search, and the keyboard would then be hidden and switch to a picker menu with all searched addresses populated. Hit “done” to either start a new search or continue to fill out the apartment/suite/unit, building name, and delivery instructions to complete the address information.

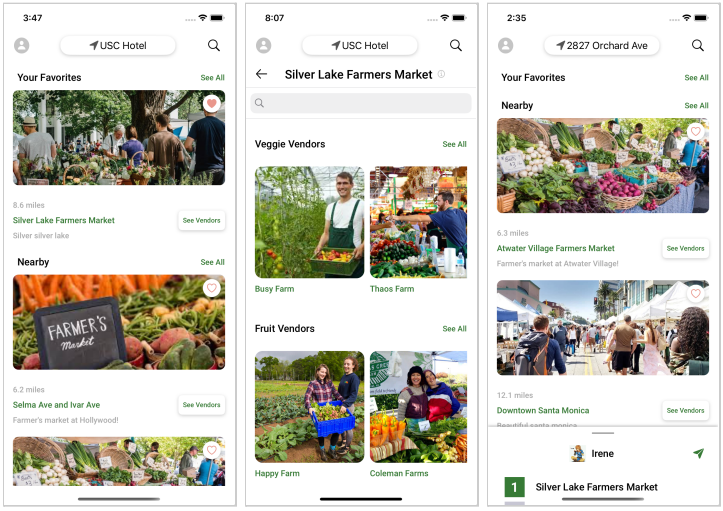
Delivery Address Page



The delivery address page could be accessed from Home Market Page, Farm Page, and Shopping Cart Page by clicking on the address button on the top. The page will list the saved address of the customer, and the current active address will be highlighted as green with grey shade background. The customer can type in the address search bar to search for an address, the tableview will be populated simultaneously as the customer types. By tapping on the

desired address from the searched results, a popup will ask the customer for his/her address details which are optional, and then hit “Save” and add this address to the saved address list. Update of saved address could be done in a similar way by tapping on a saved address that the customer wants to modify. Deleting button is used to delete a saved address and appears when swiping to the left on the target row of the address.

Home Market Page/Farm Page

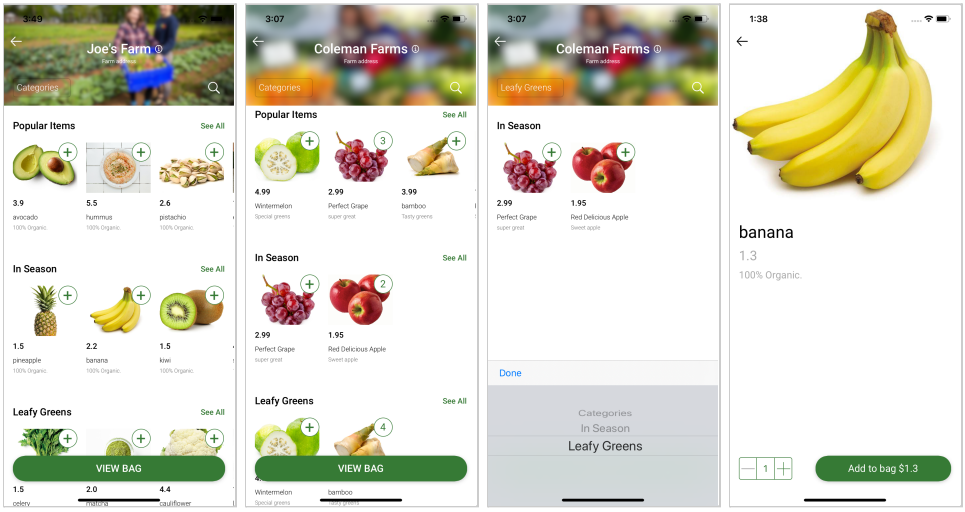


Upon successfully signing in with a valid email and password, the customer will be directed to the home page where displays favored markets and nearby markets. The saved markets can be found under the “Your Favorite” label indicated by the solid heart on the top right corner of the market’s image. The nearby markets can be found under the “Nearby” label with their respective distance displayed.

By clicking on the market, the customer will be directed to the farms page where all the different farms located at that specific market are displayed. The farms are currently categorized into “Veggie Vendors” and “Fruit Vendors”.

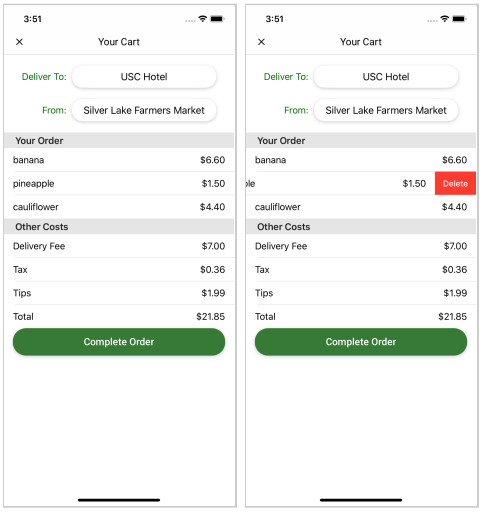
If an order that was placed is now accepted by a driver, it will appear at the bottom of the screen at the Market Home Page as a slide-up view (shown in the third screenshot). Customers could slide up to view the order detail, and contact the driver through chat.

Farm Produce Page/Produce Detail Page



Clicking on either the image or the name of the farm will direct the customer to the produce page of that farm. Further clicking on the image of the produce will direct the customer to the detailed information of that produce where the vendors could add their descriptions. The farm produce will be listed in categories. The “Category” button below the back button allows the customer to filter through different types of produce, and the plus sign button would allow the user to add the corresponding product to the shopping cart(bag). All the images of the produce are stored in Storage on Firebase.

Shopping Cart Page/Future Delivery Popup

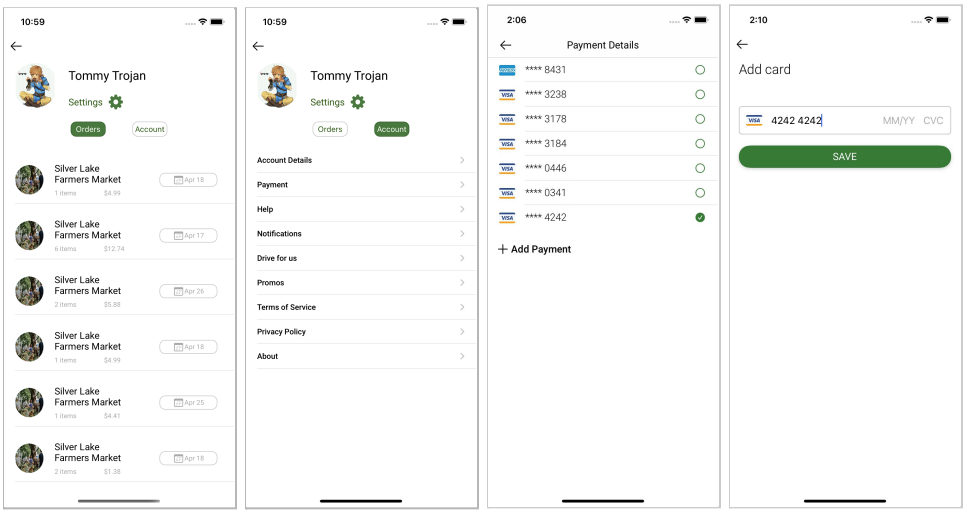


Clicking on “View Bag” from Farm Produce Page will direct the customer to the shopping cart. The added produce will be listed here with a total price. The estimated delivery fee and tax will also be calculated toward the total cost. Hitting the complete order will place the order,

push the order to the database, and redirect the customer to the Home Market Page. Deleting button is used to delete a row of produce and appears when swiping to the left on the target row of address.

Markets open in different time periods, and if this market where the customer is purchasing is currently not available, a popup for the schedule of future delivery would appear so that the order could be delivered at another time when that market is open.

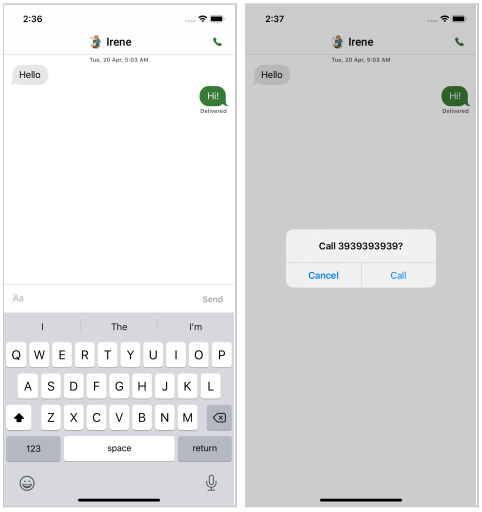
Profile Page/Payment Detail Page



The customer's profile is accessed by tapping on the user icon from Home Market Page and Farm Page. As default, the "Order" history button is turned on, and the table below listed the latest orders. To switch to the account information, hit the "Account" button, and a list of entries will be displayed. As only the core functionality is prioritized, only "Payment" on this list is implemented. Clicking on the "payment" will direct the customer to the Payment Detail Page, and the saved card information will be listed with the last 4 digits for privacy protection.

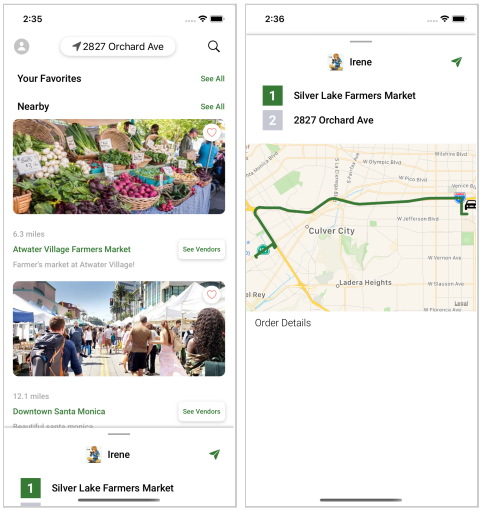
The customer can also add a new card on this page by clicking on "Add Payment", and then be directed to an add payment page which is similar to the optional one in the process of sign up, where the customer will be asked to complete card information, and be validated by Stripe. On completion of placing the order, the customer will be redirected to the Market Home Page, and the slide-up view will be displayed once a driver accepts the order.

Chat Page



A live chat section at the customer side is only available after a driver accepts an order that was successfully placed by the customer. The slide-up view indicating the active delivering status of the order will have a paper airplane button on the top right corner which will lead the customer to the chat page with the driver. The paper airplane button will only direct the customer to one driver at a time with the first driver who came and accepted an order (a queue structure). The phone symbol on the navigation bar will allow the customer to call the associated driver.

Slide-up Page

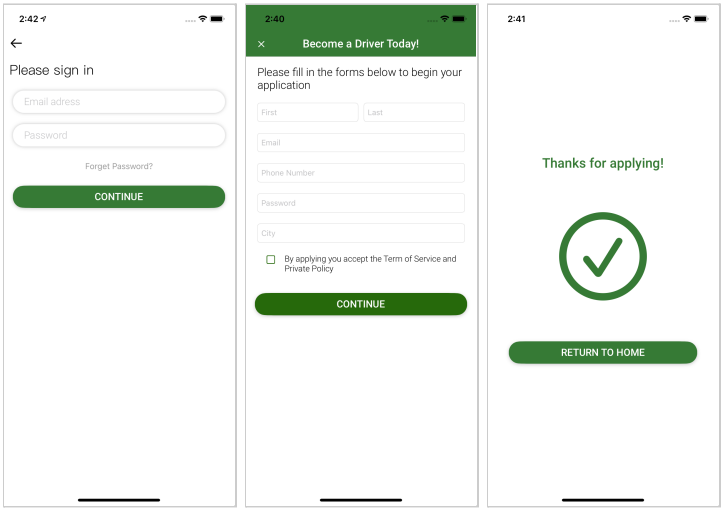


The slide-up sheet will appear after a driver accepts an order that was successfully placed by the customer. The sheet only shows the current direction of the driver when in a hidden position (only showing the top portion with the driver's name and a button to chat). When fully

swiped up the top, it will display the updated real-time location of the driver on a map generated with MapKit, as well as a list of items in that order. After the order was successfully delivered, the slide-up viewer will no longer exist.

Business Version

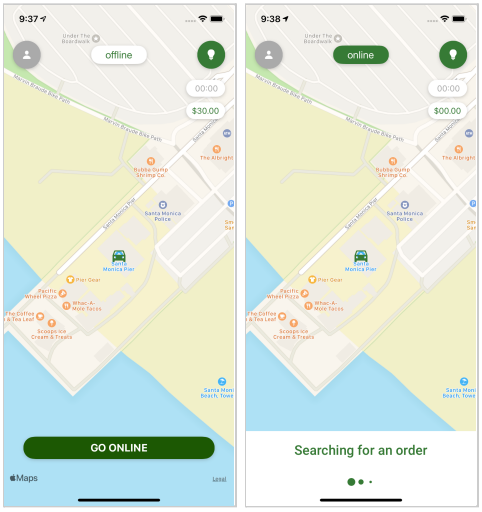
Sign Up/Sign In



The first screen of the business version of the app will ask the user to sign up/sign in as a vendor/driver. Both sign-up/sign-in pages are of little difference. Similar to the pages in the customer version of the app, the page will provide the options to sign in, sign up, reset the password, or log in with Google. The app would detect an invalid form of email and password, but the account username and password are validated with the Firebase Authentication.

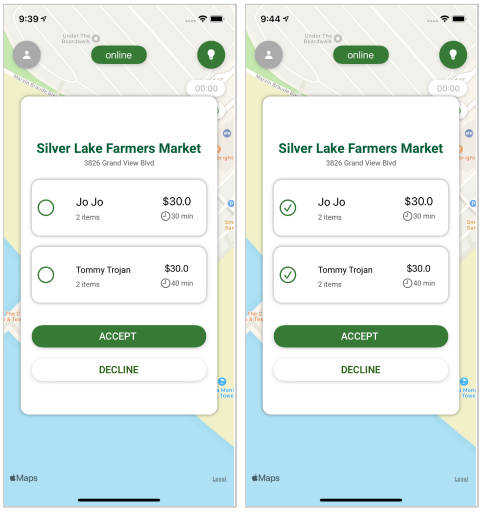
To sign up as a driver/vendor, the user is required to provide his/her first and last name, email address, phone number, and password. By checking the agreement to the terms and policies, the customer account will be successfully created in the database. Adding a card is mandatory in this case.

Home Page



The driver will be directed to the home map after successfully signed in. On the top right corner, there are two indicators, the one above indicates the amount of time that the driver has spent on the current order (which will become live when an order is accepted), and the one below indicates the amount of money that the driver has earned today. The button on the top left corner leads to the profile page of the driver. By tapping on the “GO ONLINE” button at the bottom of the screen, the app will start searching for active orders and switch the label on the top from “offline” to “online”. The search will last for 30 seconds and automatically stops to roll back to offline if there is no active order available currently.

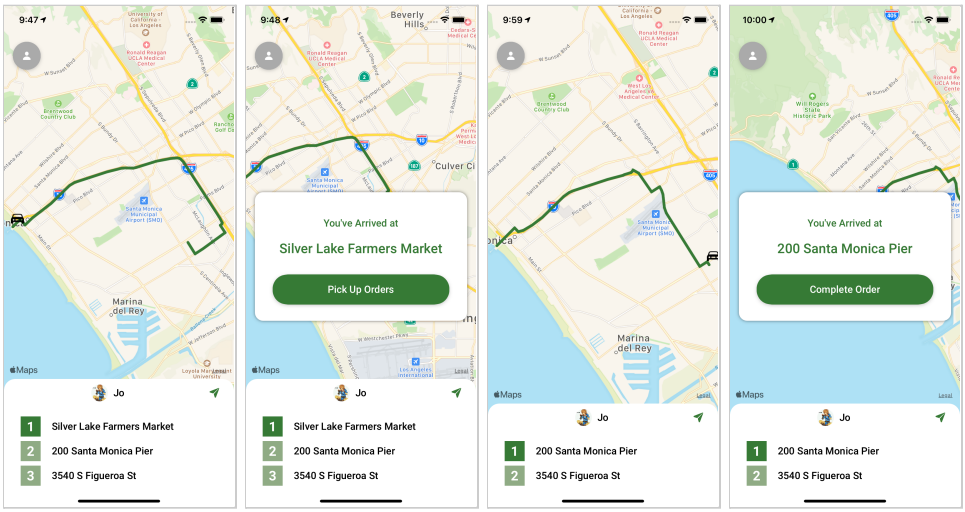
Accept Orders Page



If there are orders found during the searching process, a popup will appear with a list of available orders from a market. The driver can choose to accept multiple orders and start the

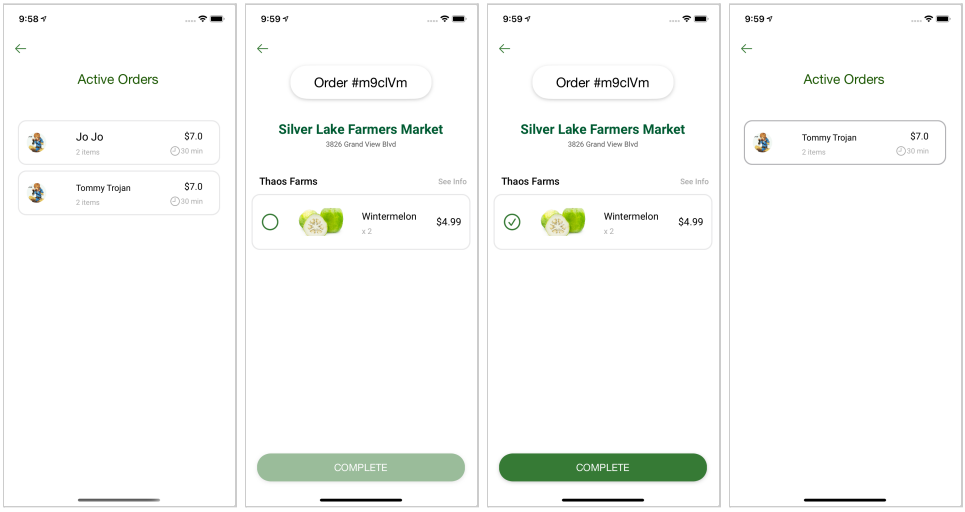
delivery session or to decline the orders found. If the driver declines, the popup will disappear, and the driver could start a new search for other available orders (queue structure for orders so that the driver will not keep retrieving the same orders that were declined). If the driver accepts, he/she will be directed to the Delivering Page to start the delivery session. The implementation of accepting orders goes through the entire order collection and sorts them in terms of distance and assigns the closest to the driver. (Currently, the maximum number of orders acceptable is 2.)

Delivering Page



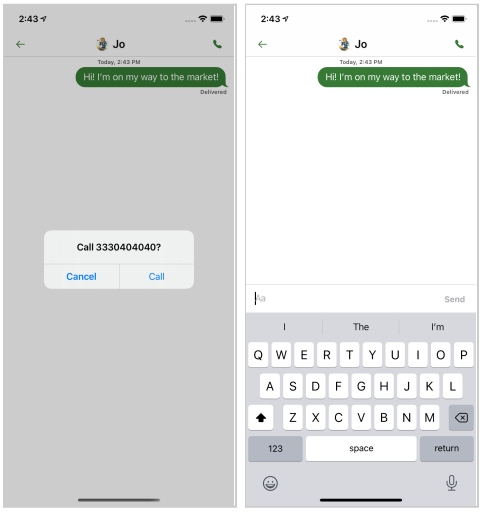
On the success of accepting orders, the driver will see a suggested route to the destination market drawn on the map. There will be a list at the bottom of the screen displaying the sequence of destinations, with the first destination being the market, and the second (potentially more if the driver accepted multiple orders) being the customer's delivery address. After the driver arrived at the market (this is automatically detected by calculating the distance from the real-time location of the driver and the market), and purchased all the listed items in the shopping list, he/she will be redirected to this page to continue the delivery to the next destination to a customer. The delivery session will be completed when the driver reaches the last customer's delivery address, which will also be automatically detected. The driver will be automatically redirected to the home page after delivery completion.

Active Orders Page/Shopping List Page



When the driver arrives at the market, the active order page will automatically appear, presenting the orders that the driver accepted with the customer's name, number of items, purchasing cost, and estimated time. By tapping on one of the orders, the shopping list corresponding to that order will be shown with an order number, market information, and the farm name for the driver to locate the farm for purchasing. The driver could tap on the row of the item to mark the item as purchased. The button at the bottom of the screen will send the driver back to the Active Orders Page if there is still an unpurchased order and will redirect the driver to the Delivering Page if all the orders are checked.

Chat Page



The chat page for the driver is implemented exactly the same as the one in the customer version. A live chat section at the driver side is only available after the driver accepts an order

retrieved. The half-screen viewer will have a paper airplane button on the top right corner which will lead the driver to the chat page with the customer. The driver will only be able to chat with one customer at a time, with the first customer listed on the active order list. The phone symbol on the navigation bar will allow the customer to call the associated customer.

Expected/Possible Future Implementation

Due to the time limitation, only the core functionalities of the customer side and the driver side were implemented fully. Below will go through some expected and possible future implementations and improvements.

Customer Side

The customer side has most of the core functionalities. One thing to be implemented is the **future delivery option** as the customer might want to order in advance. This feature would prompt the customer to enter a valid delivery time slot in the future so that this placed order only becomes active during that selected time slot. Note that every market has its own open time and close time, meaning that the customer could only choose a time slot in between. Some improvements would be more user friendly UI, more functionalities in the profile page, displaying order details in the slide-up view, displaying the number of the item in the shopping cart, and being able to place multiple orders.

Driver Side

The driver side has most of the core functionalities. Another core functionality should be implemented for the driver is the payment. The driver should **receive payment** after an order is delivered with the amount listed on the order structure, namely “driver_earned”. The payment distribution details are listed in Harvest Economics (page 9). Some improvements would be more user friendly UI, more functionalities in the profile page, showing estimated time (in the retrieved orders list) and distance for the entire trip, and better algorithm for retrieving current available orders.

Vendor Side

The vendor side was only implemented with an incomplete sign in/sign up features. Similar to the driver side, the user would sign up with the business version as a vendor. The expected outcome of a successful sign up would require the vendor to **pick a market**(s) as their selling location during the sign up process, and create a farm in the Firestore correspondingly. The home page should be a complete list of the produce that is currently in stock. The core feature of the vendor side is listing produce and getting paid when an order is finished.

Listing produce will be relatively straightforward with the given data structure and reference on Firestore. A new view controller should be added for the vendor to add the item with all

properties required. The vendor should also be able to modify/restock the number of an item in the inventory.

Receive payment will be accomplished with a stripe, and this will be implemented in the same way as the driver.

GOOD LUCK!

Challenges

General Challenge

The most outstanding challenge during the development was the process of connecting the front-end to back-end. The front-end logic was not designed as thoroughly and thoughtfully enough that we had to spend more time on re-design the structure and UI of the front-end to cooperate with a possible back-end solution. A good way to avoid this is to actively observe the provided design from the stakeholders and try to think ahead for the back-end implementation so that potential issues could be prevented effectively before committing to a completed front-end.

Design Challenge

The main challenges come with the design of the order query for drivers. The current implementation goes through the entire order collection and sorts them in terms of distance to the driver. However, that means to query all the active orders every time the driver starts a new search, which could take some time if there was a large number of orders. Some idea would be to divide regions into different sub-collection under order collection with location tag (lat, lon) so that the driver would only search in a region that is the closest.

Deployment Document

Same for both versions

1. Open Xcode
2. Git clone the project and open the project workspace in Xcode
3. Open the command line and go to the project directory
4. Pod install (You need to have CocoaPods first)
5. Build and Run the project on Xcode
6. The simulator should pop up, open the Harvest app
7. Have fun!