GETit

CS 307

Team 19

Design Document

Team Members:

Aman Wali

Kartik Mittal

Sehajbir Randhawa

Shiv Paul

INDEX

Purpose:

A lot of students have either procrastinated or been lazy on multiple occasions. Sometimes, they can even be too busy to complete a task. There exist apps to fight some of these issues, such as GrubHub food delivery but sometimes, for example, students want to buy a book from the University Bookstore, or need a Starbucks drink delivered to them. In situations like these, where we are alleviating more than just hunger, we need a simple way to complete a variety of such tasks that no existing app offers.

Our project, GETit, offers a solution to all these problems. It is an app for both Android and iOS which allows users to post requests and allows other users to accept them and complete the requested task. Through our app, one can complete all simple errands by offering whatever small amount they choose to pay and other users can earn a few extra bucks without going out of their way. For example, if a user was walking back from classes to Earhart Hall and was about to stop at Starbucks for a coffee sees a new request for a Starbucks to be delivered to Shreve can simply accept it and deliver it to the other user on their way back.

Functional requirements:

- **User account**
  - I want to be able to create a new account.
  - I want to be able to sign in to my account.
  - I want to be able to sign in with Google.
  - I want to be able to set my mobile number.
  - I want to be able to set my address.
  - I want to be able to set a profile picture
  - I want to be able to reset my password.

- **Users can post requests**
  - I want to be able to generate a request for a task.
  - I want to be able to reorder a previous order of mine.
  - I want to be able to specify comments about the request.
  - I want to be able to add a link to a product to be more specific.
  - I want to be able to set a price to pay the user that accepts the request.
  - I want to be able to cancel my request before it is accepted.
  - I want to be able to edit my request before it is accepted.
  - I want to be able to place multiple requests at the same time.
  - I want to be able to select a saved address or add a new one.

- **Users can accept requests**
    - I want to be able to select preferred categories of requests to see.
    - I want to be able to see the amount I will be paid for a request before accepting it.
    - I want to be able to see the location of a delivery before I choose to accept it.
    - I want to be able to select an option to text the user if I have any questions about the order.
    - I want to be able to select an option to call the user if I have any questions about the order.

- **Notifications:**
    - I want to receive a notification when a user accepts my request.
    - I want to receive a notification when the user arrives at the restaurant or store.
    - I want to receive a notification when the user picks up my order.
    - I want to be able to turn off notifications about my order.
    - I want to receive a notification when a request in my preferred category is posted.
    - I want to be able to turn off notifications about requests in my preferred categories.

Non-Functional requirements:

- **Architecture:** As a developer,
    - I want to use firebase database to be able to store all the user account information.
    - I want to use the database to store all the details of the requests currently present as well as history of all requests.
    - I want to use React Native to make the application so I can simultaneously develop for both iOS and Android to optimize the coding process.

- **Performance:** As a developer,
    - I want the application to work smoothly without crashing.
    - I want to use a reliable backend platform like Firebase which offers an average read time of 30 ms for my read/write calls to the database.
    - I want to be able to serve over 100,000 users so that all of them can use the application simultaneously.

- **Security:** As a developer,
    - I will use Firebase as it is built with strong user-based security to keep the user account data safe.
    - I want to use a QR code verification on completion of requests to ensure security when making payments.

- **Deployment:** As a developer, I want to deploy the app on both iOS and Android app stores.

Design Outline:

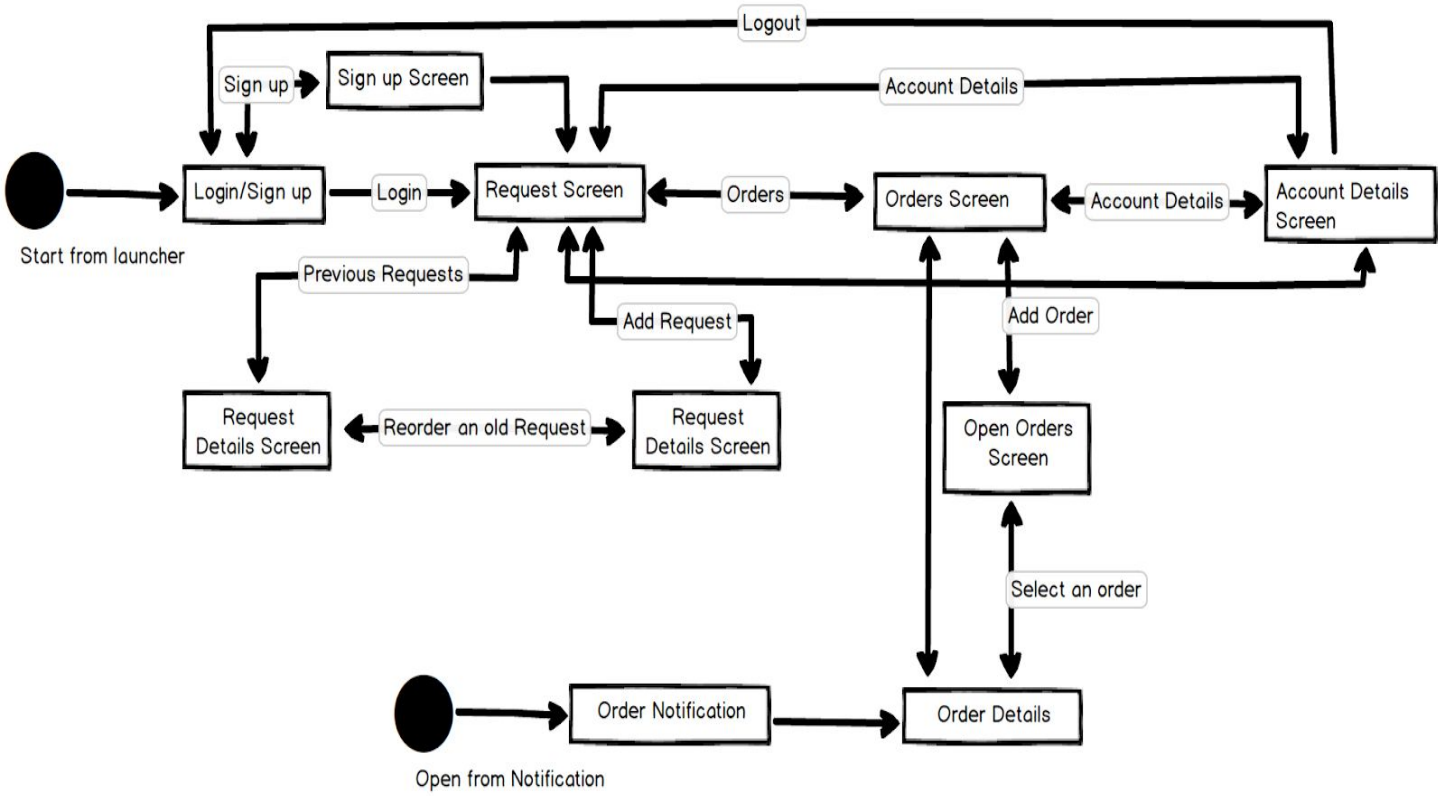
The project is an application for both iOS and Android that allows users to post requests for tasks they want completed which can range from getting a Starbucks drink delivered to getting a book from Follets.
API calls will be made for the Firebase database as well as a payment system to complete transactions.

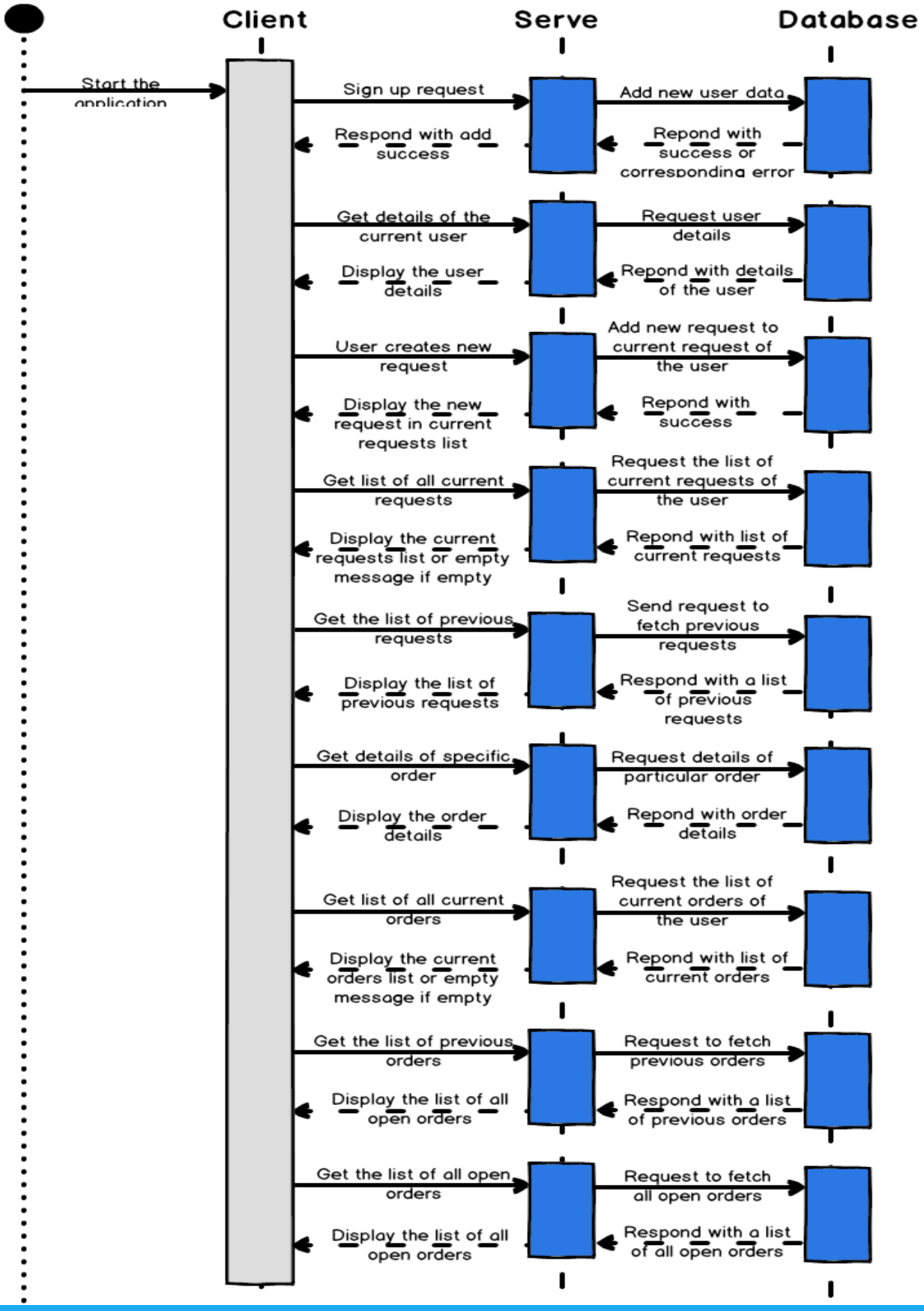
- **Client:**
  - The client will be the Android or iOS app running on the user's phone and will be the interface for our app.
  - The client offers the user options to generate requests and accept orders as well from the same account.
  - The client keeps updating data from the database and provides up to date information on requests.
- **Server:**
  - API requests are made to the Firebase database to get data about the user to log in to their account.
  - API requests are made to the Firebase database to get data about the requests and orders.
  - API requests are made for payments to be completed.

- **Database:**
    - The database stores information about the user's account including their profile picture, phone number, address, previous orders.
    - The database also holds all current orders and sends this information to the client when an API request is made.

Sequence of events:

The diagrams below shows the entire sequence of events of our application. This sequence begins when the user opens up the application and chooses whether to log in or sign up. It shows how all requests are made by the client and how information is finally received from the server and delivered back to the client.




Client
Serve
Database
Start the application
Sign up request
Respond with add success
Add new user data
Repond with success or correspondina error
Get details of the current user
Display the user details
Request user details
Repond with details of the user
User creates new request
Display the new request in current requests list
Add new request to current request of the user
Repond with success
Get list of all current requests
Display the current requests list or empty message if empty
Request the list of current requests of the user
Repond with list of current requests
Get the list of previous requests
Display the list of previous requests
Send request to fetch previous requests
Respond with a list of previous requests
Get details of specific order
Display the order details
Request details of particular order
Repond with order details
Get list of all current orders
Display the current orders list or empty message if empty
Request the list of current orders of the user
Repond with list of current orders
Get the list of previous orders
Display the list of all open orders
Request to fetch previous orders
Respond with a list of previous orders
Get the list of all open orders
Display the list of all open orders
Request to fetch all open orders
Respond with a list of all open orders

Design Issues:

Functional Issues:

1. Does the user need to sign in to the app to use it?
    a. No need to sign it
    b. Use Google/Facebook to sign in
    c. Enter username and password to sign in

    As we want the app to be used by credible users, we need users to sign in to the app to use it. For users convenience, we will allow the user to sign in using both Google and personal details with username and password.

2. Can a user make multiple requests and accept multiple orders at a time?
    a. User can make multiple requests and accept multiple orders
    b. User can make only one request but accept multiple orders
    c. User can make multiple request but accept only one order
    d. User can only make one request and accept one order

    We chose option A to give the user the ability to make multiple request so that they can get many things done at a time and not wait for one request to be completed to make a new one. Also, a user can complete as many order as possible at a time which will result in quicker completion of requests.

3. Should previous orders be accessible with the current ones?
    a. Previous orders are not shown
    b. Previous orders are shown in the settings
    c. Previous orders are shown with the current orders

    We chose to show the previous orders with the current ones to allow the users to view and issue a quick reorder of something that they order often. We believe this would make using the app much more faster due to the increased efficiency.

4. How should we verify that the person completing the task has met and completed the request of the person in question?
    a. Let them handle it themselves out of the app
    b. Add a button that says "Task Done"
    c. Show a QR Code in the person's app that the requester can scan to verify completion

    We chose to use the QR code option as that will verify that the 2 people in question have actually met and that the task is completed. This will also guarantee secure verification as the person doing the task will not be able to fake a completion in order to get paid.

Non-Functional Issues:

1. What platform do we make the app for?
    a. Android
    b. iOS
    c. React Native (Android and iOS)

    We chose React Native as a **majority of its APIs are cross platform, which means that your one component will work on both iOS and Android. We can develop complete, full blown applications that look, run and feel native – without writing a single line of platform specific code.**

2. Should we store all the data locally or everything in the database?
    a. Store locally using an SQL database/etc
    b. Store all data online

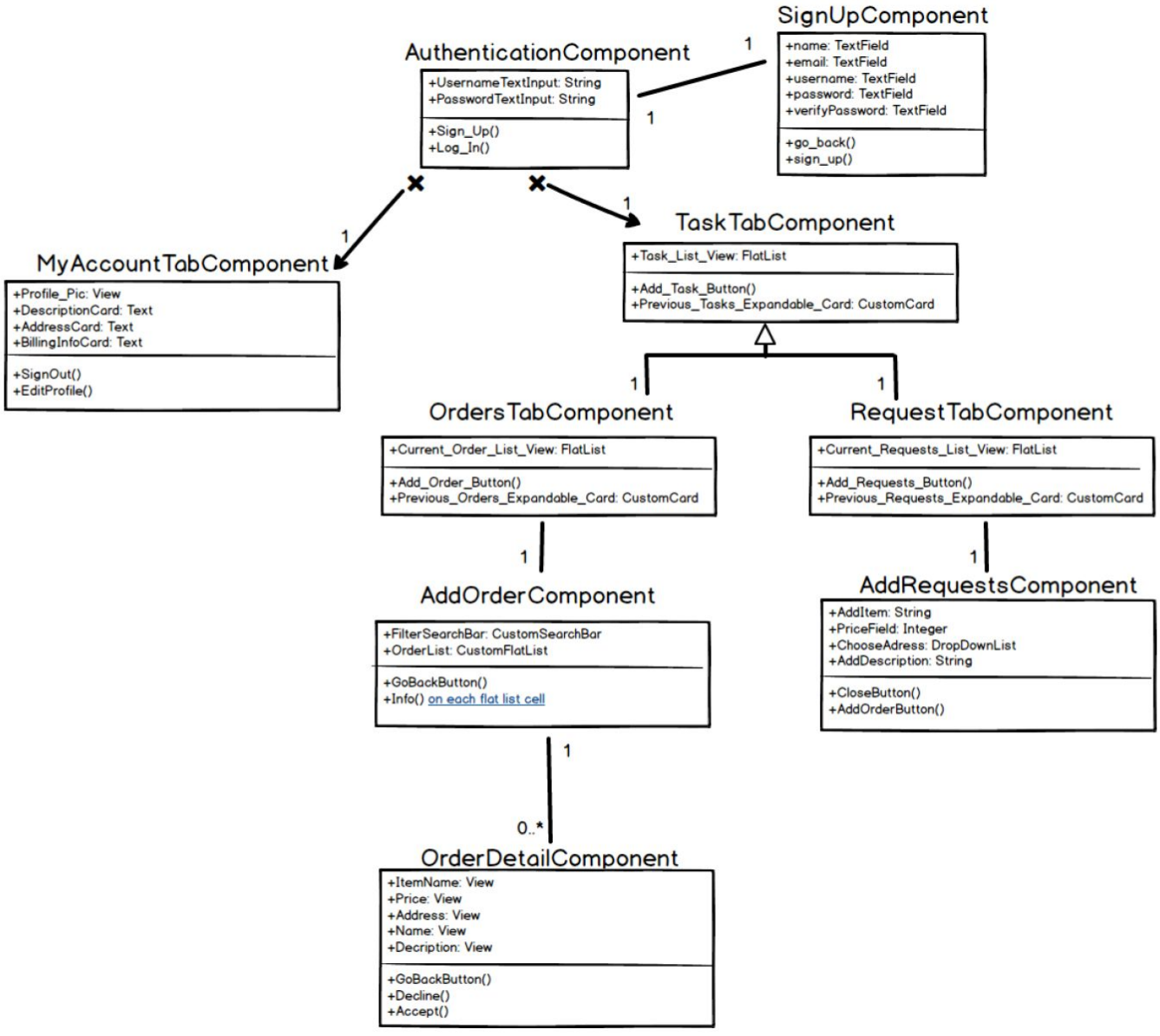
    We chose to go with storing everything online since that will give the users the ability to login into multiple devices without having to worry about shifting their data. Also, this method is convenient for development as well since the real time databases are going to be used anyway for handling the requests.

3. What do we use to store all the data of the users and their requests?
    a. A custom server hosting all the data
    b. Amazon Web Services
    c. MongoDB
    d. Google's Firebase

We chose to use Google's Firebase as its main benefit shines in its realtime database, where all the data is synchronized across the clients. Apart from this, everything from databases, analytics to crashing reports are also included in Firebase so we, as the development team, can stay focused on improving the user experience.

Design Details

Class design:



Description of Classes and their Interactions:

Our Class design is based on the different screens in our app and the routes between each of them.

AuthenticationComponent:

- This is the introductory screen to the app and is responsible for the authentication of the user.
- This component contains a text field for both username and password.
- Once the login button is pressed, the data from the text fields is used for authentication purposes through firebase and the control is passed to the OrdersTabComponent
- If the Sign Up button is pressed, the SignUpComponent is opened.

SignUpComponent:

- This component provides all information for sign up.
- This component contains text fields for name, email, username and password.
- Once the Sign-up button is pressed, the user is signed up and the control is taken to AuthenticationComponent for logging in.
- Once the back button is pressed, the control is taken to AuthenticationComponent for logging in.

TaskTabComponent:

- This component is a blueprint for the OrdersTabComponent and the RequestTabComponent
- It will contain the basic current task list, the expandable previous tasks list and the floating add task button which when pressed takes control to the next screen depending on its implementation

OrdersTabComponent:

- This screen is one of tabViews and its properties are inherited from the TaskTabComponent.
- It contains a scrollView which contains a currentOrders list card and previousOrders expandable list card as subviews.
- Once the floating add button is pressed, control goes to the AddOrderComponent.

AddOrderComponent:

- It is a component which contains a list of all the orders that are available to be taken.
- Each list cell contains an info button which when pressed takes the control to the OrderDetailComponent
- The screen also contains a back button which takes the control to the OrdersTabComponent.

OrderDetailComponent:

- This component contains the Name of request item, price, requesting user's name and address as views.
- The component also contains an Accept, Decline and back button.
- When the decline or back button is pressed, the control is shifted to the AddOrderComponent.
- When the Accept button is pressed, the order is added to the current orders and the control is shifted to the AddOrderComponent.

RequestTabComponent:

- This screen is one of tabViews and its properties are inherited from the TaskTabComponent.
- It contains a scrollView which contains a currentRequests list card and previousRequest expandable list card as subviews.
- Once the floating add button is pressed, control goes to the AddRequestComponent.

AddRequestComponent:

- This screen is used to create a new request.
- The component has text fields for the name of request item, price and the description of the request.
- This component also has a drop down list of addresses to choose from.
- The Add Request button creates the request and takes the control back to the RequestTabComponent

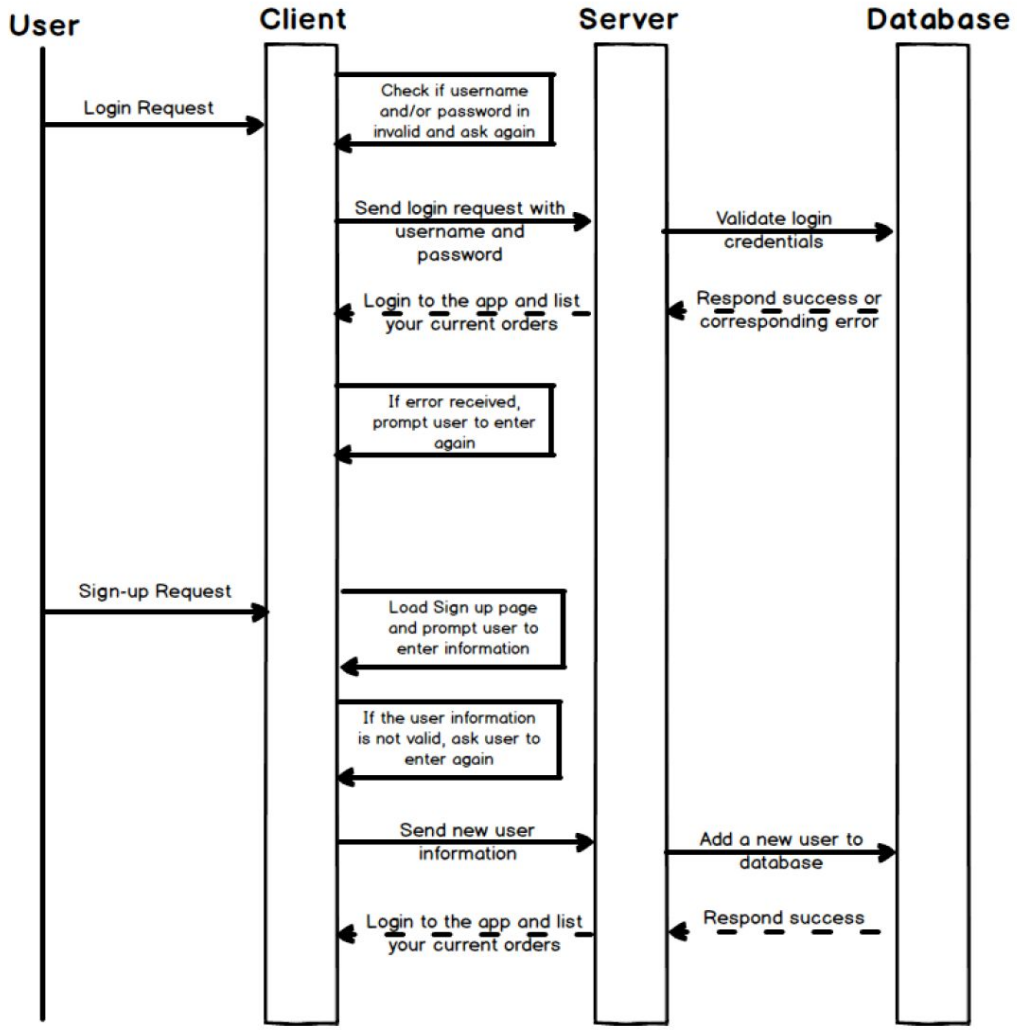
MyAccountsTabComponent:

- This component is used for showing the profile of the user.
- It consists of the user's profile picture, a card for personal details, a card for showing list of addresses and a card for billing info.
- There is also a sign out button which takes the user to the AuthenticationComponent and signs the user out in the the backend as well.

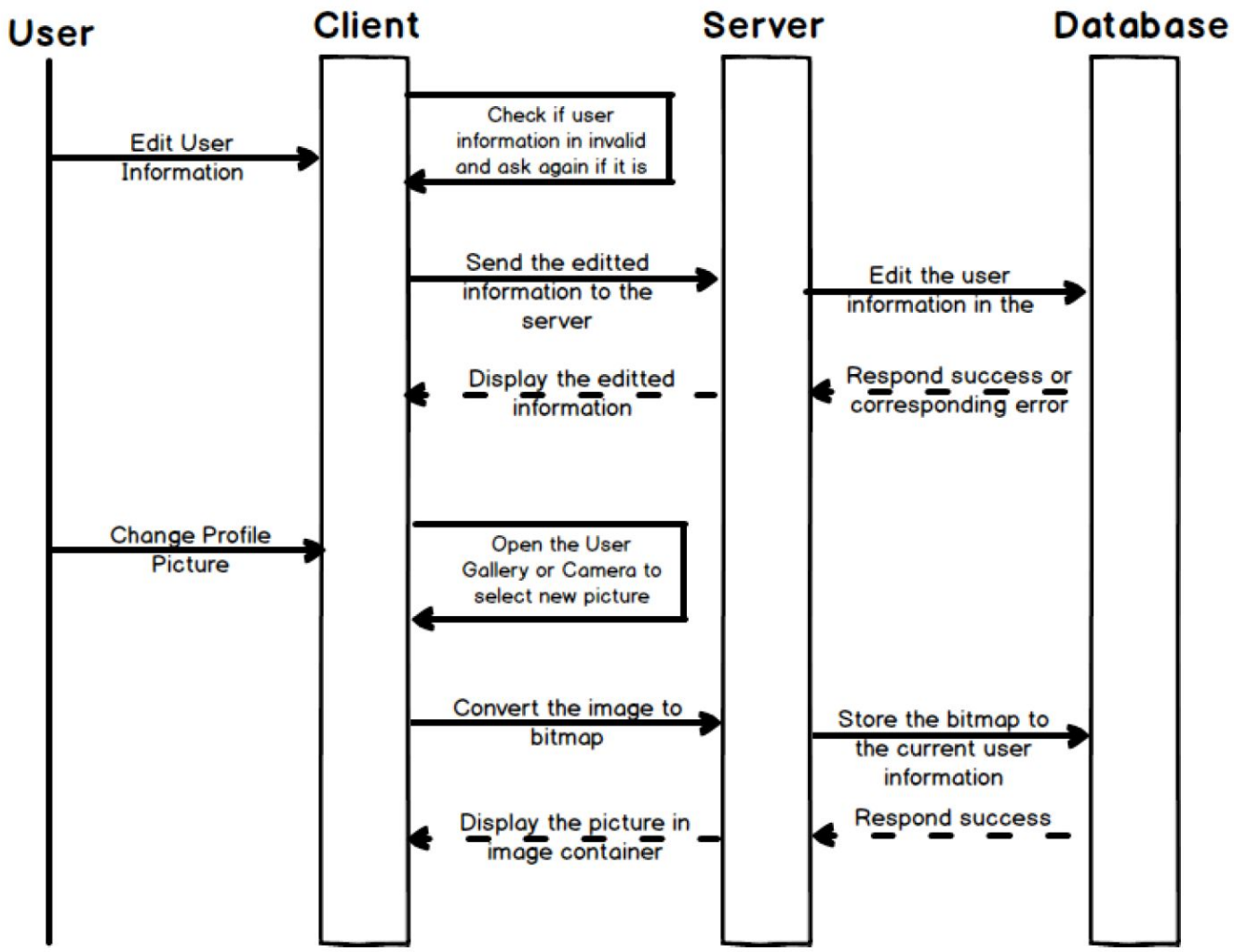
Sequence Diagrams

The following diagrams depict the sequence of events occuring in our application between the client, server and database.

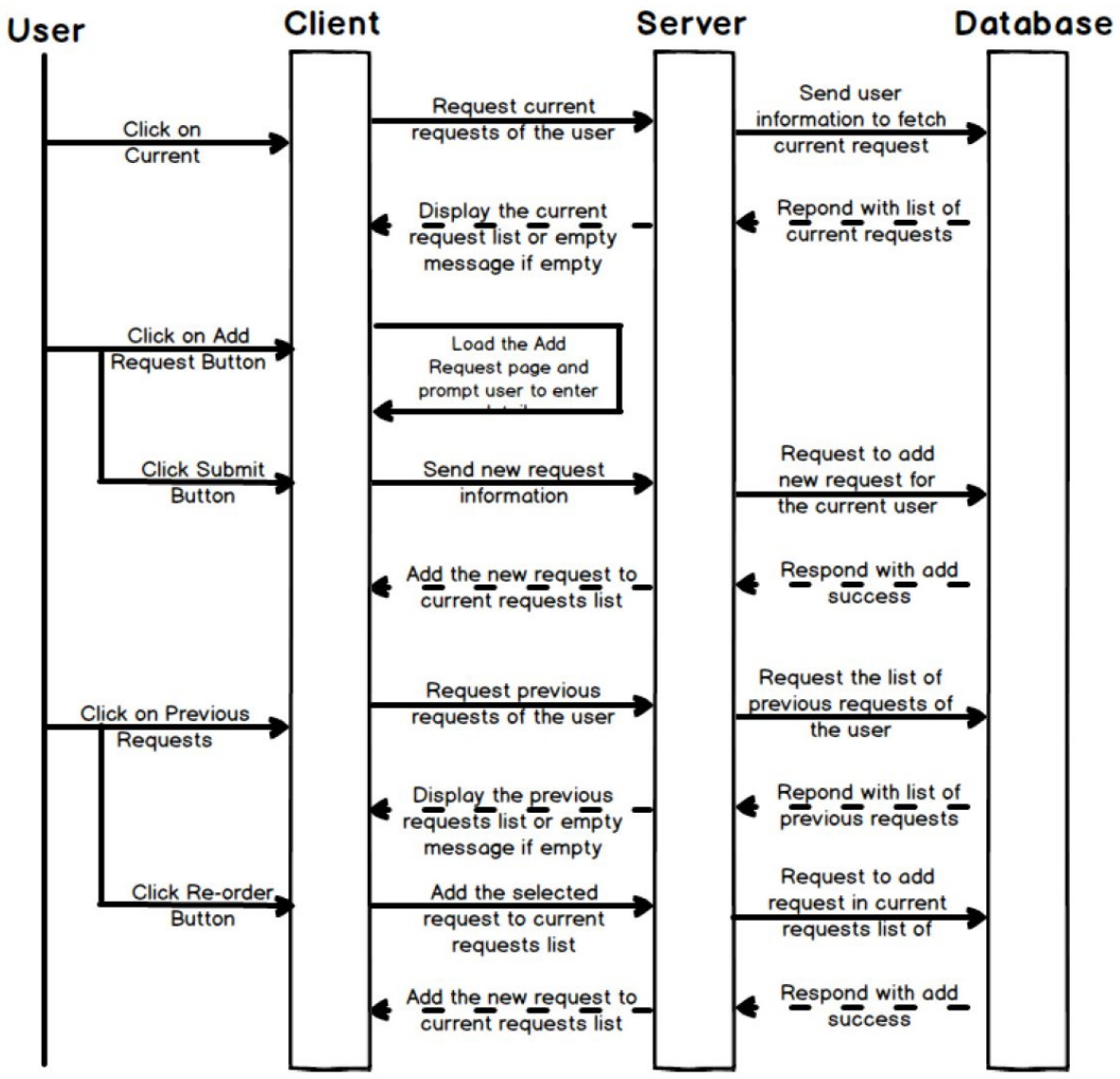
1) Sequence of events for when a user starts up the application and wants to log-in or sign-up:



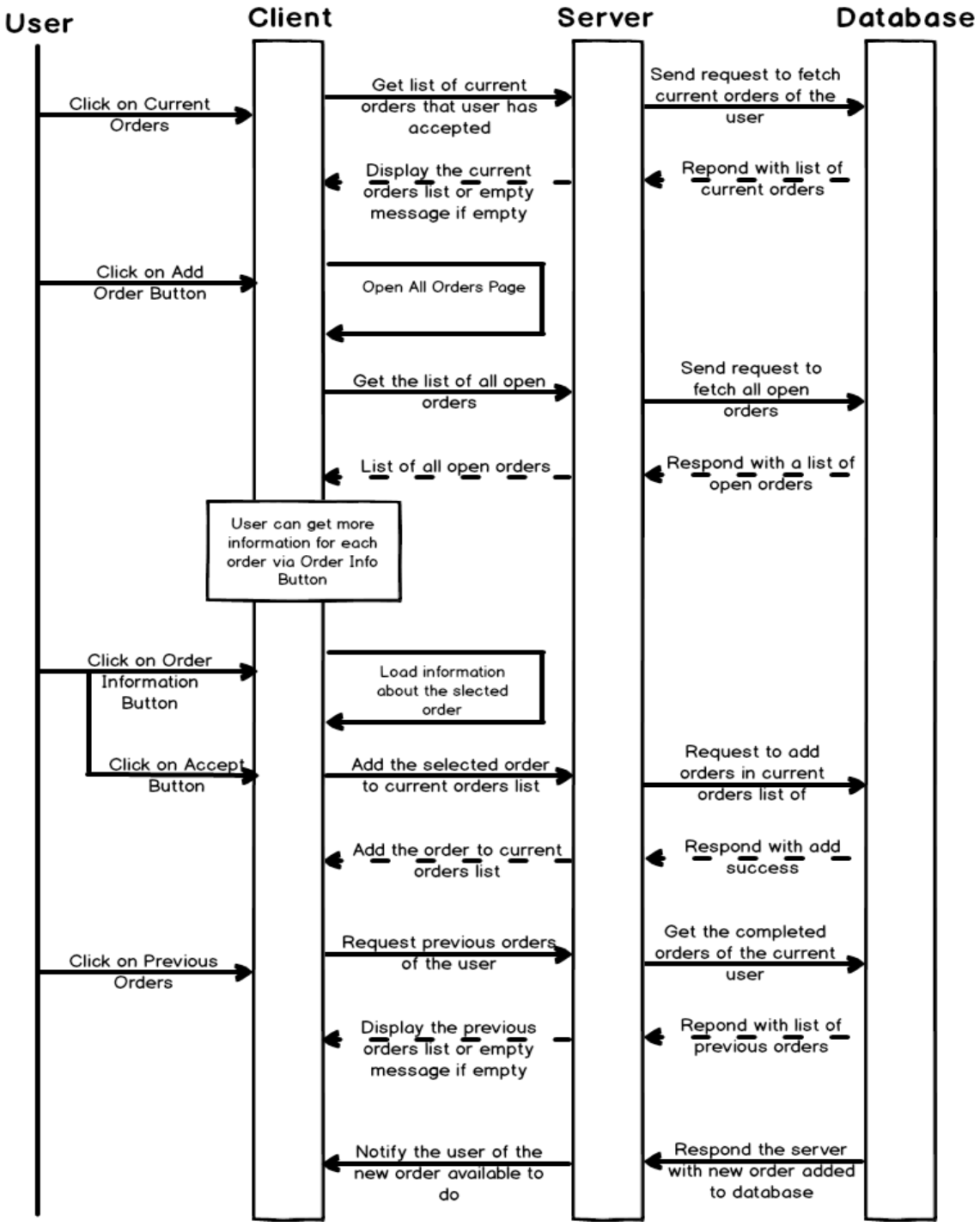
2) Sequence for when a user edits their information under the section of "My Account":



3) Sequence for when a user wants to generate a request for a task, view previous requests they have made or reorder a previous request in the "Requests" tab:



4) Sequence for when a user wants to look at orders to accept, view their information, or look at past orders they have accepted in the “Orders” tab:



UI MOCKUPS

Here we have made some mockups of what the different screens in our application will look like.

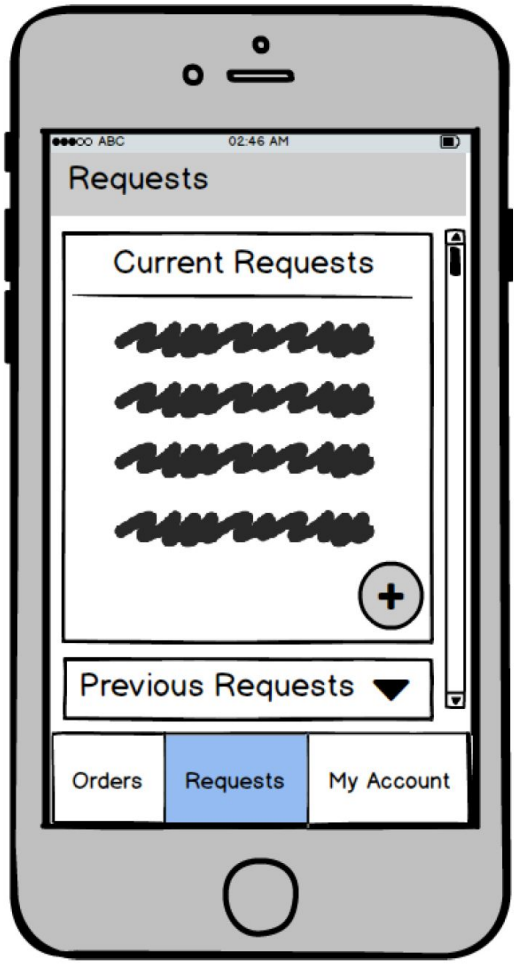
1) The first mockup we have will be the log-in/sign-up page which a user sees when they first open the app.

If a user already has an account with GETit, they will simply enter their username and password and click Log-in, otherwise they will click Sign-Up and create an account.

2) This is the page a user will see if they select Sign-Up on the previous page and want to create an account.

The user will enter the information shown on the screen and select "Sign-Up" to create an account.

3) This screen is what the user will see on logging-in. There are three tabs as seen at the bottom - Orders, Requests and My Account.

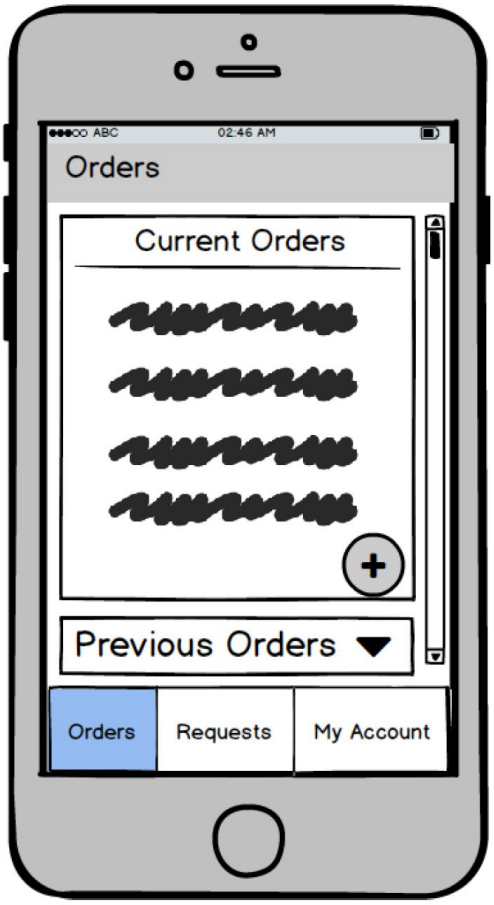


Current Requests shows any requests the user has currently put out for another user to accept. The user can click the "+" icon to go to a page to generate a new request. Previous Requests shows the user the previous requests they have made.

4) This is the page which is shown when a user clicks the button to generate a new request for a user to accept.

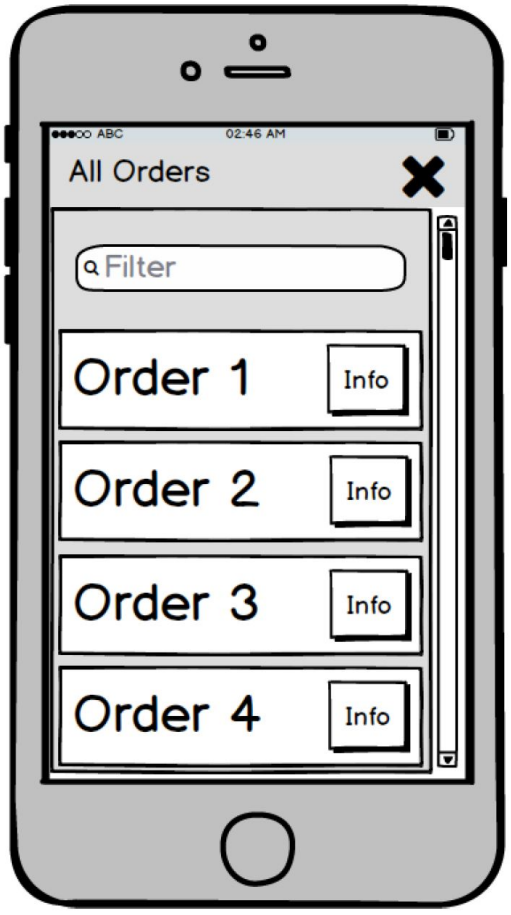
The user will enter the Item name, select a price(including delivery fee), select the delivery address and add a description for the item. If the user is then satisfied with the item, they can click Add Request to post the request or click the cancel button to not post it and go back to the previous page of their requests.

5) This is the page of Orders which a user will visit when they want to accept an order someone else has placed and complete it.



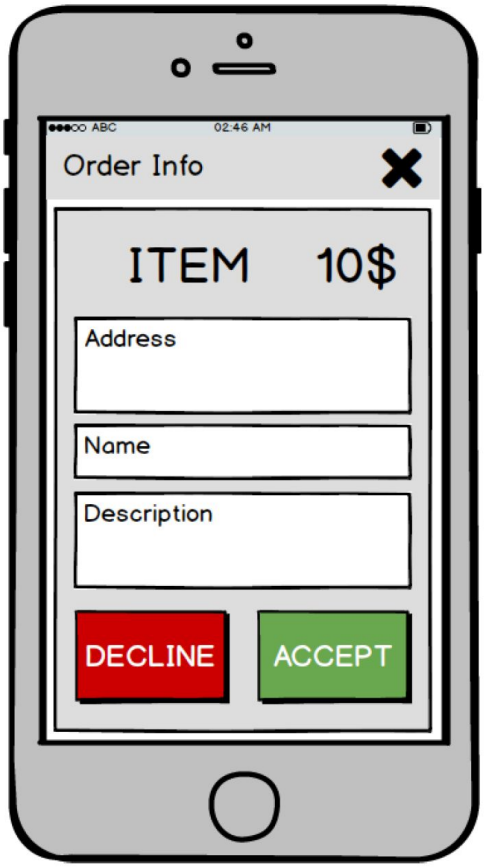
If the user is currently fulfilling someone's order, it will show up under "Current Orders". The user can view previous orders they have completed under "Previous Orders". If the user wants to complete someone else's order they can click on the "+" icon to go to a poge to do so.

6) If a user clicks the "+" on the previous page, they are taken to this page where they can view the active orders available to pick up.



If the user seems interested in any of these orders, they can click on the "Info" button to view more information regarding the item and maybe even accept it. If not, the user can click the cancel button at the top to return to the "Orders" tab.

7) On clicking the “info” button on an order, the user arrives at this page where they view all the information about the order including where it has to be delivered, how much they will be paid and a description.



If the user is satisfied with the order, they can accept it by clicking the “ACCEPT” button and it will be added to their “Current Orders” section. If the user is not satisfied with the order, they can click decline and go back to viewing other orders available.

8) This last mockup is of the "My Account" tab which the user can select to either sign-out or view their information stored in their account such as their name, profile picture, phone number or address.

If the user wants to sign-out, they can click on the "Sign-Out" button which will take them back to the first log-in page. The user can also edit their account information by clicking on the edit info icon at the top of the screen.