Merchant

Design Document

Team 4:
Domenic Conversa
Chirayu Garg
Drew Keirn
Aakarshit Pandey

Index

Purpose

As college students, we own a lot of temporary goods: textbooks, dorm supplies, furniture, and more. Once we are done using an item and no longer need it, we don't have space to store it and would rather have a way to sell it to someone else who needs it. In addition to these temporary possessions, we also have other general goods that we are interested in buying and selling, like sports tickets and clothing for example. While there already exist popular platforms for buying and selling goods online, such as Craigslist and Facebook Marketplace, these options raise a safety concern since they allow anyone, anywhere to buy and sell goods. To make the transaction of exchanging items more amicable and safe, we have designed an iOS app, called Merchant, which allows for a closed network of students from a particular university to buy and sell goods from each other in order to create a safe exchange for both parties.

Although similar, university focused platforms do exist for exchanging goods, such as Locolist or Verto, neither of them have gained much traction. This is probably because Locolist, a mobile app, appears to be outdated since it has not been updated for 2 years. Also, Verto is an online platform which does not provide the convenience of a mobile app. Thus, the market is still open for an app of this type to be developed.

The purpose of Merchant is to provide a safe, closed network for university students to be able to comfortably exchange goods on their campus. Aside from having the closed network of students as a key design element, Merchant also incorporates a map that allows students to agree on a meeting place to complete the transaction. This map provides suggested meeting locations based off of a university's "Emergency Telephone System" which includes blue light emergency call boxes across the campus. We believe that combining a trusted user base with safe meeting locations will lead to a desire to use our service over the other available options.

Functional Requirements:

1. **Users can setup and manage their account**

   As a user,

   a. I would like to be able to register for a Merchant account with my university email so that I have my user profile and other users are also students.
   b. I should be able to verify my user account with a link/OTP sent to my email.
   c. I would like to view and sign or unsign user agreement policies.
   d. I would like to be able to log in to my Merchant account by providing my university email and a password I set while registering.
   e. I would like to make my user profile in the app with a distinct username.
   f. I would like to be able to reset and/or recover my password.
   g. I would like to be able to update my account settings.
   h. I would like to add/update a profile picture for my account.
   i. I would like to be able to delete my account so I can no longer participate in the service if I don't want to.
   j. I would like to report other users, fake users or spam so that they are accountable.
   k. I would like to rate and review other users so that they are accountable.
   l. I would like to view my selling history so that I can keep a track of things that I sold in the past.
   m. If time allows, I would like to make a wish list of the items I want to buy.
   n. If time allows, I would like to get notifications about the items on my wish list.
   o. If time allows, I would like to follow other users.
   p. If time allows, I would like to get notifications if the user I follow post items.
   q. If time allows, I would like to be able to block other users.

2. **Users can buy goods**

   As a user,

   a. I would like to view items for sale so that I can find an item that I want to buy.
   b. I would like to be able to express an interest in buying items for sale.
   c. I would like to sort items for sale by price and type so that it is easier for me to search for the item I want to buy.
   d. I would like to be able to search for a specific seller or a specific product so that it is easier for me to look for specific items that I want to buy.
   e. I would like to be able to search for users so that I can interact with a user that I want to buy from.
   f. I would like to chat with a potential seller/ buyer without having to share personal information so that my privacy is secure.

g. I would like to have the option to share my location with the seller and track his/her/their location (if he/she/they permits) during the time of transaction so that meeting up is easy.
h. I would like to be able to choose a location for a transaction so that transaction can take place at a specific place.
i. I would like to view a map for safe locations for a transaction so that transaction happens at a safe place.
j. I would like to be able to delete a conversation with a buyer after completing a transaction so that I don't have an excessive amount of conversations.
k. If time allows, I would like to be able to bid for items available for bidding.

3. **Users can sell goods**

   As a user,

   a. I would like to post items for sale so that I can sell an item that I want to sell.
   b. I would like to upload pictures from my photos or take a picture using my phone camera so that I can showcase the product I want to sell.
   c. I would like to view items that I am selling currently
   d. I would like to be able to remove an item that I put for sale so that buyers no longer ask to purchase an item that I no longer want to sell.
   e. I would like to be able to update details about the product I am selling so that buyers have the most accurate information.
   f. If time allows, I would like to post items for bidding.

Non-Functional Requirements:

Architecture and Performance

As a Developer,

1. I would like to make an iOS app in swift.
2. I would like the backend to be in NodeJS.
3. I would like to have a different front end and back end so that it would allow us to expand the App to other platforms with ease.
4. I would like to be able to communicate and get data from the database in less than 1 second.
5. I would like to post data to the database in less than a second.

As a User,

1. I would like to be able to post an item in less than 1 minute.
2. I would like to enquire about a listed item in less than 30 seconds.

Security

As a developer,

1. I would like to use JWT tokens for session management and also for making protected backend routes (i.e., they shall work only with valid access tokens generated upon authentication) so that a random person can't simply make requests to our server and have access to all the data.
2. I would like to use NodeJS encryption framework like bcrypt.js in order to hash the user passwords before storing them in the database, so that the sensitive information remains protected and there is no compromise on the security.

As a user,

1. I would like to have all my information stored in a secure database so that any malicious activity won't compromise my personal data like email, password, etc.
2. I would like to make sure that I am dealing with authentic users in a safe environment so that I am not duped when I meet the other person for the final transaction.
3. I would like to have the ability to report users for inappropriate behaviour so that I can ensure that the app provides a safe and secure atmosphere for every user.

Usability

As a user,

a. I would like the application to be easy and intuitive to navigate.
b. I would like the application to be ready to use upon installation and launch.
c. I would like the interface to be responsive to my actions.

Hosting/Deployment

As an administrator,

a. I would like the server to support server-client communication.
b. I would like the server to be able to store data in a database.

As a developer,

a. I would like to access and update the database.
b. I would like to post the application on the App Store so that users can download it easily.

As a user,

a. I would like the database to be reliable so that my data is secure.
b. I would like to download the application from the App Store.

Scalability

As a developer,

a. I would like to add server resources from our provider if traffic increases.
b. I would like to add new university networks to the application with minimal effort.

Design Outline

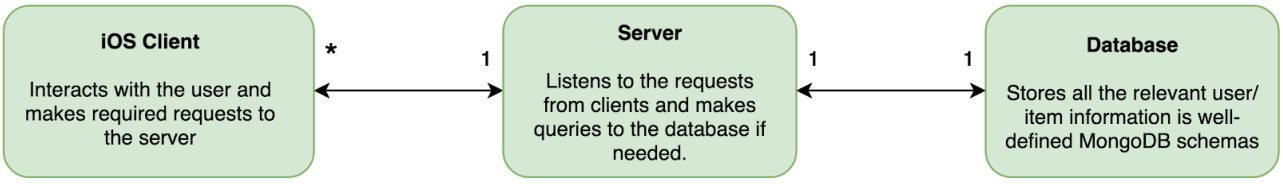
High Level Overview

Merchant will be an iOS application that allows users to sell and buy items through the app in a closed community. A Client-Server model will be used by the app. A single server will handle requests from multiple clients. When a client requests data from the server, the server will access data stored in a database and return the requested information. When a client needs to store data, it will send it to the server which will then store the data in the database. The server will allow access to multiple concurrent iOS clients via a web API.

1. **iOS client**
    a. Each user would have an iOS app on their phone which would be how the user interacts with our services.
    b. iOS clients would receive and send data to our web server through HTTP requests. The data transfer would be of JSON format.
    c. Data from the server will be parsed in Swift compatible data structures and would then be displayed to the user.

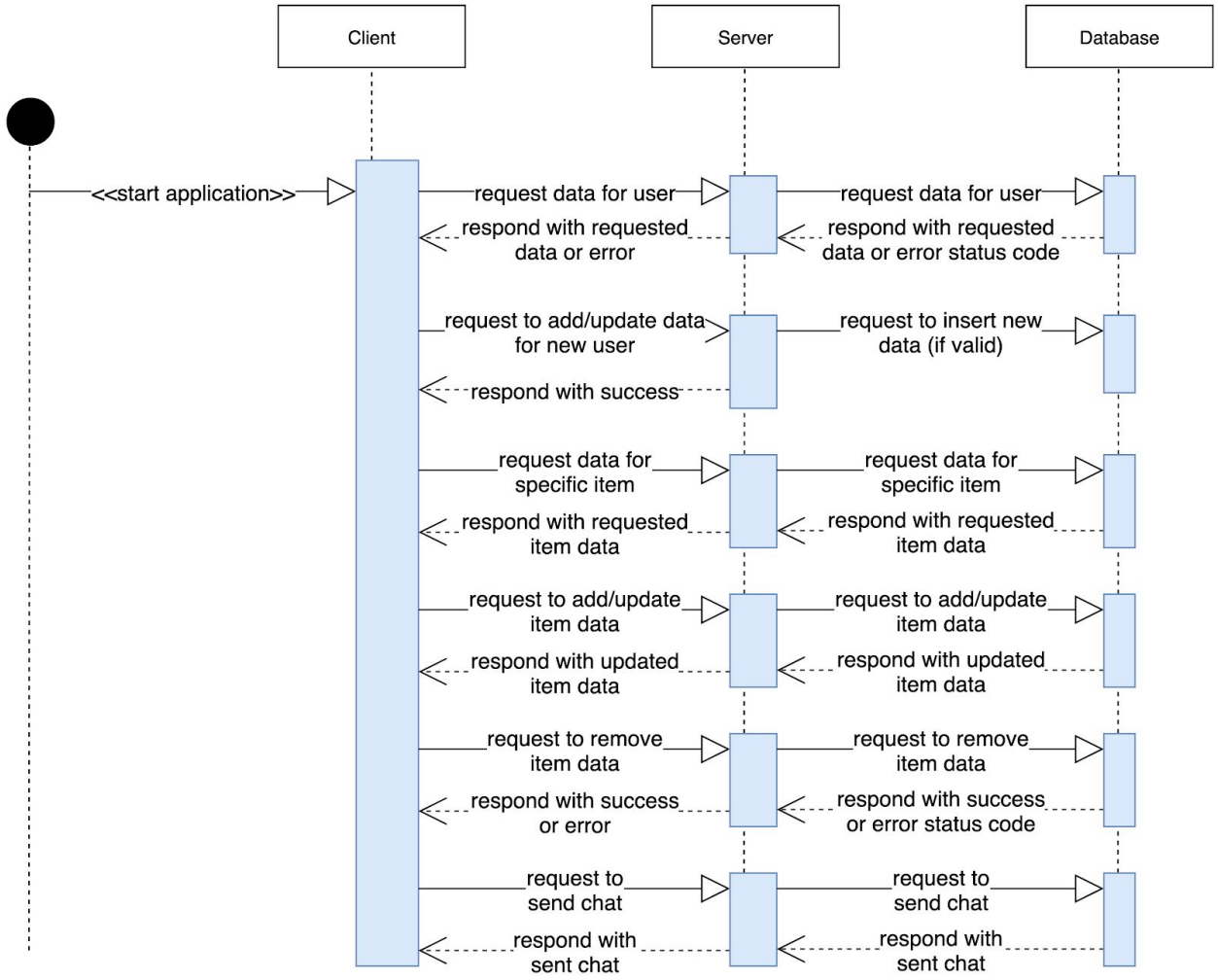
2. **API server**
    a. We would have a node server which would handle requests and serve as the connection link between our clients and database.
    b. The server would handle HTTP requests from the client and fetch the requested data from the database and return the data to the client.
    c. It would take in the data from the client and store it in the database.

3. **Database**
    a. All of the data that is used in Merchant will be stored in the database.
    b. The database would store information regarding users, listed items, encrypted chat history, etc.
    c. It would respond to the queries made by the server as required by the query.



Sequence of Events Overview

This sequence of events diagram shows the standard interaction between a client, the server, and the database. Once a user opens the app, the sequence begins and the user logs in. As soon as the user starts logging in, a request is sent to the server from the client and then the server handles this request. To finish logging in the client, a query is sent to the database and then the database sends the required data back to the server. As the user continues to use the app, the client sends more requests to the server. These requests will be made in order to perform actions like viewing items on sale, posting items for sale, and managing user accounts. The server sends queries to the database in order to obtain the necessary data to complete these actions. Finally, the server returns this information to the client.



Design Issues

Functional Issues:

1. What information would be required by the user in order to sign in?
    a. Option 1: Username and password
    b. Option 2: Username, password and university email address
    c. Option 3: Username, password, phone number and university email address

    Choice: Option 2

    Justification: The only requirement to sign up for an account on Merchant is that the user should have a university email to verify that the person is a student going to the university. This is so that Merchant is a closed and a safe place for students to sell and buy items from 'known strangers.' During the time of sign up, Merchant would send an email to the new user to confirm their university email with an OTP.

2. What information should be displayed on the home page?
    a. Option 1: User data and information
    b. Option 2: list of items that I can buy as a user
    c. Option 3: list of items that i can sell as a user

    Choice: Option 2

    Justification: The main purpose of Merchant is to buy items from other students and the fastest way to get there is to have items which can be bought on the home page. Moreover, it fits consumer psychology because an average user would use any e-commerce app more for buying than selling.

3. How much information per item should be displayed when listing various items that a user can buy?
    a. Option 1: photo and title
    b. Option 2: photo, title and price
    c. Option 3: photo, title, price and description

    Choice: Option 2

    Justification: The listing should have enough information to get the user interested in the item but not too much information. The user would be able to click on the listing to get detailed description and contact the user who posted the item for sale.

4. How would a user buying the item contact the seller?
    a. Option 1: Phone number
    b. Option 2: chatting with other users identified by their usernames
    c. Option 3: email

    Choice: Option 2

    Justification: Having users chat with other users identified by their usernames is a way to protect personal information and an efficient way to communicate at the same time. Email would be too slow for discussing the specifics of the product. Also in app chatting would enhance user experience.

5. How to set up a location to meet and complete the transaction?
    a. Option 1: have predefined locations for meet up places.
    b. Option 2: have users decide a location for transaction.
    c. **Option 3: have an interactive map which would have suggestions for safe locations to perform the transaction and at the same time would allow users to pick a transaction location according to their convenience.**

    Choice: Option 3

    Justification: Having safe locations and at the same time allowing users to decide on a meetup location gives both flexibility and credibility to the app. It allows users to have safe and convenient locations for transactions. An interactive map would help the user to finalize the location better.

6. What information must the user provide to post an item?
    a. Option 1: photo, price, title, category and description
    b. Option 2: title and price
    c. Option 3: photo, title and price

    Choice: Option 1

    Justification: While posting an item, it is important that as much information as possible is provided by the seller so that buyers can fully understand the details of the item on sale. By requiring a photo, price, title and description, this will be adequate information for buyers to decide whether or not they would like to purchase the item.

Non-Functional Issues:

1. What frontend technology should we use?
    a. Option 1: React (Web App)
    b. Option 2: React Native / Flutter (Cross Platform App)
    c. Option 3: Swift (iOS App)
    d. Option 4: Android Studio (Android App)

    Choice: Option 3

    Justification: We decided to make an iOS app because iPhones are really popular among the University students accounting for almost 50% of the market share in the USA, and with apple planning to come with cheaper versions in future, it is only going to increase. We also had the option of making a cross platform app using React Native / Flutter, but based on articles like **[1]**, we concluded that a cross platform app would mean a compromise on user experience. Moreover, we dropped the idea of making a web app because given the main idea of our app, web app wouldn't serve the purpose.

2. Which backend technology should we use?
    a. Option 1: NodeJS
    b. Option 2: Python Flask
    c. Option 3: Java Spring Boot

    Choice: Option 1

    Justification: We decided to work with a NodeJS server because it is really easy to set up servers in NodeJS using frameworks like express which require very little boilerplate code. Moreover, Javascript allows us to use JSON (an industry standard for HTTP communication), directly in our code without using countless JSON parsers (unlike Java and Python) which makes it really intuitive to use. Apart from that, NodeJS has a really good framework called Mongoose to interact with our MongoDB database in a very optimised way. Additionally, all the team members have some experience of working with Javascript which made it a natural choice.

3. Which type of database should we use?
    a. Option 1: Relational Database (SQL)
    b. Option 2: Non-relational Database (MongoDB)

    Choice: Option 2

    Justification: We decided to use a non-relational database because our storage requirements are better suited to it. Our schema is not very deeply nested, hence the performance, scalability and implementation would be better and easier. Moreover, setting up and deploying a MongoDB database is much easier, and MongoDB Atlas integrates well with Azure, which is the cloud hosting service we intend to use.

4. Which cloud hosting service should we use?
    a. Option 1: Azure
    b. Option 2: Heroku
    c. Option 3: Amazon Web Services (AWS)

    Choice: Option 1

    Justification: We had the option to use Heroku as a service for hosting our server for free. But we ruled it out because it works on free dynos (light-weight linux containers) which sleep if there is inactivity. Hence, when the first request is made to the inactive server, it's response time is conspicuously high. This brought us down to AWS or Azure. Azure was the natural choice because the CS department provided us free credits for Azure, while it couldn't support AWS. What is more, Azure offers a lot of additional useful services (e.g.: messaging service) which might be useful for us in while implementing other features.

5. How to mark a posted item as sold?
    a. Option 1: Let the seller (user) mark an item as sold. A query would update the database upon a button press from the client side.
    b. Option 2: Send a query to the database automatically after the meeting time decided by the buyer and seller ends.

    Choice: Option 1

    Justification: We chose to let the seller manually mark an item as sold because it is not necessary that the transaction gets completed every time the buyer and seller schedule

a meeting. Moreover, keeping track of so many meeting end times and scheduling queries would prove to be more costly for the backend than the first option.

6. Which emailing service should we use for sending out verification emails?
    a. **Option 1: Twilio Sendgrid**
    b. Option 2: Gmail API
    c. Option 3: Nodemailer

    Choice: Option 1

    Justification: We chose Twilio Sendgrid as our emailing service because it is one of the easiest ways to implement emailing service for free on NodeJS. It is free and also supports emails to all the domains. Gmail API is better suited for a python based back-end as there is not a lot of documentation available for Node based servers. Nodemailer, on the other hand is harder to implement, and sometimes the latency between the email request and the actual delivery of the email is too high.

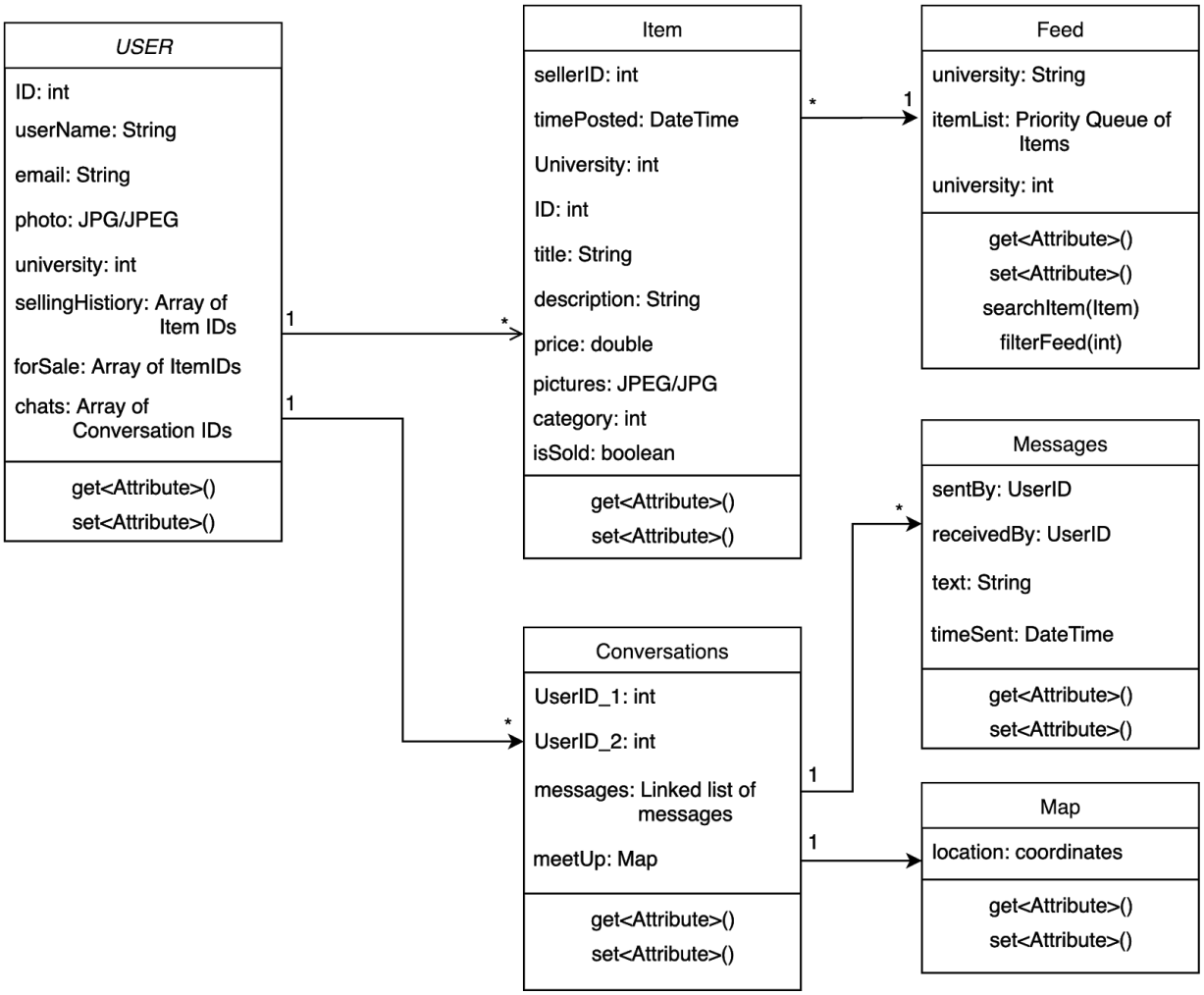
7. Which geo-location service should we use for implementing location sharing?
    a. Google Maps
    b. Algolia API
    c. Telize
    d. **Apple MapKit**

    Choice: Option 4

    Justification: We chose the apple mapkit because it is more optimized, has a smaller memory footprint, and its style is more aligned with iOS patterns. Moreover, it is easier to integrate with the App because it is Apple's own technology, while also keeping our external dependency count low.

Design Details

Class Design



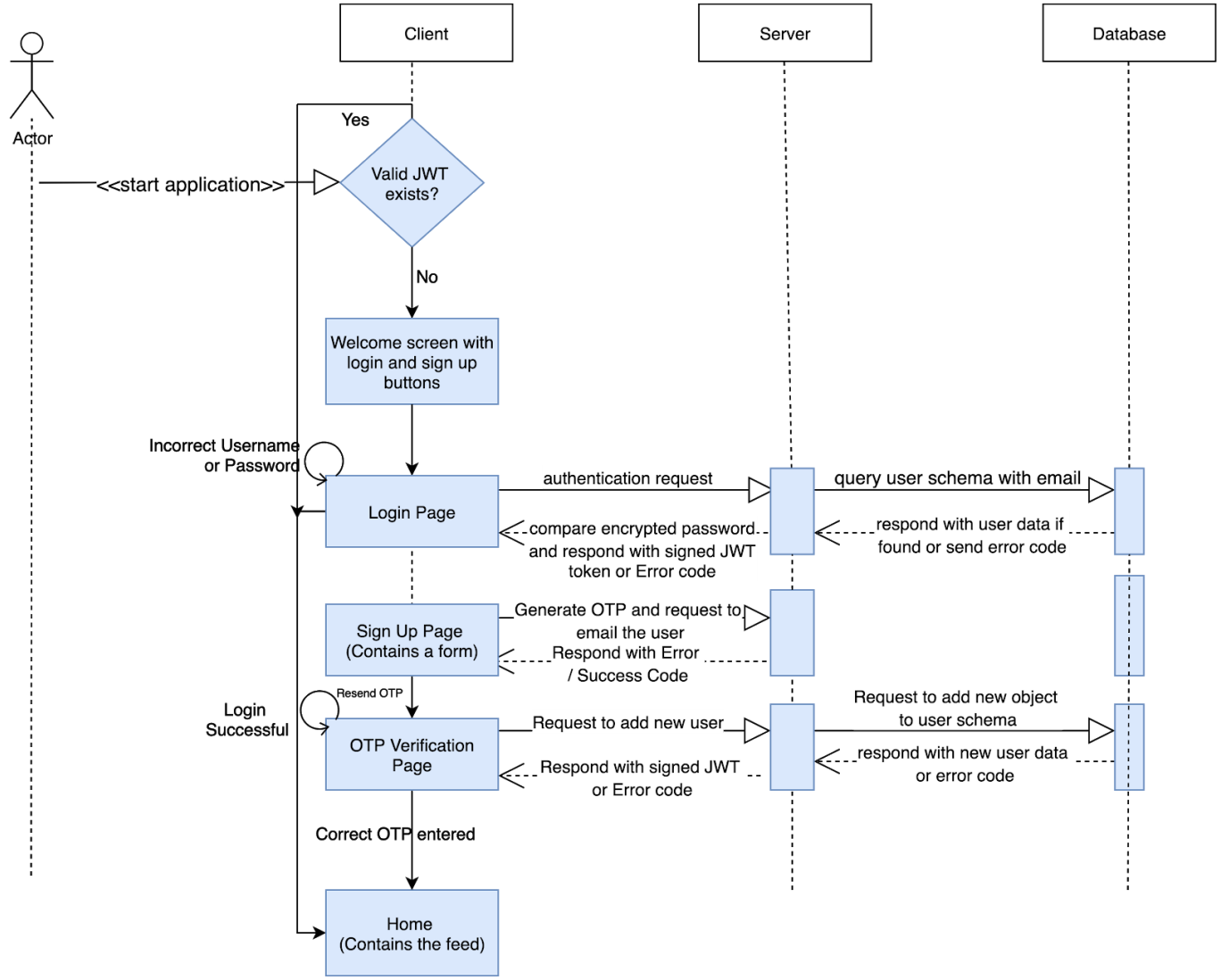
Description of Classes and Interaction Between Classes

- **User**
  - User object is initialized when they use a one-time password to verify their email.
  - Each user is assigned an integer ID.
  - Each user has an account linked to a unique username, email and password.
  - Each user has a default profile picture which can be changed if desired.
  - Each user belongs to a single university which matches their email address.
  - Each user has chat conversations with other users who they are trading with.

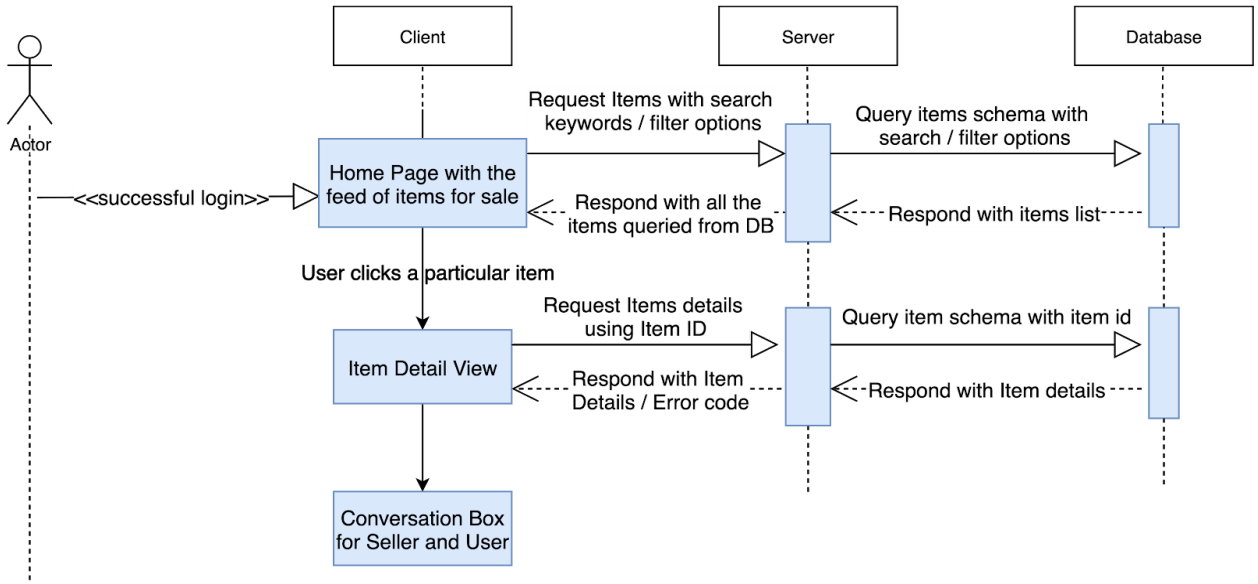
    - Each user can view the items that they are currently selling as well as their sales history.
- **Item**
    - Item object is initialized when a user posts an item for sale.
    - Upon initialization, items are marked as not sold. They are marked as sold when the seller marks them as sold.
    - Each item belongs to a seller and contains the seller ID.
    - Each item for sale is displayed in the feed.
    - Each item has a title, description, and price which is created by the user who is selling it.
    - Each item belongs to a single university.
    - Each item has a unique integer ID.
    - Each item has a category which the seller selects from a predetermined list of categories.
- **Feed**
    - The feed is a UI component that displays the list of all items for sale and is initialized upon launch of the application.
    - The feed contains the same pool of items for every user in a specific university network, including the sellers of the items.
    - Individual users can filter their feed by category or price.
    - Individual users can sort their feed by price.
    - Users can search for items by title in the search bar.
    - The feed displays information about items and takes users to the listing when they press on a given item.
- **Conversations**
    - A conversation is initialized between two users when the buyer inquires about the seller's item.
    - Conversations are between two users.
    - Each conversation has a message history for the entire conversation.
    - There is a map in the conversation view for users to decide where to meet safely.
- **Message**
    - New messages are created when one user sends a message to another user.
    - Each message has a string of text and the date and time it was sent.
- **Map**
    - There is a map in each conversation between users
    - Each map contains predetermined safe meeting spots where there are Emergency Telephone System poles.
    - Users can select a given spot on the map to share with each other to decide where to meet.

Sequence Diagrams

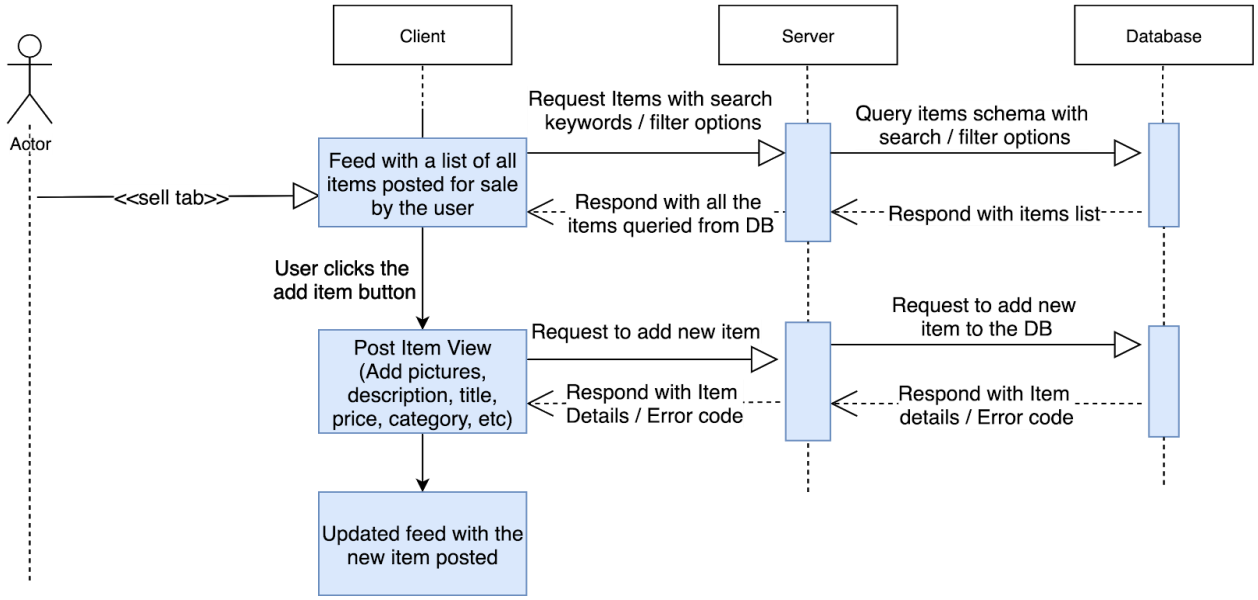
1. When the user first opens the app:



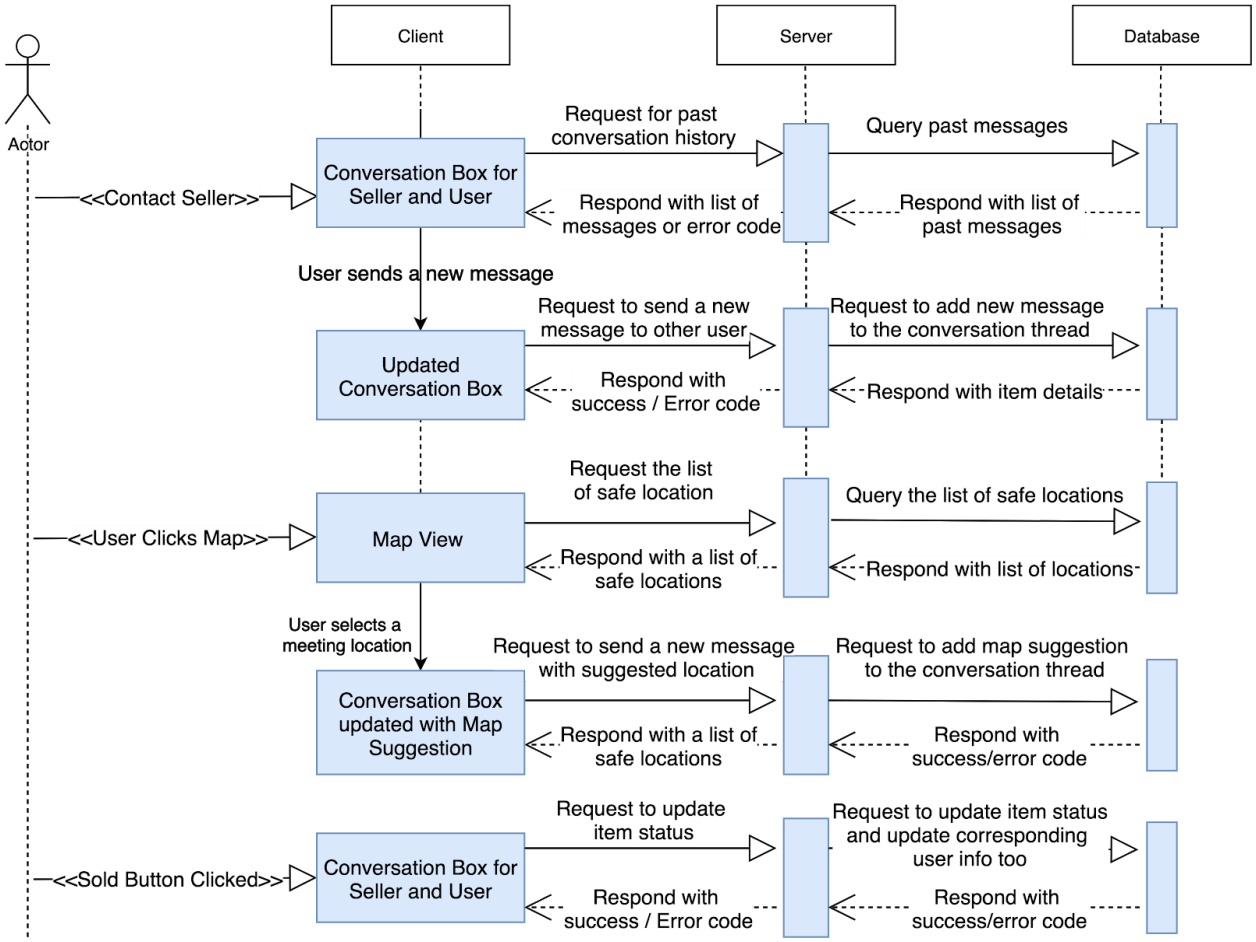
2. When the user is on the home page and wants to buy an item:



3. When the seller wants to post a new item

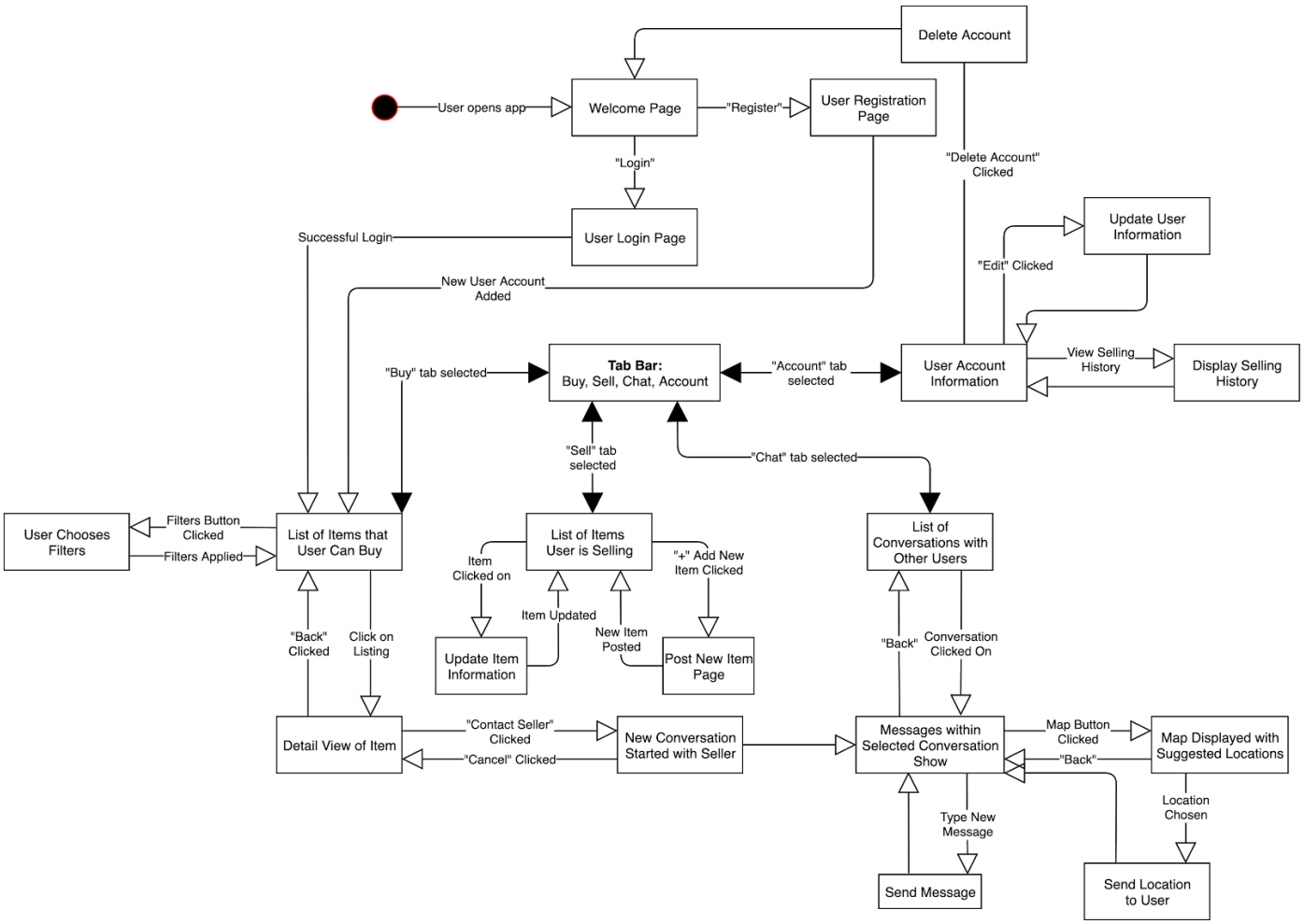


4. When the user wants to communicate with the seller:



Navigation Flow

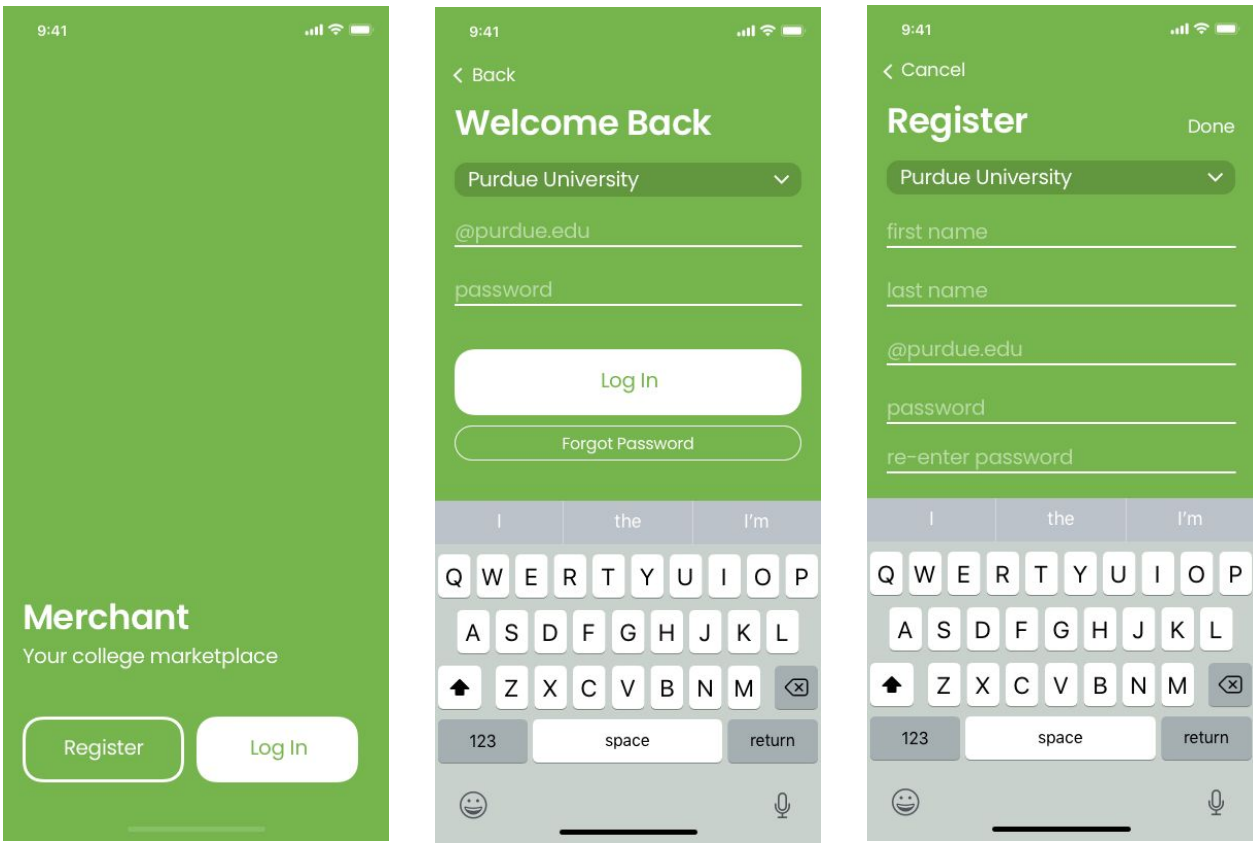
Navigation of the Merchant app has been designed to let users clearly distinguish the main features of the app. We initially prompt users to register or login with an existing account. Then, they are brought into the main part of the app which is organized by four tabs: Buy, Sell, Chat, and Account. These tabs allow users to access all of the main features quickly and easily at any time while using the app. The default tab is the “Buy” tab. The solid black arrowheads indicate the tab bar navigation element. From each tab, actions and their corresponding navigation flow are layed out. This diagram shows how central the tab bar design is and how it connects all elements of the app.



UI Mockups

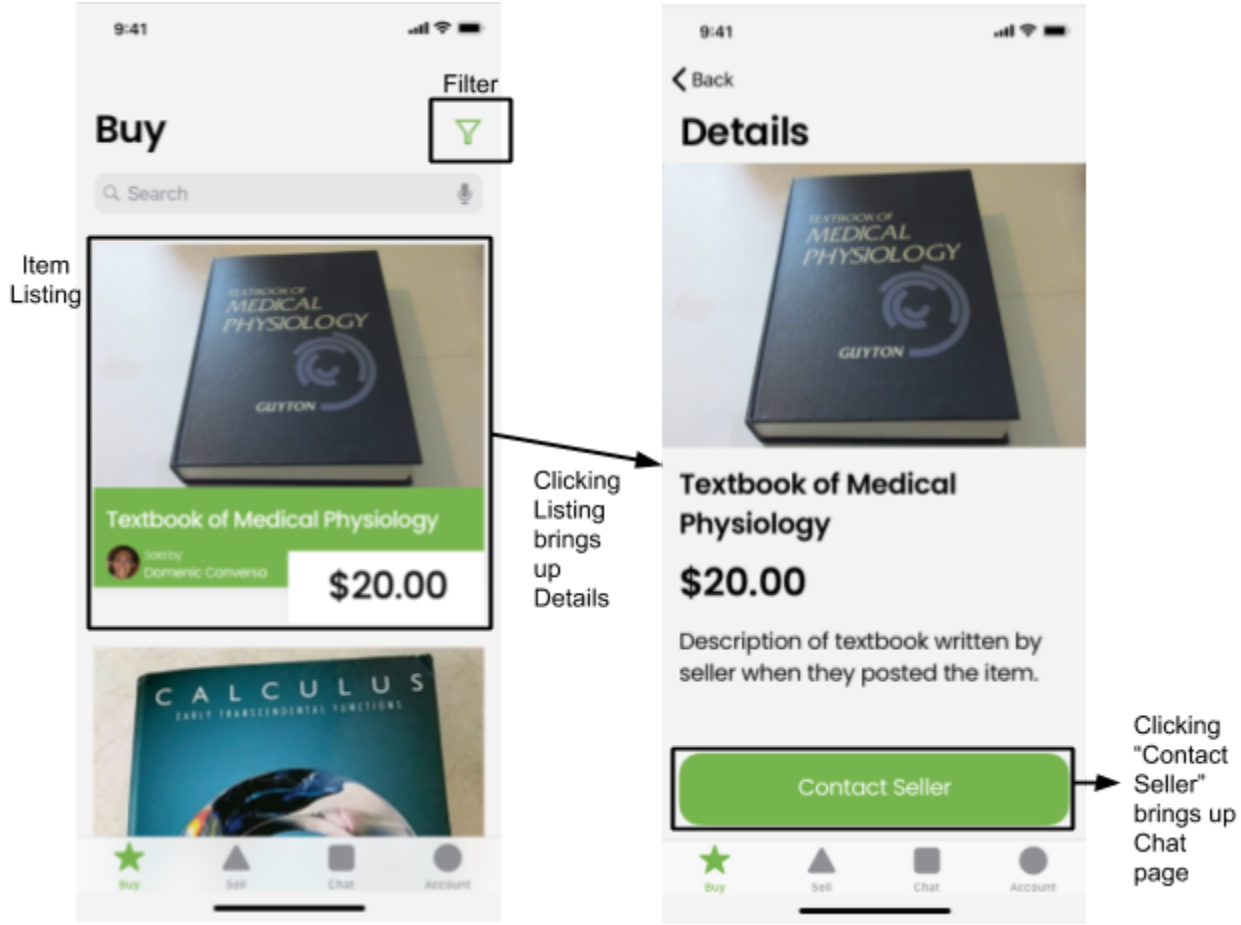
This user interface has been designed to be as clear and simple for the user as possible. The overall design structure is a tabular design with 4 tabs: Buy, Sell, Chat, and Account. These tabs allow the user to understand which part of the app they are currently interacting with and allow for quick access to all key features.

Register or Login



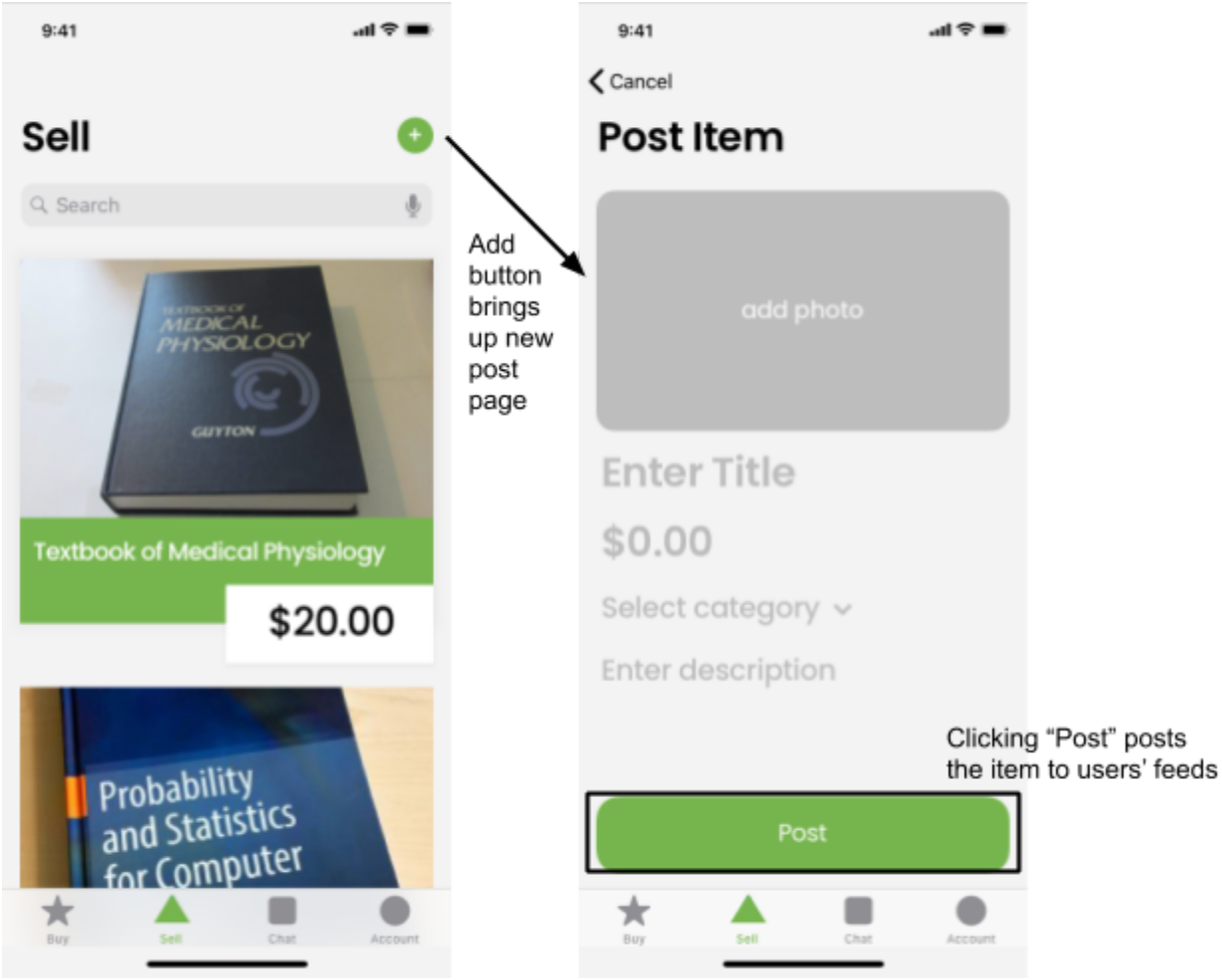
The welcome page is designed to allow users to quickly login if they already have an account or to register for a new account if they do not have one. The login page has the user select their university so that they are connected to other users at their school. It also prompts the user to enter their email and password to log in while providing a button for the user to click to reset their password if they forget it. The register page is similar to the login page, however it also collects the user's first and last name and prompts the user to re-enter their password to ensure the password they choose is the one they want.

Buy



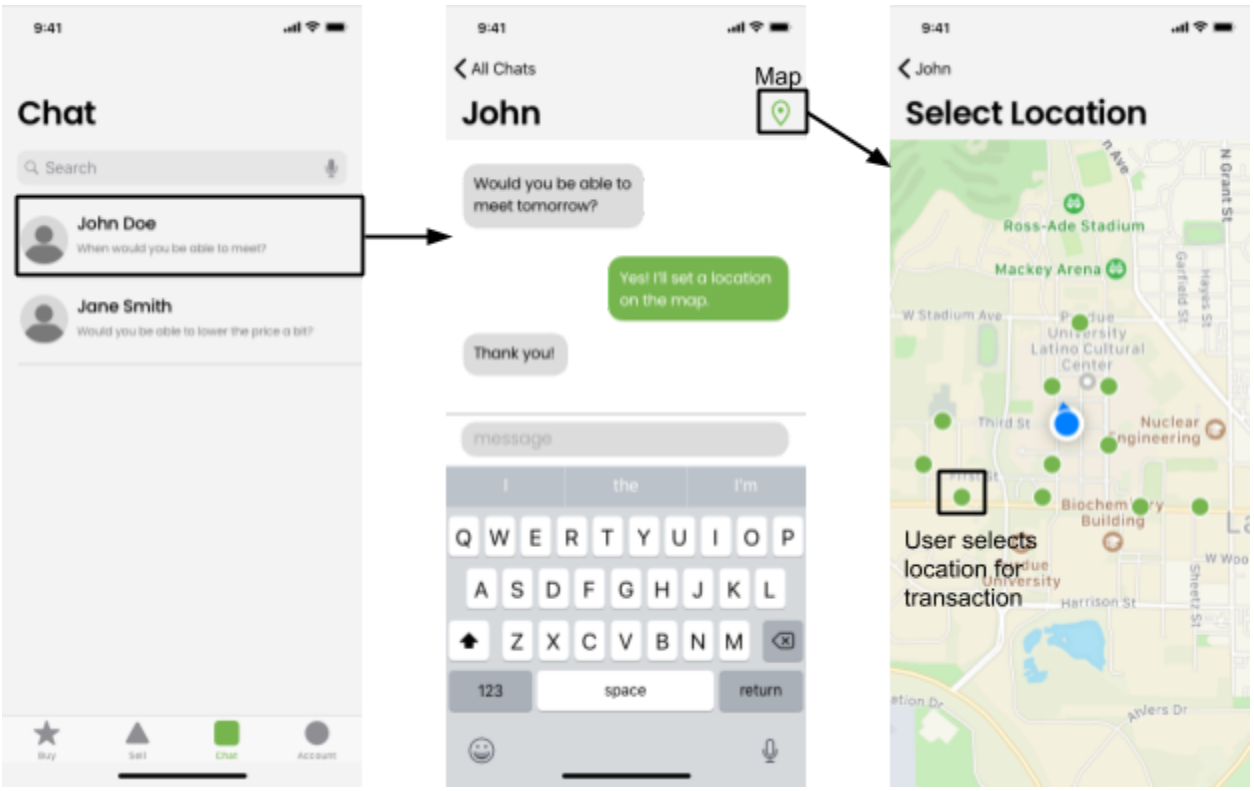
The Buy tab contains a list of all of the items that are available for a user to buy. Each item listing shows the title of the item, a photo of it, its price, and also who is selling it. The user can scroll through all of the available items and click on an item to view more details. Additionally, there is a search bar for users to be able to search for a specific item and a filter button to let users filter items based off of their category. The Details page shows the same information as the listing but it also includes a description and the “Contact Seller” button. The user can click the “Contact Seller” button to begin chatting with the seller about completing the transaction for buying the item.

Sell



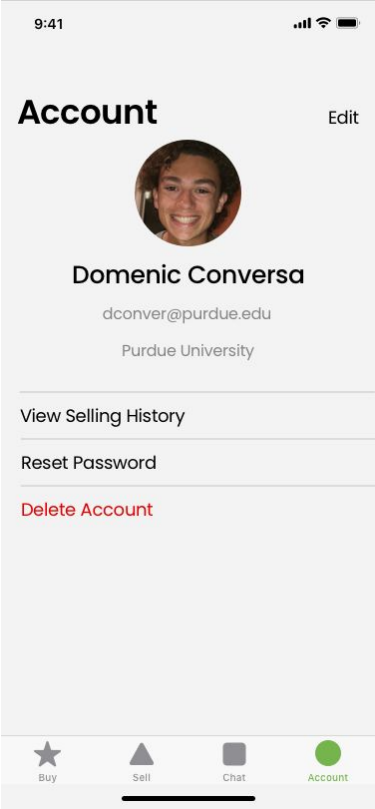
The Sell tab lists all of the items the user who is logged in is currently selling. Each listing shows the item's picture, title, and price. Clicking on an item will allow the user to edit the item's information. There is a "+" button to allow users to post a new item to sell to other users. Once they click the "+", the user is prompted to add the necessary information for selling an item.

Chat



The Chat tab contains all of the conversations that the logged in user has with other users about buying items. The user can click on a specific conversation and see the messages that have been sent between them and another user. From there, they can click on the map button in the top right and view the map. This map has suggested locations for the users to meet up in order to complete the transaction. From this map, the two users can agree on a place to meet up and complete the transaction.

Account



On the Account tab, users can edit and update their information. They also have access to viewing their selling history, resetting their password, and deleting the account if they no longer want to be registered as a Merchant user.

References

[1]https://codeburst.io/native-vs-cross-platform-app-development-pros-and-cons-49f397bb38ac