

INDEX

1. Purpose

- 1. 1. Functional Requirements
- 1. 2. Non-Functional Requirements

2. Design Outline

- 2.1 Components
- 2.2 High-level Overview
- 2.3 Flow of events

3. Design Issues

- 3.1 Functional Issues
- 3.2 Non-Functional Issues

4. Design Details

- 4.1 Class Diagram
- 4.2 Descriptions of Classes and Interactions
- 4.3 Sequence Diagrams
- 4.4 Activity Diagrams and UI Mockups

PURPOSE

Students are often in a situation where they have too much stuff or not enough. For people with too much stuff, what often happens is they simply throw away those items, even if they were still usable. If they're more industrious, they might try to make a post on FaceBook to sell it to whoever may see it, and arrange all the details by themselves. This not only limits the span of available customers (posts would only be visible to people within your network of friends), but also creates an unnecessary amount of work for the people involved.

For those with not enough stuff, what often happens is they are forced to purchase items brand new from retail or department stores. While this is certainly necessary for some items, there are plenty of other things that students may need but don't necessarily need brand new. This could include furniture, textbooks, electronics, and so much more; what's worse is most of those items brand-new cost a small fortune, and most students are already on a tight budget.

Our app seeks to address that issue by allowing students to buy and sell used items at a fraction of what their costs would be brand-new in-store. It will serve as a central hub for students to make posts for items that they no longer need--and earning themselves a bit of extra cash, while allowing other students to find those items they need without having to pay full-price. The app also extends the scope of visibility with posts: rather than having a post only visible to people within your social network, anyone close-by with this app can see your post, allowing you to find buyers more easily! UniTrade will serve as a convenient online trading-post for students to buy items they need or sell items they no longer need without any hassle.

1.1. Functional Requirements

1. Users can create and manage an account

As a user,

- I would like to register an account.
- I would like to reset my password and retrieve my username.
- I would like to view and edit my profile.
- I would like to change the theme of the App.
- I would like to change notification settings.

2. Users can browse items and see an item's details

As a user,

- I would like to find items by their category or keywords.
- I would like to view an item's details
- I would like to sort the posts I see by type, price, distance, or rating.
- I would like to see nearby items.

3. Users can save items in different ways

As a user,

- I would like to add an item to my wishlist.
- I would like to view my wishlist.
- I would like to see my purchase/sale history.
- I would like to know when the price changes for items in my wishlist.

4. Users can sell or purchase items

As a seller,

- I would like to post my items to sell.
- I would like to set the price of the item.
- I would like to add details about the item.
- I would like to include pictures of the item.
- I would like to receive payments online when the trade is made.
- I would like to be notified when a buyer has purchased my item.

As a buyer,

- I would like to make payments online.
- I would like to choose between a face-to-face exchange or shipping
- I would like to know the tracking number if I chose to have the item shipped
- I would like to see a seller's profile.
- I would like to be notified when a seller ships my item.

5. Users can give or receive feedback from other users

As a user,

- I would like to view the rating and comments of other users.
- I would like to review users after a transaction.
- I would like to report improper actions of other users.

- I would like to send messages to other users when a trade is being made.

As an administrator,

- I would like to see reports from users.
- I would like to delete inappropriate items, posts, comments, or users.

1.2 Non-Functional Requirements

1. Client Requirements

As a developer,

- I would like to run this app on Android devices.

2. Server Requirements

As a developer,

- I would like the server to be able to save data to the database.
- I would like the database for the passwords to be encrypted.

3. Design Requirements

As a developer,

- I would like to have frontend and backend separate.
- I would like the user interface to be clear and simple to use.
- I would like to implement online payment through Paypal's API.
- I would like the app to have the ability to calculate distances between users through Google Maps' API

4. Appearance Requirements

As a developer,

- I would like the user interface to be clear and simple to use
- I would like the user interface to be visually appealing

5. Performance Requirements

As a developer,

- I would like the system to be able to have the capability of 100 users at one time.

- I would like the server to be able to respond in under 800 milliseconds.

6. Administrative requirements

As a developer,

- I would like to run an administrator interface on a computer.

DESIGN OUTLINE

2.1 The components of the system

Our project is an Android application that allows users to buy and sell used items like textbooks, course equipment, electronics, and furniture. It will serve as a central hub for students to make posts for items that they no longer need--and earning themselves a bit of extra cash, while allowing other students to find those items they need without having to pay full-price. Our implementation will use a Client-Server model. The server will control queries from multiple users and will also be able to access user profiles and item information stored in the database.

Client:

- a. The client provides user an interface with our system
- b. The client will send HTTPs requests to the server.
- c. The client will receive responses from the server.

Server:

- a. The server will handle and retrieve all requests from the Android client.

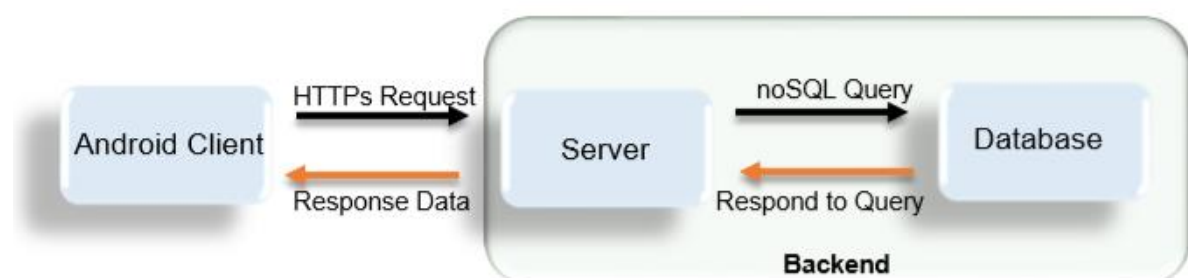
- b. The server will validate requests before processing.
- c. The server will send queries to the database to add, modify and retrieve data on demand.

Database:

- a. The database will store all of the data used in our app, things such as user profiles, item posts, or messages between users.
- b. The database will respond to queries from the server and send the requested data back to the server.

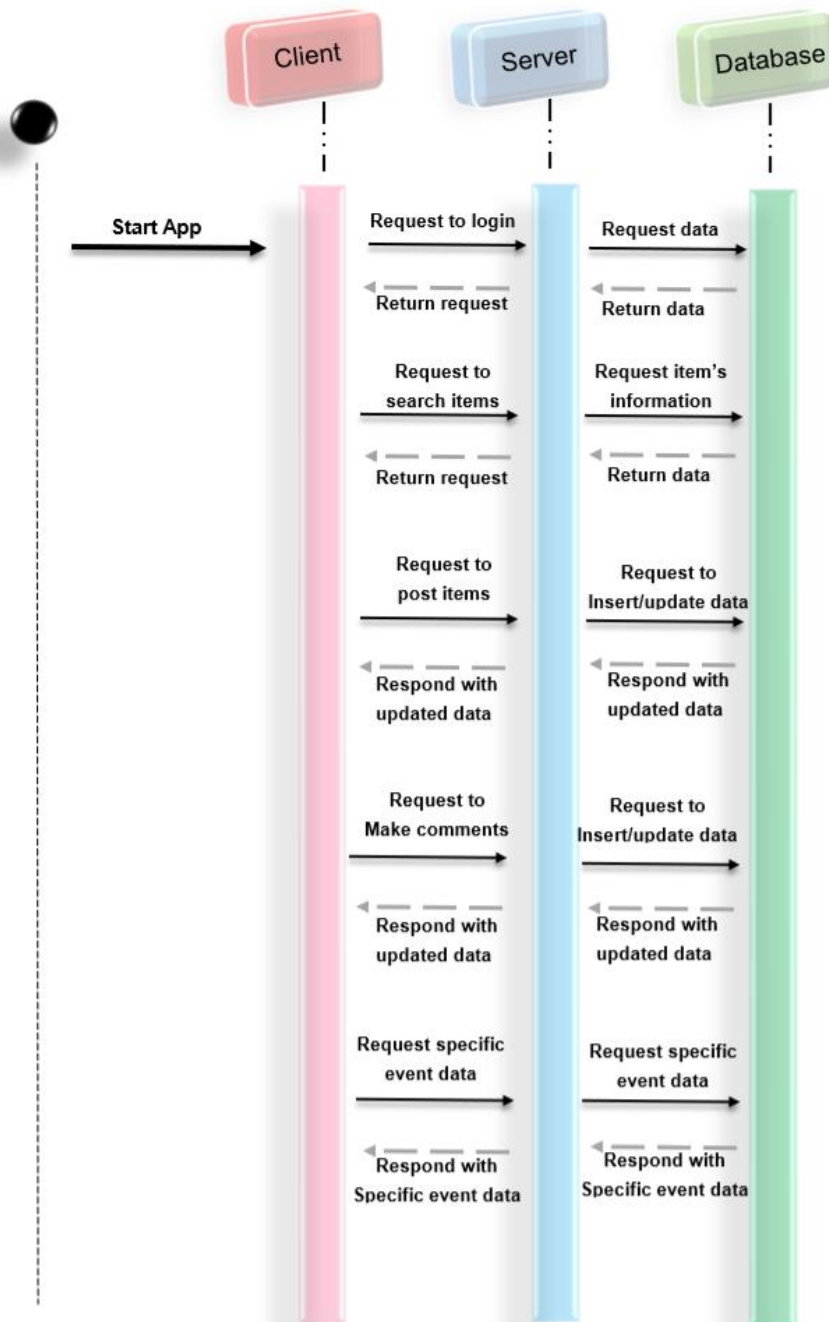
2.2 High Level Overview of the Client/Server/Database

The UniTrade application will use the client-server model. The client and server will form a many-to-one client-server architecture to serve a large number of users at the same time. The client uses the network as a way to connect with and speak to the server, as well as send and receive communication about its request. Then, the server will take the request and make sure that the request is valid. Once the request is validated, then the server queries the database for information and send it back to the client.



2.3 Flow of Events Overview

The sequence diagram below shows the typical interaction among clients, server and database. The sequence is initiated by a user starting the UniTtrade application. First of all, when users type in their username and password, the client sends a request to the server as the user click “login” button. Next, the server validates the request and then sends a query to the database to retrieve data from the database. After users log in their accounts, the client continues to send further requests as the user having more actions by clicking the corresponding buttons. In our main user interface page, users can make further actions like searching for the items they want to buy, posting items they want to sell, talking to a seller, making comments of the item you bought, etc. To response to these requests, the server sends queries to database to acquire data from the database. And after receiving updated requests, the database responds to server with updated data. Finally, the server returns the results back to the client with the requested data from the database.



DESIGN ISSUES

3.1 Functional Issues:

1. What payment options should we support?

- Option 1: online
- Option 2: online & cash

Choice: online & cash

Justification: At first, we think most people would prefer to use online payment, since it's quicker and often easier to do. However, Unitrade is an app that encourages face-to-face trading, so some users may prefer to pay with cash once the item is received. Also, we will use paypal API to accomplish online payment, otherwise the buyer may choose the cash-payment option. If time permits, we'd like to include other forms of online payment.

2. Is a phone number necessary for creating a new account?

- Option 1: Yes, it is required
- Option 2: No, it is not required

Choice: No

Justification: We'll encourage users to include their phone number in their profile, but we won't require users to include it in order to verify their accounts. We will require users to verify their accounts through email, so a phone number won't be necessary.

3. Should we should allow users to leave anonymous ratings and reviews.

- Option 1: Yes, we should allow anonymity
- Option 2: No, we should not allow anonymity

Choice: No

Justification: While some users may prefer to remain anonymous when writing a review--especially when it's a bad one--this feature may allow users to abuse other users with unnecessarily harsh or otherwise offensive reviews without any sort of consequences. We believe that users should stand by what they say about other users.

4. Is an administrative profile necessary for this app?

- Option 1: Yes, it's necessary
- Option 2: No, it's not necessary

Choice: Yes.

Justification: An administrator is necessary in order to create a safe and friendly environment for users. An administrator can receive reports from users and monitor posts to ensure that inappropriate content does not persist within the app.

5. When should we acquire a user's location data?

- Option 1: Continuously acquire it even when the user is offline
- Option 2: Periodically acquire it when users are online
- Option 3: Only acquire a user's location when absolutely necessary

Choice: Option 2 and 3

Justification: We don't constantly need a user's location for this app. However, one of the key features of the app is being able to find nearby items from the user. As such, the app requires a location of the user in order to calculate this. While users can choose not to have location services activated for the app, this would require them to input an address in their profile (which would then be periodically retrieved) in order to continue using that feature. Otherwise, the only other time we would need to retrieve addresses is for creating posts, from which the user can choose to set an item's address as their own or a different one.

3.2 Non-functional Issues:

1. What backend web service should we use

- Option 1: Amazon Web Services
- Option 2: Firebase (Google)

Choice: Firebase.

Justification: Since the majority of our group has little experience with web server infrastructure, we'd like to work with something that is more user-friendly and simple to set-up. We also found that using Firebase would allow us to better integrate Location Services using Google Maps' API.

2. How should we arrange UniTrade's UI layout?

- Option 1: Use a relative layout
- Option 2: Use a static layout

Choice: relative layout.

Discussion: Even though a static layout is very simple to implement, the app would be more visually appealing if we allow it to properly adjust relative to the phone's screen size or orientation.

3. Which IDE should we use to develop Unitrade?

- Option 1: Android Studio
- Option 2: AIDE
- Option 3: Xamarin

Choice Android Studio.

Discussion: All of our group's members have experience with android studio. Currently, we plan on making UniTrade as an Android-only app rather than having it work on multiple platforms due to time-constraints. As such, so we'd like to use an IDE that focuses more on android development. Usually AIDE is used to develop some small projects on android devices, and while Xamarin can do android development, it is more suitable for iOS development but not android.

4. What language is appropriate for implementing server?

Option 1: C/C++

Option 2: Python

Option 3: Java

Choice: Java.

Discussion: For convenience, we'd prefer to use same language in our front-end and backend. Java serves as an excellent language for dealing with back-end server management and is also supported by Android Studio.

5. What design model should we use between the client and the server?

- Option 1: thin client
- Option 2: fat client

Choice: thin client.

Discussion: Since our app often relies on updates from the server/database, we'd like to put more work on the server's end to make sure that the app remains properly updated in the user's phone.

6. For our relational database, what kind of SQL should we use?

Option 1: mySQL

Option 2: noSQL

Option 3: Oracle SQL

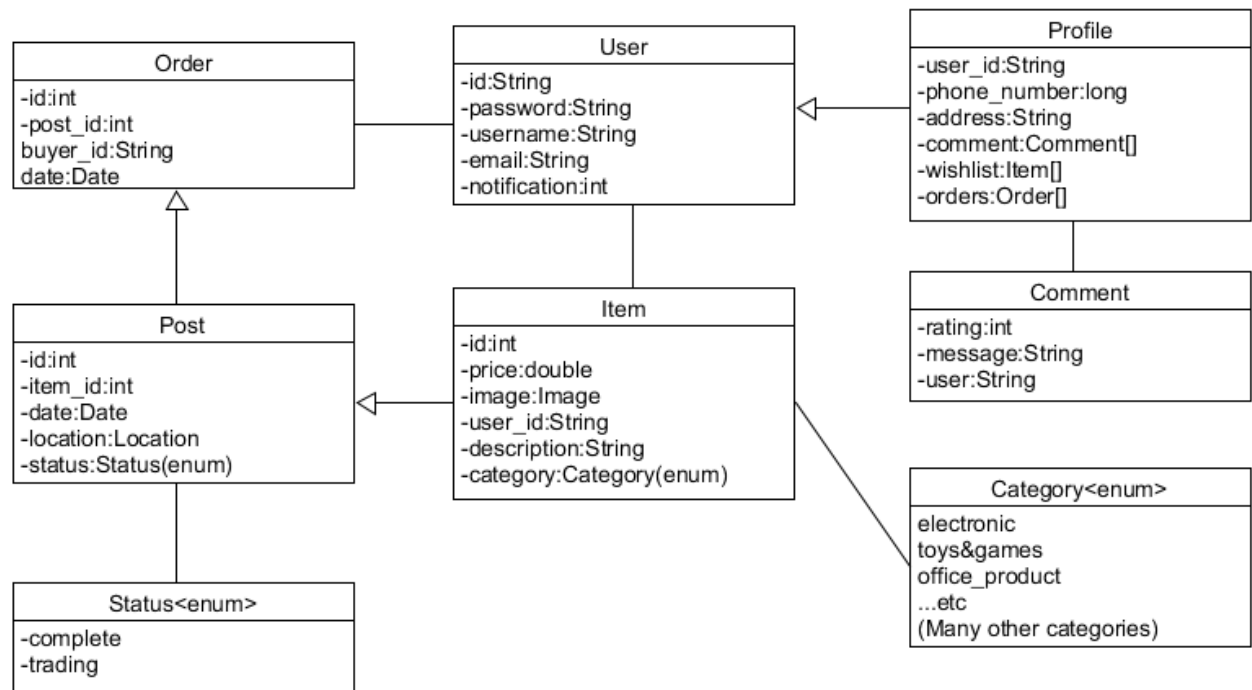
Option 4: SQLite

Choice: noSQL

Justification: The reason we chose noSQL is mainly because it's the language of choice for Firebase. Although Firebase will allow you to use other forms of SQL, it's mostly configured around noSQL.

DESIGN DETAILS

4.1 Class Diagrams



4.2 Descriptions of Classes and Interactions

User:

- The User object is created when someone signs up in our application. All users are represented as part of this class.
- Each user will be assigned a unique user ID.
- Each user will have a username, password, and email for login purposes.
- Each user can have unique notification settings
- Each user will have a profile ID of a profile object for additional details. We made the profile class for the user class mainly because there are too many detailed categories in the User class.

Profile:

- The profile object is created when a User object is created.
- Each profile will be assigned a unique profile ID.
- Each profile will contain the user's gender, address, and phone number for the basic optional information about the user.
- Each user can choose to include a profile picture
- Each profile will have a List of Items associated with items that the user has bought or sold
- Each profile will have a List of Items associated with a 'wishlist' that the user wants to save for later
- Each profile will have a List of Comments associated with that user
- Each profile will have a average value gathered from the List of Comments

Comment:

- The Comment object is created when a user makes a comment on another user's profile
- Each comment will be assigned a unique comment ID.
- Each comment will have a rating
- Each comment will have a user ID that makes the comment.
- Each comment will contain a string with the actual comment from the other user

Post:

- The post object is created when a user posts a product that he or she wants to sell.
- Each post will be assigned a unique post ID.
- Each post will have a user ID from the seller
- Each post will have the posted date and location to enable other users to sort similar products based on time-of-posting and distance from user.
- Each post will contain an availability Boolean to indicate whether it has been sold or not
- Each post will contain a deliverability Boolean indicates whether the item can be shipped or not.
- Each post will have an item ID to link to the actual item being sold in the post
- Each post will include pictures of the item being sold

Item:

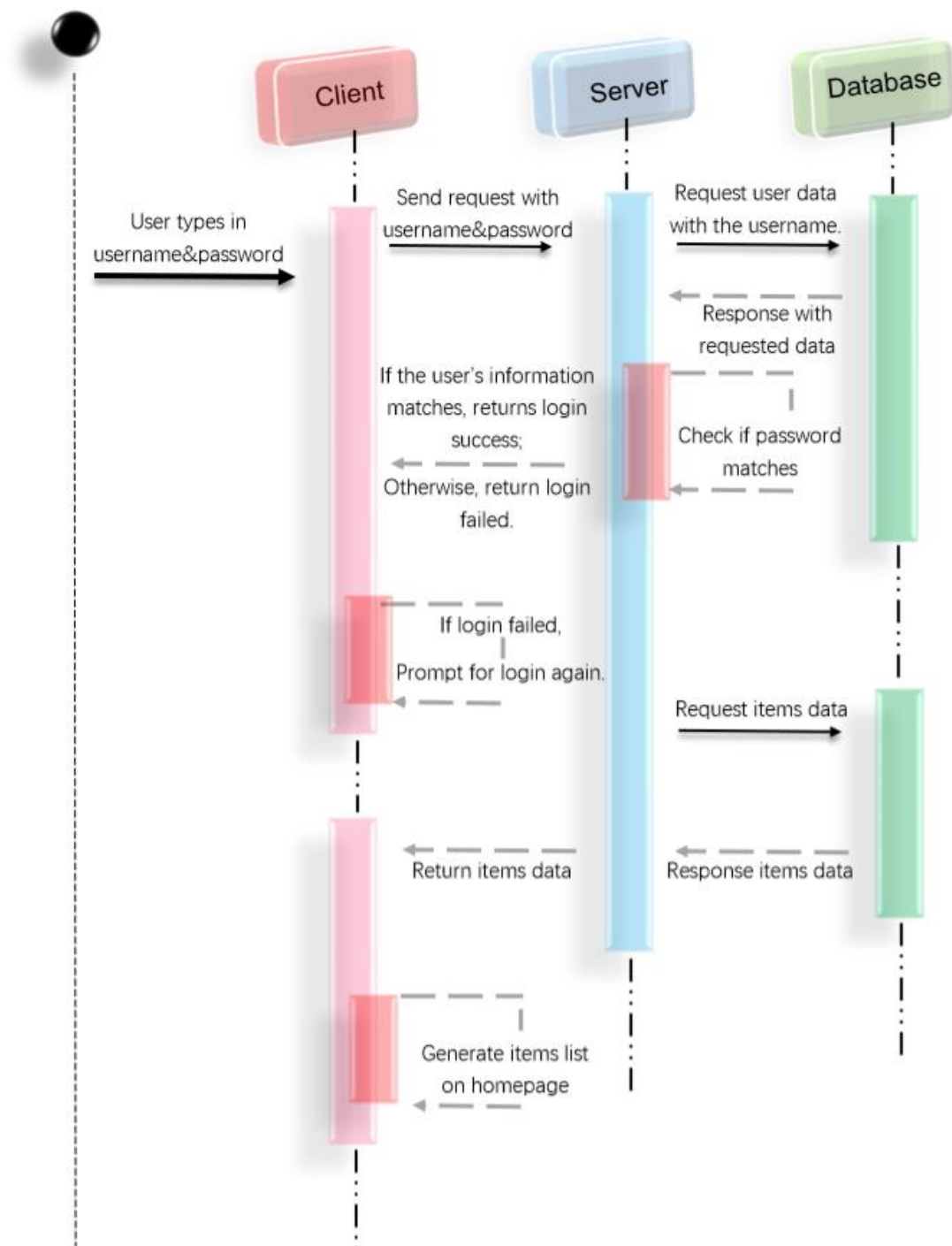
- The item object is created when a post object is created.
- Each item will be assigned a unique item ID.
- Each item will have a price for purchasing that item
- Each item will have a category to allow users to search for it
- Each item will have a description associated with it (may be included in Post class instead)

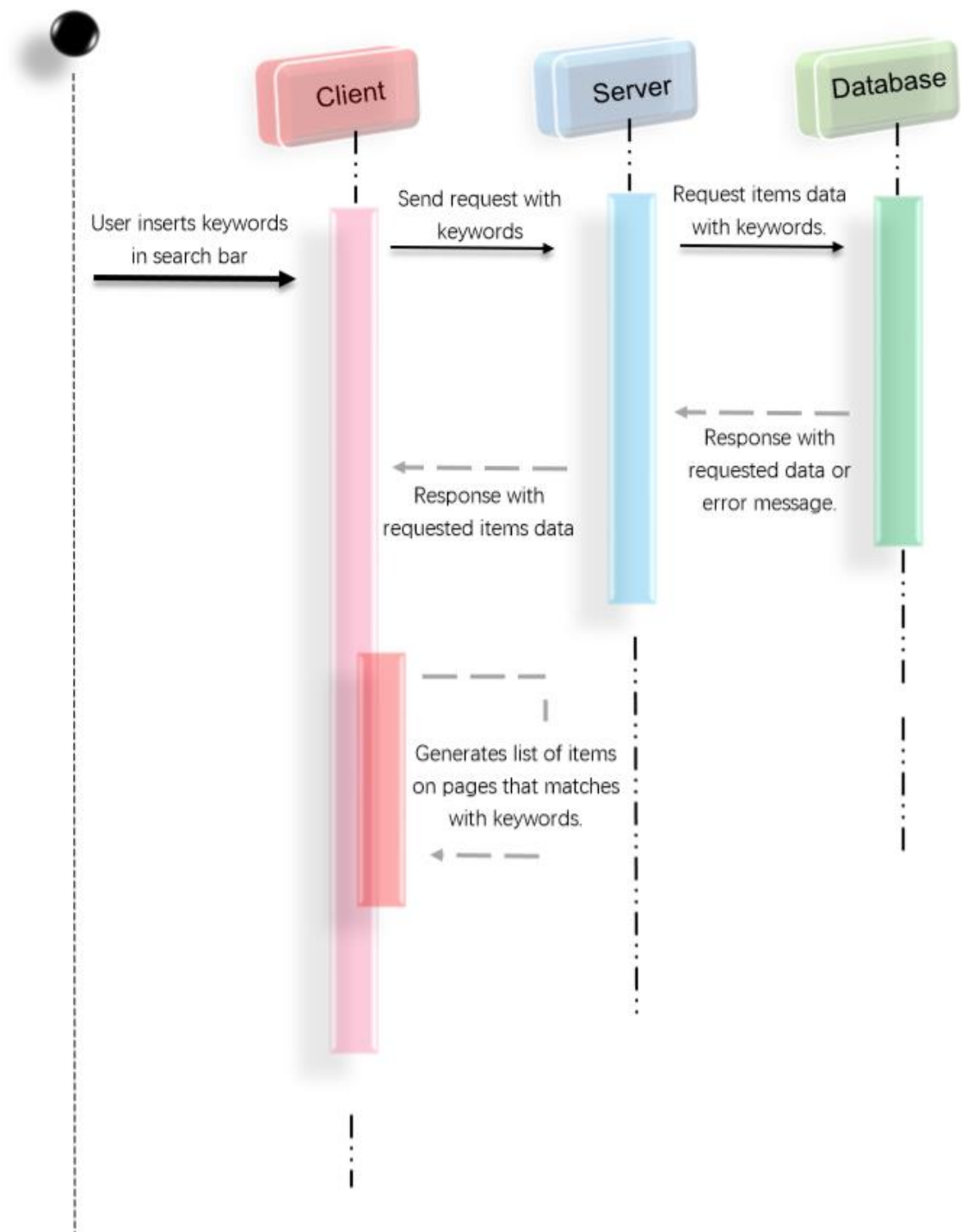
Order:

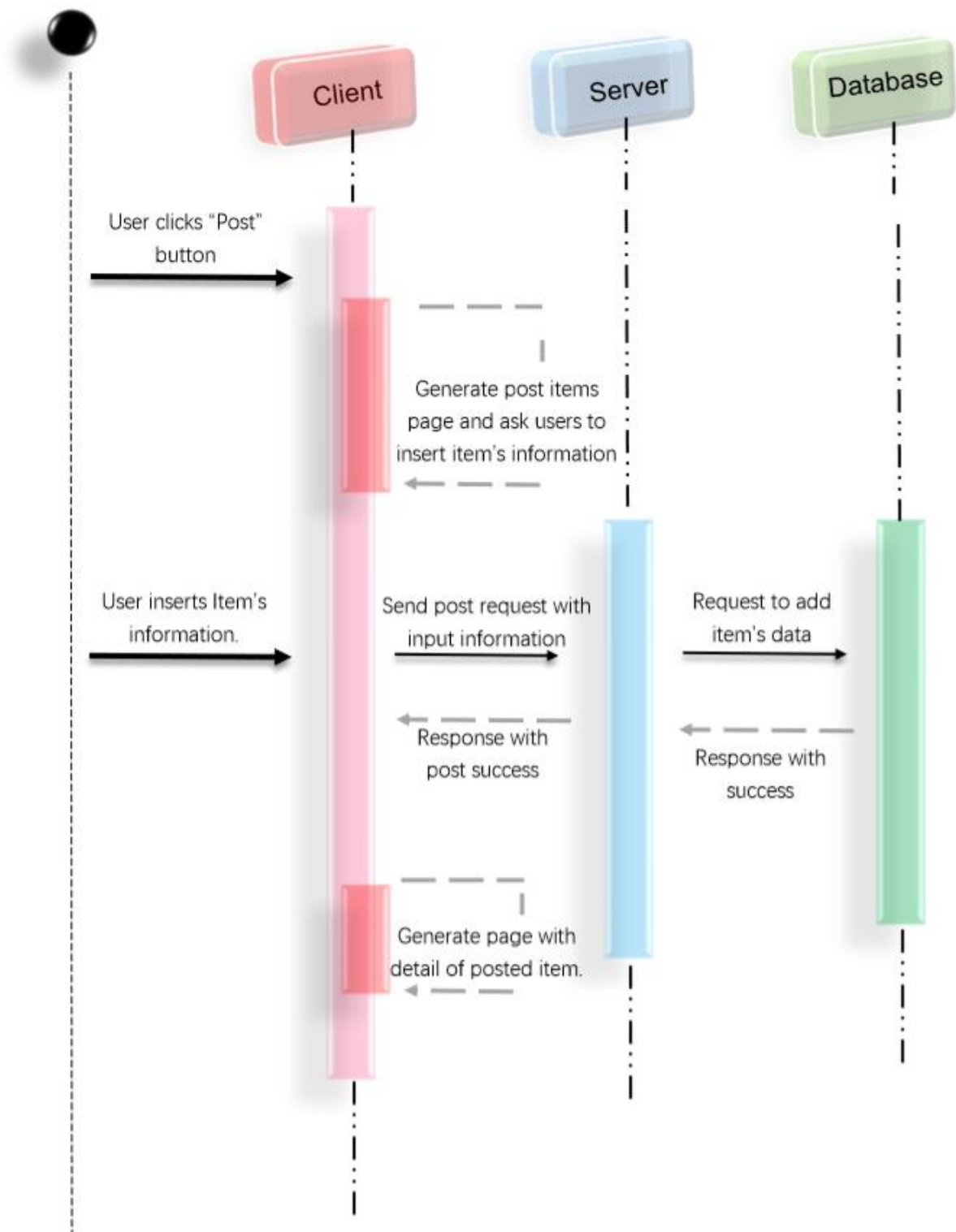
- The order object is created when a user makes a purchase.
- Each order will be assigned a unique order ID.
- Each order will have a pair of User IDs for the buyer and seller
- Each order will contain a post ID linked to the post that the buyer is purchasing.

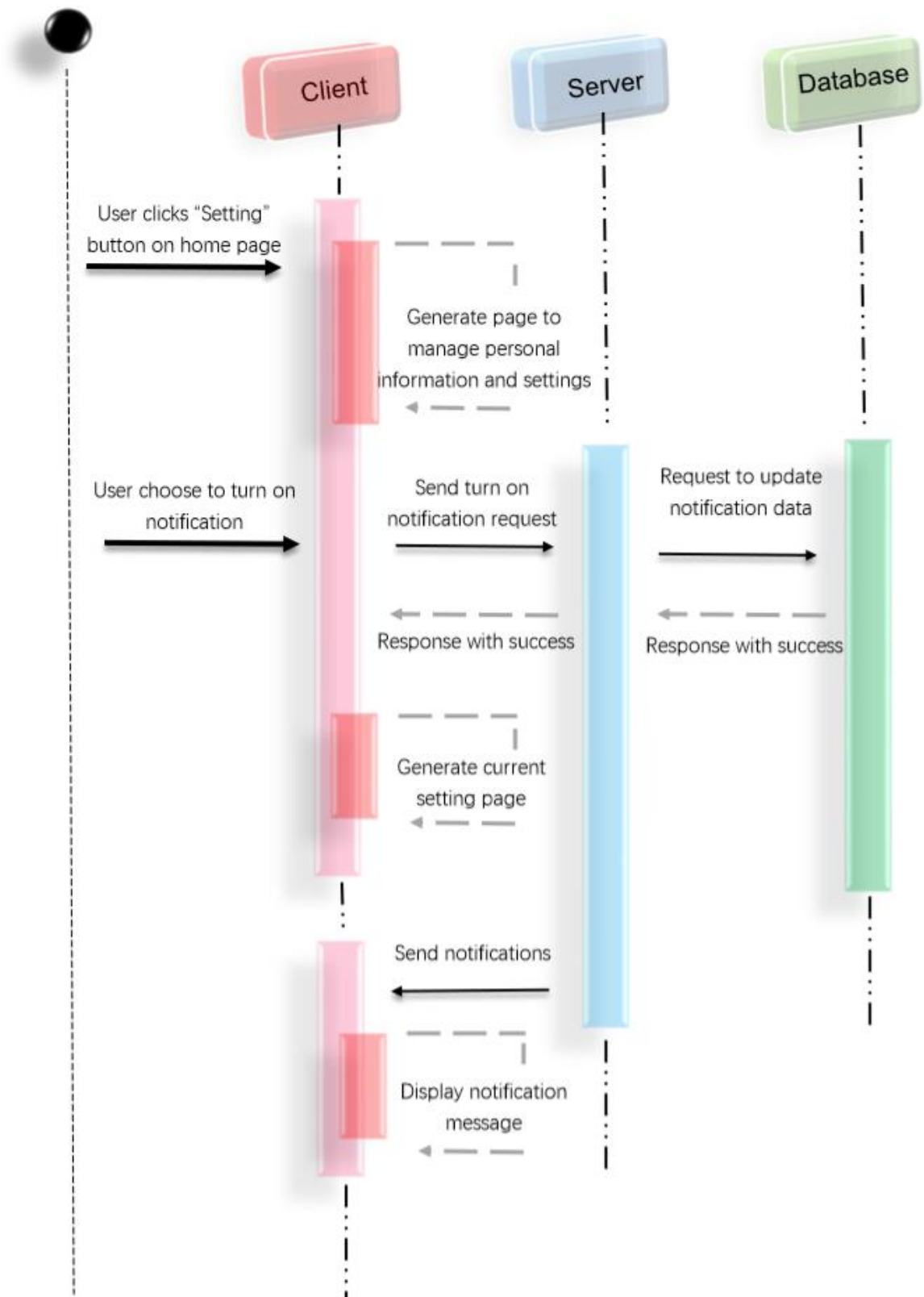
4.3 Sequence Diagram

The following diagrams show the sequence of the major events that included in our application, for example, user login, searching for items you want to buy, posting items you want to sell, making comments on an item you bought, etc. These sequence diagram show how messages exchange in a client-server model.

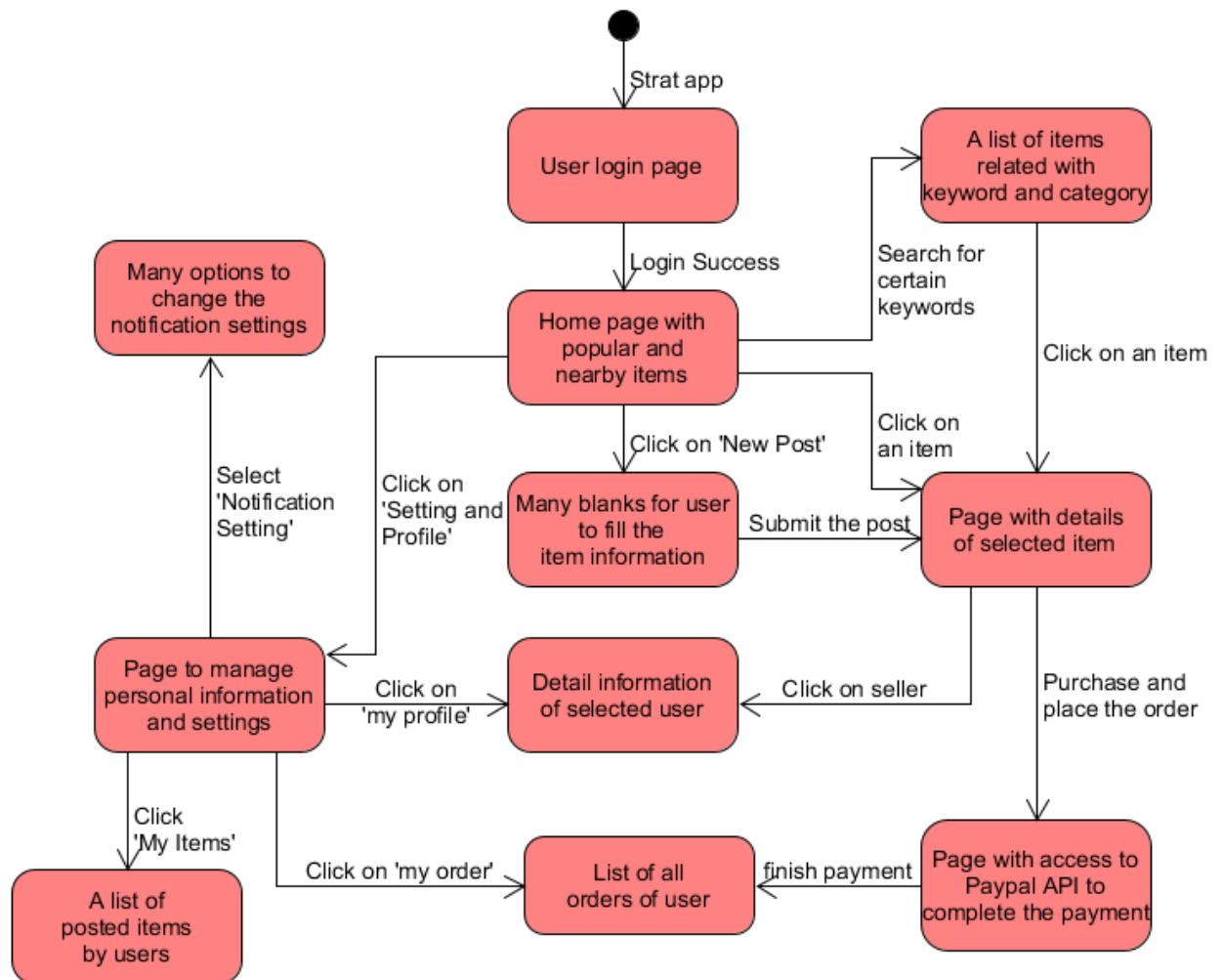
Sequence of Events When Users login.

Sequence of Events When Users Searching Items.

Sequence of Event When Users Posting items.

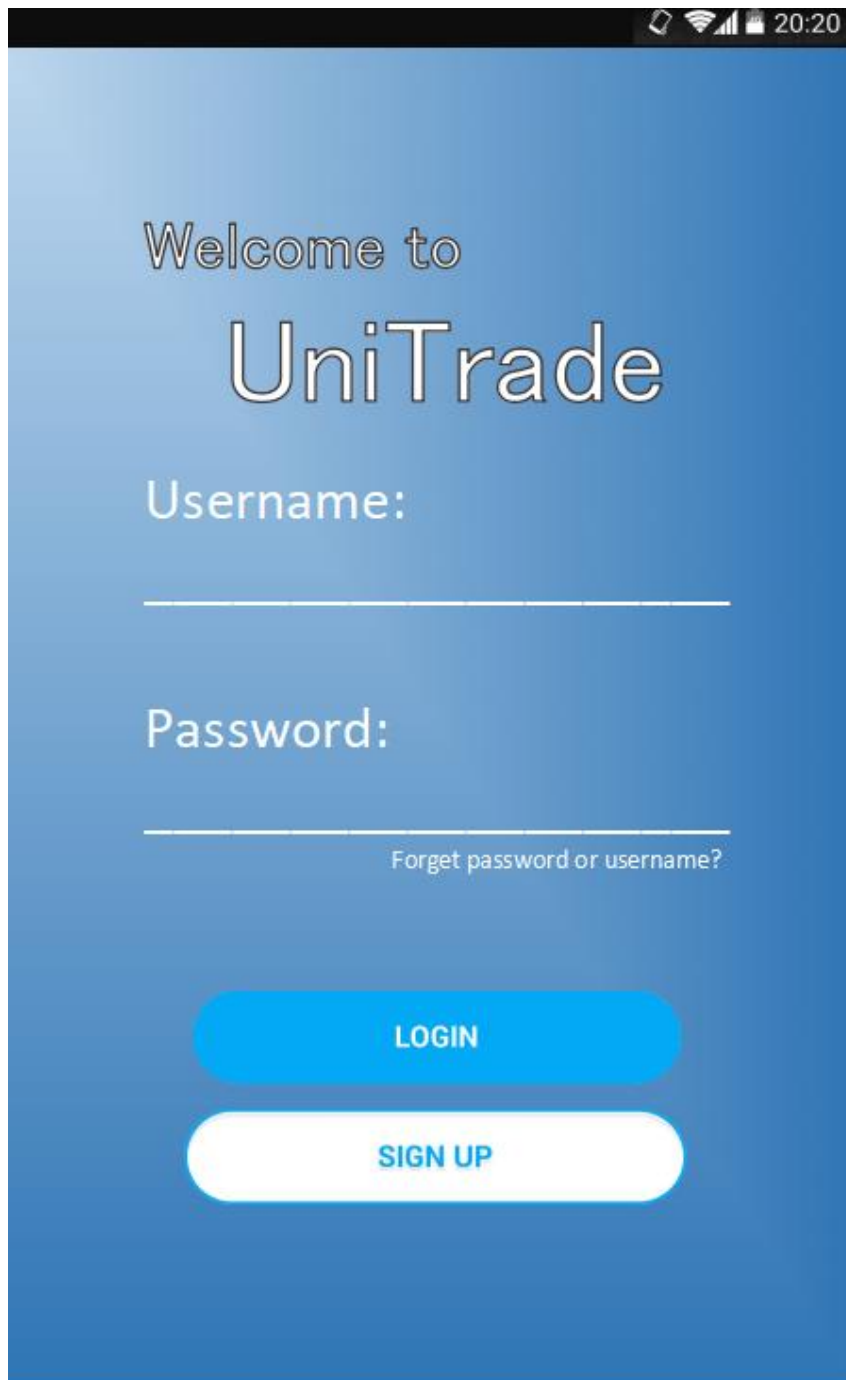
Sequence of Event When Users Turning on the Notification

4.4 Activity Map



4.5 UI mockup

① Login

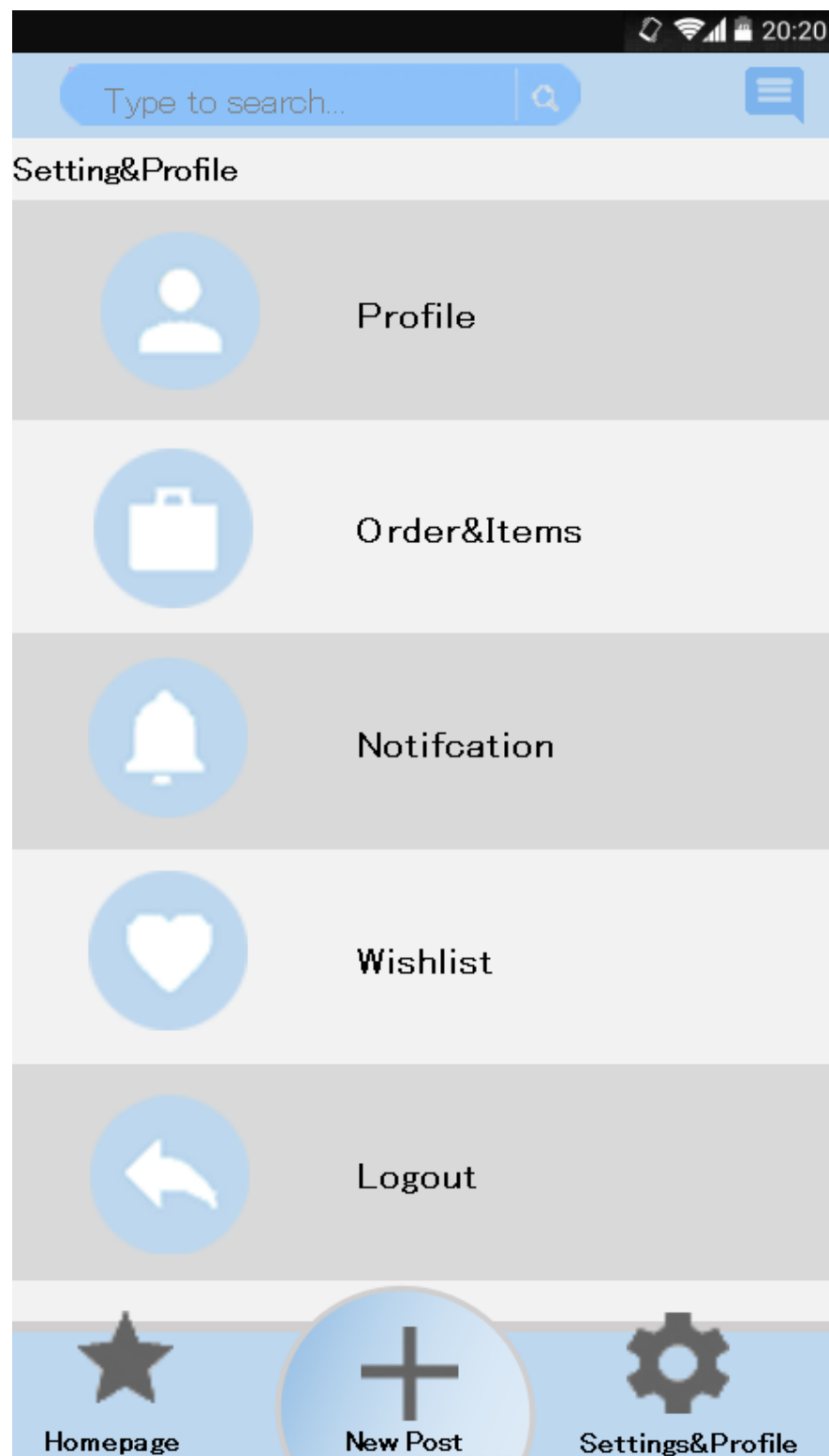


A mobile app login screen mockup for 'UniTrade'. The background is a solid blue color. At the top, there is a black status bar with white icons for signal, Wi-Fi, and battery, and the time '20:20'. The main text 'Welcome to' is in a small, white, sans-serif font, followed by 'UniTrade' in a large, white, outlined serif font. Below this, the label 'Username:' is in a white, sans-serif font, followed by a white horizontal input field. Below the input field, the label 'Password:' is in a white, sans-serif font, followed by another white horizontal input field. Below the password input field, there is a link 'Forget password or username?' in a small, white, sans-serif font. At the bottom, there are two rounded rectangular buttons: a blue one with the text 'LOGIN' in white, and a white one with the text 'SIGN UP' in blue.

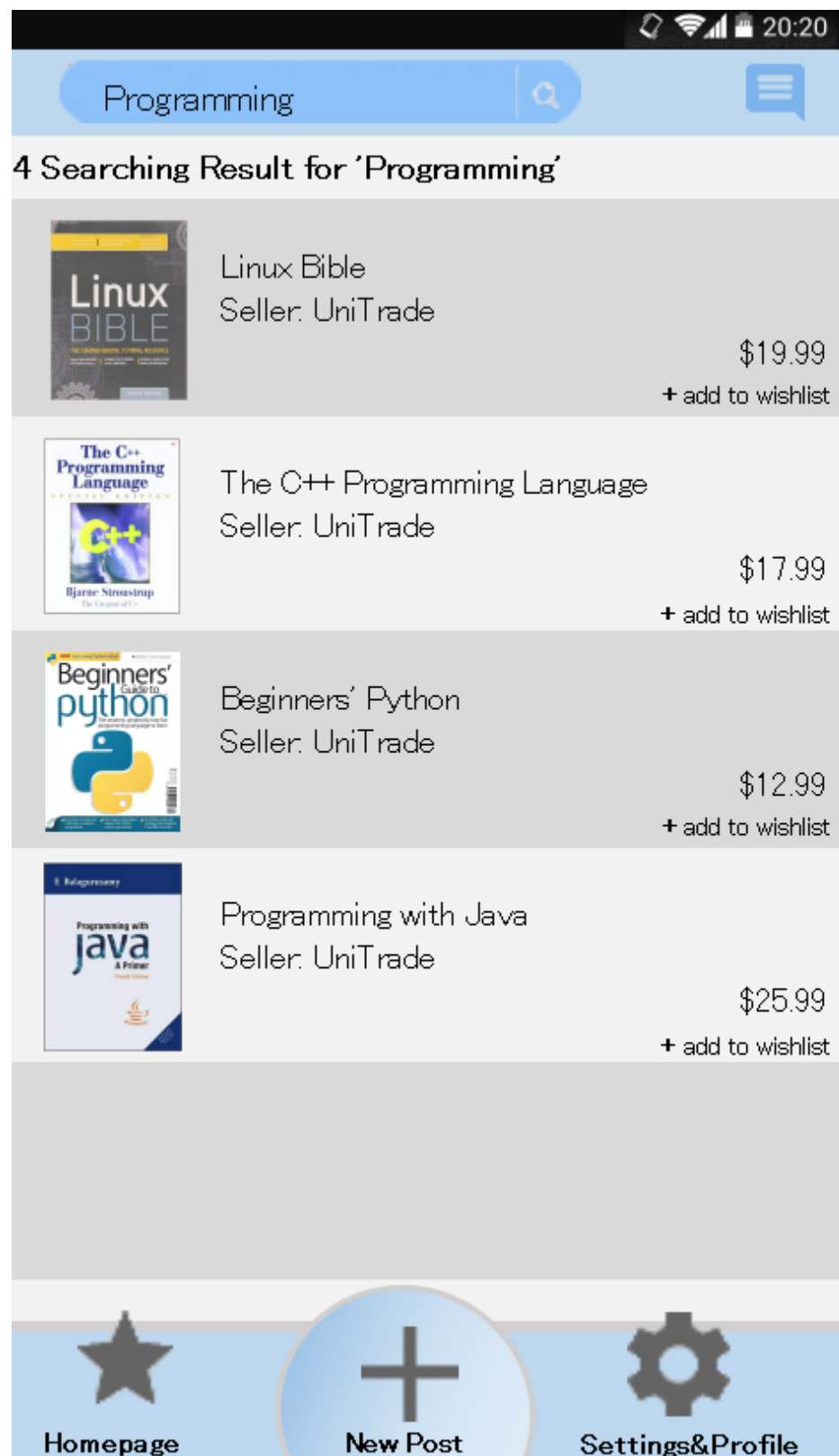
② Home Page



③Setting&Profile



④ Search Result



⑤ Profile



⑥Item Detail



⑦ New Post

20:20

Type to search...

New Post



[upload image](#)

Item Name:

Price:

Description


Homepage

Submit Post


Settings&Profile