Software Requirements Specification

for

FINDR

Version 1.6 approved

Prepared by

LEE JUIN,

SOH SHING HAO,

OI YEEK SHENG,

JERICK LIM KAI ZHENG,

LOHIA VARDHAN

Team FindR, Nanyang Technological University

2022-10-27

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
LEE JUIN	2022-10-04	Initial write-up.	1.0
LEE JUIN	2022-10-13	Included section *1.3* and *2.4*.	1.1
LEE JUIN	2022-10-18	Included section *4* and Appendix B. Updated Appendix A.	1.2
LEE JUIN	2022-10-24	Initial write-up on section *2.5, 2.6*, *2.7* and *5*.	1.3
LEE JUIN	2022-10-25	Included section *3*.	1.4
OI YEEK SHENG	2022-10-25	Included Supplementary Materials	1.5
LEE JUIN	2022-10-26	Modified System Architecture Diagram	1.5.1
LEE JUIN	2022-10-27	Included section *5*, finalizing documentation.	1.6

1. Introduction

1.1 Purpose

This Software Requirement Specification (SRS) document is intended for the *FindR* web application, build version 1.0. The purpose of this SRS document is to describe the requirements specifications for the *FindR* web application to facilitate the development and production process for all stakeholders. All aspects of the web application, which includes but not limited to, the system features, the limitations, the non-functional and interface requirements, are documented within this SRS document.

1.2 Document Conventions

This section describes the conventional standards used throughout this document. It is imperative that all readers pay attention to the standards listed in this section.

Font:	Times New Roman
Heading:	Bold, Size 18
Sub-heading:	Bold, Size 14
Content:	Italic, Size 12
Technical Standards:	IEEE 830-1998

Refer *Appendix A: Data Dictionary* for the definitions of special terms used throughout this documentation.

1.3 Intended Audience and Reading Suggestions

This document is intended for all stakeholders, which include the users of *FindR* web application, the *FindR* development team, the *FindR* testing team, the project managers and the *FindR* marketing team.

This document begins by stating the purpose of the web application and several conventions used throughout the document. Next, a high-level overview of the application functionalities is introduced, followed by several design constraints and assumptions of the application. Then, the interface requirements of the application are stated. Finally, the document includes a detailed write-up of the system features and non-functional requirements of the application.

All stakeholders are advised to begin by reading section *1.1 Purpose*, *1.2 Document Conventions* and *Appendix A: Data Dictionary* to be familiarized with the purpose of the web application, as well as the documentation standards and technical terms definition used throughout this document.

The *FindR* development team is strongly encouraged to proceed with section *2. Overall Description* to have a high-level understanding of the application functionalities, design, and constraints. Then, section *4. System Features* follows, where the developers will gain a low-level understanding of each system features to be included in the application. Finally, the developers should read section *3. External Interfaces Requirements* and *5. Other Nonfunctional Requirements* to understand the requirements specified for the application to function as desired.

On the other hand, the users of *FindR* web application, the *FindR* testing team, the project managers and the *FindR* marketing team are encouraged to proceed reading this document in sequential order.

1.4 Product Scope

The e-commerce industry experienced a surging growth amidst the COVID-19 pandemic. As the world slowly transitions into the post-pandemic era, online shopping has slowly taken root and become a part of the norm within our society.

When shopping online, customers often spend a long time trying to find the best possible deal from various e-commerce platforms. With *FindR*, we streamlined the process of cross comparison, allowing customers to easily search, compare and purchase listings for an item sold in multiple platforms.

Suppose a user is interested in purchasing the latest iPhone series. They can quickly obtain all the listings on different platforms such as Lazada Singapore and Shopee Singapore with just a single search. The user no longer needs to navigate to different platforms and check the pricing.

FindR also recommends listings of items which may interest the user, based on the user's search history. Furthermore, a wish list feature is also provided whereby a user may add listings of items they are interested in. Friends of the user may then purchase the listings on the wish list as gift for the user.

Overall, *FindR* provides a much better overall online shopping experience to our users by automating and streamlining the process of finding the best deals out there.

FindR strives to achieve the stakeholders' unified vision of revolutionary online shopping experiences, while adhering to all the stakeholders' policies as documented under section *2.5.2 Design Standards.*

1.5 References

i. *IEEE 830-1998 - IEEE Recommended Practice for Software Requirements Specifications*. IEEE Standards Association. (n.d.). Retrieved October 25, 2022, from https://standards.ieee.org/ieee/830/1222/

ii. *Django Documentation*. Django. (n.d.). Retrieved October 25, 2022, from https://docs.djangoproject.com/en/4.1/

iii. *ReactJS Documentation*. React. (n.d.). Retrieved October 25, 2022, from https://reactjs.org/docs/getting-started.html

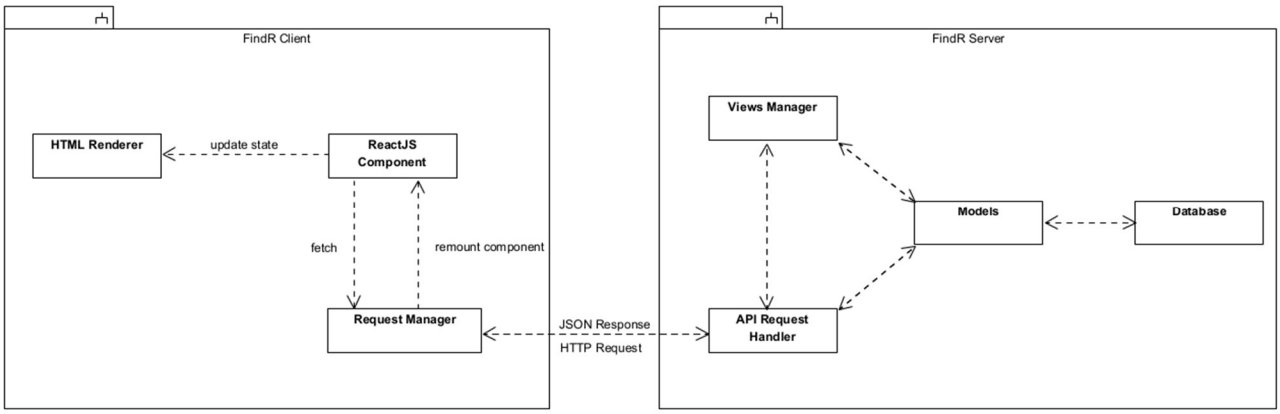
iv. Author Douglas K Barry Principal Barry & Associates, & Barry, D. K. (n.d.). *Representational state transfer (rest)*. Service Architecture. Retrieved October 25, 2022, from https://www.service-architecture.com/articles/web-services/representational-state-transfer-rest.html

v. *PostgreSQL 12.1 documentation*. PostgreSQL Documentation. (n.d.). Retrieved October 25, 2022, from https://www.postgresql.org/docs/12/index.html

2. Overall Description

2.1 Product Perspective

The *FindR* web application is a new, standalone, and self-contained web application. An overall system diagram depicting the operation of *FindR* web application is as follows:



2.2 Product Functions

The *FindR* web application's system features can be broken down into three main sub-categories — — Accounts, Friends and Main. This section provides a high-level view of the system features provided by the web application. Specifics regarding each feature, such as the activity flows, conditions, assumptions, and functional requirements achieved can be found under section *4. System Features*.

2.2.1 Accounts

i. Account Registration
ii. Account Login
iii. Retrieve Lost Account Access

2.2.2 Friends

i. Manage Friend List
ii. Send Friend Requests

iii. Accept Friend Requests
iv. Reject Friend Requests
v. Send Birthday Notifications

2.2.3 Main

i. Search for Item Listing
ii. Recommend Item
iii. Filter Search Results
iv. Manage Wish List
v. Add Wish List Item
vi. Remove Wish List Item

2.3 User Classes and Characteristics

The *FindR* web application anticipates its demographic audience to be the following, ranked by importance:

2.3.1 E-commerce Platform Buyers

Attributes	**Description**
Frequency of use	High
Subset of functions used	All
Technical expertise	Low
Characteristics	The *FindR* marketing team predicts that user of this demographic will use the web application to make better purchase decisions by comparing prices of different listings of the same product from different platforms.

2.3.2 E-commerce Platform Sellers

Attributes	**Description**
Frequency of use	High
Subset of functions used	i. Search for Item Listing ii. Recommend Item iii. Filter Search Results
Technical expertise	Low
Characteristics	The *FindR* marketing team predicts that user of this demographic will use the web application to make better business decisions, such as gauging competitors' price listing on different platforms.

2.4 Operating Environment

This section breaks down the operating environment into two sub-categories —— production environment and development environment.

All stakeholders except the *FindR* development team and the *FindR* testing team are not required to be familiarized with the specifics of the development environment. On the other hand, the *FindR* development team and the *FindR* testing team must be familiarized with both environments.

2.4.1 Production Environment of *FindR*

This sub-section describes the setting of which the web application is put into operation.

Setting	**Description**
HTML Support	The web application requires an internet browser which supports at least HTML5 or above.

	Note that HTML5 is the latest standard of HTML at the time of writing this documentation.
CSS Support	The web application requires an internet browser which supports at least CSS3 or above. *Note that CSS3 is the latest standard of CSS at the time of writing this documentation.*
JavaScript Support	The web application requires an internet browser which supports JavaScript.

2.4.2 Development Environment of *FindR*

This sub-section describes the setting of which the web application is built and tested on during development phase.

Setting	**Description**
Front-end development using ReactJS.	ReactJS is an open-source front-end JavaScript library maintained by Meta. The *FindR* development team uses ReactJS to build all user interfaces of the web applications as it features reusable components which drastically speed-up the development process. Edition: ReactJS (version 18.2.0)
Back-end development using Django.	Django is an open-source, Python-based web framework maintained by the Django Software Foundation that follows the Models – Views – Components (MVC) architectural pattern. The *FindR* development team uses

	Django to build the web server, as well as the Django REST framework to build the APIs of the web application. Edition: Django (version 4.1.1)
Database using PostgreSQL	PostgreSQL is a relational database that is SQL-compliant. The *FindR* development team deploys the web application's database on Microsoft Azure's PostgreSQL flexible server. Edition: PostgreSQL (version 12.11)
Application Hosting using Microsoft Azure	Microsoft Azure is a cloud-service provided and maintained by Microsoft Corporation. The *FindR* development team chooses Microsoft Azure over other cloud platforms due to its integration with Visual Studio. This significantly speeds up the web application's deployment process.

2.5 Design and Implementation Constraints

This section covers all constraints which have limited, or will be limiting, options available to the *FindR* development team, both during development stage and post-production maintenance stage. Additionally, this section also documents all design standards of the web application.

2.5.1 Limitations

This sub-section covers all limitations which may or may not affect certain functionalities of the web application.

Limitations	**Details**
Keyword Extractor API Limitations	The *Keyword Extractor API* utilized by the *FindR* web application adopts a freemium business model which allows only up to 300 requests per API key. Any subsequent requests will be denied until a subscription is made.
Cloud Service Provider Limitations	The *FindR* development team implements the web application through App Service provided by Microsoft Azure. However, the App Service is only free for 12 months, before a subscription fee is charged on per-use basis. The App Service will also force the deployed web application into deep sleep mode after 20 minutes of inactivity. The *FindR* development team resorts to a scheduling policy within the Django server to routinely ping the web application to keep the web application awake.
E-commerce Platform API Limitations	The e-commerce platform APIs such as Shopee Open API and Lazada Open API are not made accessible to the *FindR* development team until a valid business registration number is provided and until the *FindR* development team has passed a series of rigorous software testing by the e-commerce platforms. The *FindR* development team resorts to web scraping to obtain data from these e-commerce platforms instead. More information regarding the methodology used during web scraping can be found in *Appendix C: Supplementary Materials*.

2.5.2 Design Standards

This sub-section covers all design standards adopted by the *FindR* web application.

Design Standards	**Details**
Programming Standards	All non-class variables must adopt the camelCase naming conventions. All class variables must adopt the Pascal Case naming conventions.
User Interface Standards	All User Interface designs must adhere to the stakeholders' pre-approved colour scheme.

2.6 User Documentation

2.6.1 *FindR* API Documentations

The *FindR* testing team is strongly advised to go through each API endpoint's documentation to be familiarized with the endpoints provided by the *FindR* web application back-end server.

- Lee, J. (n.d.). *FindR API Documentation (Accounts)*. FindR API. Retrieved October 25, 2022, from https://documenter.getpostman.com/view/24005937/2s84DmwjBR
- Lee, J. (n.d.). *FindR API Documentation (Friends)*. FindR API. Retrieved October 25, 2022, from https://documenter.getpostman.com/view/24005937/2s84Dmx3yZ
- Lee, J. (n.d.). *FindR API Documentation (Main)*. FindR API. Retrieved October 25, 2022, from https://documenter.getpostman.com/view/24005937/2s84Dmx3yb

2.7 Assumptions and Dependencies

The *FindR* development team developed the *FindR* web application with the following assumptions:

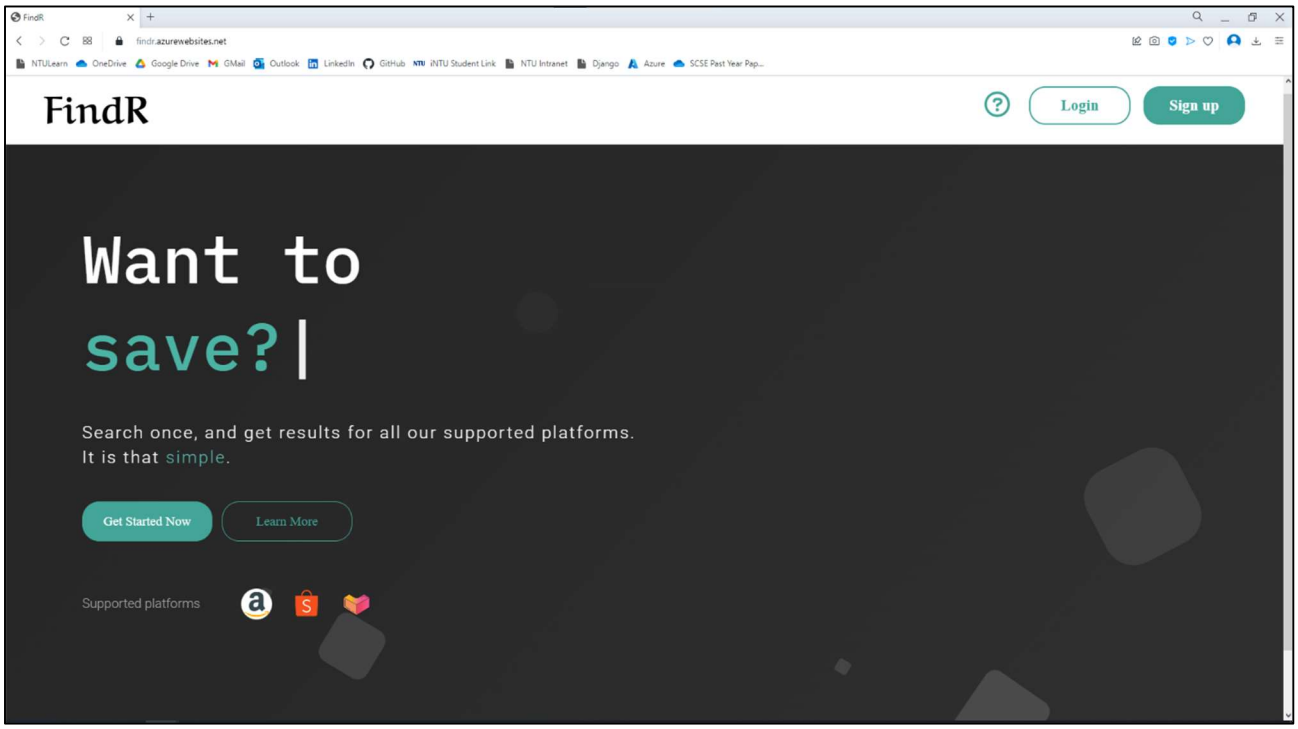
- The full-scale developed *FindR* web application will receive funding for full subscription towards all APIs required as per documented in section *2.5.1 Limitations* once the prototype application is approved by all stakeholders.
- The full-scale developed *FindR* web application will be sponsored with free access to unrestricted usage of database and cloud hosting service as per documented in section *2.5.1 Limitations* once the prototype application is approved by all stakeholders.
- The *FindR* web application is designed to be best viewed with desktop browsers.
- The *FindR* web application did not incorporate OAuth implementation. Thus, the back-end server will not logout any user unless prompted by the user. Hence, the web application is built with the assumption that users will log out of their account once they are done.

3. External Interface Requirements

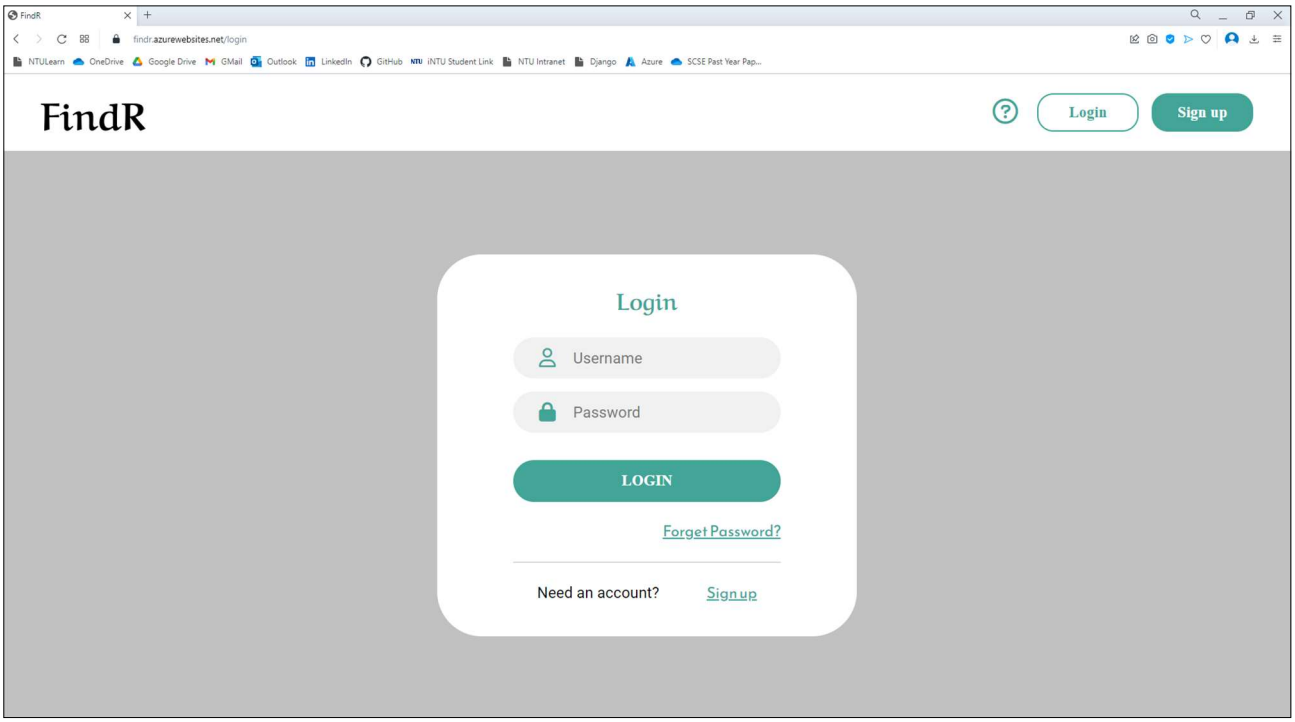
3.1 User Interfaces

The *FindR* web application is designed to be viewed with an aspect ratio which matches a desktop browser. On top of that, the desktop browser must satisfy the requirements as indicated within section *2.4.1 Production Environment of FindR*.

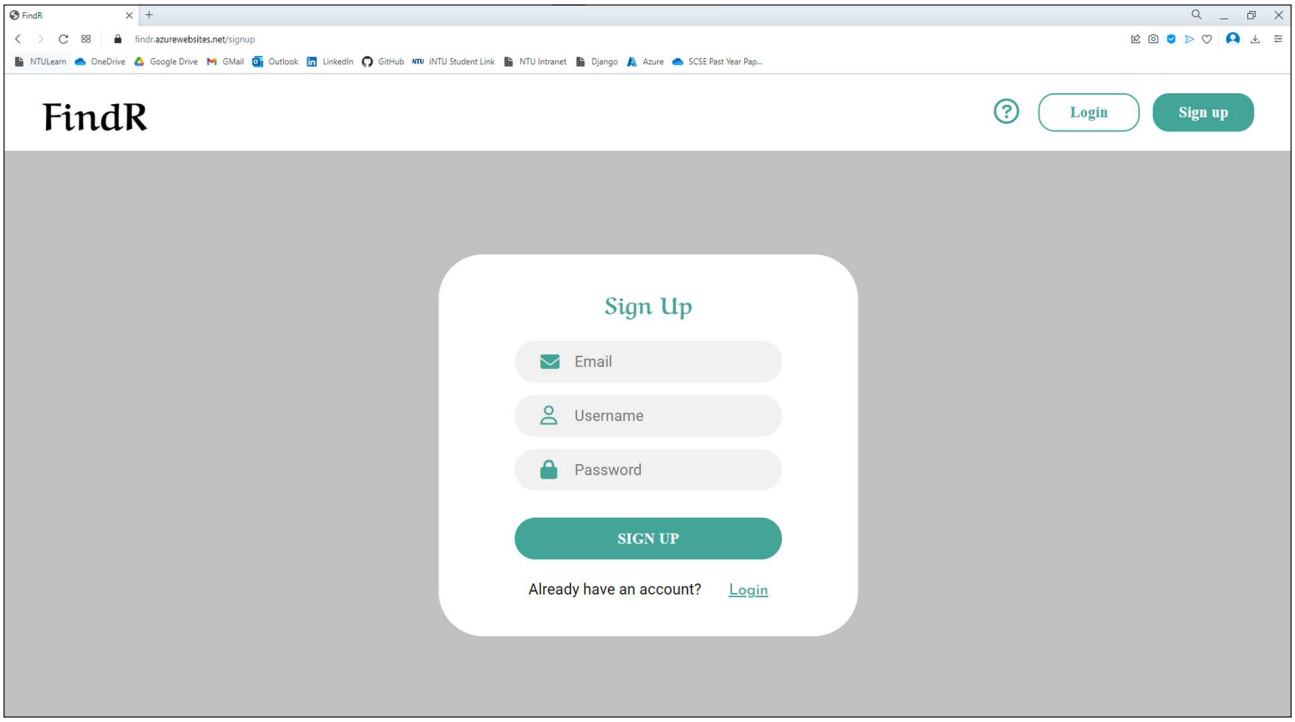
3.1.1 Landing Page



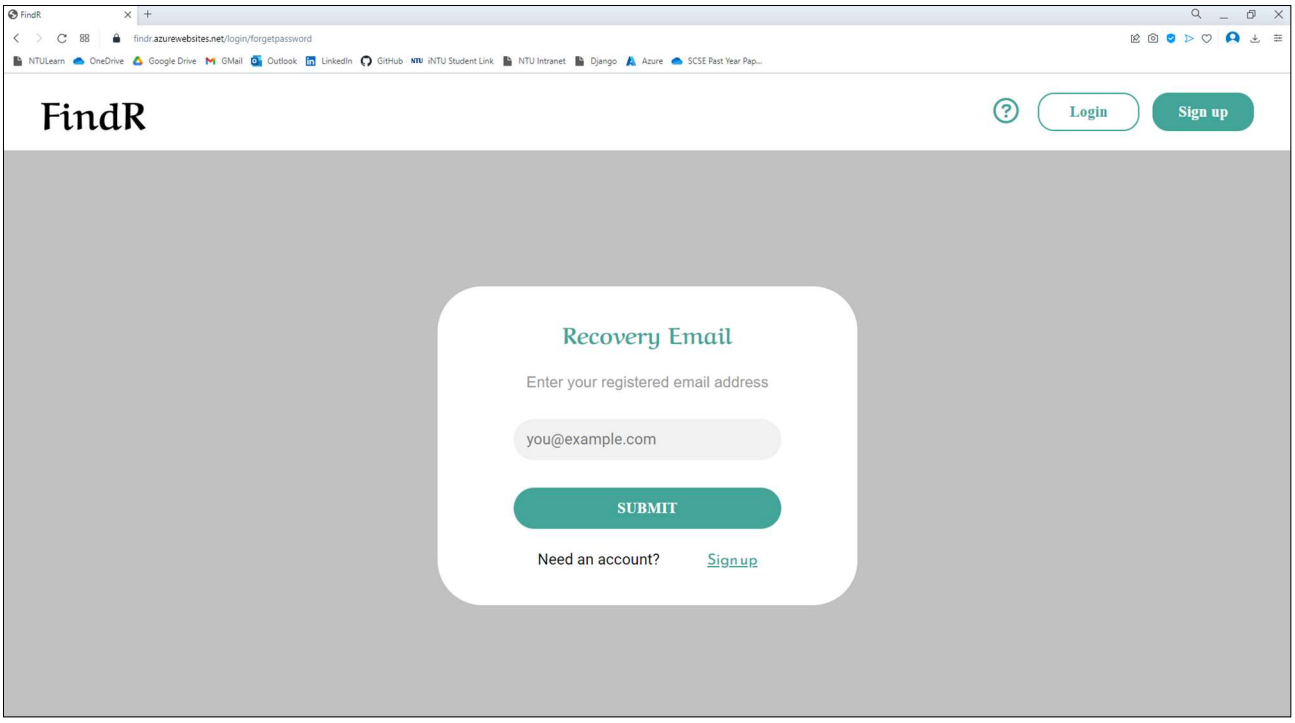
3.1.2 Login Page



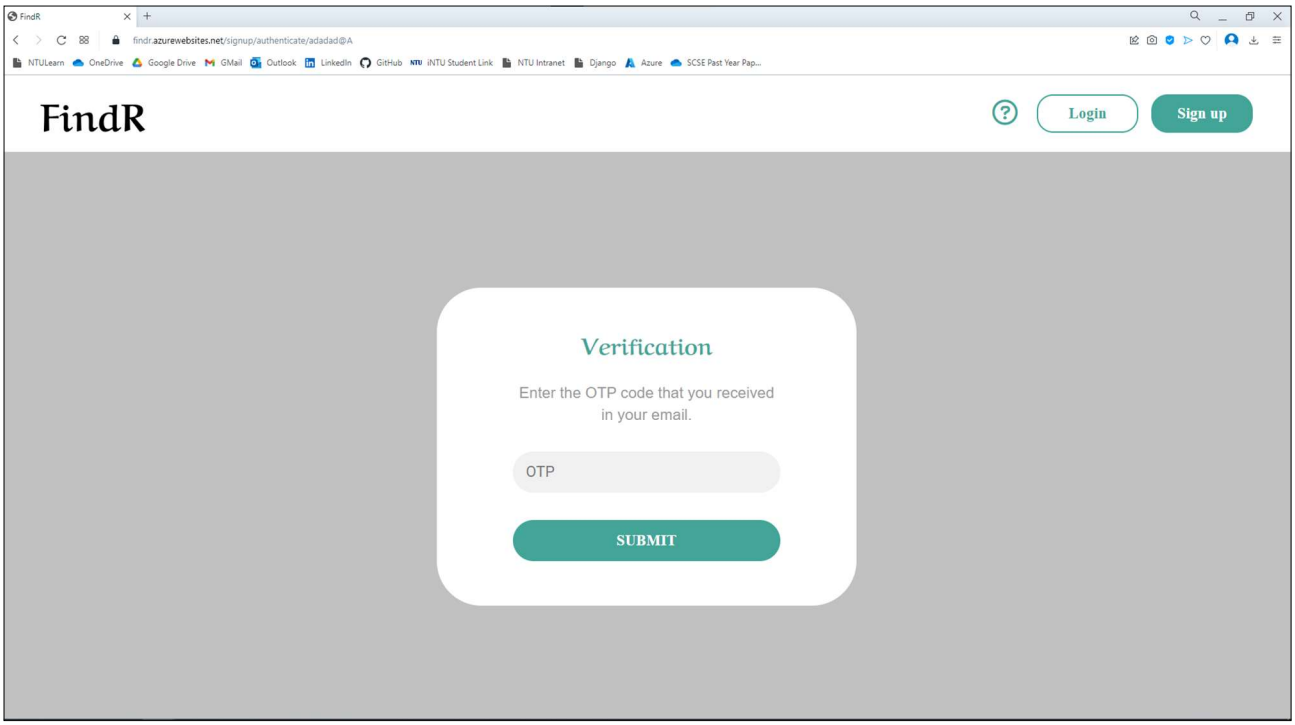
3.1.3 Signup Page



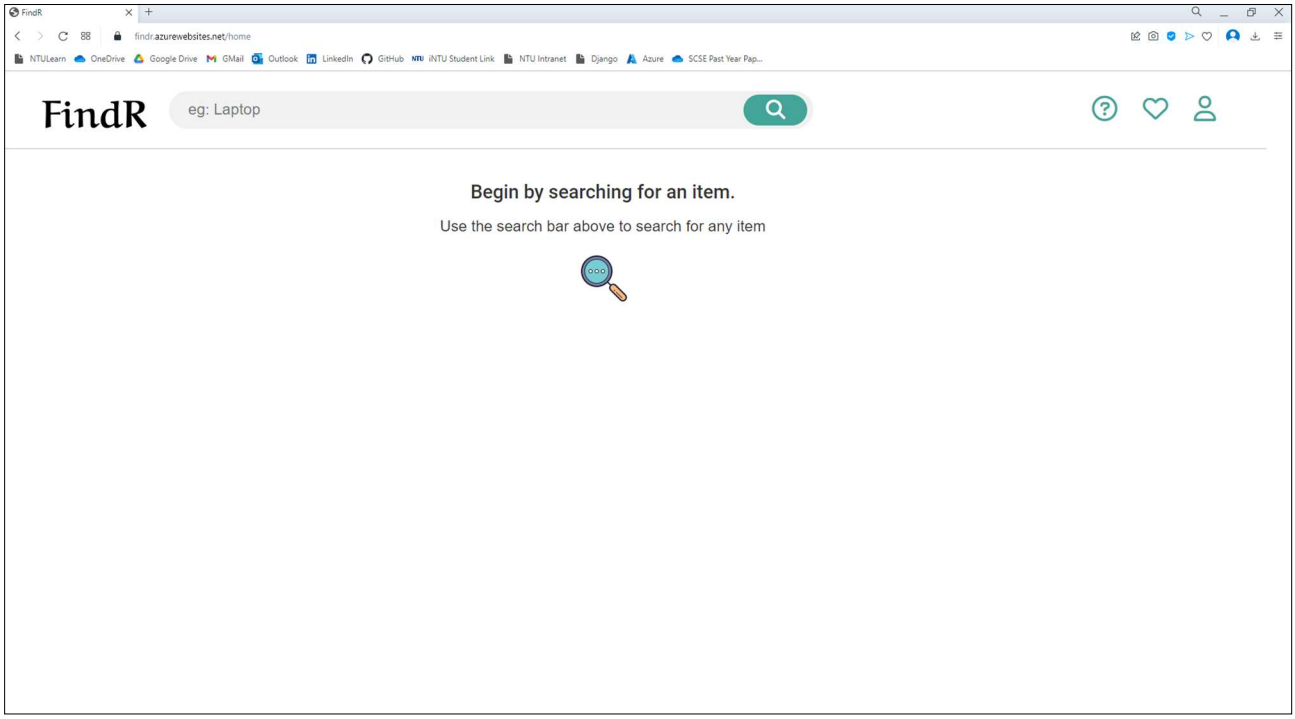
3.1.4 Forgot Password Page



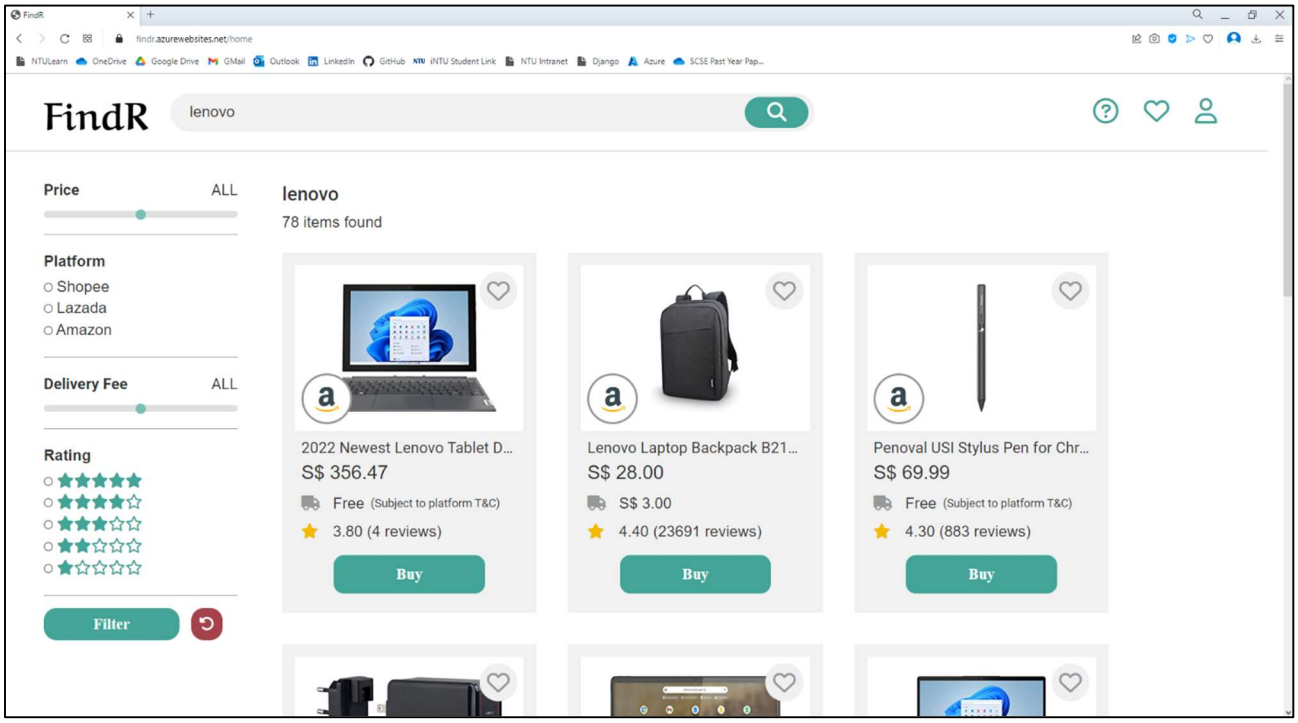
3.1.5 Email Authentication Page



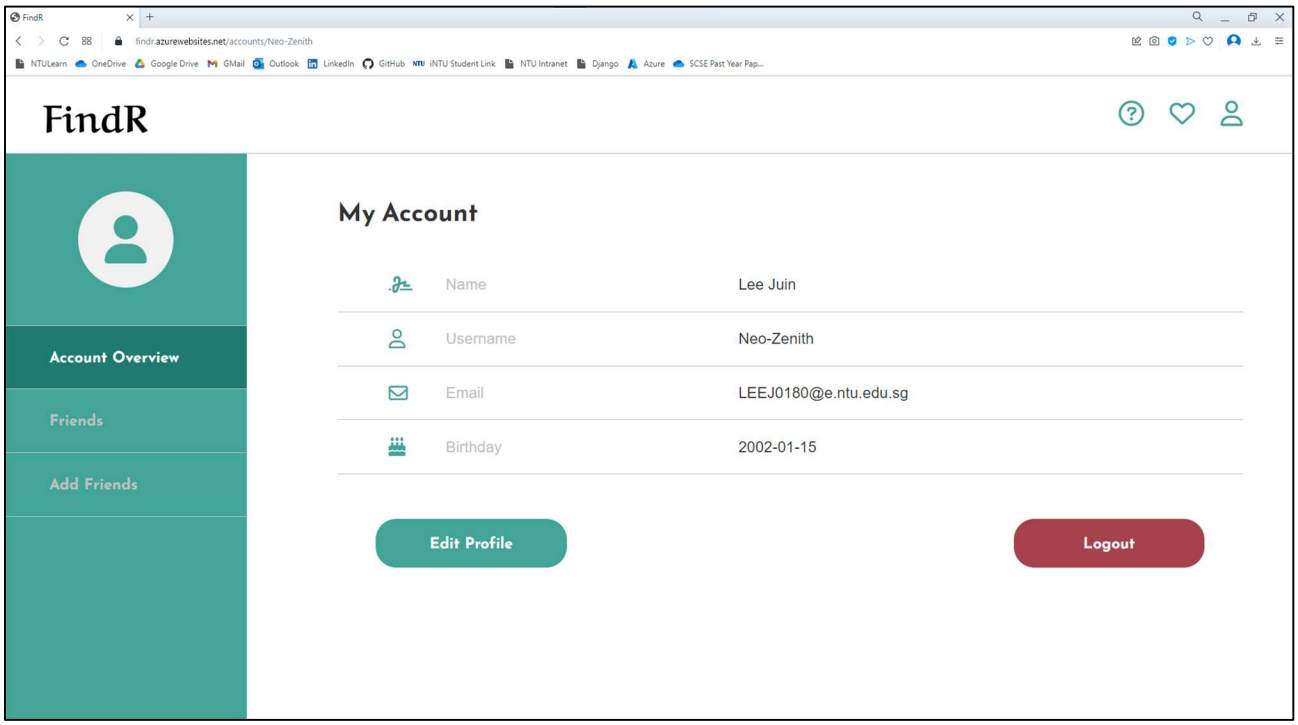
3.1.6 Home Page (items have not been searched)



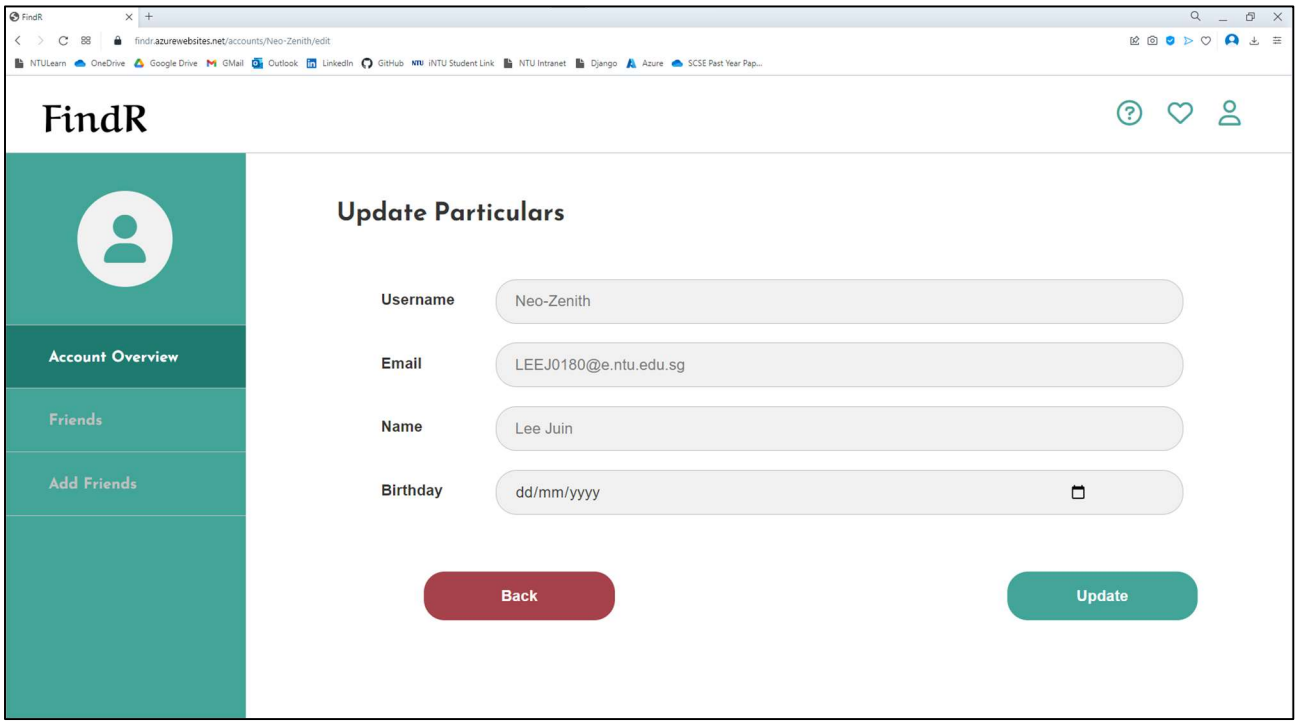
3.1.7 Home Page (items have been searched)



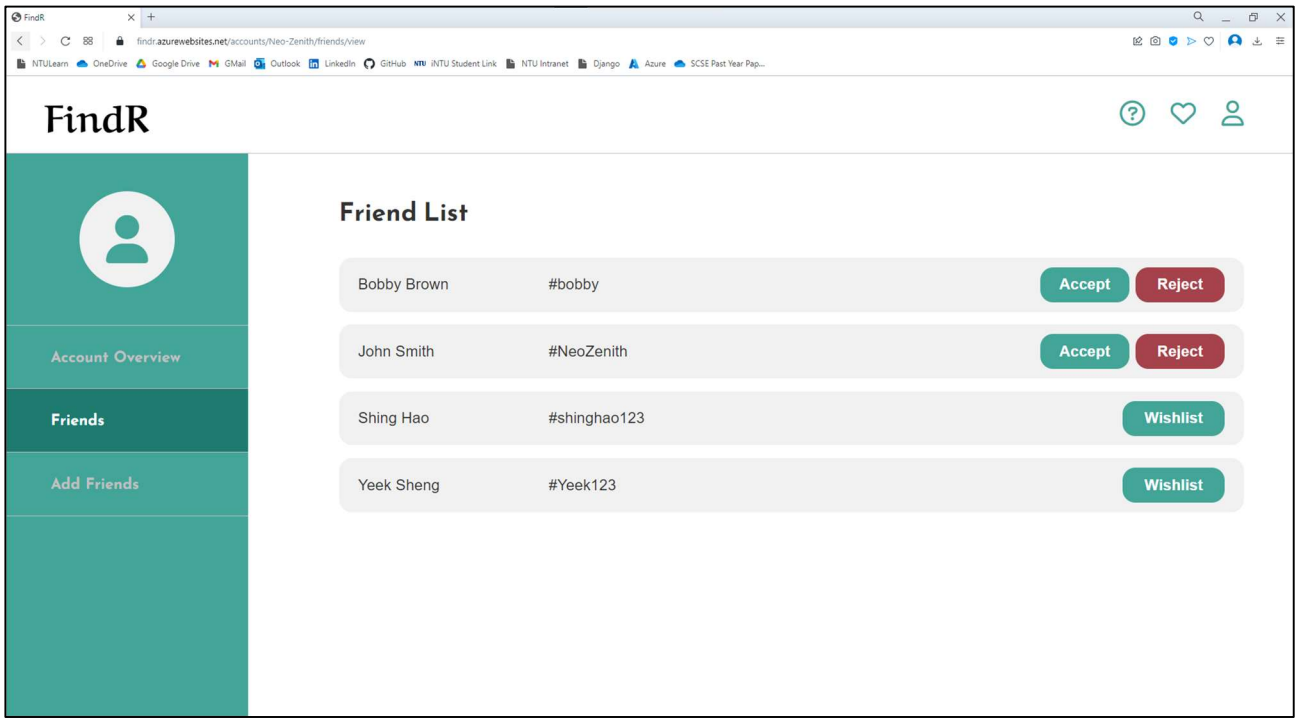
3.1.8 Accounts Details Page



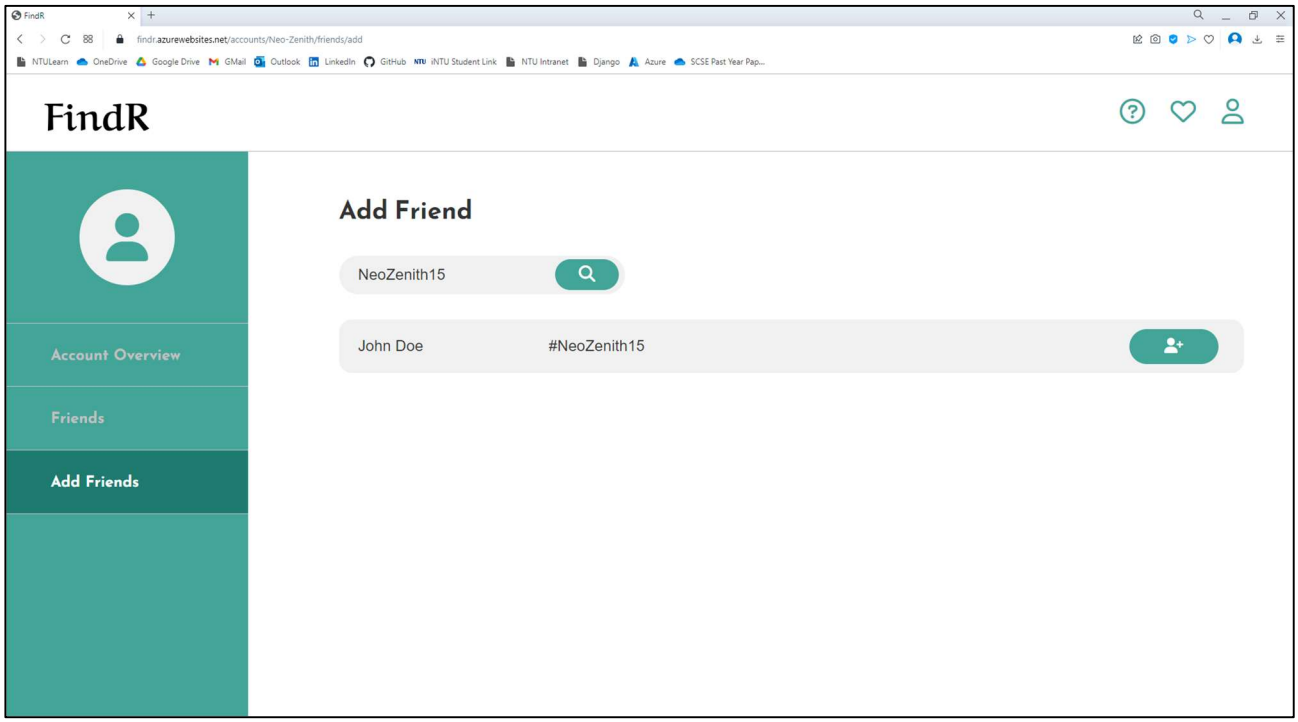
3.1.9 Edit Profile Page



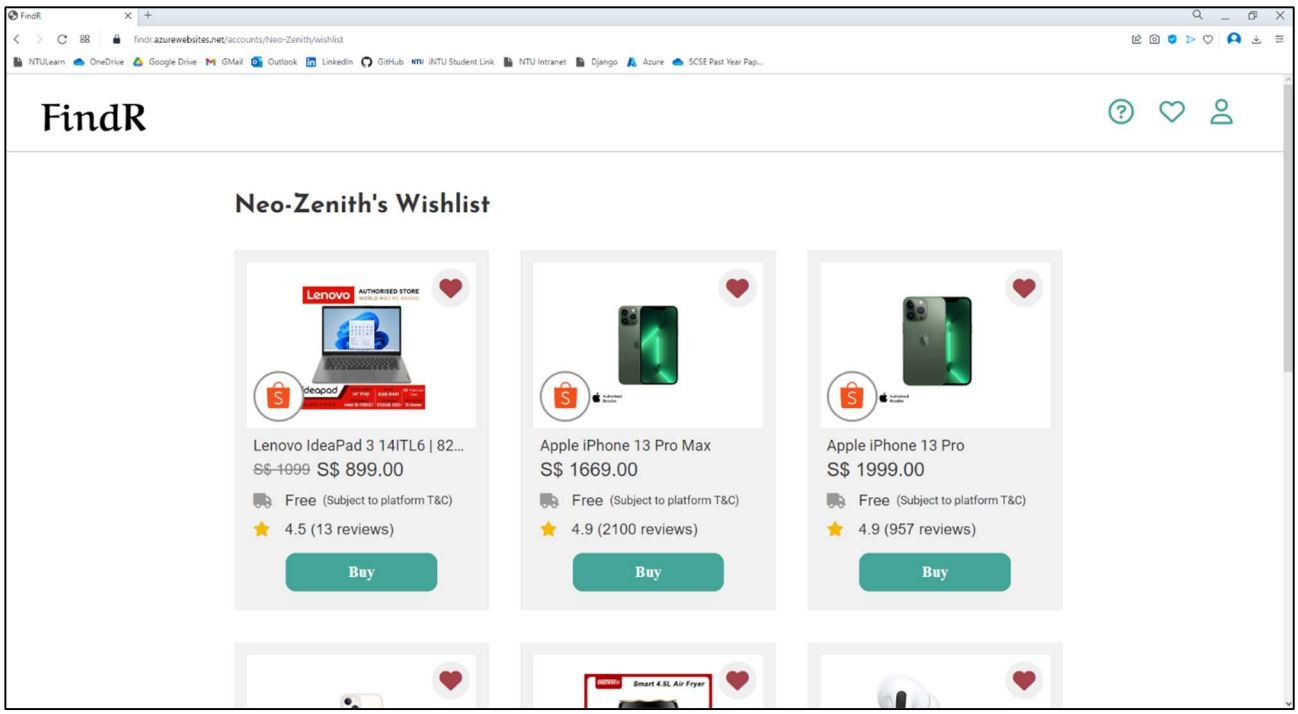
3.1.10 Friends Page



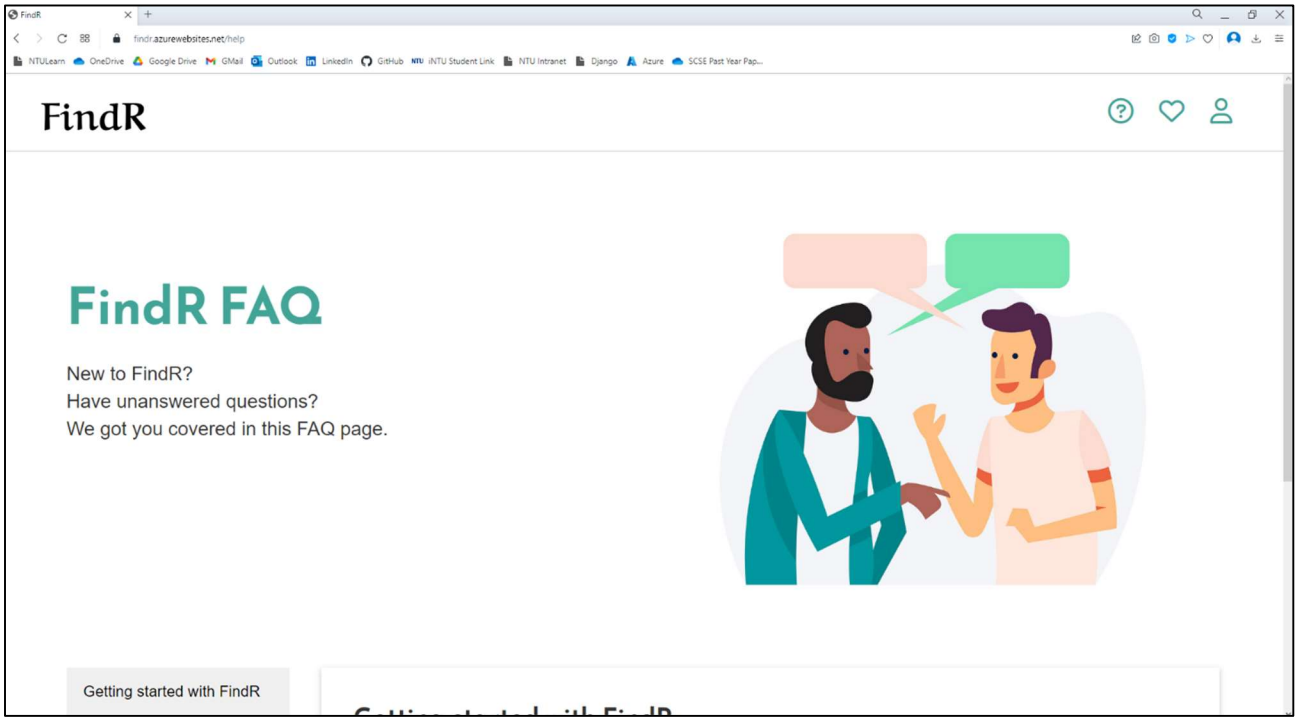
3.1.11 Add Friends Page



3.1.12 Wishlist Page



3.1.13 FAQ Page



3.2 Hardware Interfaces

This section covers all hardware interface requirements for the *FindR* web application to achieve its desired functionalities. The requirements are divided into client-side requirements and server-side requirements to facilitate easier reading for the respective stakeholders.

3.2.1 Client-side Requirements

The *FindR* web application supports all desktop computers or laptops. The device must support the usage of a desktop browser which fulfills the requirements as indicated in section *2.4.1 Production Environment of FindR.*

3.2.2 Server-side Requirements

The *FindR* back-end server must be hosted and run on a server-computer. The back-end server will perform Create, Read, Update, Delete (CRUD) operations on the back-end database. The database must be hosted and run on a server-computer.

3.3 Software Interfaces

3.3.1 Software Components and Versions

The *FindR* web application utilizes a back-end server and a database to handle all the system features implementation. The back-end server is built and implemented using the Python Django framework, version 4.1. The *FindR* development team adopts PostgreSQL, version 12.1, as the database for the web application.

3.3.2 Software Architecture

The *FindR* web application software architecture must follow the Model-View-Controller design pattern. The interface must be able to connect to a database to store persistent data in SQL format.

3.4 Communications Interfaces

The *FindR* web application communication architecture must follow a client-server model. Each communication must go through a REST-styled Application Programming Interface (API) provided by the back-end server. Each request must also be served over HypterText Transfer Protocol Secure (HTTPS).

Communication from client to server must invoke GET and POST requests. Communication from server to client must serve data standardized in JavaScript Object Notation (JSON) format.

4. System Features

4.1 Account Registration

4.1.1 Description and Priority

New user of *FindR* can register for an account. Upon registration, a record of the user's email address, username and password is stored in the database. Any subsequent updates made by the user, such as adding items to wish list,

adding friend, or updating particulars, will be using this registered account as reference.

Overall Priority	High

4.1.2 Stimulus/Response Sequences

Use Case ID:	UC-REG-001		
Use Case Name:	Register		
Created By:	Lee Juin	Updated By:	Lee Juin
Date Created:	25th August 2022	Date Updated:	17th October 2022

Actor:	App User
Description:	New App User can register for an account with an email address, username, and password. Upon registration, the App User will gain complete access to all functionalities provided by the App.
Preconditions:	1. App User must be connected to the Internet. 2. App User must not be logged in to any existing account.
Postconditions:	App User has successfully registered for an account with a unique username and password and logged in.
Priority:	High

	App User is required to have an account prior to accessing other functionalities of the App. It is imperative that this feature must be implemented first prior to other features.
Frequency of Use:	Low App User is only required to register for an account once.
Flow of Events:	1. App User clicks on "Sign Up" at the home page and is redirected to the registration page. 2. App User inputs a valid email address, a username, and a password into the submission form. 3. System verifies the username and email address are unique and the password satisfies the constraints. 4. Once verified, App User inputs a One-Time Password (OTP) that is sent to his/her email inbox by the system. 5. System verifies the OTP is valid. 6. System stores the App User's information in the database securely. 7. App User is logged into their account.
Alternative Flows:	**UC-REG-AF-01 If the username is taken by another user:** 1. System displays the message "Username has been taken. Please try again!" above the submission form. 2. System returns to Step 2 and waits for the App User inputs. **UC-REG-AF-02 If the email address has been registered:** 1. System displays the message "Email has been taken. Please try again!" above the submission form. 2. System returns to Step 2 and waits for the App User inputs.

	UC-REG-AF-03 If the password does not meet the requirements: 1. System displays the message "Password does not meet the requirements" above the submission form. 2. System returns to Step 2 and waits for the App User inputs. **UC-REG-AF-04 If App User inputs incorrect OTP:** 1. System displays the message "Incorrect OTP! Please try again!" above the submission form. 2. System returns to Step 5 waits for the App User inputs.
Exceptions:	**UC-REG-EX-01 If App User is already logged in:** 1. When App User attempts to access the Registration feature, System verifies that App User has already logged in. 2. System redirects App User to their account instead.
Includes:	NIL
Special Requirements:	1. The Registration page must contain a quick navigation to the Login page. 2. Inputs at the password field must be obscured.
Assumptions:	1. App User must be connected to the Internet throughout the registration process. 2. App User only registers for one account.
Notes and Issues:	NIL

4.1.3 Functional Requirements

1. App User must be able to register for an account with the System.
   1.1. System must provide three input fields for App User to input information.

1.1.1. One of the input fields must be username.

1.1.2. One of the input fields must be email address.

1.1.3. One of the input fields must be password.

1.2. System must ensure that App User fills in all the input fields before allowing registration.

1.3. System must verify all input information.

1.3.1. System must verify that username is not taken.

1.3.2. System must verify that email address is not taken.

1.3.3. System must verify that password meets the requirements.

1.3.3.1. Password must be at least 8 characters long.

1.3.3.2. Password must contain at least an uppercase letter.

1.3.3.3. Password must contain at least a lowercase letter.

1.3.3.4. Password must contain at least a digit.

1.3.3.5. Password must contain at least a special character.

1.3.4. System must provide error message to App User to explain why registration is unsuccessful.

1.4. System must send an 8-digit One-Time Password (OTP) to App User's email address.

1.4.1. System must provide one input field for App User to input OTP.

1.4.2. System must verify that the OTP input by App User is correct.

1.5. System must create an account for App User once verification is completed.

1.6. System must redirect App User to their account page upon successful registration.

4.2 Account Login

4.2.1 Description and Priority

Existing user of *FindR* can login to their account. App User is required to input their username and password as verification during the login process. App User must login prior to accessing all the features provided by the App.

Overall Priority	High

4.2.2 Stimulus/Response Sequences

Use Case ID:	UC-LOG-001		
Use Case Name:	Login		
Created By:	Lee Juin	Updated By:	Lee Juin
Date Created:	25th August 2022	Date Updated:	17th October 2022

Actor:	App User
Description:	App User can login to their account with the correct credentials. App User must login prior to accessing all the features provided by the App.
Preconditions:	1. App User must be connected to the Internet. 3. App User has registered for an account.

Postconditions:	The App User has successfully logged into his/her application account.
Priority:	High App User is required to login to their account prior to accessing other functionalities of the App. It is imperative that this feature must be implemented together with feature *4.1 Registration* prior to other features.
Frequency of Use:	High
Flow of Events:	1. App User clicks on "Log in" at the home page and is redirected to the login page. 2. App User inputs his/her username and password. 3. App User clicks on "LOGIN". 4. System verifies the credentials provided with the Database. 5. When the information is verified, the App User is redirected to the home page.
Alternative Flows:	**UC-LOG-AF-01: If App User inputs an incorrect username or password** 1. System displays the message "Invalid username and/or password!" above the submission form. 2. System returns to Step 2 and waits for the App User inputs.
Exceptions:	**UC-LOG-EX-01: App User forgot his/her login credentials** 1. App User clicks on "Forget Password?" on the login page. 2. App User can recover his/her account using the extended use case *UC-RLA-001*.

	UC-LOG-EX-02: App User has already logged in 1. When App User attempts to access the Login feature, System verifies that App User has already logged in. 2. System redirects App User to their account instead.
Includes:	NIL
Extends:	*UC-RLA-001*
Special Requirements:	1. The Login page must contain a quick navigation to the Registration page. 2. The Login page must contain a quick navigation to the Help page. 3. Inputs at the password field must be obscured.
Assumptions:	1. App User must be connected to the Internet when logging in. 2. App User has not logged in to their account.
Notes and Issues:	NIL

4.2.3 Functional Requirements

2. App User must be able to login to their account with the System.
   - 2.1. System must provide two input fields for App User to input information.
     - 2.1.1. One of the input fields must be username.
     - 2.1.2. One of the input fields must be password.
   - 2.2. System must ensure that App User fills in all the input fields before allowing login.
   - 2.3. System must verify all input information.

2.3.1. System must verify that username exist in the database.

2.3.2. System must verify that password matches the password of the user stored in the database.

2.3.3. System must provide error message to App User to explain why login is unsuccessful.

2.4. System must redirect App User to the home page upon successful login.

4.3 Retrieve Lost Account Access

4.3.1 Description and Priority

Existing user of *FindR* who forgot about their login credentials can reset their credentials using this functionality. App User is required to input their registered email address and enter the OTP sent to that email address. App User can proceed to reset their username and password accordingly.

Overall Priority	Low

4.3.2 Stimulus/Response Sequences

Use Case ID:	UC-RLA-001		
Use Case Name:	Retrieve Lost Account		
Created By:	Lee Juin	Updated By:	Lee Juin
Date Created:	25th August 2022	Date Updated:	17th October 2022

Actor:	App User
Description:	App User can reset their credentials using this functionality. App User is required to input their registered email address and enter the OTP sent to that email address. App User can proceed to reset their username and password accordingly.
Preconditions:	1. App User must be connected to the Internet. 2. App User has forgotten their login credentials.
Postconditions:	App User has successfully recovered access to their account by changing their credentials.
Priority:	Low
Frequency of Use:	Low
Flow of Events:	1. App User clicks on “Forgotten?” on the login page. 2. System displays the recover account page. 3. App User inputs their registered email and clicks on “Recover Account”. 4. System displays a “Security Check” submission form. 5. App User inputs the one-time password (OTP) that has been sent to their email inbox. 6. System displays a “Change Security Details” submission form. 7. App User inputs a new set of username and password and clicks on “Change”. 8. System verifies that the username is unique, and the password satisfies the given requirements before updating the App User’s information in the database securely. 9. App User is informed of the successful change in credentials and is redirected back to the login page.

Alternative Flows:	**UC-RLA-AF-01: If App User inputs an incorrect OTP** 1. System displays the message "Invalid OTP." above the submission form. 2. System returns to Step 5 and waits for the App User inputs.
Exceptions:	**UC-RLA-EX-01: If App User entered an unregistered email address** 1. When the App User clicks on "Recover Account", the system displays the message "Email not registered!" above the submission form. 2. The system returns to Step 3 and waits for the App User inputs
Includes:	NIL
Special Requirements:	1. System should include a navigation link back to the login page if App User has input a wrong email address.
Assumptions:	NIL
Notes and Issues:	NIL

4.3.3 Functional Requirements

3. App User must be able to retrieve access to lost account with the System.
   3.1. System must provide an input field for App User to input registered email address.

   3.2. System must ensure that App User fills in the input field before allowing authentication.

   3.3. System must verify that the email address exists in the database.

3.3.1. System must provide error message to App User if email address is not found.

3.4. System must provide an input field for App User to input OTP.

3.4.1. System must verify that the OTP is valid.

3.4.2. System must provide error message to App User if OTP is invalid.

3.5. System must provide two input fields for App User to update credentials.

3.5.1. One of the input fields must be username.

3.5.2. One of the input fields must be password.

3.6. System must redirect App User to the login page upon successful update of particulars.

4.4 Search for an Item Listing

4.4.1 Description and Priority

App User can search for an item listing based on keyword input using this functionality. App User inputs a keyword into the search bar and press search. System will query the database and retrieve all item listings that match with the keyword search.

Overall Priority	High

4.4.2 Stimulus/Response Sequences

Use Case ID:	UC-SEA-001
Use Case Name:	Search Item

Created By:	Jerick Lim Kai Zheng	Updated By:	Lee Juin
Date Created:	22nd August 2022	Date Updated:	17th October 2022

Actor:	App User, E-commerce Platform
Description:	App User can search for an item listing based on keyword input using this functionality. App User inputs a keyword into the search bar and press search. System will query the database and retrieve all item listings that match with the keyword search.
Preconditions:	1. App User is connected to the Internet. 2. App User has logged in to their account.
Postconditions:	The App User obtained a list of searched items based on the keywords.
Priority:	High
Frequency of Use:	High
Flow of Events:	1. App User types a keyword in the search box and clicks on the search icon. 2. System queries the database for items sold on the e-commerce platforms based on the keyword. 3. System retrieves information on the name, description, price, number of reviews, average ratings, delivery fee and platform for each item retrieved. 4. System displays the item listing arranged from item with lowest price to item with highest price.

	5. System displays more than one page if the items retrieved exceed the number of items allowed per page. 6. System allows App User to navigate through different listing pages. 7. System allows App User to filter their search using the included *UC-FIL-001* use case. 8. System recommends items to App User using the included *UC-REC-001* use case.
Alternative Flows:	**UC-SEA-AF-01: If System is unable to query any items based on the keyword** 1. System displays the message "No result found" to the App User. 2. System returns to Step 1 and wait for App User input.
Exceptions:	NIL
Includes:	1. *UC-FIL-001* 2. *UC-REC-001*
Special Requirements:	NIL
Assumptions:	NIL
Notes and Issues:	1. Due to restriction of access to APIs of famous e-commerce platforms such as Shopee and Lazada, the only alternative to obtain real-world item listing data is through web-scraping.

4.4.3 Functional Requirements

4. App User must be able to search for items based on an input keyword.
   4.1. System must provide an input field for App User to input search keyword.

4.2. System must query the database to retrieve all items which match the search keyword.

4.2.1. System must attempt to match an item's description with the search keyword.

4.2.2. System must attempt to match an item's name with the search keyword.

4.3. System must display the items queried to App User.

4.3.1. System must display information about each item to App User.

4.3.1.1. Information must contain item name.

4.3.1.2. Information must contain item price.

4.3.1.3. Information must contain item description.

4.3.1.4. Information must contain item ratings.

4.3.1.5. Information must contain platform that the item is sold on.

4.3.1.6. Information must contain the total number of reviews an item has.

4.3.2. System must display an error message to App User if there are no items which match the search keyword.

4.5 Filter Search Result

4.5.1 Description and Priority

App User can filter search results using various properties of an item listing as parameter. Once App User sets the parameters, they can filter the search result.

Overall Priority	Medium

4.5.2 Stimulus/Response Sequences

Use Case ID:	UC-FIL-001		
Use Case Name:	Filter Item		
Created By:	Lee Juin	Updated By:	Lee Juin
Date Created:	17th October 2022	Date Updated:	17th October 2022

Actor:	App User
Description:	App User can filter search results using various properties of an item listing as parameter. Once App User sets the parameters, they can filter the search result.
Preconditions:	1. App User is connected to the Internet. 2. App User has logged in to their account. 3. App User has searched for item using keyword beforehand.
Postconditions:	App User obtained a list of filtered searched items based on the parameters set.
Priority:	Medium
Frequency of Use:	Medium
Flow of Events:	1. Once App User has searched for items, System will display the queried items to App User. 2. App User can filter the searched items using price, rating, delivery fee and platform as parameters.

	3. App User adjust the price and delivery fee parameters by moving the slider. 4. App User adjust the rating and platform parameters by indicating their selection. 5. Once the parameters are set, App User presses "Filter" to filter the searched items. 6. System checks for each item and remove the item listings which do not meet App User filter requirements from display. 7. System displays the final filtered item listings to App User.
Alternative Flows:	**UC-FIL-AF-01: If App User parameters do not match any items** 1. System displays the message "No result found" to the App User. 2. System returns to Step 2 and wait for App User input.
Exceptions:	NIL
Includes:	NIL
Special Requirements:	NIL
Assumptions:	App User has searched for item using keyword beforehand.
Notes and Issues:	NIL

4.5.3 Functional Requirements

5. App User must be able to filter search for items based on parameters set.
   5.1. System must provide four parameter adjustment options to App User.
      5.1.1. One of the parameters must be price.
         5.1.1.1. The price parameter must be a slider adjustment option.
      5.1.2. One of the parameters must be ratings.

5.1.2.1. The ratings parameter must be a checkbox adjustment option.

5.1.3. One of the parameters must be platform.

5.1.3.1. The platform parameter must be a checkbox adjustment option.

5.1.4. One of the parameters must be delivery fee.

5.1.4.1. The delivery fee parameter must be a slider adjustment option.

5.2. System must filter the searched result to retrieve all items which match the parameters set.

5.2.1. Items filtered must satisfy all the parameters set by App User.

5.3. System must display the items filtered to App User.

5.3.1. System must display information about each item to App User.

5.3.1.1. Information must contain item name.

5.3.1.2. Information must contain item price.

5.3.1.3. Information must contain item description.

5.3.1.4. Information must contain item ratings.

5.3.1.5. Information must contain platform that the item is sold on.

5.3.1.6. Information must contain the total number of reviews an item has.

5.3.2. System must display an error message to App User if there are no items which match the parameters set.

4.6 Recommend Items

4.6.1 Description and Priority

Based on search history, App User is recommended similar items using this use case.

Overall Priority	Medium

4.6.2 Stimulus/Response Sequences

Use Case ID:	UC-REC-001		
Use Case Name:	Recommend Item		
Created By:	Jerick Lim Kai Zheng	Updated By:	Lee Juin
Date Created:	22nd August 2022	Date Updated:	17th October 2022

Actor:	App User, Keyword Extraction API
Description:	Based on search history, App User is recommended similar items using this use case.
Preconditions:	1. App User is connected to the Internet. 2. App User has logged in to his/her account. 3. App User has searched using keywords for at least than ten times.
Postconditions:	App User obtains a list of recommended items based on their search history.
Priority:	Medium

Frequency of Use:	Medium
Flow of Events:	1. App User inputs a keyword and clicked on the search icon. 2. System queries the database and returns a list of items matching the keyword. 3. If App User has searched at least ten times, System sends App User past search history to Keyword Extraction API. 4. Keyword Extraction API extracts the three most relevant keywords from the search history based on Natural Language Processing algorithms. 5. System receives the three keywords and queries the database using the keywords. 6. When the App User scrolls to the bottom of the page, App User can view the section of "You may also like:" which displays the recommended items.
Alternative Flows:	**UC-REC-AF-01: If App User search history has less than ten entries** 1. System displays the message "Happy Hunting" at the section of "You may also like:" to the App User. 2. System returns to Step 1 and wait until App User search history has at least ten entries.
Exceptions:	NIL
Includes:	NIL
Special Requirements:	NIL
Assumptions:	NIL

Notes and Issues:	1. The Keyword Extraction API that *Findr* is using is a freemium API. This means that at most 300 requests can be made per month before a halt to all requests.

4.6.3 Functional Requirements

6. System must be able to recommend App User items based on search history.
   6.1. System must provide at most three recommended items to App User.
      6.1.1. Recommended items must be displayed after the searched items.

   6.2. System must only recommend items if App User search history has at least ten entries.
      6.2.1. System must only keep at most 50 search history entries per App User.
      6.2.2. System must display error message to App User if App User search history has less than ten entries.

4.7 Manage Friend List

4.7.1 Description and Priority

App User can add, accept, remove, and view friends from a friend list using this use case. Each App User will have one friend list containing all the friends that they have added.

Overall Priority	Medium

4.7.2 Stimulus/Response Sequences

Use Case ID:	UC-MFL-001		
Use Case Name:	Manage Friendlist		
Created By:	Lee Juin	Updated By:	Lee Juin
Date Created:	17th September 2022	Date Updated:	17th September 2022

Actor:	App User
Description:	App User can add, accept, remove, and view friends from a friend list using this use case. Each App User will have one friend list containing all the friends that they have added.
Preconditions:	1. App User is connected to the Internet. 2. App User has logged in to his/her account.
Postconditions:	App User adds, accepts, removes, and view friends from their friend list.
Priority:	Medium
Frequency of Use:	Medium
Flow of Events:	1. App User clicks on "Friendlist" icon to access their friendlist. 2. System queries the database to find all other App Users who are friends with the current App User. 3. System also queries for any incoming pending friend requests to the App User. 4. System displays the incoming pending friend requests followed by friends of App User.

	5. App User can perform operations on the friend requests using the included *UC-AFR-001* and *UC-RFR-001* use cases. 6. App User can add other App Users using the included *UC-SFR-001* use case. 7. System will send a birthday notification to App User using the included *UC-SBN-001* use case.
Alternative Flows:	**UC-MFL-AF-01: If App User has no friends or incoming friend requests** 1. System displays the message "User has no added friends" instead. 2. System will only display friends when App User has accepted a friend request, or their friend request is accepted.
Exceptions:	NIL
Includes:	1. *UC-AFR-001* 2. *UC-RFR-001* 3. *UC-SFR-001* 4. *UC-SBN-001*
Special Requirements:	NIL
Assumptions:	NIL
Notes and Issues:	NIL

4.7.3 Functional Requirements

7. System must allow App User to manage their friend list.
   7.1. System must provide four functionalities for App User to manage their friend list.

7.1.1. One of the functionalities must be the ability to accept incoming friend requests.

7.1.2. One of the functionalities must be the ability to reject incoming friend requests.

7.1.3. One of the functionalities must be the ability to search for another App User.

7.1.4. One of the functionalities must be the ability to send a friend request to another App User.

7.2. System must display all friends of App User and any incoming pending friend requests.

7.2.1. System must display friends' information in the correct format.

7.2.1.1. Information must contain friends' username.

7.2.1.2. Information must contain friends' name.

7.2.1.3. Information must contain friends' birthday.

7.2.2. If friend has not set up their name or birthday, System must report that to App User.

7.2.3. System must display error message if App User has no added friends yet.

4.8 Accept Friend Requests

4.8.1 Description and Priority

App User can accept incoming friend request using this use case. Once a friend request has been accepted, App User may view their friend's wish list.

Overall Priority	Medium

4.8.2 Stimulus/Response Sequences

Use Case ID:	UC-AFR-001		
Use Case Name:	Accept Friend Request		
Created By:	Lee Juin	Updated By:	Lee Juin
Date Created:	17th September 2022	Date Updated:	17th September 2022

Actor:	App User
Description:	App User can accept incoming friend request using this use case. Once a friend request has been accepted, App User may view their friend's wish list.
Preconditions:	1. App User is connected to the Internet. 2. App User has logged in to his/her account. 3. App User has at least one incoming pending friend request.
Postconditions:	App User accepts the friend request.
Priority:	Medium
Frequency of Use:	Medium
Flow of Events:	1. System displays all incoming pending friend requests to App User. 2. App User clicks on "Accept" next to the friend request to accept the friend request.

	3. System displays the message "Friend request accepted" to indicate that the friend connection has been recorded in the database. 4. When App User reloads the page, App User can view their friend's wish list. 5. System updates the status in the friend's friend list to include App User as a friend.
Alternative Flows:	NIL
Exceptions:	NIL
Includes:	NIL
Special Requirements:	NIL
Assumptions:	NIL
Notes and Issues:	NIL

4.8.3 Functional Requirements

8. System must allow App User to accept incoming friend requests.
   8.1. System must provide the option to manage the friend requests in the same format.
      8.1.1. System must provide an option to accept the friend request.
      8.1.2. An option to reject the friend request follows.

   8.2. System must display a message to inform App User that the friend request has been accepted.

4.9 Reject Friend Requests

4.9.1 Description and Priority

App User can reject incoming friend request using this use case. Once a friend request has been rejected, the pending friend request will be removed from display. System allows the friend to send another friend request.

Overall Priority	Medium

4.9.2 Stimulus/Response Sequences

Use Case ID:	UC-RFR-001		
Use Case Name:	Reject Friend Request		
Created By:	Lee Juin	Updated By:	Lee Juin
Date Created:	17th September 2022	Date Updated:	17th September 2022

Actor:	App User
Description:	App User can reject incoming friend request using this use case. Once a friend request has been rejected, the pending friend request will be removed from display. System allows the friend to send another friend request.
Preconditions:	1. App User is connected to the Internet. 2. App User has logged in to his/her account. 3. App User has at least one incoming pending friend request.

Postconditions:	App User rejects the friend request.
Priority:	Medium
Frequency of Use:	Medium
Flow of Events:	1. System displays all incoming pending friend requests to App User. 2. App User clicks on "Reject" next to the friend request to reject the friend request. 3. System displays the message "Friend request rejected" to indicate that the friend request has been removed from the database. 4. When App User reloads the page, the pending request will not be shown to App User anymore. 5. System updates the status in the friend's friend list to remove the pending friend request. 6. System allows the friend to send another friend request to App User.
Alternative Flows:	NIL
Exceptions:	NIL
Includes:	NIL
Special Requirements:	NIL
Assumptions:	NIL
Notes and Issues:	NIL

4.9.3 Functional Requirements

9. System must allow App User to reject incoming friend requests.
   9.1. System must provide the option to manage the friend requests in the same format.
      9.1.1. System must provide an option to accept the friend request.
      9.1.2. An option to reject the friend request follows.

   9.2. System must display a message to inform App User that the friend request has been rejected.

   9.3. System must allow the friend to send another friend request to App User after the friend request has been rejected.

4.10 Send Friend Requests

4.10.1 Description and Priority

App User can send friend request to another App User using this use case. App User searches an App User by their username and adds the other App User as friend.

Overall Priority	Medium

4.10.2 Stimulus/Response Sequences

Use Case ID:	UC-SFR-001		
Use Case Name:	Send Friend Request		
Created By:	Lee Juin	Updated By:	Lee Juin

Date Created:	17th September 2022	Date Updated:	17th September 2022

Actor:	App User
Description:	App User can send friend request to another App User using this use case. App User searches an App User by their username and adds the other App User as friend.
Preconditions:	1. App User is connected to the Internet. 2. App User has logged in to his/her account.
Postconditions:	App User sends a friend request to the searched App User
Priority:	Medium
Frequency of Use:	Medium
Flow of Events:	1. App User inputs a username at the search bar and clicks on the search icon. 2. System queries the database and retrieve a user information based on the username. 3. System displays the user information to App User. 4. App User clicks on "Add Friend" to send a friend request. 5. System displays a message "Friend request has been sent" to indicate that a pending friend request has been recorded in the database.
Alternative Flows:	**UC-SFR-AF-01: If searched username does not match any App User** 1. System displays the message "User not found" instead.

	2. System returns to Step 1 and wait for App User inputs. **UC-SFR-AF-02: If searched username matches a friend of App User** 1. System displays the message "User is already a friend" instead. 2. System returns to Step 1 and wait for App User inputs. **UC-SFR-AF-03: If searched username matches App User** 1. System displays the message "User searched is user itself" instead. 2. System returns to Step 1 and wait for App User inputs.
Exceptions:	NIL
Includes:	NIL
Special Requirements:	NIL
Assumptions:	NIL
Notes and Issues:	NIL

4.10.3 Functional Requirements

10. System must allow App User to send friend requests to another App User.
    10.1. System must provide an input field for App User to input username.
        10.1.1. System must ensure that App User filled in the input field before querying the database for a user.

    10.2. System must display the retrieved user's information in the same format.
        10.2.1. System must display the username.

10.2.2. System must display the name.

10.2.3. System must display the birthday.

10.2.4. System must provide an option to send friend request if the user is not friend with App User.

10.3. System must display error message to App User if searched user cannot be added as friend.

10.3.1. System must inform App User that the searched username belongs to an existing friend.

10.3.2. System must inform App User that the searched username cannot be found.

10.3.3. System must inform App User that the searched username is App User themselves.

4.11 Send Birthday Notifications

4.11.1 Description and Priority

App User can receive birthday notifications of their friends if their birthdays are within seven days.

Overall Priority	Low

4.11.2 Stimulus/Response Sequences

Use Case ID:	UC-SBN-001
Use Case Name:	Send Birthday Notifications

Created By:	Lee Juin	Updated By:	Lee Juin
Date Created:	17th October 2022	Date Updated:	17th October 2022

Actor:	App User
Description:	App User can receive birthday notifications of their friends if their birthdays are within seven days.
Preconditions:	1. App User is connected to the Internet. 2. App User has logged in to his/her account.
Postconditions:	App User receives a birthday notification informing them that their friend's birthday is happening soon.
Priority:	Low
Frequency of Use:	Low
Flow of Events:	1. System will query each App User with at least one friend every day. 2. Each query checks if any of App User's friend's birthday is happening within seven days. 3. System sends an email notification to App User informing that their friend's birthday is happening soon.
Alternative Flows:	**UC-SFR-AF-01: If friends of App User have not set up their birthday** 1. System will not send any birthday notification about the friends with no birthday information recorded.

	2. System will only start tracking once the friend has updated their birthday information.
Exceptions:	NIL
Includes:	NIL
Special Requirements:	NIL
Assumptions:	NIL
Notes and Issues:	NIL

4.11.3 Functional Requirements

11. System must send email notification about a friend's upcoming birthday to App User.
    11.1. Email notification sent must contain friend's name and the date of friend's birthday.

4.12 Manage Wish List

4.12.1 Description and Priority

App User can add, remove, and view items that from a wish list using this use case. Each App User will have one wish list containing all the items that they have added.

Overall Priority	Medium

4.12.2 Stimulus/Response Sequences

Use Case ID:	UC-MWL-001		
Use Case Name:	Manage Wishlist		
Created By:	Lee Juin	Updated By:	Lee Juin
Date Created:	17th October 2022	Date Updated:	17th October 2022

Actor:	App User
Description:	App User can add, remove, and view items that from a wish list using this use case. Each App User will have one wish list containing all the items that they have added.
Preconditions:	1. App User is connected to the Internet. 2. App User has logged in to his/her account.
Postconditions:	App User adds, removes, and view items from their wish list.
Priority:	Medium
Frequency of Use:	Medium
Flow of Events:	1. App User clicks on "Wishlist" icon to access their wishlist. 2. System queries the database to find all items that App User has added into their wish list. 3. System displays the information of all wish list items to App User. 4. App User can add new items using the included *UC-AWI-001* use case.

	5. App User can remove existing wish list items using the included *UC-RWI-001* use case.
Alternative Flows:	**UC-MFL-AF-01: If App User has no wish list items** 3. System displays the message "User has no wish list items added yet" instead. 4. System will only display items inside the wish list when App User has added an item into it.
Exceptions:	NIL
Includes:	1. *UC-AWI-001* 2. *UC-RWI-001*
Special Requirements:	NIL
Assumptions:	NIL
Notes and Issues:	NIL

4.12.3 Functional Requirements

12. System must allow App User to manage their wish list.
    12.1. System must provide two functionalities for App User to manage their wish list.
        12.1.1. One of the functionalities must be the ability to add an item into the wish list.
        12.1.2. One of the functionalities must be the ability to remove an item from the wish list.

    12.2. System must display all items added into the wish list.
        12.2.1. System must display items' information in the correct format.

12.2.1.1. Information must contain item name.

12.2.1.2. Information must contain item price.

12.2.1.3. Information must contain platform of which the item is sold on.

12.2.1.4. Information must contain item delivery fee.

12.2.1.5. Information must contain item ratings.

12.2.1.6. Information must contain total number of ratings the item received.

12.2.2. System must display error message if App User has no added wish list items yet.

4.13 Add Wish List Items

4.13.1 Description and Priority

App User can add a wish list item into their wish list using this use case.

Overall Priority	Medium

4.13.2 Stimulus/Response Sequences

Use Case ID:	UC-AWL-001		
Use Case Name:	Add Wishlist Item		
Created By:	Lee Juin	Updated By:	Lee Juin
Date Created:	17th October 2022	Date Updated:	17th October 2022

Actor:	App User
Description:	App User can add a wish list item into their wish list using this use case.
Preconditions:	1. App User is connected to the Internet. 2. App User has logged in to his/her account.
Postconditions:	App User adds an item into their wish list.
Priority:	Medium
Frequency of Use:	Medium
Flow of Events:	1. When System displays item information, System allows App User to add an item to their wish list. 2. App User clicks on "Add to wish list" to add the items to their wish list. 3. System displays a message "Item added successfully" to indicate that the item has been added to their wish list. 4. App User can view the added item in their wish list.
Alternative Flows:	NIL
Exceptions:	NIL
Includes:	NIL
Special Requirements:	NIL
Assumptions:	NIL
Notes and Issues:	NIL

4.13.3 Functional Requirements

13. System must allow App User to add items to their wish list.
    13.1. System must display an option for App User to add item to wish list.
    13.2. System must display a message to indicate that the item has been added to wish list to App User.

4.14 Remove Wish List Items

4.14.1 Description and Priority

App User can remove a wish list item from their wish list using this use case.

Overall Priority	Medium

4.14.2 Stimulus/Response Sequences

Use Case ID:	UC-RWL-001		
Use Case Name:	Remove Wishlist Item		
Created By:	Lee Juin	Updated By:	Lee Juin
Date Created:	17th September 2022	Date Updated:	17th October 2022

Actor:	App User

Description:	App User can remove a wish list item from their wish list using this use case.
Preconditions:	1. App User is connected to the Internet. 2. App User has logged in to his/her account.
Postconditions:	App User removes an item from their wish list.
Priority:	Medium
Frequency of Use:	Low
Flow of Events:	1. When System displays item information, System allows App User to remove an item from their wish list. 2. App User clicks on "Remove from wish list" to remove the items from their wish list. 3. System displays a message "Item removed successfully" to indicate that the item has been removed from their wish list.
Alternative Flows:	NIL
Exceptions:	NIL
Includes:	NIL
Special Requirements:	NIL
Assumptions:	NIL
Notes and Issues:	NIL

4.14.3 Functional Requirements

14. System must allow App User to remove items from their wish list.

14.1. System must display an option for App User to remove the item if the item is already in the wish list.

14.2. System must display a message to indicate that the item has been removed from wish list to App User.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

5.1.1 Concurrent Users

1. System must be able to accommodate up-to four concurrent users.
   1.1. All stakeholders understand the constraints faced by the *FindR* development team on limited features provided by the free database hosting service. However, the System must nonetheless support at most four concurrent users to demonstrate the functionalities of the web application as documented in section *4. System Features*.
   1.2. Users must be able to login to their account concurrently within 10 seconds.

5.1.2 System Response Time

2. System must be able to respond to user's input within 5 seconds.
   2.1. System must register user's search query within 5 seconds.
      2.1.1. Search result must be returned to user within 20 seconds.

5.2 Safety Requirements

5.2.1 Accuracy of Reflected Price

3. System must be able to accurately reflect the price of an item listing sold on a platform.

3.1. System must accurately represent the price of the item in the correct currency, which is Singapore Dollar (SGD).

3.2. System must consistently represent decimal-point using a period (.) symbol.

5.3 Security Requirements

5.3.1 Enforced Login Requirement

4. System must ensure user is logged in prior to accessing any functionalities provided within the app.
   4.1. User must be logged in using their registered username and password.
   4.2. System must verify the credentials are correct using a hasher.
      4.2.1. The hasher used must match the hashing algorithm, SHA-256, used to hash the stored password.
   4.3. System must redirect user to login page if user attempts to access unauthorized contents.

5. System must hash all passwords prior to storing them in the database.
   5.1. System must use the SHA-256 algorithm to hash all user's password.

5.4 Software Quality Attributes

5.4.1 Correctness

6. System must ensure that the correct items are returned to the user based on the searched keywords.
   6.1. The searched keyword must match the item's name.
   6.2. Or the searched keyword must match the item's description.

7. System must ensure that the user's account information is always accurately displayed.
   7.1. System must accurately display the username registered by the user.
   7.2. System must accurately display the name provided by the user.

7.3. System must accurately display the email registered by the user.

7.4. System must accurately display the user's birthday in the YYYY-MM-DD format.

8. System must ensure that the user's friend list is always accurately displayed.

8.1. System must accurately retrieve and display the other users' information who are friends with the user.

8.1.1. System must accurately display the friend's username.

8.1.2. System must accurately display the friend's name.

9. System must ensure that the user's wish list is always accurately displayed.

9.1. System must ensure that an item which is in the user's wish list must be displayed as added to wish list status.

9.2. System must ensure that an item which is not in the user's wish list must be displayed as to be added to wish list status.

5.4.2 Extensibility

10. System must always be ready to accept new functionality.

10.1. System design should adopt modular programming concept using open-closed principle.

5.4.3 User-Friendliness

11. System must provide a quick navigation at appropriate pages.

11.1. One of the navigation links must be to the login page at the signup page.

11.2. One of the navigation links must be to the signup page at the login page.

11.3. One of the navigation links must be to the forgot password page at the login page.

12. System must provide indication to the user that a content is loading.
    - 12.1. A loading screen must appear to indicate to the user that a content is buffering.
    - 12.2. A loading animation on the button must appear to indicate to the user that the input is being processed.

13. System must provide a FAQ page.
    - 13.1. The FAQ page must contain general potential enquiries of a user.
        - 13.1.1. One of the enquiries must be ways to navigate around the web application.
        - 13.1.2. One of the enquiries must be the license agreement of using the web application.
        - 13.1.3. One of the enquiries must be the purpose of the web application.
        - 13.1.4. One of the enquiries must be the e-commerce platforms supported by the web application.

Appendix A: Data Dictionary

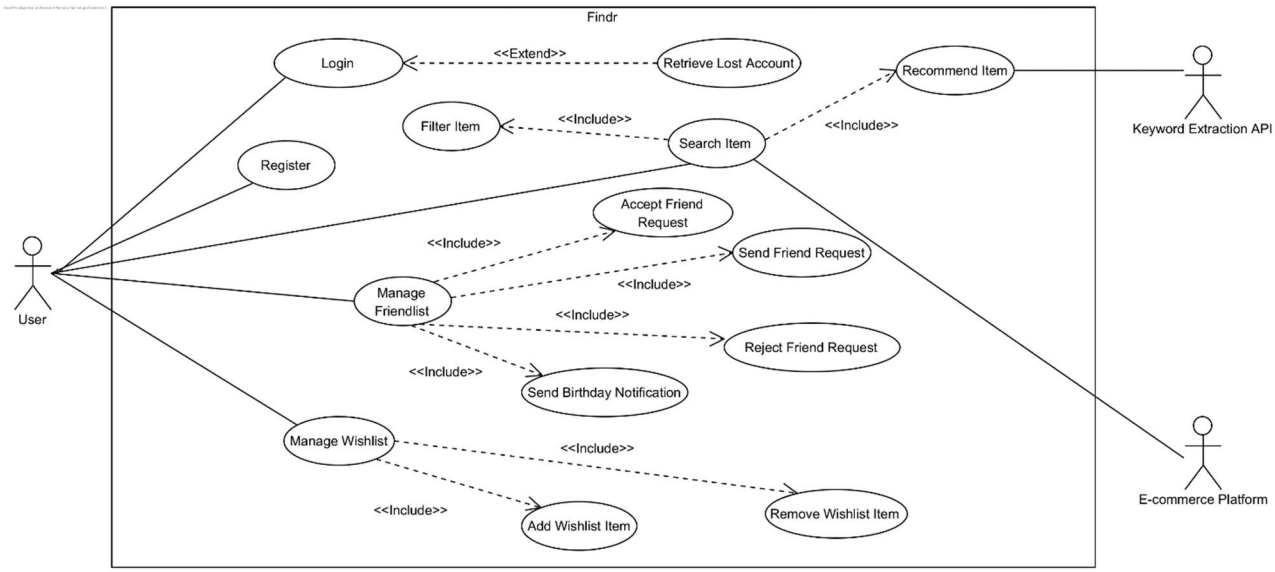
Created By:	Lee Juin	Last Updated By:	Lee Juin
Date Created:	19th August 2022	Date Updated:	18th October 2022

App User	An individual who holds an account with the *FindR* web application. The account must be formally registered via the *FindR* web application and must be retrievable from the Database. The individual is entitled to use all the services provided within the web application, which includes but not limited to, adding other App Users, searching for an item using keywords and adding an item to a wish list.
E-commerce platform	A platform which allows sellers to advertise their item listings. In the context of this web application, it is data that are scrapped by a scraper that provides all relevant information about an item sold on a particular platform, which includes the name, the price, the rating, the delivery fee, the payment methods available and any relevant rebates.
Keyword Extraction API	An API which extracts the most relevant keyword from a text based on Natural Language Processing algorithm. The API is called to extract the most relevant search history keyword of an App User when recommending items.
Wish list	A list which contains all the items that are added by the App User. The items in the wish list are defined as preferred items by the App User to be bought as gift by their friends. The wish list is publicly accessible by the App User and their friends.

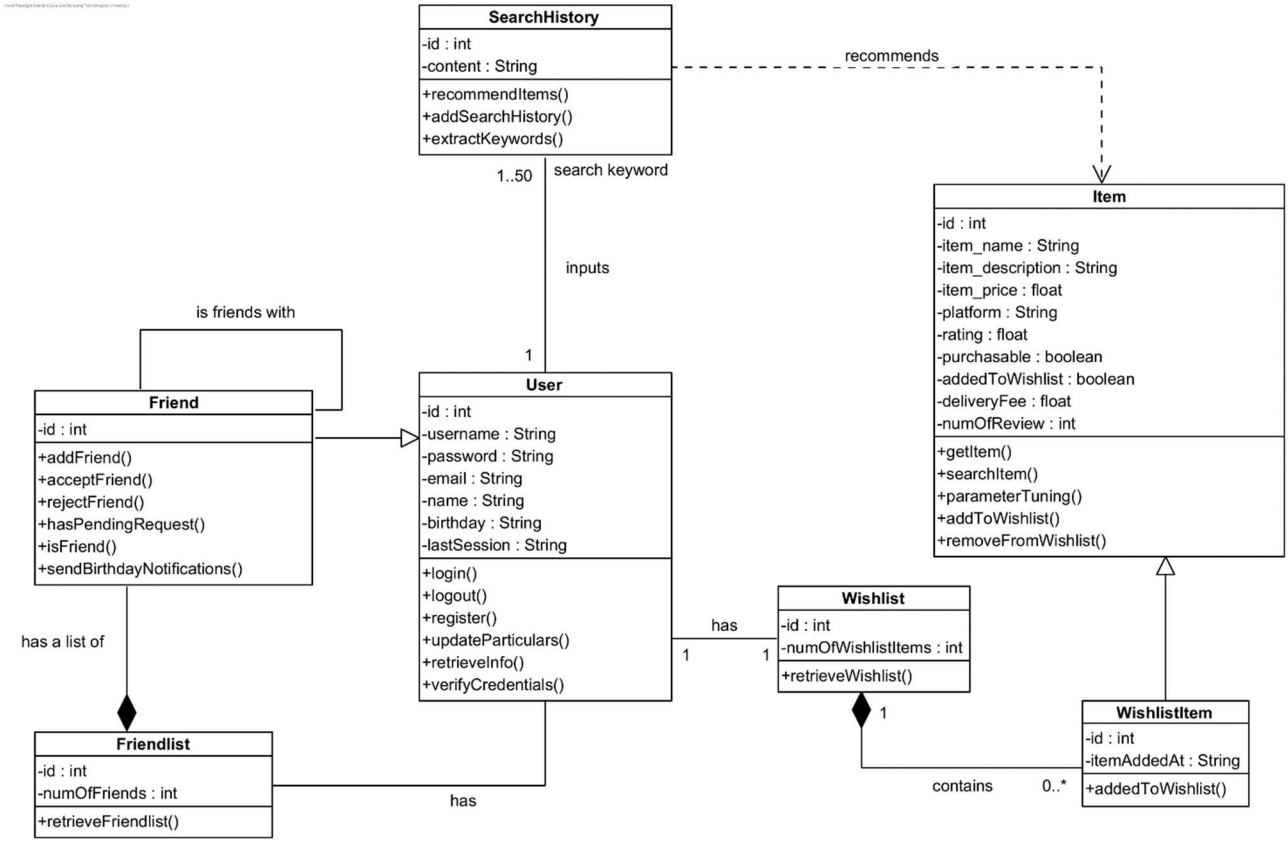
One-time Password (OTP)	An eight-digit combination which is sent to the App User via his/her registered email address. The OTP serves as an additional layer of security in the event where an App User forgets their login credentials.
Username	A unique identification set by an App User which serves as a locator. An App User may find other App Users by searching for their username.
Database	An online spreadsheet which contains all information of each App User such as their username, registered email address, hashed password, name, birthday, and wish list items. The password is hashed using Secure Hash Algorithm (SHA).
Delivery fee	A column under each item which displays the cost of delivering an item if the App User purchases it. The delivery fee column is displayed in Singapore Dollar (SGD) currency. The delivery fee will not be displayed if the item is sold out.
Rating	A scaling system implemented by individual e-commerce platforms for past customers to rate the bought items. The scale of the system is from one star to five star, where one star represents poorest experience, and five star represents best experience. The rating column displays the average of all rates given by the past customers.

Appendix B: Analysis Models

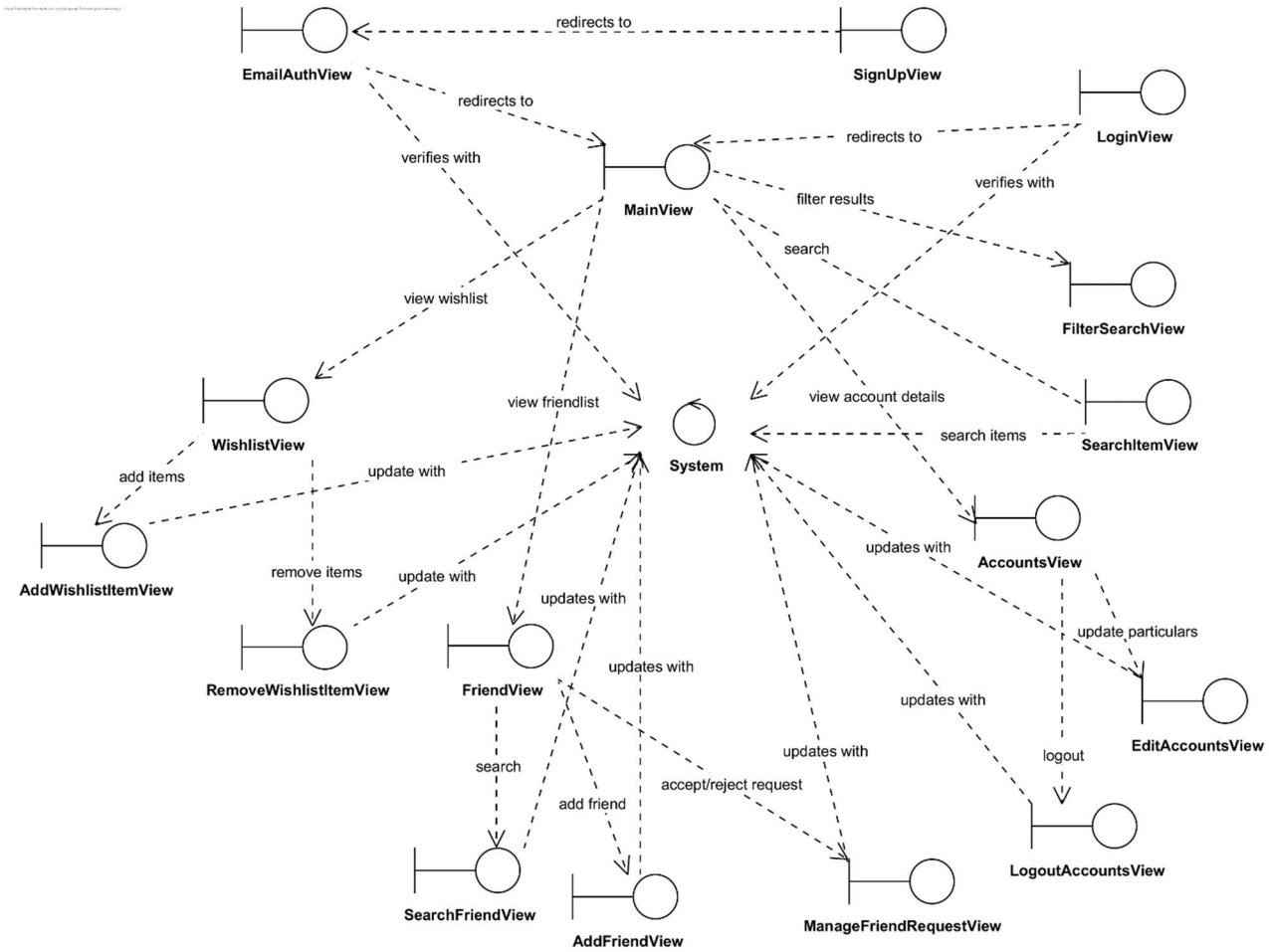
B.1 Use Case Model



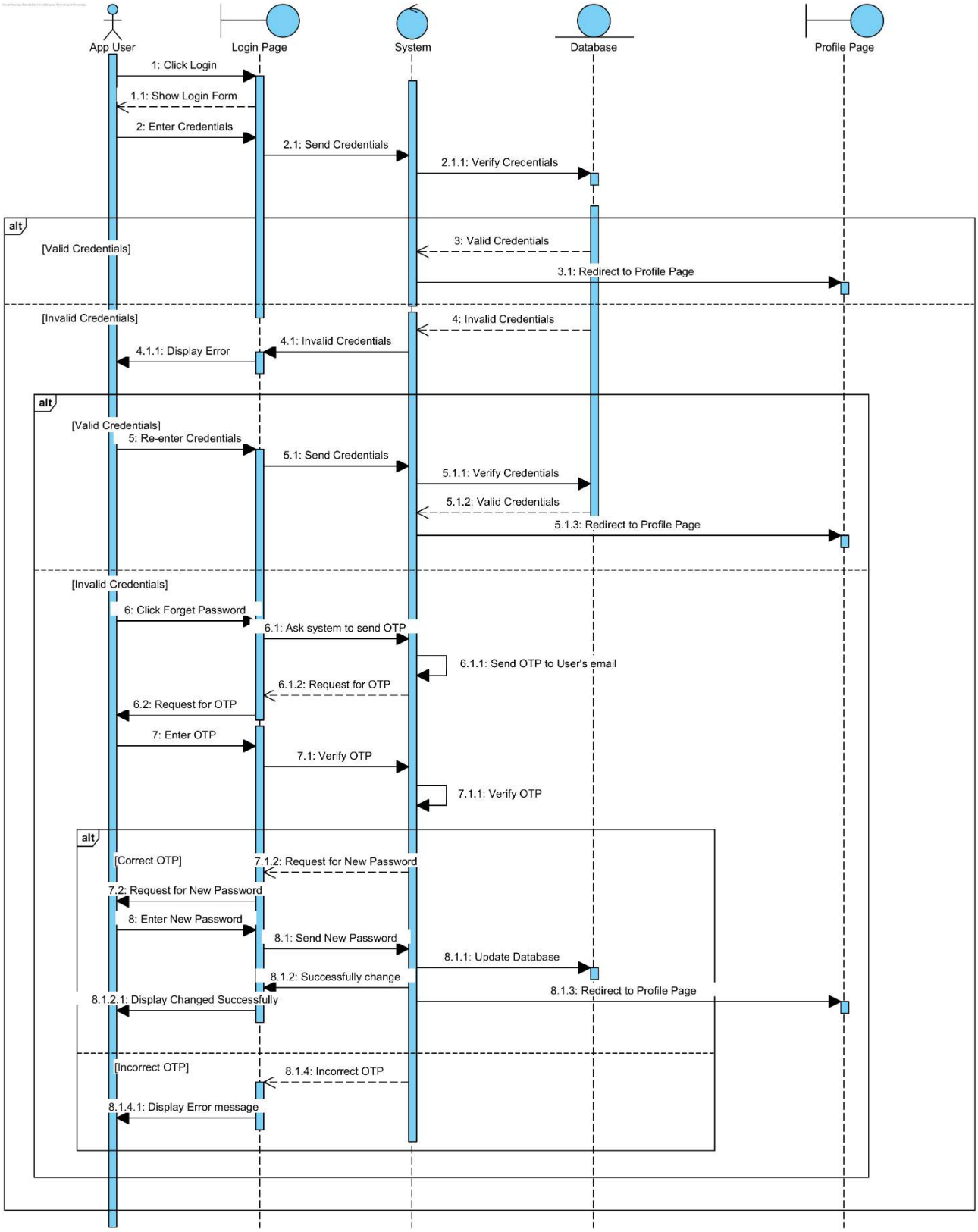
B.2 Entity Class Diagram

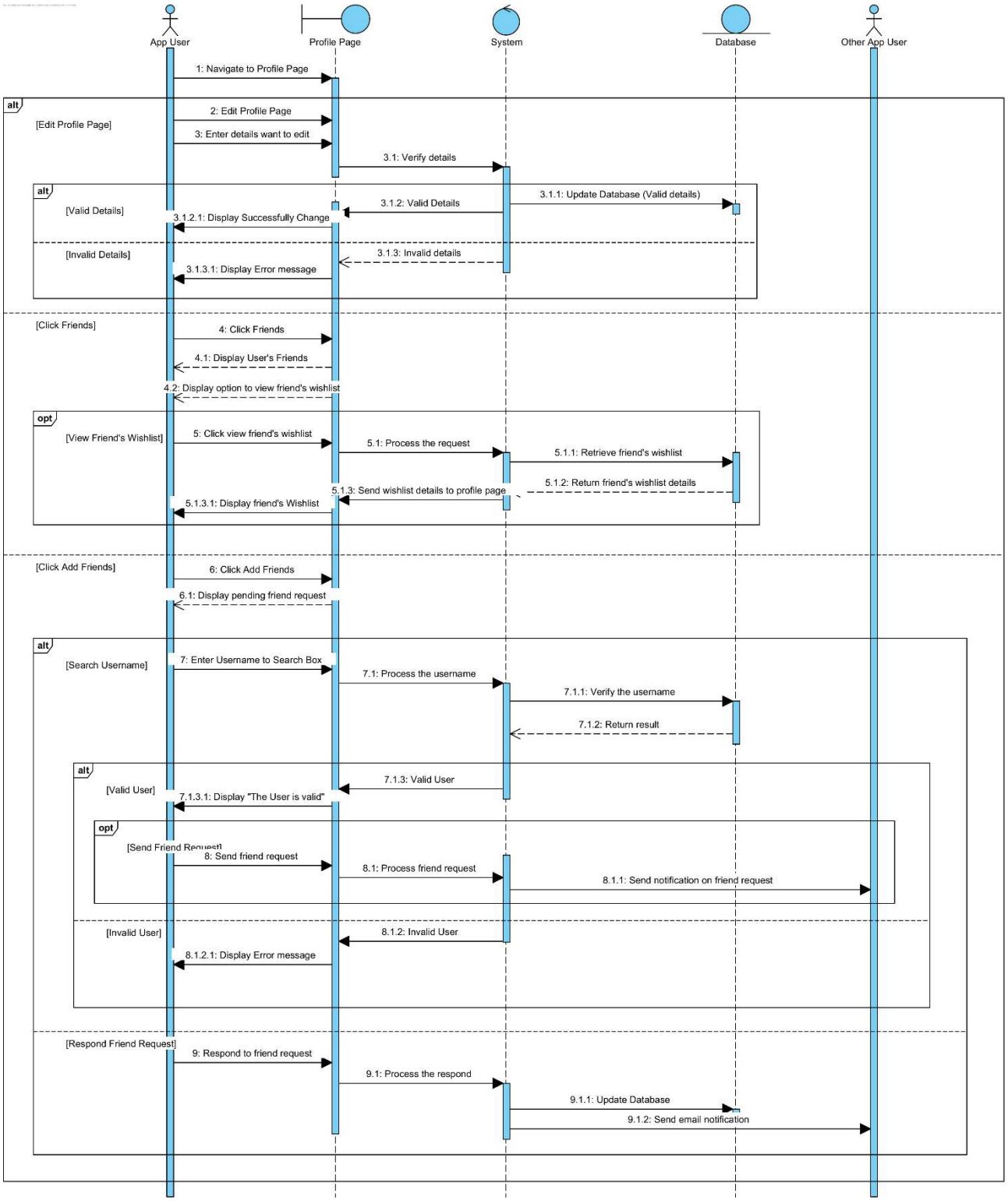


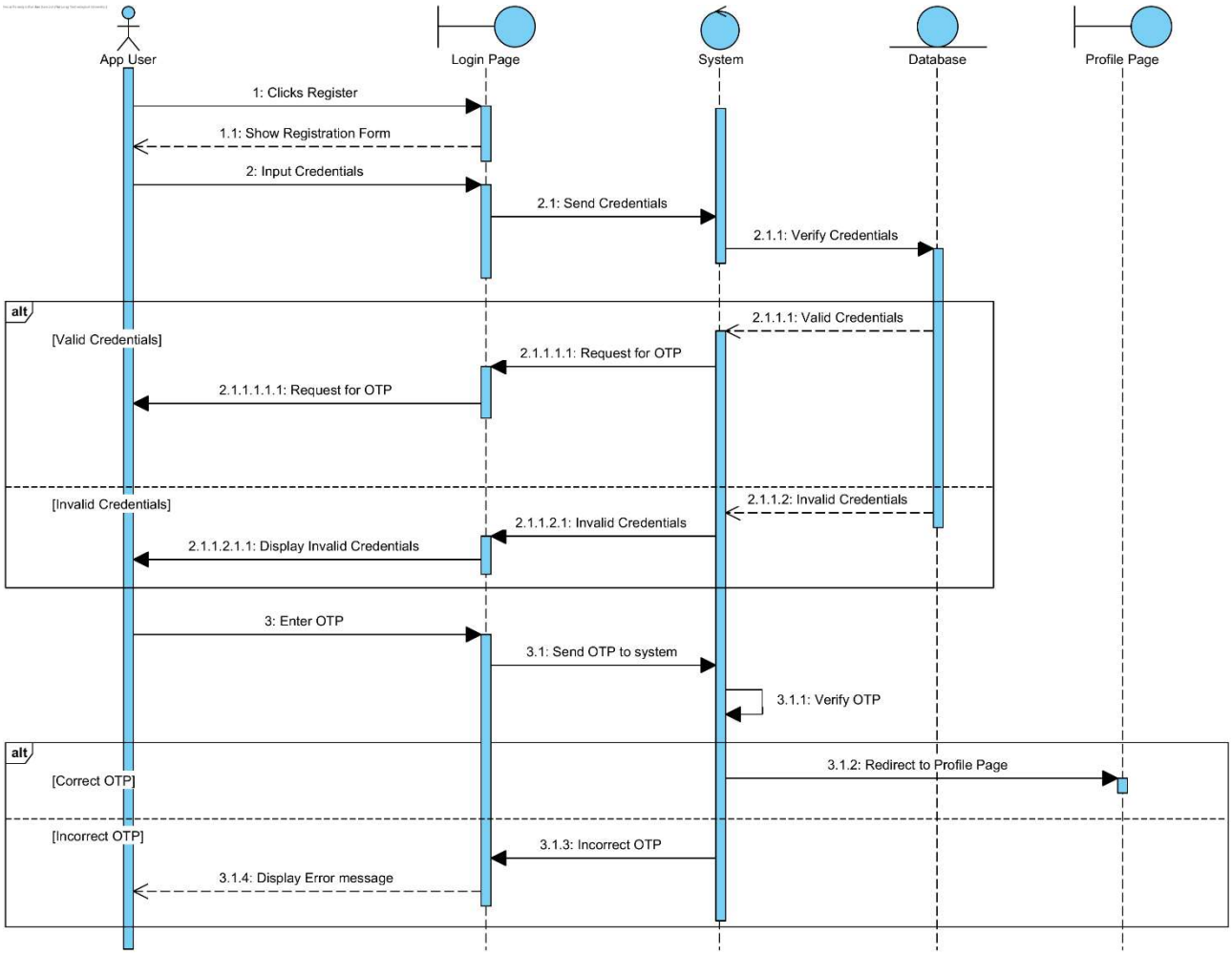
B.3 Control and Boundary Class Diagram

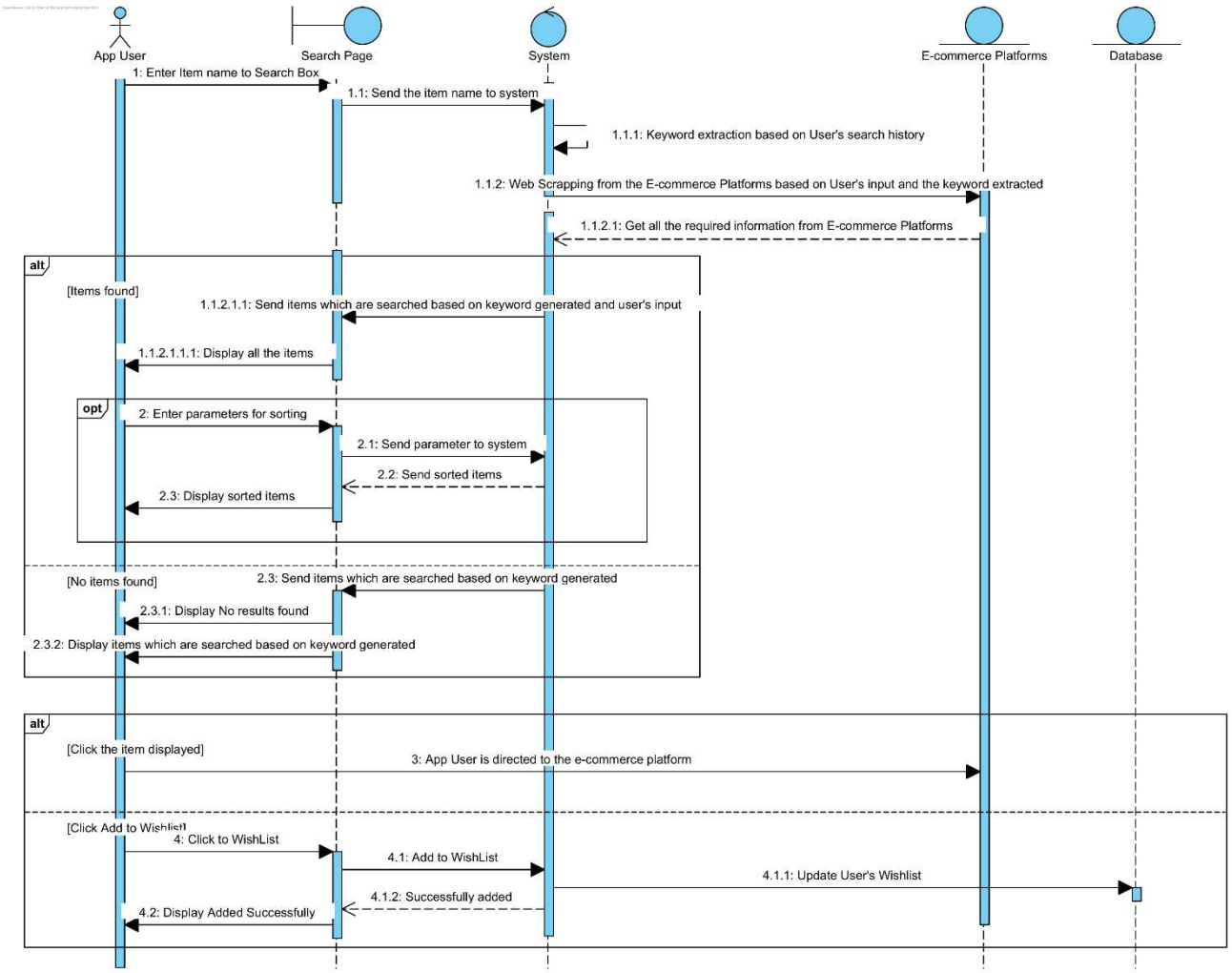


B.4 Sequence Diagram

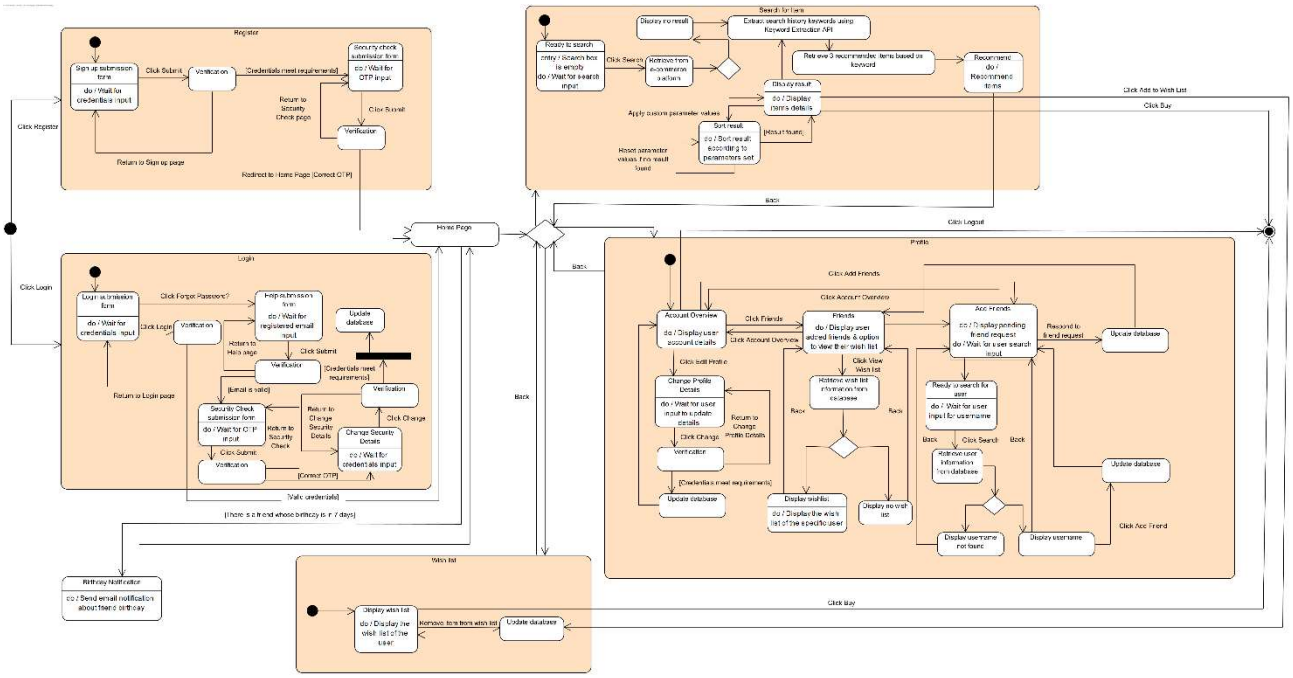



App User
Profile Page
System
Database
Other App User
1: Navigate to Profile Page
alt
[Edit Profile Page]
2: Edit Profile Page
3: Enter details want to edit
3.1: Verify details
alt
[Valid Details]
3.1.1: Update Database (Valid details)
3.1.2: Valid Details
3.1.2.1: Display Successfully Change
[Invalid Details]
3.1.3: Invalid details
3.1.3.1: Display Error message
[Click Friends]
4: Click Friends
4.1: Display User's Friends
4.2: Display option to view friend's wishlist
opt
[View Friend's Wishlist]
5: Click view friend's wishlist
5.1: Process the request
5.1.1: Retrieve friend's wishlist
5.1.2: Return friend's wishlist details
5.1.3: Send wishlist details to profile page
5.1.3.1: Display friend's Wishlist
[Click Add Friends]
6: Click Add Friends
6.1: Display pending friend request
alt
[Search Username]
7: Enter Username to Search Box
7.1: Process the username
7.1.1: Verify the username
7.1.2: Return result
alt
[Valid User]
7.1.3: Valid User
7.1.3.1: Display "The User is valid"
opt
[Send Friend Request]
8: Send friend request
8.1: Process friend request
8.1.1: Send notification on friend request
[Invalid User]
8.1.2: Invalid User
8.1.2.1: Display Error message
[Respond Friend Request]
9: Respond to friend request
9.1: Process the respond
9.1.1: Update Database
9.1.2: Send email notification

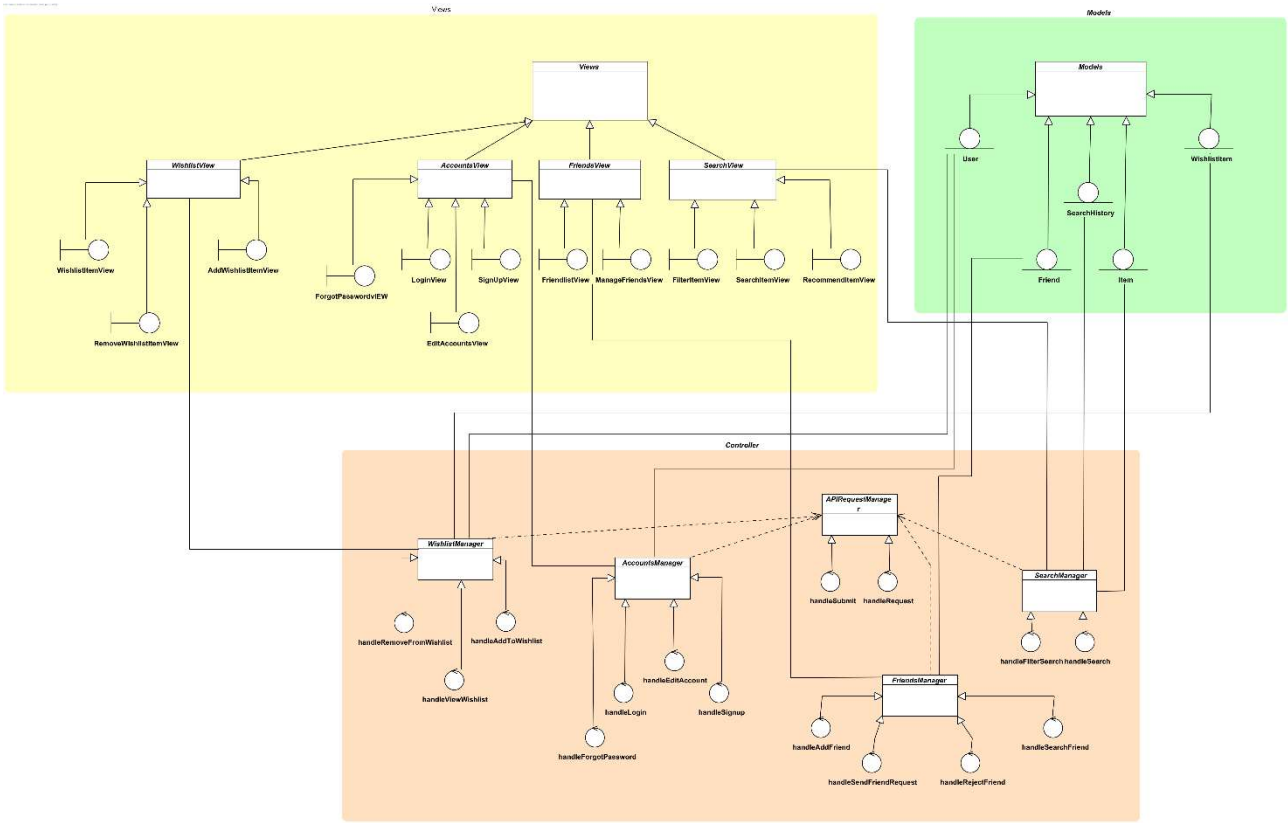

App User
Login Page
System
Database
Profile Page
1: Clicks Register
1.1: Show Registration Form
2: Input Credentials
2.1: Send Credentials
2.1.1: Verify Credentials
alt
[Valid Credentials]
2.1.1.1: Valid Credentials
2.1.1.1.1: Request for OTP
2.1.1.1.1.1: Request for OTP
[Invalid Credentials]
2.1.1.2: Invalid Credentials
2.1.1.2.1: Invalid Credentials
2.1.1.2.1.1: Display Invalid Credentials
3: Enter OTP
3.1: Send OTP to system
3.1.1: Verify OTP
alt
[Correct OTP]
3.1.2: Redirect to Profile Page
[Incorrect OTP]
3.1.3: Incorrect OTP
3.1.4: Display Error message


App User
Search Page
System
E-commerce Platforms
Database
1: Enter Item name to Search Box
1.1: Send the item name to system
1.1.1: Keyword extraction based on User's search history
1.1.2: Web Scrapping from the E-commerce Platforms based on User's input and the keyword extracted
1.1.2.1: Get all the required information from E-commerce Platforms
alt
[Items found]
1.1.2.1.1: Send items which are searched based on keyword generated and user's input
1.1.2.1.1.1: Display all the items
opt
2: Enter parameters for sorting
2.1: Send parameter to system
2.2: Send sorted items
2.3: Display sorted items
[No items found]
2.3: Send items which are searched based on keyword generated
2.3.1: Display No results found
2.3.2: Display items which are searched based on keyword generated
alt
[Click the item displayed]
3: App User is directed to the e-commerce platform
[Click Add to Wishlist]
4: Click to WishList
4.1: Add to WishList
4.1.1: Update User's Wishlist
4.1.2: Successfully added
4.2: Display Added Successfully

B.5 State Machine Diagram



B.6 System Architecture Model



Appendix C: Supplementary Materials

In the E-commerce platforms we supported, which are Shopee, Lazada, and Amazon, the developers of these websites used different strategies in managing their websites. Shopee and Lazada use dynamic website, while Amazon uses static website. This leads to the need for different tools during web scraping.

C.1 Static Website

Static website is a website that will return the HTML file and any accompanying CSS file when a user requests a page. During this exchange, the web server will send a web page that will look the same to everyone who requests it. Hence, the content is "static". In dealing with static websites, two libraries are used, which are:

- *requests 2.28.1*
  Requests allows the user to send HTTP requests to the website using Python. The method *requests.get()* is used to send a GET request to the website, and the website will then return a Response object, which contains the server's response to the HTTP request.
- *BeautifulSoup 4.11.1*
  Beautiful Soup is a Python library for pulling data from HTML and XML documents. Beautiful Soup can process the response object obtained by *requests.get()* and extract all the information required from the HTML by specifying the HTML tag name, class name, id, etc.

C.2 Dynamic Website

When dealing with dynamic website, there are some limitations with the use of *requests* and *BeautifulSoup* libraries. This is because when we request for the website, dynamic website will send the JavaScript (JS) code to be executed locally, and the *requests* library will only provide us the page source. Hence, we will need to use another library, *Selenium*, with *BeautifulSoup* in scraping dynamic websites.

- *Selenium 4.5.0*
  Selenium is a tool for controlling web browsers through programs and performing browser automation. With the combination of methods in *Selenium*, including *webdriver.get()* and

webdriver.execute_script(), we can automatically scroll the web page to the bottom, and the web page will load all the required information. Then, *webdriver.page_source* will retrieve all the information we are assessing. It will return data type that can be processed by *BeautifulSoup*.

The method of web scraping using Selenium and *BeautifulSoup* can deal with all kinds of websites, including dynamic and static websites. However, we are not using Selenium for static websites because Requests has a much faster processing time compared to Selenium. This is because when using Selenium, we need to scroll all the way down to the bottom of a web page, which is time-consuming. This can be seen from our web scraping process in Shopee and Amazon. When we scrapped data from Amazon using Requests and *BeautifulSoup*, we only took 16 seconds to get all the information on five items, while getting the same amount of information using Selenium and *BeautifulSoup* on Shopee needs 67 seconds.

```
Processing https://shopee.sg/search?keyword=apple&page=0...
The time of execution of web scraping using selenium and beautifulsoup for 5 items is : 67 s
Processing https://www.amazon.sg/s?k=apple&page=1...
The time of execution of web scraping using requests and beautifulsoup for 5 items is : 16 s
```

Figure C.1 – Processing time when web scraping in different E-commerce platforms

Hence, when dealing with different kinds of websites, we should consider the use of different combinations of the library available so that we can retrieve data in a shorter time.