



**EASYDROP**  
SEND. EASY.

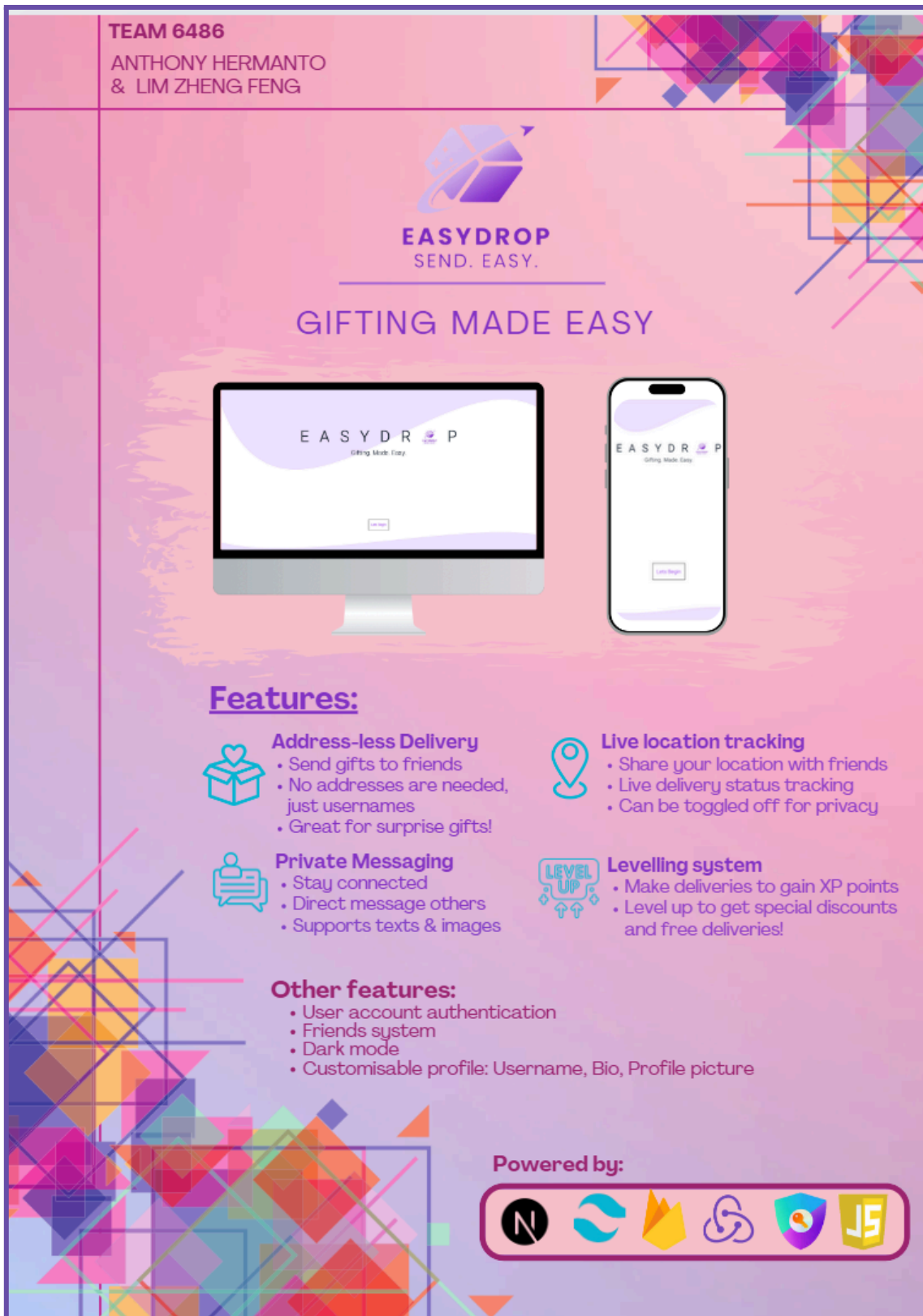
**National University of Singapore**  
**AY 23/24 CP2106 (Orbital)**  
**Proposed Level of Achievement:**  
**Apollo**

**ID: 6486 [EasyDrop]**  
**Anthony Hermanto, Lim Zheng Feng**  
**Advisor: Yi Ming**


# Table of Contents

<b>Cover page.....</b>	<b>1</b>
<b>Table of Contents.....</b>	<b>2</b>
<b>Posters.....</b>	<b>3</b>
<b>Motivation.....</b>	<b>5</b>
<b>Aim.....</b>	<b>6</b>
<b>Tech Stack.....</b>	<b>7</b>
1. Next.js with JavaScript.....	7
2. Choosing Firebase (NoSQL) Database.....	8
<b>User Stories.....</b>	<b>9</b>
<b>SWOT Analysis.....</b>	<b>11</b>
<b>Product Sample.....</b>	<b>15</b>
<b>Project Scope.....</b>	<b>16</b>
<b>Milestone 1.....</b>	<b>16</b>
1. User Registration.....	16
2. One-on-one messaging.....	27
3. Live Location Tracking.....	34
4. Profile page.....	38
<b>Timeline and Development Plan.....</b>	<b>50</b>
<b>Wireframe.....</b>	<b>53</b>
<b>System Design Diagram.....</b>	<b>54</b>
<b>Software Engineering Practices.....</b>	<b>55</b>
1. Two Week Sprints.....	56
2. Git Version Control.....	57
<b>Software Testing.....</b>	<b>60</b>
1. Unit Testing.....	61
2. Widget Testing.....	64
3. Integration Testing.....	68
4. System Testing.....	70
5. User Testing.....	75
<b>Miscellaneous.....</b>	<b>76</b>

# Posters




**Figure 1: Liftoff Poster**



**EASYDROP**  
SEND. EASY.

## GIFTING MADE EASY

By making deliveries easy,  
we strive to inject fun into  
spontaneous gifting and  
deepen the connections  
between friends and family.



### Address-less Delivery

Want to send a surprise gift to  
make someone's day? If they're  
on your friend's list, we'll make it  
happen!

### Private Messaging

Connect with your loved ones  
through texts and images!


### Level Up

Deliver gifts and earn xp to  
get exclusive discounts and  
free deliveries!

### Gifting leaderboard

A little fun and friendly  
competition!

Powered by:



Supports both mobile devices and computers



**TEAM 6486**  
**LIM ZHENG FENG**  
**ANTHONY HERMANTO**

**Figure 2: Milestone 1 Poster**

## Motivation

In today's fast-paced world, maintaining meaningful connections with friends can be challenging. We often find ourselves caught up in our hectic schedules, leaving little time to nurture the relationships that matter most. A [2021 survey](#) found only 13% of participants reported having 10 or more close friends, a significant drop from 33% over the last decade. Moreover, [research](#) has shown that adverse health outcomes including depression and anxiety are linked to loneliness and social isolation. To combat this, we wanted to create a project that would help people form stronger connections with their friends amidst the chaotic schedules of modern society.

An excellent way for people to connect is through spontaneous thoughtful gestures such as gifts or birthday surprises, which form meaningful memories that can truly make a person's day. These small acts of kindness are often cherished, forming lasting memories and deepening our connections with those we care about.

At EasyDrop, we believe that everyone deserves to experience these moments of joy and surprise, no matter where they are or how busy their lives may be. Hence, we decided to produce a web application to facilitate this which supports both phones and personal computers. On top of direct messaging features and the ability to add people as your friends, the key feature that sets us apart is our addressless deliveries. To send a gift, you simply need to add someone to your friends list.

Traditional delivery apps that require detailed address inputs and multiple steps deter spontaneous gestures. EasyDrop hence, empowers users to express care and appreciation without the usual hassle, fostering deeper connections and making it easier to show love and thoughtfulness. In the long term, we hope that EasyDrop can spark the rise of many meaningful friendships among us all.

In essence, EasyDrop is about more than just convenience—it's about reconnecting with friends, sharing joy, and making every gift a heartfelt gesture. It's about making the world a brighter place, one friendship at a time.

## Aim

Our primary goal with EasyDrop is to make it easy and delightful for users from all walks of life to surprise their friends and loved ones with gifts. By leveraging real-time location technology, we enable users to determine their friends' whereabouts and send them personalized surprises that are tailored to their current location. Whether it's a coffee delivery at the office, a bouquet of flowers at home, or a surprise dinner reservation, we aim to make every surprise uniquely special and perfectly timed.

To add an element of fun and engagement, we have incorporated a system of experience points and leaderboards. Users earn experience points for each successful gift they send, turning the act of giving into a rewarding and enjoyable experience. Our leaderboard system fosters a sense of community and friendly competition, highlighting those who are most active in spreading joy.

At the end of each month, we celebrate our top 25 users on the leaderboard by gifting them with vouchers and exclusive rewards. This not only acknowledges their efforts but also encourages continuous participation and engagement with our platform.

We are embarking on this project because we believe in the importance of fostering genuine and strong relationships in a world where many are overwhelmed with their daily routines. EasyDrop aims to revolutionise the way people connect with others and become a catalyst in the spread of happiness among us all. Our mission is to make it easier for people to show their appreciation and love for one another, to create a ripple effect of positivity and warmth.

# Tech Stack

## 1. FrontEnd: Next.js with JavaScript



Next.js with JavaScript was used instead of typescript for various reasons mentioned below:

### 1. Ease of Use and Familiarity:

JavaScript has a simpler learning curve compared to TypeScript. If your development team is more familiar with JavaScript, this can lead to faster development and fewer roadblocks. JavaScript allows for quicker prototyping and iteration. TypeScript, while powerful, requires additional setup and a deeper understanding of type annotations and type system intricacies.

### 2. Flexibility:

JavaScript's dynamic typing can be more flexible, allowing developers to write code without worrying about strict type constraints. This can be advantageous in the early stages of a project when requirements are still evolving. Without the need for type definitions, JavaScript can result in less boilerplate code, potentially making the codebase simpler and more concise.

### 3. Community and Ecosystem:

Next.js is highly popular in the JavaScript community, and there are extensive resources, libraries, and community support available for JavaScript. This can make it easier to find solutions and integrate with other tools and libraries. JavaScript has a vast array of development tools, plugins, and extensions that can enhance productivity without the additional setup required for TypeScript tooling.

## 2. Backend: Firebase (NoSQL) Database



We decided to use Firebase, a NoSQL Database for the reasons mentioned below:

### 1. **Scalability and Performance:**

NoSQL databases like Firebase are designed to scale horizontally, making them well-suited for applications with large amounts of unstructured or semi-structured data, such as user-generated content, social media posts, or IoT data. NoSQL databases can handle high-velocity data writes and reads efficiently, providing low-latency responses which are crucial for real-time applications.

### 2. **Flexibility and Development Speed:**

Firebase's NoSQL structure allows for a flexible schema, meaning you don't have to predefine the structure of your data. This can speed up development, as you can easily modify the data model without needing to migrate the database schema. Firebase offers real-time data synchronisation, which is beneficial for applications that require live updates, such as chat applications, collaborative tools, or live feeds.

### 3. **Integration and Ecosystem:**

Firebase provides built-in authentication, hosting, and other backend services, reducing the need to integrate multiple third-party services. This unified ecosystem can streamline development and maintenance. Firebase's serverless approach allows you to focus on writing front-end code without managing backend servers. This can lead to faster deployment and reduced operational overhead.



# User Stories

## 1. As a busy professional

**User Story:** “As a busy professional, I want to send personalized birthday gifts to my friends and family without having to remember dates or manually arrange deliveries, so that I can maintain my relationships even with a hectic schedule.”

**Benefit:**

EasyDrop's automated reminders and location-based gift delivery ensure that I never miss an important occasion and can effortlessly send thoughtful gifts on time.

## 2. As a frequent traveler

**User Story:**

“As a frequent traveler, I want to send surprise gifts to my loved ones based on their real-time location, so that I can make them feel special even when I am far away.”

**Benefit:**

EasyDrop's life location tracking allows me to deliver gifts directly to where my loved ones are, making my presence felt despite the distance.

## 3. As a parent

**User Story:**

“As a parent, I want to send surprise gifts to my children at school or their friend's house, so that I can brighten their day unexpectedly.”

**Benefit:**

EasyDrop's ability to send gifts based on real-time locations ensures that my children receive surprises at the perfect moments, enhancing their special days and making me feel connected to them.

## 4. As a friend planning a group surprise

**User Story:**

“As a friend, I want to coordinate with others to send a surprise gift to a mutual friend based on their location during a special occasion, so that we can collectively make their day memorable.”

**Benefit:**

EasyDrop's group coordination features and location-based delivery help us organize a seamless and impactful surprise, making the celebration more exciting and memorable.

**5. As a long-distance partner**

**User Story:**

"As a long-distance partner, I want to send spontaneous gifts to my significant other based on their daily routines, so that I can show my affection and stay connected despite the physical distance."

**Benefit:**

EasyDrop's real-time tracking and personalized delivery options allows us to send gifts that align with my partner's activities, enhancing our relationship by creating unexpected joyful moments.

**6. As an event planner**

**User Story:**

"As an event planner, I want to send thank-you gifts to attendees based on their departure locations after an event, so that I can provide a lasting positive impression."

**Benefit:**

EasyDrop's location-based delivery ensures that attendees receive their gifts promptly and at convenient times, reinforcing their positive experience with the event.

**7. As a tech-savvy user**

**User Story:**

"As a tech-savvy user, I want to utilize advanced location tracking to send gifts to my friends in unique and creative ways, so that I can stand out and impress them with my thoughtfulness and use of technology."

**Benefit:**

EasyDrop's innovative use of real-time location data allows me to personalize gift deliveries in a way that is both modern and impactful, enhancing my reputation for being thoughtful and tech-savvy.

# SWOT Analysis

## Strengths

### 1. Unique Value Proposition

EasyDrop stands out with its real-time location tracking, which allows users to send gifts precisely based on the recipient's current location. This feature adds a layer of personalization and surprise that is hard to match, ensuring that gifts are received at the perfect moment, enhancing the recipient's experience. Additionally, the automated reminders feature is a significant advantage, as it helps users remember important dates without manual effort, ensuring they never miss an occasion. This automation reduces the cognitive load on users and makes the process of sending gifts seamless and stress-free.

### 2. User Convenience

The ease of use is a core strength of EasyDrop. Its intuitive interface simplifies the process of selecting and sending gifts, making it accessible to users of all technical skill levels. By integrating with user calendars and contact lists, EasyDrop automatically identifies upcoming occasions and important dates, further simplifying the user experience. This integration reduces the steps needed to send a gift, making the process almost effortless.

### 3. Innovative Features

Group coordination tools allow friends and family to plan and execute collective surprises, enhancing social interactions and making the gifting experience more enjoyable. These tools facilitate communication and planning, ensuring that everyone involved can contribute to the surprise. Customization options allow users to choose from a wide range of gifts and personalize messages, making each gift unique and thoughtful, which enhances the emotional impact of the gesture.

## **Weaknesses**

### **1. Dependence on Technology**

EasyDrop's reliance on GPS and location services means that the accuracy of deliveries is contingent on the precision of these technologies. In areas with poor GPS coverage or in situations where the recipient's location is not accurately tracked, delivery issues can arise, leading to potential dissatisfaction. Additionally, continuous location tracking can lead to higher battery consumption on users' devices, which may deter some users from enabling this feature.

### **2. Privacy Concerns:**

Handling sensitive location data and personal information requires stringent security measures to prevent data breaches and unauthorized access. Ensuring the security of this data is complex and can be costly to implement and maintain. Moreover, convincing users to grant location access and share personal information can be challenging, as privacy concerns are increasingly prevalent. Building and maintaining user trust is critical and requires transparent communication about data usage and robust security practices.

### **3. Market Awareness**

As a new entrant, EasyDrop faces the challenge of building brand recognition in a competitive market. Significant marketing efforts are required to establish the brand and convey its unique value proposition to potential users. Additionally, initial user acquisition and retention may be slow as users might be hesitant to try a new service, especially one that requires sharing personal data.

### **4. Operational Challenges**

Ensuring timely and accurate delivery in various regions can be logistically challenging, particularly in areas with less developed infrastructure. Managing a wide network of delivery services and ensuring consistent service quality is crucial but difficult. Dependence on third-party vendors for gifts and delivery services can lead to inconsistencies in quality and reliability, impacting the overall user experience.

## Opportunities

### 1. Market Expansion

Expanding services to international markets can significantly increase EasyDrop's user base and revenue potential. By adapting to different cultural preferences and logistical environments, EasyDrop can tap into a global market. Additionally, partnering with businesses for corporate gifting solutions can open new revenue streams, as companies often seek reliable and innovative ways to send gifts to employees and clients.

### 2. Technological Advancements

Leveraging AI and machine learning for better location predictions and personalised gift recommendations can enhance the user experience. These technologies can analyse user behaviour and preferences to suggest the most appropriate gifts, making the process even more personalised and efficient. Implementing advanced encryption and security measures can address privacy concerns and build user trust, ensuring that user data is protected and compliant with regulations.

### 3. Strategic Alliances:

Collaborating with popular retail brands for exclusive gift options can differentiate EasyDrop from competitors and attract brand-conscious users. Such partnerships can also enhance the quality and variety of gifts available. Leveraging social media influencers to promote the app can increase visibility and user engagement, as influencers can effectively communicate the app's benefits to their followers.

## Threats

### 1. Competition

Competing against well-established gifting and delivery services can be challenging, particularly in terms of pricing and brand loyalty. These established players have significant market share and resources, making it difficult for a new entrant to gain traction. Additionally, the market for innovative gifting solutions is growing, and new entrants with similar ideas can pose a threat, potentially saturating the market.

### 2. Regulatory Issues

Compliance with varying data privacy laws across different regions can be complex and costly. Regulations such as the GDPR in Europe and CCPA in California require stringent data protection measures and can impose significant fines for non-compliance. Ensuring compliance with consumer protection regulations related to gift deliveries and services is essential to avoid legal issues and maintain user trust.

### 3. Technological Risks

The risk of data breaches and cyber-attacks can compromise user data and damage EasyDrop's reputation. Implementing robust cybersecurity measures is critical to protect sensitive information and maintain user trust. Additionally, technical issues or service downtime can affect user trust and lead to a loss of business. Ensuring high availability and reliability of the app is crucial for maintaining user satisfaction.

### 4. Economic Factors

In times of economic hardship, discretionary spending on gifts may decrease, impacting EasyDrop's revenue. Economic downturns can lead to reduced consumer spending, particularly on non-essential items like gifts. Additionally, increasing costs of delivery services, partnerships, and technology can affect profitability. Managing costs effectively while maintaining service quality is essential for long-term sustainability.

## Product Sample

Our product is a web application that works on both mobile devices and computers. Scan the QR code provided to access the web application or use the link provided below:



[Product Link](#)

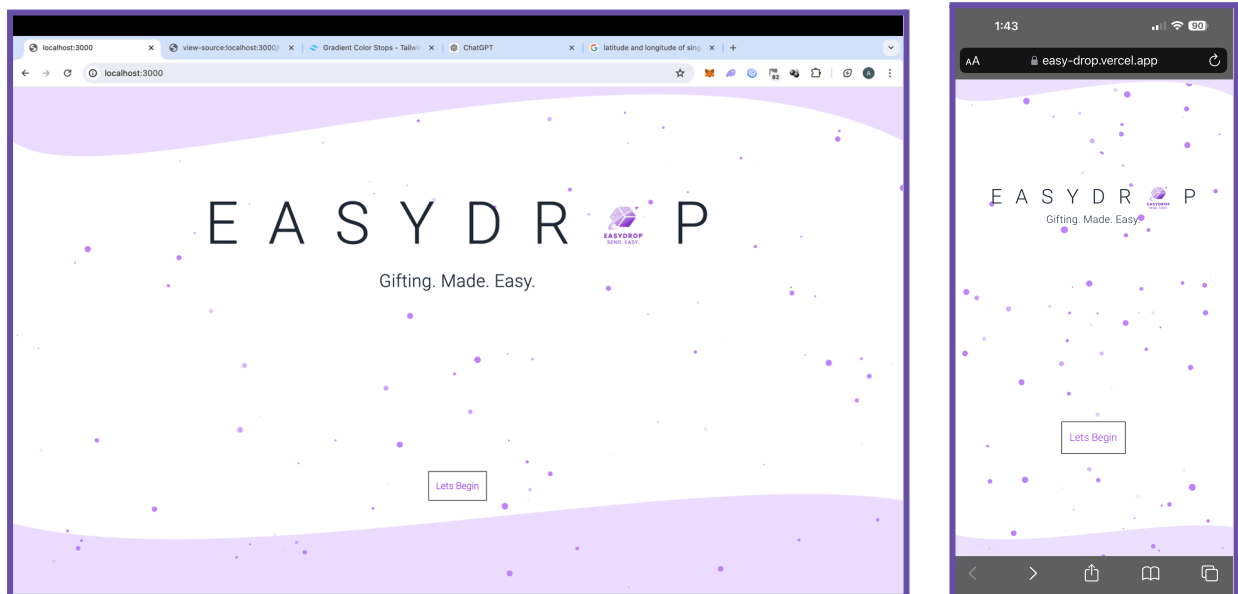
# Project Scope

## Milestone 1:

- User Registration (With Google Authentication)
- Basic one-to-one messaging
- Live tracking
- Profile page

## 1. User Registration

### Front Page



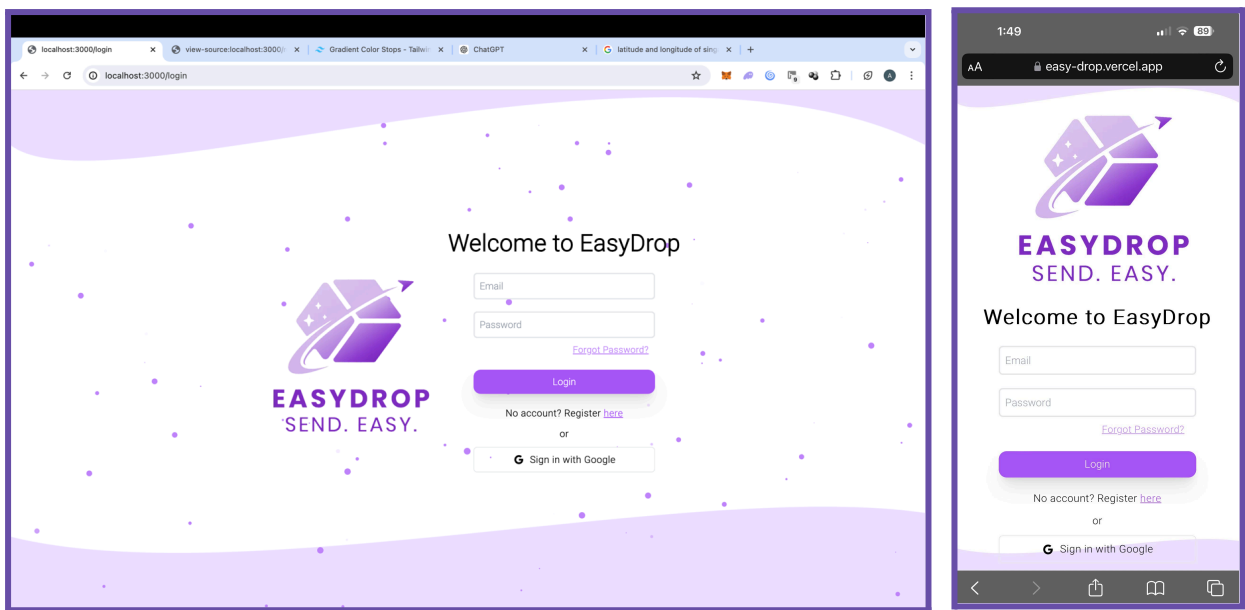
**Figure 3: Front Pages of EasyDrop**

### **Description:**

Upon entering our website, users will be greeted with this home screen. Clicking on “Let's Begin” will direct them to the login screen as shown in *Figure 4* on the next page.



## Login Page

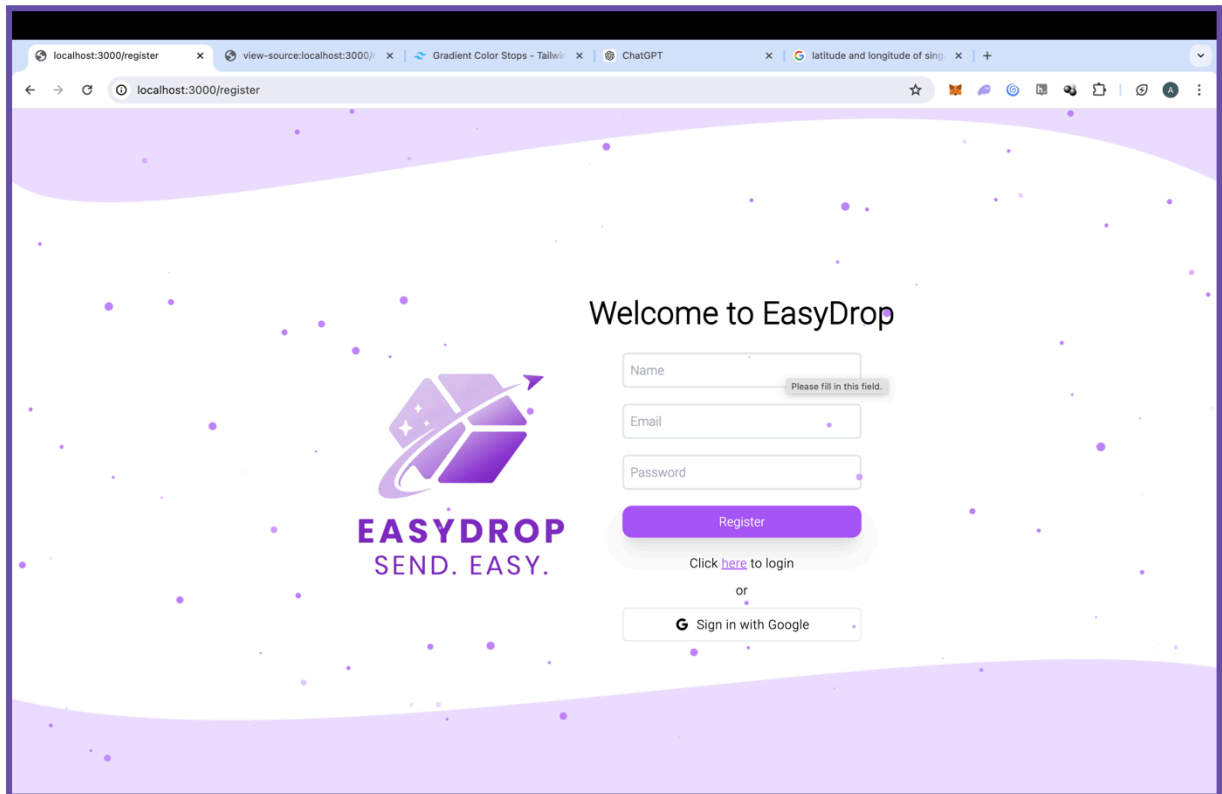


**Figure 4: Login Pages of EasyDrop**

### **Description:**

If the user does not have an account, they can proceed with registering an account by clicking the underlined word “here” in purple in the words “Register here” under the Login button. This would direct them to the register page shown in *Figure 5* on the next page.

## Registration Page

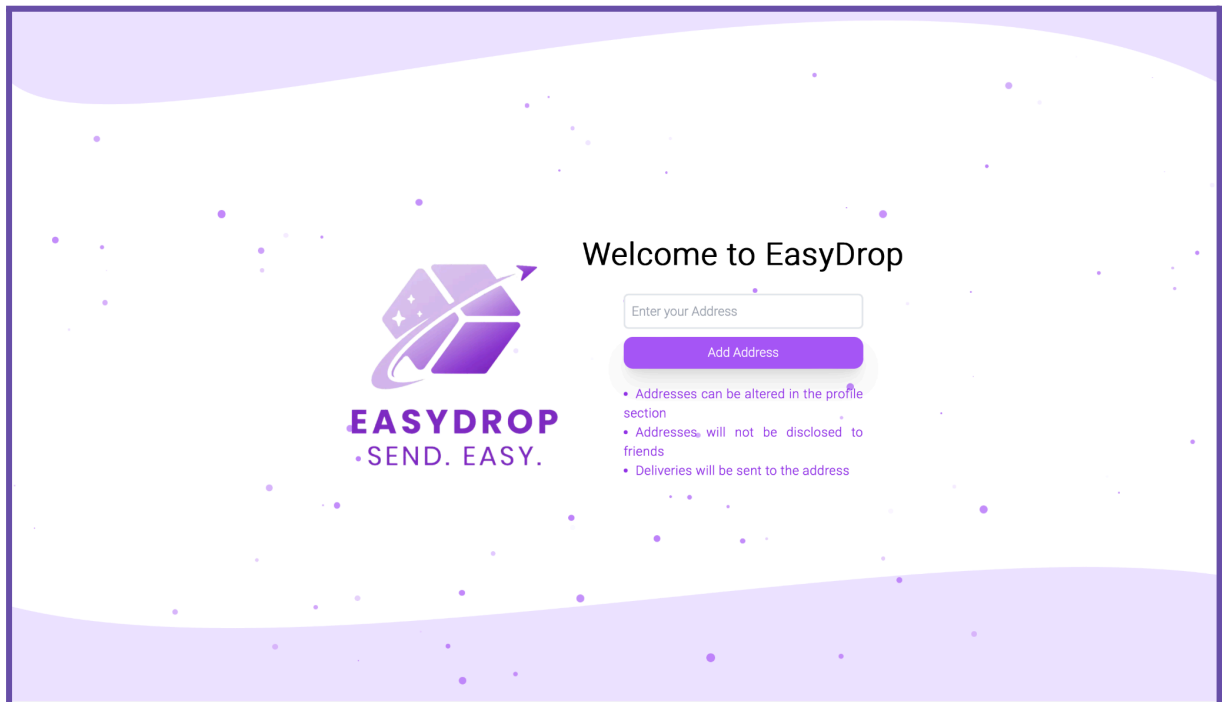


**Figure 5: Register Screen for EasyDrop**

### **Description:**

Visitors of our website can now register using their email of password OR with their Google account. Upon registering, they will be added to the firebase database under the collection “users”.

## Home Address Page



**Figure 6: Home Address Page for EasyDrop**

### **Description:**

Upon registering an account, the user will be prompted to enter their home address as seen in *Figure 6* above. Deliveries will be made to this address which can be edited later in the profile section if necessary. The address will be stored in the database and will **not be visible** to other users. For other users to send gifts to you, they simply need your username.

## Database During Registration

<div> <div> <div>🏠</div> <div>&gt; users &gt; 110388778151.</div> </div> <div>More in Google Cloud</div> </div>		
<div>(default)</div> <div> <div>+ Start collection</div> <div> <div>deliveries</div> <div>friends</div> <div>messages</div> <div>users &gt;</div> </div> </div>	<div>users</div> <div> <div>+ Add document</div> <div> <div>102157593831809076220</div> <div>11038877815143523729 &gt;</div> <div>111263234363959956760</div> <div>111996743892022341391</div> <div>8RQY8M33kvQ5zLyNA75mW00Hk2</div> <div>B3IPUsqPLsgW0I7pFREUYFFer1q2</div> <div>C39kmu6saDaQ4bjajFB9ItoNAMQ2</div> <div>HOzw1TkWMreTETrx88L2DcG19xi1</div> <div>JtNVbJl3LrOcEFkkkXo613tQ8xY2</div> <div>NqaAqHXmMuP8I0atJZ2hvhHoNuIo1</div> <div>SfMSZnNompRxpmdSsLG8AHBLt02</div> <div>Tnv8UIUR8cRJFYld78xIOsrDjRl1</div> <div>a9u2Sc3pjdQy3CeRKF2Ae3BHaXo1</div> <div>cNsLFDU91ShowtsLad1pa2sYQdR2</div> </div> </div>	<div>11038877815143523729</div> <div> <div>+ Start collection</div> <div> <div>+ Add field</div> <div> <div>bio: "yay"</div> <div>email: "bobbyleeswiswi@gmail.com"</div> <div>id: "11038877815143523729"</div> <div>img: "https://lh3.googleusercontent.com/a/ACg8ocKKiGiOdCIS3m8wuTOFARGnoIFx5JJUe2DFVag=s96-c"</div> <div>location <div> <div>latitude: 1.3726340268373096</div> <div>longitude: 103.87073380207114</div> </div> </div> <div>main: "Bobby Lee"</div> <div>name: "bob"</div> <div>xp: 2</div> </div> </div> </div>

**Figure 7a: Firebase database on “users” collection (Google)**

<div> <div> <div>🏠</div> <div>&gt; users &gt; HOzw1TkWMreT.</div> </div> <div>More in Google Cloud</div> </div>		
<div>(default)</div> <div> <div>+ Start collection</div> <div> <div>deliveries</div> <div>friends</div> <div>messages</div> <div>users &gt;</div> </div> </div>	<div>users</div> <div> <div>+ Add document</div> <div> <div>102157593831809076220</div> <div>11038877815143523729</div> <div>111263234363959956760</div> <div>111996743892022341391</div> <div>8RQY8M33kvQ5zLyNA75mW00Hk2</div> <div>B3IPUsqPLsgW0I7pFREUYFFer1q2</div> <div>C39kmu6saDaQ4bjajFB9ItoNAMQ2</div> <div>HOzw1TkWMreTETrx88L2DcG19xi1 &gt;</div> <div>JtNVbJl3LrOcEFkkkXo613tQ8xY2</div> <div>NqaAqHXmMuP8I0atJZ2hvhHoNuIo1</div> <div>SfMSZnNompRxpmdSsLG8AHBLt02</div> <div>Tnv8UIUR8cRJFYld78xIOsrDjRl1</div> <div>a9u2Sc3pjdQy3CeRKF2Ae3BHaXo1</div> <div>cNsLFDU91ShowtsLad1pa2sYQdR2</div> </div> </div>	<div>HOzw1TkWMreTETrx88L2DcG19xi1</div> <div> <div>+ Start collection</div> <div> <div>+ Add field</div> <div> <div>bio: ""</div> <div>email: "e1136989@u.nus.edu"</div> <div>id: "HOzw1TkWMreTETrx88L2DcG19xi1"</div> <div>img: ""</div> <div>location <div> <div>latitude: 1.3319409</div> <div>longitude: 103.8660025</div> </div> </div> <div>main: "Lim Zheng Feng"</div> <div>name: "Lim Zheng Feng" (string) ✎ 🗑</div> <div>xp: 0</div> </div> </div> </div>

**Figure 7b: Firebase database on “users” collection (Email)**

**Description:**

Based on *Figure 7*, the **top** image (*Figure 7a*) displays the details that will be stored when the users sign in with their Google Account. In the **bottom** image (*Figure 7b*), it displays the details that will be stored when the users signs in with email and password.

The user information contains the following fields:

1. **Bio**: Users can modify their bio in this page
2. **Email**: the email that is used to register for an account
3. **Id**: the id of each user
4. **Img**: the profile image of the user (only for google accounts)
5. **Location**: the location of the user based on latitude and longitude. The live location tracking feature is implemented such that when the users logs in to our app, their live location will be updated immediately based on where they are
6. **Main**: the user's main name, i.e. their name when they register a new account in our website
7. **Name**: the name of the user. This can be changed in the profile page that has yet to be implemented.

## Authentication Logic

```
const handleRegister = async (event) => {
  event.preventDefault();
  for (let i = 0; i < users.length; i++) {
    if (users[i] == name) {
      alert("Name Taken");
      return;
    }
  }
  const { user } = await createUserWithEmailAndPassword(
    auth,
    email,
    password
  );

  await updateProfile(user, {
    displayName: name,
  });
  dispatch(
    login({
      name: user.displayName,
      email: user.email,
      uid: user.uid,
      img: "",
    })
  );
  router.push("/");
  const userRef = doc(db, "users", user.uid);
  const userDoc = await getDoc(userRef);
```

```
if (!userDoc.exists()) {
  // User does not exist, add them to the database
  await setDoc(userRef, {
    main: user.displayName,
    name: user.displayName,
    email: user.email,
    id: user.uid,
    img: "",
    location: {},
    xp: 0,
    bio: ""
  });
}
};
```

**Figure 8: Code snippet for authentication**

**Description:**

Upon registering, users are redirected back to the front page of the website and the dispatch function is called to store the user's state globally, this is done through the use of Redux, a global state management system. This page includes functionality to listen for authentication state changes to determine if the user is logged in. This is achieved through the use of a `useEffect` hook, a powerful feature of React that allows you to perform side effects in function components.

The `useEffect` hook is set up to monitor authentication changes continuously. It subscribes to an authentication listener provided by the authentication service, in this case Firebase Auth. This listener is called whenever the authentication state changes, such as when a user logs in or logs out.

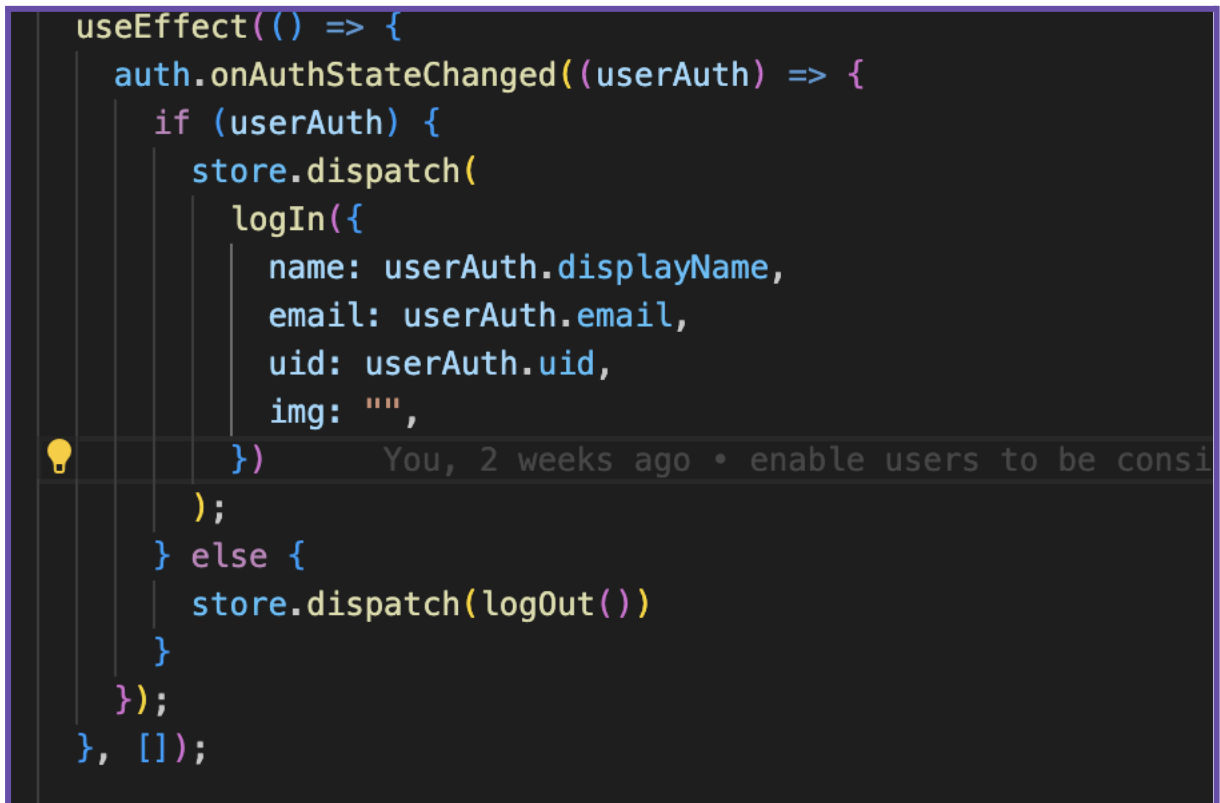
When a user successfully registers, the authentication state is updated, triggering the listener. The `useEffect` hook captures this change and updates the application's state accordingly. This ensures that the user experience is seamless and dynamic, reflecting the current authentication status without requiring manual page refreshes or additional actions from the user.

Moreover, this setup supports persistent login. When a user logs in, their session information is stored in a secure manner, typically using cookies or local storage, depending on the security protocols in place. As a result, the user remains logged in even after closing the browser or refreshing the page, providing a consistent and uninterrupted experience.

The user remains logged in until they explicitly choose to log out by pressing the logout button. Upon pressing this button, the authentication state is updated to reflect that the user is no longer authenticated. This change is again captured by the `useEffect` hook, which then updates the application's state to log the user out, ensuring that all user data and access are appropriately restricted.

### useEffect Hook

```
useEffect(() => {  
  auth.onAuthStateChanged((userAuth) => {  
    if (userAuth) {  
      store.dispatch(  
        login({  
          name: userAuth.displayName,  
          email: userAuth.email,  
          uid: userAuth.uid,  
          img: "",  
        })  
      );  
    } else {  
      store.dispatch(logOut())  
    }  
  });  
}, []);
```

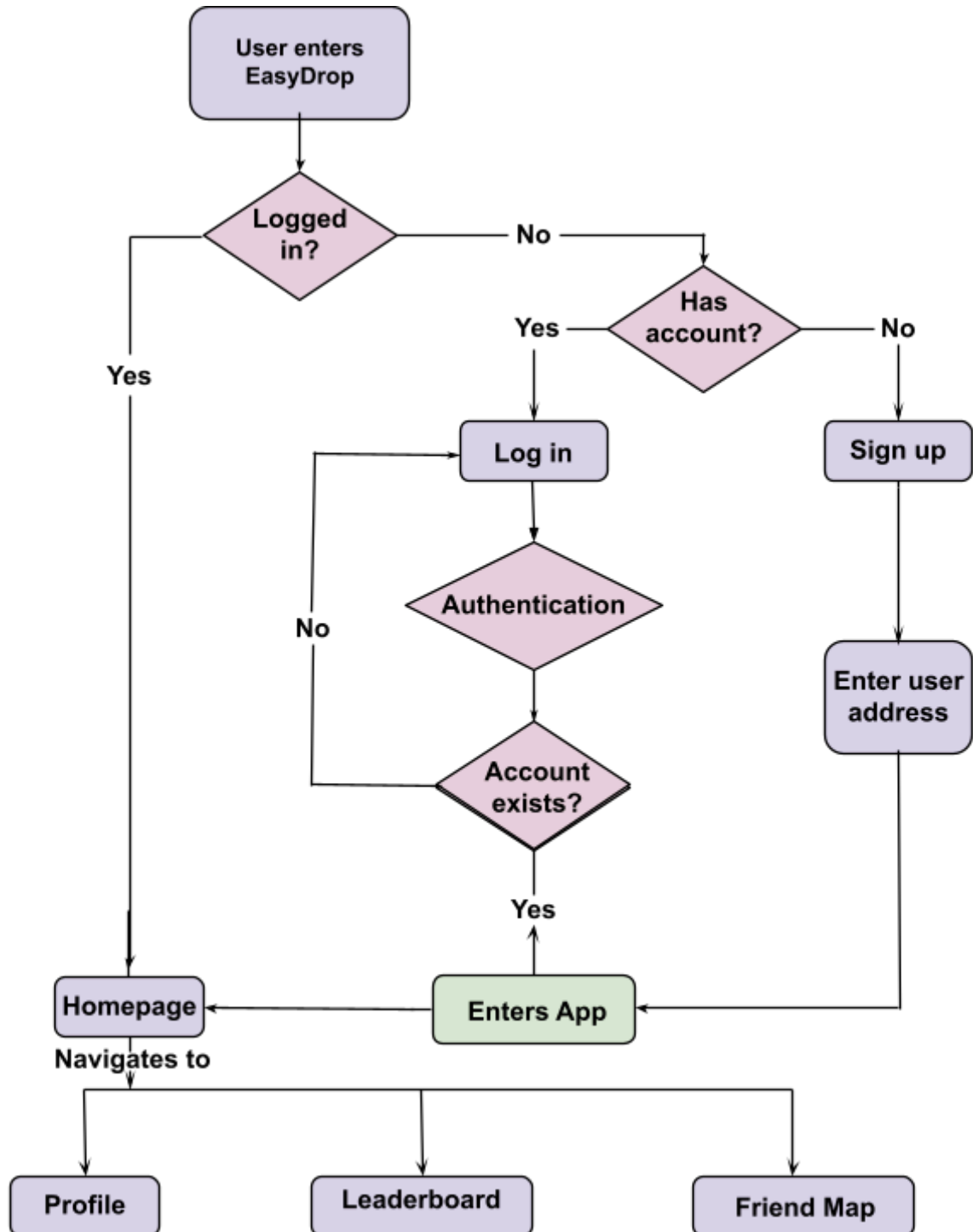


**Figure 9: useEffect hook to listen for auth changes**

#### **Description:**

In the code above, if there is a change in auth, this means that the user is successfully authenticated. In the case the user is authenticated, it will dispatch a login function from redux and logs in the user.



Authentication Flow**Figure 10: Authentication flow diagram**

### **Implementation Challenges:**

Working with the Firebase API for handling multiple forms of authentication and enabling persistent login using Redux presents several challenges that require careful consideration and implementation.

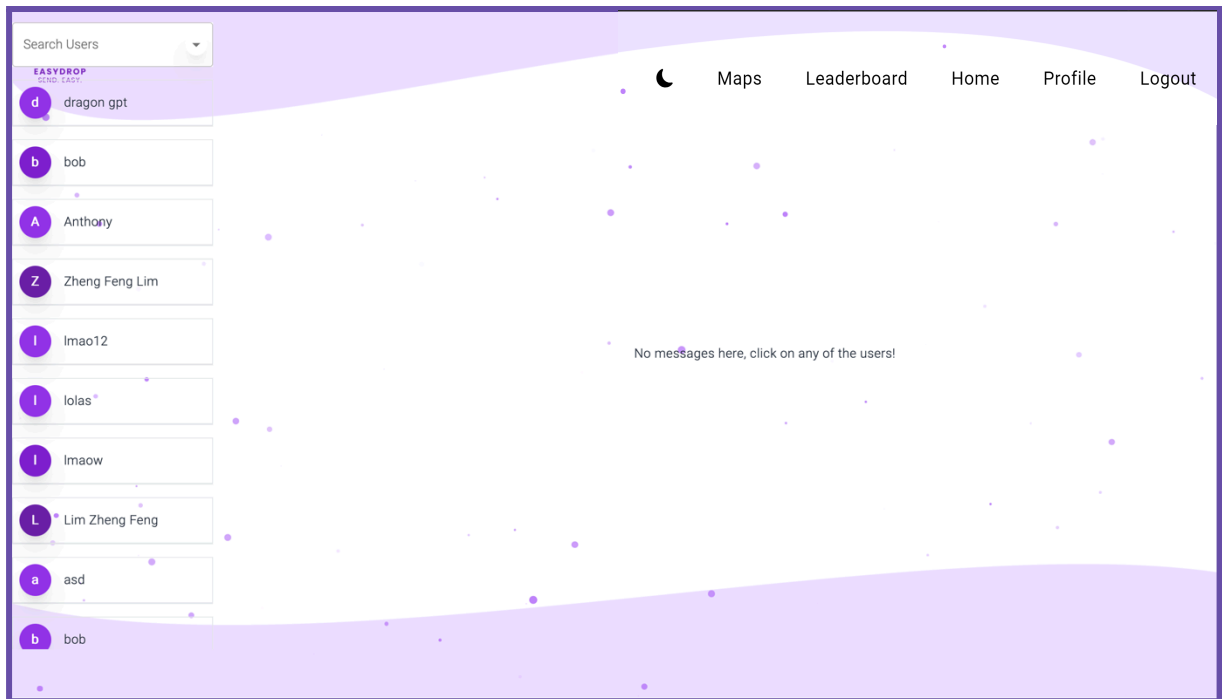
Integrating various authentication methods such as email/password and Google involves dealing with the complexities of each SDK. Each method has its unique flow for sign-in, error handling, and user data retrieval, which can complicate the development process. Ensuring a seamless and consistent user experience across these different authentication methods is particularly challenging, as each method may require different user interface elements and error messages that need to be unified for a cohesive user experience.

Enabling persistent login using Redux adds another layer of complexity. Managing a global state for user authentication across the entire application requires proper configuration of Redux to maintain and update the authentication state.

By addressing these challenges methodically, developers can create a robust authentication system using Firebase and Redux. This approach provides a secure and seamless experience for users, ensuring that the application remains functional and user-friendly despite the inherent complexities of managing multiple authentication methods and persistent login states.

## 2. One-to-one Messaging

### Home Page



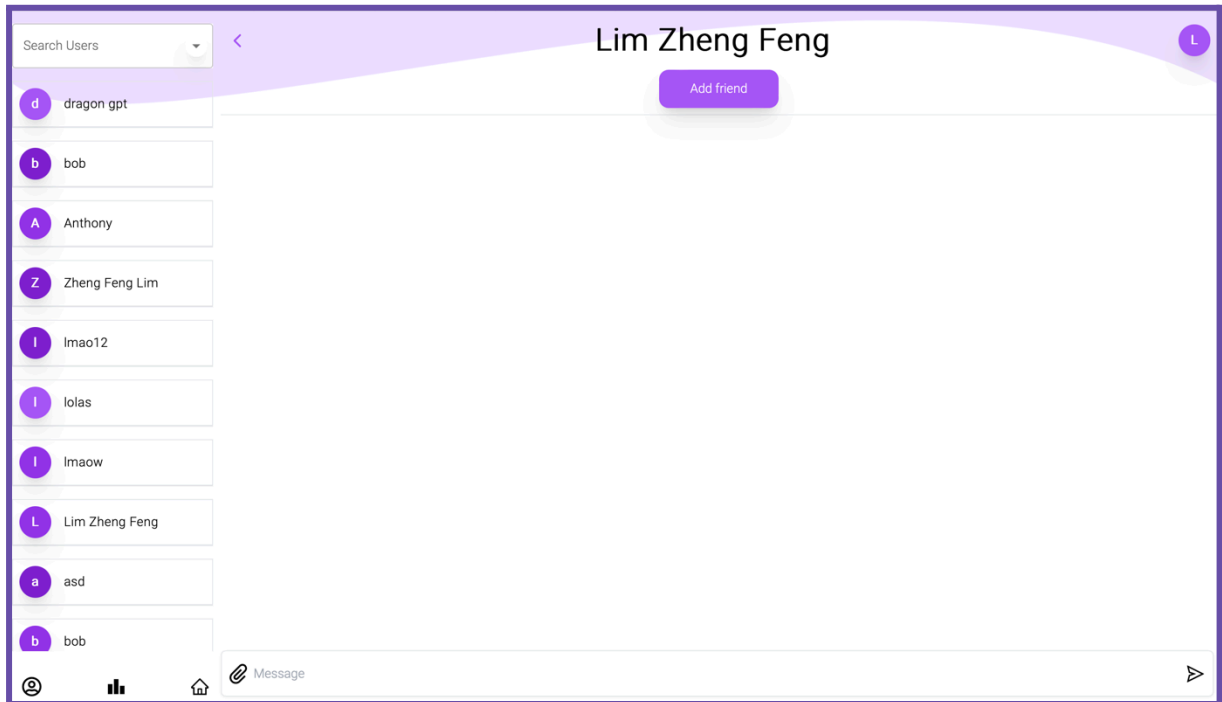
**Figure 11: Home screen with all the users**

#### **Description:**

Upon successfully entering the web app, the users will be greeted with the home screen where all the users of the application will be listed as shown in *Figure 11* above.

They can click on any of the users to text them, in this case let's say the user wants to text another user named "Zheng Feng Lim". The user can click on "Zheng Feng Lim" to begin texting him. If the user wishes to search up another user's name, there is a "Search Users" bar at the top left corner for convenience.

## Direct Message Page

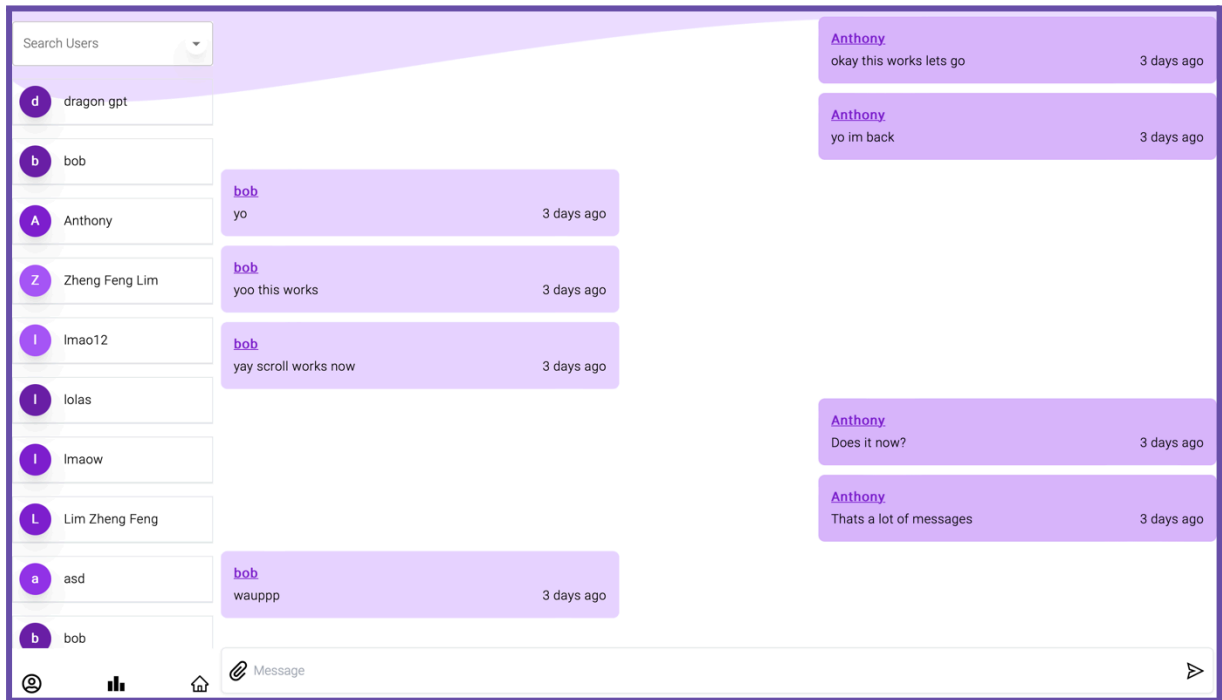


**Figure 12: Message Screen**

### **Description:**

Upon clicking on “Zheng Feng Lim”, the user is brought to the Direct Message page. The message bar is at bottom of the page and the user is now able to send Zheng Feng Lim any text message as per the usual messaging feature. Pictures can also be uploaded by the user for sending.

## Sample



**Figure 13: Sample Messages**

### **Description:**

When a text message is sent to another user, they will receive the messages and be able in reply in real time. The time since the message was sent will also be displayed on the bottom right corner of a message.

### **Flow Summary:**

User is logged in → Directed to Home Page → All users displayed → Click on the user → Message the user

## One-to-one Messaging Logic

### Description:

When the user sends a message, all the messages would be stored in the firebase database under the “messages” collection. In this case, let’s say the user is sending the message to “bob”, the following fields would be added to the “messages” collection:

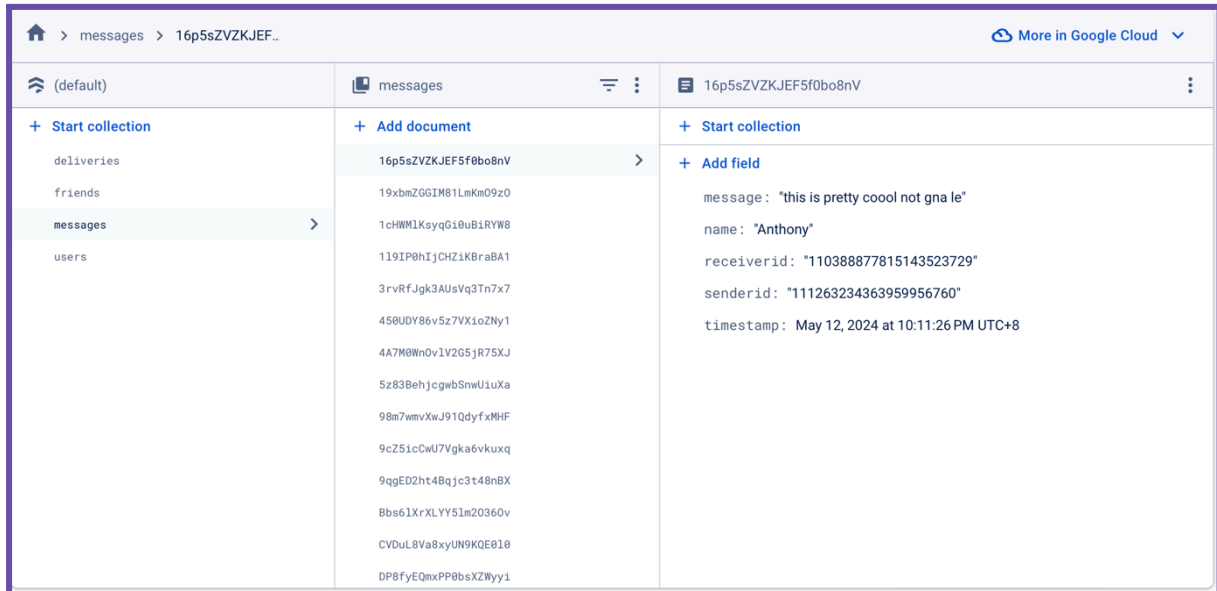
1. **SenderId:** sender id of the user
2. **Receiverid:** id of the receiver
3. **Message:** message you want to send to the receiver
4. **Timestamp:** the current time
5. **Name:** sender’s name

The code in *Figure 14* below shows how it is implemented:



```
<AiOutlineSend className="absolute right-[10px] w-[25px] h-[25px] text-black cursor-pointer" onClick = {async (e) => {
  setMessage("");
  e.preventDefault();
  if (!message) {
    return;
  }
  if (user) {
    await addDoc(collection(db, "messages"), {
      senderid: user.uid,
      receiverid: Id,
      message: message,
      timestamp: serverTimestamp(),
      name: curUser[0].data().name,
    });
  } else if (session) {
    await addDoc(collection(db, "messages"), {
      senderid: session.user.uid,
      receiverid: Id,
      message: message,
      timestamp: serverTimestamp(),
      name: curUser[0].data().name,
    });
  }
}
}/>
```

**Figure 14: Adding messages to the “messages” collection**



**Figure 15: Adding to the firebase database**

### **Description:**

The **sender id** is the user's uid, the token that is automatically generated when the user logs into the application.

The **receiver id** is retrieved from the router query parameters

```
const { Id } = router.query;
```

For example,

`http://localhost:3000/users/110388877815143523729?name=bob+`

In this case, the receiver id would be 110388877815143523729

**Description:**

We filter the messages according to whether the sender id matches the current user's uid and the receiver id matches the current router query parameter OR the sender id matches the current router.query and the receiver id matches the user's uid

```
const filter_messages = messages.filter((message) => {  
  if (user) {  
    return (  
      (message.data()?.senderid === user.uid &&  
        message.data()?.receiverid === Id) ||  
      (message.data()?.senderid === Id &&  
        message.data()?.receiverid === user.uid)  
    );  
  } else if (session) {  
    return (  
      (message.data()?.senderid === session.user?.uid &&  
        message.data()?.receiverid === Id) ||  
      (message.data()?.receiverid === session?.user.uid &&  
        message.data()?.senderid === Id)  
    );  
  }  
});
```

**Figure 16: Filtering the “messages” collection**



**Implementation Challenges:**

One of the key challenges in developing the messaging feature was the logic surrounding how messages were routed and displayed. The initial implementation only considered a single scenario: where the sender's ID matches the current user's UID and the receiver's ID matches the current router query parameter. This approach worked well for one direction of the conversation but failed to account for the reverse scenario.

In a typical messaging application, communication is bidirectional. This means that messages can be sent and received by both parties involved in the conversation. Initially, the logic was set up to display messages only when the current user was the sender. Specifically, the sender ID needed to match the user's UID, and the receiver ID needed to match the router query parameter. However, this approach overlooked the fact that the current user could also be the receiver of messages.

To address this issue, the logic had to be expanded to include both possible scenarios. The messaging feature needed to consider that the sender ID could match the current router query parameter (indicating that the other person in the conversation is the sender), and the receiver ID could match the user's UID (indicating that the current user is the receiver). This bidirectional consideration ensures that messages are displayed correctly regardless of who sent them.

### 3. Live Location Tracking

**Description:**

To integrate the Google Maps API for determining the live location of users, we began by setting up the necessary configurations in the Google Cloud Console. First, we created a new project to house all the API services required for our application. This involved enabling billing, which, while offering a free tier, is essential to ensure uninterrupted API usage. We then enabled the relevant APIs, including the Maps JavaScript API, Geolocation API, and Places API, to cover all necessary functionalities.

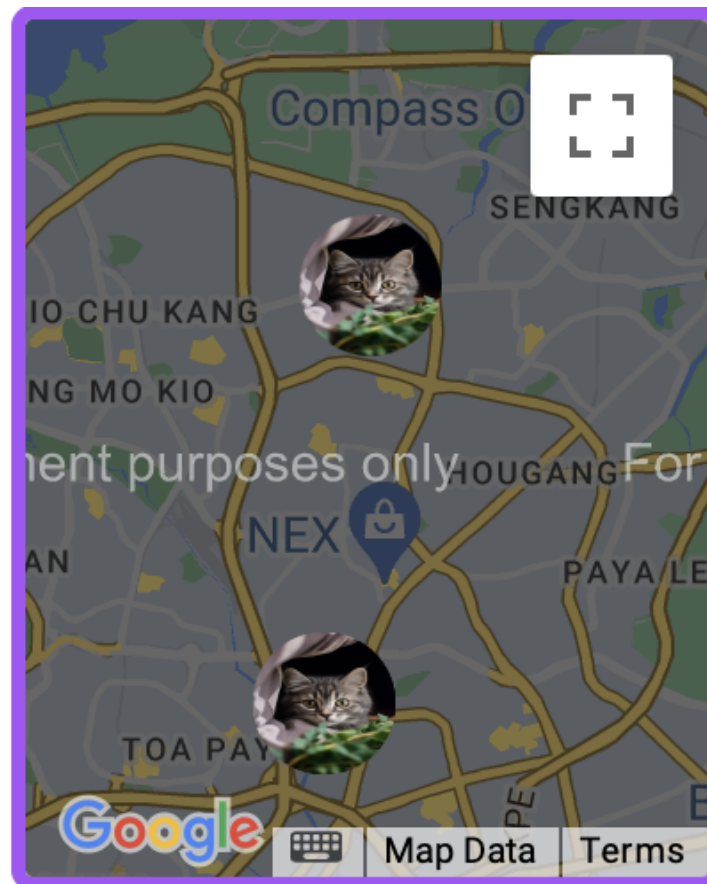
Next, we generated an API key from the Credentials section in the Cloud Console. This key is vital for authenticating our API requests. For security purposes, we restricted the API key to specific referrers or IP addresses and enabled only the required APIs, minimizing the risk of unauthorized access and usage.

With the API key in hand, we moved on to integrating the Google Maps API into our project. We included the Maps JavaScript API script in our Js file and created a function to initialize and display the map on our webpage. This setup provided a foundational map centered at a default location.

## Location Tracking Logic

### **Description:**

To determine and display the user's live location, we utilized the Geolocation API. This involved requesting the user's permission to access their location. Upon receiving permission, the Geolocation API fetched the user's current position, which we then used to update the map's center and place a marker indicating the user's location as shown in *Figure 17*. We also implemented error handling to alert users if the geolocation service failed or if their browser did not support geolocation.



**Figure 17: Markers placed for each user**

**Description:**

For real-time location updates, we connected our project to Firebase. After initializing Firebase and setting up Firestore or the Realtime Database, we continuously updated the user's location in the database. We set up listeners to track changes in the database as shown in *Figure 18*, ensuring that any movement was reflected on the map in real-time. This involved setting the marker's position to the updated coordinates whenever the user's location changed in the database.

```
useEffect(() => {
  const updateUserLocation = async () => {
    if (curUser[0]) {
      const userRef = doc(db, "users", curUser[0].data().id);
      try {
        if (navigator.geolocation) {
          navigator.geolocation.getCurrentPosition(async (position) => {
            const { latitude, longitude } = position.coords;
            await setDoc(userRef, {
              main: curUser[0]?.data().main,
              name: curUser[0]?.data().name,
              email: curUser[0]?.data().email,
              id: curUser[0]?.data().id,
              img: curUser[0]?.data().img,
              location: { latitude, longitude },
              bio: curUser[0]?.data().bio,
              xp: curUser[0].data().xp
            });
          });
        } else {
          // Geolocation not supported
          console.log("Geolocation is not supported");
        }
      } catch (error) {
        console.error("Error updating user location:", error);
      }
    }
  };

  updateUserLocation();
  const intervalId = setInterval(updateUserLocation, 3*60*1000);

  // Clean up the interval on component unmount or when session/user changes
  return () => {
    clearInterval(intervalId);
  };
}, [session, user]);
```

**Figure 18: live location tracking**

**Implementation challenges:**

The learning curve of understanding the Google Maps API was notably steep, requiring a significant investment of time and effort to grasp its various components and functionalities. Initially, familiarizing ourselves with the vast array of options and methods provided by the API was overwhelming. The documentation, while comprehensive, involved navigating through numerous examples and use cases to find the specific solutions that fit our needs.

One of the primary challenges was understanding how to integrate the API into our project seamlessly. This involved not just displaying a static map, but dynamically updating it to reflect the user's real-time location. Setting up the Google Cloud Console and enabling the necessary APIs was straightforward, but the real complexity lay in the implementation.

The process of determining and updating the user's live location required extensive trial and error. Initially, we faced difficulties in accurately retrieving and displaying the user's position using the Geolocation API. We had to ensure that the application properly requested and handled user permissions for accessing location data. Handling errors when permissions were denied or when the geolocation service failed was another critical aspect that required meticulous attention.

Once we successfully retrieved the user's location, the next challenge was to efficiently update this location on the map in real-time. We experimented with different approaches to ensure that the location data was both accurate and updated at appropriate intervals. Our initial implementations were inefficient, leading to noticeable delays and inaccuracies in the location updates.

To overcome these issues, we connected our project to Firebase, utilizing its Realtime Database for continuous updates. This integration was not straightforward and required multiple iterations to optimize. We needed to ensure that the data sync between the client and Firebase was efficient, minimizing latency while maximizing accuracy. We implemented listeners to track changes in the user's location in the database and update the map accordingly. Fine-tuning these listeners to handle real-time data efficiently without overwhelming the client or the server took considerable effort.

## 4. Profile page

### **Description:**

In our application, we designed a user information page that serves as a central hub for users to view key metrics and insights about their social interactions and progress within the platform. This page is integral to enhancing user engagement by providing clear and concise information about their current status and achievements.

Displaying Basic User Information:

1. **Total number of friends:** Number of friends in the friends list.
2. **Total number of friend requests:** Number of incoming friend requests.
3. **Current XP:** The experience points obtained with each successful delivery.
4. **User Level:** The level of the user from accumulated XP points.

### **1. Total Number of Friends:**

This section shows the total number of friends a user has added to their network. By displaying this metric prominently, users can quickly gauge the extent of their social connections within the application. This can also encourage users to expand their network by sending more friend requests.

### **2. Total Number of Friend Requests:**

Users can see the total number of friend requests they have received. This includes both pending and accepted requests. By providing this information, users are kept informed about the social activities directed towards them, prompting them to respond to pending requests and manage their social interactions more effectively.

### **3. Current XP (Experience Points):**

The XP system is a key component of the application's gamification strategy. Users earn XP for various activities and interactions within the app, such as sending and receiving gifts, making new friends, and completing certain tasks. The current XP displayed on this page allows users to track their progress and understand how close they are to reaching the next level.

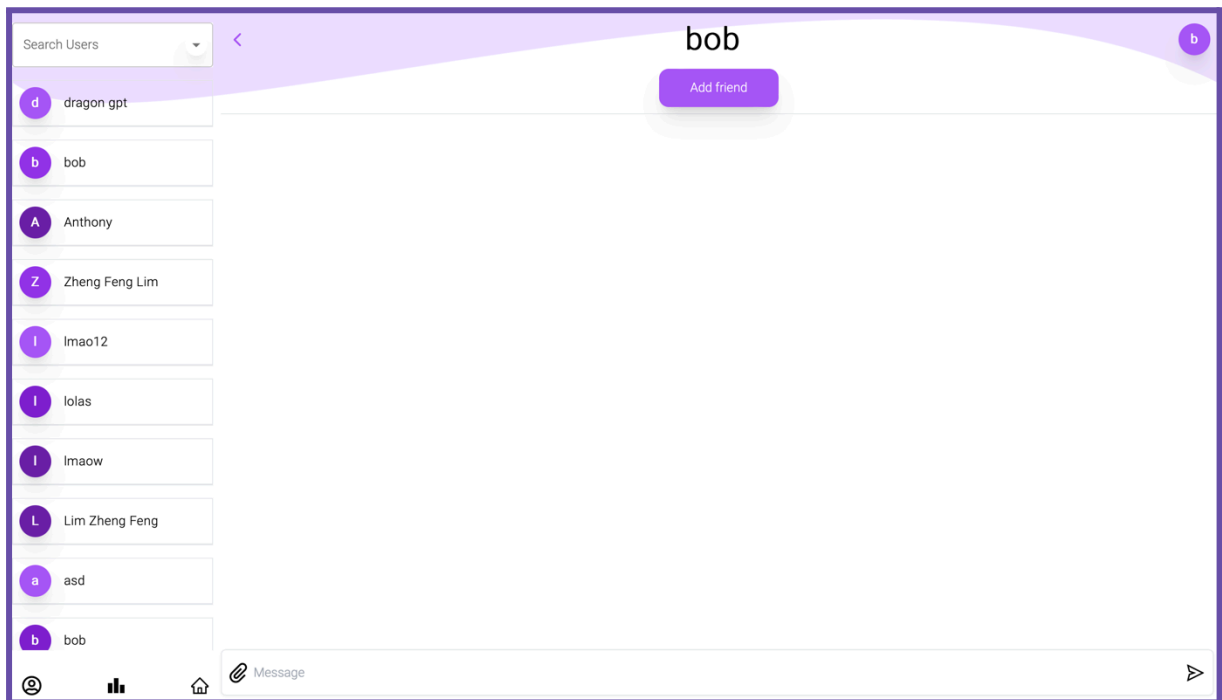
### **4. User Level:**

Alongside the current XP, the user's level is displayed, indicating their overall progress and standing within the application. Levels are typically achieved by accumulating XP and serve as milestones that unlock new features or rewards. Displaying the user level motivates users to engage more with the app to achieve higher levels and gain more benefits.

## Add Friend Logic

### **Description:**

Upon messaging a new user, if they are not your friend, a purple “Add Friend” button would be present at the top of the screen as shown in *Figure 19* below.

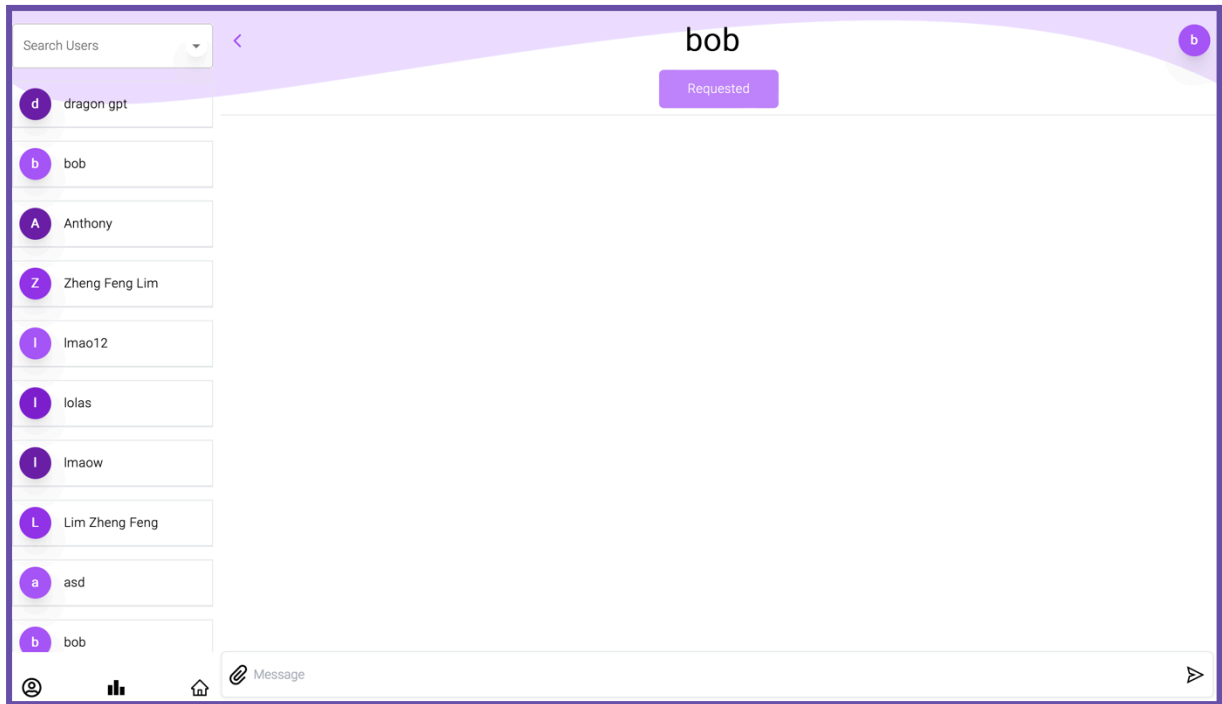


**Figure 19: “Add friend” button on top of the messaging screen**



**Description:**

When the user presses the “Add Friend” button, the button will change to “Requested” as shown in *Figure 20* below to indicate that a friend request has been sent.



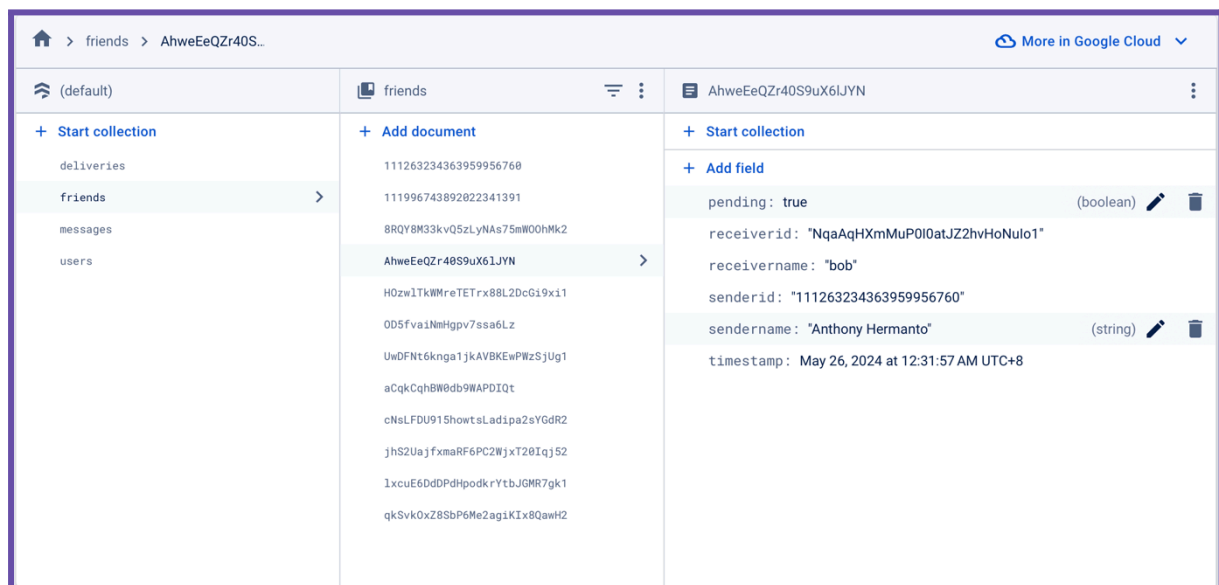
**Figure 20: “Requested” button on top of the messaging screen**

**Description:**

In the Firebase database, the user's information will be stored in the collection as "friends" with the following fields:

1. **Pending:** default to true indicating that they are not friends yet
2. **Receiverid:** the person's id that the user wants to be friends with
3. **Receivername:** the person's name that the user wants to be friends with
4. **Senderid:** the user's id
5. **Sendername:** the name of the user
6. **Timestamp:** when the user sends the friend request

Notice how pending is **true**, indicating that both users are not friends yet.



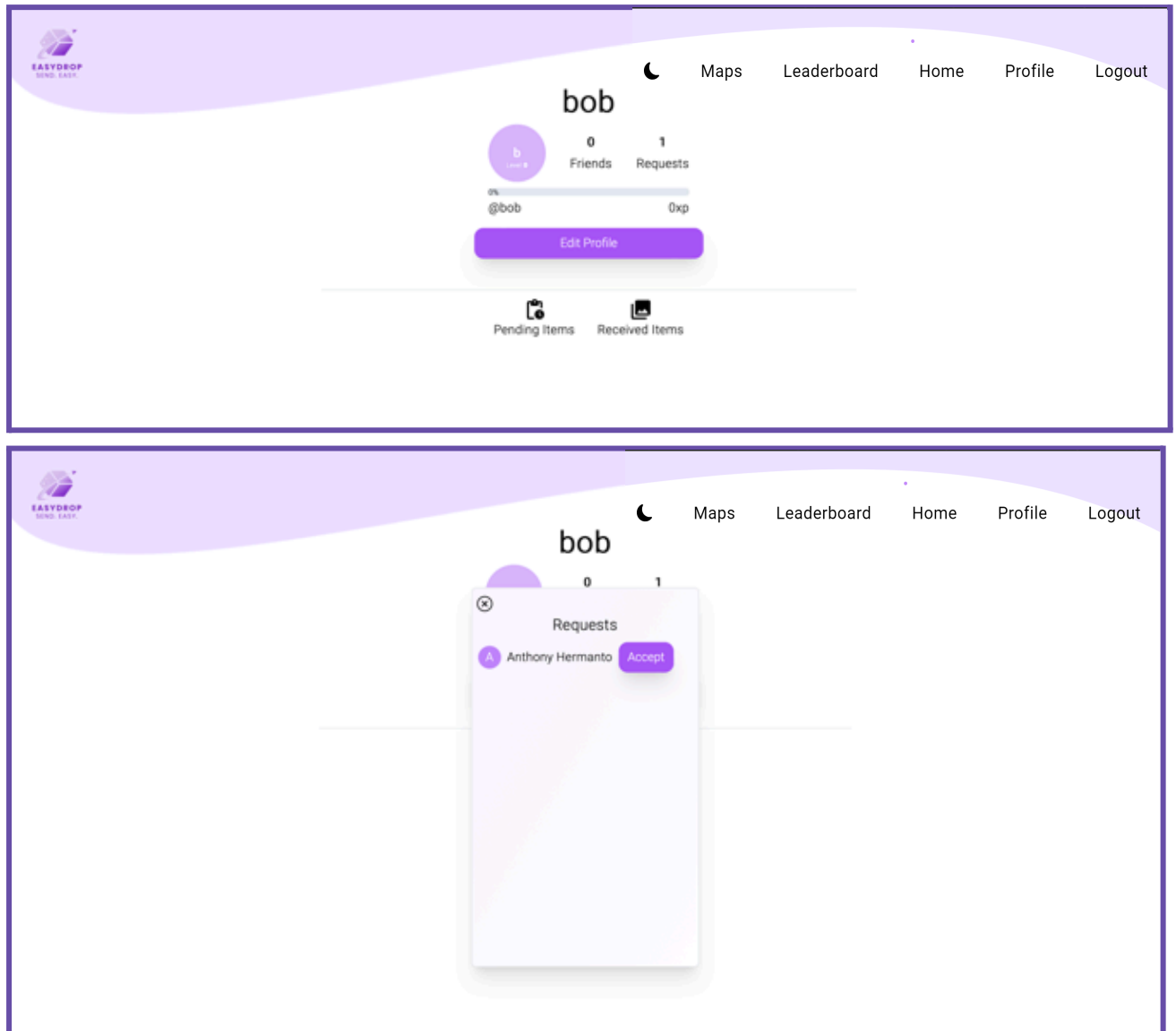
**Figure 21: Firebase "friends" collection for unaccepted request**

```
<button
  className="p-3 w-[150px]"
  onClick={async () => {
    if (user) {
      await addDoc(collection(db, "friends"), {
        senderid: user.uid,
        receiverid: Id,
        timestamp: serverTimestamp(),
        sendername: user.name,
        receivername: router.query.name,
        pending: true,
      });
    } else if (session) {
      await addDoc(collection(db, "friends"), {
        senderid: session.user.uid,
        receiverid: Id,
        timestamp: serverTimestamp(),
        sendername: session.user.name,
        receivername: router.query.name,
        pending: true,
      });
    }
  }}
>
  Add friend
</button>
```

**Figure 22: Code snippet to add to “friends” collection**

**Description:**

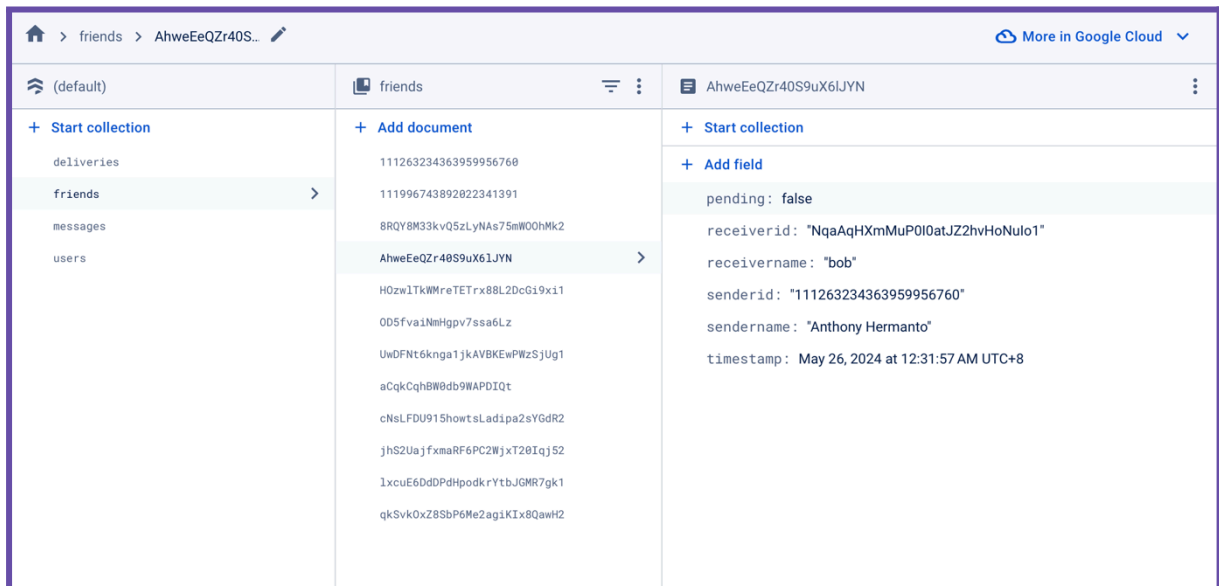
In the profile page for “bob”, the request should appear from the user that requested to be “bob”’s friend as shown below in *Figure 23*. Once “bob” accepts “Anthony Hermanto”’s friend request, both are now friends and it will be displayed in their friend’s list.



**Figure 23: Profile page for bob**

**Description:**

In the Firebase database, “Pending” has changed from true to false, indicating that both user’s are now friends.



**Figure 24: Firebase “friends” collection for accepted friend request**

```
<button
  className="px-3 py-2"
  onClick={async () => {
    await setDoc(doc(db, "friends", r.id), {
      pending: false,
      receiverid: r.data().receiverid,
      senderid: r.data().senderid,
      receivername: r.data().receivername,
      sendername: r.data().sendername,
      timestamp: r.data().timestamp,
    });
  }}
>
  Accept
</button>
```

**Figure 25 code snippet when user accepts friend request**



**Description:**

In order to display their friends, we filter through the database and check for 2 things,

1. The user's uid is equal to the sender's id
2. The user's uid is equal to the receiver's id

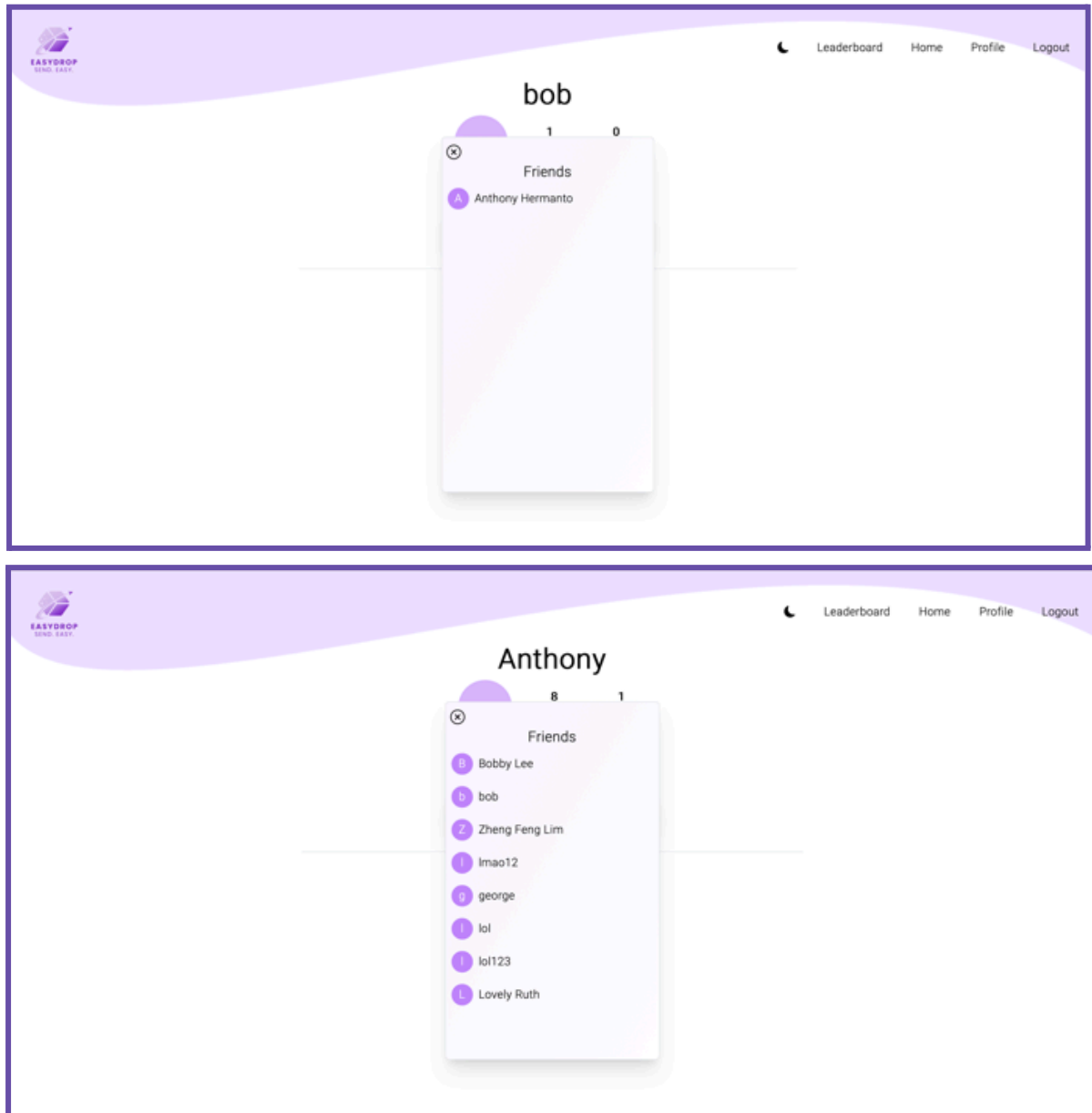
```
const friends1 = requests
  .filter((x) => x.data().senderid == id)
  .filter((x) => x.data().pending == false);

const friends2 = requests
  .filter((x) => x.data().receiverid == id)
  .filter((x) => x.data().pending == false);
```

**Figure 26: Code to filter friends from “friends” collection in Firebase**

**Description:**

As shown in *Figure 27* below, “bob” is “Anthony Hermanto”’s friend and “Anthony Hermanto” is “bob”’s friend.

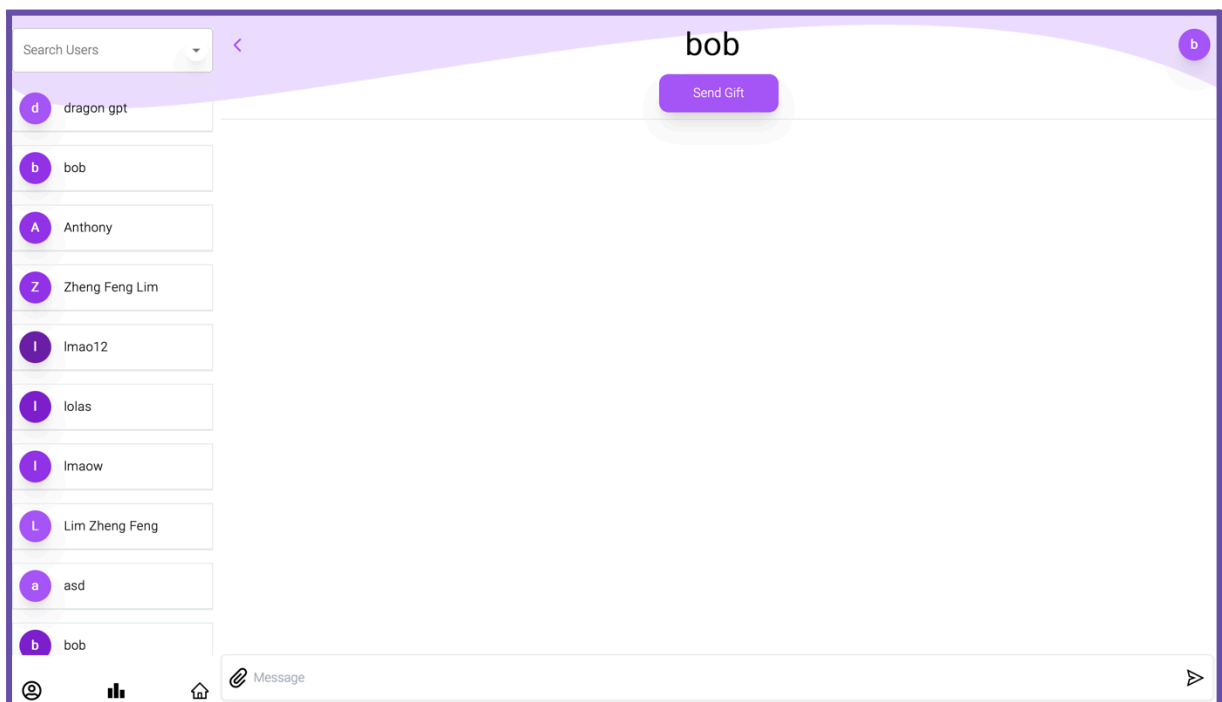


**Figure 27: Profile page for Anthony and bob**



**Description:**

The purple button also changed from “Requesting” to “Send Gifts” because “bob” is now “Anthony”’s friend. This means that Anthony now has access to the addressless delivery feature to send a gift to his friend “bob”.



**Figure 28: “Send Gift” button on top of the Messaging screen**

### **Implementation challenges:**

The core implementation challenge we faced in our project was related to the logic of determining friendships between users. Initially, our implementation only considered scenarios where the user's UID matched the sender ID in the friend request data. This approach overlooked the fact that friendships are bidirectional, similar to the logic required for messaging features.

In a social networking context, friendships are inherently bidirectional. This means that if User A sends a friend request to User B, and User B accepts it, they become friends regardless of who initiated the request. Therefore, to accurately determine if two users are friends, we need to consider both the sender and receiver IDs.

Our initial logic was flawed because it only checked if the user's UID matched the sender ID in the friend request record. This meant that we were only recognizing friendships that the user had initiated, ignoring any friend requests they had received and accepted. This incomplete logic led to inaccuracies in displaying the total number of friends and friend requests.

# Timeline & Development Plan

Week	Period	Key Targets	Remarks
<b>Week 0</b>	Before 13 May (Sprint 0)	1) Ideation as well as discussion on the key features of the website 2) Read through the documentation of the Tech Stacks we are going to use <ul style="list-style-type: none"> <li>- Next.js</li> <li>- Redux</li> <li>- Tailwind CSS</li> <li>- firebase</li> </ul> 3) Life Off preparation (Video and poster) 4) Meeting with advisor 5) Asset Design, designing our logo	Status: Completed
<b>Week 1</b>	13 May - 19 May (Sprint 1)	<b>Week 1 goals:</b> 1) Priority 1 - Tech Stack familiarization <ul style="list-style-type: none"> <li>- Get familiar with Next.js and Tailwind CSS so that we can start working on the frontend</li> </ul> 2) Priority 2 - Understand firebase authentication so that we are able to have google authentication and authentication with email and password 3) Priority 3 - Use wireframe to plan first iteration of UI 4) Priority 4 - Create the first iteration of the UI for the login, register and home page	Lift Off Submission at <b>20 May</b>  Status: Completed
<b>Week 2</b>	20 May - 26 May (Sprint 1)	<b>Week 2 goals:</b> 1) Priority 1 - Understand and Implement Redux so that we can implement persistent login 2) Priority 2 - Work on Messaging feature	Status: Completed
<b>Week 3</b>	27 May - 2 Jun (Sprint 2)	<b>Week 3 goals:</b> 1) Priority 1 - Work on profile feature 2) Priority 2 - Work on live location feature 3) Priority 3 - based on user feedback revamp the user interface of our website	Milestone 1 Evaluation (ideation) on <b>3 Jun</b>  Status: Completed

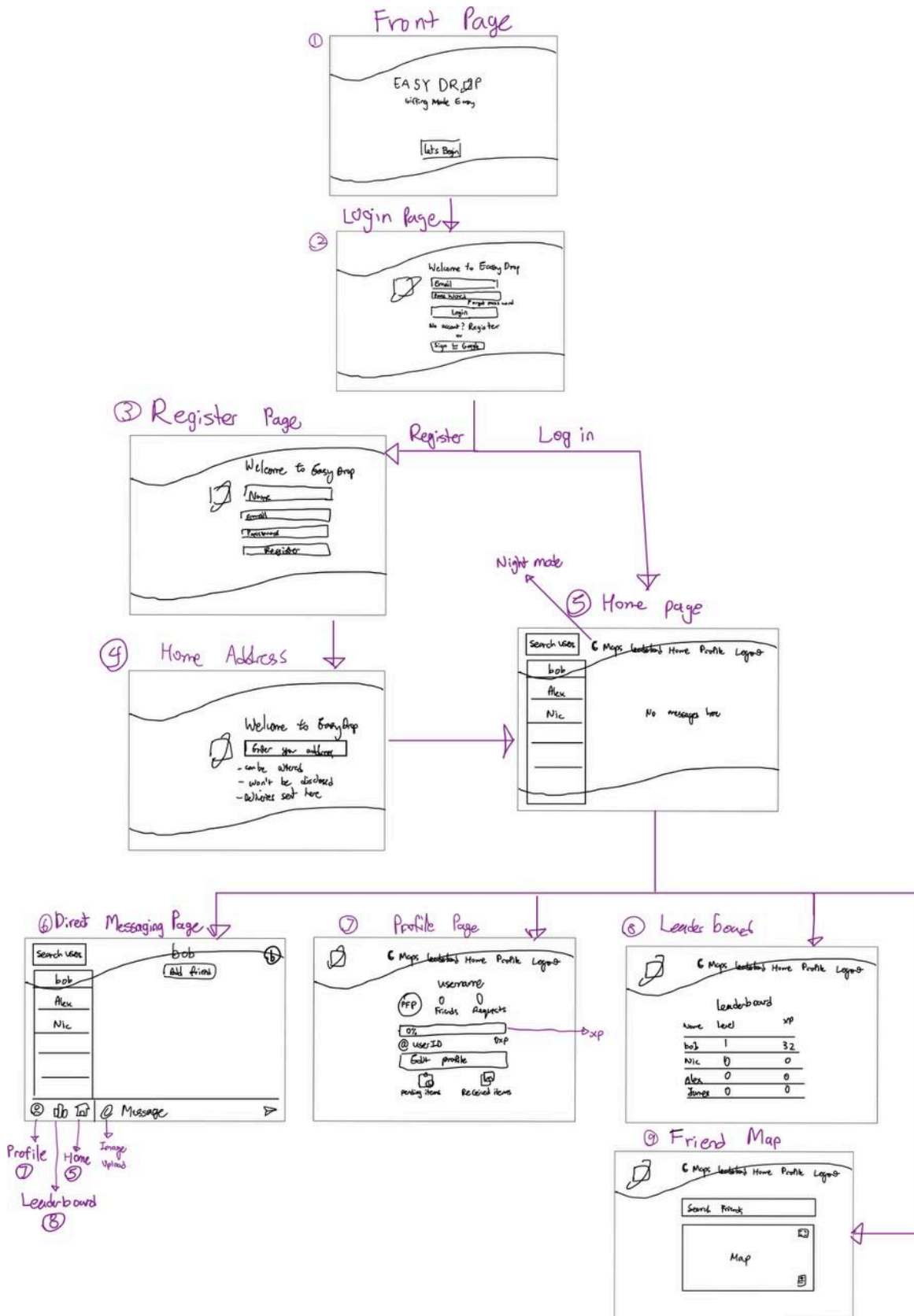
## EASY DROP

<b>Week 4</b>	3 Jun - 9 Jun (Sprint 2)	<b>Week 4 goals:</b> 1) Priority 1 - Work on the google maps feature that determines where all the friends are 2) Priority 2 - update the authentication flow such that the user needs to key in their home address before they can start using our app	
<b>Week 5</b>	10 Jun - 16 Jun (Sprint 3)	<b>Week 5 goals:</b> 1) Priority 1 - Gather user feedback on what needs improving, beta testing 2) Priority 2 -Implement miscellaneous feature such as dark mode e.t.c	
<b>Week 6</b>	17 Jun - 23 Jun (Sprint 3)	<b>Week 6 goals:</b> 1) Priority 1 - Gather more user feedback and refine product 2) Priority 2- Finalise miscellaneous features	
<b>Week 7</b>	24 Jun - 30 Jun (Sprint 4)	<b>Week 7 goals:</b> 1) Priority 1 - Fix any bugs for miscellaneous features implemented 2) Priority 2 - Start on milestone 2 project poster, video, logs and readme 3) Priority 3 - Ensure current state of product works smoothly and achieves it's intended purpose	Milestone 2 Evaluation (Prototyping) on <b>1 Jul</b>
<b>Week 8</b>	1 Jul - 7 Jul (Sprint 4)	<b>Week 8 goals:</b> 1) Priority 1 - Start implementing extension features <ul style="list-style-type: none"> <li>- Scheduled deliveries feature</li> <li>- Multi-address delivery feature</li> <li>- Admin account for delivery personnel</li> </ul> 2) Priority 2) - Try to fix as many bugs as possible	
<b>Week 9</b>	8 Jul to 14 Jul (Sprint 5)	<b>Week 9 goals:</b> 1) Priority 1 - Think of any other potential extensions for our product 2) Priority 2 - Plan out development of new extensions if any 3) Priority 3- Fix all bugs present up to this point	

## EASY DROP

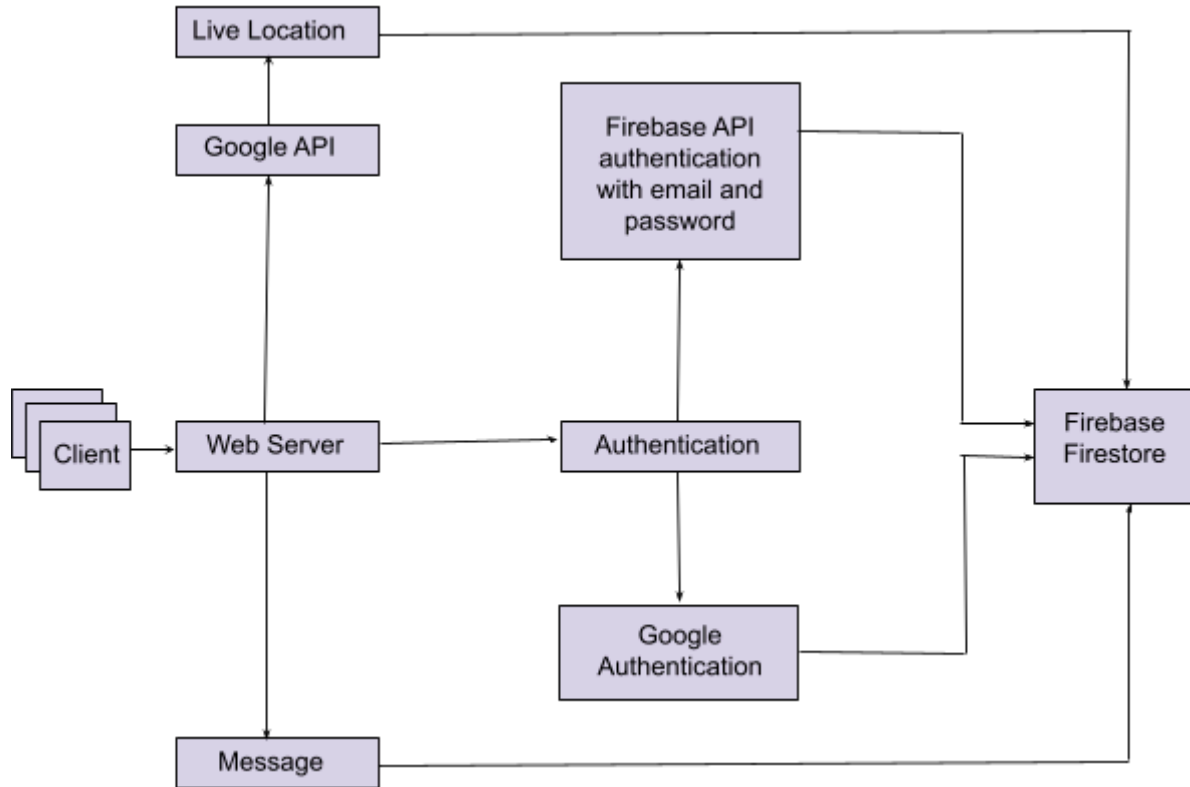
<b>Week 10</b>	15 Jul to 21 Jul (Sprint 5)	<b>Week 10 goals:</b> 1) Priority 1 - User testing on small groups of people with simulated delivery 2) Priority 2 - Refine product based on feedback received 3) Priority 3 - Start on milestone 3 project poster, video, logs and readme	
<b>Week 11</b>	22 Jul - 28 Jul (Sprint 6)	<b>Week 11 goals:</b> 1) Priority 1 - More user testing with simulated delivery for greater sample size 2) Priority 2- Refine product based on feedback received 3) Priority 3- Finalise submissions for milestone 3	Milestone 3 Evaluation (Extension) on <b>29 Jul</b>
<b>Week 12</b>	29 Jul - 4 Aug (Sprint 6)	<b>Week 12 goals:</b> 1) Priority 1 - More user testing 2) Priority 2- Bug fixes 3) Priority 3 - Implement final extensions if needed	
<b>Week 13</b>	5 Aug - 11 Aug (Sprint 7)	<b>Week 13 goals:</b> 1) Priority 1 - Wider user testing 2) Priority 2- Bug fixes 3) Priority 3 - Ensure the UI/UX is smooth	
<b>Week 14</b>	12 Aug - 18 Aug (Sprint 7)	<b>Week 14 goals:</b> 1) Priority 1 - Wider user testing 2) Priority 2 - Bug fixes/ System feature refinement 3) Priority 3 - Start finalising poster 4) Priority 4 - Try to find potential partnering delivery companies	
<b>Week 15</b>	19 Aug - 25 Aug (Sprint 8)	<b>Week 15 goals:</b> 1) Priority 1 - Last minute product refinement/ Bug fixes 2) Priority 2- Ensure whole product runs smoothly 3) Priority 3- Start Splashdown preparations	
<b>Week 16</b>	26 Aug - 28 Aug (Sprint 8)	<b>Week 16 goals:</b> 1) Priority 1- Final round of product refinement/ Bug fixes 2) Priority 2 - Complete preparation for Splashdown	Splashdown <b>28 Aug</b>

# Wireframe



**Figure 29: Initial wireframe of our product UI**

# System Design



**Figure 30: System Design Diagram**

# Software Engineering Practices

Software engineering is a disciplined approach to designing, developing, and maintaining software systems. It provides a structured framework that enhances efficiency, quality, and reliability through standardized practices and methodologies. For our project, we decided to employ two week sprints, git version control as well as other practices which will be explained in the following pages.

1. Two- Week Sprints
2. Git Version Control
3. To be continued



## 1. Two-week Sprints

Our workflow is structured around two-week sprints, a common practice in the software engineering industry following Agile methodology. The duration is long enough for us to make substantial developmental changes but also short enough for us to stay actively involved in the project rather than have idling phases. Every two weeks, we hold a sprint review meeting to assess the progress made during the previous sprint. In these meetings, we evaluate the completed tasks, verify if the project is on schedule, and plan the activities for the next sprint.

The bi-weekly sprint cycle offers several benefits to us in the development of our project. Firstly, it ensures that we maintain a steady pace of development by breaking down the project into manageable chunks. This frequent evaluation allows us to quickly identify and address any deviations from the timeline, ensuring that deadlines are consistently met.

Moreover, the two-week interval provides enough time to accomplish significant work, yet is short enough to remain adaptable. This flexibility is crucial for accommodating any unforeseen issues that may arise, allowing us to adjust our plans and priorities accordingly. Regular sprint reviews will also foster better team communication and collaboration between us as it encourages us to discuss challenges, share feedback, and align on our project development goals.

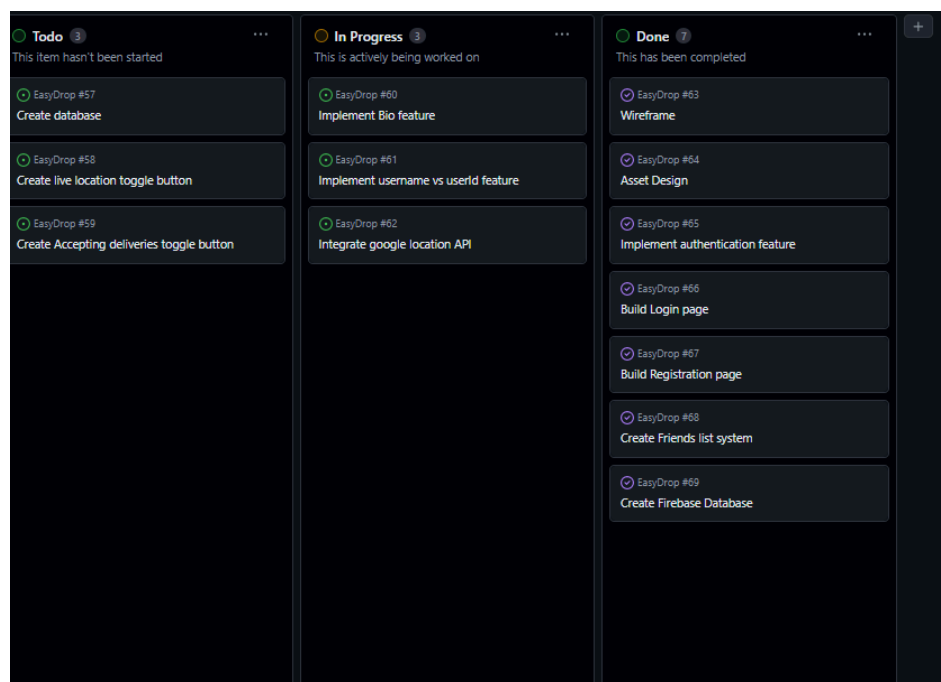
In essence, adhering to two-week sprints helps us to maintain a disciplined and agile development process, which is essential for delivering high-quality software on time. The rough scheduling of our sprints as well as what we will accomplish is listed in the *Development Plan* section previously covered, as well as in our project log which can be accessed from the *Miscellaneous* section of this document .

## 2. Git version control

Version control is crucial in software development and teamwork. Git allows several developers to collaborate on the same project at the same time without disrupting each other's work by using branches. Additionally, Git maintains a comprehensive record of all changes made to a project. This history enables developers to review, revert, or compare various versions of the code, simplifying the process of identifying and resolving issues. The main ways we utilised github in version control was through an **issue management board**, the use of **branching**, and use of the **commit history**.

### Issue Management Board:

Firstly, github has a kanban board style interface for developers to organise and manage their tasks effectively. It has a simple and intuitive GUI where issues can be labeled and assigned accordingly. We can place the issues or tasks into different stages of our workflow with customisable titles such as “To Do”, “In Progress” and “Done”. We can edit the purpose of the issues as well as the developer in charge of the issue, making the development process much more organised. *Figure 31* below displays an example of our project’s kanban board:



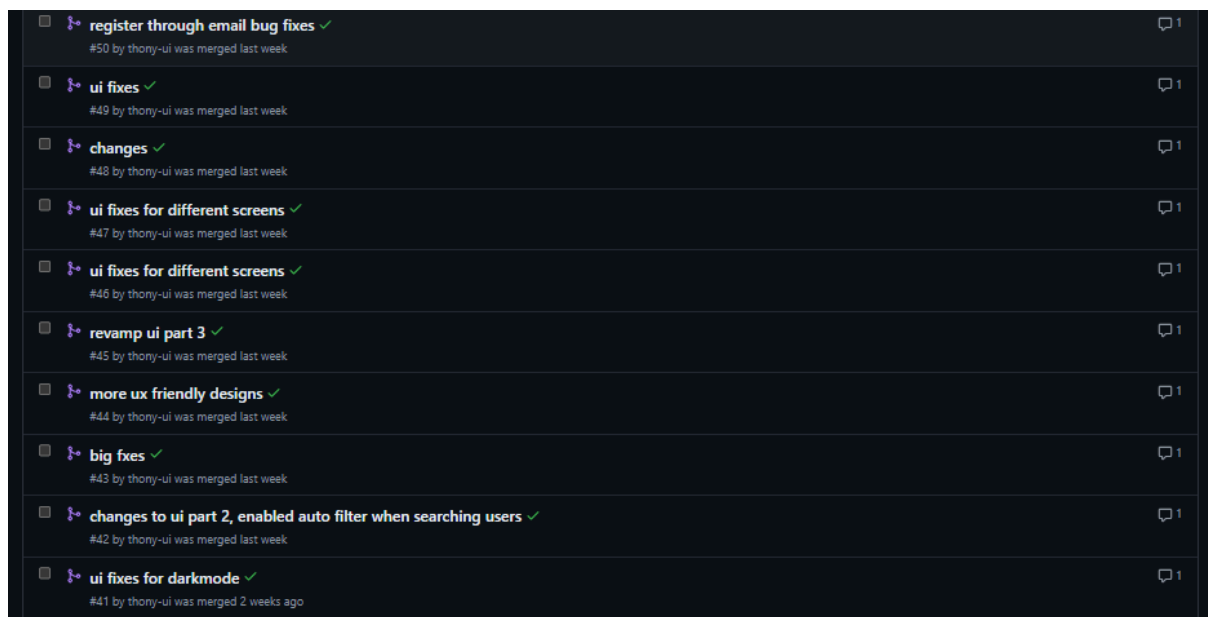
**Figure 31: Github kanban board**

## **Branching, Pull Requests, Code Reviews:**

To effectively manage our code, we employed branching strategies in Git. Whenever we tackled a specific issue or feature, we created a separate branch dedicated to that task. This approach ensured that our main branch, which usually holds our development build, remained unaffected by ongoing work.

Throughout the development of a particular feature, we made regular commits to our working branch to document our progress. Once the feature was complete, we initiated a pull request to merge our changes into the main branch. During this process, we carefully handled any conflicts that arose between the feature branch and the main codebase. This methodical approach helped us maintain the integrity of the main branch and prevented any unintended changes from being introduced.

Additionally, the use of pull requests facilitated code reviews and collaboration among team members, allowing us to ensure that all changes were thoroughly vetted before integration. By following this branching and merging strategy, we maintained a clean and stable main branch, which was crucial for our development workflow. *Figure 32* below shows an example of our branching page in Github:

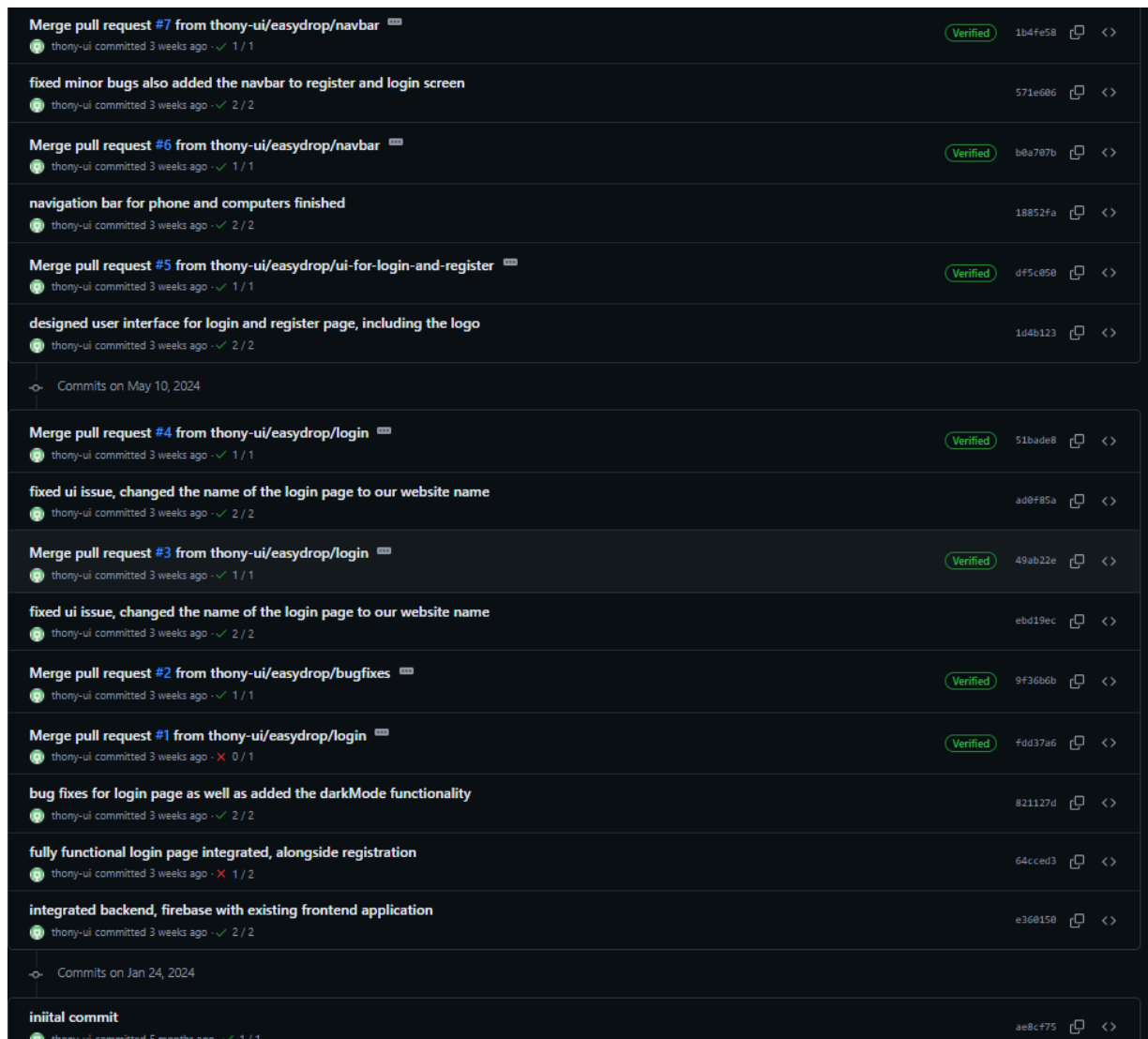


<input type="checkbox"/>	register through email bug fixes ✓ #50 by thony-ui was merged last week	1
<input type="checkbox"/>	ui fixes ✓ #49 by thony-ui was merged last week	1
<input type="checkbox"/>	changes ✓ #48 by thony-ui was merged last week	1
<input type="checkbox"/>	ui fixes for different screens ✓ #47 by thony-ui was merged last week	1
<input type="checkbox"/>	ui fixes for different screens ✓ #46 by thony-ui was merged last week	1
<input type="checkbox"/>	revamp ui part 3 ✓ #45 by thony-ui was merged last week	1
<input type="checkbox"/>	more ux friendly designs ✓ #44 by thony-ui was merged last week	1
<input type="checkbox"/>	big fxes ✓ #43 by thony-ui was merged last week	1
<input type="checkbox"/>	changes to ui part 2, enabled auto filter when searching users ✓ #42 by thony-ui was merged last week	1
<input type="checkbox"/>	ui fixes for darkmode ✓ #41 by thony-ui was merged 2 weeks ago	1

**Figure 32: Github Commit History**

## Commit History:

The commit history is an integral part of our development process as it allows us to keep a detailed record of all the changes we have made in our code base including what was changed, who made the change and when it was made. This makes it easier for us to coordinate with each other as we better understand what the other person is working on. Moreover, when a bug is introduced, the commit history enables us to identify when and where the problematic code was added which enables us to fix it as soon as possible. Additionally, if a recent change breaks the build, we can always revert to a previous stable state with the commit history. Thus, maintaining the product stability. We have included an example of our commit history in *Figure 33* below:



**Figure 33: Github Commit History**

## Software Testing

Software testing is vital to ensure the overall quality and reliability of our product. With software testing we are better able to identify and rectify bugs and issues at various stages of development. We have utilised 5 software testing methods in our project which are listed below:

1. **Unit Testing:** Ensures each component functions correctly on its own.
2. **Integration Testing:** Ensures the components work seamlessly with each other.
3. **Widget Testing:** Ensures the buttons and the rest of the UI operates as designed.
4. **System testing:** Verifies the complete and integrated software product to evaluate its compliance with requirements.
5. **User testing:** Focuses on the end-user experience, ensuring the application is intuitive and meets user needs.

These layers of testing collectively enhance the app's performance, security, and usability, leading to a robust, user-friendly, and dependable product that meets user expectations and maintains trust. The data obtained from the tests are presented over the following pages.

## 1. Unit testing

### Rationale:

Unit testing is a fundamental aspect that focuses on verifying if individual components or units of code function correctly. It involves isolating each part of the program and showing that the individual parts are correct in terms of their behavior. Through unit testing we can detect bugs early and check the quality of our code.

The results of our unit testing are shown in *Figure 34* below and the individual units test cases are collated in a table presented in the following pages:

```
(base) anthonyhermanto@AnthonySLaptop2 health-hack % jest
(node:74697) [DEP0040] DeprecationWarning: The `punycode` module is deprecated. Please use a userland alternative instead.
(Use `node --trace-deprecation ...` to show where the warning was created)
PASS __test__/Stars.test.js
(node:74696) [DEP0040] DeprecationWarning: The `punycode` module is deprecated. Please use a userland alternative instead.
(Use `node --trace-deprecation ...` to show where the warning was created)
PASS __test__/Address.test.js
(node:74695) [DEP0040] DeprecationWarning: The `punycode` module is deprecated. Please use a userland alternative instead.
(Use `node --trace-deprecation ...` to show where the warning was created)
PASS __test__/GenerateStars.test.js
PASS __test__/Modal.test.js
PASS __test__/FrontPage.test.js
PASS __test__/Loading.test.js
(node:74694) [DEP0040] DeprecationWarning: The `punycode` module is deprecated. Please use a userland alternative instead.
(Use `node --trace-deprecation ...` to show where the warning was created)
PASS __test__/NavBar.test.js
PASS __test__/HomePage.test.js
(node:74693) [DEP0040] DeprecationWarning: The `punycode` module is deprecated. Please use a userland alternative instead.
(Use `node --trace-deprecation ...` to show where the warning was created)
PASS __test__/Login.test.js
(node:74691) [DEP0040] DeprecationWarning: The `punycode` module is deprecated. Please use a userland alternative instead.
(Use `node --trace-deprecation ...` to show where the warning was created)
PASS __test__/Maps.test.js
(node:74692) [DEP0040] DeprecationWarning: The `punycode` module is deprecated. Please use a userland alternative instead.
(Use `node --trace-deprecation ...` to show where the warning was created)
PASS __test__/Header.test.js

Test Suites: 11 passed, 11 total
Tests: 22 passed, 22 total
Snapshots: 0 total
Time: 1.85 s, estimated 2 s
```

**Figure 34: Unit Testing Result**

```

  ✓ __test__
    ✓ JS Address.test.js
    ✓ JS FrontPage.test.js
    ✓ JS GenerateStars.test.js
    ✓ JS Header.test.js
    ✓ JS HomePage.test.js
    ✓ JS Loading.test.js
    ✓ JS Login.test.js
    ✓ JS Maps.test.js
    ✓ JS Modal.test.js
    ✓ JS NavBar.test.js
    ✓ JS Stars.test.js

```

**Figure 35: Unit Testing test cases**

## EASY DROP

S/N	Unit Tested	Test		Result
		Description	Expected Result	
1	Address	Renders correctly	Snapshot match	Pass
		Check if user is logged in	Successfully checked that user is logged in	Pass
		Check if navigate to home screen after keying in address and pressing submit	Successfully navigates to the home screen when pressing the submit button	Pass
2	FrontPage	Renders correctly	Snapshot match	Pass
3	GenerateStars	Renders correctly	Snapshot match	Pass
4	Header	Renders correctly	Snapshot match	Pass
		Check if google maps API is working	Successfully renders the google maps API using dummy values	Pass
5	Homepage	Renders Correctly	Snapshot match	Pass
		Check is user is successfully logged in using firebase email and password/google authentication	Successfully checked that user is logged in	Pass
		Render 2 child component	2 component	Pass
6	Loading	Renders Correctly	Snapshot match	Pass

## EASY DROP

S/N	Unit Tested	Test		Result
		Description	Expected Result	
7	Login	Renders Correctly	Snapshot match	Pass
		If user forgets password navigates to the reset password screen	Snapshot match	Pass
		If user have no account, navigate to the register screen	Snapshot match	Pass
		Check if google maps API is working	Successfully renders the google maps API using dummy values	Pass
9	Modal	Renders Correctly	Snapshot match	Pass
		Add to database	Successfully added to firebase firestore	Pass
10	NavBar	Renders Correctly	Snapshot match	Pass
		Test dark mode (expect false)	false	Pass
		If the user clicks "home", navigate to the home page	User navigates to home page	Pass
		If the user clicks "profile", navigate to the profile page	User navigates to profile page	Pass
		If the user clicks logout, the user is logged out.	User logs out	Pass
11	Stars	Renders correctly	Snapshot match	Pass



## 2. Widget testing

### **Rationale:**

Widget testing focuses on the smallest interactive parts of an application's UI, testing their behavior in isolation to ensure that they not only function correctly but also render correctly. It involves creating test cases that simulate user interactions with the widget, such as clicking buttons, entering data into input fields, selecting options from dropdown menus, and verifying the resulting actions or changes in the widget or the underlying application. The data produced from our widget tests is presented in the table below and in the following pages.

Test #	Widget Name	Test Case Description	Test Data	Expected Outcome	Actual Outcome	Pass (P) /Fail (F)
1	Login	Verify rendering of login form with all fields and buttons	None	Login form renders with email, password fields, and login button	Login form renders with email, password fields, and login button	P
2	Login	Verify login button click with valid input	Email: user1369@gmail.com, Password: correctpassword	Login successful, navigate to dashboard	Login successful, navigate to dashboard	P
3	Login Registration	Verify login button click with invalid input	Email: user1369@gmail.com, Password: abcdefg	Login unsuccessful	Login unsuccessful	P

## EASY DROP

Test #	Widget Name	Test Case Description	Test Data	Expected Outcome	Actual Outcome	Pass (P) /Fail (F)
4	Registration	Verify rendering of registration form with all fields	None	Registration form renders with all input fields and submit button	Registration form renders with all input fields and submit button	P
5	Registration	Verify registration with valid input	Valid user details	Registration successful, navigate to address page	Registration successful, navigate to address page	P
6	Registration	Verify registration with invalid input	Invalid email format	Registration unsuccessful	Registration unsuccessful	P
7	Profile	Verify rendering of profile page	Logged in user	Profile page displays user details	Profile page displays user details	P
8	Profile	Verify rendering of profile page	New Bio: "Test bio123"	Bio updated successfully, new bio displayed	Bio updated successfully, new bio displayed	P

## EASY DROP

Test #	Widget Name	Test Case Description	Test Data	Expected Outcome	Actual Outcome	Pass (P) /Fail (F)
9	Friends list	Verify rendering of friends list	Logged in user	Friends list displays all friends	Friends list displays all friends	P
10	Friends request button	Verify sending friend request	Friend Username: bob	Friend request sent, confirmation message displayed	Friend request sent, confirmation message displayed	P
11	Message input	Verify rendering of message input field	None	Message input field and send button displayed	Message input field and send button displayed	P
12	Message inputNotification Badge	Verify sending a message	Message: "Hello!"	Message sent successfully, appears in chat	Message sent successfully, appears in chat	P
13	Notification Badge	Verify rendering of notification badge	Logged in user	Notification badge displays correct count	Notification badge displays correct count	P

## EASY DROP

Test #	Widget Name	Test Case Description	Test Data	Expected Outcome	Actual Outcome	Pass (P) /Fail (F)
14	Notification Badge	Verify updating notification badge upon receiving new request	New friend request received	Notification badge count updated	Notification badge count updated	P
15	Location Sharing Toggle	Verify rendering of location sharing toggle	Logged in user	Location sharing toggle displayed	Still implementing	F
16	Location Sharing Toggle	Verify toggling location sharing on/off	Toggle action	Location sharing status updated	Still implementing	F

### 3. Integration Testing

#### **Rationale:**

For Integration testing, individual units or components of a software application are combined and tested as a subgroup of the whole system. The goal is to identify issues that arise when different components interact with each other. It helps ensure that integrated components work together as expected and that the interfaces between them function correctly. The test results we produced from integration testing is presented in the table below:

Test #	Integration component	Test Case Description	Expected Outcome	Actual Outcome	Pass (P) /Fail (F)
1	Login & Authentication API	Verify successful login and token retrieval	User is authenticated and token is returned	User is authenticated and token is returned	P
2	Registration & Database	Verify new user registration and data storage	User data is stored in the database	User data is stored in the database	P
3	Profile Update & Storage	Verify profile updates are saved and retrieved correctly	Updated profile information is displayed	Updated profile information is displayed	P

## EASY DROP

Test #	Integration component	Test Case Description	Expected Outcome	Actual Outcome	Pass (P) /Fail (F)
4	Messaging UI & Backend	Verify sending and receiving messages	Messages are correctly sent and displayed in the chat	Messages are correctly sent and displayed in the chat	P
5	Friend Request & Notification	Verify friend request functionality and notifications	Friend requests are sent, accepted, and notifications received	Friend requests are sent, accepted, and notifications received	P
6	Location Sharing & Privacy	Verify toggling location sharing on/off	Location sharing status is updated correctly and private data is secure	Location sharing status is updated correctly and private data is secure	P

## 4. System Testing

### **Rationale:**

System testing is designed to evaluate the complete and integrated system as a whole. It enables us to verify the end-to-end functionality of our product, identifying defects and issues before the software is released to users. By thoroughly testing all aspects of the system, we aimed to deliver a robust and reliable application that could handle real-world usage effectively.

### **Planning:**

For our project, we started by planning and designing our test cases based on the system's requirements and functionalities. The key components identified for testing included user authentication, profile management, friend requests, real-time location tracking, messaging, and notifications.

### **Test Case Development:**

We developed comprehensive test cases to cover various scenarios, including both valid and invalid inputs. Each test case included the test number, type, data, reason, expected outcome, actual outcome, and pass/fail status. The aim was to ensure thorough testing of all functionalities. We collated all our test cases in a table, along with the results produced, and presented the data obtained in the following pages. For the test cases that failed, we plan to refine the system and conduct further system testing once our product is finished.

## EASY DROP

Test #	Test Type	Test Data	Reason	Expected Outcome	Actual Outcome	Pass (P) /Fail (F)
1	Valid	Email: user1367@gmail.com	Verify valid email login functionality	User should be able to log in	User is able to log in	P
2	Valid	Email: test1367@gmail.com	Verify Google email login functionality	User should be able to log in with Google	User is able to login with google	P
3	Invalid	Email: invalid@com	Verify system handles invalid email format	System should reject the invalid email	System did not reject invalid email	F
4	Valid	Password: 123	Check system response to invalid password	System should reject the login attempt	System did not respond to login attempt	P
5	Valid	Address: 123 Main St	Verify address entry functionality	Address should be saved in the database	Address saved in database	P
6	Valid	Address: 456 Elm St	Verify another valid address entry	Address should be saved in the database	Address saved in database	P
7	Invalid	Address: ""	Check system response to empty address	System should prompt for a valid address	System did not respond	P
8	Valid	Location: Enabled	Verify real-time location tracking functionality	User's location should be accurately tracked	Location was tracked	P
9	Valid	Send Friend Request	Check sending friend request functionality	Friend request should be sent successfully	Friend request sent successfully	P



## EASY DROP

Test #	Test Type	Test Data	Reason	Expected Outcome	Actual Outcome	Pass (P) /Fail (F)
10	Valid	Accept Friend Request	Verify accepting friend request functionality	Friend request should be accepted	Friend request accepted	P
11	Valid	Reject Friend Request	Verify rejecting friend request functionality	Friend request should be rejected	No reject button	F
12	Valid	View Profile	Ensure profile page displays user details correctly	Profile page should show correct user details	Correct details displayed	P
13	Valid	Update Bio	Verify updating user bio functionality	User bio should be updated successfully	Bio updated	P
14	Valid	Profile Image	Check if profile image is displayed correctly	Profile image should be displayed if available	Profile image not displayed	F
15	Valid	Notification: Delivery Status	Ensure notifications are received for delivery	User should receive delivery status notifications	No notification received	F
16	Valid	Logout	Verify logout functionality	User should be logged out successfully	Logged out	P
17	Valid	Password Recovery	Check password recovery via email	User should receive a password recovery email	Email received	P
18	Valid	Admin Delivery Tracking	Ensure admin can track deliveries	Admin should be able to track all deliveries	No delivery yet	F
19	Valid	Registration : Email without @	Check email validation during registration	System should reject the email	System rejects email	P
20	Invalid	Registration : Password less than 6 chars	Check password strength validation during registration	System should reject the weak password	System rejects password	P

## EASY DROP

Test #	Test Type	Test Data	Reason	Expected Outcome	Actual Outcome	Pass (P) /Fail (F)
21	Valid	Registration : Valid email and password	Verify successful registration	User should be registered and data saved in database	Registered and saved	P
22	Invalid	Login: Unregistered email	Check login with unregistered email	System should reject login attempt	Login rejected	P
23	Valid	Home Page Access	Ensure home page loads correctly	Home page should load with all elements	Home page Loaded	P
24	Valid	Profile: Update Profile Picture	Verify updating profile picture functionality	Profile picture should be updated successfully	Profile picture not updated	F
25	Valid	Search Friend	Ensure search friend functionality works	Search results should display relevant users	Relevant users displayed	P
26	Valid	Send Message	Verify sending message functionality	Message should be sent successfully	Message sent	P
27	Valid	Receive Message	Verify receiving message functionality	Message should be received successfully	Message received	P
28	Valid	Real-Time Chat	Ensure real-time chat functionality	Messages should appear in real-time	Message appeared in real time	P
29	Valid	Edit Home Address	Verify editing home address functionality	Home address should be updated successfully	Home address not editable currently	F
30	Valid	Notification: Friend Request	Ensure notifications for friend requests	User should receive notifications for friend requests	Notification received	P

## EASY DROP

Test #	Test Type	Test Data	Reason	Expected Outcome	Actual Outcome	Pass (P) /Fail (F)
31	Valid	Notification: Friend Request Acceptance	Ensure notifications for accepted friend requests	User should receive notification of acceptance	No notification but visual cue present	P
32	Valid	GPS Permission	Verify GPS permission request functionality	System should prompt for GPS permission	Prompted	P
33	Valid	GPS Denial Handling	Check system response to denied GPS permission	System should handle GPS denial gracefully	Handled gracefully	P
34	Valid	Error Handling: Server Down	Ensure proper error message when server is down	User should see appropriate error message	Error message	P
35	Valid	Performance: High Load	Verify system performance under high load	System should remain responsive	Responsive	P
36	Valid	Security: SQL Injection	Check for SQL injection vulnerabilities	System should prevent SQL injection	Prevented	P
37	Valid	Security: XSS Attack	Check for XSS vulnerabilities	System should prevent XSS attacks	Prevent	P
38	Valid	Data Encryption	Ensure sensitive data is encrypted	Data should be encrypted in transit and at rest	Encrypted appropriately	P
39	Valid	Data Decryption	Verify data decryption functionality	Data should be decrypted properly	Decrypted appropriately	P
40	Valid	Third-Party API: Google Maps	Ensure Google Maps API integration works	Map should display user location correctly	Displayed correctly	P

## 5. User Testing

### **Rationale:**

User testing involves collecting feedback from users who have tested our product and using that feedback to refine our product. This helps us to identify and fix bugs, enhancing the app's stability before release. By understanding how users interact with the app, we can improve the user interface and overall experience, making it more intuitive and satisfying. Thus far we have had several feedback about the UI however we will require more feedback to determine the direction of refinement we should take.

The QR code provided below leads to a google form that enables you to key in feedback if you would like to be involved in our user testing. Do note that the product is currently in its infancy stage.



[Form Link](#)

# Miscellaneous

**Proof of Concept:**

[EasyDrop-Website](#)

**Project Log:**

Refer to the attached log: [EasyDrop-Project Log](#)