SkiTracker - Advanced Mobile App Development

Bradley Wilson - 19809461

I confirm that I have a Learning Support Plan for ‘extension’ as recommended by the Disability and Dyslexia Team, and agreed by the School. I understand the deadline for my assessment has been adjusted (as per the required School protocol) and that this should be taken into consideration when my assessment is marked/ graded.

1) Introduction

For this project, I set out to create a mobile app that would allow someone to send their location in real-time to another user or multiple users. Some potential users could be a family that goes on skiing holidays, but one of them does not ski and would like to see where their family is. Additionally, if an emergency like a skier becomes lost, it will be much easier to find them because anyone who is connected can see their exact location.

1.1) Requirements

Must

- Friend system
    - Easy to exchange friend codes
    - Join a friend to see their location
    - Save friend information to file
- Live location updates
    - Remote server to handle requests
- Location visible on a map

Should

- Change your name
- Online/Offline status
- Multiple users can join a single user

Could

- Display information like altitude, speed, and distance travelled
- Saving ski runs

Won't

- Ski data analysis

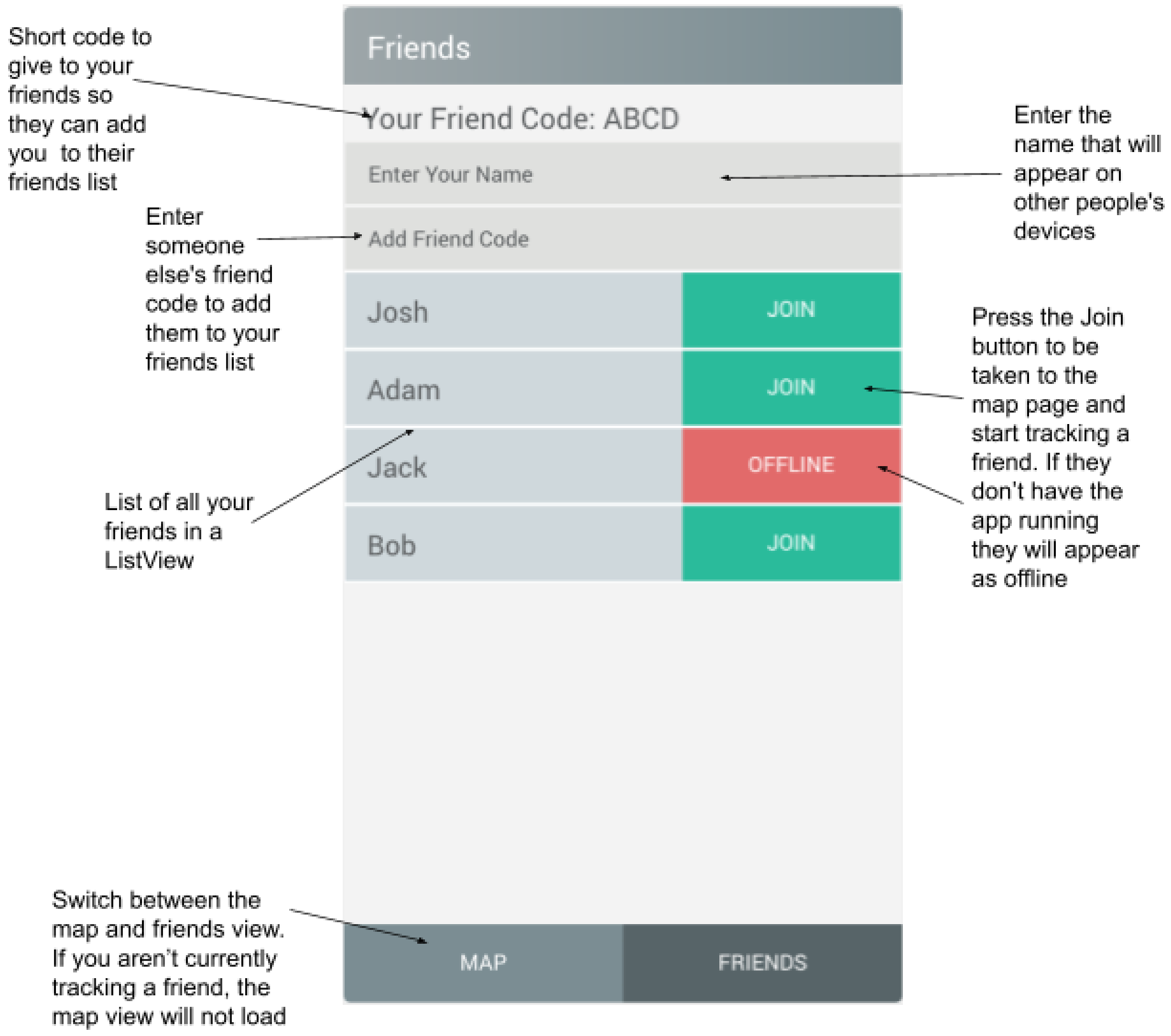
2) Technical

2.1) User interface

2.1.1) Friend View

The friend view is the first page the user sees when they start the app so it needs to be visually appealing, simple, and intuitive.

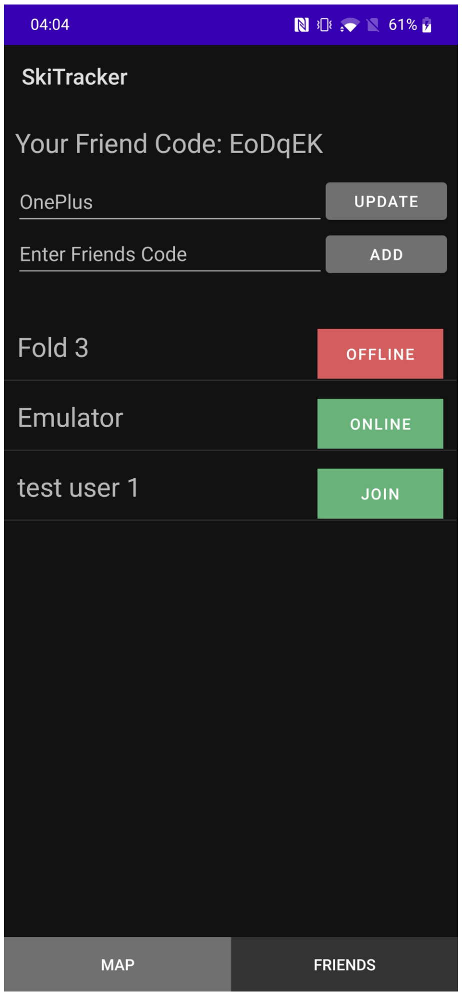
The image below is a wireframe mockup. All the buttons are as large as possible to make it easy for the user to press while wearing gloves. The colours are also bright to ensure they are visible if the user is skiing with goggles with a colour filter.


Short code to give to your friends so they can add you to their friends list
Enter someone else's friend code to add them to your friends list
List of all your friends in a ListView
Switch between the map and friends view. If you aren't currently tracking a friend, the map view will not load
Friends
Your Friend Code: ABCD
Enter Your Name
Add Friend Code
Josh
JOIN
Adam
JOIN
Jack
OFFLINE
Bob
JOIN
MAP
FRIENDS
Enter the name that will appear on other people's devices
Press the Join button to be taken to the map page and start tracking a friend. If they don't have the app running they will appear as offline

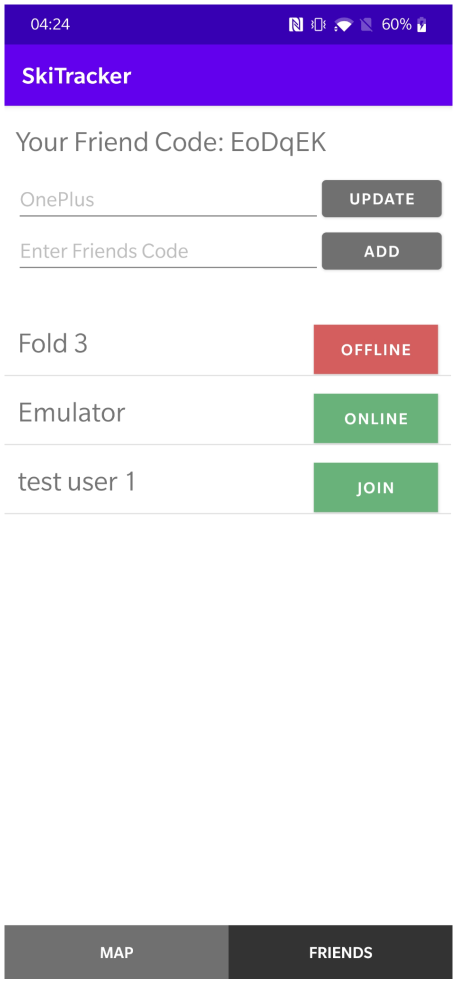
(Fig 1. Friend View wireframe)

The final version of the friend screen is similar to the wireframe, however, there are now buttons for both the name and friend code text inputs. This is because during development there was a continuing issue where the text would not submit when the user pressed enter on their keyboard. Adding the button resolved this and it works perfectly now.

Additionally, the page reacts to the device theme. Figure 2 is set to a dark theme while Figure 3 is light.



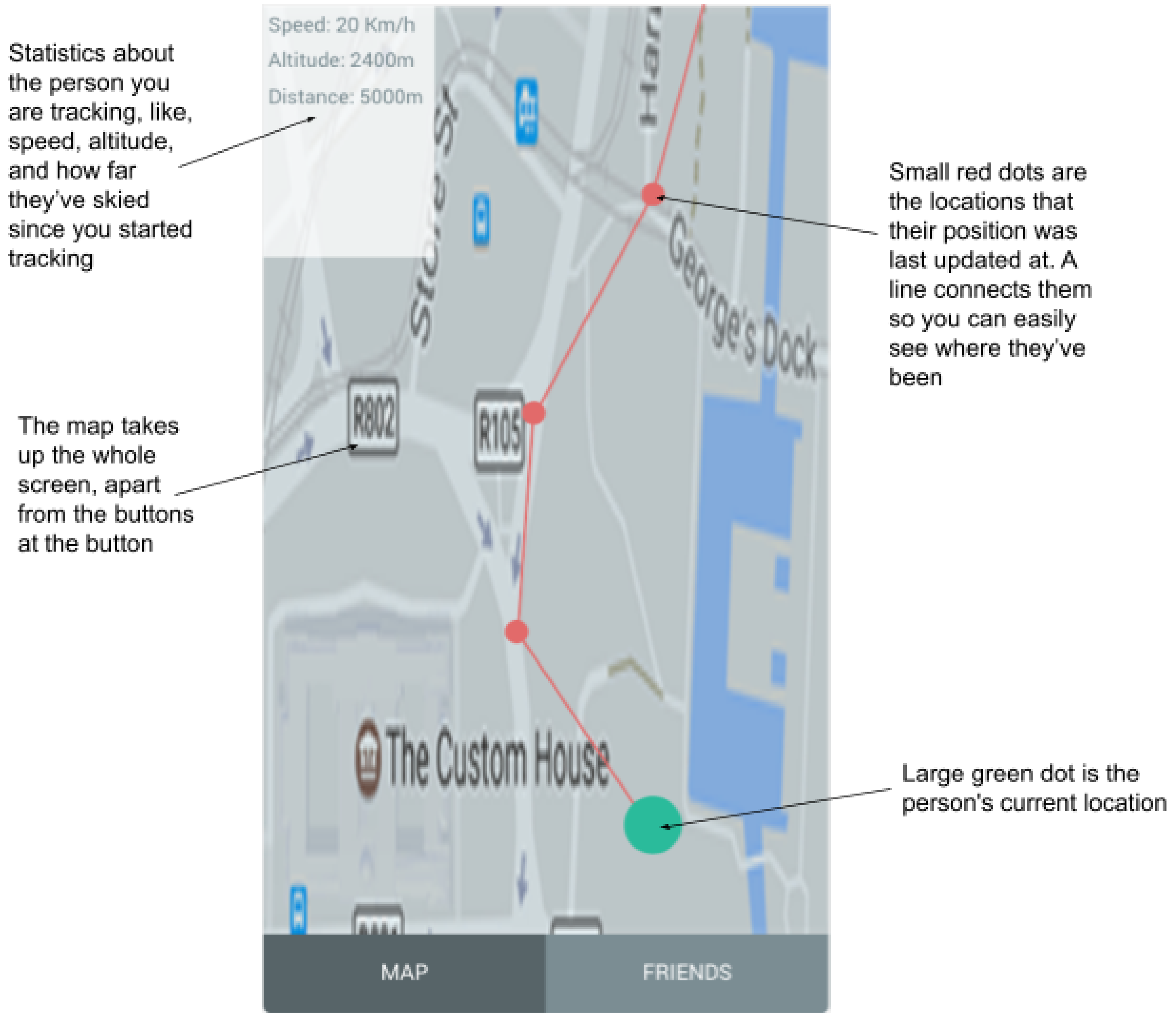
(Fig 2. Final Friend View on actual device Dark theme)



(Fig 3. Final Friend View. Light theme)

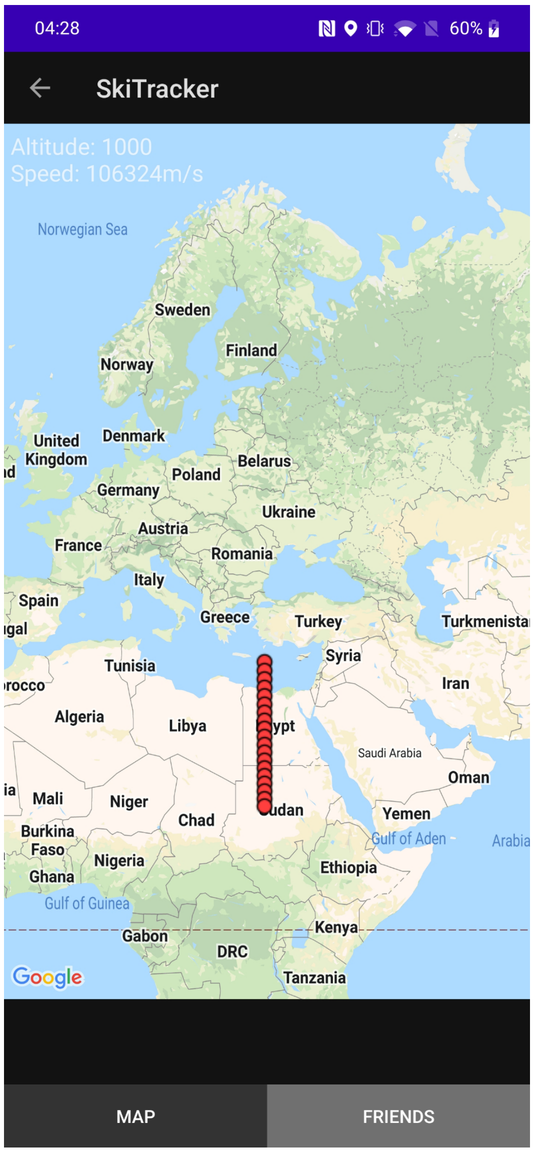
2.1.2) Map View

Similarly to the friend view, the map view needs to be as simple as possible, while giving the user as much information as possible. If there is too much data they will become confused and give us, however, if only the data they need is presented they will feel much more comfortable and enjoy using the app.


Speed: 20 Km/h
Altitude: 2400m
Distance: 5000m
Statistics about the person you are tracking, like, speed, altitude, and how far they've skied since you started tracking
Small red dots are the locations that their position was last updated at. A line connects them so you can easily see where they've been
The map takes up the whole screen, apart from the buttons at the button
Large green dot is the person's current location
George's Dock
R802
R105
The Custom House
MAP
FRIENDS

(Fig 4. Map view wireframe)

The final version of the map view is slightly different from the initial wireframe. Firstly, there are not any lines between the dots. This is because I was not able to find a way to achieve it properly. Secondly, in the wireframe, the information panel in the top left has a total distance counter, however, the final version is missing this.

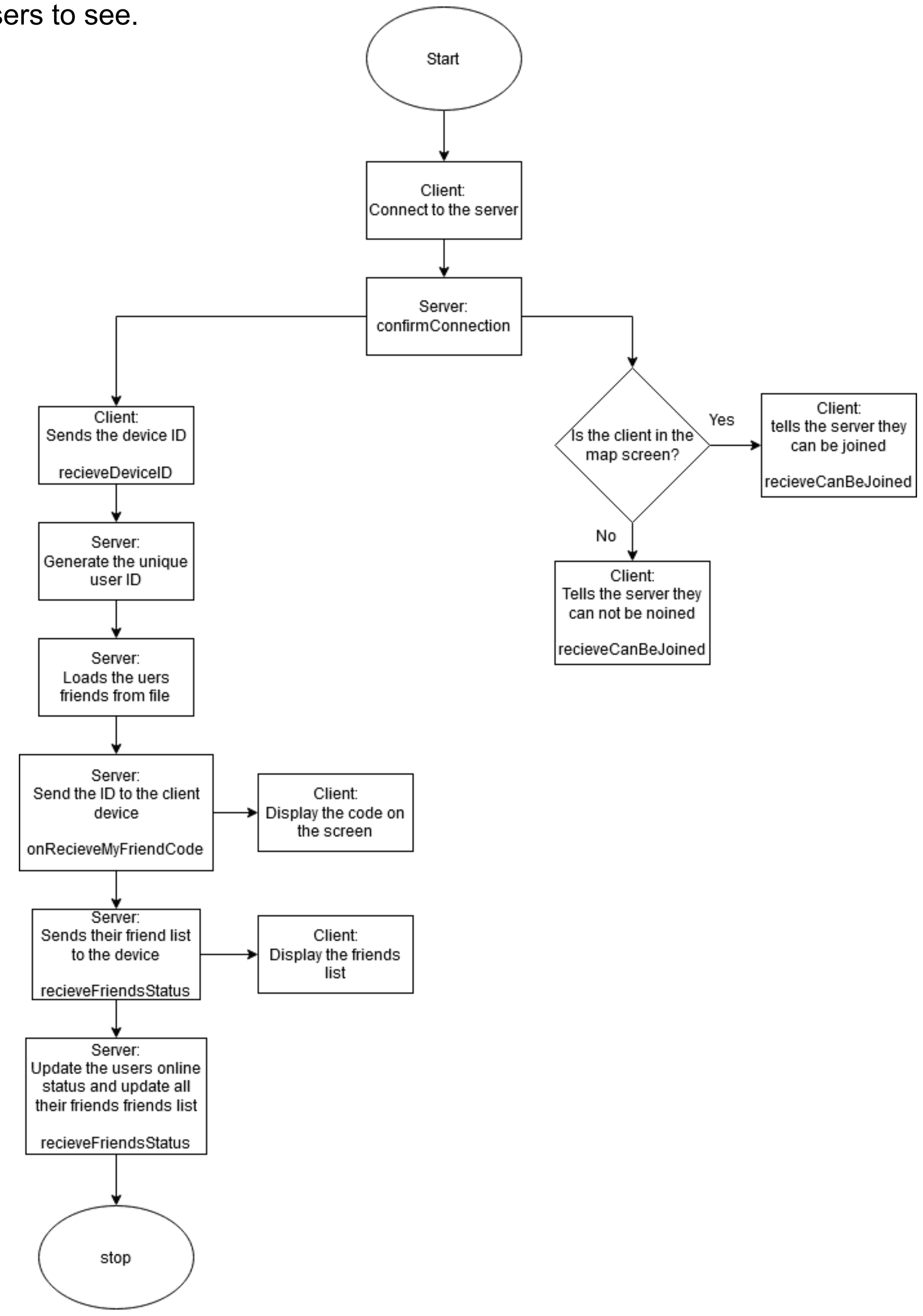


(Fig 5. Map view on an actual device)

2.3) Server/Socket

Due to the nature of the app, the majority of the processing happens on a remote server. The server is written in NodeJS and uses Socket.Io for communication. Having the app communicate with a remote server means that everything needs to be as secure and efficient as possible.

When the user first opens the app it automatically connects to the socket. Once a handshake has been performed and the user has been authenticated the server sends the user all the information they need, like their friend code and friend list as well as setting their status to online for other users to see.


Start
Client: Connect to the server
Server: confirmConnection
Client: Sends the device ID recieveDeviceID
Server: Generate the unique user ID
Server: Loads the uers friends from file
Server: Send the ID to the client device onRecieveMyFriendCode
Client: Display the code on the screen
Server: Sends their friend list to the device recieveFriendsStatus
Client: Display the friends list
Server: Update the users online status and update all their friends friends list recieveFriendsStatus
stop
Is the client in the map screen?
Yes
Client: tells the server they can be joined recieveCanBeJoined
No
Client: Tells the server they can not be noined recieveCanBeJoined

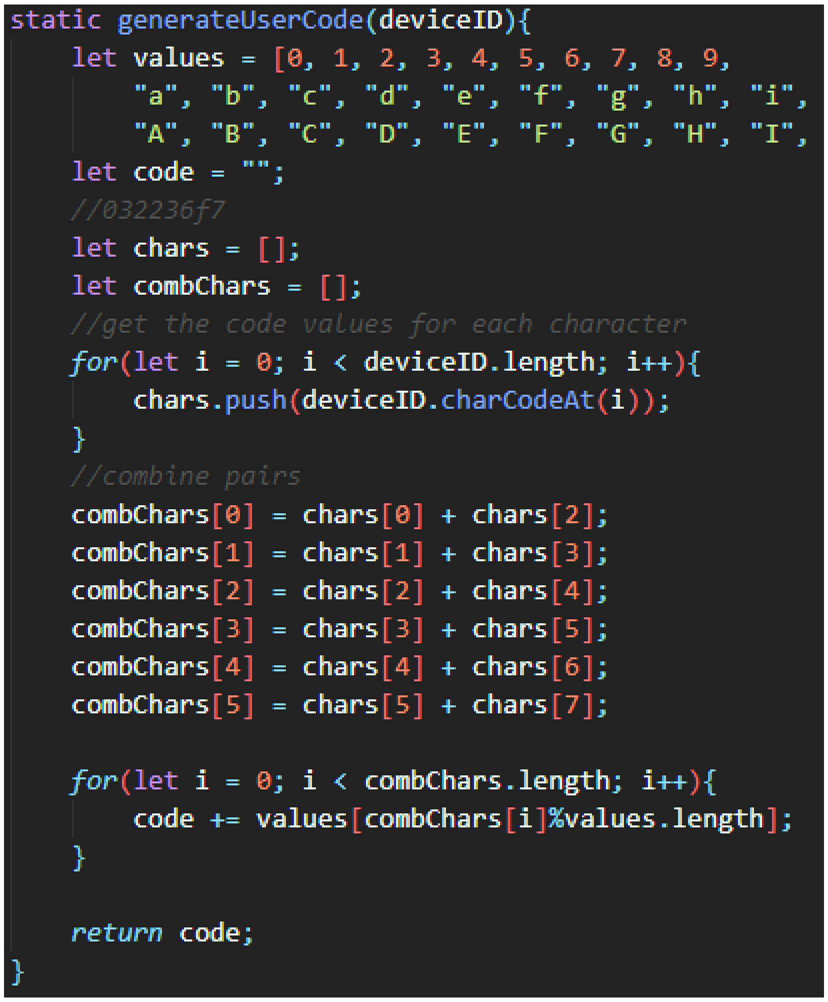
(Fig 6. First connection flow diagram)

Being able to give each user their own unique identification code was a challenge because the code needs to be the same every time the user starts the app otherwise they would have to give their friends a new code every time. Additionally, it needs to be impossible to generate a duplicate code.

The custom system that is being used works by scrambling the first eight characters of the client's device ID. Using the device ID ensures it is the same every time. Only using the first eight makes sure that no sensitive data is sent over the internet because these characters mean nothing without the full ID, which is almost impossible to get when only knowing the first eight.

The code that is given to the user is 6 characters long and made up of numbers, lowercase letters, and uppercase letters. This results in the number of unique codes being practically unlimited so it is incredibly unlikely that two users would have the same code.

An improvement could be to run the code generation algorithm on the device, which means part of the device ID is never sent over the internet, however, this would allow someone to potentially send their own malicious in place of the real one to impersonate another user.

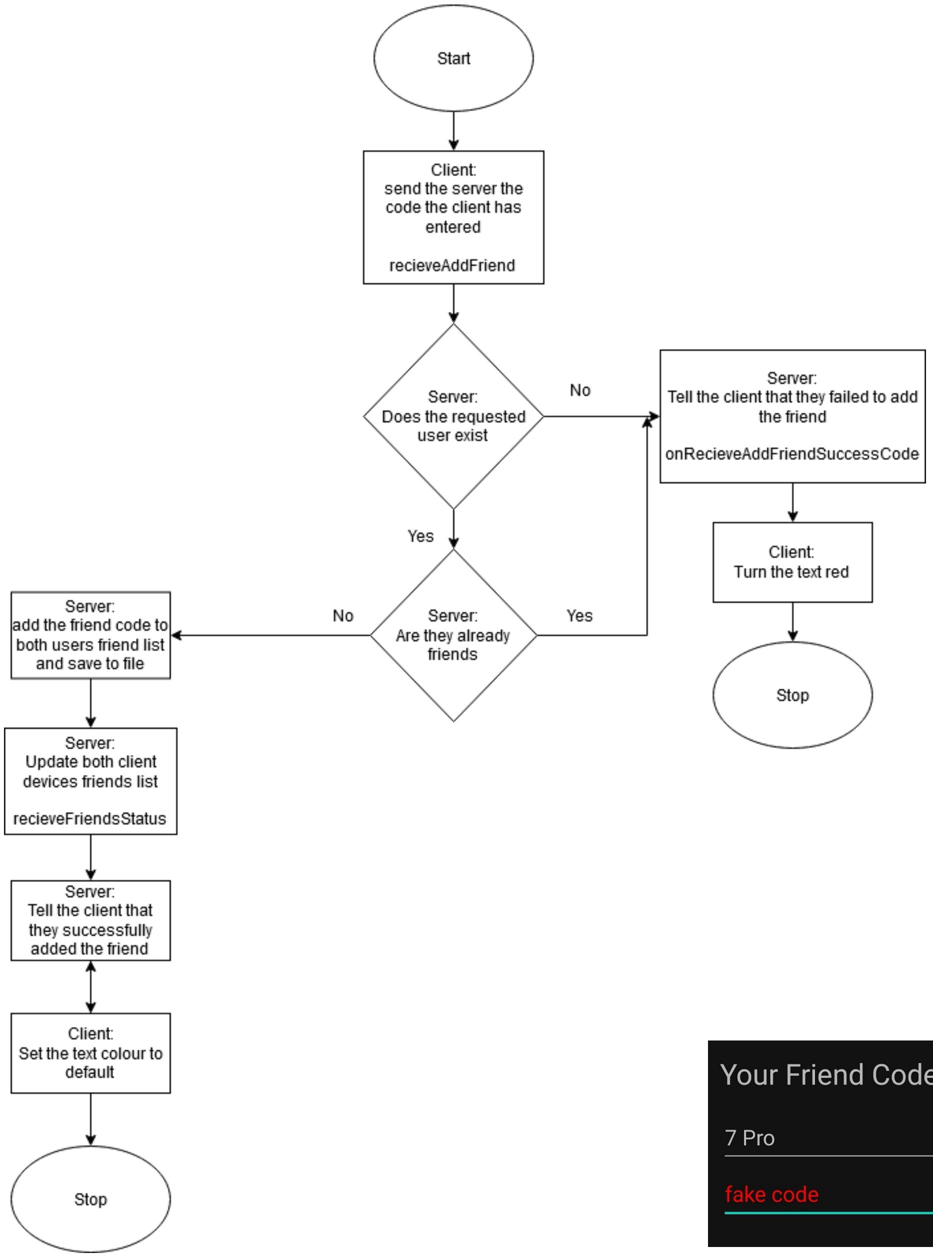


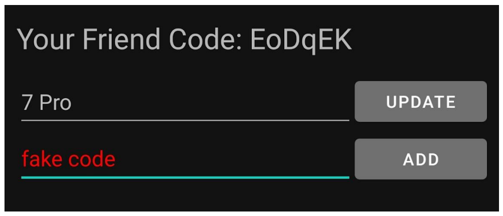
```
static generateUserCode(deviceID){
    let values = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
        "a", "b", "c", "d", "e", "f", "g", "h", "i",
        "A", "B", "C", "D", "E", "F", "G", "H", "I",
    let code = "";
    //032236f7
    let chars = [];
    let combChars = [];
    //get the code values for each character
    for(let i = 0; i < deviceID.length; i++){
        chars.push(deviceID.charCodeAt(i));
    }
    //combine pairs
    combChars[0] = chars[0] + chars[2];
    combChars[1] = chars[1] + chars[3];
    combChars[2] = chars[2] + chars[4];
    combChars[3] = chars[3] + chars[5];
    combChars[4] = chars[4] + chars[6];
    combChars[5] = chars[5] + chars[7];

    for(let i = 0; i < combChars.length; i++){
        code += values[combChars[i]%values.length];
    }

    return code;
}
```

(Fig 7. Unique ID generation algorithm)

2.3) Friend System

The friend system is the backbone of the whole app because everything passes through it. To add a friend all that is needed is another user's ID. If the code is valid the server will add them to their friend list, however, if the code is invalid the app will notify the user by turning the code text red.


Start
Client: send the server the code the client has entered
recieveAddFriend
Server: Does the requested user exist
No
Yes
Server: Tell the client that they failed to add the friend
onRecieveAddFriendSuccessCode
Client: Turn the text red
Stop
Server: Are they already friends
No
Yes
Server: add the friend code to both users friend list and save to file
Server: Update both client devices friends list
recieveFriendsStatus
Server: Tell the client that they successfully added the friend
Client: Set the text colour to default
Stop

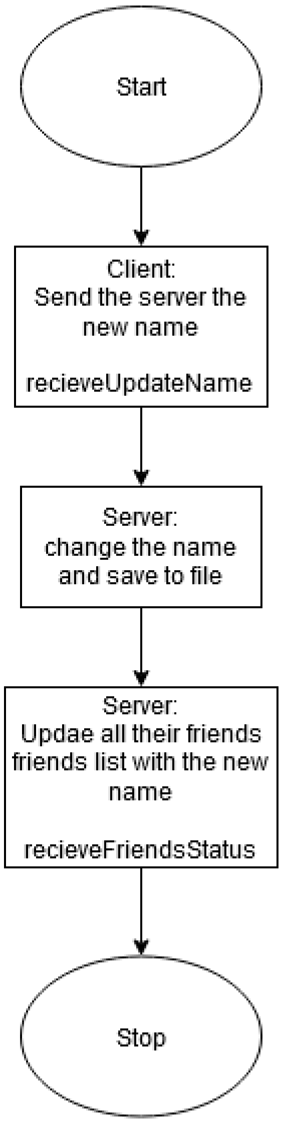

Your Friend Code: EoDqEK

7 Pro

fake code

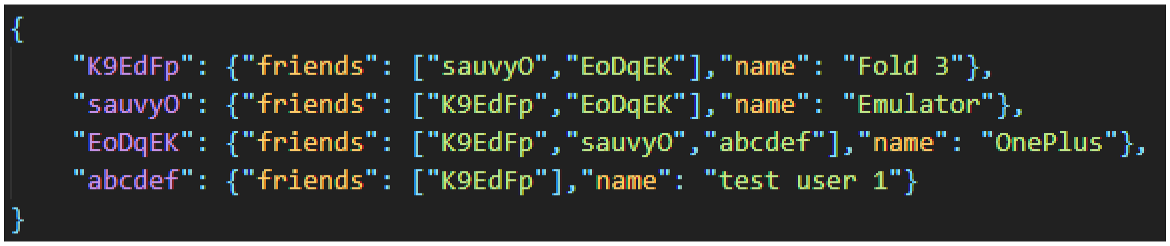
(Fig 8. Data flow diagram for adding a new friend) (Fig 9. Entering an invalid code makes text red)

A feature of the friend system is that it lets the user change their name. This works by taking the text the user inputs and simply changing the name field in the user file on the server, so that way it is immediately updated for all other users. One way this could be improved is by checking the names for malicious or offensive text. This would be very easy to imploment because it just needs to check against an existing blacklist of words.


Start
Client:
Send the server the new name
recieveUpdateName
Server:
change the name and save to file
Server:
Updae all their friends friends list with the new name
recieveFriendsStatus
Stop

(Fig 10. Change name data flow diagram)

All the data for each user is stored on the server in a JSON format. This makes it very easy for the server to load and save user information. This could be improved by using a database with relationships because it would result in better performance when there are lots of users connected, however, a simple JSON file works perfectly on this small scale.

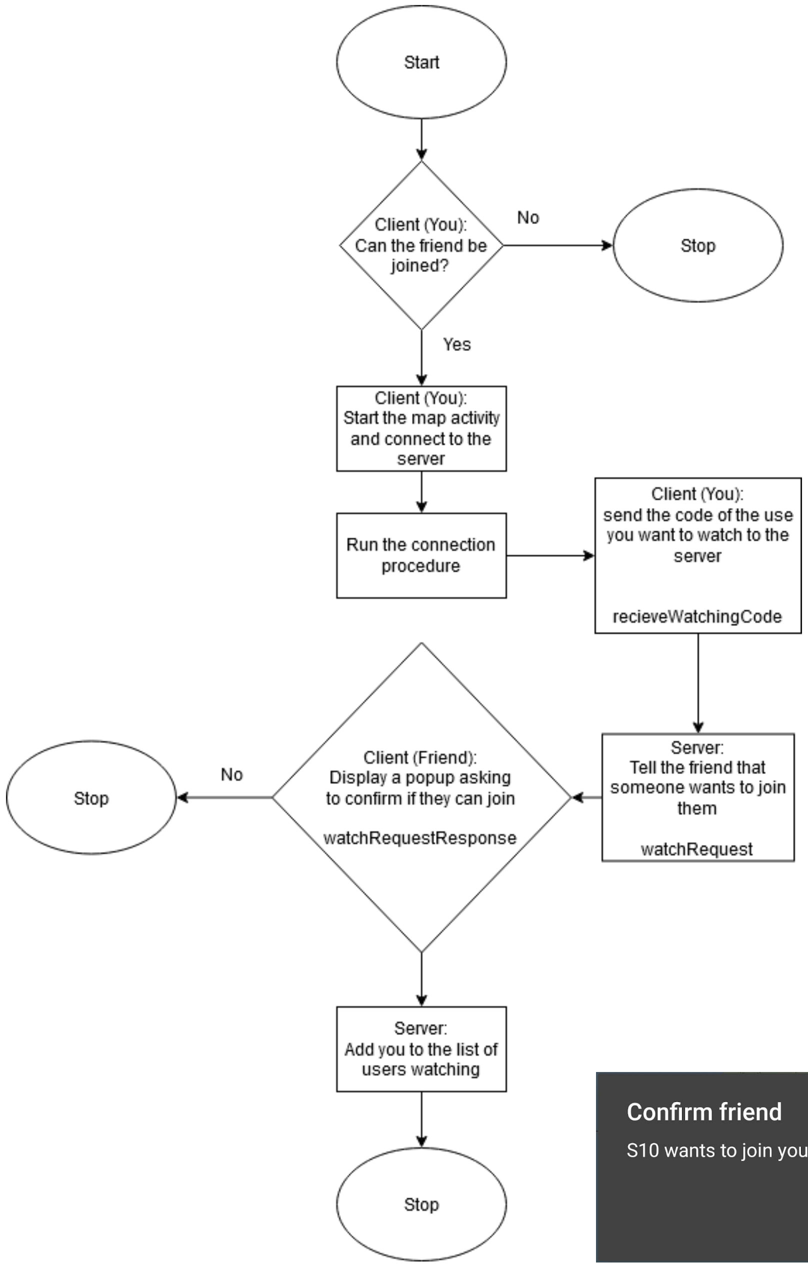


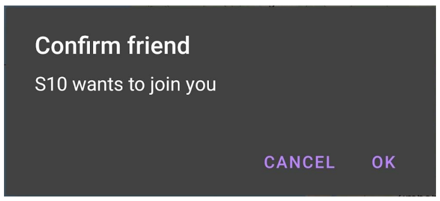
```json
{
    "K9EdFp": {"friends": ["sauvyO","EoDqEK"],"name": "Fold 3"},
    "sauvyO": {"friends": ["K9EdFp","EoDqEK"],"name": "Emulator"},
    "EoDqEK": {"friends": ["K9EdFp","sauvyO","abcdef"],"name": "OnePlus"},
    "abcdef": {"friends": ["K9EdFp"],"name": "test user 1"}
}
```

(Fig 11. JSON file showing the user data)

2.4) Map Screen/Tracking

When the client connects to the server there is a check to see if they are on the friend screen or the map screen. If they are on the map screen their status is so other users know they can be viewed. When another user tries to join the recipient is set a popup asking them to confirm the connections. This confirmation is an important security measure because without it anyone that knows your code could connect without your knowledge.

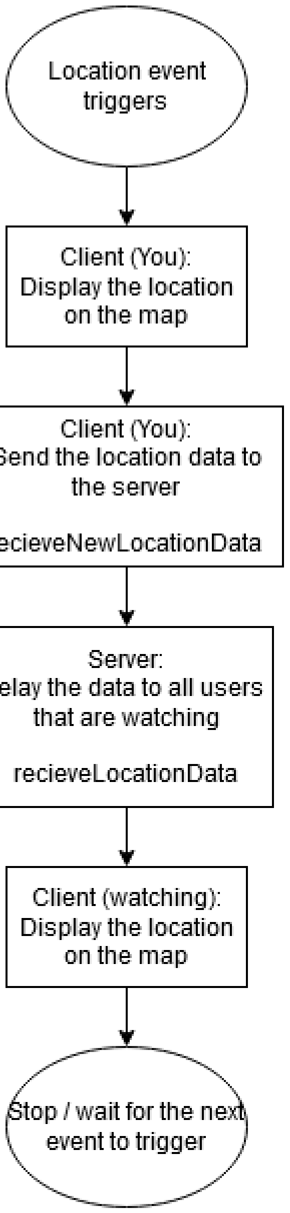

Start
Client (You): Can the friend be joined?
No
Stop
Yes
Client (You): Start the map activity and connect to the server
Run the connection procedure
Client (You): send the code of the use you want to watch to the server
recieveWatchingCode
Server: Tell the friend that someone wants to join them
watchRequest
Client (Friend): Display a popup asking to confirm if they can join
watchRequestResponse
No
Stop
Server: Add you to the list of users watching
Stop


Confirm friend

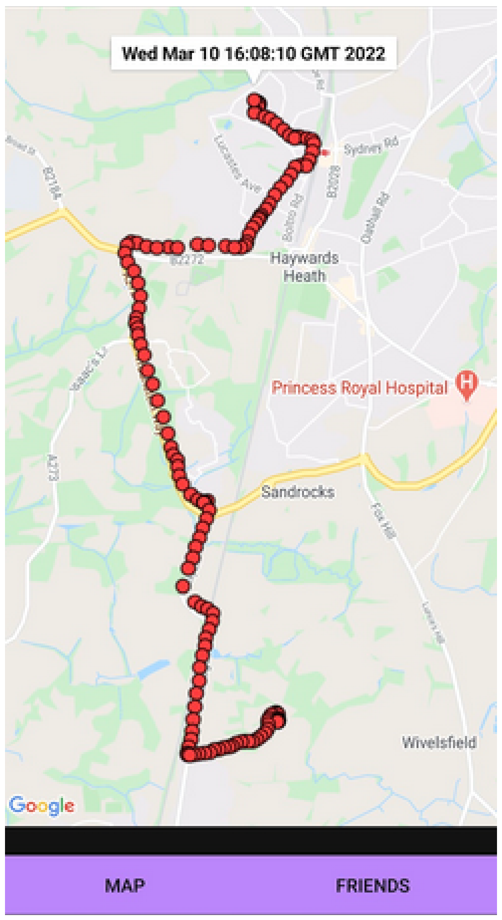
S10 wants to join you

(Fig 12. Data flow diagram for joining a user) (Fig 13. Confirm popup)

When watching a friend you can see their GPS position in real-time on google maps. The friend is sending their location to the server which then distributes it to everyone that is watching. Having the server send the location to the server was done to reduce the amount of bandwidth that is required from the friend because instead of sending it to everyone watching they only have to send it once, to the server. This could be important when high in a mountain and the cellular signal is low. It also acts as a security layer because there is no direct communication between users, all communication is done via the server.


Location event triggers
Client (You): Display the location on the map
Client (You): Send the location data to the server
recieveNewLocationData
Server: relay the data to all users that are watching
recieveLocationData
Client (watching): Display the location on the map
Stop / wait for the next event to trigger

(Fig 14. Data flow diagram for sending location data)



(Fig 15. Showing a tracked trip)

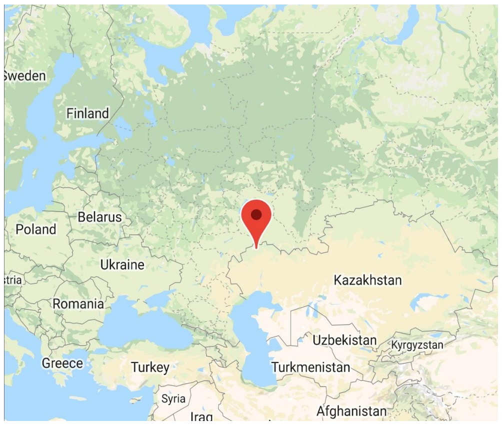
3) Testing

Throughout the development process, I was testing each component.

I encountered a few major errors. Firstly, the client wouldn't connect to the server. I spent around 2 days diagnosing this, but the solution was extremely simple. The client version of Socket.Io was slightly newer than the server. Once I had made them the same it worked as expected.

Secondly, when attempting to update all friends lists the server would crash if a user has logged on and gone offline. This occurred because the client never sent the message to the server saying the user had closed the app which caused the server to try and connect to something that doesn't exist. The solution was simple because all I had to do was add a disconnect event.

Finally, the most annoying error issue, which took me 4 to 5 days to resolve. The issue was that the location showed fine on the friend's device, but was wrong on anyone watching. This was happening because I had accidentally mixed up the latitude and longitude so after switching then it was working perfectly.



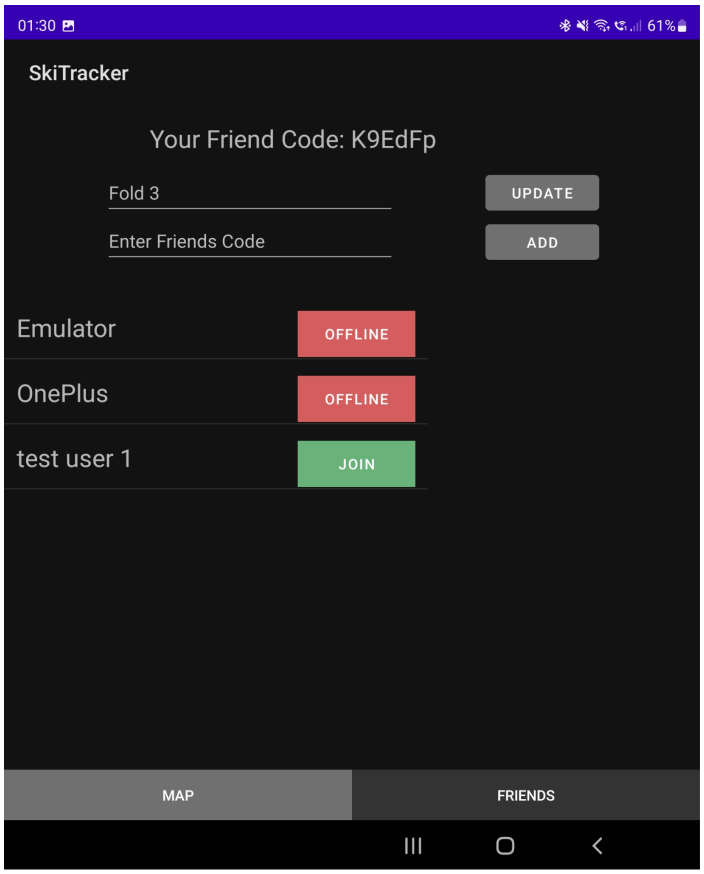
(Fig 16. Showing the incorrect location)

There is currently one issue to which I cannot find a solution. When asking the user to give consent for device permissions nothing happened so they are required to go into the settings and manually accept them.

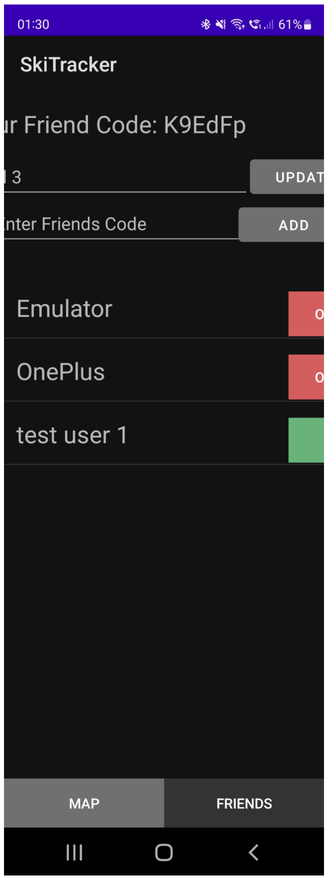
Due to this being an android mobile app the variety of devices it could be installed on is very large. The app was tested on multiple devices with different screen sizes. Firstly, the OnePlus 7 Pro has a slightly taller screen than most phones so on the map screen there is a black bar under the map, this is a minor issue and is not very concerning because everything else functions as intended.

Secondly, the Pixel 3a has a more traditional screen size and all the interface elements matched perfectly.

Finally, the Samsung Galaxy Z Fold 3 has 2 screens. First, the outside screen is incredibly tall and narrow which results in content being cut off on the left and right. This is an issue because the content is not visible and the buttons are harder to press. The inside screen is similar to a tablet which makes gives lots of empty space. Although on the outside screen the app is hard to use and has an extremely bad user experience it is not much of a concern because devices like this are far from ideal and they are not widely used.



(Fig 17. Fold 3 inside screen)

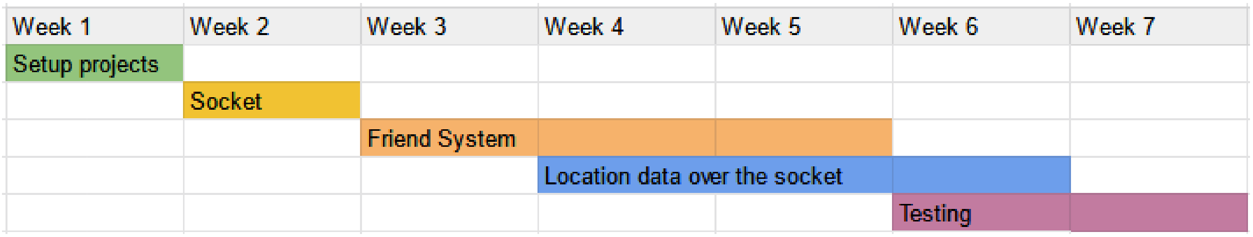


(Fig 18. Fold 3 outside screen)

4) Project Management

4.1) Time Management

In order to properly plan my time, I created a Gantt chart. Using this was extremely useful as it allowed me to see predicted start and finishing dates for each feature so I can see if I was ahead or behind schedule.



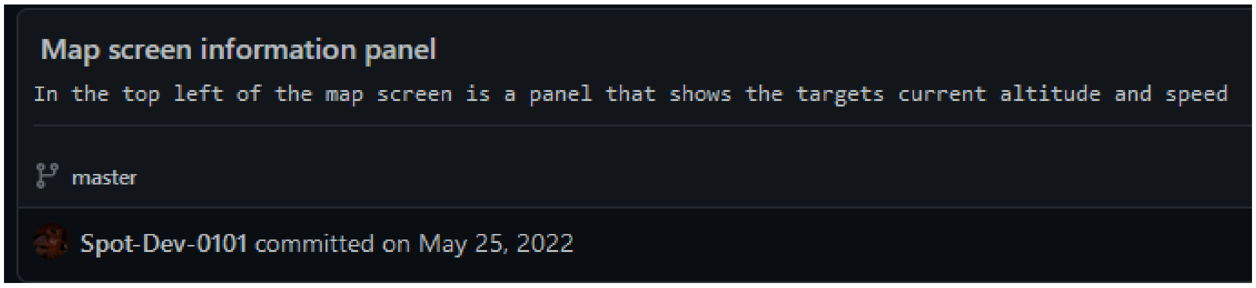
Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7
Setup projects						
	Socket					
		Friend System	🟧	🟧		
			Location data over the socket	🟦	🟦	
					Testing	🟪

(Fig 19. Gantt chart)

4.2) File Management

To manage the files for this project there is really only one option, and that is to use git. Git is perfect for this because it allows me to keep a record of every change and lets me revert back to a previous version if something goes wrong. Whereas, using a normal cloud storage service I would lose all my files if something goes wrong.

I made sure to use the GitHub feature that lets me write a description for each commit to provide even more information about each change.



Map screen information panel

```
In the top left of the map screen is a panel that shows the targets current altitude and speed
```

master

Spot-Dev-0101 committed on May 25, 2022

(Fig 20. GitHub commit with expanded information)

5) Conclusion

Overall I am very happy with the final product. The brief states that the app must have two advanced features which are satisfied by the inclusion of Google Maps and remote server communication. Additionally, I have achieved all the “must” and “should” requirements, as well as one of the “could” requirements.

When coming up with the initial timeframe I drastically overestimated how long it would take to complete, as well as the order in which I would have to develop components. For example, I thought it would take 2 weeks to set up the server connection, but it only took 2 days, and I thought I could do location over the socket before creating the friend system, but I needed the friend system to enable location over the socket. I also estimated it would take 8 weeks to complete the whole app, but it only took 5 weeks of 1 hour of work per day.

If I was to redo this project or expand it I would have focused on the visual design more because it is very simple and doesn’t look amazing. If I had put as much effort into the design as I did into the backend it could have looked very good and been much more pleasing to use.

6) References

Android Developers. (2020). *Documentation | Android Developers*. [online] Available at: https://developer.android.com/docs

Arrachequesne, D. (2015). *Native Socket.IO and Android*. [online] Socket.IO. Available at: https://socket.io/blog/native-socket-io-and-android/

Socket.IO. (n.d.). *Get started*. [online] Available at: https://socket.io/get-started/chat.

Google Developers. (2019). *Get Started | Maps SDK for Android | Google Developers*. [online] Available at: https://developers.google.com/maps/documentation/android-sdk/start.

Stack Overflow. (n.d.). *android - How to change the color of a button?* [online] Available at: https://stackoverflow.com/questions/32671004/how-to-change-the-color-of-a-button

Stack Overflow. (n.d.). *java - How to get current location in Android*. [online] Available at: https://stackoverflow.com/questions/17591147/how-to-get-current-location-in-android