Project Report
Catch me if you can

https://mobut-project.web.app/

DM2518 Mobile Development with Web Technologies

Oscar Rosquist, Alessandro Iop, Hannes Runelöv, Ramtin Erfani

May 2020

Contents

1 Introduction

In these times of confinement and quarantine due to the spread of COVID-19, many opportunities for meeting with friends and have a fun time together have gone amiss. Especially during the longer, sunny days of spring, a time of the year when enjoying the outdoors seems the best activity, sacrificing such a relevant habit of our social-being is very difficult, and we all hope to get back to a normal life as soon as possible. With that in mind, for this project we wanted to develop something aimed at the future, even if not the most immediate one: giving people an opportunity to strengthen their bonds of friendship while spending time outdoors and exercising by walking long distances was our personal goal in the past weeks. In addition to that, our aim was to bring a classic game to a more challenging and exiting level, by relying on people's more competitive side; that is why we developed a scaled-up version of the popular game of *Tag*, extending it to (potentially) the size of an entire city.

In our game, after signing in, players start off by joining a lobby, from which they are redirected to the main view after a minimum amount of players has been reached and a countdown has expired. The main view of the game consists of a geographical map displaying an arrow placed on the player's current position and centered on it; the arrow points to the position of the opponent that the player got assigned as a target. Moving around in the surrounding geographical area, and eventually "tagging" the target when it is close enough, eliminates the opponent from the game and triggers the re-shuffle of all the players remaining. The last player standing wins the game.

2 Project overview

Provided that one of our aims was to use as many technologies presented during the lab sessions as possible, we came up with several features that would implement the mechanics of our game, as listed here below, and proposed them in the initial project specification:

- A lobby where players could see each other and join or exit from freely;
- Active GPS tracking to get each player's position during the game;
- An arrow placed a the map, pointing to the target player, with different colors according the distance;
- Real time sharing of position data between players using Firestore, which we however ended up implementing though PubNub instead, as we deemed it to fit better for this purpose;
- A tag button that appears when the player is close enough to the target;
- A visual indication of how close the person hunting the player is, a feature that was ultimately left out due to time constraints and more urgent debugging duties to be done;
- Random reassignment of the target once a player gets eliminated, which resulted in a complete re-shuffle of the players left in the game, instead of just a single player's re-assignment;
- Display final scores and/or leaderboards at the end of the game, ultimately left out for the same time reasons explained above.

Another change of the original specification made during the development process were the addition of a login mechanism through Google authentication, a simpler but more effective solution to user's temporary profiling compared to a custom authentication.

2.1 Time frame

We began planning and working on this project on April 7, 2020 and presented it on May 14. During that month, we had multiple meetings together as a group, before starting the development phase individually, splitting the main components of the project amongst ourselves. After the first components were built, we assembled the skeleton of the application and tested it (the map component came in a bit later). From there on, we started always working in groups of two, three or four people with increasing frequency. From meeting on Discord a couple of times a week, we ended up gathering almost every day towards the end of the project, especially when bugs and obstacles started arising.

The initial scope of the project may have been a bit ambitious as it included more than what we had the time to develop. Most of the features described in the original project specification were however included in the final result. As mentioned before, our initial timeline from the original project specification were followed somewhat accurately. The parts that took longer time than initially expected were implementing the game mechanics and a synchronised game play between users. The User Interface was another part we did not expect to take as long as it did, due to what we thought was a lack of documentation of Onsen UI for React components. Appearance and aesthetics of the UI were however worked on according to the timeline and then postponed until the more important bugs were fixed.

3 Technical overview

3.1 Firebase

We decided to use Firebase [2] to store user data as well as handle Authentication of users. The reason we chose to use Firebase was mainly how quick its setup is, as well as how easy it is to use. Thanks to the Firebase SDK, we were able to handle login through Google authentication. The API provided by the SDK came out to be easy to use and quick to integrate with. For storing data we employed a Firestore database to collect users' email addresses, names, their Google profile pictures and unique IDs, all in a JSON format. These attributes were all fetched from each player's Google profile when logging in. Additionally, Firebase was also used for hosting the application, which was deployed to Firebase Hosting through its CLI.

3.2 Onsen UI

We employed Onsen UI [5] to build the user interface of our mobile web app. Onsen UI provides support to multiple web frameworks as well as vanilla Javascript. We decided to use its React.js components for this projects since it was our chosen framework, and we had perceived it as easy to employ from experimenting with it during the lab sessions. We had, however, previously used just its vanilla Javascript components, and noticed, hile working on the project, that the documentation for React.js was not thorough/complete.

3.3 PubNub

The idea of introducing PubNub [6] as a technology for relaying every player's position in real time during the game came after realising that updating a user's position attribute in the Firestore documents would take too much time, as fetching it from another client would. In addition, using this technology was one more opportunity to apply the knowledge gained during the lab sessions of the course. If we did not relay all players' positions as quickly as possible to each other, we would certainly end up having synchronization problems, especially when they're close to each other or they're moving fast.

3.4 React.js

Since two of the group members were already familiar with the React.js framework [7], and choosing another one would have meant that everyone had to learn it from scratch (a potential cause of disorientation for the whole group), we agreed to use React as main infrastructure for our project. Nevertheless, all group members experienced difficulties in managing props propagation and state update in React, for example when passing player information from the database management script to the main game component. The reason we ended up spending a relevant amount of time on implementing a working data flow is that some services we used (e.g. Firestore and Google Maps) returned promises that have to be resolved before updating each component's inner state and then displaying the correct content in the UI. Only by properly synchronising all components we were able to make the game mechanics work as intended, but many obstacles arose in the attempt.

It ended up being fairly easy to adopt for a mobile web app, all the while keeping his power that would perhaps have gone missing if we decided to use React Native. Considering that we were developing a web app, the only real concern during the design phase was making it responsive to all possible platforms. On the other side, it is still accessible from a desktop environment, and this flexibility adds value to it.

3.5 Google Maps

Considering that the game is entirely based on the use of a geographical map, the Google Maps API [4] that we learned during the first lab session was an obvious choice for implementing its main view, where the map is displayed full-screen and is augmented with information about each player's own and target positions. An entire branch of the project was therefore dedicated to adding it to a React component that also combined GPS tracking and computation of the relative distance between players. The API provided a solid starting point for the later implementation of the game mechanics as well as for the update of the app UI as the game progresses.

4 Implementation

4.1 Authentication

In order to implement authentication in our project we used the Firebase authentication API. The Firebase SDK provides an authentication function that enabled us to authenticate a user through Google OAuth, which is an authentication handled on by Google servers. When a user is successfully authenticated we will receive a token with an expiration date. The token is then set to invalid if the user decides to sign-out or if the expiration date cease. In order to keep track of the user sign-in/sign-out state we used a observer-function. The observer-function verifies that a user is signed-in if the user data is retrievable from Firebase [1, 3].

The application also utilises the single sign-on functionality that is supported by the Firebase SDK. This is used to keep the user signed in until they manually click the log out button or clear their browser data. This will prevent the users from having to always log in. Since we are expecting them to play on their phone, it would be the same user most of the time. This continuity of their session should therefore be very useful. Even if the users accidentally close the web application, their current state in the lobby will be persisted. The view of the app before signing in is shown in Figure 1.

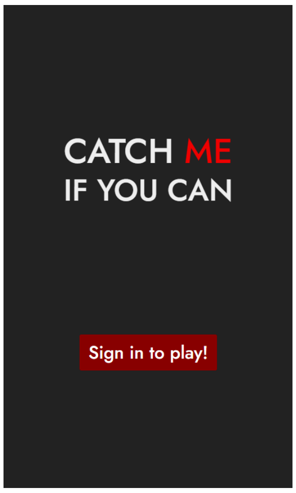


Figure 1: Homepage of the web app before a user signs in.

4.2 Lobby component and view

Once the user has logged in, and only then, will they see the the lobby (Figure 2). The lobby consists of the Onsen UI Toolbar component which displays a Google profile picture and the name of the logged in user as well as a button to logout. Below the toolbar the user will see a list of players that are currently in the lobby as well as a floating action button to press that will add the user to the lobby. The lobby component will be listening to any changes made to a collection called *lobby* in the Firestore database. The *lobby* collection contains the players that are in the lobby as separate documents. By having the lobby component listen to any changes made to that collection, it will trigger a function in the app that makes the component render again if the players in the lobby change. Since the Firebase SDK provides this listening API, we do not have to continuously poll the database for changes and can instead let the database notify the clients when changes occur. When there are 2 or more players in the lobby, a countdown will start and be displayed at the top of the screen as a Onsen UI toast, showing the time left until the game starts, and it is reset whenever a new players joins in.

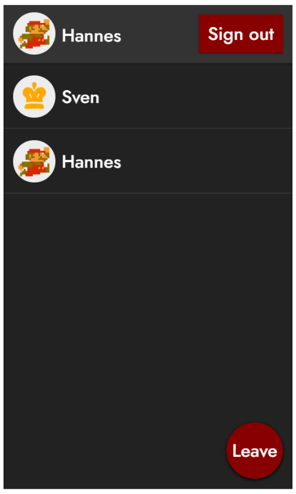


Figure 2: UI of the game lobby, presented after the user signs in.

The countdown functionality was not as easy to implement as we first believed. Since the countdown triggers the move of players from the *lobby* collection to the *game* collection described in section 4.3, the clients that had not yet been moved would still act on any changes in the *lobby* collection as described above. This would occasionally cause non-desirable behavior such as players not getting redirected to the game because there were now to few players in the lobby because almost all of them had been redirected.

4.3 Game component and view

The game component will be shown to the users in the lobby once the countdown reaches zero. It comprises of a full-screen Google Maps view. At the start of the game the players that were in the *lobby* collection will be moved to a collection in Firestore called *game*.

This move of players proved to be more challenging then we first thought. We have each client move themselves from the *lobby* collection to the *game* collection. Since all clients listen to changes in the collections, as described in section 4.2, they will act on each of the players moving themselves between the collections. As an example, any event that triggers on changes in the *lobby* collection therefore needs to know if it's the game starting or simply a player moving in and out of the lobby.

Immediately after starting the game, the user will see a loading screen built from the Onsen UI modal component. This loading screen will be showed while the Google Geolocation API fetches the users position and until the user have had their target players location assigned through PubNub. Once this data has been loaded, the map will be centered on the users location where an arrow will be placed. This arrow will change its color depending on how close the user is to the target player. The closer the distance, the warmer the hue will be. Displaying the arrow at the center of the map and making it point in the direction of the target required a more customized solution compared to the standard implementation of marker object with the Google Maps API. Instead of using a regular svg image as the marker's icon, we resorted to defining it as a specific path string so that we could later on update its orientation whenever either the player's, the target's or both positions changed.

Once the user and its target is within a distance of 10 meters of each other, the arrow will be replaced by a tag button, allowing the player to tag the other. Once a player is tagged, they are displayed with an Onsen UI toast message telling them that they lost and then they are redirected to the lobby view. If there are multiple players left in the game, everybody gets assigned a new random target to chase. The consequence of this is that one player can be hunted by multiple players, but such mechanism keeps the game interesting and unpredictable. If there is only one player left, however, that player will see an Onsen UI toast message saying that they won. Throughout the game, the clients will broadcast their locations to each other using PubNub whenever they change (i.e. when the players move), thus keeping the player target location of each player up to date.

The three stages of the game – i.e. loading, chasing and tagging – are all presented in Figure 3 as mobile screenshots.

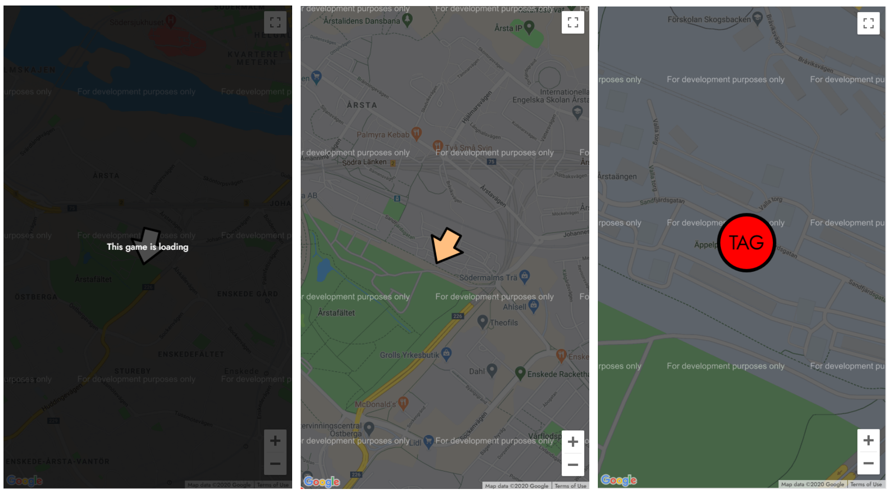


Figure 3: From left to right: loading screen displayed before the game starts; main map view, where a white arrow placed on the player's current position indicates that the target is far away; main map view, where a red tag button replacing the arrow indicates that the target is within ten meters of the player.

4.4 Synchronisation of user data

In order to keep track of the user's coordinates we used the Google Geolocation API. This API provides a function that listens on user position updates. Whenever the user position would change the updated location was broadcast through PubNub to all clients. This implementation enabled us to synchronize users in the game. Since we wanted to implement the game so that a user only could access the position of the player that they were chasing, we needed to filter the data that was broadcast. We Implemented a filter method that ignores data that are received from players other than the one being chased.

4.5 Firestore Component

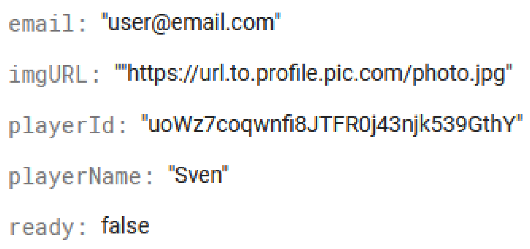
To handle the communication between the clients and the database, we developed a Firestore component to handle it. It will perform all communication that the clients need. Some examples are:

1. add player to the *lobby/game* collection;

2. remove player from the *lobby/game* collection;

3. listen to changes in the *lobby*/*game* collection.

The Firestore database uses a JSON-like structure when saving data to it. The entries in the *lobby* and *game* collections, also known as documents, contain the users' email addresses, their Google profile picture, their Google user ID as player ID as well as their Google profile name as the player name. We also added a flag that indicates if a player is ready or not, which later on became unused. The JSON structure can be seen in Figure 4. This structure made it very easy to work with the data in JavaScript as objects.



```
email: "user@email.com"
imgURL: ""https://url.to.profile.pic.com/photo.jpg"
playerId: "uoWz7coqwnfi8JTFR0j43njk539GthY"
playerName: "Sven"
ready: false
```

Figure 4: The structure of the Elements in the *lobby* and the *game* collections

5 Further development

5.1 Point system and leaderboard at the end of the game

Whenever a game ends and a player wins a toast is shown with a congratulation message. In order to make the app more interactive we thought that it would be interesting to have a component that displays user statistics and high-scores. In order to implement this we store user statistics in a Firebase collection.

5.2 Circular target assignment

The application currently assigns targets randomly. It would therefore be interesting to implement different targets strategies in order to see which one that is best suited for our application. One target strategy that we discussed but never implemented is to target players in a circular manner. In a hypothetical circular target strategy the idea is that person-1 chases person-2, person-2 chases person-3 and person-3 chases person-1. When a player eliminates their target from the game, they get the tagged player's target as a new player to chase.

5.3 Chat in the lobby

In order to make the application experience more social, we thought that it would be interesting to create a chat in the lobby. With a chat, users would be able to communicate with each other while they are waiting for other players to join. A possible way to implement a chat is to use PubNub.

5.4 Better looking UI

From the start of the project we wanted a good looking User Interface. We started out with a simple User Interface and then focused on the desired features and game mechanics. Because of the time limit and unforeseen issues while developing the functionality of the application, before the project presentation the User Interface was left behind a bit more than we had hoped for. After realizing that having a better looking UI would not only be aesthetically pleasing, but also improve the overall user experience, we decided to continue working on it after the presentation. Starting off with the homepage, the toolbar we had used before was not really needed, so we removed it and let the screen consist of a logo and the sign-in button. We also changed the color scheme to a dark theme with red details, and embedded a custom typeface (called "Jost") from Google Fonts. Moving on to the lobby screen, the color scheme was mostly carried over from the home screen with some minor adjustments out of necessity. We had to rearrange some elements though, as well as increase the font size and player icon size. We also added a circular background to each icon to make them appear more consistent.

Finally, we were also planning on styling the map. Although it looks adequate as it is, we wanted to implement a dark-style theme for the its view, in order to better fit with the overall theme of the web app and to make the compass arrow stand out even more. Furthermore, the arrow itself could use some better styling – with dark colors for the map, the arrow doesn't need a black contour to stand out. After implementing all the aforementioned other possible improvements, then, new elements of the interface would probably need additional styling as well.

5.5 Known bugs

As of now, the database has one lobby collection and one game collection. When a game starts, players are moved from the lobby to the game. This is a problem because the game does not handle multiple games at the same time. Players can join the lobby and be redirected to the game even thought the game has already started. A possible solution for this problem would be to enable users to create their own custom lobbies with unique collection names and once a game starts the lobby will be deleted from the database which will disable players to join the lobby at a later point in time.

6 Use cases

Here below two different use cases are presented as potential scenarios that cover all the features of our game. It can be used both by groups of players that already know each other, or by single users with the aim of making new friends.

6.1 Alice, Bob and Carl wants to play tag with each other

The corona epidemic is over. Alice, Bob and Carl are finally allowed to go outside. They all feel like they need to compensate for all the lonely hours that they spent at home. Since the three friends live far from each other, they decide to play a game of tag through the web application called "Catch Me if You Can".

After signing in, they all join the game lobby and 30 seconds later they are redirected to the game map. The only thing that is visible in the map view is a player marker that represent who they are chasing and an arrow that represents themselves. They can not see the name of the person that they are chasing. Alice and Carl are randomly assigned Bob as a target, while Bob is randomly assigned Alice. As they get closer to each other, the shade of their respective arrows transitions from white to an increasingly warmer color (e.g. yellow, orange, red). When Alice finds Bob, within a 10 meter radius, her arrow turns into a tag button. She immediately tags him and Bob is unfortunately eliminated and redirected back to the lobby. As Alice is distracted, celebrating her success, Carl suddenly pops up from nowhere and smashes the tag button like if it was life or death.

Carl wins the game! After remaining the only player standing, he is automatically redirected back to the lobby.

6.2 Donald wants to play a game of tag

Donald is new in town and has no friends. But Donald has a gift, he is particularly good at chasing people, hiding in bushes, and running fast. He stumbles across the application called "Catch Me if You Can" and realises that it is a perfect way for him to meet new friends. Donald signs in to the app with his new Google mail account and joins the game lobby. After 30 seconds he and several other players, from the lobby are redirected to the map view. In the map view Donald sees an arrow that is representing his current position and a marker that is representing the location of the player that he is hunting. Donald's instincts activate and he starts hunting. As Donald gets closer his arrow starts to change color. Its hue transitions from white to yellow, orange and finally red.

Finally, Donald's arrow is replaced by a tag button. Donald knows what that means; the target is close at hand. Skillfully, he hides in the closest bush, waiting for his prey to fall in his trap. But little does he know that his target is also chasing him. Donald gets tagged and therefore eliminated from the game; at first he is sad, but the other player comforts him and they become best friends. After all, Donald cannot complain.

7 Resources

You can find the code of our project in our GitHub repository:
`https://gits-15.sys.kth.se/hrunelov/mobut-project`

The game is deployed on the following website:
`https://mobut-project.web.app/`

References

[1] Firebase Authentication. `https://firebase.google.com/docs/auth`. Accessed: 2020-05-21.

[2] Firebese documentation for software developers. `https://firebase.google.com/docs`. Accessed: 2020-05-21.

[3] Google Authentication for software developers. `https://developers.google.com/identity/protocols/oauth2/service-account`. Accessed: 2020-05-21.

[4] Google Maps documentation for software developers. `https://developers.google.com/maps/documentation`. Accessed: 2020-05-21.

[5] Onsen UI 2 homepage. `https://onsen.io/`. Accessed: 2020-05-21.

[6] PubNub documentation for software developers. `https://www.pubnub.com/docs`. Accessed: 2020-05-21.

[7] React.js homepage. `https://reactjs.org/`. Accessed: 2020-05-21.