SPORTSMATE:

FIND YOUR SPORTS PARTNER ONLINE

by

JIANRUI HU

A report submitted to the faculty of

The University of Utah

in partial fulfillment of the requirements for the degree of

Masters

in

Software Development

School of Computing

The University of Utah

December 2018

TABLE OF CONTENTS

Acknowledgments…………………………………………………………………………………4

Exist Application……………………………………………………………………….………….5

Class Design……………………………………………………………………………………….9

Lay Design……………………………………………………………………………………….12

Firebase……………………………………………………………………………….….……....18

Result…………………………………………………………………………………………….20

Conclusion………………………………….……………………………………………………23

Reference…………………………………………...………………………………...………….24

LIST OF FIGURES

Figure 1.1 RacketPal Screenshot……………………………………………………………….....6

Figure 1.2 Playo Screenshot………………………………………………………………………8

Figure 2.1 User Class in firebase cloud storage …………………………………….…………...10

Figure 2.2 Ticket Class in firebase cloud storage…………………………………….………….10

Figure 3.1 overall layout workflow………………………………………………………….…..12

Figure 3.2 welcome page………………………………………………………...………………13

Figure 3.3 login page…………………………………………………………………………….13

Figure 3.4 sign up page………………………………………………………………….……….13

Figure 3.5 profile create/edit page……………………………………………………………….14

Figure 3.6 home page………………………………………………………………….…………14

Figure 3.7 profile page…………………………………………………………………………...14

Figure 3.8 ticket page………………….…………………………………………………………17

Figure 3.9 post page…………………………………………………………………………...…17

ACKNOWLEDGMENTS

I like sports; however, most of the sports are require two or more people. Sometimes it becomes pretty hard to find people to play with, special with someone who has closer skill. So, I want to build an android application that helps me find their sports partner.

Here are the features that I am looking for in my app:

1. Sports search/match
2. Location search/match
3. Shill level search/match
4. Schedule search/match

To achieve those features, each user can post their ticket, which contains sports type, skill level, location, and time. All the users can view the currently exist ticket and pick the ticket that you want to join. In this way, people who don't know each other can play sports together.

EXIST APPLICATION

Before I start to design my application, I went to the Google Play Store and search for the existed that can provide similar features.

**Basketball:**

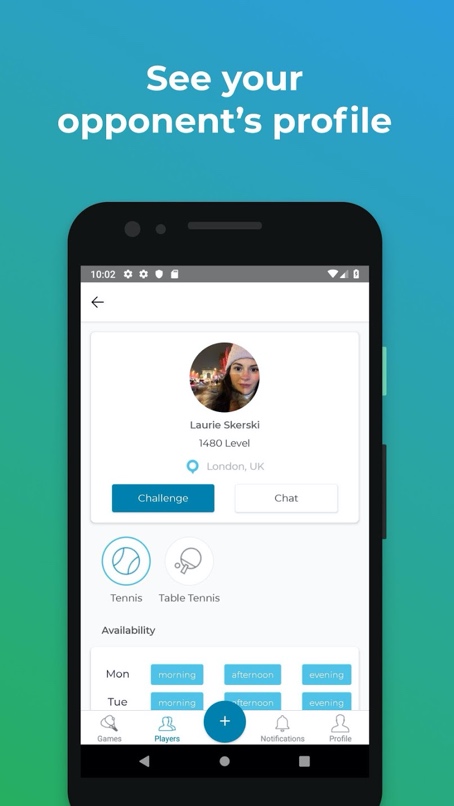
First, I try to find the applications that can help people to find partners to play basketball together. However, there is no such application on the Google Play store. I think the reason for that is most of the basketball court are indoor, and usually there is a lot of people playing basketball all the time. So, people can just come to basketball anytime and not worry that there are not enough people to have a game. Another reason is that basketball is not like other sports. Even you are the only one in the basketball court, you can still play basketball alone, like practice shooting skills.

**Tennis:**

Next, I searched for the application the help people find tennis partners. In the Google Play Store, there is more than ten different application that can help people to search for tennis partners. Here I will choose an app called "RacketPal: Find local racket sports partners today” (<https://play.google.com/store/apps/details?id=com.racketpal>). This application not only helps people to find partners in tennis, but it also helps people find partners in Badminton, Table Tennis, Squash, and Padel. Base on the information that provides by Google Play Store, there is more than one thousand people have installed this application. Here are some screenshots for this application (See the figure on the next page). From the screenshots, we can easily know that the app somehow gives everyone a level number. Base on ether, you won a game or lose a game with other users in this application, you will increase or decrease the level number. In my opinion, people should more enjoy sports itself, but not beat someone else. I do agree that win or lose part of the sport's charm, however, to measure in a number skill is not the right way. It will make people only looking for the player who has a lower level number than their selves. Also, it is not a good thing to know the people that you play with are better than you based on the application level number. So, in my application, I do want to have some way to measure the skill level for each sport, but I won't use the system like this level number. I am more willing to use “beginner, intermediate, advanced, professional” to range all the users. In this way, people can play with the people that have a similar skill level, but no idea who is better than others. I believe people will more enjoy sports in this way.

A screenshot of a cell phone

Description automatically generated A screenshot of a cell phone

Description automatically generated 

A close up of a computer

Description automatically generated A screenshot of a cell phone

Description automatically generated A close up of a computer

Description automatically generated

*Figure 1.1 RacketPal Screenshot*

**Badminton:**

When I search for the application that helps people find badminton partner, I found an app called "Playo - Find Players, Book Venues, Manage Groups” (<https://play.google.com/store/apps/details?id=com.techmash.playo>). I believe it is the most popular application that provides this feature. Base on the information that provides by Google Play Store, there is more than five hundred thousand people have installed this application. This application not only includes partner searching for badminton, tennis, and basketball, it also provides partner search for other fifty-nine different kinds of sports. Here are some screenshots for this application (See the figure below and the next page). From the review and the number of people using this application, we can know that this application is excellent. This application not only provides the feature to find sports partners, but it also has a lot of built-in social features that help people communicate with each other. It will bring users closer and easy to become friends. However, I want my application to be simple. We already have lots of apps that provide communication, such as Facebook, WeChat, Line. There is no need to have built-in social features in this application. If they want to become friends, they can have a connection in other more popular communication application.

A screenshot of a cell phone

Description automatically generated A screenshot of a cell phone

Description automatically generated A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated A screenshot of a cell phone

Description automatically generated A screenshot of a cell phone

Description automatically generated

*Figure 1.2 Playo Screenshot*

CLASS DESIGN

The main idea for my application is that every user can submit their ticket online. And each ticket will contain sports type, skill level, location, date, time, and the people that involve in the ticket. At home page, everyone can look at the existing tickets and pick the ticket that they are interested in. In this way, people can join the game together.

Two main classes are using in this application, one is the User class, and another is the Ticket class. The User class will store the data for each user like a profile. The Ticket class will store the data for each ticket. In this way, the tickets can easily search by the user and filter out to be displayed.

**User Class:**

There are eight different variables in User class, there are

1. private String id;
2. private String name;
3. private String gender;
4. private String zipCode;
5. private String badminton;
6. private String basketball;
7. private String tennis;
8. private List<String> ticketID;

The string id is the key to this entire class. This id is generated by the firebase Authentication, which I will talk about later (See Page X). All the information for this user is store at the firebase cloud storage under this key, which is the string id. By using this string id, the user is allowed to have access to their user information. They can either change their profile and write to the cloud storage or read their profile from the cloud storage. String name, gender, and zipCode are working as their naming, their store the user's name, gender, and zip code. String badminton, basketball, tennis is working slightly differently. They represent the skill level for these three different kinds of sports. There are four different skill levels that you can choose; they are "beginner, intermediate, advanced, professional." The List<String> ticketID contains all the ticket id that created by this user. In this way, the post section (See Page X) can easily show all the tickets that involve by the current user. In this way, the application doesn't need to search the entire database to find all the tickets related to this current user.

In all these eight different variables, string name, gender, zipCode, badminton, basketball, tennis are the front end information, which will show in the android application to the current user. The string id and List<String> ticketID are the back end information which will not show in the android application to the user. They are the information that helps the app to extract other information. Here is how the User class looks like in the firebase cloud storage (See the figure below).

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated

*Figure 2.1 User Class in firebase cloud storage Figure 2.2 Ticket Class in firebase cloud storage*

**Ticket Class:**

There are seven different variables in Ticket class, there are

1. private String id;
2. private String sports;
3. private String level;
4. private String zipCode;
5. private String address;
6. private String date;
7. private String time;
8. private List<String> userID;

The string id is the key to this entire class. All the information for this ticket is store at the firebase cloud storage under this key, which is the string id. By using this string id, the user can extract the ticket information. More importantly, the user can search through all the tickets by this string id. This id is combining by the current timestamp and the current user id. When a user is creating a new ticket, the id for this ticket will be current DATE + TIME + USERID. If you look at the figure 2.2 Ticket Class in firebase cloud storage, an example will be “2019106124143710KxjofP3KWZdb5E62iiklQaP4r853”. When this ticket created, the date was 2019 Oct 6th; the time is 12:41:43 and 710 milliseconds. The rest string is the user id. The reason I add the user id to the ticket id is that if we have a huge amount of users for this application. There will have a high chance that creates conflict when people try to create new tickets at the same time. Adding the user id to the end of the ticket id can easy fix the conflict. Because one user can never submit two tickets online at the same time.

String sports, level, zipCode, and address are working as their naming, their store the sports name, skill level, zip code, and address. String date and time are also working as their variable names. They represent the event date and time when this game will setup. The List<String> userID contains all the user id that involve by this ticket. In this way, the list section (See Page X) can easily keep track of how many people have joined this event. In this way, the app doesn't need to search the entire database to find all the users related to this ticket.

In all these eight different variables, string sports, level, zipCode, address, date, time are the front end information which will show in the android application to all the user. The string id and List<String> userID are the back end information which will not show in the android application to the user. They are the information that helps the app to extract other information. Here is how the Ticket class looks like in the firebase cloud storage (See the figure above).

LAYOUT DESIGN

The overall workflow is shown in the figure below.



*Figure 3.1 overall layout workflow*

**Welcome Page:**

When the user opening the application, I will first load a welcome page (See figure 3.2 below). The welcome will stay for display for 3 seconds to 5 seconds. The reason that I build a welcome page at the very beginning of the application is that I am using firebase (See page X) as my back end. It usually takes about 10 to 20 seconds to build the connection between the application and firebase cloud storage. Rather than spend time waiting on the login page, I separate the waiting time into two parts. A little bit of waiting time on the welcome page and some waiting time on the login page. In this way, the user experience will be much better than waiting in the login in page for 10 seconds. Here is what my welcome page looks like.

A close up of a logo

Description automatically generated A screenshot of a cell phone

Description automatically generated A screenshot of a cell phone screen with text

Description automatically generated

*Figure 3.2 welcome page Figure 3.3 login page Figure 3.4 sign up page*

**Login Page:**

In the login page (see figure 3.3 above), the users need to enter their email address and the password. All that information is stored in the firebase Authentication; all the information is encrypted and hidden for everyone. Even as the administrator for this application and severe, like me still can't know the user's password. I can see all the user's emails and delete their accounts, but cannot achieve their password. It benefits the server and user's information security. Firebase Authentication will auto-check for the password; if the password matches the server, the application will jump to the home page and show a "successful login" task at the bottom of the screen. If the password does not match the server, the application will stay on this login page. The email will remain what the user has typed, but the password part will wrap out and become blank. At the same time, it will show a "connection fail. Please try again” task at the bottom of the screen.

A screenshot of a cell phone

Description automatically generated A screenshot of a cell phone

Description automatically generated A screenshot of a cell phone

Description automatically generated

*Figure 3.5 profile create/edit page Figure 3.6 home page Figure 3.7 profile page*

**Sign-Up Page:**

For the people who are using this application the first time, they will need to click the "sign up" button to jump to the sign-up page (See figure 3.4 above) to create their account. It requires their email address and password. Because everything typing in the password will not be displayed. To make sure the user is typing what they want without any mistake, I request a second time to enter the password. Only the two passwords enter match each will allow you to create the new account. Otherwise, it will show a "the password doesn’t match" task at the bottom of the screen and wrap out the second line of the password to make it blank.

**Profile Create/Edit Page:**

After the signup, the application will jump to the profile creation page (see figure 3.5 above). The user will enter their name, gender, zip code, badminton skill level, basketball skill level, and tennis skill level. Submit button will grab all the information that the user has entered and using that information to create a new User class. And then, send this class to the firebase cloud storage (See page X) and jump to the home page (see figure 3.6 above). If you already have an account, after login, you will also jump to the home page.

**Home Page:**

There are three features on the home page (see figure 3.6 above). First is the list recycler view for all the ticket, second is the pick button, and the last one is the navigation bar.

The list recycler view can be scrolling up and scrolling down. It will display all the tickets after the time filter and the skill level filter. The time filter will filter out all the tickets that are already expired. Users are only interested in the game that will happen in the future by not the game that already passed. At the same time, the skill level filter will choose the game that matches your profile skill level. For example, if you set your badminton skill to "intermediate" in the profile, then it will only show the badminton ticket, which is also intermediate. In this way, the application will help the user to ignore the extra information that they don't need. It will help the user looking through all the tickets easily and find the ticket that matching their schedule.

The list recycler view is selectable. Users can select each ticket by clicking it. And then, clicking the "pick" button below will add the current user to the ticket that you chose. Technically, it will get the ticket id that you are selected and pull out all the information from this ticket class. By clicking the "pick" button, it will add your user id to the List<String> userID variable under this ticket class. In this way, the user has joined the game that they want. Last, at the very bottom of the screen, there is the navigation bar, which helps the user to jump into different pages.

**Ticket:**

In each ticket, there are six pieces of information showing up. On the left side is an image that will represent the sports type. The reason I am using the picture rather than the text words is that people absorb information from the picture is faster and easier than from the text words. On the right side, it shows the event date, time, skill level, the number of people is currently involved in this event, and address. The date, time, skill level, and address are directly extracted from the ticket class. The number of people involved in this event is calculated by the length of the member variable List<String> userID inside the ticket class.

**Navigation Bar:**

The navigation bar will display all the time on the very bottom of the screen. There are four buttons on the navigation bar, home, profile, ticket, and logout. Each button will help the user to jump into different pages. The "home" button will take the user to the home page (see figure 3.6 above). The "profile" button will take the user to the profile page (see figure 3.7 above). The "ticket" button will take the user to the ticket page (see figure 3.8 below). And the "log out" button will allow the user to log out of their account and jump back to the login page (see figure 3.3 above).

**Profile Page:**

The profile page (see figure 3.7 above) shows all the user information that they entered when they create the account. The data, including their name, gender, zip code, badminton skill level, basketball skill level, and tennis skill level. At the bottom of this page, there is an "edit" button. By clicking this button will allow the user to update their profile information, so it will take the user to the profile create/edit page.

**Ticket Page:**

The ticket page (See figure 3.8 below) is pretty similar to the home page. There are also three sections on this page. First is the list recycler view for all the tickets that involve the current user. Second is the post button, and the last one is the navigation bar. In this list recycler view, there are also two filters. One is the time filter, which will be the same as the time filter on the home page. It will filter out all the tickets that are already expired. Another filter is the user id filter. This filter will searcher through all the tickets and pick the tickets that involve by the current user. The tickets can be either created or joined by the current user. Below the list recycler view is the "post" button, which will bring the user to the posting page.

**Post Page:**

The post page (See figure 3.9 below) is how the user can create a new ticket online. On this page, it will ask the user to enter the sports type, skill level, zip code, date, and time. The application will check each link to make sure that all the input is valid. After clicking the "submit" button, the app will use that information to create a new ticket class and send it to the firebase cloud storage. Now the new ticket will show on both the home page list recycler view and the ticket page list recycler view.

A screenshot of a cell phone

Description automatically generated A screenshot of a cell phone

Description automatically generated

*Figure 3.8 ticket page Figure 3.9 post page*

FIREBASE

Firebase is a Backend-as-a-Service (BaaS) developed by Firebase, Inc. in 2011. Google acquired the company in 2014. Then the firebase grew up into a mobile and web application development platform on Google Cloud Platform. As of October 2018, the Firebase platform has 18 products [2], which are used by 1.5 million apps. In my application, I am using two features from the firebase. There are Authentication and Realtime databases.

**Set-Up:**

Before adding Firebase to my Android app, I need to create a Firebase project and then connect it to my Android app. When we create a new Firebase project in the Firebase console, we're creating a Google Cloud Platform (GCP) project behind the scenes. A GCP project is a virtual container for data, code, configuration, and services, and a Firebase project is a GCP project that has additional Firebase-specific configurations and services.

A Firebase project is the top-level entity for Firebase. It provides a Firebase configuration file (Android/iOS) or a configuration object (web) that needs adding directly to the local project.

For iOS, we need to add a “GoogleService-Info.plist” configuration file;

For Android, we need to add a “google-services.json” configuration file;

For the web, we need to add a Firebase configuration object for our initialization scripts;

In this case, we only need to add an android application to the firebase project. After adding the "google-services.json” configuration file to the "app/src/" folder, the app now is connecting to the firebase project that I created.

Next, I need to add Firebase SDKs to my application by implementation dependencies into the “build.gradle” file. After sync, my application will have all the environments that allow it to be interactive with the firebase. Now everything is set up and ready to go.

**Authentication:**

My application needs to know the identity of the user. Knowing a user's identity allows my app to securely save user data in the cloud and provide the same personalized experience across all of the user's devices.

Firebase Authentication provides backend services, easy-to-use SDKs, and ready-made UI libraries to authenticate users to my app. It supports authentication using passwords, phone numbers, popular federated identity providers like Google, Facebook, and Twitter, and more. In my case, I am using the password.

To sign a user into my app, it first gets authentication credentials from the user. The credential is the user's email address and password. Then, my application passes these credentials to the Firebase Authentication SDK. Firebase backend services will then verify those credentials and return a response to my application.

After a successful sign-in, the user can access their basic profile information and the data stored in the Firebase Realtime Database.

**Realtime Database:**

Firebase realtime database is a NoSQL cloud database. Data is synced across all users in realtime and remains available when the application goes offline. Data is stored as JSON and synchronized in realtime to every connection. All of my application users share one Realtime Database instance and automatically receive updates with the newest data.

My realtime database has two children. One is the user; another is the ticket. The user child will store all the user profile information, which is the user class that I created (see figure 2.1). The user class can automate format as a JSON file and push into the realtime database. Similarly, the ticket child will store all the ticket information, which is the ticket class that I created (see figure 2.2). The ticket class can also automate format as a JSON file and push into the realtime database. In this way, the user can extract and update the information from the realtime database back and forward.

The firebase makes up my entire back-end server. It provides the identity for the user and the data cloud storage.

RESULT

At this point, all the basic features had achieved in my application. The user can log in to the application, pose their ticket and pick other's tickets. In this way, a different user can join the same game.

**Auto Tickets Filter**

In the beginning, I design to autoload user information from the profile and filter the tickets form them. There will have two main filters in this part. One is the zip code filter, so the application will only show the tickets that match the user's zip code. Another filter is the skill level filter. It will filter out the tickets that don't match the user's skill level. As I wrote my application, I change the design guild a little bit. I use a zip code search bar instead of an auto filter, and also cut out the filter for skill level.

Instead of allowing the application to auto-filter that tickets by user zip code, I add a zipcode search bar inside the home page. In this way, the user can see the tickets from all the zip codes, and when it is necessary, the user can also search for a specific zip code. The reason that I made this change is that sometimes people may travel to other zip code to play the game if the game is more fit their schedule. A zip code search bar will give the user more freedom to pick the right tickets for them.

Similar to the zip code, I used to design an auto skill level filter that will filter out all the tickets that don't match their skill level. However, not all users like to play the game with someone who has a similar skill level. They might want to find someone who has a lower skill level and have some fun to relax. Or they might want to challenge themselves to improve their skills. I removed the auto skill level filter to make it possible. Another reason that I remove the filter is that the skill level I provide is not an accurate measurement. "beginner, intermediate, advanced, professional" are more like a self-define. What skill level you think you are might not actually what you are. Also, even users are good at self-define in skill level; these four categories still are a big range of people. The skill level in my application is only used for helping people to get some ideas about other's skill levels. But not for accurate measurement, as I said. Now the user needs looking through the tickets list to find the ticket with the skill level that they want.

**Address and Location Search**

I made a mistake when I first design the ticket class. I only include the zip code as a member variable in the ticket class but not the address. After I finish write the posted ticket and pick the ticket feature, I realized that I didn't include the address for each ticket. The users don't know where they should go to meet the people after they pick the ticket. I add a new member variable in the ticket class called "address." It stores the address as a string in the database.

In fact, latitude and longitude are the best way to identify the location. So far, in my application, the user can only search the location by zip code and then look at the address shown in each ticket. However, there are two disadvantages. One is that zip code is a range of the area. So, the user is unable to search for the ticket base on the distance. Another disadvantage is that zipcode next to each other might not next to each other in physical address — for example, the zip code 84102, 84104, 84108. From the zip code, we might think that 84104 represents the physical area between 84102 and 84108. However, 84102 area is between 84104 and 84108. In other words, the user can only search for the zip code that matches their zip code, or they know what zip that they want. By searching for the next or previous zip code doesn’t represent the physical are next to you.

So, adding a mile search that allows users to search for all the tickets that within some mileage is a very good feature to have. In order to achieve this feature, the application needs to send the address to the google map. Base on the address, the google map locates it and find the latitude and longitude. It sends the latitude and longitude back to the application so that it can store them in the database. Now the user now can search for the tickets within some mileage. This is a really useful feature that I would like to add to my application.

**Time Filter**

Another feature that I am missing is schedule searching. I only have a time filter that will filter out all the tickets that already pass the current time. I should add another search option on my home page that allows the user each for the ticket for a special date. The code will be pretty similar to the time filter and the zip code filter that I have right now. However, because of lacking time, I did import this feature. If I have time, I will add another search bar next to the zip code. And then using the same method to look through all the tickets and pick the tickets that have the same date that we are looking for and display them.

**Pop-out Ticket Details**

This is also another feature that I would like to add. Right now, each ticket only shows how many people are joined the game. The name for each user is also essential. So, my application needs to find some way to show the user's information that joined the game to other users. The solution that I think is using a pop-out page as the tickets instead of the selectable list view. Instead click the ticket to select it in the list view, when the user clicks the ticket, it will bring the user to the ticket page. In the ticket page if will have more detail information about this ticket including all the user's name that was joined this ticket. Furthermore, by click each user’s name, the current user can view their profile to know more about them.

CONCLUSION

Even though I achieve all the basic features that I want. I am still not very satisfied with my capstone. As the more features that I import to my application, I could have more time and add more features to make it better.

The capstone project is different from all the projects that I wrote before. All projects that we had in the class are already done by the professor and students many times. Professor can provide a clear guild and all the knowledge to achieve it. But in the capstone, we only know the problem and need to find a way to solve it by ourselves. I learn a lot of new skills and techniques during the capstone. Also, I find some weakness from myself.

The first thing is that I need to learn how to think about the problem more in a big picture. I have trouble with the design part. I don't think I design my capstone well. There is some feature that I should consider before starting to implement my application. I meet many problems during my implement. Many of them I should realize when I design my app. And those problems cost me lots of time to fix. Because I have to warp out what I have so far and many changes at a more fundamental level. Like the pop-up view, I talked before. If I want to change everything to the pop-up view, and then some of the features for the list view are not necessary anymore. Those are unnecessary work. Design is essential for writing an application. With more experience than I have, I will get improve on it.

One improvement that I already learn it that I should download the similar applications from the google store, rather than look at the screenshot and the user review. By really using those applications, I will have a more precise idea about what is good and what is bad. It will help me to shape my application. Another improvement might be asking people for suggestions. Ask people if they are the user, what feature they want to have in this application. This might also help me to shape my application.

I like my capstone project, and I learn a lot during design and implement it. With the experience of this capstone, I will do better in my next application. And pay more attention to the design part.

REFERENCE

1. Tamplin, James. "Firebase is Joining Google!". Firebase, Inc. Retrieved October 22, 2014.
2. "Firebase Products." Firebase, Inc. Retrieved October 31, 2018.