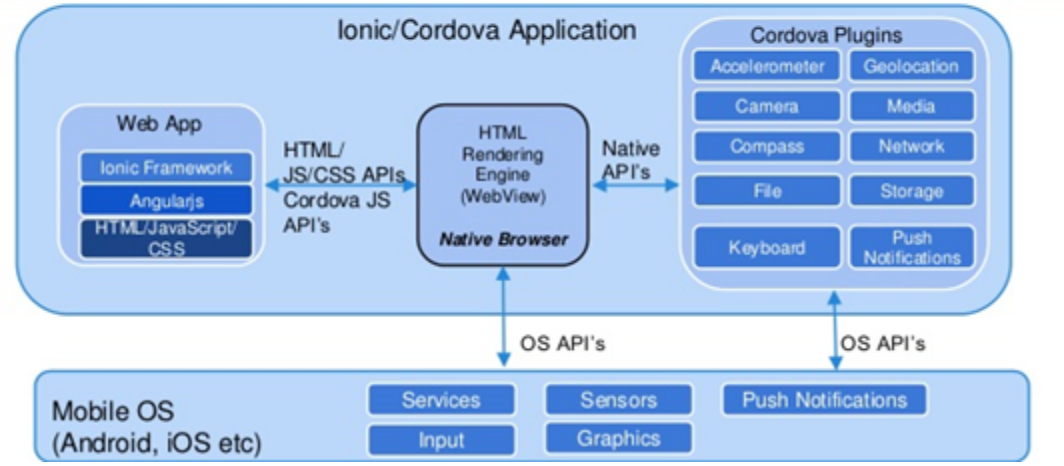
❖ Original Project Ideas

- Easy to find your race. Run Anywhere is a race management tool for runners.
- Connect with friends and share your adventure. Run Anywhere is the social network for runners. Record a race, where your friends and followers can share their own races.

❖ Concepts

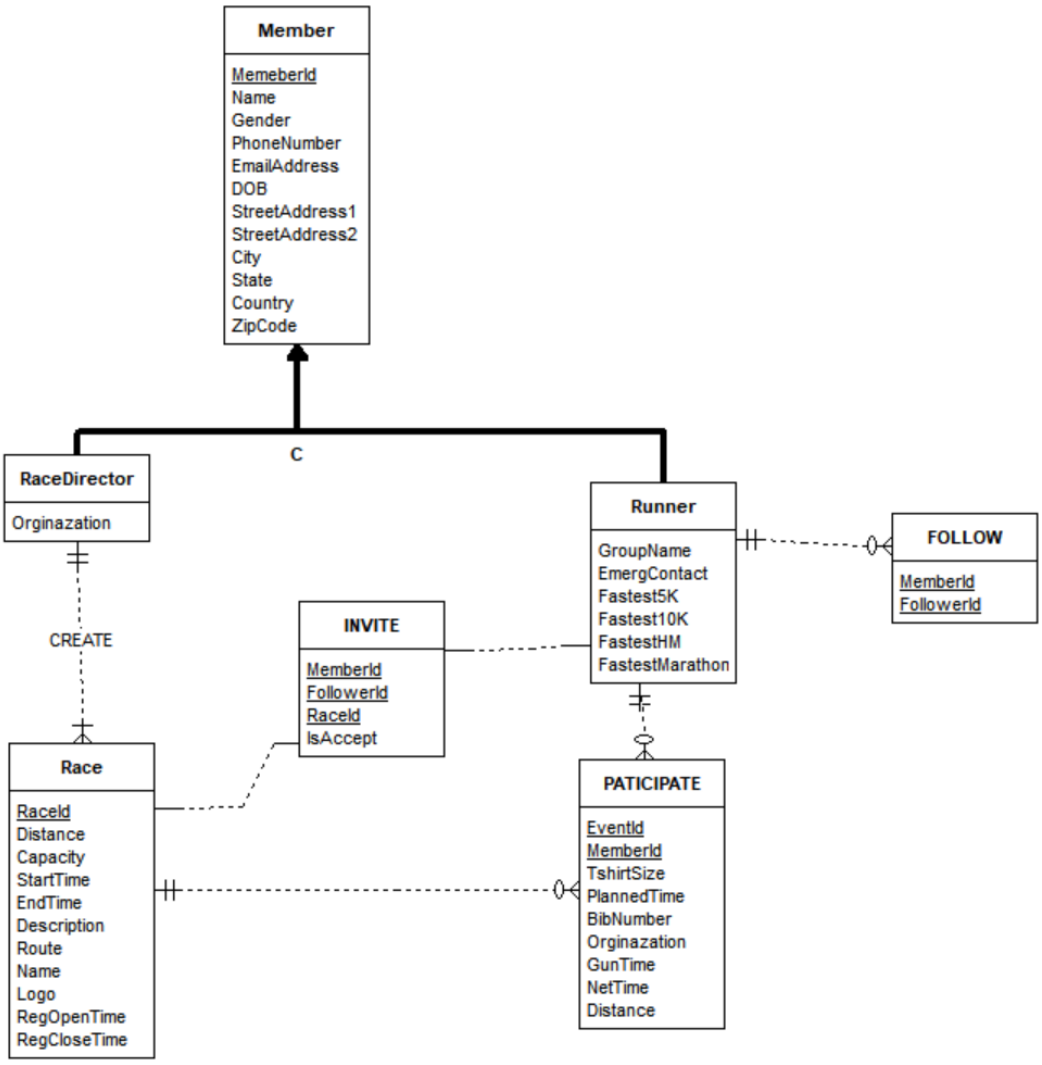
In our project, we use the following technology:

- Html/Javascript/CSS
- Cordova plugin(Geolocation, File, Camera)
- Third party API(Firebase, Google Map, MobileUI)



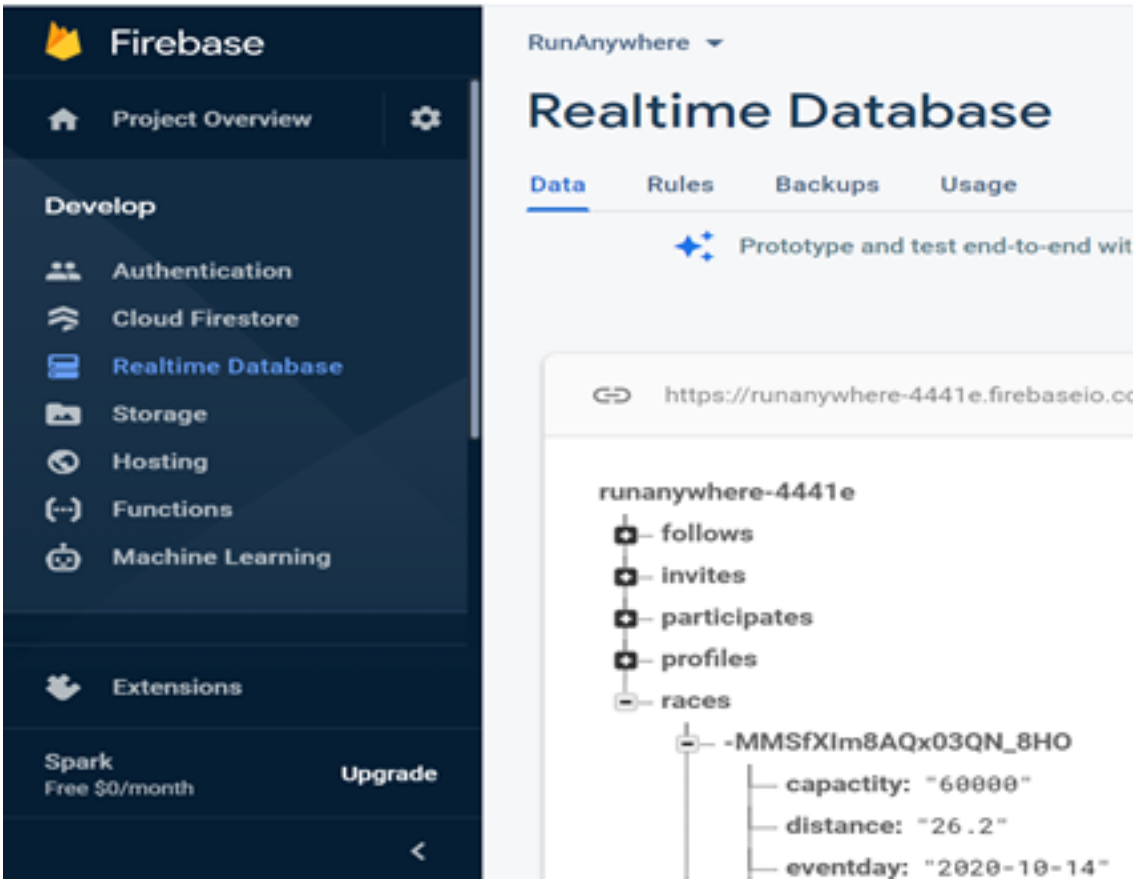
❖ Data Related

- EER Diagram



- DDL
  Refer to [Run Anywhere API Manual]

- Data Store
  Firebase offers a hosted NoSQL database. This data store is JSON-based, offering quick, easy development from webview or UI logic to data store. It syncs our app's data across multiple connected devices in a matter of milliseconds, and is available for offline usage as well. In effect, it provides an API for accessing and querying these JSON data stores in real-time for all connected users.
  We choose Firebase for our project. We have used firebase authentication, real-time database and storage in our app.



➔ Authentication
Authentication with Firebase provides various backend services and SDKs to help developers manage authentication for an app, so that we can easily implement login functionality using the industry standard OAuth 2.0 and OpenID Connect protocols. This service supports many different providers, including Facebook, Google, EMail.

```
firebase.auth.GoogleAuthProvider.PROVIDER_ID,
firebase.auth.FacebookAuthProvider.PROVIDER_ID,
firebase.auth.EmailAuthProvider.PROVIDER_ID,
firebaseui.auth.AnonymousAuthProvider.PROVIDER_ID
```

➔ Real-time database
The Real-time Database offers a hosted NoSQL data store with the ability to quickly and easily sync data.

We use a real-time database to store our main data: races, profiles, participants, invites, follows. We implement our UI/UX by callback function of reading and writing the data from firebase.

Read:

```
function getRaces(cb)
 {
        var query = firebase.database().ref("races");
        query.orderByChild("eventday").once("value", cb);
 }
```

Write:

```
var updates = {};
 updates['/profiles/' + newKey] = profileData;
firebaseRef.update(updates);
```

Remove:

```
var raceRef = firebase.database().ref("races/" + raceId );
raceRef.remove();
```

Actually, Firebase supports query weakly. It's hard to implement join, sort, multiple condition, and string matching. For example, when we want to get our followed people's activities information, we need to implement the logic on the mobile side. It's not efficient. In the future, we will change the database API library to improve this part of functionality.

```
function getFollowsActivities(cb)
{
    getCurrentUser( firebase.auth() ).then(function(user){
        if(user) {
            let uid = user.uid;

            firebase.database().ref("races").orderByChild("eventday").once("value", snapshot =>{
                snapshot.forEach( childSnapshotRace=>{

                    firebase.database().ref("follows").orderByChild("uid").equalTo(uid).once("value", snapFollow => {
                        snapFollow.forEach( childSnapshotFollow =>{
                            firebase.database().ref("participates").orderByChild("raceId").equalTo(childSnapshotRace.key).once("value", snapParticipate => {
                                snapParticipate.forEach( childSnapshotParticipate =>{
                                    if( childSnapshotFollow.val().followId == childSnapshotParticipate.val().uid )
                                    {
                                        firebase.database().ref("profiles/" + childSnapshotFollow.val().followId ).once("value",function(snapProfile){

                                            cb(childSnapshotRace, snapProfile, childSnapshotParticipate);
                                        });
                                    }
                                });
                            });
                        });
                    });
                });
            });
        }
        else redirectToLogIn();
    });
}
```

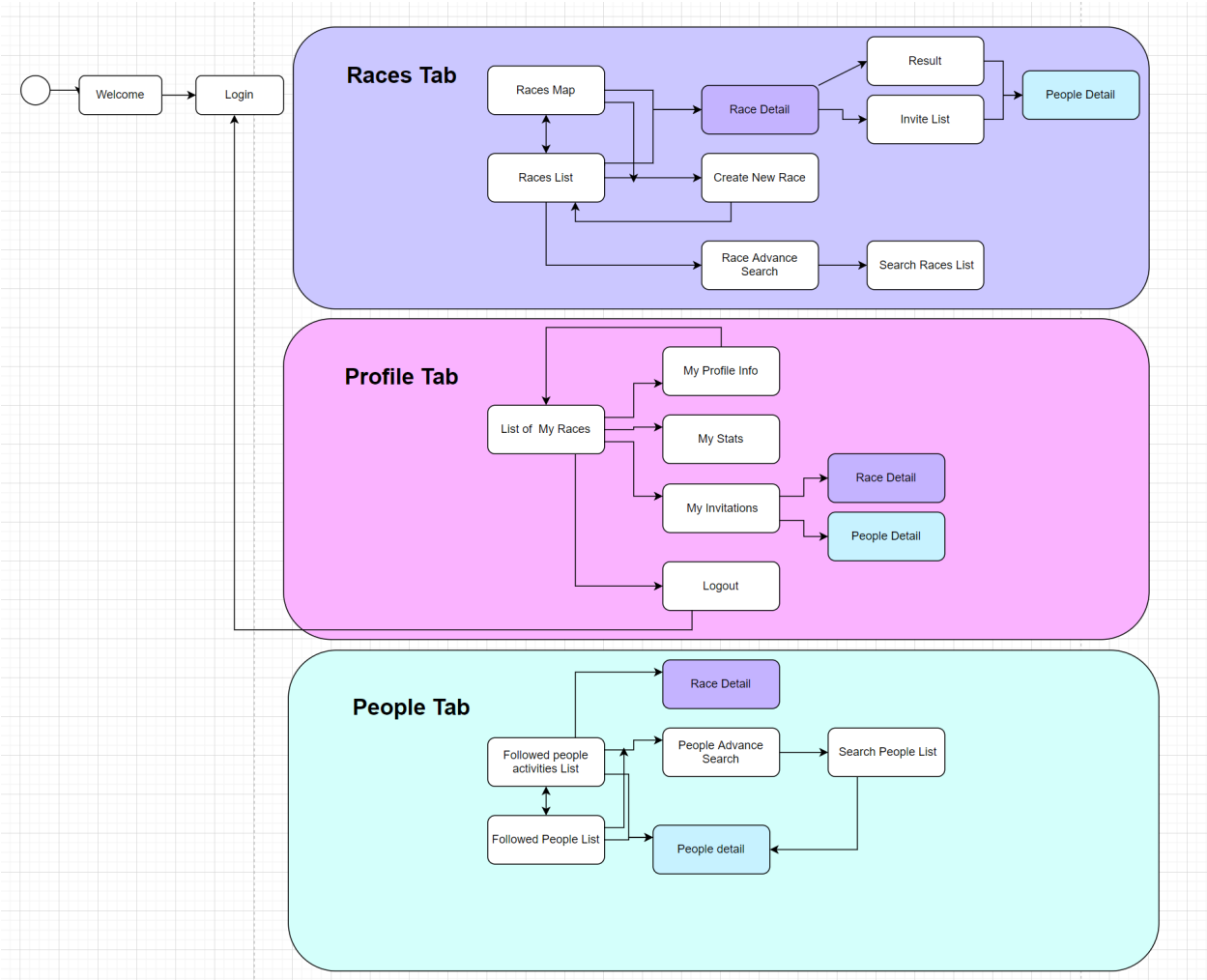
➔ Storage

Firebase storage supports upload, store, and download files. Files are usually stored in a Google Cloud Storage bucket, which means that they should be accessible using either Firebase or Google Cloud.
WE Use firebase storage to upload, store, and download pictures and GPX(.kml) files for our app.Different platforms may need to use different upload functions. In our code, we use putstring on browser platform, and use put on android platform.

```
if( platformId == "browser" ){
var uploadTask = firebase.storage().ref(filePath).putString( file,
firebase.storage.StringFormat.DATA_URL);
}else{
uploadTask = firebase.storage().ref(filePath).put(blob);
}
uploadTask.snapshot.ref.getDownloadURL().then(function(downloadURL) {
        cb(downloadURL);});
```

❖ Navigation Options

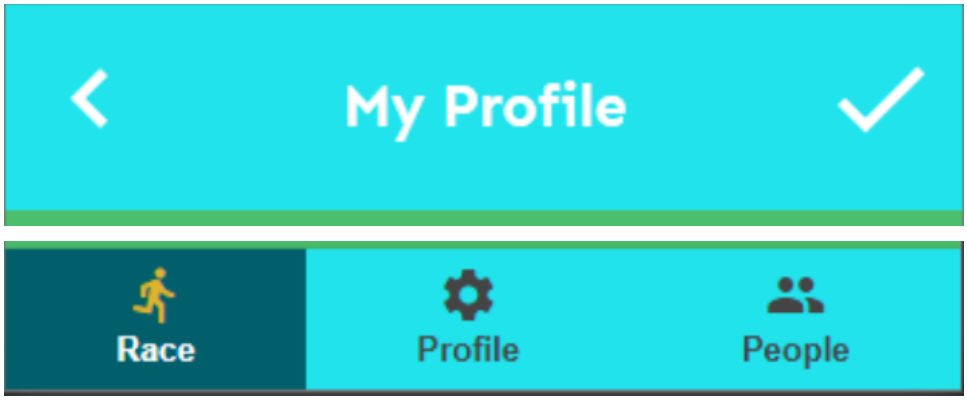
The following is our project UI Flow Diagram. Each box represents a page in the app. Each arrow represents jumping from current page to another page by click event.



❖ Consistency

- UI Consistency
  - ➔ Color Theme
    Color Palette:
    Logo: Blue (21E3EC, 100%) and Green (1BDD32, 100%) and black (000000, 100%)
    Background: (1BDD32, 70%)

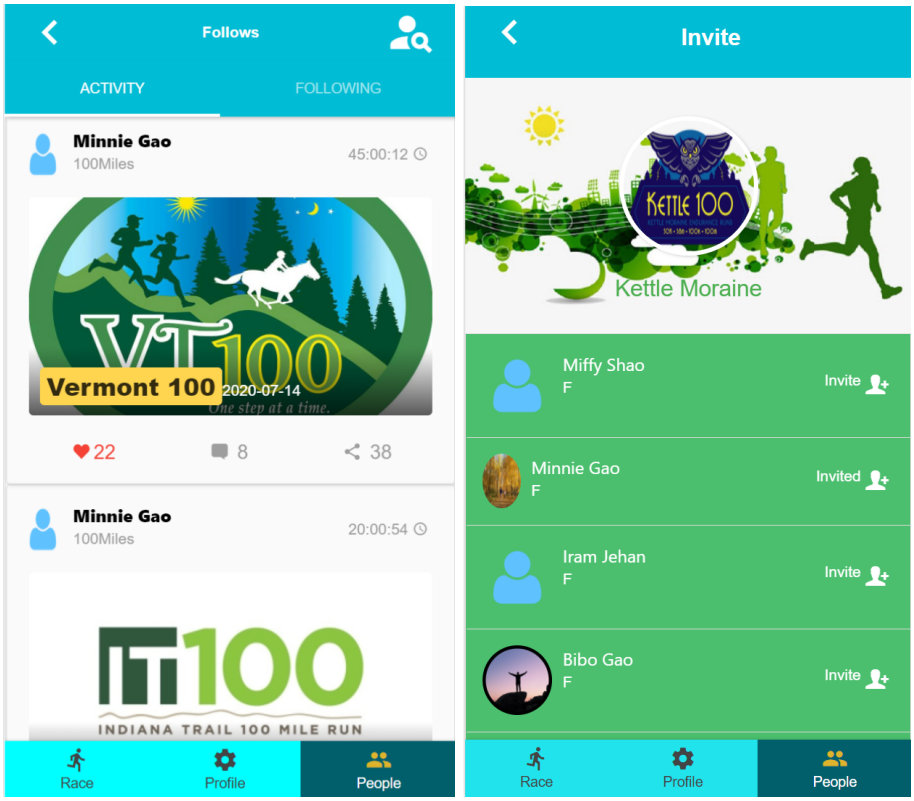
  - ➔ Header and Footer for each page



➔ Mobile UI

MobileUI provides tabs, side menu, stack navigation and tons of other components such as lists and forms. Some of the existing components are displayed differently on Android and iOS, with automatic styling that will change the appearance of the app based on the platform.

In our project, we use Swiper, Cover, Tab, List, Loading components provided by MobileUI to keep the page style consistency.



- UX Consistency

  ➔ Call back function

We use a callback function to get the current snapshot value for the data stored.

```
function getRaces(cb)
 {
  var query = firebase.database().ref("races");
  query.orderByChild("eventday").once("value", cb);
 }

var isLoading = true;
loading('Please wait...');
  getRaces( snapshot => {
    snapshot.forEach(function(childSnapshot) {
      var raceData = childSnapshot.val();
           races_list_html += '<div class="w3-cell-row">' +
                  '<div class="w3-cell" style="width:30%">' +
                   '<img class="w3-circle" src="' + raceData.imagePath + '"
style="width:64px;height:64px;">' +
                   '</div>' +
                   '<div class="w3-cell w3-container">' +
                   '<h3><a class="article-title" href="../races/race_detail.html?id=' +
childSnapshot.key + '">' + raceData.raceName + '</a></h3>' +
                   '<p>'+ raceData.eventday +'</p>' +
                   '</div>' +
                   '</div>' +
                   '<hr>';
    });
    races_list.innerHTML = races_list_html;
    if( isLoading ){
     closeLoading();
     isLoading = false;
    }
  });
```

➔ Loading(spinning)
We use the Loading component when we read data from firebase to prevent user interaction. The code is shown on above.

- Code Consistency
  We put common functions(data, map, storage) together as API to keep code consistency. There is still the same code in different files. Next step we can refactor it.

❖ Pattern

- Async : Promise
  In our project, we used the Promise function to fix the Async problem. The Firebase SDKs are that its APIs are all asynchronous. So that firebase.auth().currentUser sometimes returns to null because the current user object is obtained asynchronously. onAuthStateChanged always triggers with the initial state even when it is null. So we write the promise function in addition to onAuthStateChange that would eventually return the user but would return null if no user exists in persistence storage or user require sign in from expired token.

```
let userLoaded = false;
function getCurrentUser(auth) {
 return new Promise((resolve, reject) => {
   if (userLoaded) {
      resolve(firebase.auth().currentUser);
   }
   const unsubscribe = auth.onAuthStateChanged(user => {
     userLoaded = true; unsubscribe();
            resolve(user);
   }, reject);}); }
```

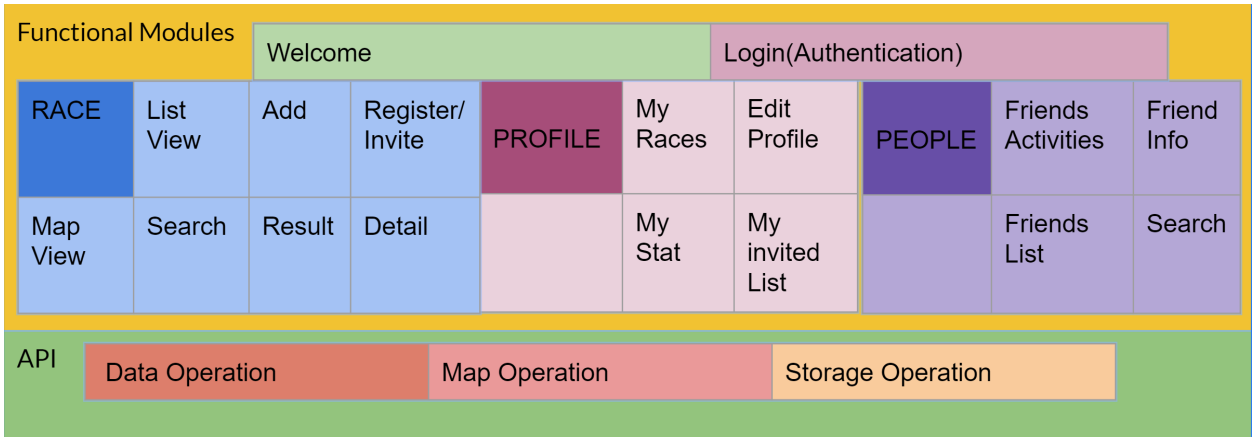
- Listener/Observer
  Firebase includes native support for Promises and associated chains. But we use listener pattern instead of using a Promise. As the listener is notified of updates to the online database, we need the callback function to be executed. In fact, we execute multiple times for many updates to the stored data. The callback may simply be executed each and every time there is an update to the DB.
  In our project, we connect and retrieve data once, and we also use the firebase on(“child_added”) method to set up a listener event that polls the DB for any changes in value. If we change any data in the online database, this listener will automatically execute the defined success callback function.

```
profilesRef.orderByChild("birthdate").startAt(peopleBirthMin).
endAt(peopleBirthMax).on("child_added", snap => {
```

```
        if( peopleGender=="" || snap.val().gender == peopleGender){
            cb(snap);
        };
    });
```

❖ App Module Architecture

The following is our app module architecture, including functional modules(Login, Race, Profile, People) and our API (Data, map, storage).



API detail Refer to [Run Anywhere API Manual]

❖ Cordova plug-in

In our app, We use Geolocation to show nearby events on a map and use File to upload picture to firebase storage. We use Camera to select or take profile picture and race logo picture.

- Geolocation

This plugin provides information about the device's location, such as latitude and longitude. This code shows we get current position's latitude and longitude by calling the geolocation function. We use this position to set the center of the map and set markers on the map.

```
function markCurrentLocation()
{
        navigator.geolocation.getCurrentPosition(onSuccess, onError, {timeout: 30000});
        function onSuccess(position) {
        var lat = position.coords.latitude;
        var lang = position.coords.longitude;
        var myLatlng = new google.maps.LatLng(lat, lang);
        map.center = myLatlng;
```

```
        new google.maps.Marker({ position: myLatlng, map: map, draggable: true,
        animation: google.maps.Animation.DROP })
}
```

- File

On the android platform, when we take a picture or select a picture , the device will return a file URL. Each file URL is in a special form, so we need use window.resolveLocalFileSystemURL() to convert it to a Directory Entry. This plugin also implements a File API for read and write access to files on the device.

```
window.resolveLocalFileSystemURL(file, function (fileEntry) {
        fileEntry.file(function (f) {
        var reader = new FileReader();
        reader.onloadend = function () {
                // This blob object can be saved to firebase
                var blob = new Blob([new Uint8Array(this.result)], { type: "image/jpeg" });
                uploadTask = firebase.storage().ref(filePath).put(blob);
        };
        reader.readAsArrayBuffer(f);
        });
        }, function (error) {
        ...         });
```

- Camera

  This plugin defines a global navigator.camera object, which provides an API for taking pictures and for choosing images from the system's image library. We use camera plugin to select or take user profile picture and race logo picture.

```
function getPhoto(camera) {
        if (camera === true) {
        navigator.camera.getPicture(onSuccess, onFail, {
        quality: 50, correctOrientation: true,
        sourceType: Camera.PictureSourceType.CAMERA,
        destinationType: Camera.DestinationType.FILE_URI
        });
        } else {
         navigator.camera.getPicture(onSuccess, onFail, {
        sourceType: Camera.PictureSourceType.PHOTOLIBRARY,
```

```
        destinationType: Camera.DestinationType.FILE_URI
        });
  }}
```

The use of the camera or media plugin is slightly different depending upon context, for example mobile webview compared to web browser. So when writing the OnSuccess callback function, we wrote different code based on platform when we got the platform information onDeviceReady().

```
function onSuccess(imageData) {
      var captureDataUrl = imageData;
     if( platformString == "browser")
     {
        captureDataUrl = '' + imageData;
     }
     var image = document.getElementById('raceLogo');
     image.src = captureDataUrl;
   }
```

❖ Third-party API

In addition to using MobileUI mentioned above, we also use Google Maps to show the races based on the location and show the race route. We also parse kml file to get the race start point latitude and longitude, then show the marker on the map.

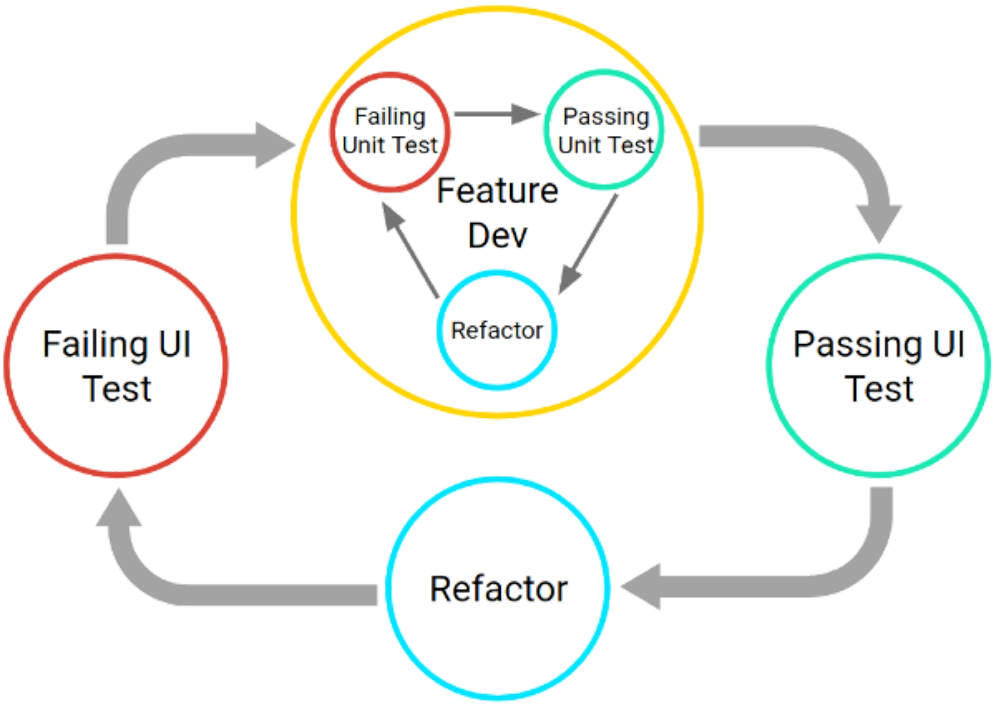
```
Init:
map = new google.maps.Map(document.getElementById("map-canvas"), {
        center: { lat:curLat, lng: curLng },
        zoom: 13,
        mapTypeId: "roadmap",
        });
Set route:
        var kmlLayer = new google.maps.KmlLayer(kmlPath);
        kmlLayer.setMap(map);
 }
Set Marker:
var marker2 = new google.maps.Marker({.....});
```

Testing and Iterative Design
– how have you tested your application relative to the following considerations,
* any usability testing
* any unit testing
* any design feedback and reviews
* any other relevant testing

❖ Testing and Iterative Design

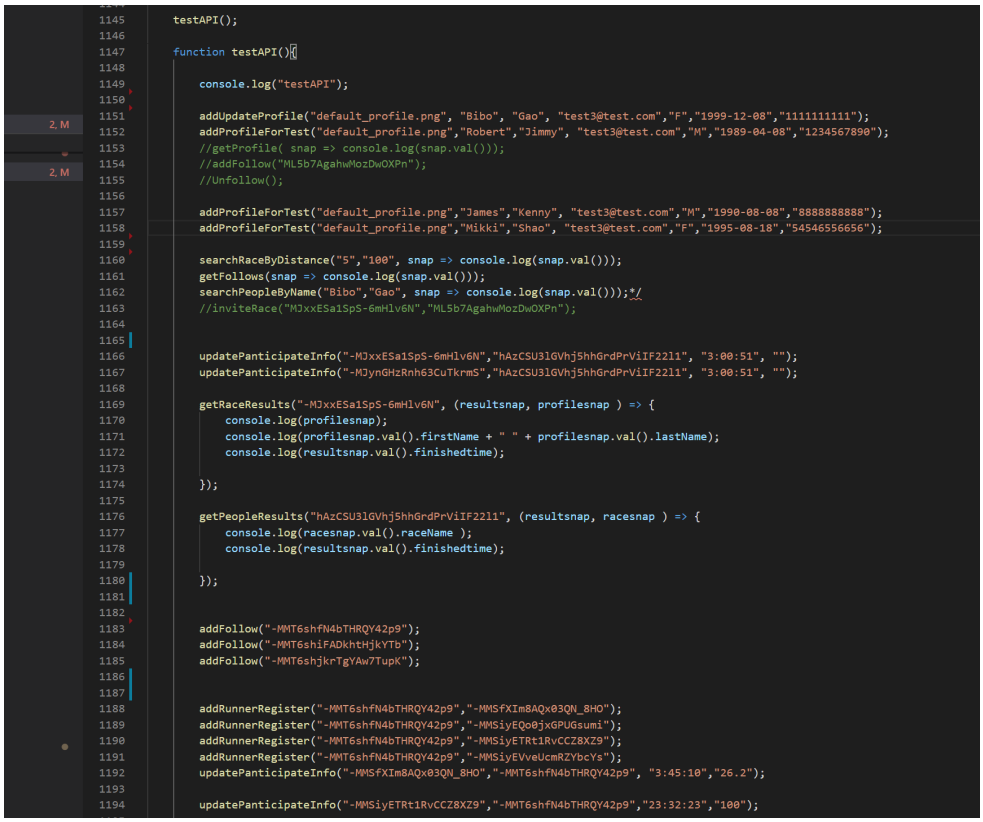
While we iteratively develop our app, we do unit test, UI test and integration test on browser and android platform emulator to test a variety of use cases and interactions.

The following is our full test workflow, which contains two iterative cycle series: unit test and UI test. This set of cycles continues until outapp satisfies every use case.



Unit test

For each function of API, we write a corresponding unit test. We test the units themselves using shorter, faster development cycles. When adding a new API function, we would add a unit test calling for it.



```
testAPI();

function testAPI(){

    console.log("testAPI");

    addUpdateProfile("default_profile.png", "Bibo", "Gao", "test3@test.com","F","1999-12-08","1111111111");
    addProfileForTest("default_profile.png","Robert","Jimmy", "test3@test.com","M","1989-04-08","1234567890");
    //getProfile( snap => console.log(snap.val()));
    //addFollow("ML5b7AgahwMozDwOXPn");
    //Unfollow();

    addProfileForTest("default_profile.png","James","Kenny", "test3@test.com","M","1990-08-08","8888888888");
    addProfileForTest("default_profile.png","Mikki","Shao", "test3@test.com","F","1995-08-18","54546556656");

    searchRaceByDistance("5","100", snap => console.log(snap.val()));
    getFollows(snap => console.log(snap.val()));
    searchPeopleByName("Bibo","Gao", snap => console.log(snap.val()));*/
    //inviteRace("MJxxESa1SpS-6mHlv6N","ML5b7AgahwMozDwOXPn");


    updatePanticipateInfo("-MJxxESa1SpS-6mHlv6N","hAzCSU3lGVhj5hhGrdPrViIF22l1", "3:00:51", "");
    updatePanticipateInfo("-MJynGHzRnh63CuTkrmS","hAzCSU3lGVhj5hhGrdPrViIF22l1", "3:00:51", "");

    getRaceResults("-MJxxESa1SpS-6mHlv6N", (resultsnap, profilesnap ) => {
        console.log(profilesnap);
        console.log(profilesnap.val().firstName + " " + profilesnap.val().lastName);
        console.log(resultsnap.val().finishedtime);

    });

    getPeopleResults("hAzCSU3lGVhj5hhGrdPrViIF22l1", (resultsnap, racesnap ) => {
        console.log(racesnap.val().raceName );
        console.log(resultsnap.val().finishedtime);

    });


    addFollow("-MMT6shfN4bTHRQY42p9");
    addFollow("-MMT6shiFADkhtHjkYTb");
    addFollow("-MMT6shjkrTgYAw7TupK");


    addRunnerRegister("-MMT6shfN4bTHRQY42p9","-MMSfXIm8AQx03QN_8HO");
    addRunnerRegister("-MMT6shfN4bTHRQY42p9","-MMSiyEQo0jxGPUGsumi");
    addRunnerRegister("-MMT6shfN4bTHRQY42p9","-MMSiyETRt1RvCCZ8XZ9");
    addRunnerRegister("-MMT6shfN4bTHRQY42p9","-MMSiyEVveUcmRZYbcYs");
    updatePanticipateInfo("-MMSfXIm8AQx03QN_8HO","-MMT6shfN4bTHRQY42p9", "3:45:10","26.2");

    updatePanticipateInfo("-MMSiyETRt1RvCCZ8XZ9","-MMT6shfN4bTHRQY42p9","23:32:23","100");
```

UI Test

We tested UI showing and tested multiple combinations of screen sizes. By iteratively developing the css and html file, we did our best to make the UI consistency and get better aesthetic design.

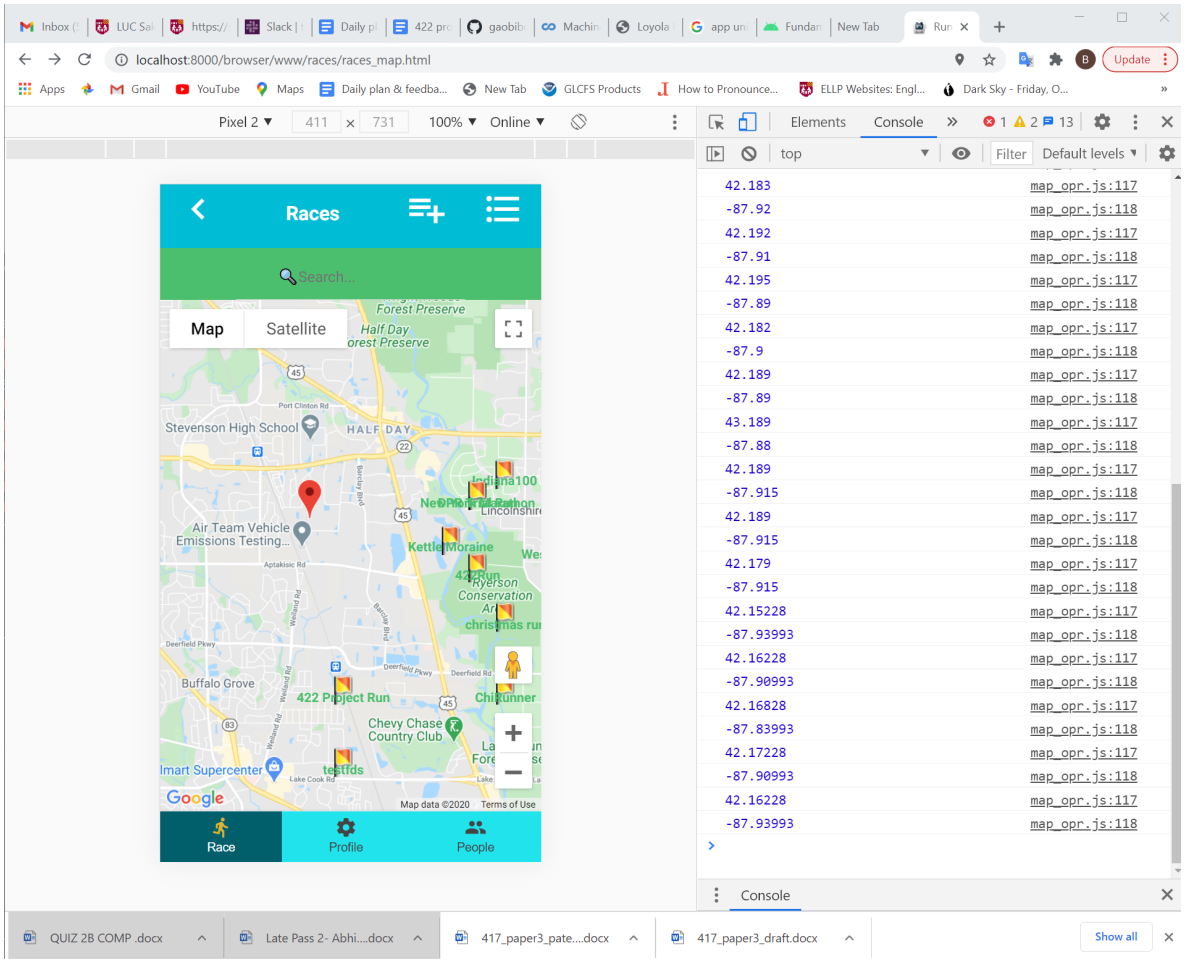
Integration test

When we finished the base functionality, we did integration tests on browser and android platform emulator.

Browser

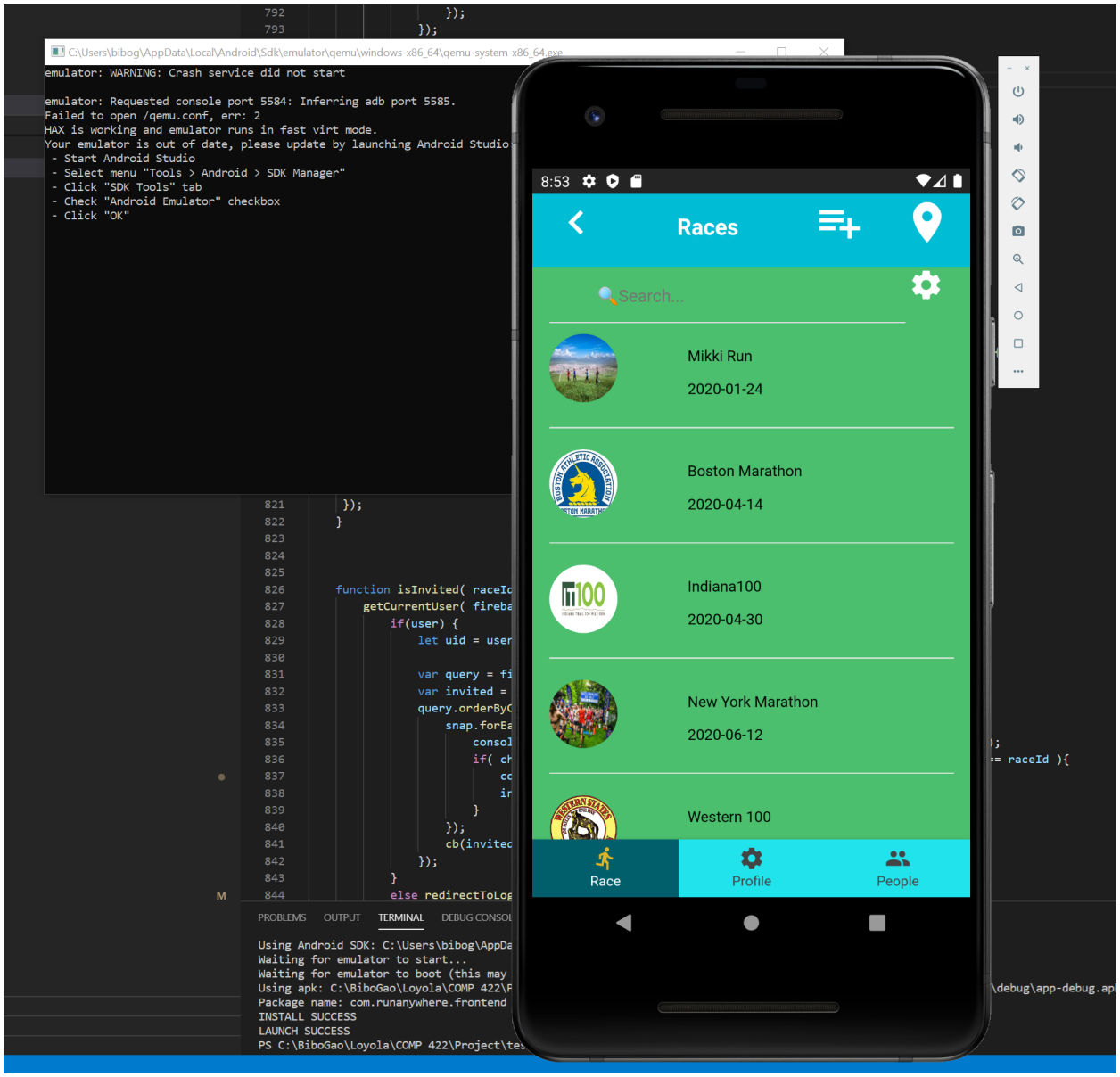
On the browser, We tested UI showing and interaction, we also tested fetch data from a server. We mainly did tests and checked our test result on the console window

of browser, which would output the content by console.log() function.



Android Platform Emulator

On an android platform emulator, We focused on the android platform specialties, and tested the interaction with the device's sensors, such as camera and geolocation, so that we can find the problems based on the platform. For example, when we use OAuth2.0 by firebase authentication to login google account, Google sign-in is automatically configured on our connected web apps. But to set up Google sign-in for our Android apps, we need to add the SHA1 fingerprint on our Project Settings. Aslo, just like we mentioned above, the camera plugin function on android also is different with on browser. By testing on an android platform emulator, we Found and fixed these problems.


C:\Users\bibog\AppData\Local\Android\Sdk\emulator\qemu\windows-x86_64\qemu-system-x86_64.exe
emulator: WARNING: Crash service did not start
emulator: Requested console port 5584: Inferring adb port 5585.
Failed to open /qemu.conf, err: 2
HAX is working and emulator runs in fast virt mode.
Your emulator is out of date, please update by launching Android Studio
- Start Android Studio
- Select menu "Tools > Android > SDK Manager"
- Click "SDK Tools" tab
- Check "Android Emulator" checkbox
- Click "OK"
PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOL
Using Android SDK: C:\Users\bibog\AppDa
Waiting for emulator to start...
Waiting for emulator to boot (this may
Using apk: C:\BiboGao\Loyola\COMP 422\F
Package name: com.runanywhere.frontend
INSTALL SUCCESS
LAUNCH SUCCESS
PS C:\BiboGao\Loyola\COMP 422\Project\te
8:53
Races
Search...
Mikki Run
2020-01-24
Boston Marathon
2020-04-14
Indiana100
2020-04-30
New York Marathon
2020-06-12
Western 100
Race
Profile
People