FitBoard - *A Real Time Real Life Marathon Game*

Arnav Chakravarthy
Abhishek Chaudhary

1. Introduction

The problem currently is that:

- On average there are 11.4 marathons/cyclothon held in each state in the USA in a year, and moreover these marathons are held in only a handful of states.
- The marathon fees range from 60$ to 300$, and for cylcothons it is even more. This could be financially infeasible for many people.
- Since it is held in a handful states, people willing to take part will have to travel, making it geographically infeasible too.
- In addition, to organize a marathon, lots of manpower is needed for making sure participants are following the correct path, handling registrations,etc
- Also, currently no application exists which can track real time positions of participants in a marathon/cyclothon which would help the participants to know which position they are in at a certain point of time, which would help them improve their running/cycling strategy on the spot

We aim to build a cloud application which is able to solve all of these problems, which is described in greater detail further in the report.

2. Background

There exist an abundance of fitness tracking applications such as Nike Running, Google Fit, Adidas Running, Strava, etc. But these applications are only meant for personal tracking of a jog or run, and possibly record health statistics too. But what if you want to organize a marathon/cyclothon with a group of people around your area? And how would it be possible to track user locations in real time during the marathon to ensure that users are following the defined marathon path, and not taking a shortcut? Also, there is no way for marathon participants to know what position they are in with respect to other participants at any point in time, in order to improvise their running strategy on the spot.

There also exist applications such as Sporty which allows users to create and register for events nearby, but it is more of a management automation platform, where users can manage bookings, payments,etc for the people participating in the events. There is no platform to keep track of these events in real-time while they take place, which could be a more exciting domain to delve into.

In addition, in order to organize a marathon, lots of manpower is needed, and event organizers actually place people at almost every corner of the marathon path, to ensure participants are adhering to the path and not trying to take a shortcut.

Solving these problems is crucial as we believe participants willing to take part in these events are discouraged by the fees as well as geographical locations of the events. In addition, we believe gamification increases competitiveness, and a detailed leaderboard as well as the potential prize money could incentivize people enough to take part in these events.

Users who create the events could set an affordable registration fee, which would attract and encourage more people to participate.Taking part in it is as simple as a click of a button. (and of course running/cycling).

In addition our technology could relieve all the extra work event organizers have to go through to organize these events, as we could tie up with event organizers to host their events on our platform.

FitBoard (Our technology) solves all these problems by combining features of creating events, registering for nearby events as well as tracking all users taking part in the event in real-time and in synchronization across all devices. We have slightly been inspired by the mobile game "Fun Run". It is basically a mobile game where users control their game character, can view each other's positions in real-time at all points in time, and once they finish the race, a leaderboard is generated which is updated in real-time as other users complete the race.

We figured, why not make this a real-life game and extend this concept to marathons and cyclothons, in which the goal is essentially the same. Hence, we created a real-time real-life marathon/cyclothon game, where users could themselves create events, which would be available to other users in the same locality if they are willing to participate. They could then easily register for whichever event they wanted to take part in.

Once the event starts, our technology is able to keep track of all users participating in real-time, as well as provide a user interface for users to see their relative position in the race as well as positions of other participants in real-time. It also automatically records when users finish the race and update the leaderboard in real-time too. Users can also view meaningful statistics regarding their event, post their event.
They can view these details post the event too, in their event history in order to know how much they have improved since previous marathons/cyclothon.

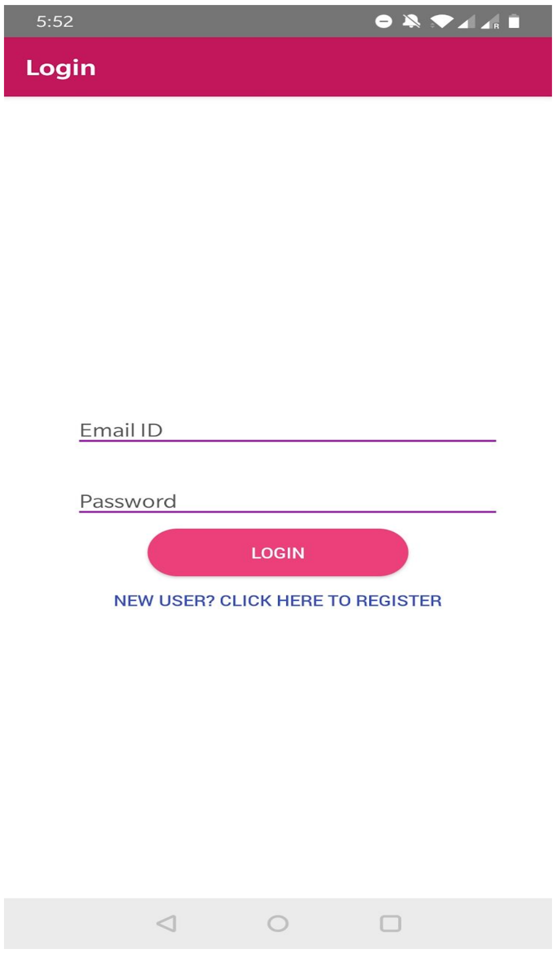
The scope of this application is huge, as in addition for users creating their own events, we could easily host an event organizing event on our platform, providing them tracking services and analytics of all the users taking part in their event.
Users will have strong incentives to use this application as the prize money could be as big as the number of users taking part in the event. The registration fee could be decided by the user, and they would not necessarily have to pay a large amount to take part in the marathon.

We also provide a real-time leaderboard which gamifies the whole experience, incentivizing users to rise up in the leaderboard.

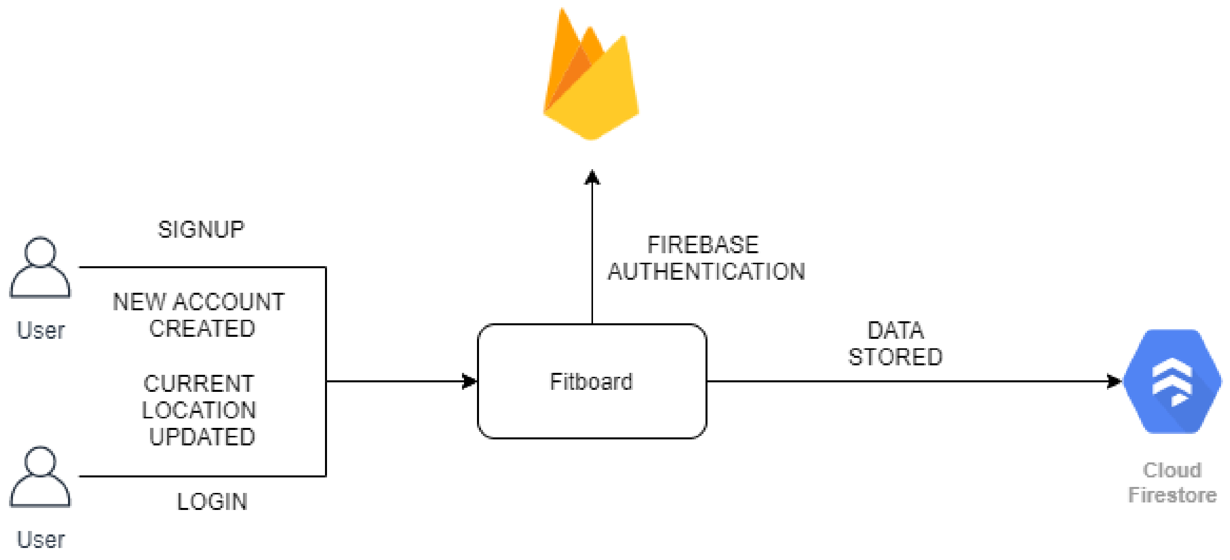
3. Design and implementation

3.1 Architecture

3.1.1 Login/Signup component



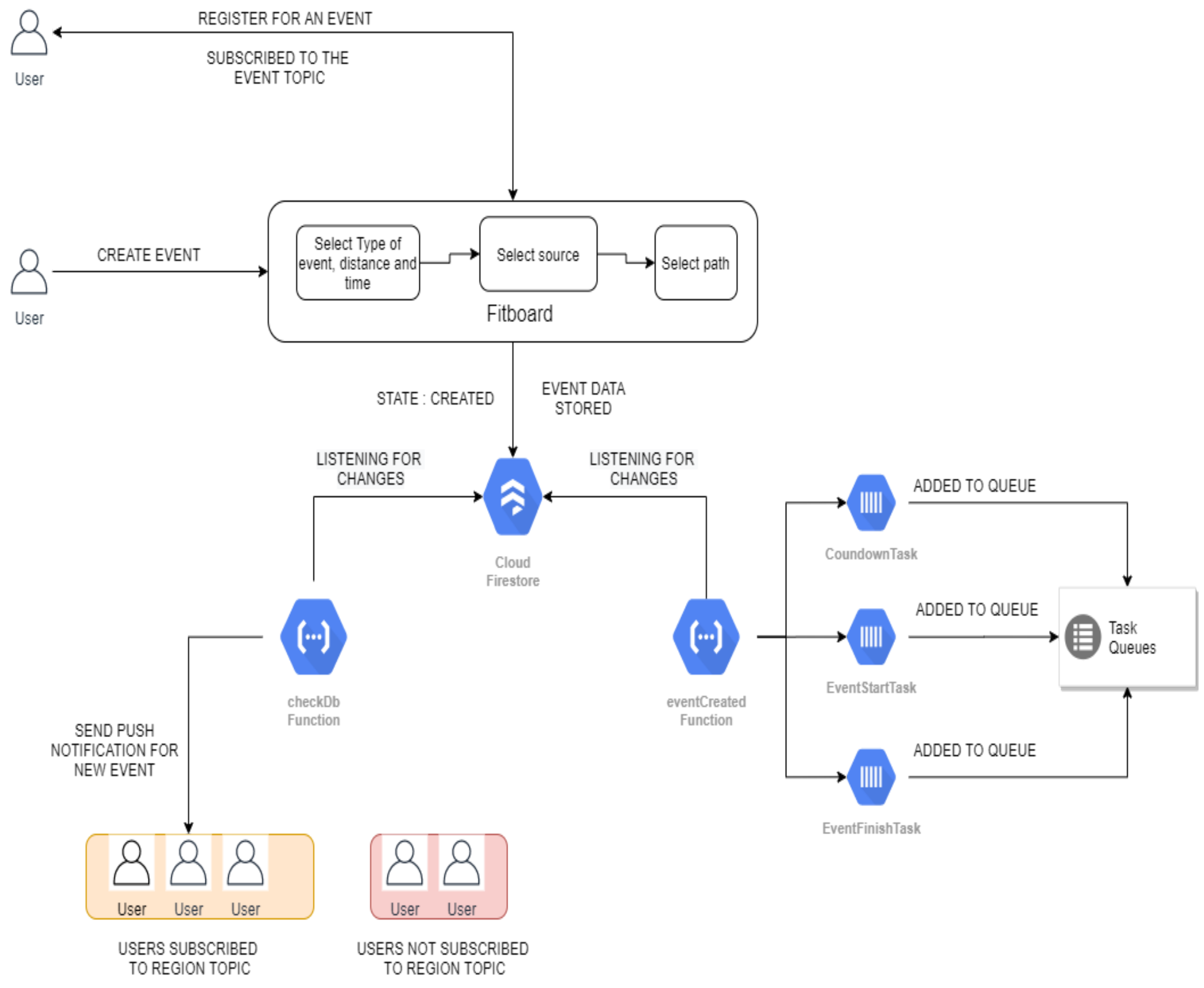
Design



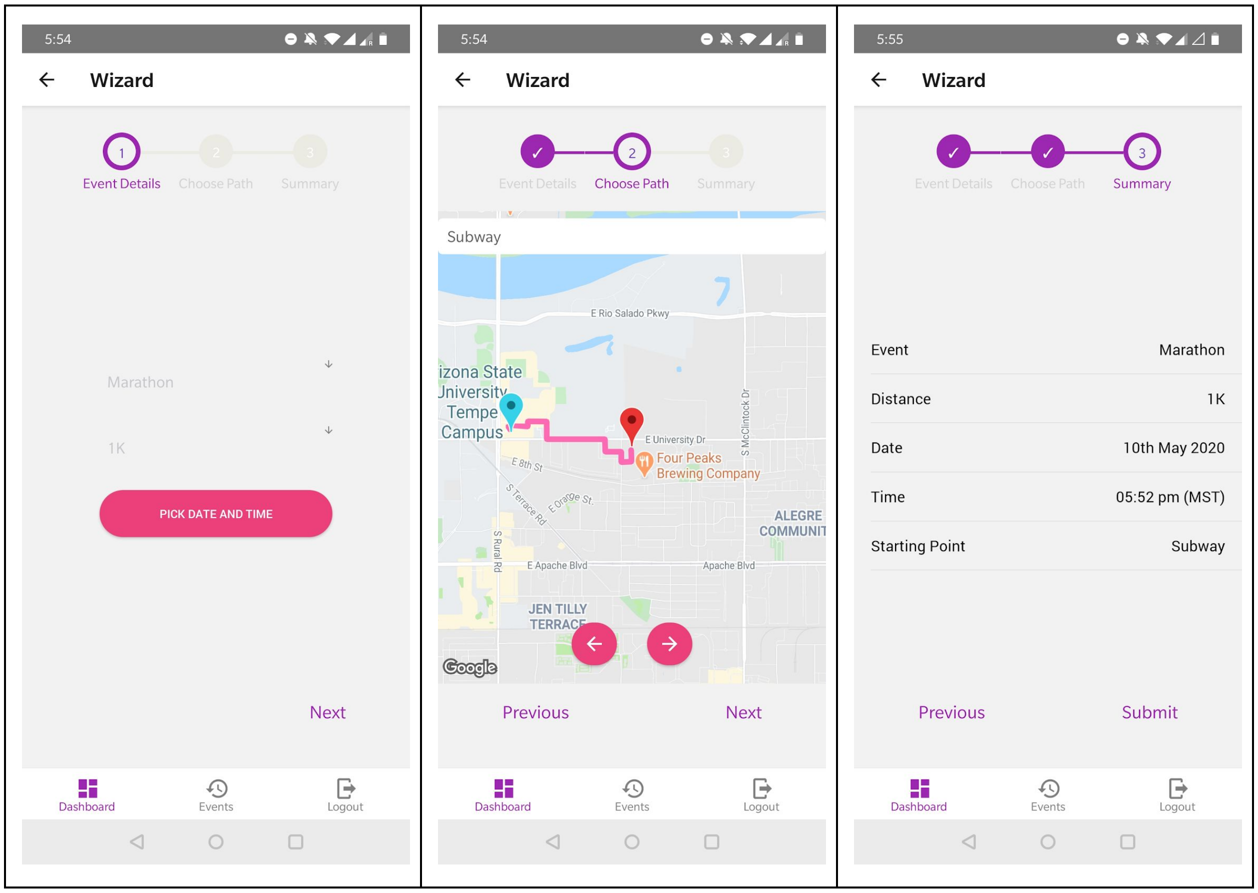
<u>Architecture</u>

Inorder to make use of the App a user needs to register by creating a new account or login using an existing one. We are making use of firebase authentication for both *login* and *signup* operations.
After signup/login every time a user opens an app, his/her location is updated in the firestore. This updated location is used to subscribe the user to a ‘region’ topic, an example of the topic can be “**Az-Tempe-85281**” which is composed of “**State-City-Zipcode**”. This topic is then used to give push notifications to the user everytime a new event is added nearby them.

3.1.2 Event component



<u>Architecture</u>



Event	Marathon
Distance	1K
Date	10th May 2020
Time	05:52 pm (MST)
Starting Point	Subway

Design

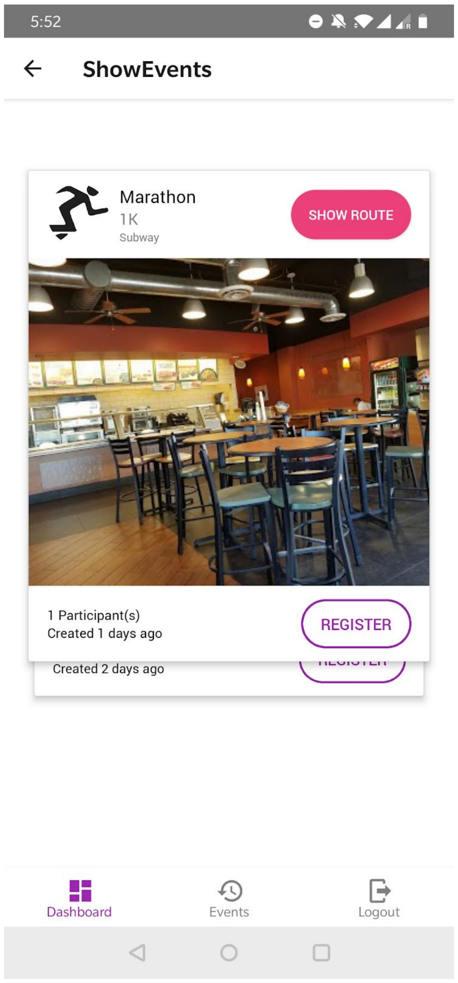
3.1.2.1 Creating Event

While creating an event following details are taken from the owner of the event :-

1. Type of the event : Marathon or cyclothon
2. Distance of the event
3. Start time of the event
4. Start location which uses google places autocomplete to show results places for the search query.
5. Using the start location and the distance we then make use of google places api and google distance api to find places at the distance specified by the user.
6. Users can then select multiple paths from the results of the previous step and set the destination. This step allows you to get the destination and waypoints to the destination.

After all the required information for the event is obtained then the event data is stored in a cloud firestore component. Alongwith the information we obtain directly from the user, we store additional information for the event like the topic the event would belong to. This topic is obtained by using the source location and converting into the format of **State-City-Zipcode** as specified in section 3.1.1

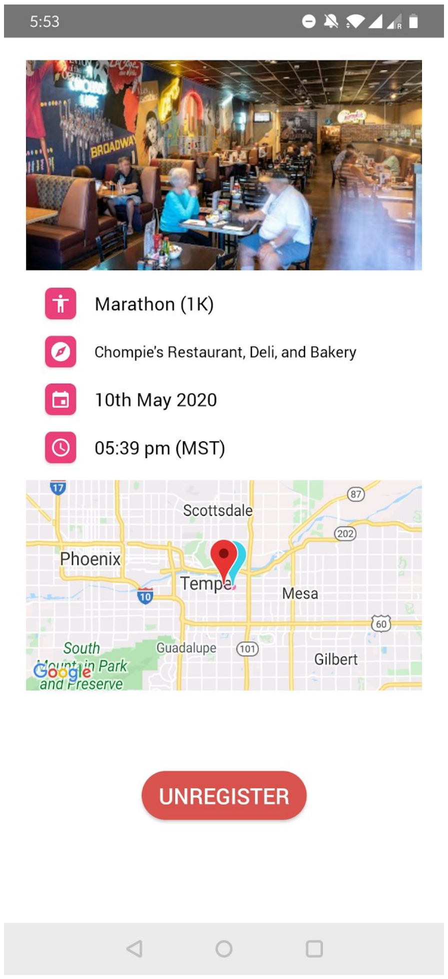
3.1.2.2 Registering Event



Marathon
1K
Subway

1 Participant(s)
Created 1 days ago

Created 2 days ago



Marathon (1K)

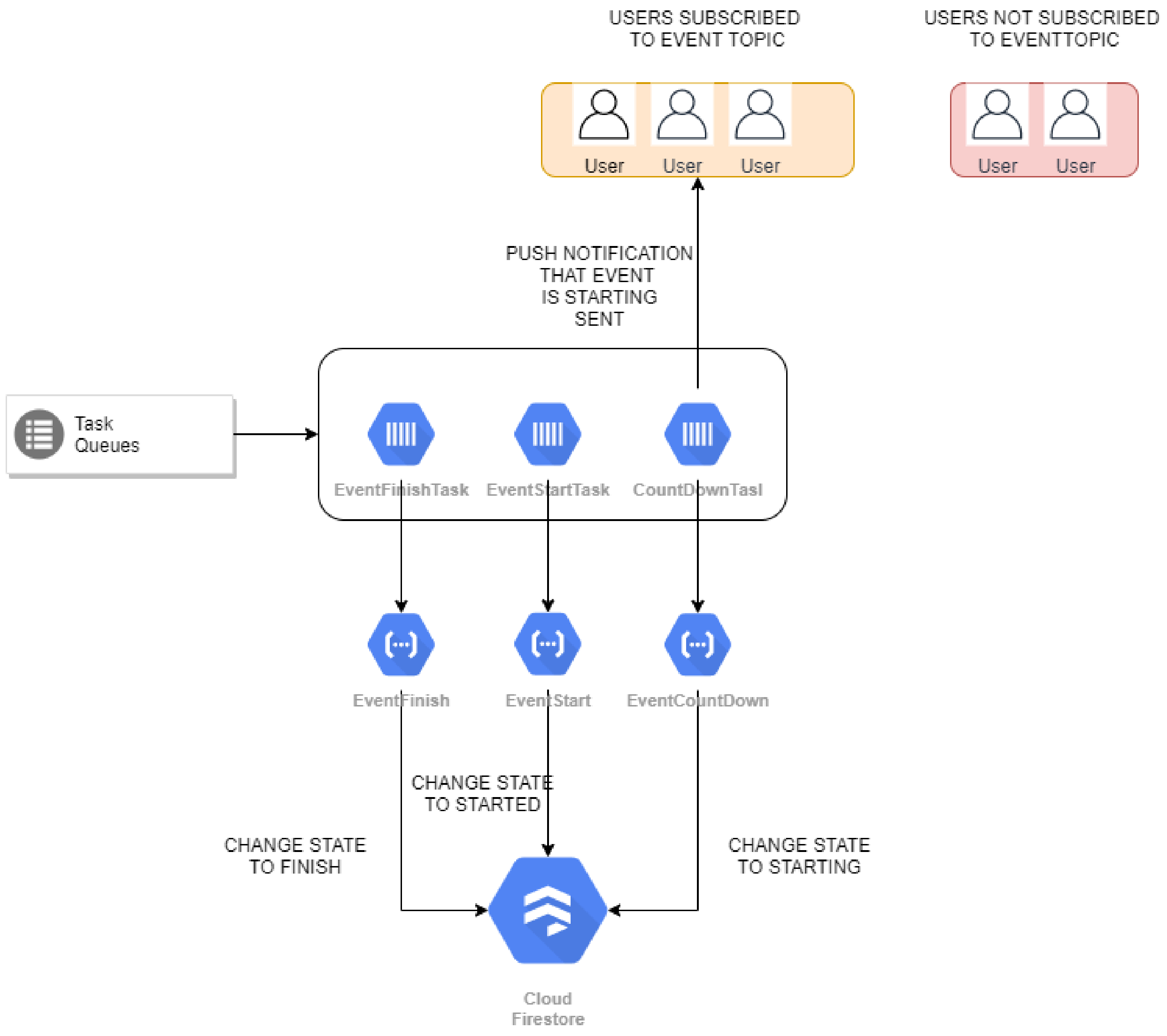
Chompie's Restaurant, Deli, and Bakery

10th May 2020

05:39 pm (MST)

Users can register for any event whether it’s near his/her location or not. When the user registers for an event they are subscribed for that event topic. This topic is then used to send push notifications about that event.

3.1.2.3 Cloud Task and Cloud Functions



3.1.2.3.1 Cloud Tasks

After an event is created, its data is safely stored in the cloud firestore. We have made use of cloud functions (*checkDB* and *eventCreated*) which listen for any update to the *event* collections.
CheckDB is responsible for sending push notifications to the user which have subscribed to the topic **State-City-Zipcode,** to which the event belongs.
eventCreated is responsible for three operations.

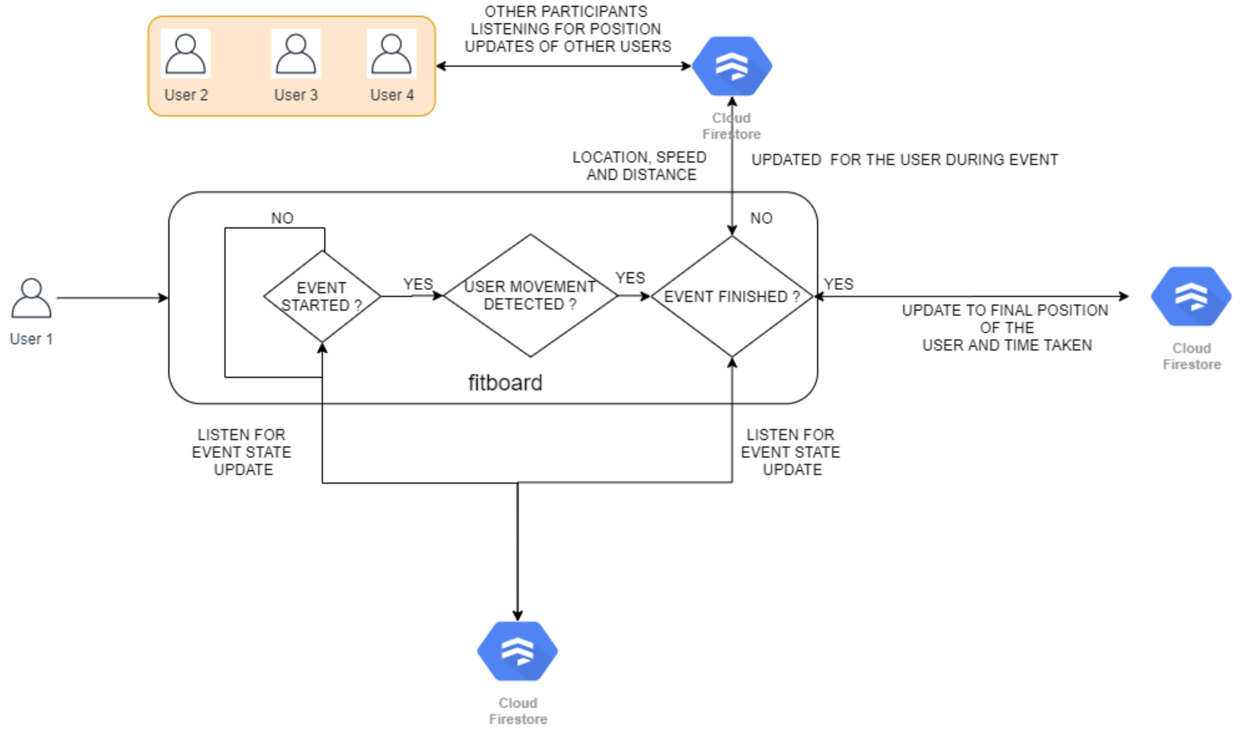
- Creating **CountDownTask**
- Creating **EventStartedTask**
- Creating **EventFinishedTask**

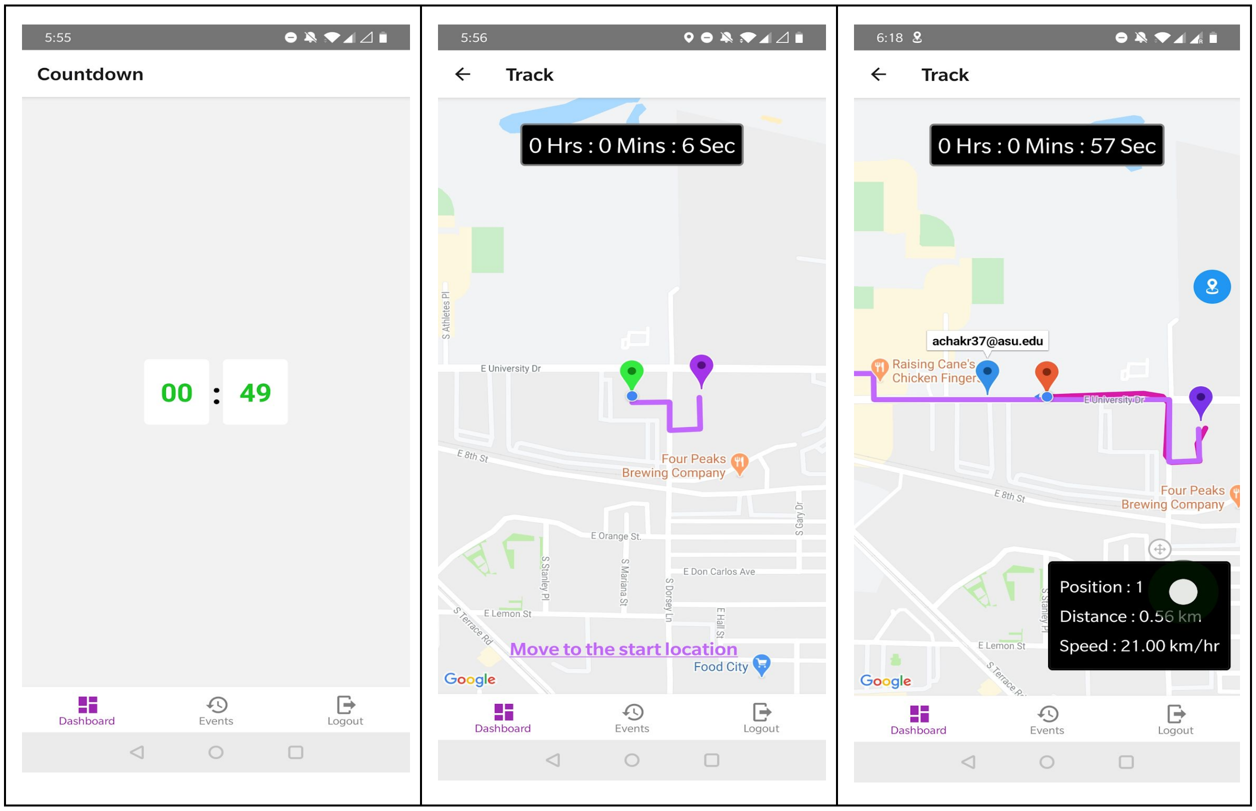
After creating every task it is added to the Cloud tasks queue. Each of the tasks is created asynchronously and independently of the others.

3.1.2.3.3 Cloud Function

- **CountDownTask** is responsible for invoking the **EventCountDown** function. **EventCountDown** then sends push notifications to the users who have registered for the event. The notifications are sent a few minutes prior to the start of the event. This allows the user to get ready for the event. It also changes the state of the event to **starting**
- **EventStartedTask** is responsible for invoking the **EventStarted** cloud function. **EventStarted** is responsible for changing the state of the event to **started.** Map for the event is rendered only after the event starts.
- **EventFinishedTask** is responsible for invoking the **EventFinished** cloud function. **EventFinished** changes the state of the event to **finished**. After an event is finished it moves to the past event section where the user can view his/her ranking and analysis.

3.1.3 Event Participation

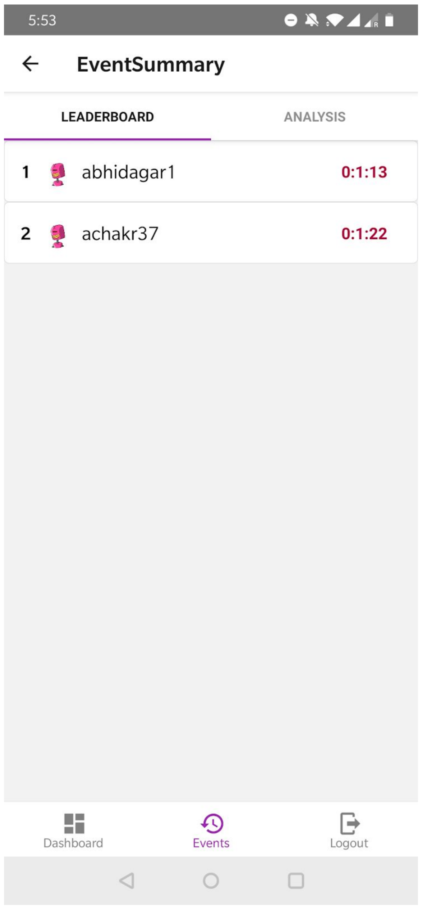

User 2
User 3
User 4
OTHER PARTICIPANTS LISTENING FOR POSITION UPDATES OF OTHER USERS
Cloud Firestore
LOCATION, SPEED AND DISTANCE
UPDATED FOR THE USER DURING EVENT
NO
NO
User 1
EVENT STARTED ?
YES
USER MOVEMENT DETECTED ?
YES
EVENT FINISHED ?
YES
UPDATE TO FINAL POSITION OF THE USER AND TIME TAKEN
Cloud Firestore
fitboard
LISTEN FOR EVENT STATE UPDATE
LISTEN FOR EVENT STATE UPDATE
Cloud Firestore



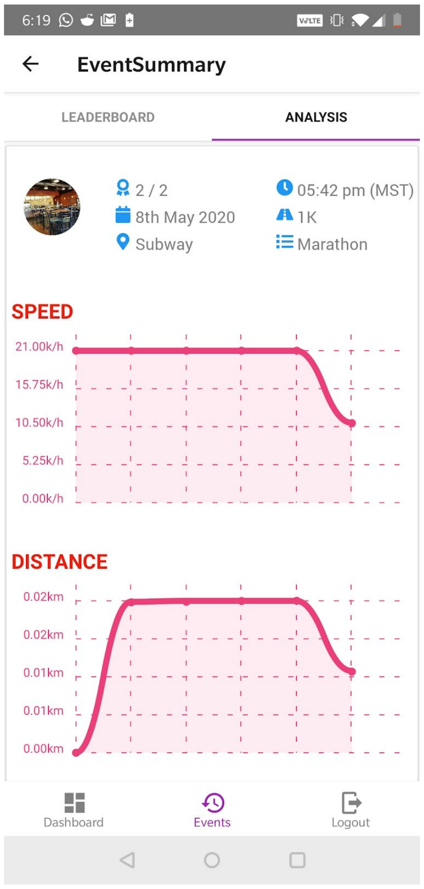
After registering for an event, users get a notification a few minutes before the event starts. In the meantime (before the event starts) users are displayed a countdown timer. After the event starts every user movement results in an update of his/her location. On movement we record the current speed, position and distance travelled since the last time we recorded the position. We also have a firestore listener which keeps checking the state of the event. If the event is finished then the data collected above is not sent to the firestore otherwise the user data is sent to the firestore.
Every user has another firestore listener which listens to any update to the particular event collection. Any kind of update means that some participating has moved hence such a movement needs to be reflected by other users on their map. This updated information is then used to render updated marker locations on the event map thus giving us real time location updates for the event.
When a user reaches the event destination the event is marked finished for him/her in the firestore. Users are then shown the option to view their analytics and leaderboard for the event.

3.1.4 Leaderboard and Analytics

After the event is over, users can then view the leaderboard and analytics for the event. Leaderboard shows the user rankings for the event based on the time it took them to finish. Analytics for the event are calculated using the speed and distance for the user which are recorded every time the user moves.



1	abhidagar1	0:1:13
2	achakr37	0:1:22



2 / 2
8th May 2020
Subway
05:42 pm (MST)
1K
Marathon

3.2 GCP Components

1. **Cloud Firestore**

Cloud Firestore is a flexible, scalable database for mobile, web, and server development from Firebase and Google Cloud Platform. It is built upon the

traditional Firebase Real-Time database and is enhanced with additional features.
We needed a realtime database in order to reflect user position states on each user's device. We did this with the help of Snapshot listeners which listens to a change in a document or a whole collection. In our case we listened for changes in the collection which signified an update in the user's location which should be reflected on all user's devices taking part in the event.
We also use firestore triggers in order to launch cloud functions which could be used for features such as push notifications.
Firesore's guarantee for automatic multi region data replication, scaling with increase in user traffic and real time updates, made it a very strong candidate for our choice of database. Our data schema could become complex since we need to update the user positions and store them as an array of objects, and these hierarchical and complex data structures are supported by Firestore

By default, Firestore also persists data on the user application, and is used by the SDKs for offline usage and caching
Sharding is handled automatically by Firestore.

In our Case, firestore is used for:

- Storing Static user information when the register for the first time
- Storing dynamic user information such as their last updated location
- Storing static event information such as the start location, destination, waypoints, date, time, distance and type of the event.
- Storing dynamic event information such as updates of the event states (created,starting,started,finished) as well as real time user's locations taking part in the event once the state has changed to "started"
- Storing event's history information after the event's state has changed to "finished"

2. Cloud Functions

We wanted to make our application entirely serverless and google cloud functions was the best way to go about it.
Google cloud functions provided a serverless execution environment for building and connecting our cloud services.

It is much easier to debug as functions are the most basic unit of the application, so we could exactly pinpoint through the logs as to which function is causing errors.
We have ensured that the tasks executed by cloud functions are short-lasting, such as changing our firestore collection’s state or sending push notifications.
Google also provides a guarantee to automatically scale cloud function instances based on the load, and each function scales independently.
Cloud Functions also provide a smart way to avoid the cold start problem by keeping a few idle instances which the users are not billed for.

All our cloud functions are single purpose and reusable such as, we have a single cloud function for sending push notification which takes arguments for sending various kinds of push notifications.
We have integrated Cloud functions for the following use cases:

- Changing our event states from Created->Starting->Started->Finished. These functions are triggered using Task Queues which schedules them at a specific time based on the Event’s start time and finish time.
- Sending location-aware push notification, which is triggered by an update in out firestore Event collection. Which means, whenever a new event is added, we send a push notification to all users subscribed to the ‘region topic’ in which the Event is taking place.
- Sending Event reminder push notifications a few minutes before the Event starts to all users subscribed to that specific ‘event topic’, in order to remind them of the start of an event. The firing of this function is scheduled by the task queue according to the Event’s start time.

3. Cloud Messaging

Cloud messaging is a cross platform messaging service which allows developers to push messages to user’s devices. This could be done by sending messages to a single device, groups of devices or devices subscribed to topics. We make use of devices subscribed to topics since it Is easier to keep track of. This is done by making use of firebase topics.
The two kinds of topics we use in our application are ‘regional topics ‘and ‘event topics’. Users can subscribe to these topics, and Cloud messaging allows us to

target users based on the topics they have subscribed to, such as sending location aware push notifications.
Cloud Functions are used to send messages and our React native application receives these messages using the device's transport service.

We use Cloud Messaging in order to send two types of push notifications:

- Location aware push notification - This informs users if there are new events in their locality. This is done by sending messages to all users subscribed to a specific 'region topic'
- Event reminder push notifications - This reminds users that an event they registered in is starting in a few minutes. This is done by sending messages to all users subscribed to a specific 'event topic'

4. Task Queues

Cloud Tasks is a fully managed service that allows us to manage the execution, dispatch, and delivery of a large number of distributed tasks. Cloud Tasks help to easily manage the execution, dispatch, and delivery of tasks. Cloud tasks take a target to execute which can be a Pub/Sub channel or an HTTP target (Compute Engine, Google Kubernetes Engine, Cloud Run, Cloud Functions).
For our application we are making use of three types of tasks

- **CountDownTask**
- **EventStartedTask**
- **EventFinishedTask**

Each of these tasks is responsible for invoking a cloud function on execution. A specified time is scheduled for each of them at which point they are executed.
We are using cloud tasks for the following functions:-

- Invoking an HTTP target (Cloud function)
- Changing state of an event in the firestore DB.

3.3 Autoscaling

Our application makes use of google cloud functions and task queues which makes it serverless. One of the advantages of serverless applications is that they can be deployed without the front-work of setting up infrastructure. As such, it is possible to launch a fully-functional site in days. The best part is that serverless backends scale automatically with demand. No need to fear crashing when you get sudden surges of traffic.

Moreover our Google cloud functions are invoked by cloud tasks and on certain firestore events hence there is no need to keep certain instances of functions always running.
For the cloud functions software and infrastructure are fully managed by Google. Furthermore, provisioning of resources happens automatically in response to events. This means that a function can scale from a few invocations a day to many millions of invocations without any work from us.
Also we are making use of google tasks queues which can perform work asynchronously outside of a user or service-to-service request. Moreover task handlers are implemented using dedicated services, which allows microservices to scale independently.

3.4 Solving the Problem

As explained above, our cloud architecture is scalable as well as fault tolerant. Since the amount of users using our application could be unpredictable, building an architecture which is easy to scale was extremely important and cloud was the way to go.
Since we did not want to worry much about handling the scaling out of the application ourselves, Platform as a Service took care of this aspect for us, as well as installing all relevant dependencies for our application and packaging and distributing it to run in an isolated environment. We could hence focus on building more features for the application.
By giving the ability for users to themselves create events with a flexible registration fees which would be available to participate in by other users in the same locality, giving users the ability to register for these events with a click of a button as well as being able to keep track of the participants in real time (during the duration of the event), and providing this information with a clean User interface, as well as ensuring the users do not go off path, we were able to successfully solve the problems addressed before.

3.5 **What other solutions lack**

While other applications, though similar in a few aspects, are still quite different from our application. Our application being novel cannot be fairly compared to existing solutions.

For example there exists an abundance of fitness tracking applications such as Nike Running, Google Fit, Adidas Running, Strava, etc. But these applications are only meant for personal tracking of a jog or run, and possibly record health statistics too. But what if you want to organize a marathon/cyclothon with a group of people around your area? And how would it be possible to track user locations in real time during the marathon to ensure that users are following the defined marathon path, and not taking a shortcut? Also, there is no way for marathon participants to know what position they are in with respect to other participants at any point in time, in order to improvise their running strategy on the spot.

There also exist applications such as Sporty which allows users to create and register for events nearby, but it is more of a management automation platform, where users can manage bookings, payments,etc for the people participating in the events. There is no platform to keep track of these events in real-time while they take place, which could be a more exciting domain to delve into.

In addition, in order to organize a marathon, lots of manpower is needed, and event organizers actually place people at almost every corner of the marathon path, to ensure participants are adhering to the path and not trying to take a shortcut. And none of these applications provide a way to automate ensuring users stay on the correct path.

4. Testing and evaluation

Fitboard’s main goal is to allow users to create and register for events in their locality as well as show users participation in an event. To simulate the user’s movements we made use of a GPS simulator - Joystick. The joystick app lets us select a particular location and provides us with controls to move in any direction and thereby affects our location based on those movements.

We created several events for distance 1K and 5K each. For every there were three tasks that were pushed to the cloud tasks queue. The coundownstart task was executed one minute prior to the eventStart task and the eventFinish task was executed when the event got over.

For all the events we verified that events were compartmentalised and that a user can participate in a single at a point of time.

To test the real time location aspect of the app we made use of 8 user accounts all of which made use of joystick application and participated in the same event. No latency was observed in rendering the map markers for these users. Moreover all the user rankings were correct and analysis was on point.

We also ran events simultaneously (with exclusive sets of users) to verify that one event didn’t interfere with the working of another. No such effect was observed, moreover there was no latency due to simultaneous events.

Since, this application is not intended to be used with the GPS joystick and actually involves the user's physical movements, we created, registered and participated in events ourselves by going for jogs. Around 5 of us went for a run together and the application worked seamlessly providing each of us real-time location updates of each other, as well as tracking once we finished the event. (Arrived at the destination point). It also generated meaningful running statistics for us, as well as updated the leaderboard in real time as and when other users completed the event.

For eventCreated cloud function average execution time was ***1666ms*** which involved creating and adding 3 tasks to the cloud task queue.
Similarly for the checkDB cloud function the average execution time was **507ms** which involves sending push notification to the users which are subscribed to the “**region topic**” .
We set our maximum number of instances to be 10, in order to not get billed too high, and tested this with 8 users simultaneously creating, registering and participating in events. The above latency increased very slightly but remained more or less the same.

5. Code

5.1 FitBoard App

5.1.1 Authentication Module

This is the core logic responsible for taking care of user sessions, and is divided into 3 parts namely Registration, Login and Logout. We make use of Firebase to streamline the process of handling user sessions

1. **Registration**
    - In the case of a new user, the user is made to register with FitBoard with their details. Their email and password security is taken care of by firebase authentication, whereas extra information such as their name is stored in Firestore.
    - Since it is a new user, we send a verification email in order to confirm that they have registered with their valid email Id

2. **Login**
    - When a user has successfully registered and verified their email Id, they can enter their details to login. This is when firebase starts the user session.
    - The user credentials are sent to Firebase authentication backend and exchanged for a Firebase ID token and a refresh token
    - We persist the user authentication state across user sessions by persisting the session state in the device cache
3. **Logout**
    - This simply deletes the authentication state from the device cache, hence clearing the user session and requiring the user to login again

5.1.2 Landing Page Module

- This is the module which is responsible for taking care of when the user first opens the application. This is also known as the user dashboard module.
- As soon as the user logs in, or is already logged in and opens the application, he/she is redirected to this screen.
- We use the **GetLocation.getCurrentPosition()** to get the user's current position and update this in the user's collection in firestore.
- The user is also subscribed to a 'region topic' based on his location. This would be used later to send push notifications.

- The user's location is also used to render events nearby the user's location in order for him to participate.
- We implement a Firestore snapshot listener to listen for event state changes which the user has registered for. These states could change from Created->Starting->Started->Finished.
- When the event is in the "created" state, it is rendered with a "View" button in order to view event details.
- When the event is in the "starting" state, it is rendered with a "Participate" which redirects to a countdown timer, which counts the time until the event starts, after which the user is directed to the real time game screen.
- When the event has changed to the "finsihed" state, the event disappears from the user dashboard and appears in the Event History Screen.

5.1.3 Create Events Module

- This is the core logic for creating an event such as marathon/cyclothon
- The user chooses which event he wants to take part in, and also chooses the distance.
- He is then directed to a screen where he has to pick a start location for the event.
- We make use of the Google Places API to provide the user a list of places nearby his/her own location .
- Based on the distance provided by the user, we find all paths from the start location and whose distance is equal to that chosen by the user, and provide a user google maps interface with the paths highlighted. The user can then choose which path he intends to use for the event. We use the **GetPaths()** function which we created, and takes in as arguments the start location and the distance, and returns a list of waypoints.
- In the end, the user is then provided a summary of the event, which can be confirmed and submitted. This then creates a new document in our Firestore "Events" collection where the event details are stored.

5.1.4 Register Events Module

- This module handles the registration of events for the users.
- Based on the current / last updated user location, we provide the users a list of events taking place in their locality, provided the user has not registered for the event and the event has not been finished already (event state != "finished").
- The user can view event details such as the start location,distance, date as well as the actual path of the marathon/cyclothon
- The user can then wish to register for the event, which would then pop up on his/her's dashboard.

5.1.5 Track Module

- Track module forms the core logic for real time rendering of the players taking part in the event.
- This module first gets the information for the current user and event using **getCurrentUser()** and **populateEventDetail()**, then the current location of the user is fetched using **getCurrentLocation().**
- After getting the current location of the user we check if the user is at start location or not using **checkIfAtStart().** If the user is not at the start location then directions to the start location from the current location are shown. On every movement of the user we check if the user has reached the start location or not. If the user reaches the start location then the actual map for the event is shown.
- **watchPosition()** is responsible for checking the location of the user on every movement. This function also calls the **checkIfStarted()** to check for the start
- **watchPosition()** is responsible for getting the current speed, distance and location of the user. This information is then sent to firestore using the function **update_firestore().**
- To show the real time ranking of the users we make use of a listener **listenForUpdate()** which listens to any kind of update to the event collection. These updates are then reflected on the map using **displayOtherPlayer().**
- Watch position is also responsible for checking if the user has finished the event or not. If the event has finished then the user is shown a popup which prompts the user to view his/her event summary page.
- Track module also has a timer which keeps track of the time spent by the user during the event. This recorded time is then updated to the firestore when the user finishes the event and is used to give the final ranking of the users on the event summary leaderboard.
- Distance travelled by the user is calculated using **calcDistance().** Which makes use of the haversine module. Haversine is an npm package that is responsible for finding the distance between two coordinates along the curvature of the earth.

5.1.6 Event Summary Module

- The Event Summary module is shown when the event is finished for the user. This module is responsible for showing the leaderboard and the analysis for the event.
- Leaderboard is rendered using the **userRanking** module.
- **userRanking** has a listener over event collection. Whenever any update to the rankings parameter of the event collection is received, the list of **(users, finish_time)** is sorted in ascending order of finish time. Then the final list is used to render the leaderboard.
- Due to the presence of the listener the updates to leaderboard are real time and don't require any refresh.
- The analysis for the event makes use of the event collection information for each user.
- During the event various speeds and distance covered by the user are continuously updated to the firestore. So at the end of the event the user has a list of speeds and distance covered per time interval. That information is then used to render the line graphs using **getLineGraph().**
- Similar to Track module, Event Summary module also makes use of the following function :-
    - **getCurrentUser() -** To get the information for the current user.
    - **populateEventDetail() -** To get the information for the current user.

5.1.7 Past Events Module

- The Past Events module is responsible for showing the finished events for the user.
- Firstly the events a user has participated in is fetched by querying the firestore event collection.
- Then the events are shown as a list. Users can view the event summary for any event by clicking on the view button.

5.2 FitBoard Cloud Functions and Cloud Tasks

5.2.1 checkDB function

- This cloud function is responsible for listening to any updates on **"Events/{docId}".**
- When a new event is created, push notifications are sent to the users who have subscribed to the region topic.

5.2.2 eventCreated function

- Similar to checkDB function eventCreated also listens for updates on **"Events/{docId}".**
- Any update here means that a new event has been created. Then this function is responsible for three tasks.
    - Creating a cloud task (**CountDownTask**) to change the status of the event to **starting. CountDownTask** is set to execute one minute before the start of the event. This task invoked the **eventCountDownStart** function when it's executed.
    - Creating a cloud task (**EventStartedTask**) to change the status of the event to **started. EventStartedTask** is set to execute at the start time of the event and invokes **eventStarted** when executed.
    - Creating a cloud task (**EventFinishedTask**) to change the status of the event to **finished. EventFinishedTask** executes when the event is over and invokes the eventFinished **function.**

5.2.3 eventCountDownStart function

- This function is responsible for sending the push notification to all the users who have subscribed to the event topic (by registering for the event).
- eventCountDownStart() also changes the value of **state** parameter in the event collection to **starting**

5.2.4 eventStarted function

- eventStarted() changes the value of **state** parameter in the event collection to **started**
- After this change only can the users start the event.

5.2.5 eventFinished function

- eventFinished() changes the value of **state** parameter in the event collection to **finished**
- After this change the event moves to the past events section for the user where they can view their event summary.

5.2.6 createTasks function

- All the above cloud functions use the createTasks function to create a cloud task which is then added to the cloud tasks queue and executed at some specified time.
- To create a task we need to specify certain parameters to it
  - Service account email which gives it permission to execute private cloud functions.
  - The cloud function (url) to execute
  - Scheduled time of the tasks
  - Cloud task queue name and location
- All these parameters are then used to create a task which is then added to the cloud tasks queue.
- Scheduled time varies for different tasks. For the countdown task the scheduled time is one minute before the event start time. For the eventStart task the scheduled time is the event start time. For finishedEventTask the scheduled time is the event finish time.

5.3 Installations instructions

1. **React-Native Application Setup**
    - Install node.js and Npm by following these instructions depending on your operating system. (Downloading and installing Node.js and npm)
    - Since the application uses react-native environment, that needs to be set up first.
    - Unzip the code file and navigate to the root folder, then run `npm install`, this would install all the required node modules mentioned in the package.json file.
    - In the root folder run `react-native eject` which will generate the android and ios folders.
    - In order to run the app we'll need to setup android studio as well as mentioned in [7].
    - Enable the Directions, Places, Distance Matrix, Geocoding, Maps SDK for Android Google API from the Google Cloud Console.
    - Configure the Google API Key in the Android Manifest File.(Get an API Key | Maps SDK for Android)
    - Since Firebase is needed, we need to create an Android firebase project and insert the google-services.json file in the $PATH_TO_PROJECT/android/app folder
    - Refer to [9] to set up firebase for android and IOS.
    - After android setup is done, run **npm run android** command, this would then start building the code and if you are using an android emulator, that will be opened or if you are making use of you phone (with USB debugging enabled) app will be opened on your phone.
    - After creating an account and logging into it, you will be able to make use of the app.

2. **Cloud Function Deployment**
    - Ensure the firebase project is setup.
    - Install the firebase CLI on your operating system.
    - Initialize and configure your project using the firebase CLI.
    - In the root of the "functions" folder run: `firebase deploy --only functions`

6. Conclusion

Through Fitboard we learnt to implement a truly serverless app by making use of various GCP components like Cloud firestore, Firebase authentications, Cloud Messaging, Cloud tasks queues and Cloud functions. Connecting all these components helped us in implementing an app which can autoscale according to the user demands and traffic. We also learned to manage user authentication states by storing them in the device cache, to persist user sessions.
By implementing the real time event map we learnt about tracking user locations, updating them to the firestore and various firestore listeners which we then used to render updated user data on the map in the form of the markers.
We made use of react-native to build the app interface and functionalities. We heavily used google APIs like Places API, Directions API, Distance API, Places Autocomplete API etc. hence we learnt to make asynchronous queries to fetch the data and render them appropriately using react states. We learnt about react hooks and their advantages in navigations over react class components.

Possible ideas for improvement :-

- Add a global leaderboard for the users so that they can be compared across the events and provide more incentive to a user to take part in the event.
- Add anti-gps simulator checks so that users can’t use simulators to take part in the event.
- Partnering up with event organizers and allowing integration of their events with our platform.
- Introduce a feature of monetary rewards in-order to provide better incentives to the user.
- Improving the UI of the application to make it more appealing and user friendly.
- Reduce the number of reads during the event by using a firestore listener running on a common controller, which pushes the location updates via web sockets to all devices participating in the event, instead of using a listener on each device.

7. References

[1] Google cloud tasks documentation - https://cloud.google.com/tasks/docs
[2] Google places API - https://developers.google.com/places/web-service/intro
[3] Google AutoComplete - https://developers.google.com/places/web-service/autocomplete
[4] Google Maps Distance API - https://developers.google.com/maps/documentation/distance-matrix/start
[5] Google Maps Direction API - https://developers.google.com/maps/documentation/directions/intro
[6] React Native Android Setup - https://reactnative.dev/docs/environment-setup
[7] Android studio setup - https://developers.facebook.com/docs/react-native/configure-android/
[8] Cloud Functions For Firebase - https://firebase.google.com/docs/functions
[9] React-Native Firebase - https://rnfirebase.io/
[10] Serverless architecture - https://cloud.google.com/serverless/whitepaper
[11] Cloud Messaging Evaluation - https://cse.buffalo.edu/~demirbas/publications/gcm.pdf
[12] PaaS - https://www.omg.org/cloud/deliverables/CSCC-Practical-Guide-to-PaaS.pdf
[13] Microservices - https://cloud.google.com/files/Cloud-native-approach-with-microservices.pdf