EVENT-ME

1. Project Title & Authors

Authors: Sri Doupaty, Mourad Zeynalov, James Fan
Team Number: #11
Team Name: Sauced Up SWEs
Emails: doupaty@usc.edu, zeynalov@usc.edu, jlfan@usc.edu

2. Preface

In a world that is becoming increasingly virtual, real human connection has become something that is hard to come by. However, now with EventMe, we can change this reality. With just the click of a few buttons, you can find any type of social event based on your choosing. You can attend these events and meet lifelong friends! In our post covid world, this app will change the way people live and interact with those around them, and foster a global sense of community.

This document will detail the implementation of EventMe as a process, the changes that were made, and the features that we implemented. This document will explain the functionality of the app and how it is meant to work.

The expected readership of this document is technical professionals and users interested in learning about the application's functions. It will allow these people to gain a deeper understanding of how EventMe was built.

Version 1.0 - Basic features (Bottom Navigation Bar, Login/Register, User Profile, Event Box)

3. Introduction

EventMe is an up and coming new app on the app store designed for the post-Covid world. For the first time in years people are now able to leave their homes on a regular basis to go out and have fun, however, this creates a problem. These events that people want to go to are often very expensive, and it might be difficult to find out about them.

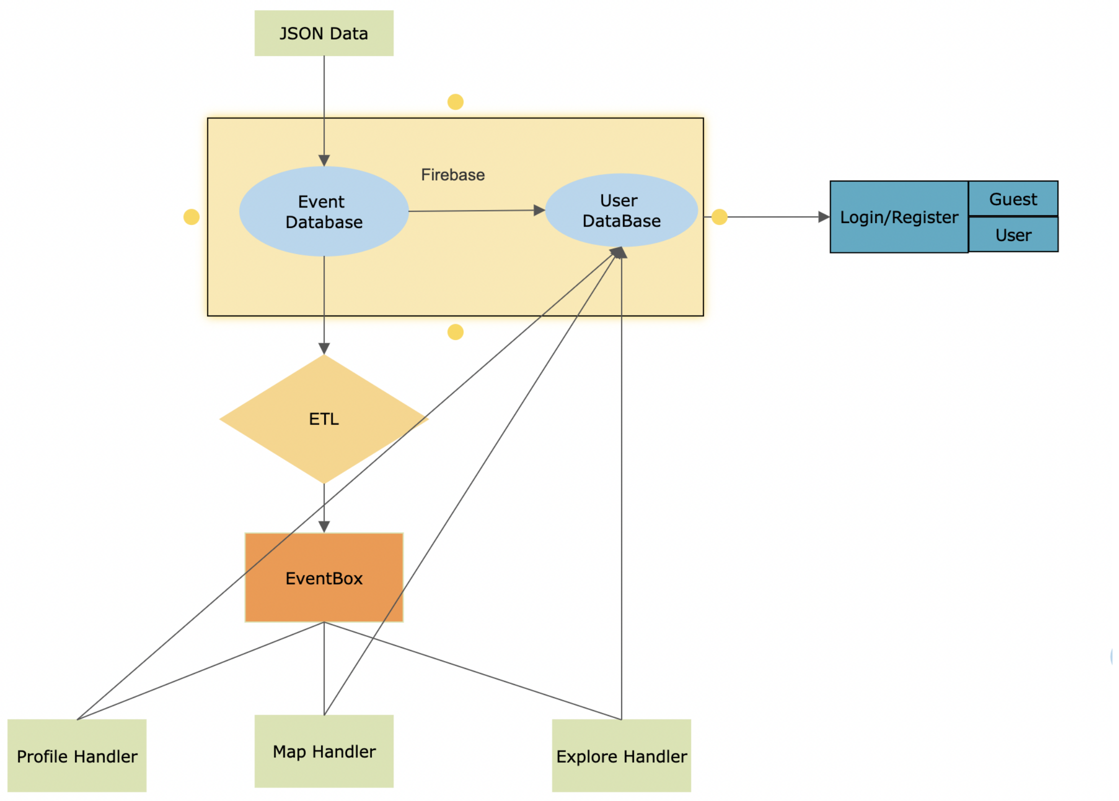
A lot of these cheap events need people, but don't have any way of advertising their events as they might not have large budgets. This is where EventMe comes in. EventMe connects these events to people looking for fun activities for cheap. It's a win-win situation for everyone involved.

This document will detail the process of implementing the major features of this app such as the NavBar, Google Maps, the Explore Page, The Profile Page, Login/Register, and the Register Events page.

We acknowledge that the software development process is not one that is generally very clean, and lots of changes generally need to be made in order to deliver a working product for the customer, and as such a lot of this document's focus will also be placed on explaining the various changes we made from our ideation/pre-implementation phase.

We will be detailing the changes we made and our rationale first at a high level architectural level, and then going into a detailed description of what is different as well as why we made things the way they are.

4. Architectural Design



Architectural Configuration: We decided to go with the Repository architectural configuration. This option serves as the best configuration for EventMe due to the data centered nature of the app, whereby the app must maintain a large repository of user information and event information of which the app must be able to access and filter on the fly. As such we designed the infrastructure of the app around manipulating and displaying the information stored in the central database. We also desired the information stored in the database, such as user profile information, to persist after app shutdown.

Most aspects of the previous architectural diagram will remain the same, there are only a few small changes that we made during the implementation of the program.

JSON data of events still gets transferred through an ETL Data pipeline, parses the JSON data into EventBox objects, which are then stored in an event database.

Event Database stores information about all searchable events, including the necessary information about a particular event, such as event name, location, date, time, cost, and sponsoring organization.

Simultaneously, new users register their name, email, username and password. For users who have already registered they can have their name, birthday, photo, and a list of events for which they are already signed up for, which are all then stored in a user profile database.

We made a few changes to this diagram however, notably in that we made connections between Explore Handler and Map Handler to the User Database. Both of these functions access the User Database as some User can register for events through both of these functionalities, whereby the registered event is then sent to the User's respective UserBox in the database.

We made these changes as we realized that users needed to be able to register for events through these functionalities, and thus needed a connection to the User Database in order to insert these registered events into the database, a fact that we hadn't considered before.

Components

Database Components

Event Database: Firebase database that stores the raw JSON data for the event information. The attributes stored should be the event's name, location, date, time, cost, and sponsoring organization.

User Profile Database: Separate JSON table that contains user information. The attributes stored should include the user's name, email, birthday, password, and a photo that they can upload.

Event Box: The event box component accesses the memory from the event database, whereby the event database returns an eventbox object that can either be displayed, or passed on to the map

Application Components

Explore Handler, Map Handler, and Profile Handler are the classes that define the behavior of the Android pages. Each is composed of a List of EventBox. This is because they use the attributes and methods that were defined in Event Box, especially those associated with information about events.

Explore Handler: The Explore Handler queries the Database for a list of EventBox objects, whereby it can then call attributes and methods from EventBox to display these events. As an example, Explore Handler would query the database for all of the events, and then say the user would want to filter by cost, explore handler would sort the array by cost by accessing each EventBox's getCost() method to get the cost of attendance for each event, and then displaying these boxes in the proper order.

Profile Handler: The Profile Handler component accesses the memory from the EventBox object and the UserBox object. Profile Handler calls most of the attributes and methods from EventBox as profile handler needs to display the User's registered events. It also calls on UserBox however as it needs to display the User's attributes such as birthday, profile picture, etc, in the profile tab.

Map Handler : The Map Handler component accesses the list of allEvents from the JSON, whereby it plots the locations of the Events on the map using the longitude and latitude attributes of the events. It then calculates the current location of the user in order to calculate the distance from these events.

Login/Register - The Login/Register component is in charge of managing the user experience with the app. The user can use EventMe without logging in, but any time they would like to use the main functionality of the app, i.e, registering for events, they are prompted to login or register to EventMe. These also access the database in multiple ways, and are the main gatekeeper of communication between the app and the database, without being registered and logged into the app one cannot access the database of EventMe in any form.

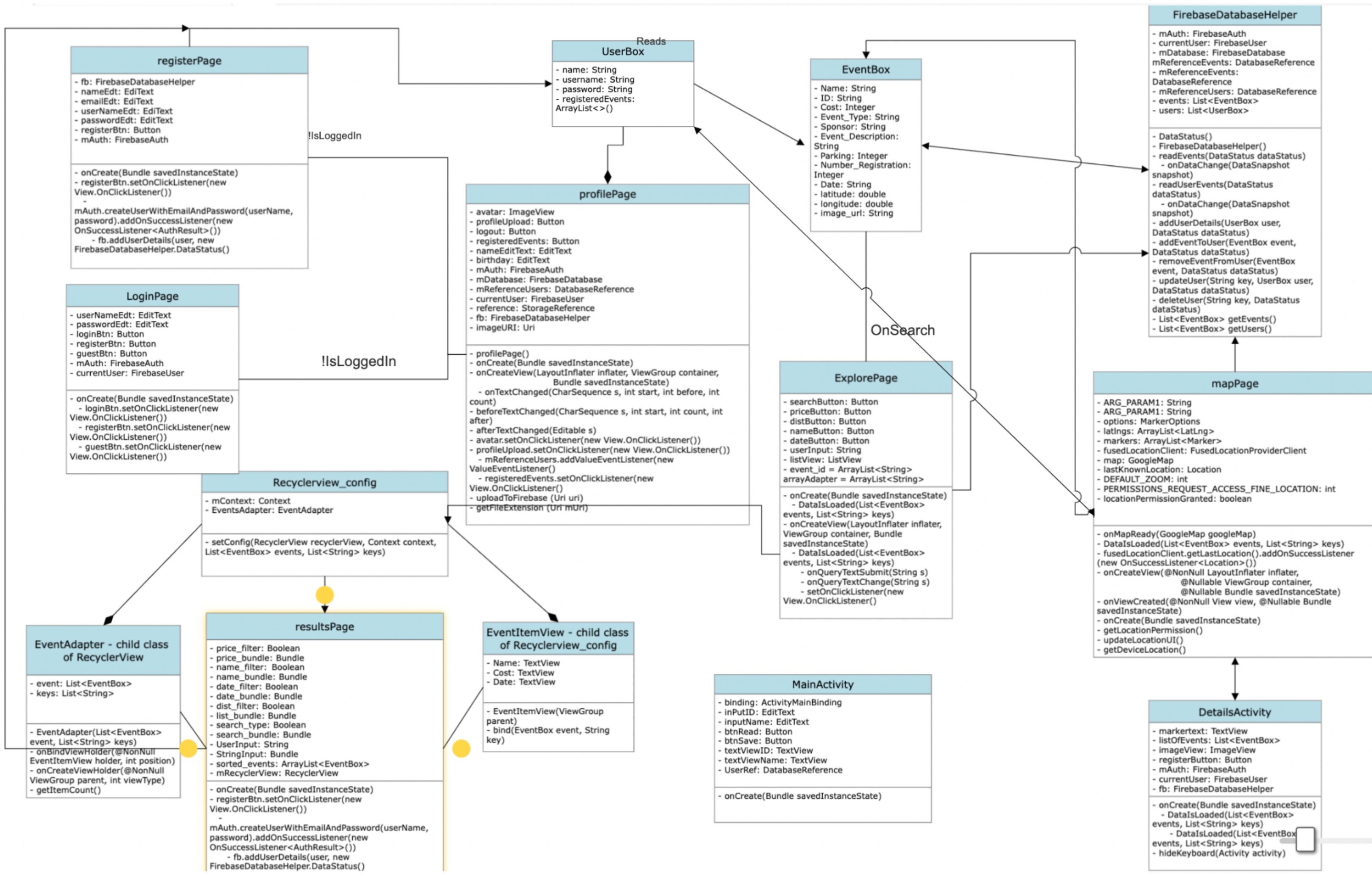
5. Detailed Design

1. Class Diagram:

There were quite a few changes here which we needed to make in order to streamline the implementation of EventMe, as well as features of which we didn't take into account prior to the implementation of EventMe.

The new features:

FirebaseDatabaseHelper: The FirebaseDatabaseHelper file works as an ETL (Extract Transform Load) pipeline where it read events from the Firebase Realtime Database, collects the events information (e.g. price, latitude,


registerPage
- fb: FirebaseDatabaseHelper
- nameEdt: EdiText
- emailEdt: EdiText
- userNameEdt: EdiText
- passwordEdt: EditText
- registerBtn: Button
- mAuth: FirebaseAuth
- onCreate(Bundle savedInstanceState)
- registerBtn.setOnClickListener(new View.OnClickListener())
- mAuth.createUserWithEmailAndPassword(userName, password).addOnSuccessListener(new OnSuccessListener<AuthResult>())
- fb.addUserDetails(user, new FirebaseDatabaseHelper.DataStatus()
!IsLoggedIn
LoginPage
- userNameEdt: EditText
- passwordEdt: EditText
- loginBtn: Button
- registerBtn: Button
- guestBtn: Button
- mAuth: FirebaseAuth
- currentUser: FirebaseUser
- onCreate(Bundle savedInstanceState)
- loginBtn.setOnClickListener(new View.OnClickListener())
- registerBtn.setOnClickListener(new View.OnClickListener())
- guestBtn.setOnClickListener(new View.OnClickListener())
!IsLoggedIn
Reads
UserBox
- name: String
- username: String
- password: String
- registeredEvents: ArrayList<>()
profilePage
- avatar: ImageView
- profileUpload: Button
- logout: Button
- registeredEvents: Button
- nameEditText: EditText
- birthday: EditText
- mAuth: FirebaseAuth
- mDatabase: FirebaseDatabase
- mReferenceUsers: DatabaseReference
- currentUser: FirebaseUser
- reference: StorageReference
- fb: FirebaseDatabaseHelper
- imageURI: Uri
- profilePage()
- onCreate(Bundle savedInstanceState)
- onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState)
- onTextChanged(CharSequence s, int start, int before, int count)
- beforeTextChanged(CharSequence s, int start, int count, int after)
- afterTextChanged(Editable s)
- avatar.setOnClickListener(new View.OnClickListener())
- profileUpload.setOnClickListener(new View.OnClickListener())
- mReferenceUsers.addValueEventListener(new ValueEventListener()
- registeredEvents.setOnClickListener(new View.OnClickListener()
- uploadToFirebase (Uri uri)
- getFileExtension (Uri mUri)
EventBox
- Name: String
- ID: String
- Cost: Integer
- Event_Type: String
- Sponsor: String
- Event_Description: String
- Parking: Integer
- Number_Registration: Integer
- Date: String
- latitude: double
- longitude: double
- image_url: String
OnSearch
ExplorePage
- searchButton: Button
- priceButton: Button
- distButton: Button
- nameButton: Button
- dateButton: Button
- userInput: String
- listView: ListView
- event_id = ArrayList<String>
arrayAdapter = ArrayList<String>
- onCreate(Bundle savedInstanceState)
- DataIsLoaded(List<EventBox> events, List<String> keys)
- onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState)
- DataIsLoaded(List<EventBox> events, List<String> keys)
- onQueryTextSubmit(String s)
- onQueryTextChange(String s)
- setOnClickListener(new View.OnClickListener()
FirebaseDatabaseHelper
- mAuth: FirebaseAuth
- currentUser: FirebaseUser
- mDatabase: FirebaseDatabase
mReferenceEvents: DatabaseReference
- mReferenceEvents: DatabaseReference
- mReferenceUsers: DatabaseReference
- events: List<EventBox>
- users: List<UserBox>
- DataStatus()
- FirebaseDatabaseHelper()
- readEvents(DataStatus dataStatus)
- onDataChange(DataSnapshot snapshot)
- readUserEvents(DataStatus dataStatus)
- onDataChange(DataSnapshot snapshot)
- addUserDetails(UserBox user, DataStatus dataStatus)
- addEventToUser(EventBox event, DataStatus dataStatus)
- removeEventFromUser(EventBox event, DataStatus dataStatus)
- updateUser(String key, UserBox user, DataStatus dataStatus)
- deleteUser(String key, DataStatus dataStatus)
- List<EventBox> getEvents()
- List<EventBox> getUsers()
mapPage
- ARG_PARAM1: String
- ARG_PARAM1: String
- options: MarkerOptions
- latlngs: ArrayList<LatLng>
- markers: ArrayList<Marker>
- fusedLocationClient: FusedLocationProviderClient
- map: GoogleMap
- lastKnownLocation: Location
- DEFAULT_ZOOM: int
- PERMISSIONS_REQUEST_ACCESS_FINE_LOCATION: int
- locationPermissionGranted: boolean
- onMapReady(GoogleMap googleMap)
- DataIsLoaded(List<EventBox> events, List<String> keys)
- fusedLocationClient.getLastLocation().addOnSuccessListener (new OnSuccessListener<Location>())
- onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup container, @Nullable Bundle savedInstanceState)
- onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState)
- onCreate(Bundle savedInstanceState)
- getLocationPermission()
- updateLocationUI()
- getDeviceLocation()
Recyclerview_config
- mContext: Context
- EventsAdapter: EventAdapter
- setConfig(RecyclerView recyclerView, Context context, List<EventBox> events, List<String> keys)
EventAdapter - child class of RecyclerView
- event: List<EventBox>
- keys: List<String>
- EventAdapter(List<EventBox> event, List<String> keys)
- onBindViewHolder(@NonNull EventItemView holder, int position)
- onCreateViewHolder(@NonNull ViewGroup parent, int viewType)
- getItemCount()
resultsPage
- price_filter: Boolean
- price_bundle: Bundle
- name_filter: Boolean
- name_bundle: Bundle
- date_filter: Boolean
- date_bundle: Bundle
- dist_filter: Boolean
- list_bundle: Bundle
- search_type: Boolean
- search_bundle: Bundle
- UserInput: String
- StringInput: Bundle
- sorted_events: ArrayList<EventBox>
- mRecyclerView: RecyclerView
- onCreate(Bundle savedInstanceState)
- registerBtn.setOnClickListener(new View.OnClickListener())
- mAuth.createUserWithEmailAndPassword(userName, password).addOnSuccessListener(new OnSuccessListener<AuthResult>())
- fb.addUserDetails(user, new FirebaseDatabaseHelper.DataStatus()
EventItemView - child class of Recyclerview_config
- Name: TextView
- Cost: TextView
- Date: TextView
- EventItemView(ViewGroup parent)
- bind(EventBox event, String key)
MainActivity
- binding: ActivityMainBinding
- inPutID: EditText
- inputName: EditText
- btnRead: Button
- btnSave: Button
- textViewID: TextView
- textViewName: TextView
- UserRef: DatabaseReference
- onCreate(Bundle savedInstanceState)
DetailsActivity
- markertext: TextView
- listOfEvents: List<EventBox>
- imageView: ImageView
- registerButton: Button
- mAuth: FirebaseAuth
- currentUser: FirebaseUser
- fb: FirebaseDatabaseHelper
- onCreate(Bundle savedInstanceState)
- DataIsLoaded(List<EventBox> events, List<String> keys)
- DataIsLoaded(List<EventBox> events, List<String> keys)
- hideKeyboard(Activity activity)

longitude, etc.) and stores said information in a list. What's changed from our original implementation of our ETL data pipeline is that FirebaseDatabaseHelper also handles storing information for user detail storing, especially in terms of adding and removing events from the user object.

DetailsActivity: A newly added file from our original class diagram, DetailsActivity shows the events on the Google Maps graphic user interface on the Map tab. The events are displayed through a series of red tick markers - which also display the name of the events - which highlight where the events are situated visually on the map.

resultsPage: A newly added file from our original class diagram, resultsPage handles displaying the filtered events (passed in from ExplorePage). As it recognizes how the events should be filtered, it handles how the events are filtered and how they are displayed using RecyclerView.

RecyclerConfig_view: This class is needed to implement RecyclerView, which displays elements in a list like order top to bottom. This class is used to help display resultsPage after searching or querying in Explore Page, as well as used to display any series of events throughout the app.

Old Features with some changes:

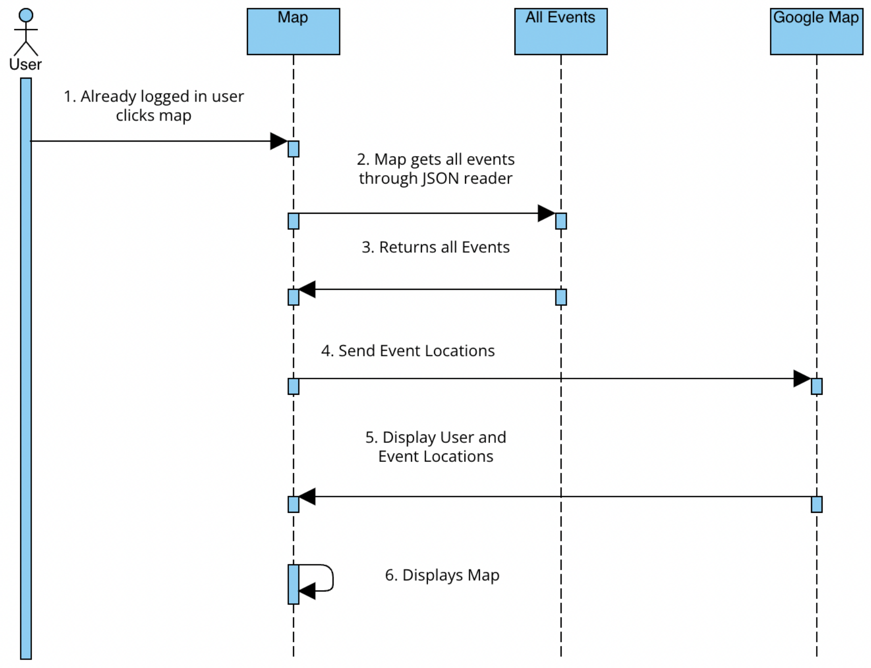
Map Page: Mostly everything stayed the same in this page, however we added some functionality to get the user's current location so that the user can filter the events by distance from the user. This functionality wasn't included in the previous document.

ExplorePage: It displays how the user can search for or filter by available events. It uses buttons for filtering, which (when clicked) notifies the resultsPage to display the events by the filter that the user "clicked on". Users can choose from a variety of filters, such as by price, alphabetically, nearest distance, and soonest by date.

2. Sequence Diagrams:

Most of the events which were described by the sequence diagrams in document 2.2 ended up following the same program flow in our implementation of EventMe. However, following our implementation, only one sequence diagram changed. The updated Sequence Diagram is displayed below.

User uses Map to find Nearby Events


User
Map
All Events
Google Map
1. Already logged in user clicks map
2. Map gets all events through JSON reader
3. Returns all Events
4. Send Event Locations
5. Display User and Event Locations
6. Displays Map

1. The "Map" is part of the Map handler, a component in the architectural design which deals with the Map feature of the program, a feature which will allow the user to locate nearby events as well as pinpoint their location on the map.
2. "AllEvents" is a function of the explore handler which queries the database for all its events, and then returns them to an array list.

This sequence event was changed as we no longer have implemented the swipe up feature on the mapPage. Instead, the mapPage uses Google Maps API to display the map, including the user's current location (based on the Emulator's designated location) and the events that are read in from the Firebase Realtime Database.

6. Requirements Change

a)

In our previous document regarding requirements, we mentioned that we would implement a Swipe-Up feature on the Maps page. Specifically, this swipe-up action was to be implemented when the user navigates to the Map tab and clicks on an event from the Map that they wish to register for; swiping up would be the action to register for the action.

As a group, we decided that the swipe-up functionality was not necessary to be implemented. This is because we have implemented red tickers on events that are available to be registered for. Tapping/clicking on the red tickers activates the resultsPage, where it displays a card that displays information about the selected event: a picture of the event, event name, price, etc. There, the user can read and learn more about the event. As such, we implemented a register event option on the resultsPage for the specified event.

As a result, we feel that this provides a more user-interactive experience, and clicking on the red tickers achieves the same effect of registering to a higher degree - if not better because the user can read information about the event on the resultsPage rather than simply swiping up on the event even though the user may not know anything about the event.

Moreover, the ExplorePage offers many filter options that are available for the user to choose. Lastly, the removal of the swipe-up feature eliminates accidental swiping up on the home button of Android devices, which may cause the app to unexpectedly quit (from personal experience). This improvement in user experience was carefully accounted for to enrich the most interactive yet seamless app experience.

b)

<u>Changed Requirements:</u>

<u>Removed Swipe-Up functionality</u>
i) Yes, this changes our design. There is no swipe-up functionality when attempting to register events through this function.
ii) N/A
iii) We changed that clicking on a red ticker brings up the resultsPage. The resultsPage for the clicked on event also has a register event functionality. This makes the swipe-up functionality redundant and unnecessary.