

CMPE 277 – Spring 2016 – Final Project

Event Share – Android App



Submitted By,

Team Members	SJSU ID
Anjaneya Murthy Gabbiti	010090444
Bhuvan Teja Guddanti	010096099
Prashanth Mudhelli	010105095
Priyanka Ravikanti	010090405
Swarna Viswanathan	010764403



Under the guidance of:
Prof. Chandrasekar Vuppalapati

Table of Contents:

PROJECT DESCRIPTION	4
REQUIREMENTS.....	5
MOBILE UI DESIGN PRINCIPLES.....	5
Story Board	6
Mobile UI Wire Frames.....	7
HIGH LEVEL ARCHITECTURE DESIGN	11
WORK FLOW DIAGRAM	12
INFOGRAPHICS – SEQUENCE FLOW.....	13
MOBILE & CLOUD TECHNOLOGIES USED.....	14
Loopj Library	14
Node JS Design.....	14
MongoDB.....	15
Amazon EC2	16
Build IO	16
INTERFACES – RESTFUL & SERVER SIDE DESIGN	18
CLIENT SIDE DESIGN	19
Albums and Photos	19
Maps Tracking.....	21
TESTING.....	21
AUTOMATION TESTING	26
DESIGN PATTERNS USED.....	28
Cloud Pattern.....	28
MVC Pattern	28
INDIVIDUAL CONTRIBUTION.....	29
SCREEN SHOTS.....	30

Table of Figures:

Figure 1: Story Board	6
Figure 2: Login	7
Figure 3: Create Event Wireframe.....	7
Figure 4: Image View Wireframe	8
Figure 5: Navigation Menu Wire Frame	8
Figure 6: Accepted Requests Wireframe	9
Figure 7: Pending Requests Wire Frame	9
Figure 8: Maps Wire Frame.....	10
Figure 9: Work Flow Diagram	12
Figure 10: Infographics	13
Figure 11: NodeJS Architecture	15
Figure 12: Mongo DB Screen Capture	16
Figure 13: MVC Architecture.....	29

Project Description:

Currently there are many Android applications, which are used to track the users and share their current location with all the other users. This will enable all the users in an event to track their friends and view their details on a Map. There are some other applications, which are used to share the photographs with their friends. But there is no application, which provides location tracking and sharing of photos between friends. So we have come up with this idea of creating events and sharing the photos among all the users in the event and tracking all your friends.

Event share is an android application that is used to create events and share events among various users based on their email Ids. User can login to the application using their Gmail ID, we have used Gmail oath authentication to login to the application. Once user logs in to the application he can be able to create an event and invite his friends to an event based on their Gmail Ids. Once an event is created the request will be sent to all the invited users to do an RSVP of the event. Create an event and that's it our application will take care of pics sharing and tracking the users. Users can be able to view the list of events to which he was invited and can be able to accept or reject the event. If the user accepts the event he will be notified about the event in advance. Once an event starts all the friends who accepted the event can be able to track the real time location of their friends on a map. Friends can be able to share the photos taken during the event to all the people in the event. When a user creates an event he will mention start time and end time of the event, so based on this photos taken during this time are being shared among all the friends after the event.

Let us consider a scenario where a group of friends planned to go to a picnic spot, then a person will create an event giving the details such as location, name, start and end time of the event and the list of invitees. So invitees will get a request to do an RSVP. Based on this RSVP list their location will be shared and tracked among all the accepted people. After the event end date all the images are shared among all the invitees. Thus reducing the time of photo sharing.

Requirements:

Event share is an app that is mainly used to track the people who are attending the event and share the photos after the event among the people who attended the event. Thus based on the end user perspective we have considered the following requirements.

- Gmail Authentication for users.
- When an event is created a request has to be sent to all invitees which user has entered.
- Location tracking should be enabled among all the people who accepted the event and every person in the event should be able to track all the people in the event.
- Photos taken during the time of the event must be shared among all the people who accepted the event.
- Photos taken from all the friends should be shared among all the people in the event.
- User should be able to accept or reject invitation for an event.
- Photos uploaded should be sorted based on event name and are to be displayed to the user.

Mobile UI Design Principles:

We have created a user storyboard to explain about the flow of the UI pages. We also have created layout files (xmls) well in advance. We have designed an overview of items needs to be displayed.

The below diagram shows the UI layouts and the activities corresponding to each layout.

Story Board

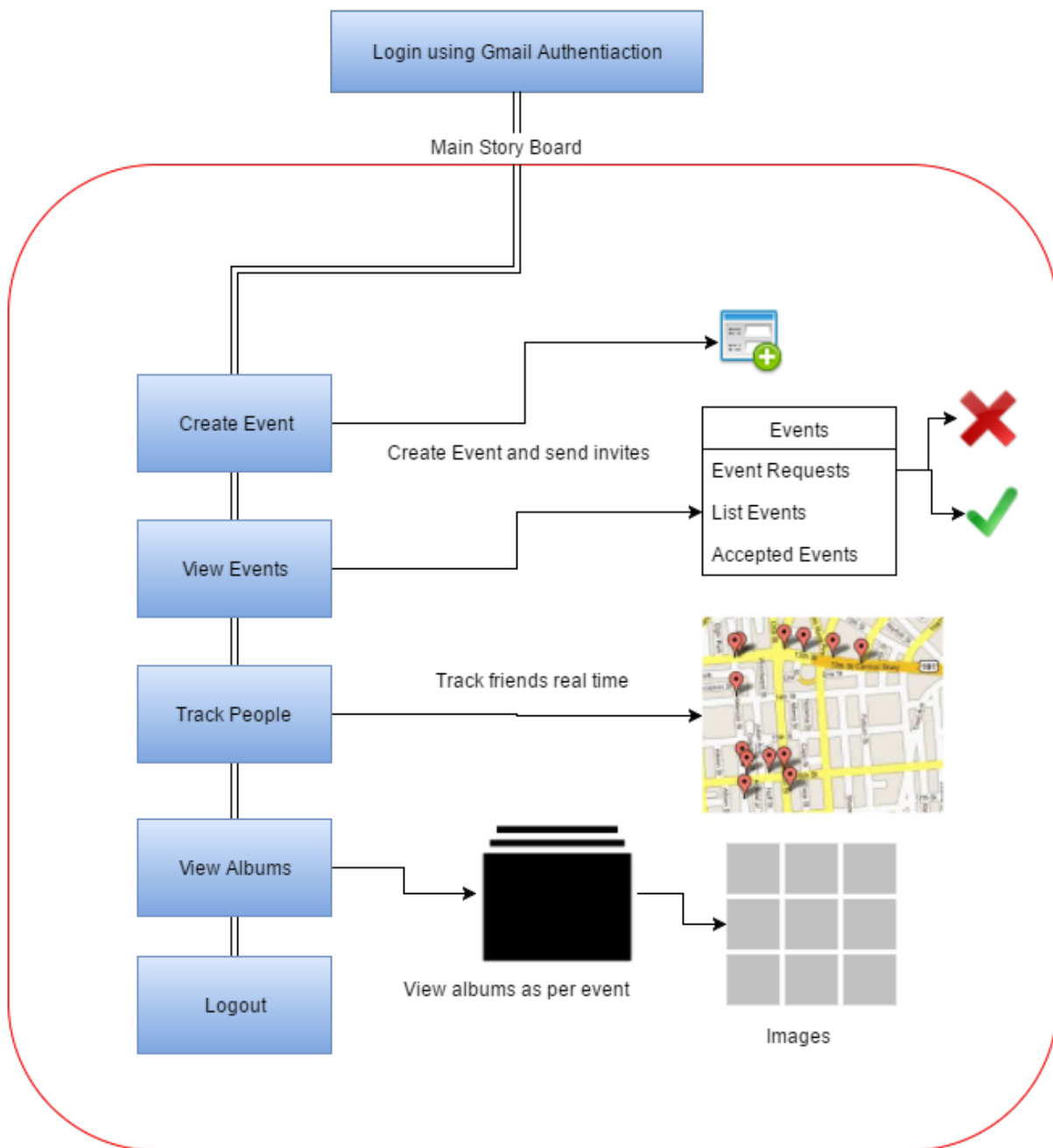


Figure 1 Story Board

Mobile UI Wire Frames:

We have build UI wireframes to understand the functionality of the application well.

Login Screen of Application:

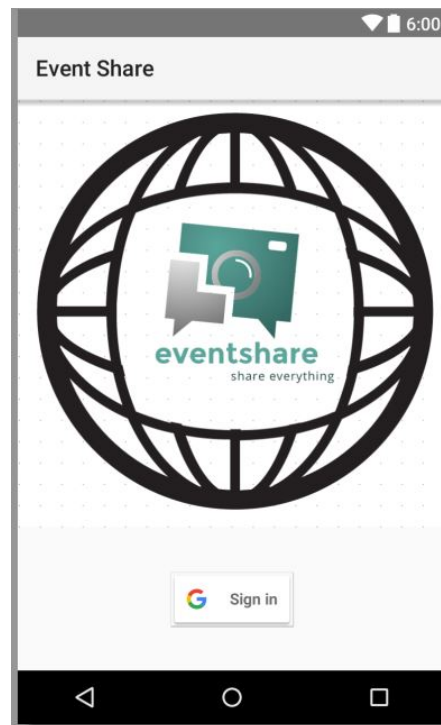


Figure 2 Login

Create Event Screen:

A mobile app "Create Event" screen wireframe. At the top, a status bar shows signal, battery, and time (6:00). Below is a header bar with the text "Event Share". The form contains several input fields: "Enter event name", "Enter location", "Start Day" and "Start Time" (side-by-side), "End Day" and "End Time" (side-by-side), and "Invite friends: e.g: xyz@gmail.com". Below these fields is a wide, light-grey button labeled "CREATE EVENT". The bottom of the screen shows the standard Android navigation bar with back, home, and recent apps icons.

Figure 3 Create Event Wireframe

Image View of the Application:

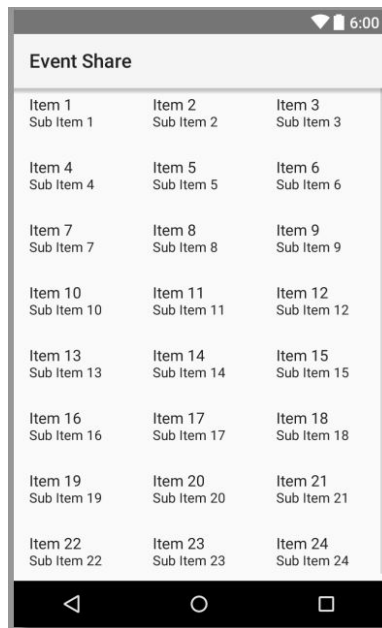


Figure 4 Image View Wireframe

Navigation Menu of the application:

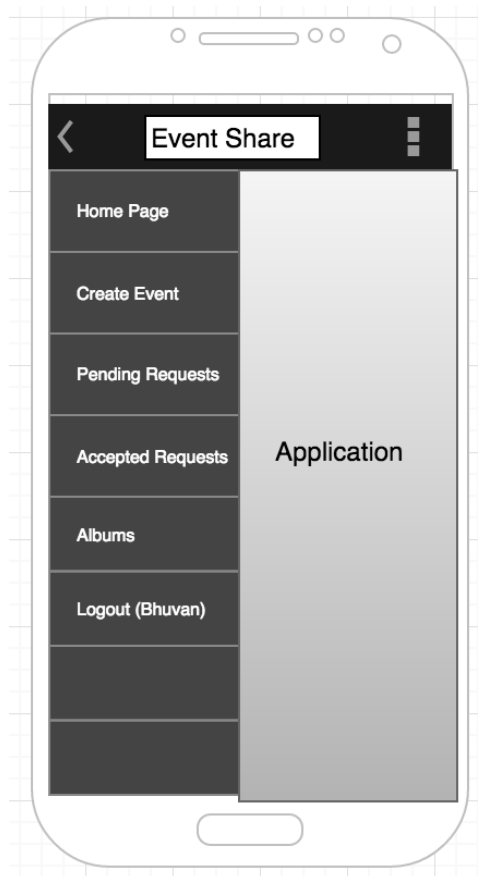


Figure 5 Navigation Menu Wire Frame

Accepted Requests of the Application:

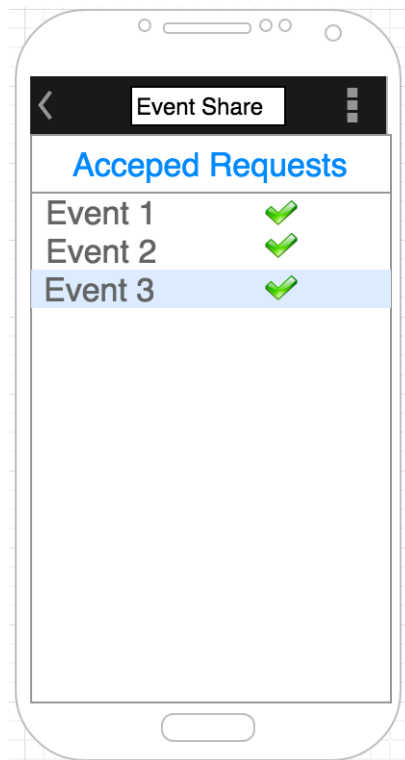


Figure 6 Accepted Requests Wireframe

Pending Requests of the Application:

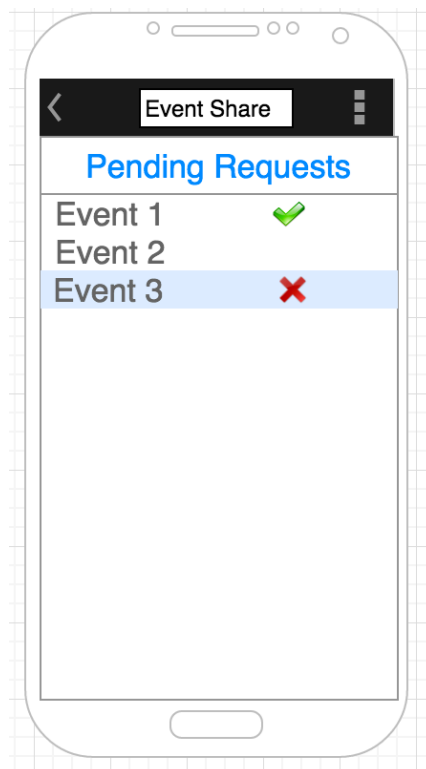


Figure 7 Pending Requests Wire Frame

Maps Activity Tracking Screen of the Application:

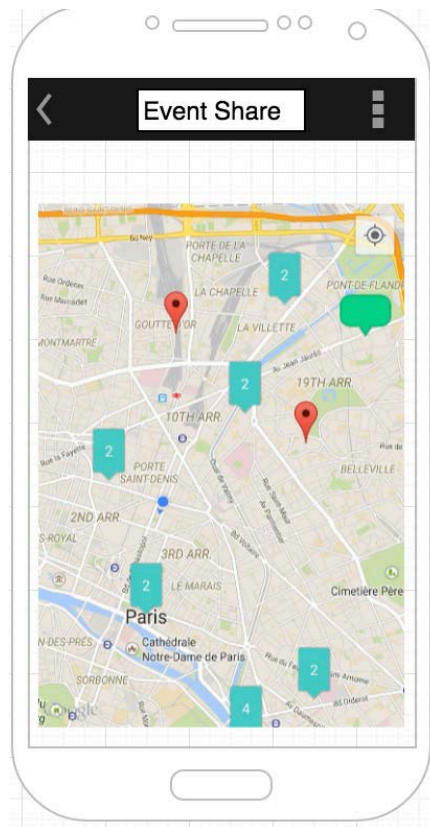


Figure 8 Maps Wire Frame

High Level Architecture Design:

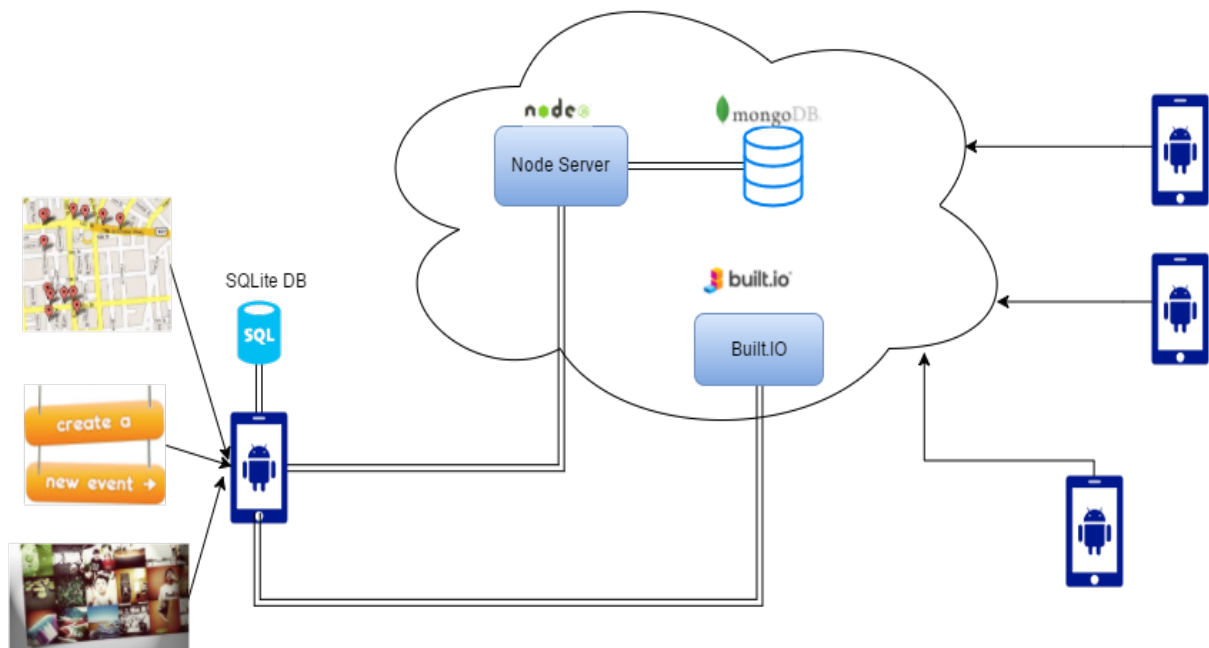


Figure 8 High Level Architecture

In this architecture we are using a Node Server as the backend server to process the data from mobile. We are calling the REST APIs from mobile using a HTTP request to GET or POST the data. We are using MongoDB as a backend database for storing event details and location details of the individual. We have used BUILT.IO which provides backend as a service for storing the images on the cloud. We are using loopj api to make HTTP Rest calls to the NodeJS server. From this we are implementing the functions to add data to MongoDB. Built.IO provides APIS to call from mobile to push the data on to the cloud. Built.IO defines the data to be stored in the form of classes and we can call setters and getters by providing the class name from the android activity. We are using android default SQLite database to save the events data locally in order to save the hits to MongoDB. In Android, we are using AsyncTasks and services to continuously update the location data to the MongoDB. We have defined various API's in order to send and get the data from Mongo DB. We will discuss them in detail in the following sections.

Work Flow Diagram:

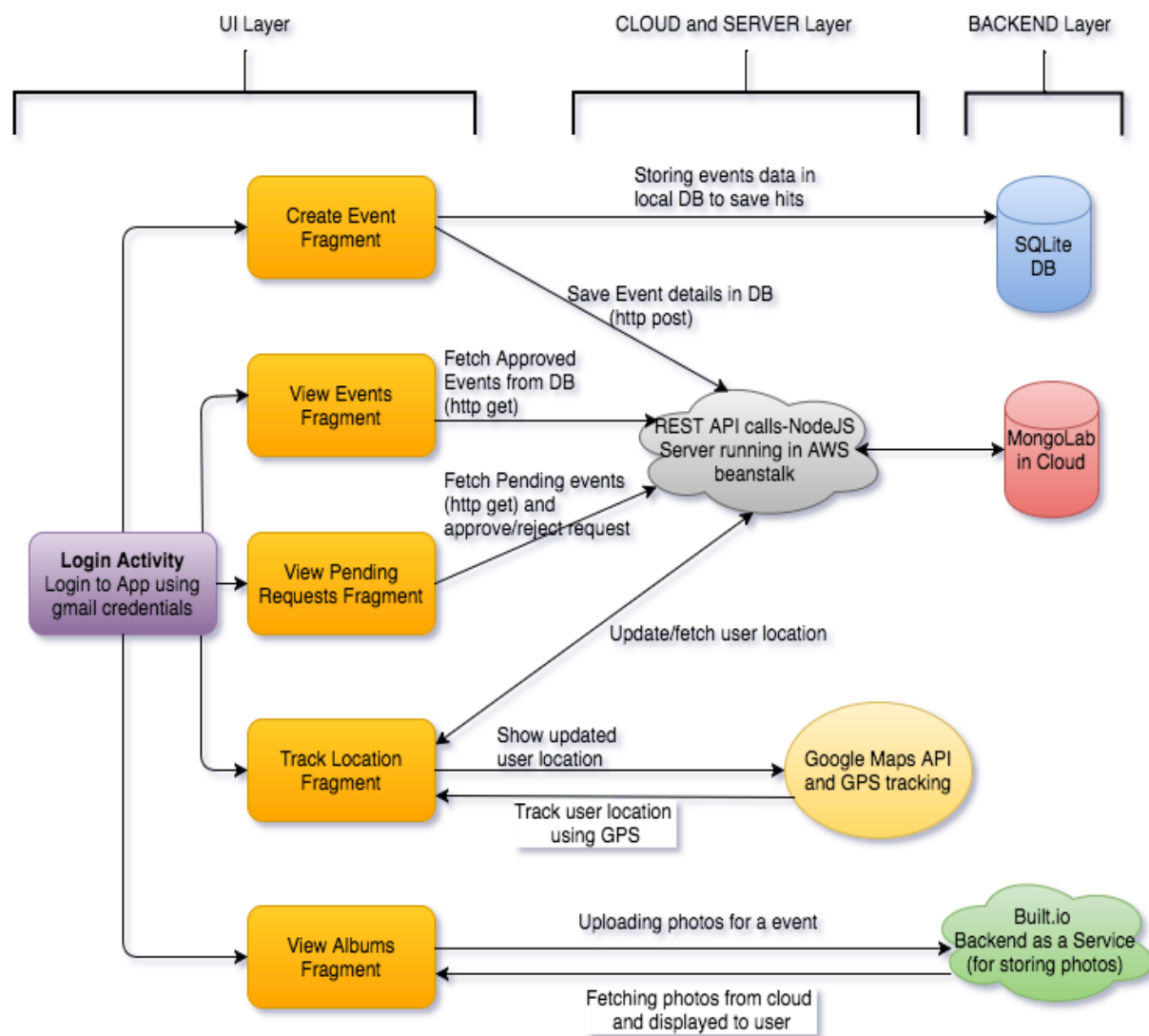
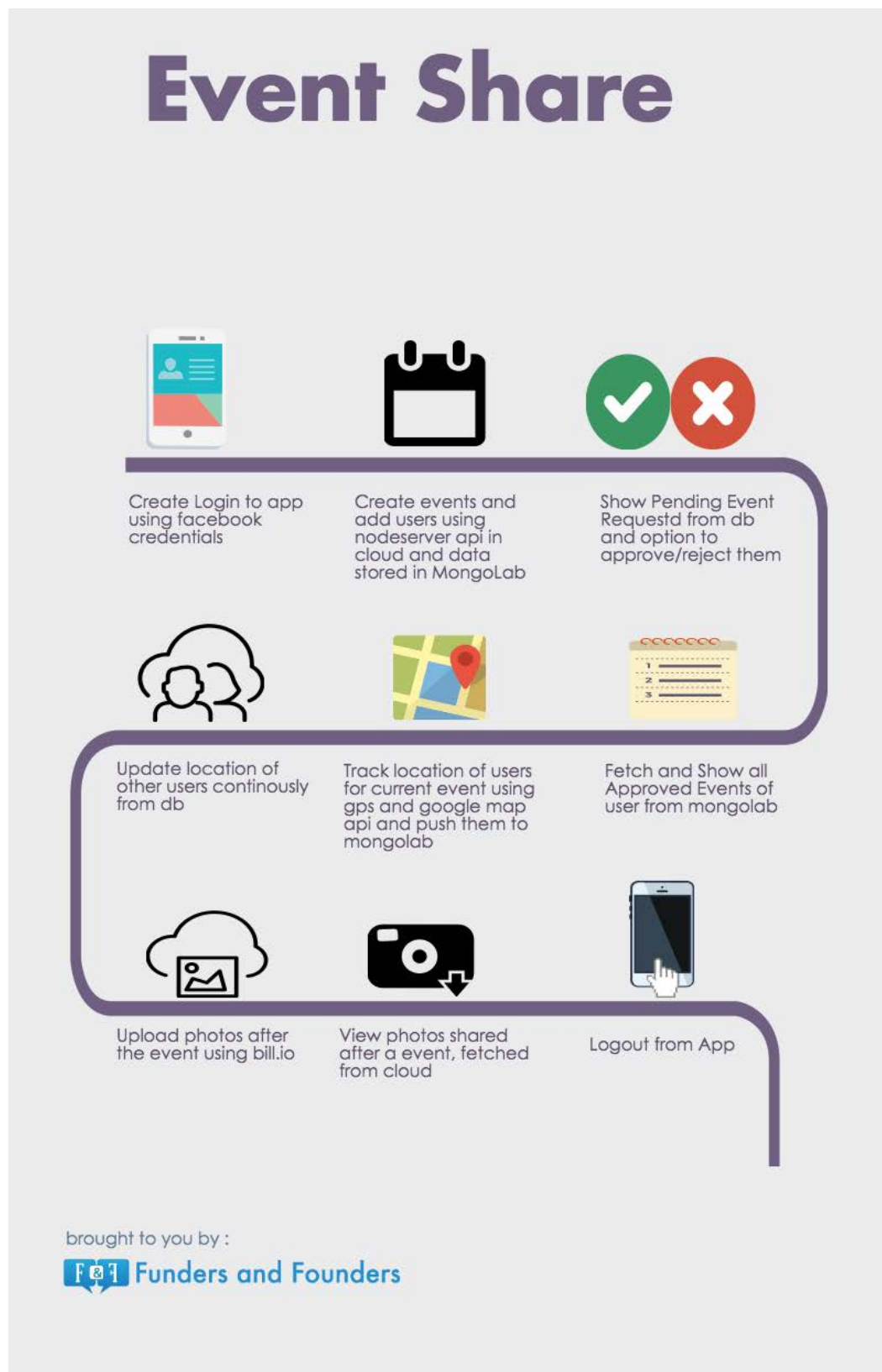


Figure 9 Work Flow Diagram

In this diagram we have discussed about various APIs that are being called from various activities. There are five modules from which APIs are being called. Location fragment and View Albums fragment are using services to make calls to the APIs continuously to update the current location of the person and to upload the pictures onto cloud. APIs are built on NodeJS, which is integrated with MongoDB. Built.IO is a separate cloud, which is used as image repository.

Infographics – Sequence Flow:



Mobile & Cloud Technologies Used:

Loopj Library

Loopj provides APIs for asynchronous Http callbacks for Android. These APIs are built on top of Apache's Http Client libraries. By asynchronous, it means that all the requests are made outside of the applications main thread (UI thread) and are handled using callback functionality once the request is complete. This library can also be used for background services and it inherently understands the application's context in which the operation/service is run.

We used Loopj java library to make asynchronous http calls to the backend Node server from Android. Http based REST APIs implemented in Node JS are called using Loopj library APIs and the corresponding Http responses are processed and rendered using Android.

Node JS Design

We used Node JS to implement the REST API calls. Node JS is a cross platform runtime environment and an open source software to develop web applications (server side). It is not a JavaScript framework, but many of its modules are written in JavaScript. It uses Google's JavaScript V8 engine to interpret JavaScript. It follows an event driven architecture and inherently supports asynchronous I/O operations. Node JS is popular for optimizing throughput and hence it is extensively used for applications that require high scalability and involve heavy IO operations or real time operations.

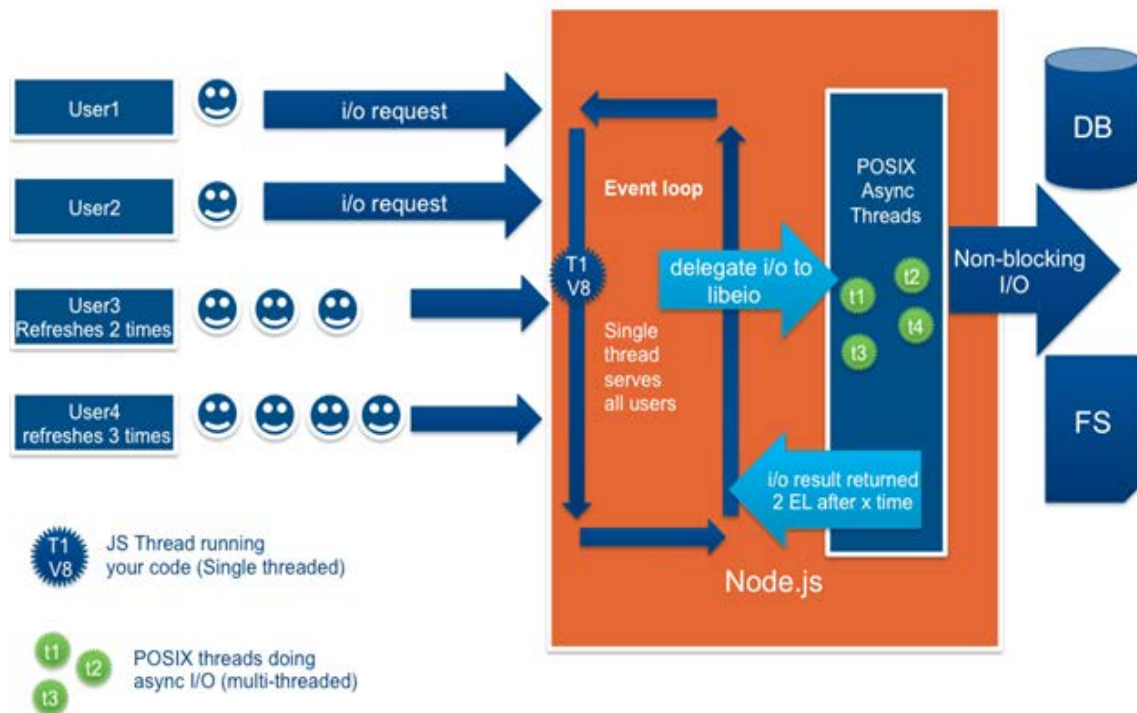


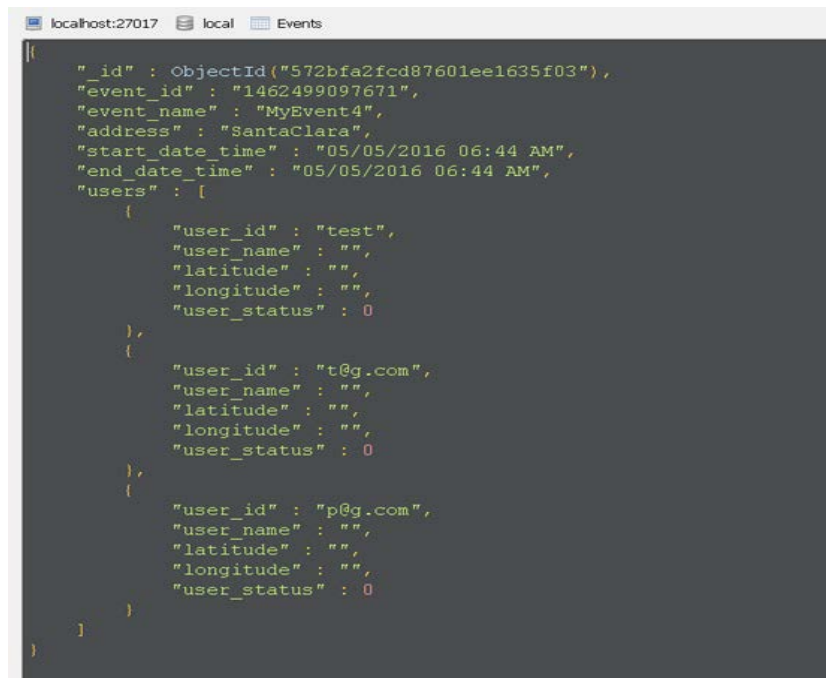
Figure 11 NodeJS Architecture

MongoDB

MongoDB is an open source NoSQL document oriented database. It stores data in JSON like documents and supports dynamic schemas. It claims to offer high performance, scalability and flexibility. We used MongoDB as the backend database to store event related information. Below is a sample event document:

We have considered the below schema for MongoDB

Document Structure: [{event_id, event_name, address, start_date_time, end_date_time,
Users: [{user_id,user_name, latitude, longitude, user_status}]
}]

A screenshot of a MongoDB web interface. The top bar shows 'localhost:27017', 'local', and 'Events'. The main area displays a JSON document. The document has fields: '_id' (ObjectId), 'event_id' (string), 'event_name' (string), 'address' (string), 'start_date_time' (string), 'end_date_time' (string), and 'users' (array of three user objects). Each user object contains 'user_id', 'user_name', 'latitude', 'longitude', and 'user_status'.

```
{
  "_id" : ObjectId("572bfa2fcd87601ee1635f03"),
  "event_id" : "1462499097671",
  "event_name" : "MyEvent4",
  "address" : "SantaClara",
  "start_date_time" : "05/05/2016 06:44 AM",
  "end_date_time" : "05/05/2016 06:44 AM",
  "users" : [
    {
      "user_id" : "test",
      "user_name" : "",
      "latitude" : "",
      "longitude" : "",
      "user_status" : 0
    },
    {
      "user_id" : "t@g.com",
      "user_name" : "",
      "latitude" : "",
      "longitude" : "",
      "user_status" : 0
    },
    {
      "user_id" : "p@g.com",
      "user_name" : "",
      "latitude" : "",
      "longitude" : "",
      "user_status" : 0
    }
  ]
}
```

Figure 12 Mongo DB Screen Capture

Amazon EC2

We made use of Ubuntu virtual machine on the Amazon EC2 instance to deploy our node js components

Below are the following steps to create an EC2 instance:

- Register for Amazon EC2 instance
- Obtain PPK key for login to the instance
- Configure the OS as per the requirements
- Install node js and mongo db NPM modules
- Open the required security ports for access of the application

Build IO

We have integrated our application with Built.io, which provides mobile backend as a service (MBaaS). We are using Built.io only to upload and download our images, as it provides easier access using their straightforward API's without any memory limit. Built.io provides an sdk, which needs to be added to our app/libs folder. Compile statement to compile the sdk needs to be added to the app's gradle file. There are many android-permissions, ranging from Internet to Network-State; all this needs to be added to the manifest file.

Below is the sample code where we use Built.io API.

```
public class DownloadImages extends AsyncTask<String, Void, String> {
    @Override
    protected String doInBackground(String... params) {
        BuiltApplication builtApplication = null;

        builtApplication = Built.application(getApplicationContext(),
"bltc86a080623f49dda");

        BuiltQuery projectQuery =
builtApplication.classWithUid("images").query();
        projectQuery.containedIn("event_id", db.registeredEventIds(userId));
        projectQuery.ascending("event_id");
        projectQuery.execInBackground(new QueryResultsCallBack() {
            @Override
            public void onCompletion(BuiltConstant.ResponseType responseType,
QueryResult queryResultObject, BuiltError error) {
                if (error == null) {
                    List<BuiltObject> objects =
queryResultObject.getResultObjects();
                    String temp = "TEMP";
                    int i = 0;
                    ArrayList<String> indivImageUrls = new
ArrayList<String>();
                    for (BuiltObject object : objects) {
                        i++;

                        if(temp.equals(object.getString("event_id")) ||
temp.equals("TEMP")) {
                            indivImageUrls.add(object.getJSONObject("image").get("url").toString());
                        }
                        else {
                            imageUrl.put(temp, indivImageUrls);
                            indivImageUrls = new ArrayList<String>();
                        }
                        indivImageUrls.add(object.getJSONObject("image").get("url").toString());
                    }
                    temp = object.getString("event_id");
                    if(objects.toArray().length == i) {
                        imageUrl.put(temp, indivImageUrls);
                    }
                }
            }
        });
        return "Fetched";
    }
}
```

Interfaces – RESTful & Server Side Design

We have considered some APIS and started implementing them; we have identified some APIS before we start the Application. Please find below attached list of APIS we used.

Type	API	Param
POST	/createEvent	event_id,event_name,user_id(email_id),user_name, address,start_date_time,end_date_time (yyyy-MM-dd HH:mm:ss)
PUT	/acceptEvent	event_id,user_id,user_name
PUT	/rejectEvent	event_id,user_id,user_name
GET	/getUserEvents	user_id
GET	/getUsersLocation	event_id
PUT	/updateCurrentLocation	event_id,user_id,latitude, longitude

MongoDB Schema:

We have considered the below schema for MongoDB

Document Structure: [{event_id, event_name, address, start_date_time, end_date_time,
Users: [{user_id,user_name, latitude, longitude, user_status}]
}]

Client Side Design

Albums and Photos

Capturing photos is an integral part of any event. The main problem with photos would be getting them to our devices when they were clicked from some others device. We would have faced this experience in requesting all our friends to share our pictures and some of them might get busy in sharing them with us. To solve this problem what we offer in our Event Share app is, no matter on which device our pictures were taken, once the event is done we can see them all in our devices without requesting our friends. The only thing that needs to be done is all the users needs to register the event through Event Share app.

User can navigate to Albums module by clicking on Albums on the navigation bar. When the albums page is opened, user can see all the albums named with the event name in a grid view. When user clicks on any album, that is a particular event, he can view all the images captured by the registered users of that event between event's start and end time. For example, if there is a one-hour event between 5pm to 6pm and 10 users are registered to it, let's say each of them captured around 10 pictures between 5pm to 6pm. Once the clock ticks 6pm, all the 10 registered users can view all the 100 images on their device in the albums section of our app.

Built.io:

We have integrated our application with Built.io, which provides mobile backend as a service (MBaaS). We are using Built.io only to upload and download our images, as it provides easier access using their straightforward API's without any memory limit. Built.io provides an sdk, which needs to be added to our app/libs folder. Compile statement to compile the sdk needs to be added to the app's gradle file. There are many android-permissions, ranging from Internet to Network-State; all this needs to be added to the manifest file.

Uploading Images:

To upload the images from user's device, we have written an Android Service, which runs in the background to upload all the images when the event ends. We are using AlarmManager to set an alarm to the end time of an event, which runs our image upload service. All the

event details like Event id, name, start time, end time can be easily fetched from sqlite and thus saves battery life by not running service forever in the background. Alarm is set to a particular event using the WakefulBroadcastReceiver. When the Android OS triggers the alarm, this receiver will start our image upload service, which uploads the images using Built.io APIs.

Below is the sample code that sets an alarm to our image upload service.

```
public class ServiceAlarmSetter extends WakefulBroadcastReceiver {
    private AlarmManager alarmMgr;
    private PendingIntent alarmIntent;
    @Override
    public void onReceive(Context context, Intent intent) {
        Intent service = new Intent(context, ImageUploadService.class);
        startWakefulService(context, service);
    }
    public void setAlarm(Context context, String eventId, int hours, int mins) {
        alarmMgr = (AlarmManager)context.getSystemService(Context.ALARM_SERVICE);
        Intent intent = new Intent(context, ServiceAlarmSetter.class);
        alarmIntent = PendingIntent.getBroadcast(context, 0, intent, 0);

        Calendar calendar = Calendar.getInstance();
        calendar.setTimeInMillis(System.currentTimeMillis());
        calendar.set(Calendar.HOUR_OF_DAY, hours);
        calendar.set(Calendar.MINUTE, mins);

        alarmMgr.set(AlarmManager.RTC_WAKEUP, calendar.getTimeInMillis(),
            alarmIntent);
    }
}
```

When a device is restarted due to any issue, all the set alarms will be lost. So to reset all these alarms, we are using a BroadcastReceiver, which receives BOOT_COMPLETED action from the Android OS.

Downloading Images:

As image downloading and displaying is a time taking process, what we are doing is fetching the entire image URI's when user navigates to albums page based on the user's registered event. When a user clicks on any album, first we are placing an empty image in the grid view based on the count of image URI's for that particular event. Then we are displaying all the images by replacing the empty image on the grid view. When a user clicks on any image in the grid view, we are displaying the same image on a different page in full screen mode using RecyclerView view to get better user experience.

Maps Tracking

Google Maps API is being used to represent different user locations on map. To obtain Google API we need to follow the below steps:

- Enable Google API in the Google Developer Console
- Generate API key for by providing package name and SSH key of the system.
- Obtain the API key and use it in the google_maps_api.xml

Initially we are getting the current user location through the location manager either by GPS or network. We fetch location from the provider whichever is more reliable. We implement a location listener that is triggered for each location change of the current location. We have implemented a handler, which sends the user location of a registered event in a specific interval of time. In parallel we are pulling the locations of the users who are registered to that event in a specific time interval. For each change in user locations we are updating the maps with different markers for different users. This way each user location is updated on the map for each change of the user's' location who are registered in the event.

As the users should be tracked for a particular event irrespective of whether they open the map module or not, we have implemented a service which checks for registration of users to an event and tracks their location.

User is requested location permissions if the location services are not enabled in the android mobile device.

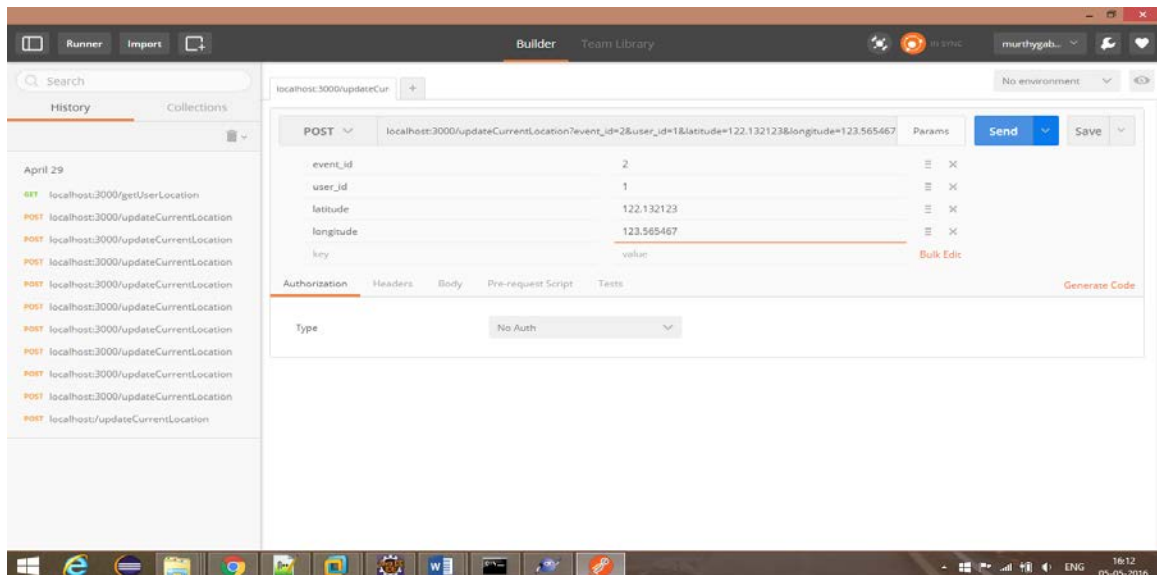
Testing

To Perform Android Mobile testing we have used ANDROIDJUNITRUNNER class, which is extended from JUNIT4 framework. Similar to Java JUNIT framework this also includes assertions and assumptions to validate the functionality. There are @BEFORE CLASS and @AFTER CLASS to perform operations that needs to be taken care before and after the test.

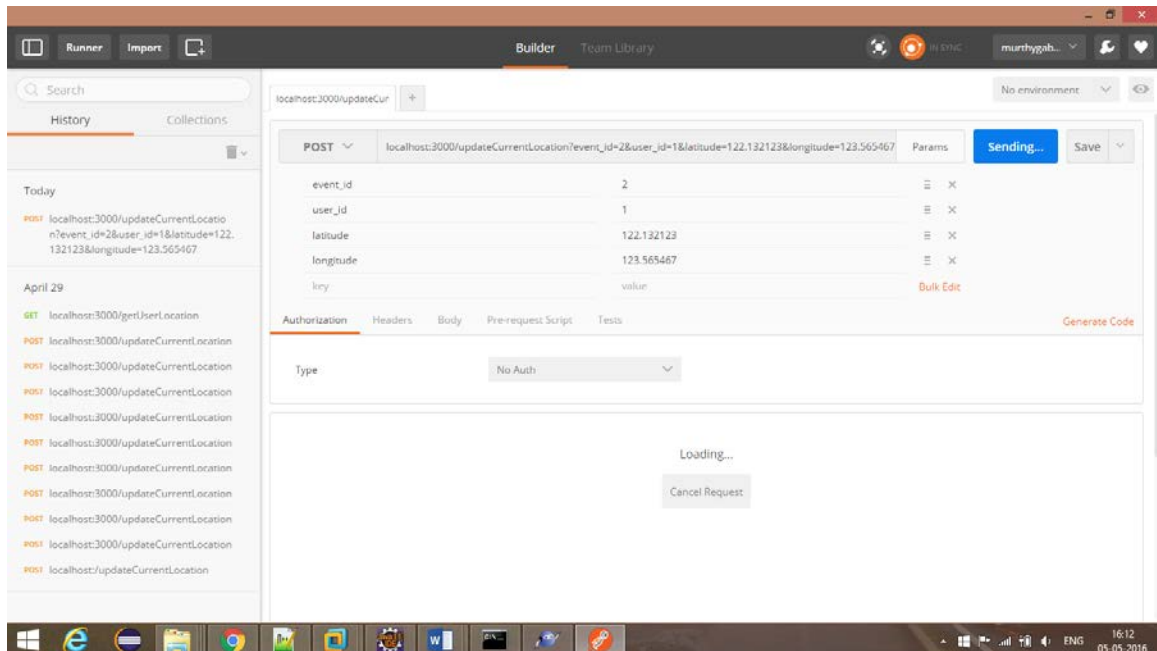
We have used POSTMAN client to test the APIs, which we have developed in server. Since NodeJS is the backend we performed API testing of the application.

We have considered updateLocationDetails API that updates the current latitude and longitude of the user and updates the data. Please find below the screenshots of the POSTMAN Client.

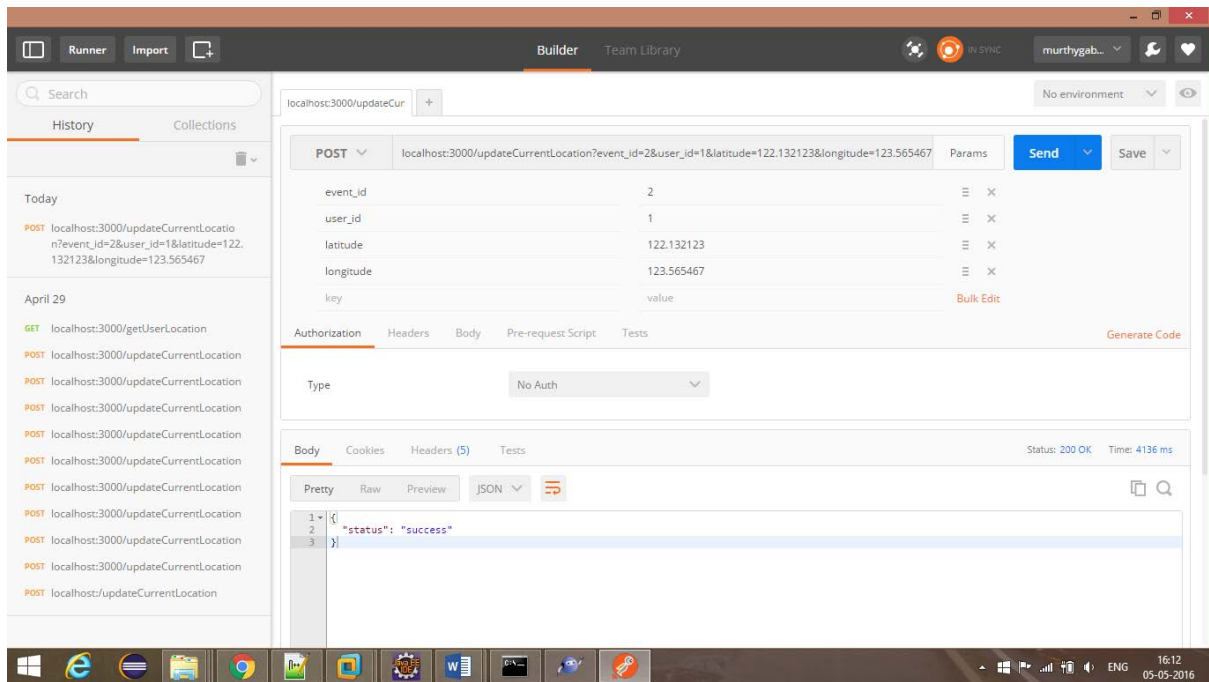
Added parameters and send request to the server.



Request Sent and waiting for response.

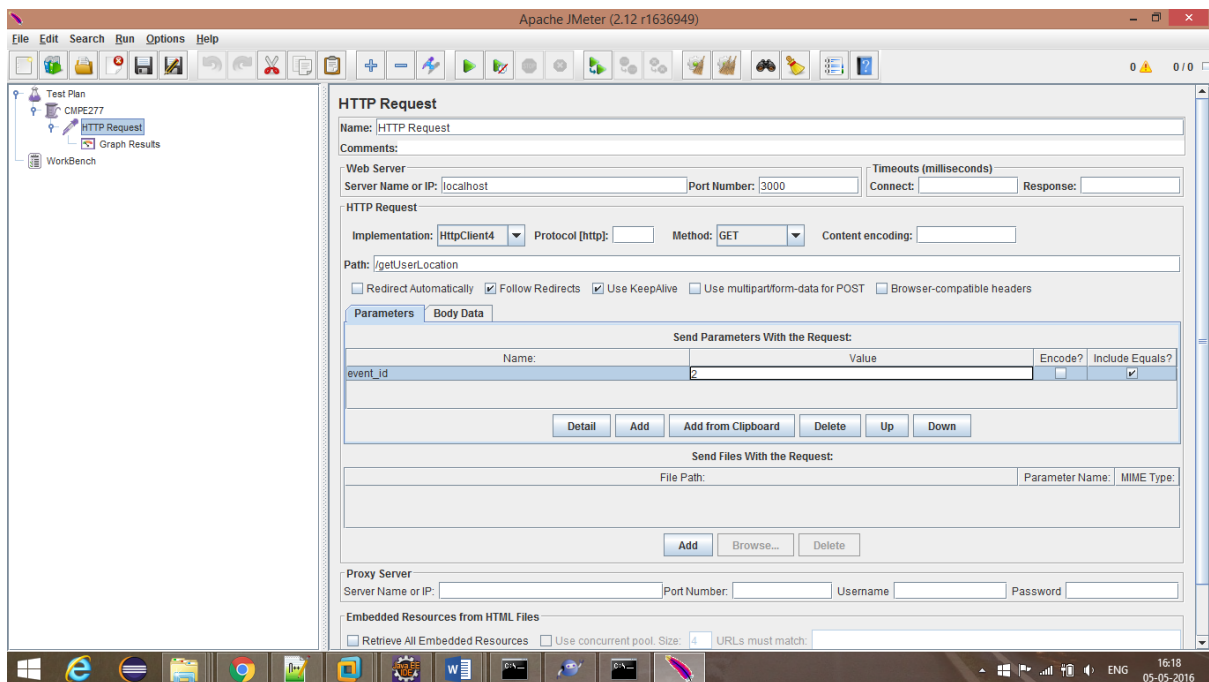


Response Received: Status Success

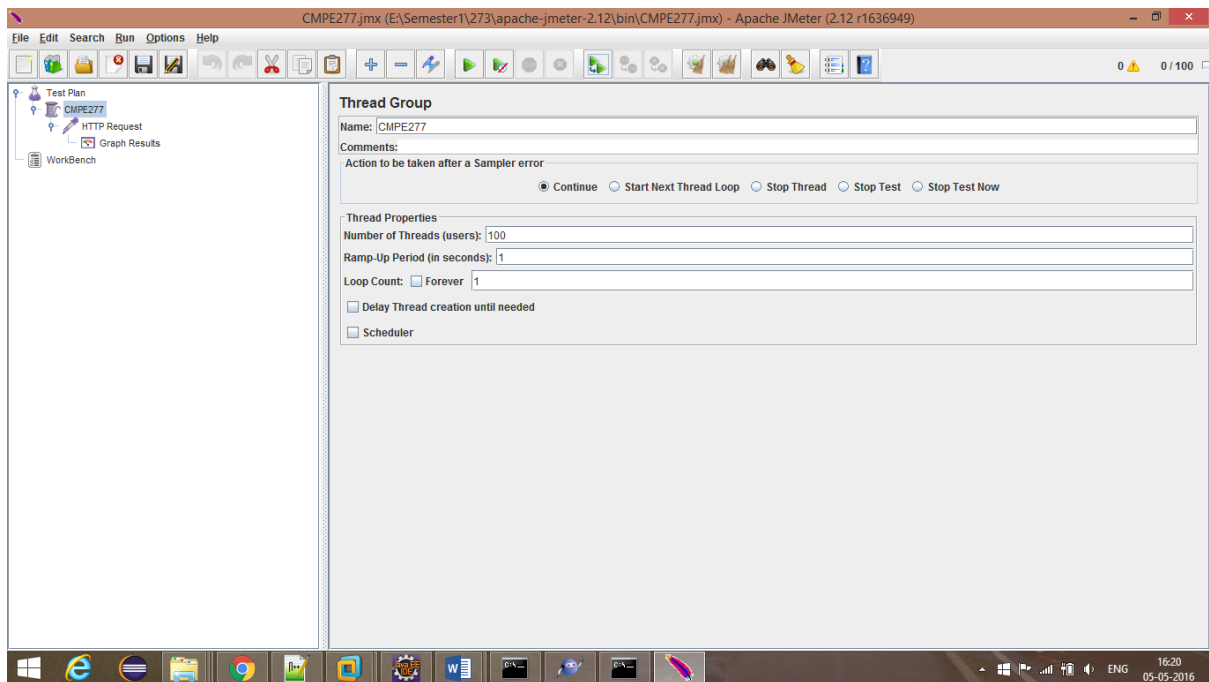


Load or Stress Testing:

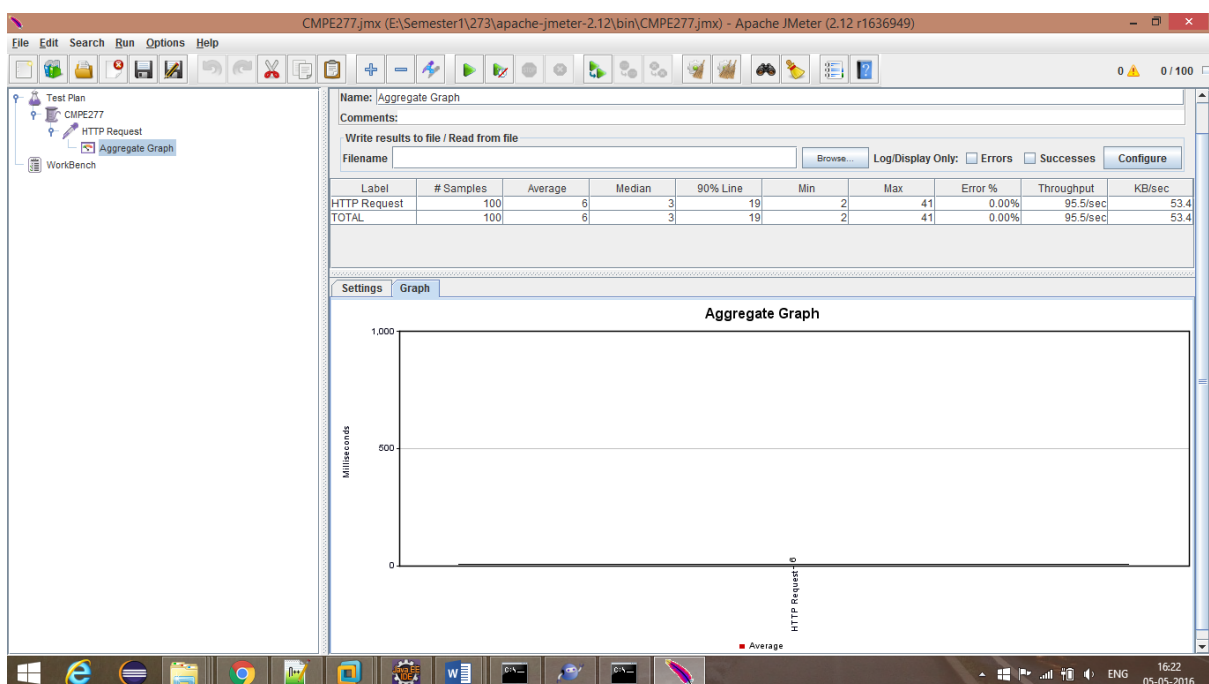
We have considered getLocationDetails API that gives the current location details of the user. This takes Event ID as input and provides the latitude and longitude of all the users present in the event. This is called whenever an event starts. Thus we have considered this API for load testing and performed load testing using JMeter and shared the details. Please find below the details of the load test results.



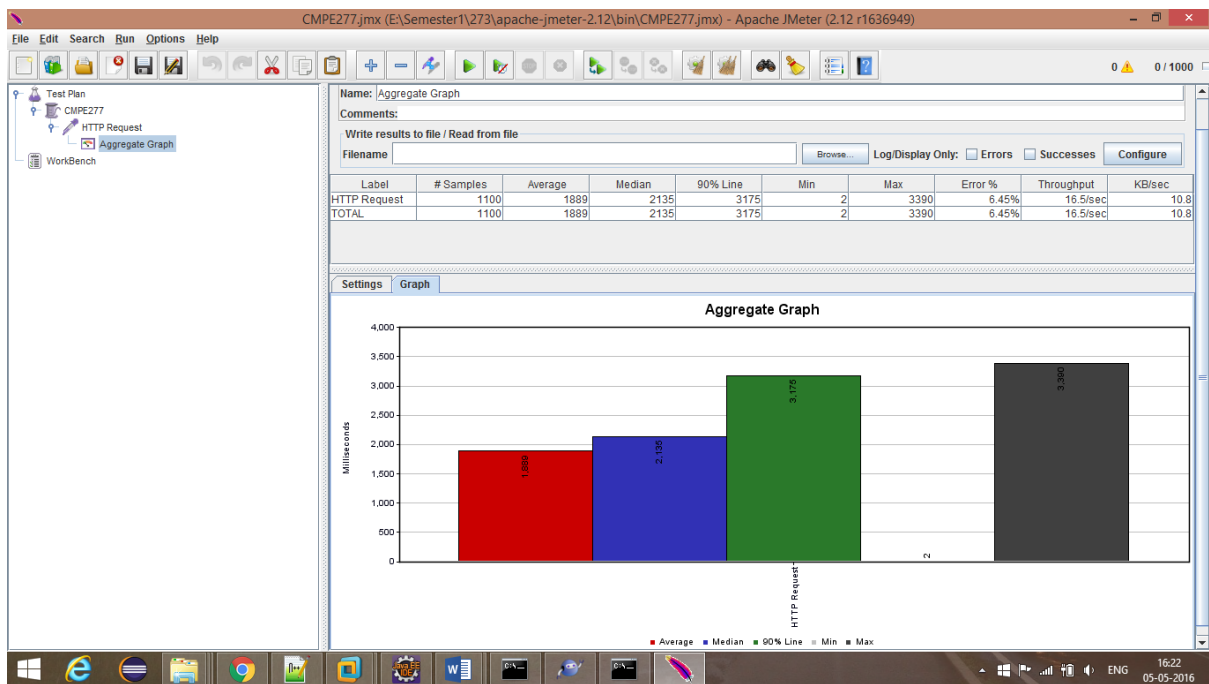
We have called the REST API from Jmeter and tested the API for 100, 1000, 5000 users and the server didn't crashes for 5000 requests. Thus our application is able to handle 5000 users concurrently. Calling the API and selecting the number of calls to be made



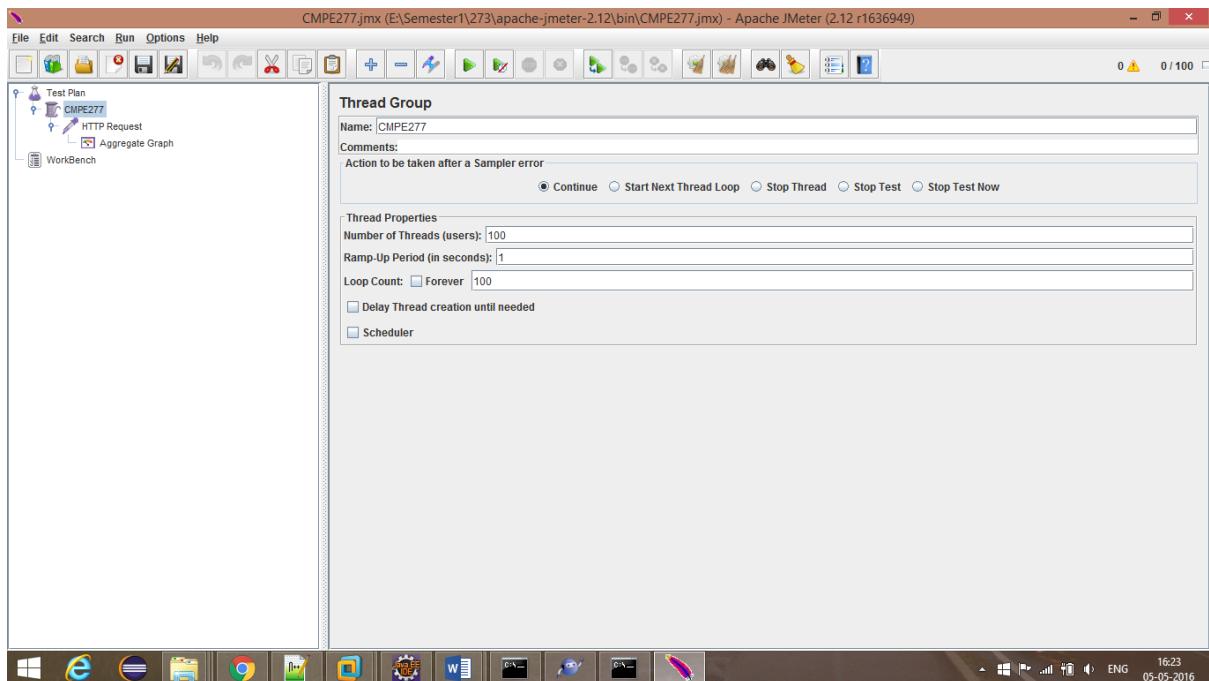
For 100 Users:



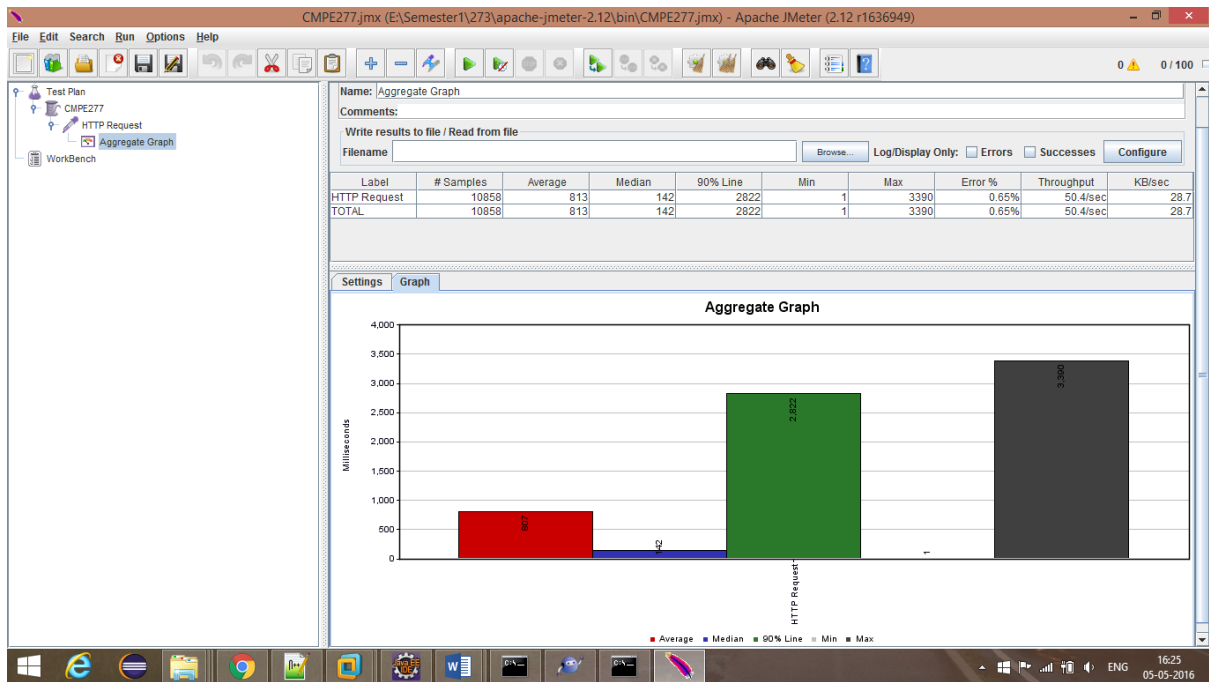
For 1000 Users:



For 100 Users making 100 calls simultaneously



The server didn't crashed even for 100 users making 100 calls simultaneously, please find below the screen shots of the same.



Automation Testing

We have performed automation testing on our virtual device using Selendroid, which means Selenium for Android. Selendroid is based on the Android instrumentation framework and currently only the testing of one specific app is supported. We have automated testing the basic functionality of our application, which helped us to make sure it works even after some last moment enhancements.

Below are the requirements for Selendroid.

- Latest Java and Android SDK
- Internet permissions in the manifest file
- Latest Selendroid standalone jar

To test an application using Selendroid, app's apk file should be generated and placed on a location where selendroid server will be started. This is because the test starts only when both server and app under test are signed using same certificate.

Below is the syntax to launch the test.

MSJCIT1032082:Selendroid prashanth.mudhelli\$ java -jar selendroid-standalone-0.17.0.jar -app eventshare.apk

This will start an http server at 4444 port and will retrieve the available virtual device.

All the applications and devices recognized by the selendroid can be seen by navigating to <http://localhost:4444/wd/hub/status> in a JSON format. Below is the JSON.

```
{
  status: 0,
  value: {
    "os": {
      "name": "Android"
    },
    "build": {
      "browserName": "selendroid",
      "version": "0.17.0"
    },
    "supportedApps": [{
      "appId": "eventshare",
      "mainActivity":
"com.cmpe277.mobileninjas.eventshare.MainActivity",
      "basePackage": "com.cmpe277.mobileninjas.eventshare"
    }],
    "supportedDevices": [{
      "screenSize": "320x480",
      "targetPlatform": "ANDROID17",
      "emulator": true,
      "avdName": "latest"
    }, {
      "screenSize": "320x480",
      "targetPlatform": "ANDROID10",
      "emulator": true,
      "avdName": "AVD_for_api10"
    }
  ]
}
```

Below is a sample test.

```
SelendroidCapabilities capa = new
SelendroidCapabilities("com.cmpe277.mobileninjas.eventshare");
WebDriver driver = new SelendroidDriver(capa);
WebElement inputField = driver.findElement(By.id("event_name"));
Assert.assertEquals("true", inputField.getAttribute("enabled"));
inputField.sendKeys("Varun Marriage");
Assert.assertEquals("Selendroid", inputField.getText());
driver.quit();
```

Design Patterns Used

We have implemented MVC based architecture and Cloud based architecture. We have considered MVC at the android level and Cloud architecture at Server Level.

Cloud Pattern

Cloud pattern encourages use of cloud services wherever necessary. It can be offloading resources to cloud like database, services, and applications etc to cloud. In this project, we used cloud based AWS EC2 hosted REST APIs for making http calls to the backend. Also used Build.io APIs to implement the photo sharing functionality. We have also used Mongo Lab (mlab) as backend NoSQL database in cloud.

MVC Pattern:

MVC pattern is also known as Mobile View Controller pattern. It is a popular pattern adopted across many languages and frameworks. The purpose of MVC is to separate data from the view. Model represents the business data and logic. View represents visual part of the data. View is something, which is seen by the application users. View components being the android XML layout files and controller being the Android Activity classes (JAVA classes). Model being the Android SQLite database and MongoDB (mlab). All these layout files are bound to various activities and these activities are tightly bound to the model (SQLite and Mongo DB).

The advantages of MVC can be categorized as follows:

- Separates problems
- Developer can focus on particular module.
- Parallel development

This supports reusability of business logic across different applications. It allows one to develop user interface without affecting business logic. Each module is individual and can be parallel developed.

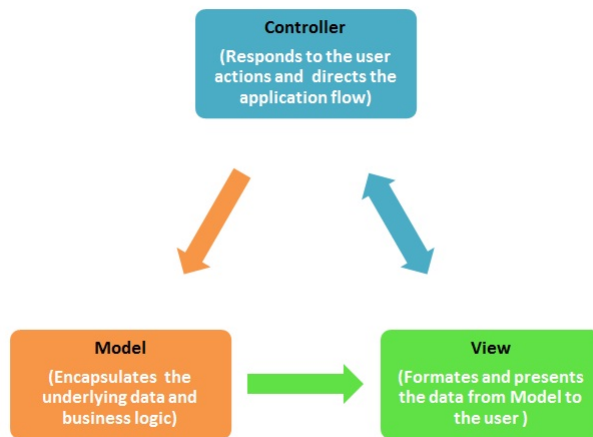


Figure 13 MVC Architecture

Individual Contribution

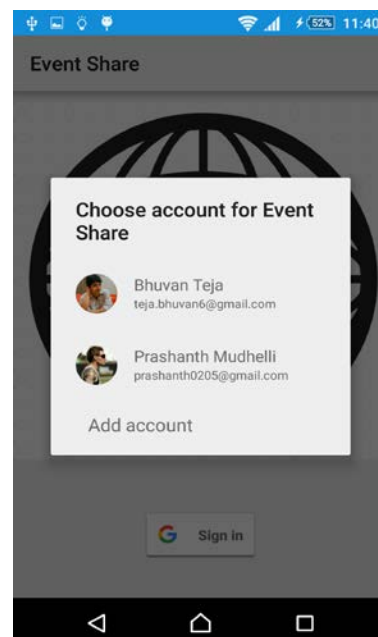
Sno	Name	Tasks Assigned
1	Anjaneya Murthy Gabbiti	Worked on Web Server Part, Created a POC for developing end-to-end flow for functionality. Created Fragment based Navigation bar and Created APIS for Maps related Activities. Worked on Albums. Deployed the Application on EC2 and MongoLab and worked on SQLite related classes and queries. Worked on documentation.
2	Bhuvan Teja Guddanti	Worked on Maps related part of the application. Developed Activities related to tracking functionality and Integrated the individual application components of individuals to a single module. Worked on Maps fragment and displayed various Map markers of various individuals on Map. Worked on Android Services and REST APIs.

3	Prashanth Mudhelli	Worked on Albums module, uploading the photos, downloading and displaying the photos as per event. Integrated the backend for images using Built.IO. Worked on Android Services and REST APIs. Also worked on creating a Gmail authentication login. Worked on application testing, API testing, automation testing, load testing and fixing bugs. Worked on Documentation.
4	Priyanka Ravikanti	Worked on Create Event related activities on Mobile and developed related APIS in web server. Worked on Documentation.
5	Swarna Viswanathan	Worked on accepting and rejecting the events module and designed related backend APIS in webserver as well. Worked on documentation.

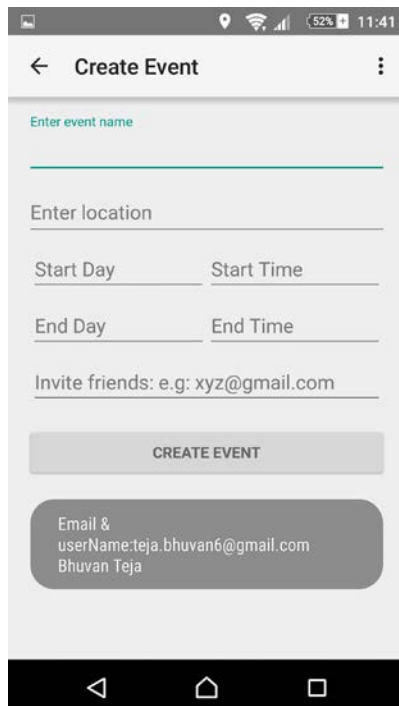
Screen shots

Device1:

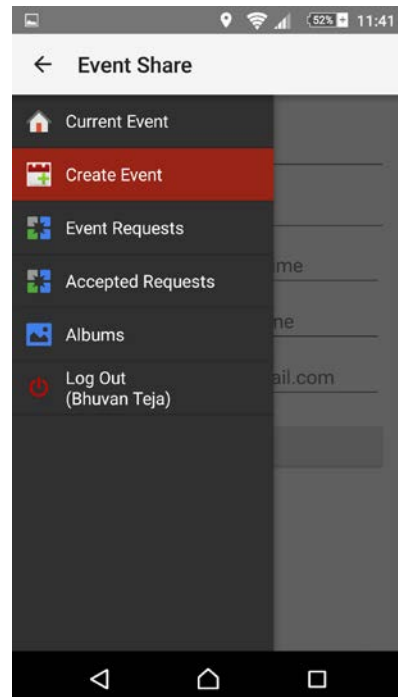
Application Login Activity:



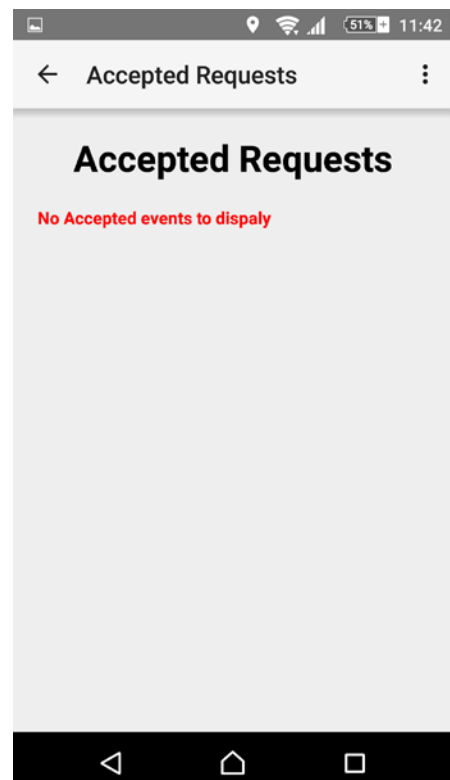
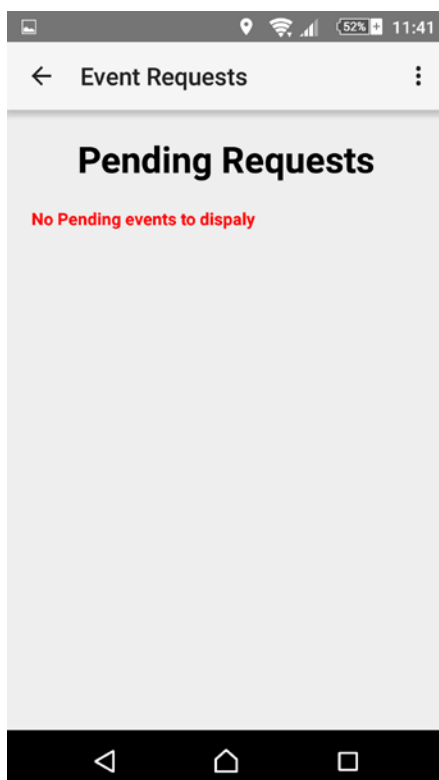
Home Page and Slide Menu:



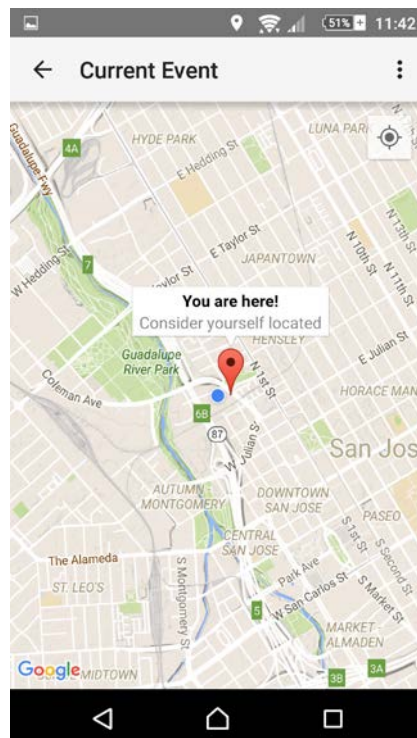
The screenshot shows the 'Create Event' screen. At the top, there's a back arrow and the title 'Create Event'. Below the title, there are input fields for 'Enter event name', 'Enter location', 'Start Day', 'Start Time', 'End Day', and 'End Time'. There's also a field for 'Invite friends: e.g: xyz@gmail.com'. A 'CREATE EVENT' button is at the bottom. A grey box at the bottom displays the user's email and username: 'Email & userName: teja.bhuvan6@gmail.com Bhuvan Teja'.



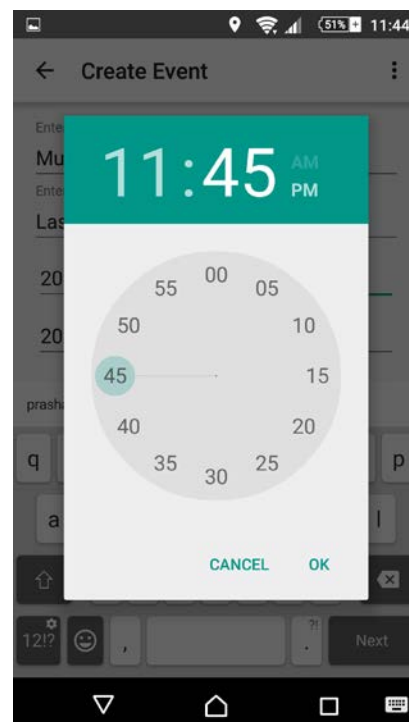
Default Event Requests:

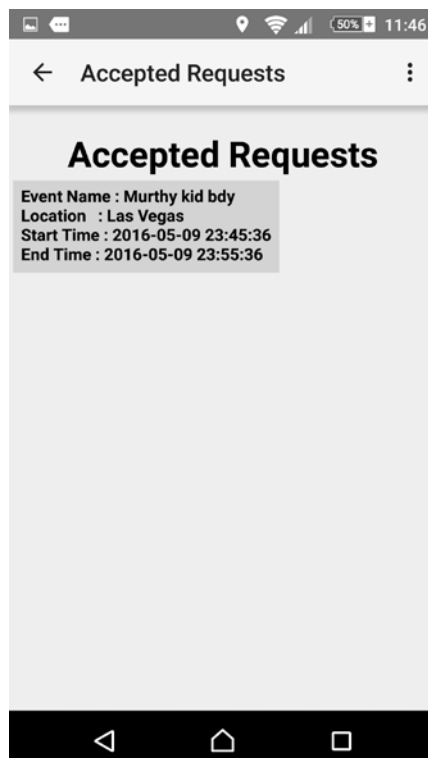
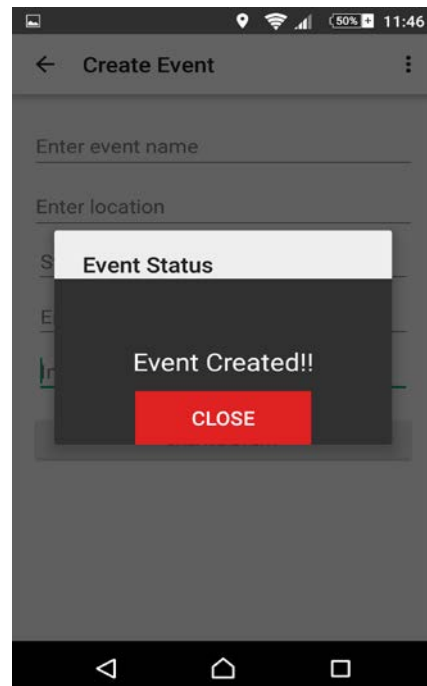
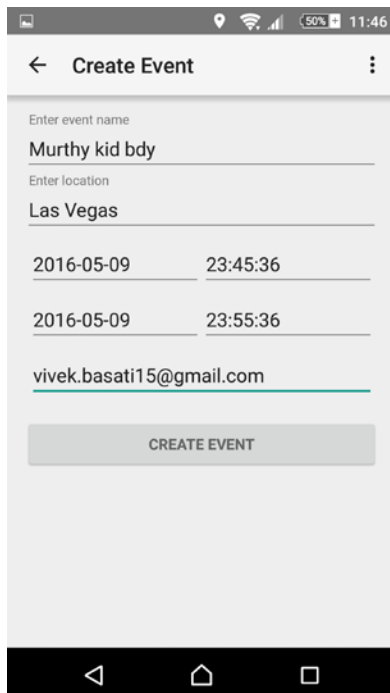


Current Location:



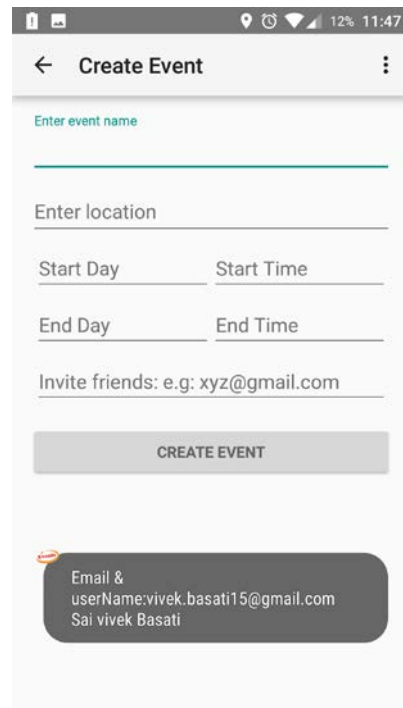
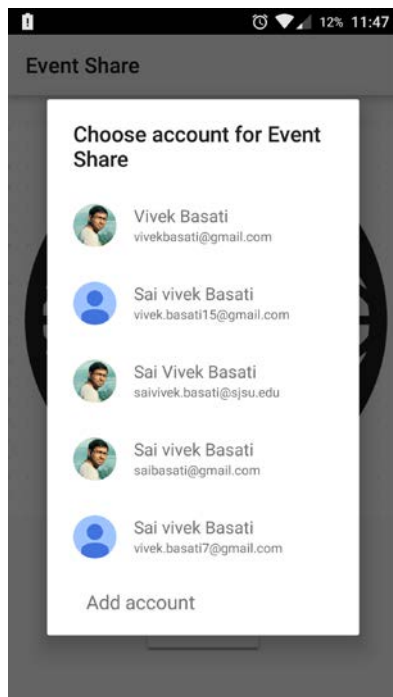
Event Creation:

A screenshot of a mobile application titled "Create Event". The form contains the following fields: "Enter event name", "Enter location", "Start Day", "Start Time", "End Day", "End Time", and "Invite friends: e.g. xyz@gmail.com". At the bottom of the form is a "CREATE EVENT" button. Below the button, a grey message box says "Please enter valid inputs". The status bar at the top shows a battery level of 51% and the time 11:42.

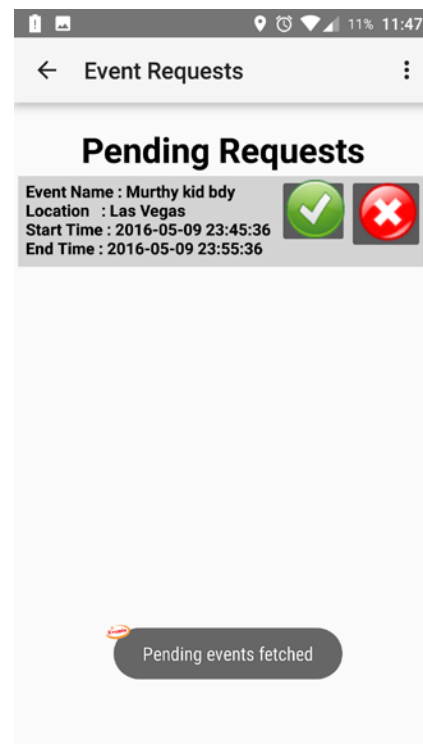
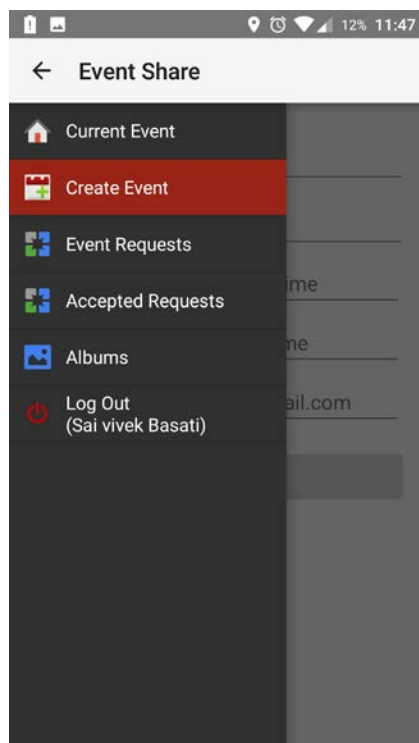


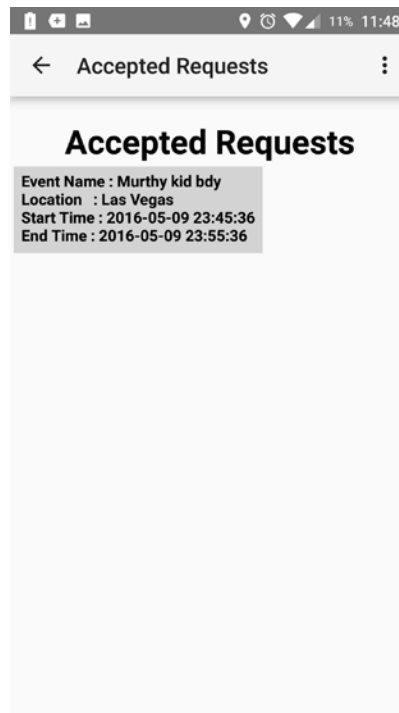
Device 2:

Login Activity:



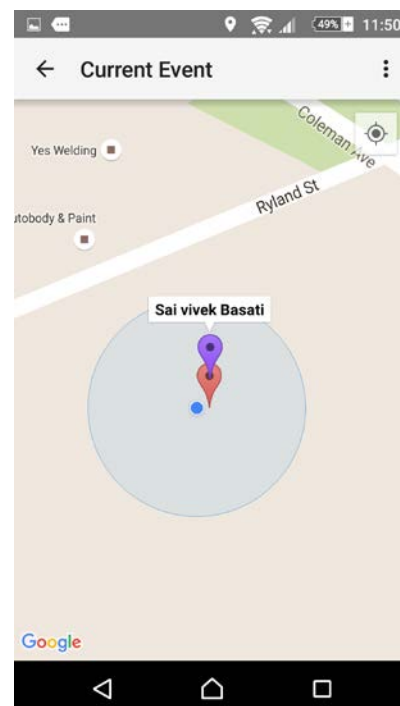
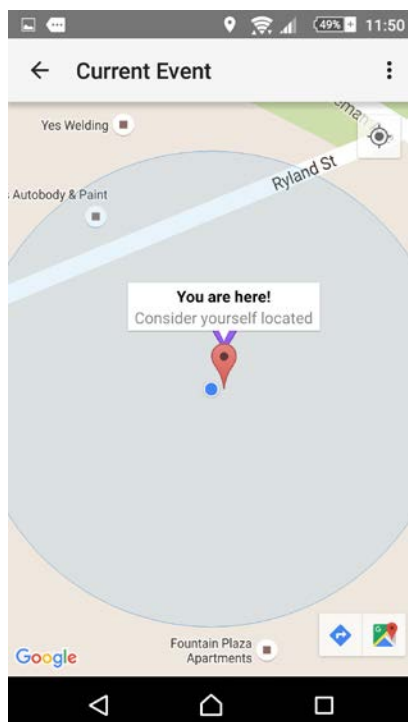
Accepting Event Request:



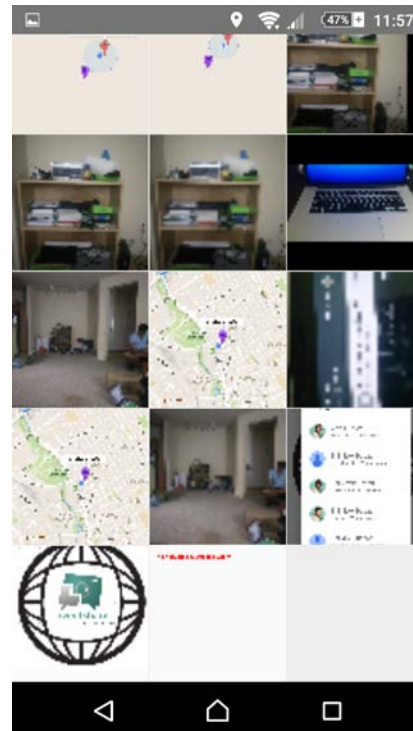
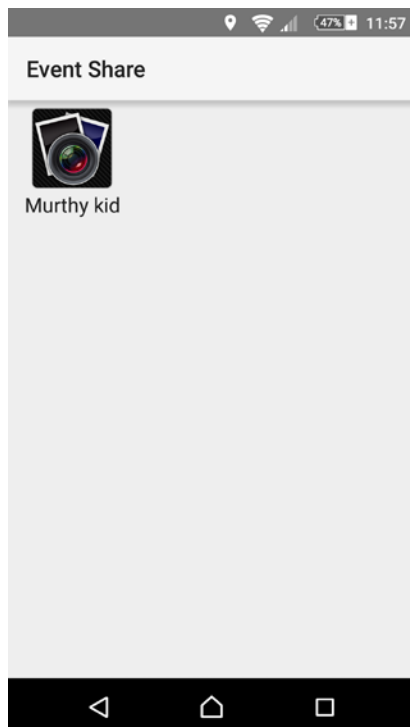


During Event – Device 1:

Tracking Users Location:

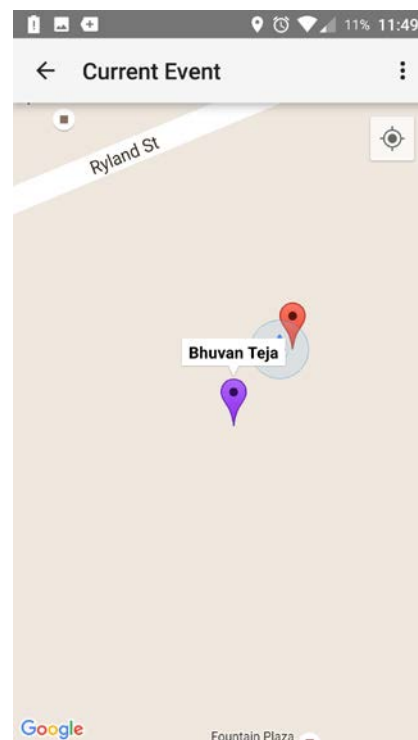
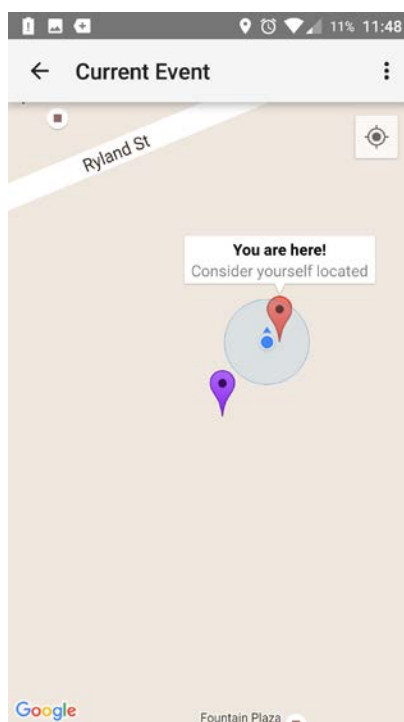


Fetching and Displaying Event Photos:



During Event – Device 2:

Tracking Users Location:



Fetching and Displaying Event Photos:

