Mobile Application Location Based Tracker

A dissertation submitted in partial fulfilment of

the requirements for the degree of

BACHELOR OF SCIENCE

in Computer Science

in

Queen's University of Belfast

By

Rory Magee

02/05/2018

SCHOOL OF ELECTRONICS, ELECTRICAL ENGINEERING and COMPUTER SCIENCE

CSC3002 – COMPUTER SCIENCE PROJECT

Dissertation Cover Sheet

A signed and completed cover sheet must accompany the submission of the Software Engineering dissertation submitted for assessment.

Work submitted without a cover sheet will **NOT** be marked.

Student Name: Rory Magee Student Number: 40154009

Project Title: Mobile Application Location Based Tracker

Supervisor: Dr Paul McMullan

Declaration of Academic Integrity

Before signing the declaration below please check that the submission:

1. Has a full bibliography attached laid out according to the guidelines specified in the Student Project Handbook
2. Contains full acknowledgement of all secondary sources used (paper-based and electronic)
3. Does not exceed the specified page limit
4. Is clearly presented and proof-read
5. Is submitted on, or before, the specified or agreed due date. Late submissions will only be accepted in exceptional circumstances or where a deferment has been granted in advance.

I declare that I have read both the University and the School of Electronics, Electrical Engineering and Computer Science guidelines on plagiarism - http://www.qub.ac.uk/schools/eeecs/Education/StudentStudyInformation/Plagiarism/ - and that the attached submission is my own original work. No part of it has been submitted for any other assignment and I have acknowledged in my notes and bibliography all written and electronic sources used.

Students Signature: Rory Magee *Date of submission: 02/05/2018*

Abstract

Having an effective method for using space within a building is very important but often very difficult, especially in larger organizations where it is harder to have everyone on the same page. Combatting this problem through the use of more primitive, paper-based solutions isn’t efficient or successful as a technological solution could be. In this project I aim to produce a system that will allow an organization to properly utilize space within their building by allowing them to manage locations and events in one central system viewable by everyone.

Table of Contents

Contents

1 Introduction and Problem Description

1.1 Introduction

A problem which is faced by many businesses and organisations is that they lack an effective way to manage space within their buildings. For example, the usage of meeting rooms, labs, classrooms etc. Having a centralised system to organise and manage the usage of space within a building and therefore allow for the space to be used as effectively as possible would be very beneficial. Additionally, especially for large organizations it can be very difficult to organize events, invite users and manage the attendance of the invited users in a seamless manner. Solutions such as attendance sheets or having a roll call take time and effort which could be avoided with an automated solution. This is the problem I aimed to solve in this project.

1.2 Problem Description

As mentioned in the introduction, a problem faced by many different types of organisations is that it is often difficult to manage space whether that be offices, lecture halls, labs etc. For example, if someone was trying to organise a meeting with their peers they may not have the necessary information available to them that would allow them to pick and time and place when they would have access to a certain room. Additionally, in the case of meetings involving a lot of people it would be difficult to ascertain whether everyone who was supposed to attend the meeting did so.

When trying to implement a software solution to this problem there are many hurdles which must be overcome. Firstly, there is the problem of how to track the users. There are various technologies available that would allow us to obtain the location of a use. For example

[1]RFID – Radio frequency identification (RFID) uses a small chip that could be attached to something belonging to the user. The chip continuously transmits a signal which would be read by RFID readers around the building and would inform the system of the location of the user. This has the benefit of being relatively unobtrusive as it would only track the user's location within the building where the system is implemented which is a positive in regard to

privacy. However, having to issue every student/employee with an RFID chip would create additional cost for the project.

GPS – Global positioning system (GPS) uses an array of satellites to pinpoint the location of a GPS receiver (which are found in most smart phones). GPS is typically usually accurate to around 4.9 metres under an open sky however can become less accurate due to interference from buildings. With this taken into consideration 4.9 metres is still extremely accurate and would work perfectly within the scope of our project. GPS however would allow the system to track a user’s movements outside of the organisation which may raise some privacy/ethical issues.

Bluetooth Beacons – Placing Bluetooth beacons in various locations around the building would be an easy and relatively inexpensive solution to the problem and it carries the additional advantage of being able to utilize the Bluetooth on the user’s phone rather than having to carry an additional signal emitting device like would be necessary with RFID. Using Bluetooth also shares the advantage of being very inobtrusive with RFID.

Indoor Atlas – Indoor Atlas is a newer tracking technology that uses magnetic field observations in order to know where a device is in a space. Using these magnetic signals, it is able to return consistent latitude, longitude and altitude values within a building that can then be used to determine where in a building the user is. This technology has the advantage of being completely free as opposed to the cost of buying Bluetooth/RFID beacons. Additionally, the system is only able to detect user locations in mapped areas so it would not pose any privacy concerns.

Another problem is how users will be able to add new locations within a building that can be used for events. Depending on the technology used to manage how user locations are tracked there are a few different ways in which the locations may need to be added in order for comparisons to be made between the user location and the location of events they are invited to. In the case of Bluetooth beacons or RFID it would be as simple as adding an additional beacon to in the location that needs to be added and registering it with the backend. However, when it comes to adding a location using GPS it becomes more complicated as requiring a user to enter lengthy latitude and longitude values is not very user friendly, so it would be important to find a more user-friendly alternative.

The system would also need to be capable of handling user profiles as this would be necessary to differentiate between users. For this we could use Google or Facebook authenticators as these provide a simple way for the user to login without having to create an entirely new account and would allow us to obtain information about the user such as their name. Alternatively, we could create a database to store usernames and passwords and simply have the users enter the information needed for the application. This implementation would also require us to use encryption in the interest of user privacy.

1.3 Solution Approach

For this project I intend to create the application on the cloud using Amazon web services as the backbone. This application will handle all the information on users, buildings, rooms, events including the creation of events. Front end applications such as a mobile app will be able to send requests to the cloud application to book rooms, create events and will also repeatedly update the application with it's location allowing for attendance of events to be checked.

This approach will allow for all the processing to take place remotely on a server which is beneficial as it allows for the system to be centralised. If all the processing took place on the mobile application, there could be potential conflicts in the event of multiple users trying to interact with the system at the same time.

For this project I intend to create a REST API in Node.js which will be hosted on an Amazon EC2 instance and be linked to a MongoDB database which will be used to store all necessary information on users, buildings, locations, events etc. The API will have a number of routes that will allow nodes interacting with it to do various things such as create and account, login, request event/location information, create events/locations and allow the user to update their location automatically so they can be checked into events.

The front end of the application will be built on Android and will implement the volley package to make [2]HTTP requests to the API. The app will run a background service that is constantly running that sends a request to the server that updates the user's location so that even when the app is not focused, the user's location is still being updated constantly.

1.4 Software and Hardware Environment

1.4.1 Backend

The API will be hosted on an Amazon EC2 instance using instance type T2.Micro which has 1 CPU and 1GB of memory. This instance type was chosen as it provided an adequate platform for the project to run on during development and testing however, if the project was being opened up to more users then the hardware would need to be upgraded to handle the additional requests.

The database for the project is a MongoDB instance. Like with the web server the database hardware is very basic having only 512MB of storage and having to share CPU time. This was done in an effort to reduce the costs of the project. As mentioned above, if the project was being opened up to more users then the hardware would need to be upgraded.

The server software will be programmed in Node.js using Express and will handle various different requests that allow for the frontend to make login requests and get information on events and locations as well as create accounts, events and locations.

1.4.2 Frontend

The frontend of the application will be developed using Java in Android Studio for use on Android mobile devices. The app will implement the Indoor Atlas API as the technology to track where the user's device is within the space as well as the Volley API to make HTTP requests to the server. The mobile application should provide functionality that allows the user to create an account or login if they already have one, display all events that the user is invited to, display only upcoming events that the user is invited to, create events and invite other users to those events and create new locations that can be used in event creation

1.5 Success Criteria

The project could be deemed a success if it is able to track user locations within a building and automatically update user's attendance for any events that they attend. Additionally, users should be able to create new locations and events in a seamless user-friendly manner from within the mobile application.

1.6 Acceptance Tests

User Story	Acceptance Test
User attends and event, they have been invited to.	The user’s attendance for the event is automatically updated.
User uses the add location feature to add a new location.	The location specified by the user is added to the database and will be useable in the creation of future events.
User uses the create event feature to add a new event	The event will be added to the database and will be viewable in the app by the invited users.
User creates a new account using the create account dialogue.	The new account is created provided there isn’t an account already registered with the same email address and can now be used to login to the system.
User wishes to see a history of all events they have been invited to.	The API should have a route that allows a user to request all events which has them listed as an invited user which will be displayed through an activity within the mobile application.
User moves about within the building.	Users movement is detected, and updated location is sent to the backend at regular intervals.

1.7 Development Plan

The first task will be to develop the REST API and create the endpoints that the mobile application will use to interact with the system. I will need to implement basic functions such as a login function that issues an authentication token so that I can verify requests being made to the API are from an authorized user. Next, I will create the MongoDB models that will be needed to store the information for the system. This will include a model for events, locations and users. Then I will create some basic routes that will allow interaction with the API such as creating accounts/events/location, deleting events/locations and patching the user’s location variables.

Having these basic functionalities implemented will allow me to begin developing the frontend application. First, I will implement the login/create account activities using Volley to make the appropriate POST request to the API and make sure the front end is able to retrieve the authentication token that will be needed to perform various actions within the application. Next, I will create a background service within which I will instantiate the Indoor Atlas API that will be used to retrieve the user’s location. Within this service I will have to write some additional code that posts the location to the backend at a timed interval. Then I will be able to begin working on implementing the activities that will allow the user to create new events and new locations. These activities will consist of some basic UI elements that will allow the user to specify information about the event/location such as name, date/time, specify invited users etc. which will then be parcelled into a JSON request and sent to the API via a POST request in Volley. I will also need to implement activities to display information pulled from the API via a GET request such as lists of events/locations etc.

1.8 Software Used

Program	Explanation
Visual Studio Code	Text editor with syntax highlighting, debugging and integrated Windows command line
Android Studio	The Android development IDE used to develop and debug the android application
Postman	REST client used to test the API endpoints
Git	Used for version control
Windows Command Line	Used for pushing updates to git repository and deploying need code to Amazon EC2 instance.

2 System Requirements Specification

The main functionality of the system lies with the REST API hosted on a remote web server to allow access at any time provided the user has an internet connection. A front facing Android application will interact with the API in order to create events/locations and update user's locations, so their event attendance can be managed.

2.1 Server Overview

The server for the system is running on Amazon web services. All interactions with the server require authorization in the form of a JSON web token. JSON web token is an open standard that can be used to securely transmit information between parties as a JSON object. Requests sent to the server with a valid token can be trusted as they are digitally signed. The token is issued to the user upon logging in with a valid username/password combination and expires after a set length of time so that if a token is compromised its usefulness is limited. When HTTP requests are made to the server the token must be present in the header of the request or else the server will issue a 401 response.

All interactions with the server are made using HTTP methods such as POST, GET, PATCH and DELETE. For example, sending a GET request to the "/users" directory of the server will return a list of all users registered with some basic information about each. When making POST requests to the server to do things like create events all the necessary information such as the name of the event, location, invited users, start/end dates are sent as a JSON object in the body of the request which is then parsed on the backend use the node module body-parser. Once the body of the request is parsed the event/location is saved to the database using the corresponding Mongo model.

2.2 Database Overview

For the database MongoDB was used as it provided adequate features for storing everything I needed as well as allowed for easy interaction with Node using the Mongoose module. The database is used to store information on events, users and locations and a model was created for each. The models are also linked together were necessary for example the event location field in the events model is linked to the locations model and the invited users field in the events model is linked to the user's model. All interaction with the database is done through the API so there is no direct communication between the end user and the database.

2.3 Mobile Application Overview

The mobile application provides the user interface through which the user will interact with the API and the database to do things like add events. It is also what will be used to send the users location to the backend so that attendance can be checked for events and allow the user to view events that they are invited to.

Once a user logs into the application the user's id and authentication token will be stored by the application for making HTTP requests. A background service will be launched that sends the users location to the "/users/userId" route every 15 seconds in the form of a patch request.

The mobile application will also periodically retrieve a list of events that the user has been invited to using a GET request and display these events in an activity using a RecyclerView. A basic form will be used within the application to allow the user to input the necessary information to create events which will then be sent to the server in a POST request using the authorization header obtained when the user logged in.

Locations in the backend are stored using the co-ordinates of the four corners of the room. Locations will be added through the app by having the user specify the four corners of the room. A corner is specified by placing the phone at the corner of the room and pressing a button. This process is repeated for all four corners of the room and then the user can give the location a name and send it to the database using a POST request.

2.4 Location Tracking

Location tracking will be done inside the application using the Indoor Atlas API to get accurate latitude, longitude and altitude values for the user. The Indoor Atlas API when implemented in favorable conditions can achieve accuracy of around 4-5 meters which is perfectly suitable for our use in this application. Due to the high accuracy of the API, implementing Indoor Atlas as the sole tracking technology for the system seemed like an advantageous choice as it allows for relative simplicity when compared with combining data from multiple sources to try and triangulate the location of the user. Not only would implementing more tracking technologies increase the complexity of the solution but they would increase the overall cost and cause limited scalability.

2.5 Functional Requirements

2.5.1 Server

- Issue valid authentication token to user when logging in
- Verify that all requests made to the server contain a valid authorization token
- Return JSON objects in response to requests made to the server
- Return a list of all events with basic information about each
- Return a list of all events that a specified user has been invited to with basic information about each
- Return a list of current and future events that a specified user has been invited to with basic information about each

- Return details of a specific event specified by an event ID
- Allow user to POST a new event
- Allow user to POST new location
- Allow authorized users to DELETE an event
- Allow authorized users to DELETE a location
- Allow authorized users to DELETE a user account
- Automatically update a user’s attendance
- Calculate whether specified co-ordinates are within the bounds of a location
- Return a list of all locations with basic information about each
- Allow a user to edit event information
- Return a list of all users with some basic information about each
- Return a specific user specified by a user ID with some basic information about the user
- Allow new users to create an account on the system

2.5.2 Mobile Application

- Allow users to login with their email address and password
- Make background POST request to the server containing the user’s current location periodically
- Display a list of current and future events that the logged in user has been invited to with some basic information about each event
- Display a list of all events that the user has been invited to with some basic information about each event
- Allow the user to select an event and see more detailed information about that event including a list of other users that have been invited and whether they have attended the event
- Allow the user to add a new location
- Allow the user to add a new event
- Routinely update event information so that the display data is accurate

2.6 Non-Functional Requirements

2.6.1 Server

- Server should be accessible by anyone as long as they have an internet connection

- The system should have measures in place to prevent downtime
- The system should hash and salt passwords for security
- All requests should return a response
- Responses should be issued with the correct status code
- The server should be easily scalable
- All processing should be done on the central server

2.6.2 Mobile Application

- Once logged in the user's ID and authentication token should be stored by the app
- Design must be consistent throughout the app
- Information must be displayed in a consistent easy to read manner
- All inputs should be validated before being sent to the server
- The app should display error messages in an easy to read unobtrusive manner (Toast notification for example)

2.7 Hardware and Software Requirements

The API that the entire system relies on to operate is hosted on a remote Amazon web server and will handle all requests coming from the mobile application to minimize the amount of processing that needs to be done by the application and also to have continuity of data across all user's devices. The mobile application will run on Android and will be used to make calls to the backend API and display the information retrieved from those calls. The app will also be used to send POST requests to create resources on the back-end as well as implement the Indoor Atlas API in a background service in order to send the users location to the server. Indoor Atlas has an API also available for IOS development so in the future it would be possible to port the application to Apple devices however this was not possible due to time constraints with the project.

2.8 Back-end Application Routes

All requests made to the back-end are done through the use of HTTP functions such as GET, POST, DELETE, PATCH etc. For any functions that make changes to data or create new resources the user needs to be authenticated (essentially anything other than a GET request). Authentication is carried out using the Authorization header in the HTTP request. Requests made without this header get a 401 response with an “Auth failed” message. The token issued when logging in expires after 1 hour so there is not explicit method for allowing the user to logout. This could be implemented by simply deleting the token saved in the mobile application.

2.9 Event Routes

Function	URL	Description
GET	/events/	Returns a list of all events saved in the database
GET	/events/:userId	Returns a list of all present and future events that specified user is invited to
GET	/events/:eventId	Return information about a specific event
GET	/events/all/:userId	Returns a list of all events that specified user is invited to
POST	/events/	Posts a new event to the database
PATCH	/events/:eventId	Allows for event details to be changed
DELETE	/events/:eventId	Deletes the record for a specific event
DELETE	/events/	Deletes all events

2.10 User Routes

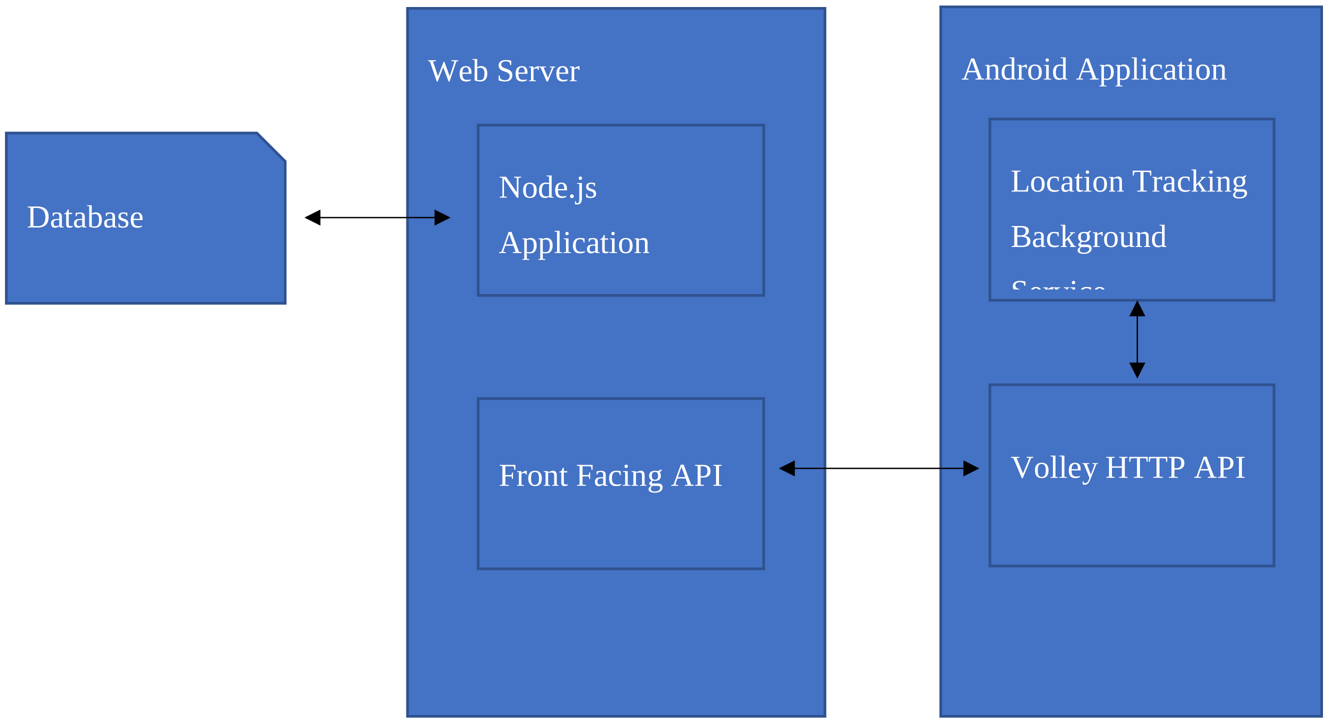
Function	URL	Description
GET	/users/	Returns a list of all users in the database
GET	/users/:userId	Returns information about a specific user
POST	/users/signup	Creates a new user account
POST	/users/login	Allows user to login and issues an authentication token
DELETE	/users/:userId	Deletes the record for a specific user
DELETE	/users/	Deletes all users
PATCH	/users/:userId	Route used to update users location

2.11 Location Routes

Function	URL	Description
GET	/locations/	Returns a list of all locations
GET	/locations/:locationId	Returns information about a specific location
POST	/locations/	Creates a new location
DELETE	/locations/:locationId	Deletes the record for a specific location
DELETE	/locations/	Deletes all locations
PATCH	/locations/:locationId	Allows for location details to be changed

3 Design

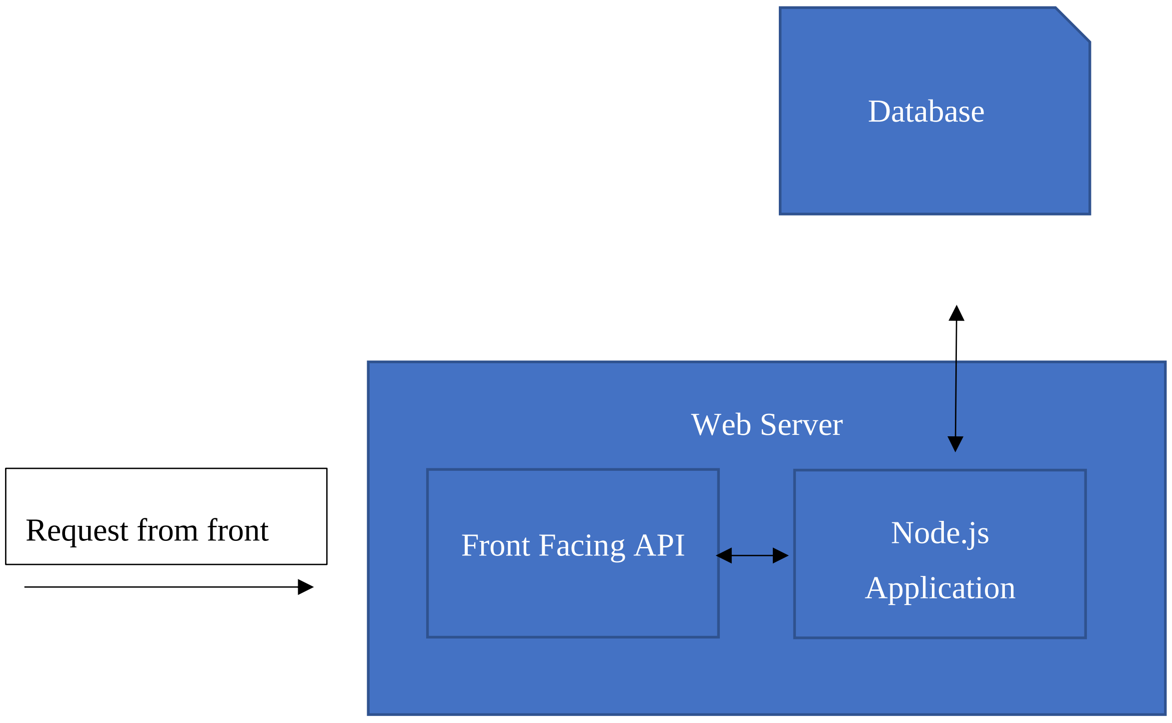
3.1 System Architecture



The system is comprised of two main elements. The web server that hosts the API and the front end mobile application. The database in this instance is hosted on a different server however this would probably not be the case if the system was deployed for public use.

3.2 Web Server

The web server runs on a linux box hosted on Amazon web services. The application is run in an elastic beanstalk instance allowing for easy configuration of the backend including adding additional databases, view application health, server monitoring, load balancing and security. All requests to the server are passed to the node application. All requests made to the server are passed to the node application which will then reach out and access the Mongo database. A JSON response will be generated and sent back to the user. In the instance that the node application encounters a problem and crashes the elastic beanstalk will repeatedly try and reboot it and provide logs allowing for the problem to be debugged. It is also possible to set up the instance to notify you via email when something goes wrong with the application.



When the server receives a request from the front end it will be made to one of the routes of the front facing API. The server will interpret this request and call the appropriate controller function in the Node.js application. Depending on the route accessed and the HTTP function used the node.js application may need to retrieve information from the Mongo database or it may need to save new data to the database. Once this is done the application will generate an appropriate JSON response and send it back to the user.

3.3 Mobile Application

The primary functions of the mobile application are to update the back-end with the users location, allow the user to make POST events to create new users, events and locations, allow the user to make GET requests to retrieve information such as upcoming events, lists of users and lists of locations and display all of this information in an easy to consume manner.

The app updates the user's location by intermittently pulling the device co-ordinates from the Indoor Atlas API and sending these values to the server in a PATCH request using Volley. This happens every 15 seconds. This logic was placed in a background server as it allows the code to run without interrupting other code running in the application and also so that even if the application is minimized the code will still run. When the user logs into the app, a GET request is made to the back-end to retrieve all upcoming events so that information can be displayed to the user using a CardView inside a RecyclerView. The app will also be used to allow the user to enter information needed to create a new event using a form interface. The form will contain a drop-down menu populated with locations pulled from the back-end with a GET request and a list of users with checkboxes that allows the user to invite selected people which is also populated via a GET request to the backend.

In order for a user to add a location they will need to specify four points that will form a perimeter of the area. Then they will place the phone at each one of these points and prompt the app to save that point by clicking a button on the screen. Once all four points have been added, the user can select a name for the location and then the new location will be sent to the back end in a POST request.

3.4 Database Design

For this project I decided on using MongoDB with the Mongoose as the package to interact with the database. I chose MongoDB because data is stored in a JSON like format which is how the system serves data in responses. Additionally, mongoose allows for very easy interaction with the database using JavaScript/Node. Mongoose also has the ability to create very detailed queries which allowed me to pull records from the database with a lot of specificity. This was particularly useful when checking user attendance as I needed a lot of very specific information such as events that are currently happening that the user is invited to. From this record I also needed to pull detailed information on the location so that it could be compared to the user's location. With MongoDB and Mongoose this was all doable in a single query. MongoDB also allows for linking between models which was useful for linking the location variable in the event model to an actual location object (locations have their own separate model). This was also the case with linking an event invited users to actual user objects.

For security purposes all passwords are hashed and salted before being stored in the database using 10 salt rounds which is reliably secure and doesn't cost too much processing power to hash.

3.5 Schema Design

For the schema we have a total of three models. Users, locations and events each with it's own unique properties.

Users

Name	Type	Properties
_id	Mongoose.schema.Types.ObjectId	Primary Key
Email	String	Required, Unique, Regular expression to validate email
firstName	String	Required
lastName	String	Required
Password	String	Required
Longitude	Number	
Latitude	Number	
Altitude	Number	

3.6 Locations

Name	Type	Properties
_id	Mongoose.schema.Types.ObjectId	Primary Key
Name	String	Required
Altitude	Number	Required
P1Lat	Number	Required
P1Long	Number	Required
P2Lat	Number	Required
P2Long	Number	Required
P3Lat	Number	Required
P3Long	Number	Required
P4Lat	Number	Required
P4Long	Number	Required

The location model needs to store a latitude and longitude co-ordinate for each of the four corners of the location so P1Lat would be the latitude point for the first corner and P1Long would be the longitude point for the first corner.

3.7 Events

Name	Type	Properties
_id	Mongoose.schema.Types.ObjectId	Primary Key
name	String	Required
location	ObjectId of location	
startDate	Date	Default: Current date

endDate	Date	Default: Current date
createdBy	ObjectId of user	Required
invitedUsers	ObjectId of user	
	Boolean	Default: false

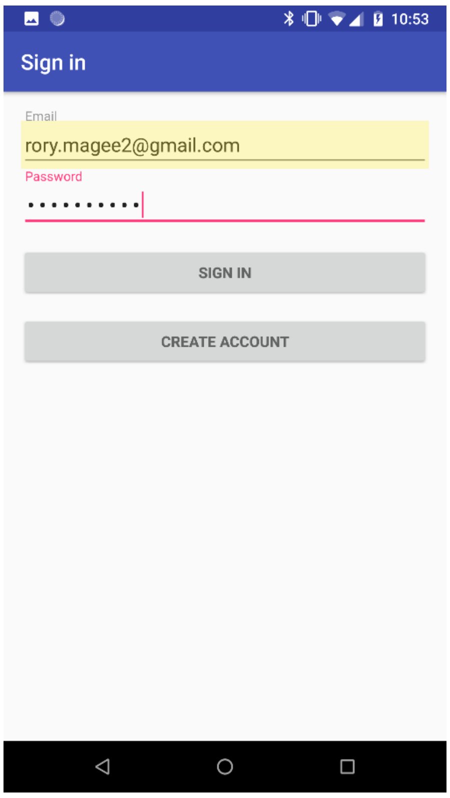
The invitedUsers variable is an array of JSON objects. Each object representing a user that has been invited and includes the users Id which is linked back to the users model and a Boolean value representing whether or not that user has attended the event which is set to false by default.

3.8 User Interface Design

The interface for the mobile application was built with the aim of being consistent and functional throughout the application as well as give the user feedback after any interaction for example Toast nonfictions.

Login Screen

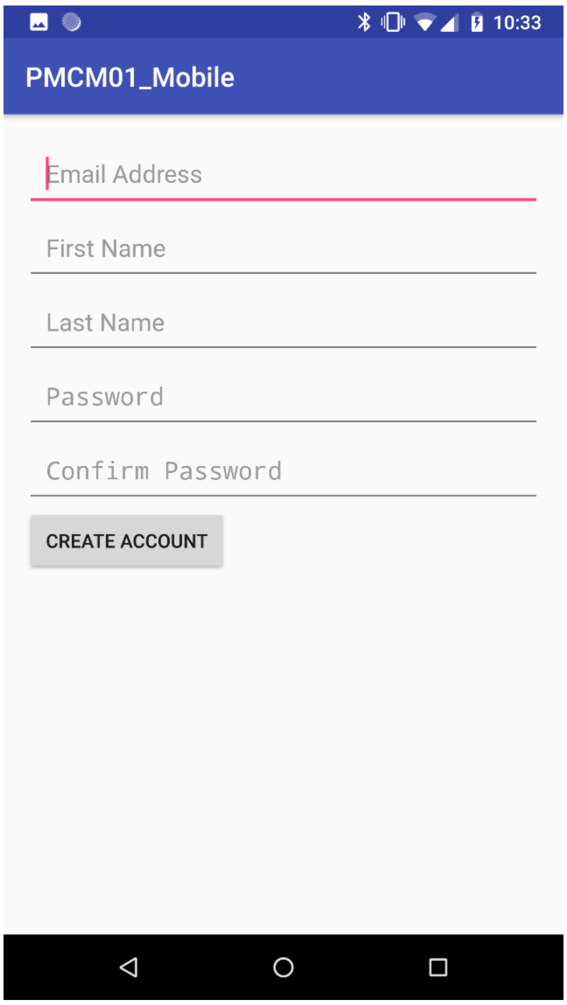
The login screen has a simple design consisting of two text boxes for the user to enter their email address and password combination and a button to sign in as well as a "Create Account" button that will take the user to a different activity which will allow them to create an account. When the user clicks the "Sign In" button the contents of the email address and password field are checked to make sure they have a value. If they do, then the contents of the text boxes are placed into a StringRequest object and sent to the "/users/login" in a POST request using Volley. If the email address and password combination are valid the server will give a response with HTTP status code 200 and a JSON response containing the users Id as well as an authentication token generated by bcrypt. If the email and password

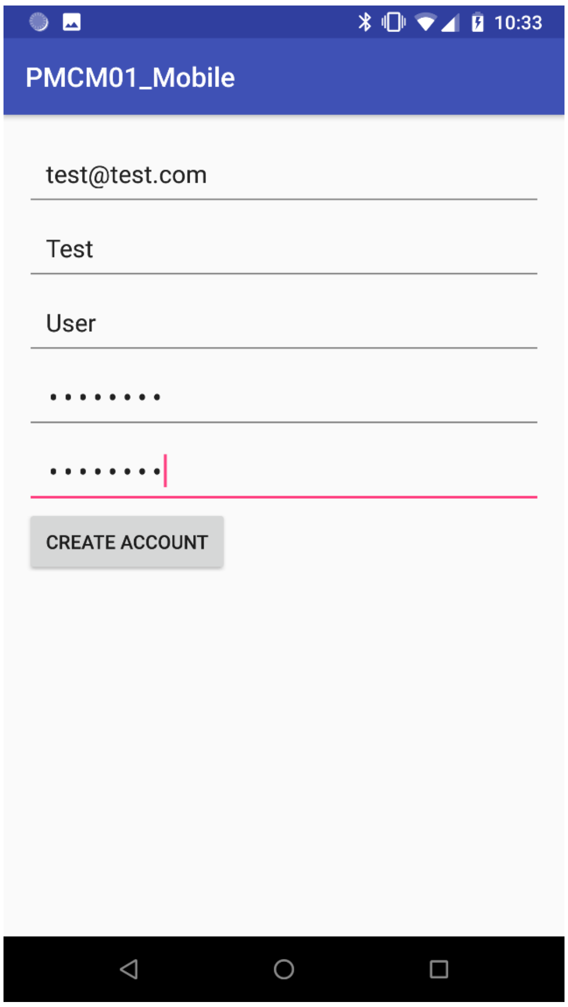


combination to not match any existing user, the server will issue a 401 response. In this case both text boxes will be cleared, and a Toast notification will appear informing the user of the failed login.

3.9 Create Account Screen

The create account screen is very similar to the login screen. It is just a simple form allowing the user to enter their information such as email address, first name, last name and password. When the user presses the "Create Account" button the app will check to make sure none of the fields have been left blank. If they are all filled in it will send a POST request to the server with all the information to create the users account. The account will not be created if there is already an account registered with the same email address and the server will issue a 409 response. If the account is successfully created the user will be sent back to the login screen where they can use their credentials to login.

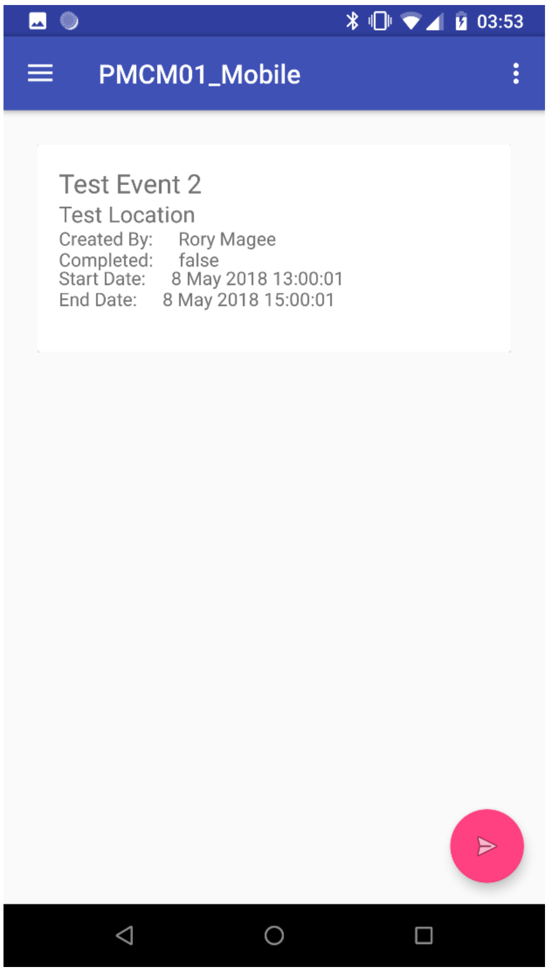




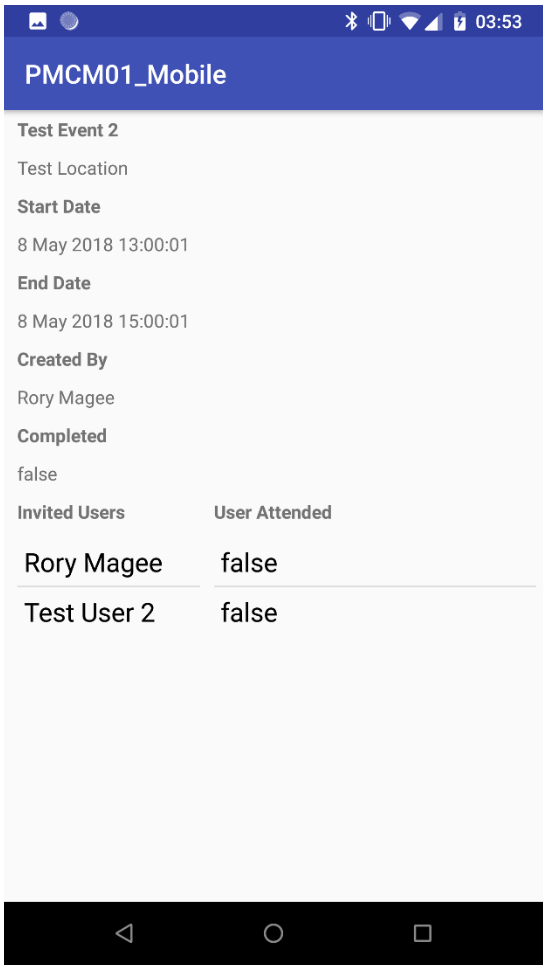
test@test.com
Test
User
••••••••
••••••••

3.10 Application Homepage

When the user logs in successfully as the main application homepage is loading a GET request is made to the server using the Id and token acquired during the login to retrieve a list of upcoming and current events that the logged in user has been invited to. This information is stored in an array of Events and displayed using a CardView inside a RecyclerView. The CardView for each event displays basic information about the event such as the name, location, who it was created by as well as the start and end times. To get more detailed information on the event the user can click on each individual card which will then allow them to see the list of users who have been invited as well as those who have attended and those who haven't.



Test Event 2
Test Location
Created By: Rory Magee
Completed: false
Start Date: 8 May 2018 13:00:01
End Date: 8 May 2018 15:00:01



Test Event 2

Test Location

Start Date

8 May 2018 13:00:01

End Date

8 May 2018 15:00:01

Created By

Rory Magee

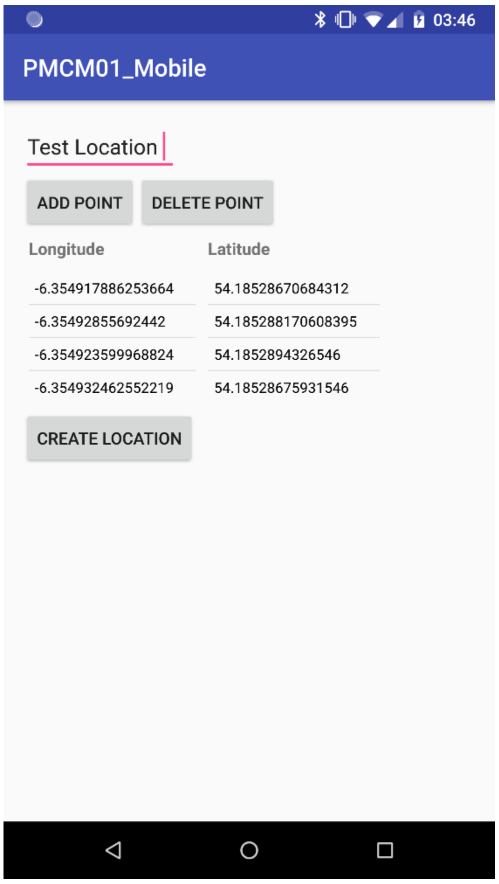
Completed

false

Invited Users	User Attended
Rory Magee	false
Test User 2	false

3.11 Add Location

The add location activity consists of a text box that allows the user to give the location a name, a button for adding a new point and button for deleting the most recently added point and a button for creating the location which only becomes active when the correct number of points have been added. When the user clicks the “Add Point” button, the app starts reading co-ordinates from the Indoor Atlas API repeatedly until it reads the same values twice in a row. This is because the Indoor Atlas API is very sensitive to movement and sometimes if the phone is moving the retrieved co-ordinates are not completely accurate so when the app reads the same co-ordinates twice in a row it knows the location is accurate. Each time a point is added a list view in the activity is updated to show the co-ordinates. Although latitude and longitude co-ordinates won’t be easily understood by the user, displaying them allows for some good feedback so the user knows that their point has been added correctly. When the user clicks the “Create Location” button the name of the location as well as the latitude and longitude points for the four corners are added to a String request and sent to the server in a POST request. If the location is successfully created on the back-end the server will issue a 201 response and the user will be redirected back to the application homepage.

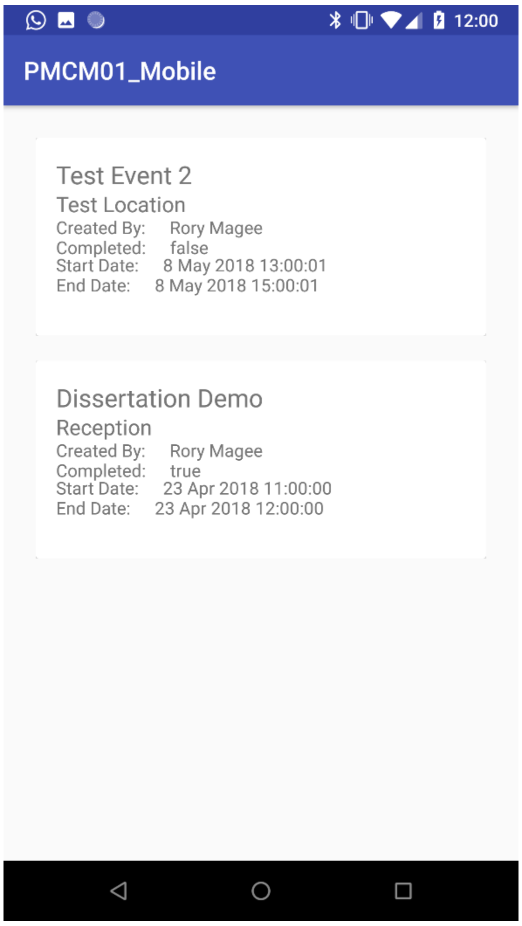


Test Location

Longitude	Latitude
-6.354917886253664	54.18528670684312
-6.35492855692442	54.185288170608395
-6.354923599968824	54.1852894326546
-6.354932462552219	54.18528675931546

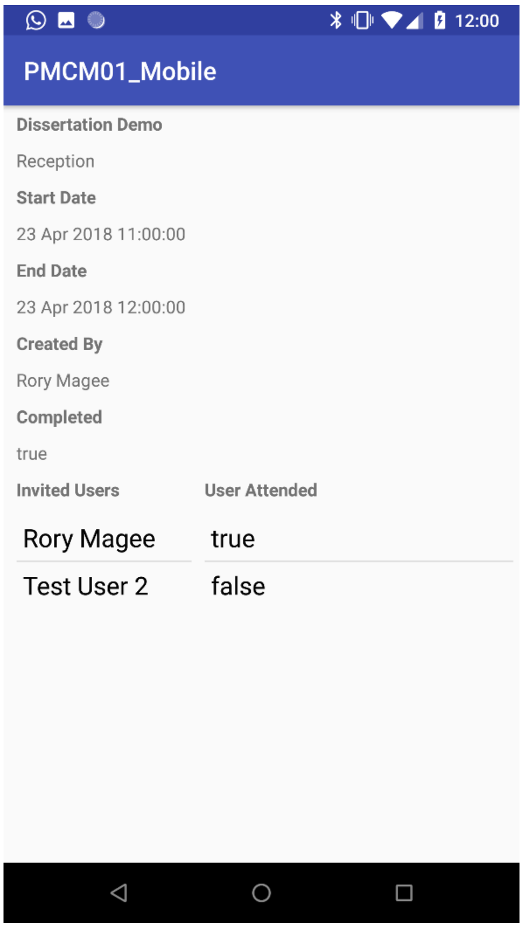
3.12 All Events

The all “events activity” is identical to the application homepage however instead of simply showing events that are upcoming it shows events that happened in the past as well. The user can also individually select each CardView to get more information about the events.



Test Event 2
Test Location
Created By: Rory Magee
Completed: false
Start Date: 8 May 2018 13:00:01
End Date: 8 May 2018 15:00:01

Dissertation Demo
Reception
Created By: Rory Magee
Completed: true
Start Date: 23 Apr 2018 11:00:00
End Date: 23 Apr 2018 12:00:00



Dissertation Demo

Reception

Start Date

23 Apr 2018 11:00:00

End Date

23 Apr 2018 12:00:00

Created By

Rory Magee

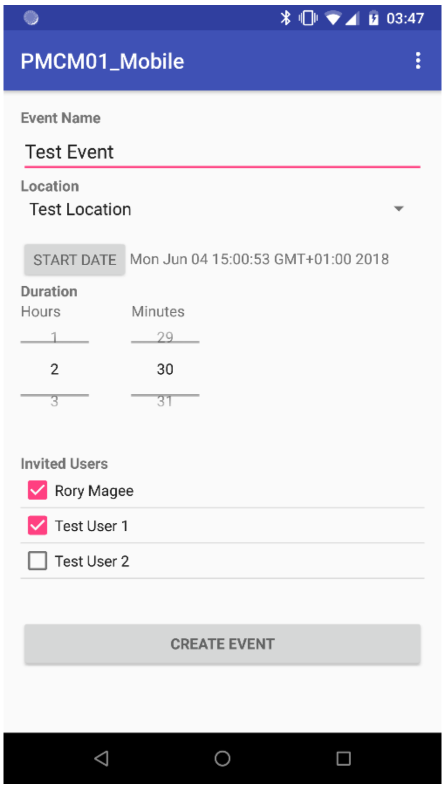
Completed

true

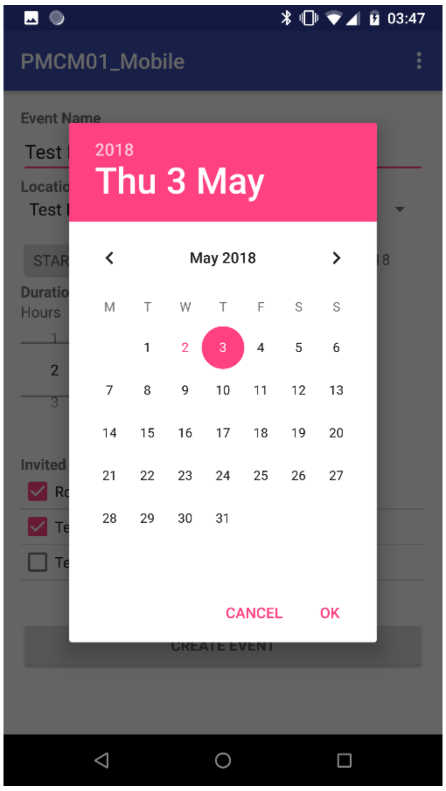
Invited Users	User Attended
Rory Magee	true
Test User 2	false

3.13 Create Event

The create event activity is accessed by pressing the floating action button on the bottom right of the main homepage. Again, it is just a form that allows the user to give the event a name, specify a location, select a start date, specify the duration of the event and select which users to invite from a list. The date and time for the event are selected using a DatePickerDiaglogue and a TimePickerDialogue. The lists containing locations and users are populated when the activity is created using two separate GET requests. Once all the relevant information has been input by the user and they click the "Create Event" button all of the form data will be parceled into a JSON object and sent in a POST request to the server using a JsonObjectRequest. If the event is successfully created a 201 response is issued by the server and the user will be redirected back to the main app homepage.



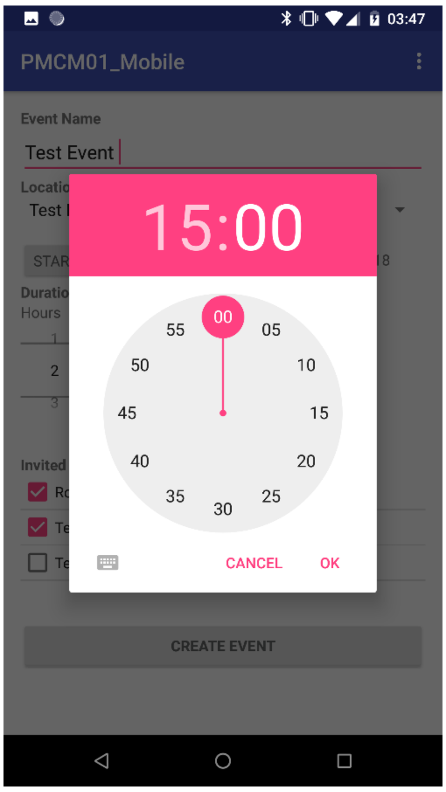
Event Name
Test Event
Location
Test Location
START DATE Mon Jun 04 15:00:53 GMT+01:00 2018
Duration
Hours 2
Minutes 30
Invited Users
[✓] Rory Magee
[✓] Test User 1
[ ] Test User 2
CREATE EVENT



2018
Thu 3 May

May 2018

M	T	W	T	F	S	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			



15:00

4 Implementation and Testing

4.1 Choice of Languages

Node.js was selected as the language for the backend of the system as it is a very fast capable language meaning that in production a lot of users would be able to use to system simultaneously. Additionally, Node has a large number of useful libraries available through npm allowing for easy implementation of things like password hashing and token authentication. Java was chosen as the development language for the front end mobile application as it is the language Android was developed in.

4.2 Server

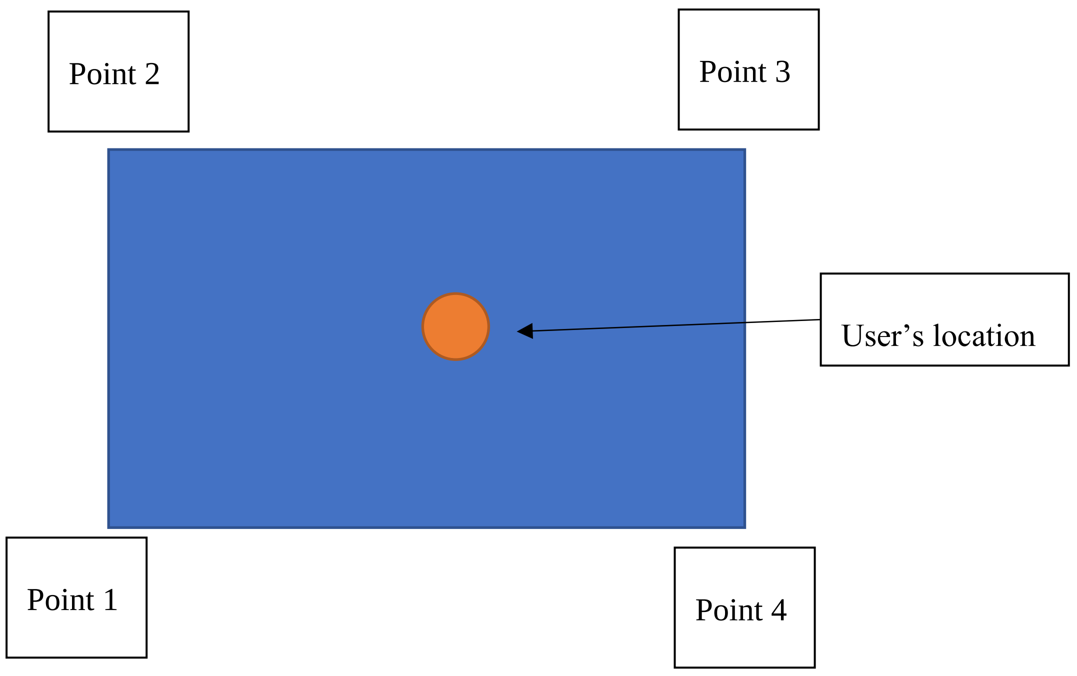
Node.js was selected as the development language for the server as it is a popular language in backend development. It also allows for relatively quick construction of a REST API. Another benefit to node is how easy it is to interact with the database using the Mongoose package. Complex queries can be constructed in a single operation.

4.3 Mobile Application

The mobile application was developed using Java as it is the main development language for Android applications. The Android application implements the Indoor Atlas API for location tracking which allows for accurate longitude and latitude values to be obtained even when the user is indoors. The mobile application also implements the Volley API in order to communicate with the backend with HTTP requests.

4.4 Important Functions/Algorithms

At the core of this project is the ability for the application to be able to detect if a user is in a certain location. When I originally decided to use the Indoor Atlas API I thought it would make this very simple as an algorithm to detect if a point is inside a square is fairly simplistic. One thing I did not account for however is that the building being mapped and used for testing the application may not be sitting perfectly on the axis of latitude and longitude. This means when trying to detect whether a point is inside a perimeter made of four points you have no point of reference for anything apart from the corners of the object, so you must work out if the point is within the perimeter only using references to these corners. I implemented a method called boundryCheck which essentially takes two co-ordinates (an x and a y value) and checks to see if those co-ordinates lie on the line between another set of co-ordinates. If the initial co-ordinate does lie on the line between the other two co-ordinates the function returns 0 and for every point off the line the point is the function will return a bigger or smaller number depending on how far the point strays from the line and which side of the line the point strays on. Every time a user's location is updated the system will look to see what events that are currently happening the user has been invited to and where those events are located. So for each event the system will then acquire the co-ordinates for each of the four corners in the room as well as the users reported co-ordinates and carry out the following process: It will run the boundryCheck function to compare how close point 3 is to the line connecting point 1 and point 2. Next it will use the boundryCheck function to check how close the user's location is to the line connecting point 1 and point 2. By comparing these two values we will be able to work out if the user's location is on the correct side of the line connecting point 1 and point 2. We repeat this same process for each of the following lines (i.e. line connecting 2-3, 3-4 and 4-1) and if the user's location is on the correct side of all of them then we know they are in the correct location and the users attendance for the event will be updated.



4.5 Server Hardware/Software

Name	Information	Use
Amazon Linux	Version 4.4.3	The operating system running on the server
EC2 Instance	1vCPU/0.5GiB memory	Hardware the server is running on
Software Used	Node.js v6.11.5	Programming language used
	MongoDB v3.4.14	Database used
NPM Modules used	Aws-sdk v2.182.0	Package used to interact with Amazon Web Services
	Bcrypt v1.0.3	Used for encryption
	Body-parser v1.18.2	Used to parse the body of requests
	Dotenv v5.0.0	Used to store sensitive data instead of hard coding them into the application (e.g. DB password)
	Express v4.15.5	Web framework
	Jsonwebtoken v8.2.0	Used to generate/check authentication tokens
	Mongoose v5.0.7	Used to interact with the database
	Nodemon v1.17.1	Used to restart the server immediately when changes

		are made (primarily used in development)

4.6 Mobile Application Hardware/Software

Name	Information	Use
Android SDK	v26	Software development kit used for the application
Android Build Tools	v26.0.1	The version of the Android build tools used
Android Permissions	ACCESS_FINE_LOCATION	Used to access the devices location (specific)
	INTERNET	Used when making HTTP requests with volley
Android Features	Accelerometer	Used by Indoor Atlas for detecting location
	Compass	Used by Indoor Atlas for detecting location
	Gyroscope	Used by Indoor Atlas for detecting location

5 System Evaluation

5.1 Functionality

Functionality is one of the areas I think the system did very well in. The system is reliable able to track a user's location within the building and automatically update the user's attendance when they are in a given location at the time of an event they have been invited to. This is done in an unobtrusive manner. The system is centralised on a server allowing for access to anyone as long as they have an internet connection and also allows for processing and storage to take place in a centralised location which was one of the things outlined in the requirements. Additionally, the system allows for the user to easily add new locations to the backend. This was one of the more challenging aspects to get right from a design standpoint however I think the implemented solution of having the user map out the four corners of the room using their phone is reliable and still relatively user friendly. Adding events through the application was also quite tricky from a design standpoint as there is a lot of information needed to be input by the user and again I think the proposed solution does the job quite well.

During the testing phase, I discovered that when the device is moving the co-ordinates being pulled from the Indoor Atlas API are not completely accurate. This means that it is possible that when the location service running in the application sends the users location to the backend there is the possibility that it is sending inaccurate co-ordinates. This is problematic as it could mean that users don't get checked in for events that they have attended. However, this is unlikely as if a user spends any meaningful amount of time at an event they should be checked in once the API stabilizes. A more realistic problem with this inaccuracy is that it could be possible for a user to be checked in to an event that they didn't actually attend. This could happen if the user is perhaps nearby and the small inaccuracy places them in the location of an event long enough to be detected. The solution to this would be taking a similar approach to how locations are added and only allow for a user's location to be updated once the app is certain that the co-ordinates it's retrieving are accurate (i.e. detecting the same co-ordinates twice in a row).

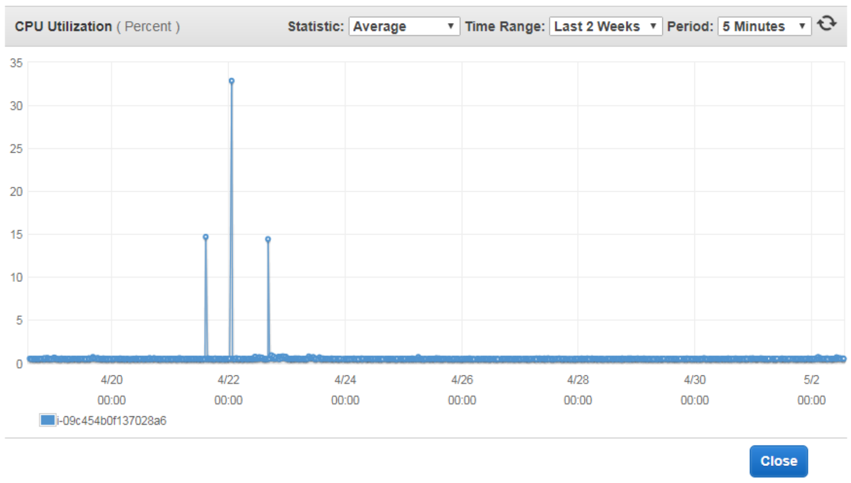
5.2 System Reliability and Speed

With the main system being centralized on a server the reliability is quite good. Amazon web services provides a reliable backbone to the system. In it's current configuration the application would only be suitable for a small number of users as it is running on very low-end hardware.

This was fine for the testing and development of the application but if it was being pushed into production the hardware would need to be improved for both the Amazon EC2 instance and the Mongo database. Another upside to using AWS to host the backend is the comprehensive logging process that occurs. This means that if there is a problem on the backend it can easily be identified and rectified. The asynchronous nature of Node.js also lends itself to being very fast as it is able to carry out a lot of operation simultaneously even though it uses just a single thread.

5.3 Performance

As mentioned above the API is running on very barebones hardware and therefore performance is not as good as it could be. The Mongo database is in a similar situation. Amazon web services however does provide options to migrate to more powerful hardware. These more powerful systems do however cost more money. Below is a screenshot of the CPU utilization of the web server. As you can see there are times when the utilization spikes by a large amount. Although this seems like an infrequent occurrence it is important to keep in mind that this is the system with only one user testing it so adding more would undoubtedly cripple performance.



5.4 Installation and Operation

Due to the entire backend of the system (aside from the database) being contained within one application it’s easy to assume that the system could be installed on any web server as long as it has Node.js installed. The system files would simply need uploaded to the server and a call made to the npm start script. However, inside the mobile application all requests are made to the URL provided in this case by Amazon’s web server which is hard coded into the application so if the server address changed someone would need to go into the mobile app and manually change all the HTTP requests to point at the correct URL.

5.5 Cost

During the testing and development, this system incurred absolutely zero cost. One of the main reasons for this is that the backend and the database are using free tiers rather than paid tiers meaning that they are running on less powerful hardware. If this system was pushed into production, then it would be impossible to avoid the cost of upgrading to a more powerful server setup. However, the Indoor Atlas tracking technology used in the system is completely free for non-commercial use which is one of the main benefits outlined in the beginning of using this technology over something like Bluetooth beacons that would cost money to implement and then have additional costs whenever new areas need implemented into the system. For an organization like a university that already has a lot of server real-estate it's likely that some space could be found to host this application meaning that the costs incurred would be minimized greatly.

6 Conclusion

This project was intended to create a system that tackled the problem of organizations not having an effective method for organization space within their buildings. The system produced needed to be able to allow users to specify locations within a building, create events in those locations with invited users and be able to automatically detect when a user attended or didn't attend an event and update the system accordingly. All this functionality is achieved in the system that was produced as it provided a centralized system that allows users to interact with it through the use of an Android mobile application. Users are able to define locations they wish to add in an accurate way, create events using these locations, invite other users on the system to the event and automatically have attendance monitored by the application.

The solution is not perfect and could be made a lot better in time however it does provide the main functionality I think is required to work effectively. Given more time I think it would be useful to have the mobile application ported onto IOS as it is a very popular platform having between 15-20% market share of smartphone operating systems in 2016[2]. In it's current form only users with Android devices would be able to make use of the system. Having more time, I would also have liked to have been able to implement more functionality on the front end such as altering the details of an event after the event has been created. This is currently possible on the backend however, there is no way to do it through the mobile application.

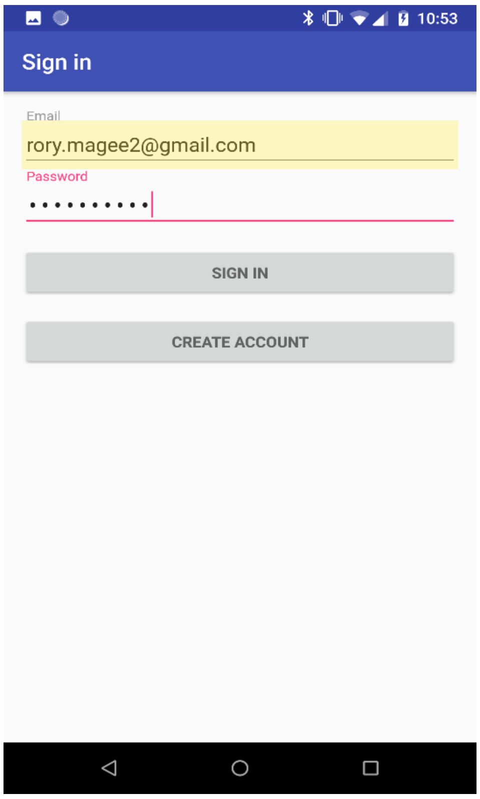
In conclusion the system developed meets the requirements set at the beginning of the project which is to give an organization an effective method for organizing space within a building and automatically allowing users to have their attendance checked without the need for primitive forms of administration such as sign in sheets. The system produced does this efficiently in an easy to use manner.

7 References

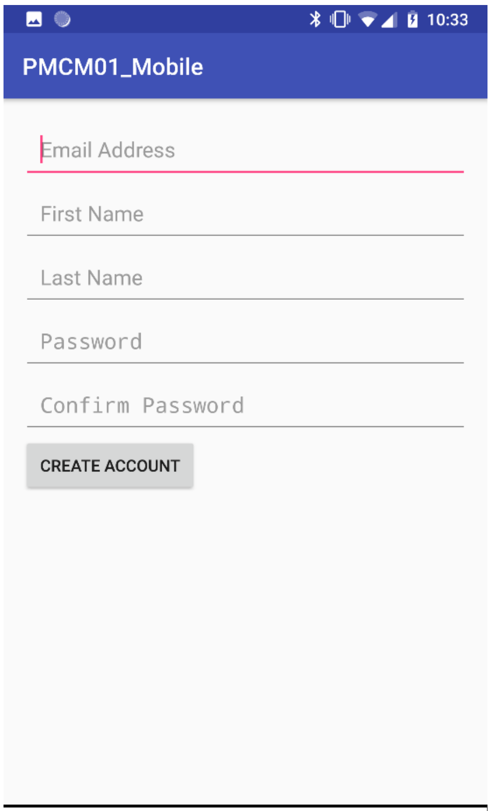
[1] EPC-RFID. 2018. *What is RFID? - EPC-RFIDEPC-RFID*. [ONLINE] Available at: https://www.epc-rfid.info/rfid. [Accessed January 2018].

[2]IDC: The premier global market intelligence company. 2018. *IDC: Smartphone OS Market Share*. [ONLINE] Available at: https://www.idc.com/promo/smartphone-market-share/os. [Accessed May 2018].

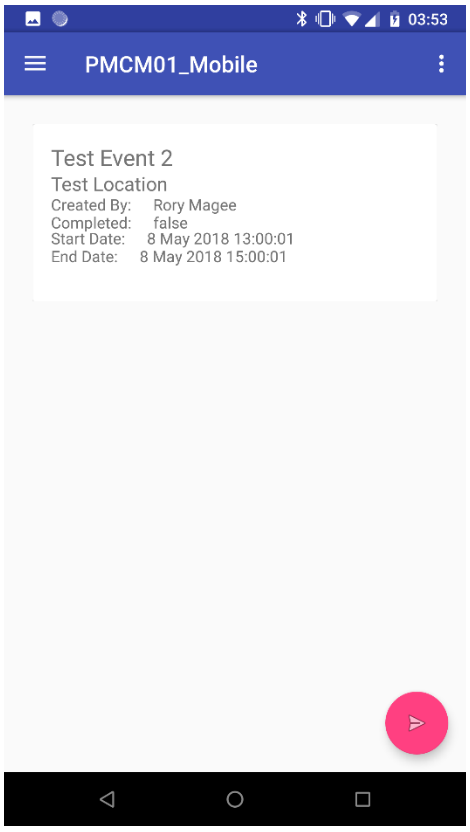
8 Appendices



Appendix 1 Login Screen

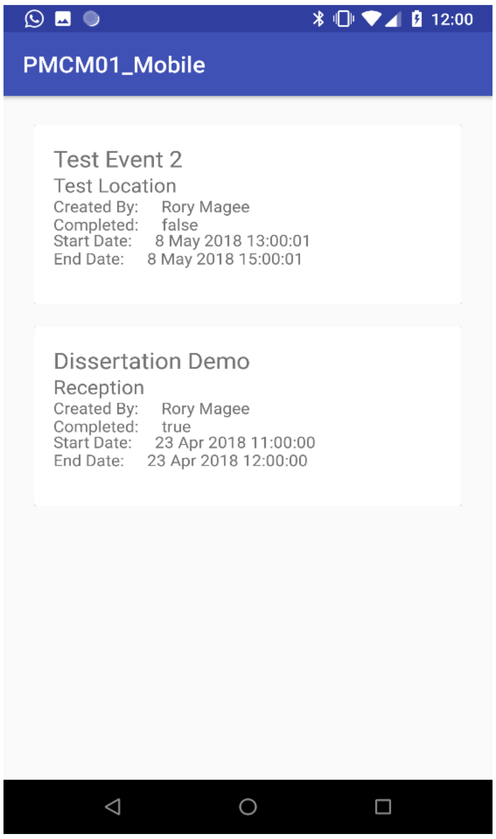


Appendix 1 Create Account Screen



Test Event 2
Test Location
Created By: Rory Magee
Completed: false
Start Date: 8 May 2018 13:00:01
End Date: 8 May 2018 15:00:01

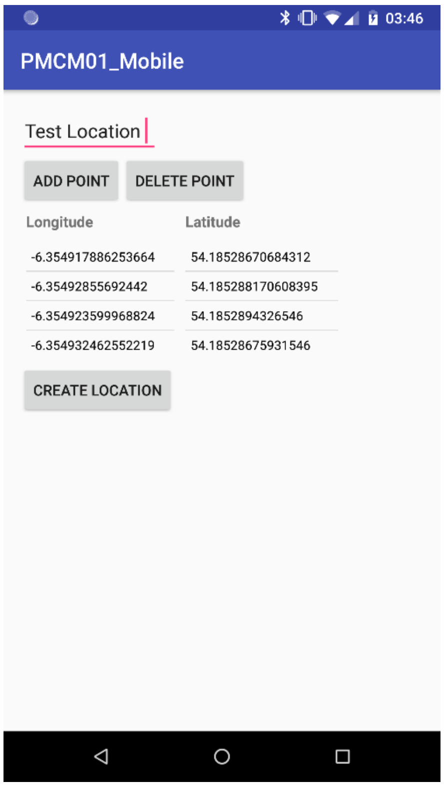
Appendix 3 Upcoming Events Screen



Test Event 2
Test Location
Created By: Rory Magee
Completed: false
Start Date: 8 May 2018 13:00:01
End Date: 8 May 2018 15:00:01

Dissertation Demo
Reception
Created By: Rory Magee
Completed: true
Start Date: 23 Apr 2018 11:00:00
End Date: 23 Apr 2018 12:00:00

Appendix 4 All Events Screen


03:46

PMCM01_Mobile

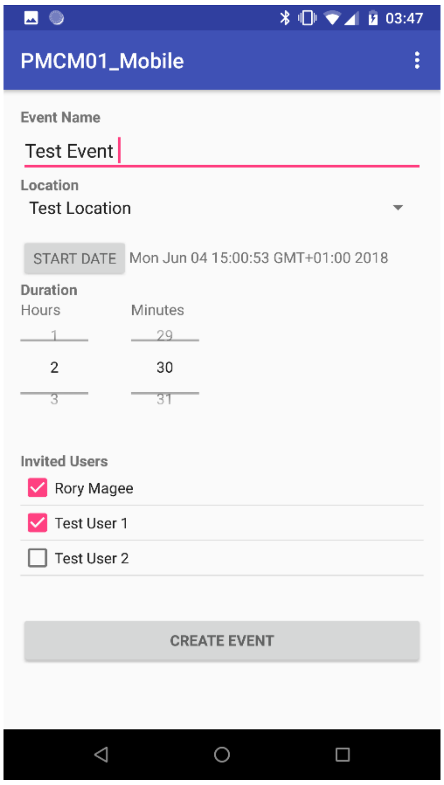
Test Location

ADD POINT DELETE POINT

Longitude	Latitude
-6.354917886253664	54.18528670684312
-6.35492855692442	54.185288170608395
-6.354923599968824	54.1852894326546
-6.354932462552219	54.18528675931546

CREATE LOCATION

Appendix 5 Create Location Screen


03:47

PMCM01_Mobile

Event Name
Test Event

Location
Test Location

START DATE Mon Jun 04 15:00:53 GMT+01:00 2018

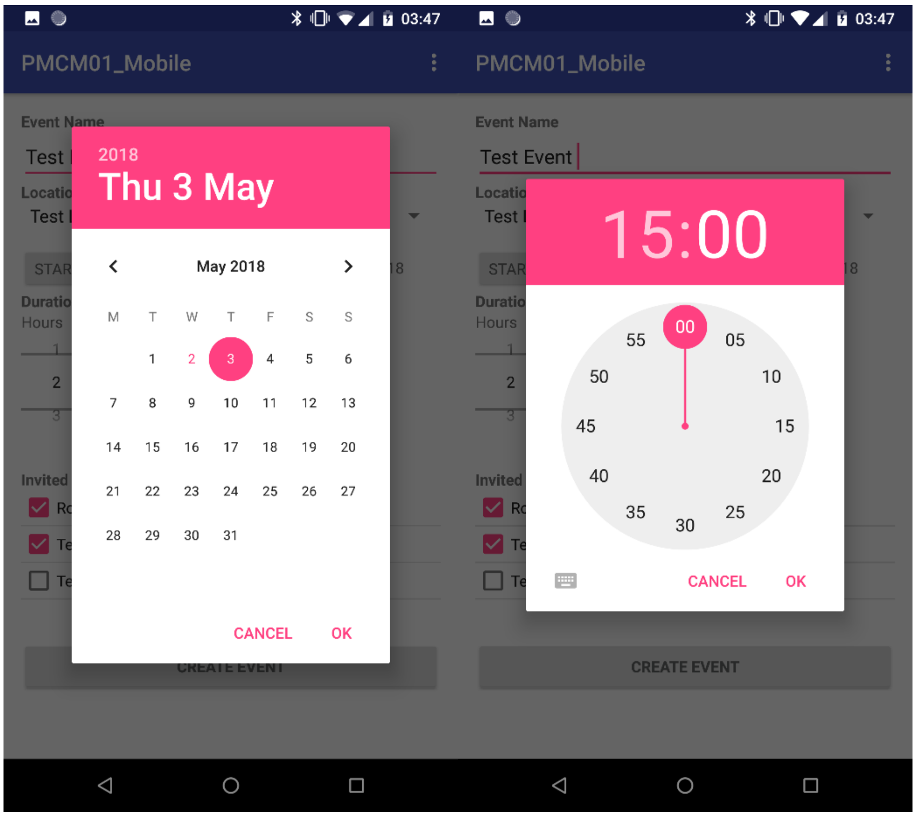
Duration

Hours	Minutes
1	29
2	30
3	31

Invited Users
- [✓] Rory Magee
- [✓] Test User 1
- [ ] Test User 2

CREATE EVENT

Appendix 6 Create Event Screen


03:47

PMCM01_Mobile

2018
Thu 3 May

May 2018

M	T	W	T	F	S	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

CANCEL OK

03:47

PMCM01_Mobile

Event Name
Test Event

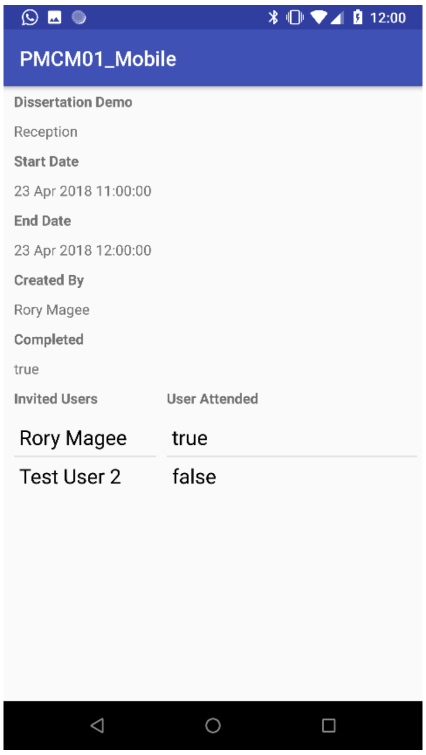
15:00

00 05 10 15 20 25 30 35 40 45 50 55

CANCEL OK

CREATE EVENT

Appendix 7 Selecting the date/time for an event



Dissertation Demo

Reception

Start Date

23 Apr 2018 11:00:00

End Date

23 Apr 2018 12:00:00

Created By

Rory Magee

Completed

true

Invited Users	User Attended
Rory Magee	true
Test User 2	false

Appendix 2 Individual Event Details