Calendar+
An Android Application to Manage Multiple Calendars

GRADUATE PROJECT REPORT

Submitted to the Faculty of

The School of Engineering & Computing Sciences

Texas A&M University-Corpus Christi

Corpus Christi, TX

in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Computer Science

by

Sandeep Kanaparthi

Spring 2015

Committee Members

Dr. Ajay Katangur _____

Committee Chairperson

Dr. Dulal Kar _____

Committee Member

Abstract

Calendar+ is an android application which accesses the calendars of multiple users, for instance members of a family or staff in a particular department and will pull out the events from every calendar. Hence, everyone in the group can see others' events on one screen in customized format. It also helps the user to create, edit, and delete events of his/her calendar. Additionally, this application will use an efficient strategy to update data only if there is any change in the events, making it faster. There are many other applications in the Play store but they cannot get rid of redundant events and reminders of events belonging to other users. This causes a lot of disturbance to the user. On the contrary, Calendar+ application will use an intelligent algorithm to compare the events in many ways based on the time, name, and place of event. Furthermore, this application will only generate reminders for the user events.

TABLE OF CONTENTS

Topics	Page No.
Abstract	ii
Table of Contents	iii
1. Background and Rationale	1
1.1 Introduction	1
1.1.1 Android	2
1.1.2 Calendar	3
1.2 Previous Work	4
1.2.1 Existing Applications	5
1.3 Solution	7
2. Narrative	8
2.1 Motivation	8
2.2 Project Description	9
2.3 System	10
2.4 Project Scope	10
2.5 Functionality	11

3. System Design 12

3.1 System Design 12

3.2 Design Flow 13

3.3 Use-Case Diagram 15

3.4 Sequence Diagram 16

3.5 Libraries 17

3.6 Dependencies 17

3.7 Data Models 17

3.8 Android Components 21

3.9 Requirements 21

3.10 Outcome 21

4. System Implementation 22

4.1 Login 23

4.2 Registration 25

4.3 Dash Board 27

4.3.1 View My Profile 28

4.3.2 View Group Members/ Add Group Member 29

4.3.3 Synchronize Contacts 30

4.3.4 Refresh 31

4.3.5 Logout 32

4.4 List Events 33

4.4.1 Add Event 34

4.4.2 Update/Delete Event 35

4.4.3 View by 36

4.4.4 Merging/Unmerging events 37

4.4.5 Refresh 39

5. Testing and evaluation 43

5.1 Login 44

5.2 Registration 44

5.3View My Profile 44

5.4 View Group 48

5.5 List Events 49

5.6 Merging and Unmerging events 49

5.7 Contacts Synchronization 49

5.8 Add/Update and Delete data 50

5.9 Logout 55

6. Conclusion and Future work 56

7. Bibliography and References 57

List of Figures

Figure No.	Name of the Figure	Page No.
1.1	Increase in smart phone market over years	1
1.2	Android System Architecture	2
3.1	System Design	12
3.2	Design Flow	14
3.3	Use-Case Diagram	15
3.4	Sequence Diagram	16
3.5	Code snippet for Request-Data Model	19
3.6	Code snippet for Result-Data Model	20
4.1	Login Screen	23
4.2	Code snippet for login validation	24
4.3	Code snippet to get user list from Parse Web Server	24
4.4	Registration Page	25
4.5	Code snippet for submitting user details to the PWS	26
4.6	Code snippet for submitting user details to local database	26
4.7	Dashboard and Dashboard with Option Menu	27
4.8	Code snippet for calendar screen fragment	28

4.9	Code snippet for updating username in the local database	28
4.10	View Profile Screen	29
4.11	Add Member Screen	30
4.12	Syncing contacts progress screen	31
4.13	Code snippet for format calendar screen	32
4.14	Logout dialog	32
4.15	List view and List view with option	33
4.16	Code snippet for navigation to add event screen	34
4.17	Code snippet for adding event to Google calendar	34
4.18	Code snippet for adding event to the Parse Web Server	35
4.19	Code snippet to delete events	35
4.20	Viewing events by day/week month and year	36
4.21	Code snippet for selecting dates for view by functionality	37
4.22	Code snippet for merging/ unmerging algorithm	38
4.23	Code snippet for fetching events for Google calendar	39
4.24	Code snippet for fetching events from the PWS to Local DB	40
4.25	Code snippet for getting details about deleted events	41
4.26	Code snippet to get new and updated events	41

4.27	Pseudo code of the list activity	42
5.1	Testing for the Login Screen	45
5.2	Testing for the Registration Screen	46
5.3	Testing for My Profile Screen	47
5.4	Testing for Group Members	48
5.5	Testing for Contact Synchronization	50
5.6	Testing for adding an event	52
5.7	Testing for updating an event	53
5.8	Testing for deleting an event	54
5.9	Testing for back navigation	55

1. BACKGROUND AND RATIONALE

1.1 Introduction

Smart phones are extremely versatile gadgets, offering unlimited chances to utilize our time efficiently. On average, a smart phone has 41 applications, and every day more new phones are being activated [1].

Perhaps the most startling statistic is that currently, 1.3 million Android devices are activated every day which means that every 24 hours, more than four times as many new smart phones and tablets are set up than babies are born. And once the devices are activated, they are checked on average 150 times a day once every 6.5 minutes. The apps in both Apple's and Google's app store combined is over 2 millions [1].

Admittedly, the apps are the purpose behind the smart phone. Furthermore, on average every smart phone contains 41 apps which are used for regular needs. The Statistics in figure 1.1 shows the increase in smart phone usage [1].

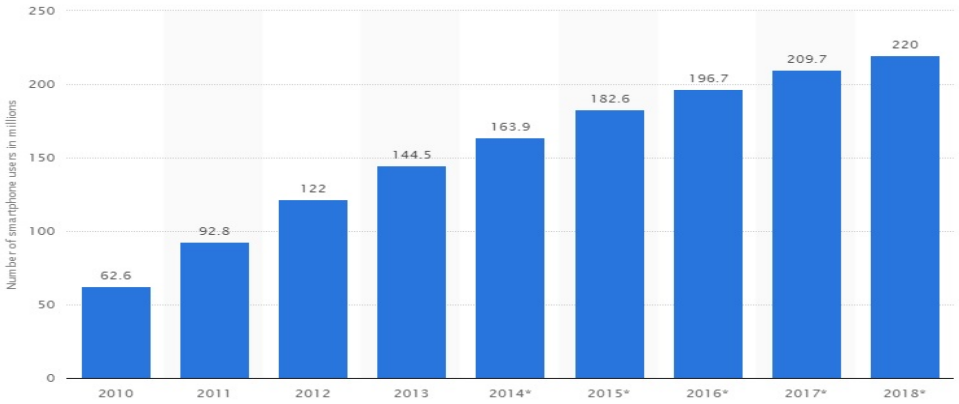


Figure 1.1: Increase in smart phone market over years

According to the Pew Research Center, as of January 2014, upwards of 60% of American adults are using smart phones and very close to 45% of American adults have tablets. Adding to that, mobile phone users are switching to smart phones as they are advancing with 3G and 4G networks. On the other hand, out of 2 millions of mobile application about 60% of the apps are not being downloaded once. Very few of the applications are used on a regular basis. As a matter of fact, the calendar application is the one app which is widely used [2].

1.1.1 Android

Android is an operating system which runs on Linux kernel. This system is mainly used on handling devices such as smart phones, tablets, and also extends to portable devices as well. Google's android mobile operating system has more users than any other operating system. According to research firm IDC, android has 78% users compared to 22% users for Apple IOS, Windows and Blackberry together [3].

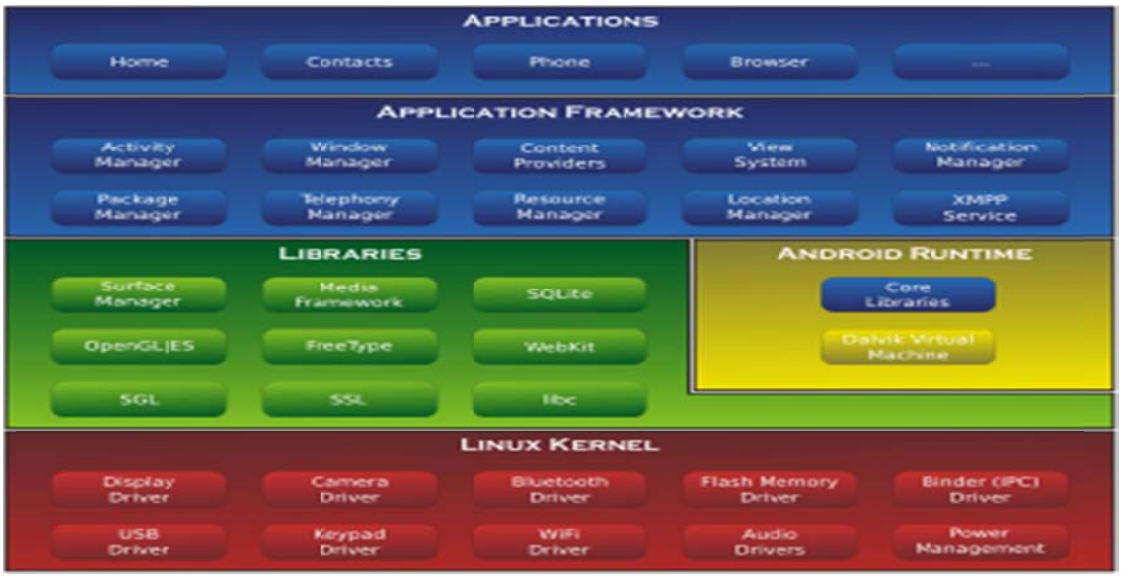


Figure 1.2: Android System Architecture

As shown in the figure 1.2, the android system is typically divided into four levels:

1. Applications (contacts, home, phone, browser…)

2. Application framework (Manager for Activity, Window, Packages…)

3. Libraries (SQLite, OpenGL, SSL...) and Runtime (Dalvik VM, Core libs...)

4. Linux kernel (Display camera, flash, wifi, audio, IPC (binder)…)

Google Play is a preeminent place for selling and distributing the apps. After completing all levels and phases of the project, Google offers a market place to where the owners can upload apps and costumers can download and install apps. This Google play will automatically notify the users when a new version of an app is available in the market [4]. Every smart phone comes with a default calendar application which allows user to synchronize mail accounts.

1.1.2 Calendar

A calendar is a tool that keeps every aspect of our life focused in one place, allowing you to worry less and accomplish more. Using the calendar is the best way to avoid procrastination and general tardiness.

The Calendar applications contain the following fields and controls:

- Calendar Date Field – Provides the current date or enables a user to select a date that he/she wants to view.
- Calendar View Options – Enables a user to choose day view, week view and year view.
- Calendar Selection – User can have multiple accounts synchronized to the device. Calendar Selection enables to choose the calendar account that he/she wants to view.

- Calendar Controls – Provides controls for each individual calendar to set date, month or year details.

1.2 Previous Work

Calendar app is the most ancient and best way for time management. The first cellular device with the calendar app was Simon [5]. Though, it is not called a smart phone but it is the first cellular device which has address book, calendar, and appointment scheduler apps with its touch screen display. Since then many works have been going on in improving the calendar app [5].

Mobile Operating System is an operating system designed for smart phones, tablets and other mobile devices. It is just like a regular operating system for a computer but it combines other features along with camera, audio, video, radio, bluetooth, speech recognition and many others. In the present market, there are eight well known mobile operating systems [5]. They are listed below:

- Android
- iOS
- Windows phone
- Blackberry
- Firefox OS
- Sailfish OS
- Tizen
- Ubuntu Touch OS

In fact, out of all mobile operating systems working on calendar app, Google's android Calendar app has been dominating the market since 2010 [5]. The main reasons behind the demand of the Google android calendar are its features. They are listed below:

1. User Interface
2. Contact access
3. Sharing Calendars
4. Device Synchronization
5. Google integration

Google Calendar is integrated with other Google services:

1. Gmail
2. iGoogle
3. Google desktop

Google android calendar can be used to store event details such as meetings, conferences, seminars, family functions etc. Most recently, Microsoft acquired a calendar app called Sunrise. With this app users can connect with Google calendar, iOS calendar and with Outlook calendar. He/she can also connect to the third-party integrations like Ever note, Google tasks, GitHub and many others [6]. However, this app also support accessing events from multi calendars but does not share calendar with other users.

1.2.1 Existing Calendar Applications

One of our problems with existing calendar applications is accessing a calendar by multiple users. Other developers have tried to solve this issue in a different way.

Some existing calendars are available in the android play store are listed below:

- Google Calendar
- aCalendar
- Agenda
- EverCalendar

However, the existing apps have their own disadvantages. Google calendar is most popular application in the present market. In spite of its popular features, it has some drawbacks too. Some of them are discussed below.

Users need to add their email account in the default calendar on the device to view events. But, to know others' events, for instance one of the family member's events we need to add his/her account to the default calendar on the device. In this way, we can view their events but most of us don't want to share email password with others. This is one of the major disadvantages.

Assuming the email address and password are being shared, others' account can be added to the calendar. By doing so, the default calendar app will give notification alerts for the newly added account events as well which creates a lot of disturbance to the current user. On the other hand, there is chance where current user can edit/delete and add unnecessary events to the other account. Moreover, if there are any similar events in both the accounts, the default calendar app shows every event as individual events which may irritate the current user. In the same way, the application like aCalendar, Agenda, EverCalendar will also generate notification alerts for others' events and also they also do not get rid of redundant events.

1.3 Solution

This project is based on an idea to overcome the limitation discussed in section 1.2.1 by getting the all the users' event details and accessing them in a single calendar. Android operating system provides more advantages for creating and accessing a single calendar with many users. So it's most appropriate to develop this project as a calendar application in android platform.

The Calendar+ application developed solves the problem of more than one user accessing the calendar to save their events. This application has a login page and registration page. In the register page the user creating a new group will create a shared-password. Later, he/she will add members to the group and also share this password with other users so that they can register and login. There is no limit on the number of users accessing the Calendar+ application. The database will store all the registration details in local database and also in Parse Web Server (PWS) by creating tables [7]. After this process is completed, users can add, update, and delete their events directly from this application.

Additionally, the application facilitates viewing the calendar by day, week, month or year. Furthermore, the application also has options to create events with or without notifications.

2. Narrative

The world is moving so fast with the rapid advancement in the technology. Proper scheduling of events has become a primary thing to keep up in the present world.

2.1 Motivation

In this fast moving world, most of the users need applications which can save their time. Android calendar is one of the applications which can save a lot of time by scheduling event details and thus user can worry less and accomplish more.

But, in this modern era, it is also important that we don't waste much time in scheduling an event. It is necessary to know about others schedule before meeting them so that we don't need to wait. For an instance if a group of 10 people are planning to get-together, it takes a great amount of time to know each of the group member's calendars and schedule an event. This is the motivation behind this application to store the event details in a single calendar with multiple user access. The other reason behind this project is that few applications like aCalendar application, can support multiple user accounts in a single calendar but they generate alert notifications for unrelated events to the user and also cannot get rid of redundant events. For an instance, in the above example if 10 people have the same event, the present available apps give alert notification 10 times for a single event which creates a lot of disturbance to users. The Calendar+ makes sure that the users are not disturbed by the alert notifications for the events they are not related to and also gets rid of redundant events.

2.2 Project Description

The Multiple Access Calendar application overcomes the limitations discussed in section 1.2.1. This application uses Parse Web Server to store all the users' events and can be accessed in a single calendar. The app synchronizes with the mobile device when it is installed.

Synchronization is the process of initiating consistency among the source data to target data storage and vice versa. This process takes several minutes to synchronize the data from the source to target storage. In android mobile operating system, user has facility to synchronize with the account in the device. By doing so, the application will synchronize the data from the mobile device. With the storage once the synchronization is completed, it will automatically collect the data like contacts, groups and calendar resources from the device. Similarly, the Calendar+ application synchronizes with the Google account in the default calendar on the device and stores it on the Parse Web Server. Later, the user can directly add, delete and update event details directly from the Calendar+ application.

This application also has many user friendly functionalities like getting rid of redundant events by using an algorithm to merge and unmerge the events. Furthermore, a color scheme is used to differentiate the users' events. More information on color scheme is mentioned in 4th chapter. Adding to that, the app provides customized format to view the calendar in a day, week, month or year mode.

2.3 System

Calendar+ application continuously shares the event details with others using Parse Web Server. User need to login into the application by registering into the system. If the User is new to the calendar+ application and want to create a new group to share the event details he/she should register into the system by creating a shared password. ON the other hand, if the user wants to join in an existing group, he/she should login directly with the shared password of the group in which he/she wants to join. The details are stored in Local database and also in Parse Web Server. Once user enters into the application, it will automatically retrieve the event and contact details from the given Gmail account. After synchronization is done, events details are stored in Parse Web Server. Other users' name and event details will be automatically retrieved from the Parse Web Sever.

2.4 Project Scope

This application can be downloaded from the android play store and installed in the device. Once the application is installed, a user can login and logout at his/her convenience. When the user login, the events are retrieved. After the entire events are retrieved the user can add, update and delete events in this multiple access calendar. Before using this application by multiple users, one of the users must create a shared password and this can be shared with other users to use this calendar. Other users must login with the share-password. This multiple access calendar is provided in apk format and is shared through email.

2.5 Functionality

This application provides multiple functionalities to make it more user-friendly with an interactive user interface.

- Single interface for user convenience.
- Single touch login and logout.
- Create / update / delete events
- Merging/Unmerging the events
- Add Members
- Synchronizing contacts
- Notifications
- Settings
- Reminders

3. SYSTEM DESIGN

This chapter discusses the design of the architecture of the entire system. This chapter also discusses the flow diagrams, use case diagram, modules, important components and class diagrams.

3.1 System Design

System design of Calendar+ application is shown in figure 3.1

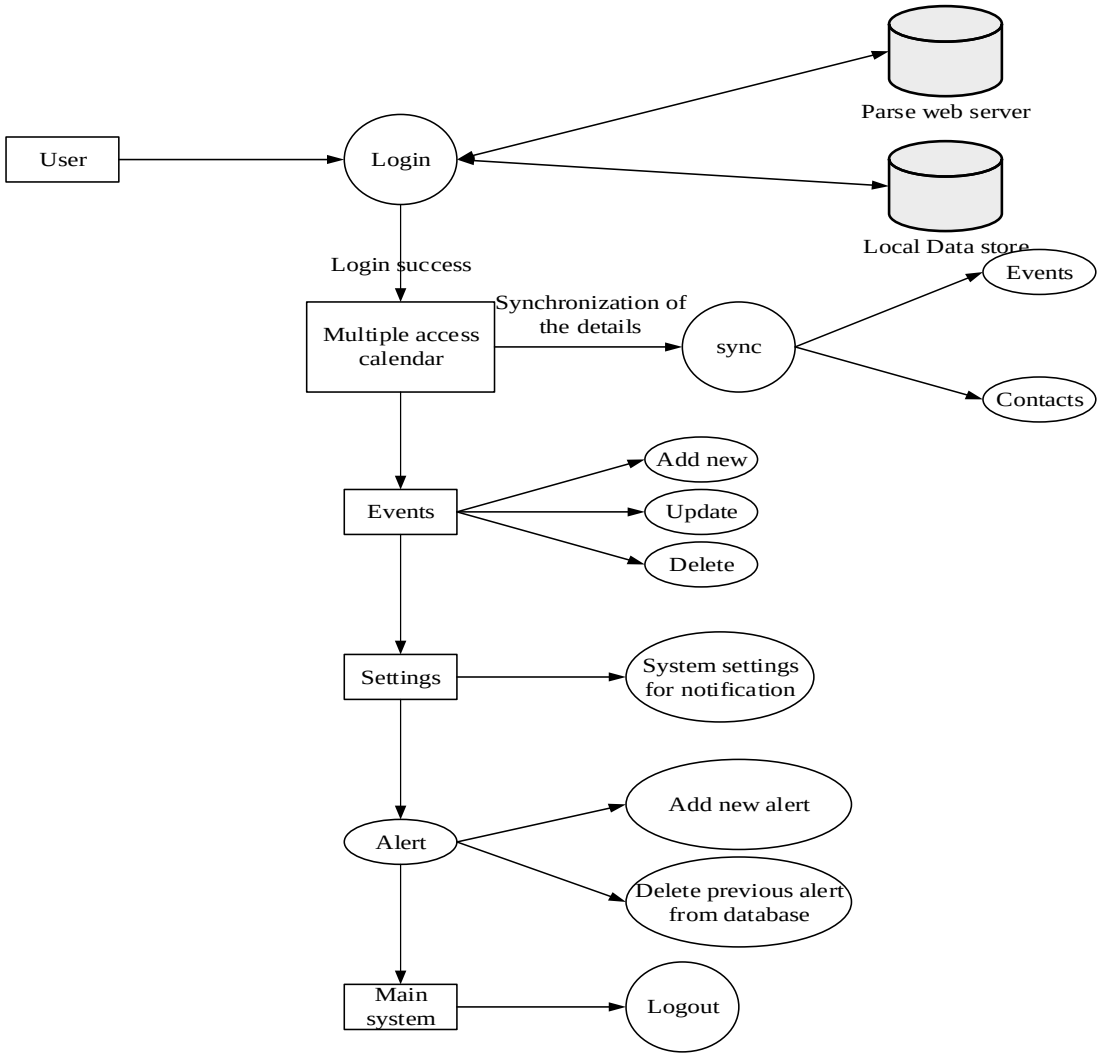


Figure 3.1: System Design

The Calendar+ application uses Parse Web Server to store username, password when the user signs up. The *Login* validation checks the username and password entered with the username and password in the Parse Web Server and confirms or rejects login accordingly. Upon confirmation, this application will connect to Gmail and synchronizes events and contacts. After doing so, this application will also store all the events in the Parse Web Server. This application allows user to add, update and delete the events. User can add a member to his group and an invite will be sent to the new member through Gmail. Members added in a group can view events of other members in the group.

3.2 Design Flow

Figure 3.2 shows the design flow of Calendar+ application. The app starts with *login* screen. The *Login* Screen has options for the user to sign in or signup. If the user selects the signup option, app goes to *register* screen. This screen uses *RegisterAcitivity* class to handle the details such as username, email and group password. With the help of *DBHelper* and *ParseHelper* classes user details are stored in local database and Parse Web Server. On successful registration, *calendar_screen* appears.

If the user chooses login option from the *login* screen, the data from Parse Web Server are saved to the local database. This Screen uses *LoginActivity* class which calls the *DBHelper* class for validation. On Successful login, user will be shown *calendar_screen* where the calendars view is displayed.

The *calendar_screen* has also option items to view/update their profile information, add/view/delete members in their group and logout from the app. When user selects a date in *calendar_screen*, the *DBHelper* class loads all events that come after the

selected date. It moves to *List_events* screen which lists the events with summary, start date, start and end time, and list of attendees. When user selects an event, *list_item_event* screen appears where users have option to update/delete the selected events. The *EditEventsActivity* class handles changes in event details and updates to *DBHelper* and *ParseHelper* classes. From *List_events* screen, user can add event using add event action item. When the user adds a new event, it is updated to his/her Google calendar and Parse Web Server as well. From *calendar_screen*, user can move to the *new_group_member* screen, where the user name can be edited. From *calendar_screen*, user can selects, *view group* action, *group_members* screen is appears. In the backend this screen uses *ListMembersActivity* class which lists all the users included in the current user. From this screen, user can edit/delete group members from the list.

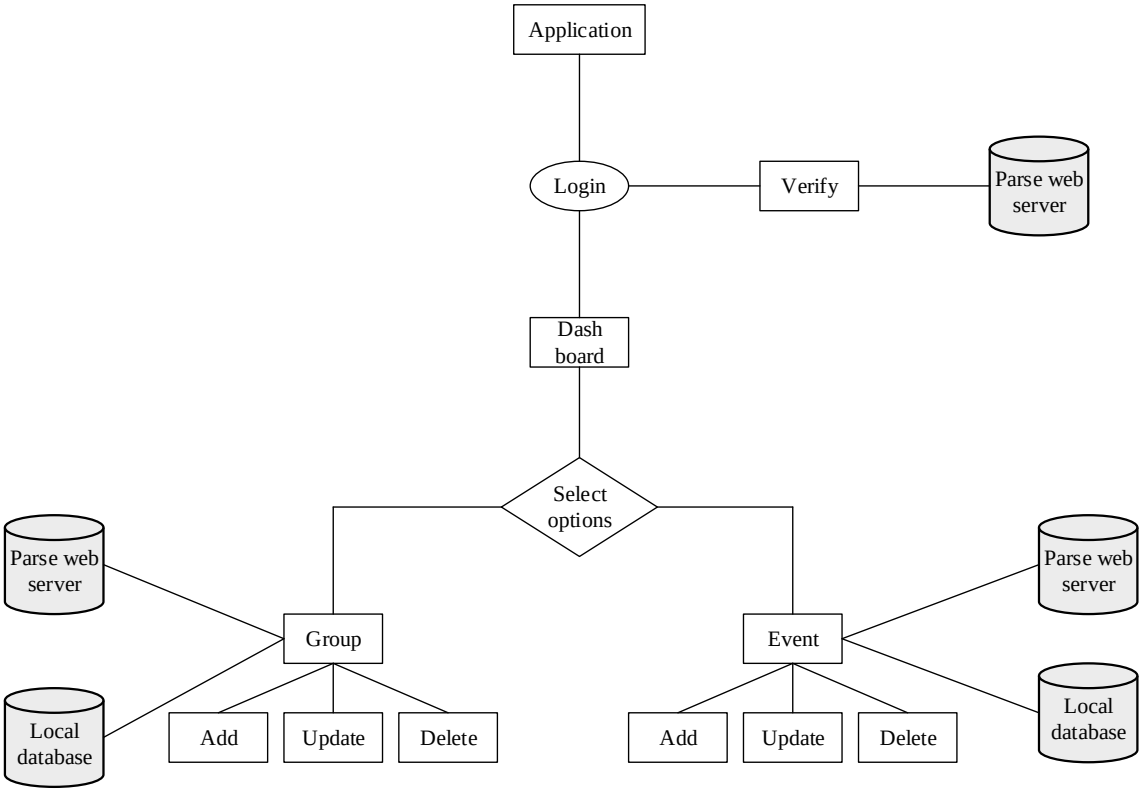


Figure 3.2: Design Flow

3.3 Use-Case Diagram

Figure 3.3 shows the use case diagram for the system with two actors and components. The actors are the user and the database. The use case diagram shows the interaction between the actors, components and database.

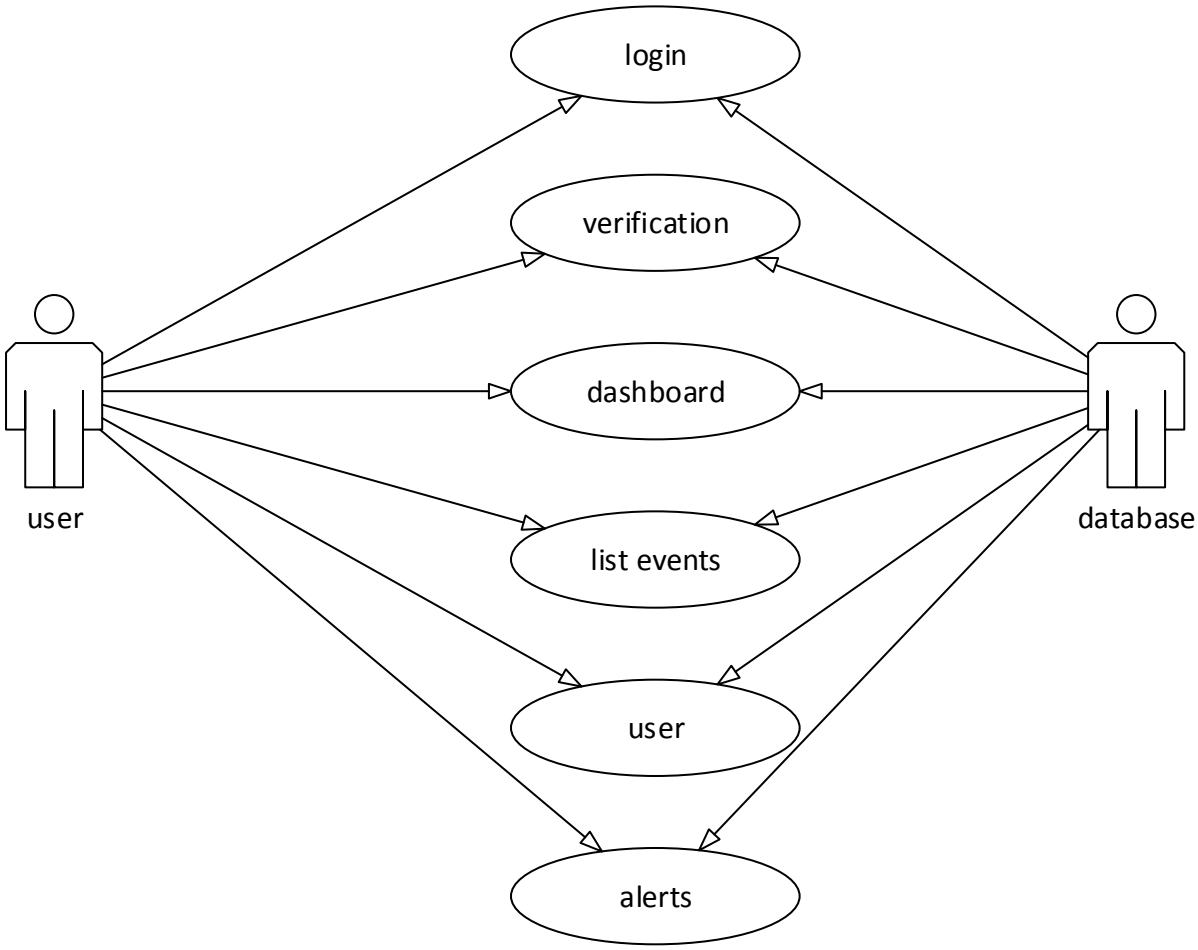


Figure 3.3: Use-case Diagram

3.4 Sequence diagram

Figure3.4 shows the sequence diagram for the activities in the application

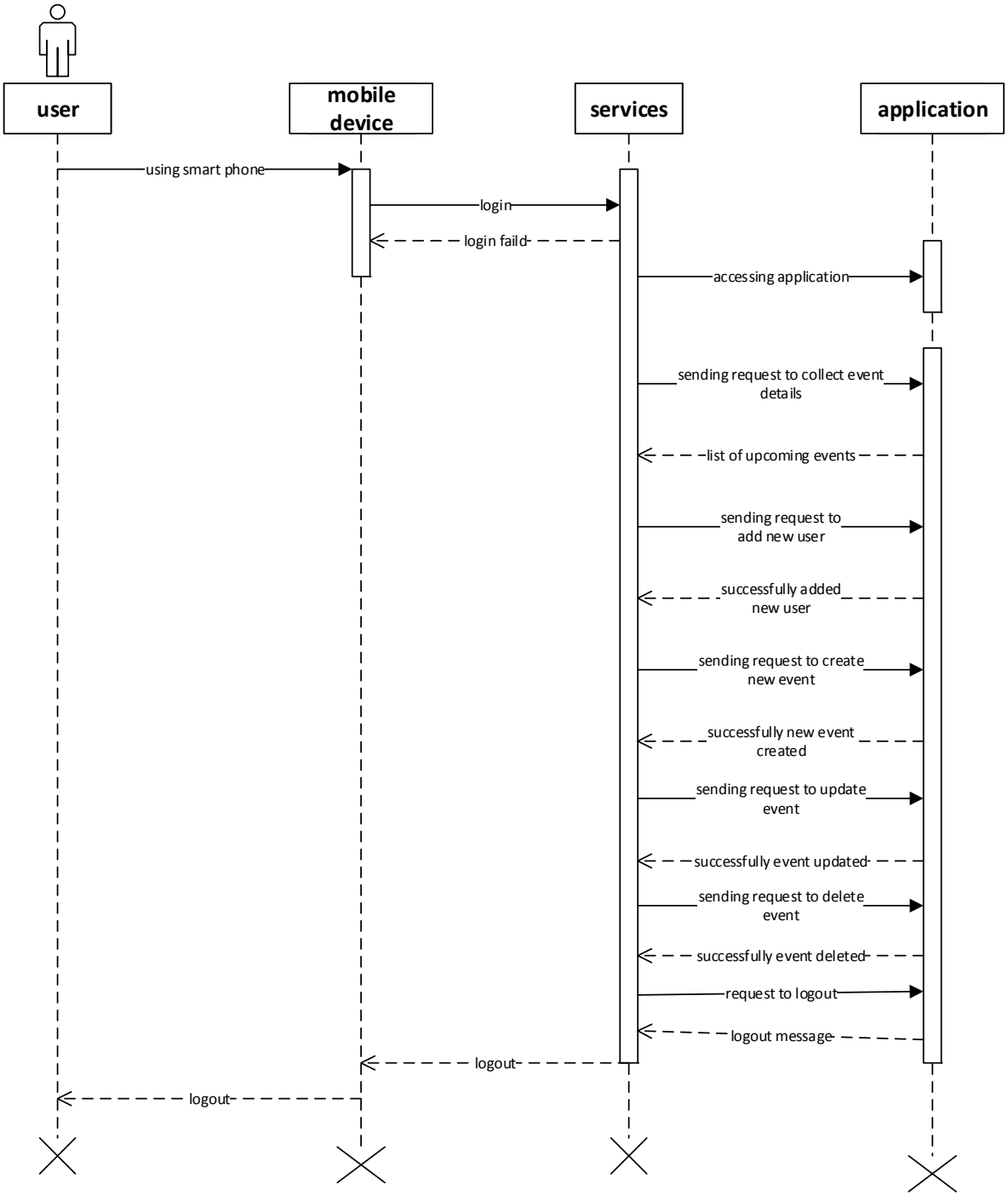


Figure 3.4: Sequence Diagram

3.5 Libraries

The following are the libraries used in the project for developing the Calendar+ application:

1. android-support-v4
2. google-play-services
3. date4
4. Bolts-android-1.1.4
5. Parse-1.8.3

3.6 Dependencies

The following are the dependencies of Calendar+ application:

1. google-play-services_lib
2. roomorama-caldroid_lib

3.7 Data Models

This app uses data models to store the data which are retrieved from Google Calendar and Parse server. The main purpose of creating these data models is to avoid multiple instances to save the event details which make the application slower.

There are four data models which we are using in this app. They are listed below.

1. UserEventVO
2. UserVO
3. RequestData
4. ResultData

1. UserEventVO

This data model is used to save user event details which need to be updated in PWS, local database and in Google calendar. For instance, if user updates an existing event, this model saves the edited data along with the event ID and updates in Parse web server, local database and also in Google Calendar by passing request to Request-Data Model.

2. UserVO

This data model is similar to the UserEventVO model but the only difference is that the data stored in this model is about user personal details like user name, ID and password. Initially, it gets all the user names from the Parse web server into this data model and saves it into local database. Additionally, by using this data model the application updates the changes in profile activity to the local database.

3. Request-Data

This local data model is used to store UserEventVO data for performing different type of actions in modules such as list events, add event, update event and delete events from Google calendar. To put it briefly,

1. In the case of adding event, the newly added event data is stored.
2. In the case of Update/Delete event, the event with updated field is stored.
3. In the case of list events, start date and view by options are stored.

In case of add event, Google Calendar Provider class uses the request data and processes it and returns the result data.

Likewise updating event, deleting event and also for listing the events. The *RequestData* class is shown in figure 3.5

```
public class RequestData {
        public static final int REQUEST_EVENTS_LIST = 1;
        public static final int REQUEST_INSERT_EVENT = 2;
        public static final int REQUEST_UPDATE_EVENT = 3;
        public static final int REQUEST_DELETE_EVENT = 4;

        private UserEventVO event;
        private int type;
        private String eventId;
        private long startDate;
        private VIEWBY viewBy;

        public UserEventVO getEvent() {
                return event;
        }
        public void setEvent(UserEventVO event) {
                this.event = event;
        }
        public int getType() {
                return type;
        }
        public void setType(int type) {
                this.type = type;
        }
        public String getEventId() {
                return eventId;
        }
        public void setEventId(String eventId) {
                this.eventId = eventId;
        }
        public long getStartDate() {
                return startDate;
        }
        public void setStartDate(long startDate) {
                this.startDate = startDate;
        }
       public VIEWBY getViewBy() {
                return viewBy;
        }
        public void setViewBy(VIEWBY viewBy) {
                this.viewBy = viewBy;
        } }
```

Figure 3.5: Code snippet for Request-Data Model

4. Result-Data

This local data model is used to hold the calendar events retrieved from Google Calendar. The *get* and *set* methods are used to store and retrieve event details from Google calendar. The *ResultData* class is shown figure 3.6

```
public class ResultData {

        private List<UserEventVO> eventsList;
        private boolean success;
        private long eventId;
        private UserEventVO event;

    public List<UserEventVO> getEventsList() {
                return eventsList;
        }

        public void setEventsList(List<UserEventVO> eventsList) {
                this.eventsList = eventsList;
        }

        public String getEventId() {
                return eventId+"";
        }

        public void setEventId(long eventId) {
                this.eventId = eventId;
        }

        public UserEventVO getEvent() {
                return event;
        }

        public void setEvent(UserEventVO event) {
                this.event = event;
        } }
```

Figure 3.6: Code snippet for Result Data-Model

3.8 Android Components

This application uses the grid view to build the interface. SQLite Database is used to store the name, user ID and password temporarily before sending it to the Parse Web Server. Google calendar's Content Resolver is used to fetch events from the calendar app. Account Manager is used to sync Google accounts which exist on the android device. The List of mail IDs can be retrieved from these Google Accounts using Contacts Provider API.

3.9 Requirements

The following are the requirement for developing this application:

1. Android device with version 4.0 or higher.
2. Eclipse IDE 4.4.2
3. JDK 1.6 or higher
4. Parse Web server to store group events

3.10 Outcome

The outcome of this project is the Calendar+ application. The application can be installed on android devices by using .apk file. This can be used as regular calendar application to create/update and delete events. Additionally, users can view events of the members in the group. It can be distributed via email for testing and usage purposes.

4. SYSTEM IMPLEMENTATION

The application is developed in Eclipse IDE and the user data is stored in the local database and Parse Web Server. The application implementation can be divided into the following stages.

1. Login
2. Registration
3. Dash Board
    a. Calendar Screen
    b. View my profile
    c. View Group/ Add Member
    d. Synchronize contacts
    e. Refresh
    f. Logout
4. List Events
    a. Add Event
    b. Update/Delete Event
    c. View by Date, Week, Month and Year
    d. Merging / Unmerging Events
    e. Refresh

4.1 Login Screen

The login screen is shown in figure 4.1. It allows the user to sign in based on two conditions:

1. He/she must be a registered user
2. Gmail account should be synchronized to the device and also to the default calendar app on the device

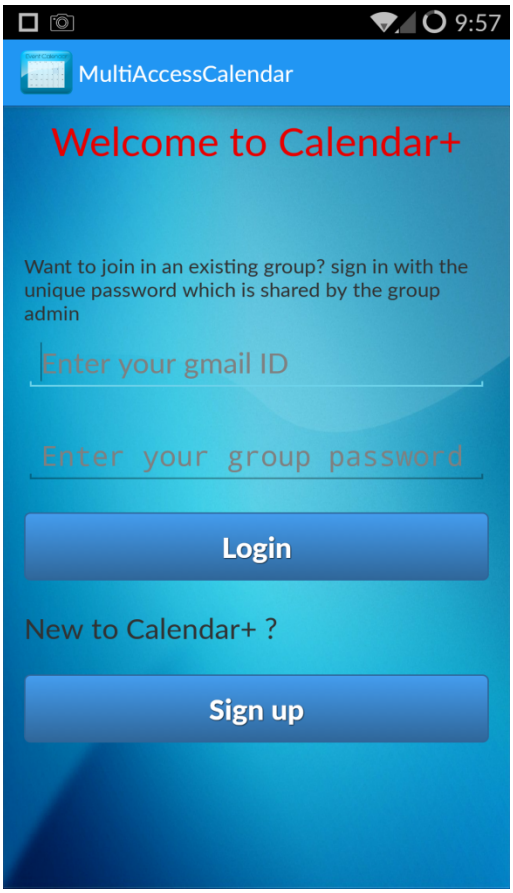


Figure 4.1: Login Screen

When the user enters the user ID and group password, it checks the Parse Web Server using *Parsehelper.finduser*() to validated the credentials as shown in the figure 4.2.

```
Public static void findUser(final FindUserCallback callback, String user ID,
                        String password) {
                ParseQuery<ParseObject> query = ParseQuery.getQuery("Group");
                query.whereEqualTo("user ID", user ID);
                if (password != null) {
                        query.whereEqualTo("groupPassword", password);
                }

if (postList != null) {
        if (postList.isEmpty()) {
           callback.onFail(Constants.USER_NOT_EXIST);
        } else {
           for (ParseObject post : postList) {
                        fallback.onSuccess(post.getInt("group ID"));
                        break;
           }
        }}
```

Figure 4.2: Code snippet for login validation

On success, the group ID of the user is retrieved, and the list of group members' details will be fetched from Parse Server, using *ParseHelper.getUserList*() as shown in figure 4.3 below and stores the user list in the local database. On failure, an alert dialog will be shown and asks the user to register before login.

```
public static void getUserList(final ListUserCallback callback, int groupId) {
                ParseQuery<ParseObject> query = ParseQuery.getQuery("Group");
                if (groupId >= 0) {
                        query.whereEqualTo("groupId", groupId);
                }
                query.findInBackground(new FindCallback<ParseObject>() {
@Override
        public void done(List<ParseObject> postList, ParseException e) {
                if (e == null) {
                        List<UserVO> userWSList = new ArrayList<UserVO>();
                        for (ParseObject post : postList) {
                                // gets the list
                                }
                                callback.done(userWSList);
                                } else {
                                callback.onFail(e.getMessage());
                                }
                        }}
```

Figure 4.3: Code snippet to get user list from Parse Web Server

After storing the user list in the local database, dashboard activity is called through intent along with the username and password details

4.2 Registration

The registration page is shown in figure 4.4. When the user enters all the details and presses the register button it checks to see if the user ID already exists in the database on Parse Web Server by using *ParseHelper.finder*() as shown in Figure 4.2.

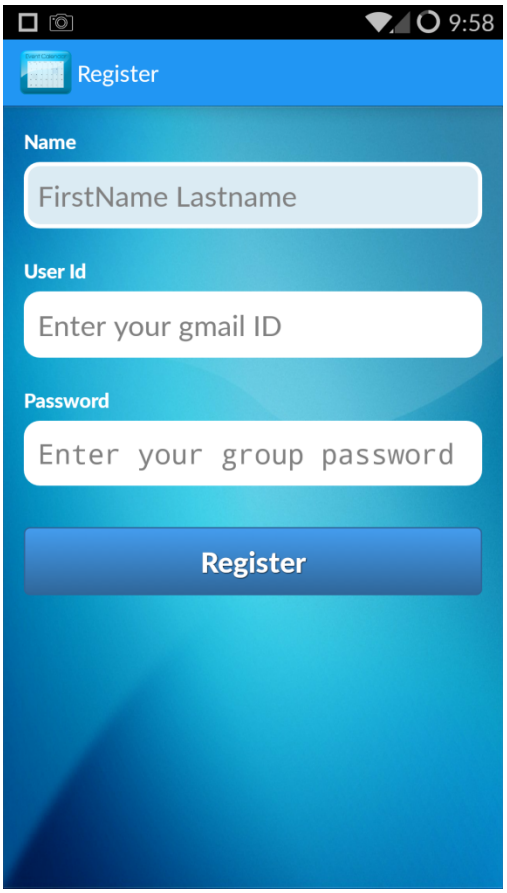


Figure 4.4: Registration Page

On success, it displays an error Dialog saying “User already exists. Please login directly with user ID and group password”. On failure, it submits the user data to the Parse Web Server as shown in Figure 4.5.

```
public static void submitUserData(final UserVO
                    final SubmitDataCallback submitUserCallback) {
        ParseObject post = new ParseObject("Group");
        post.put("userName", userVO.getUserName());
        post.put("groupPassword", userVO.getGroupPassword());
        post.put("user ID", userVO.getUser ID());
        post.put("group ID", userVO.getGroup ID());
        post.saveInBackground(new SaveCallback() { ..... }
```

Figure 4.5: Code snippet for submitting user details to the Parse Web Server

Consequently, on successful operation of adding user into the database on Parse Web Server it also adds the user details in the local database as shown in Figure 4.6.

```
public void insertUser(UserVO userVO) {
        try {
                SQLiteDatabase db = this.getWritableDatabase();
                userVO.setId(getNextID(db));
                db.insert(Constants.TABLE_USERS, null, userVO.getContentValues());
                db.close();
        } catch (SQLException ex) {
                Log.e(LOG, ex.getMessage());
        }
}
```

Figure 4.6: Code snippet for submitting user details to data to local database

In the end, it calls the dashboard activity through an intent along with username and password details.

4.3 Dashboard

The dashboard screen is shown in figure 4.7. The dashboard screen contains the calendar screen and the option menu.

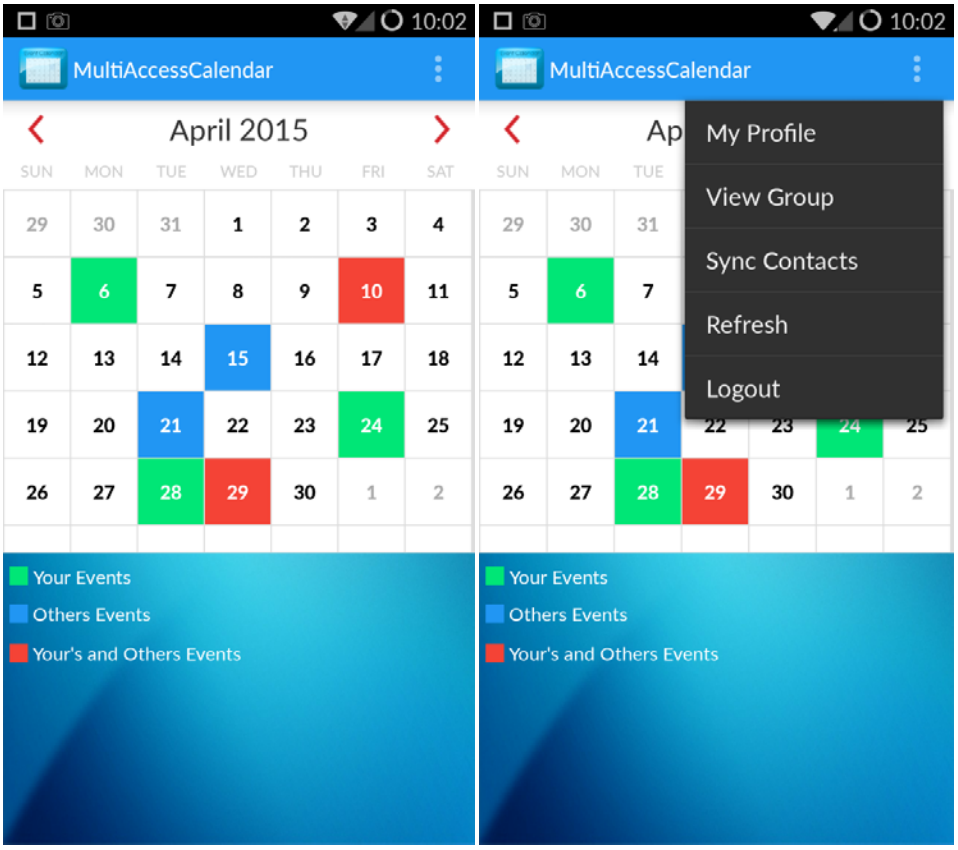


Figure 4.7: Dashboard and Dashboard with Option Menu

In option menu user can see:

1. View My profile
2. View group Members/ Add Group Member
3. Sync Contacts
4. Refresh
5. Logout

Calendar screen is displayed in the dashboard as a fragment. The *Caldroid* Listener gets the date and time when the user selects any date on the calendar. As a result, it calls *list_events* screen with date attached to it as shown in figure 4.8

```
FragmentTransaction t = getSupportFragmentManager().beginTransaction();
t.replace(R.id.calendar, caldroidFragment);
t.commit();

final CaldroidListener listener = new CaldroidListener() {

                    @Override
                    public void onSelectDate(Date date, View view) {
                            gotoEventListScreen(date.getTime());
                    }
```

Figure 4.8: Code snippet for Calendar Screen fragment

4.3.1 View My Profile

The view my profile screen is shown in figure 4.10. When the user selects the "my profile" option from the menu, the *my profile* screen appears. The dashboard activity calls the profile activity through intent along with the username. The user can change the user name in this screen. Other fields such as user ID and password are disabled. The changed user name is only updated in local database as shown in figure 4.9, not in the Parse Web Server to give flexibility to the user to create nicknames for the group members on their devices which is not shown to the group. This functionality gives user the ability to save a contact by desired name on their device.

```
private void updateUserToDB() {
                DBHelper dbHelper = DBHelper.getInstance(ProfileInfoActivity.this);
                UserVO userVO = dbHelper.getUserData(currentUser);
                userVO.setUserName(userName.getText().toString());
                dbHelper.updateUser(userVO);
                refreshUserList(userVO, false);  }
```

Figure 4.9: Code snippet for updating username in the local database

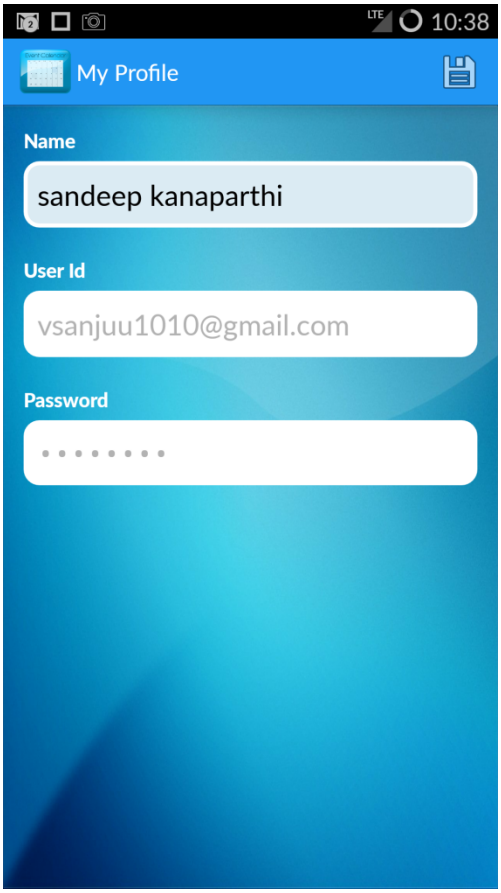


Figure 4.10: View Profile Screen

4.3.2 View group Members/ Add Group Member

When the user selects "view group" option from the menu, the *ListGroupMembers* activity starts and the Group Member list screen appears. The dashboard activity calls the *ListGroupMembers* activity through an intent along with the username attached to it. On selecting "Add User" menu item, a dialog will be shown as in figure 4.11. Using the username and user email id new user can be added to the group. On clicking Ok, the user will be added to the current group and a mail invite will be composed and shown to the user. The user can send the invitation from their Gmail app. The invitation contains sender information, receiver information, group password and a

link to download the app. However, it is not mandatory to send an invitation by Gmail. User can also send the group password directly by others forms of communication.

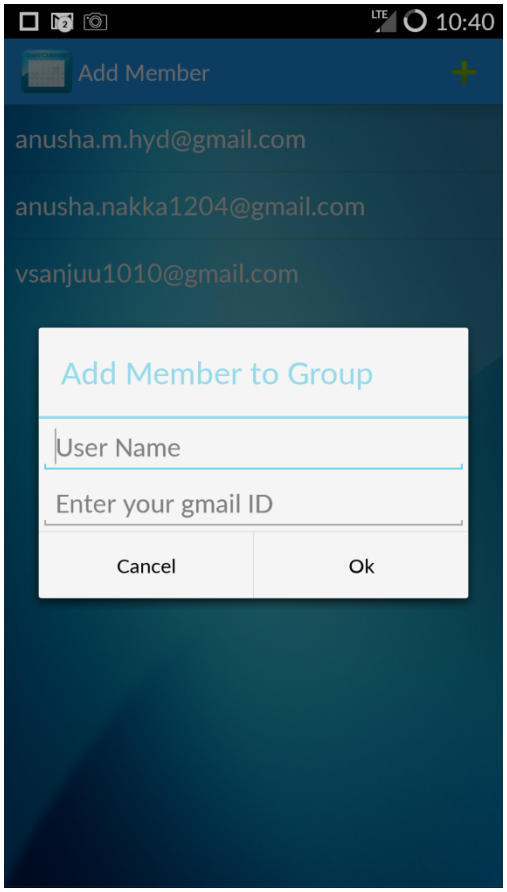


Figure 4.11: Add Member Screen

4.3.3 Synchronize Contacts

When the user selects the “Sync Contacts” option from the menu, all Google contacts will be synced with the Gmail app as shown in figure 4.12. It fetches the list of contacts associated with the email Id. This contacts list will be used to add attendees in *add_events* screen. Contacts from Google Account are synchronized using Google Content Resolver.

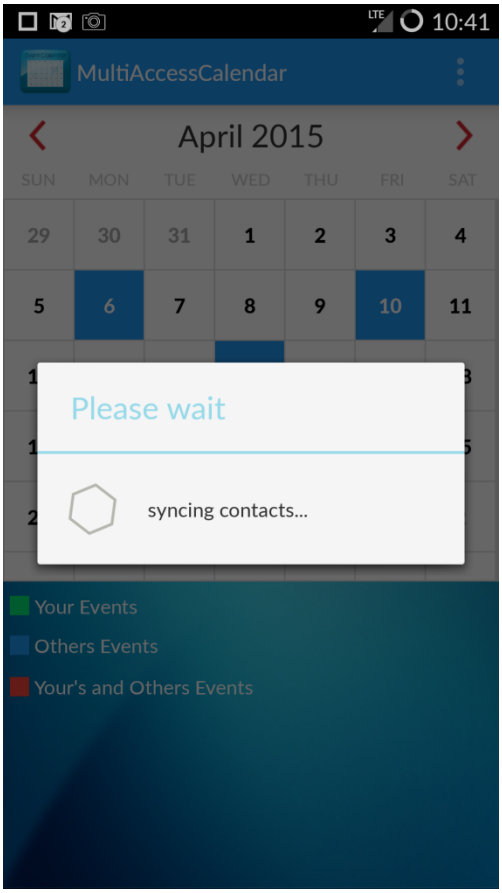


Figure 4.12: Syncing contacts progress screen

4.3.4 Refresh

This activity gets events from Parse Web Server and Google calendar and colors calendar dates based on the user events. It fills green if the date has only current user events, fills blue if the date has others' event but not current user events and red if the date contains current user events and others' events. First, the activity sends request to Google calendar app to list events from specified start date and view type. After getting the event dates, the dates are colored green as shown in figure 4.13. Similarly the blue and red colors functionalities are implemented.

```
for (Date greenDate : eventDates) {
        if (caldroidFragment != null)
        caldroidFragment.setBackgroundResourceForDate(R.color.green,greenDate);
        caldroidFragment.setTextColorForDate(R.color.white, greenDate);
```

Figure 4.13: Code snippet for format calendar screen

4.3.5 Logout

When user selects "logout" option, it displays a dialog which asks the user to confirm logout from the app. If user confirms to logout, the app will quit. The logout dialog is shown in figure 4.14

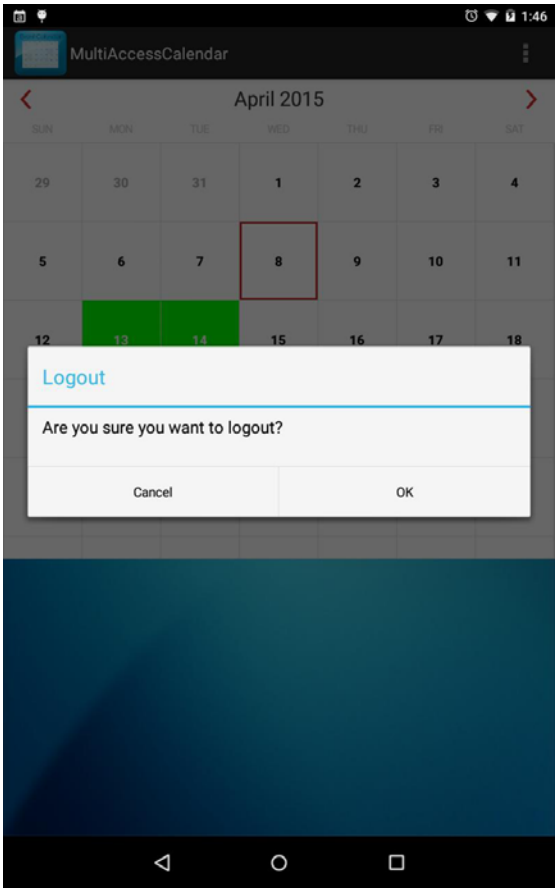


Figure 4.14: Logout dialog

4.4 List Events

ListEventsActivity is a class where events of given date range are listed. Similar to calendar screen, the events are highlighted in list view with different colors as shown in figure 4.15. Green color shows current users events and blue color for others' events. If there are any common events, they get merged and will be highlighted in orange. Green and orange highlighted events can be edited by the current user. Blue events are not editable since they don't belong to the user.

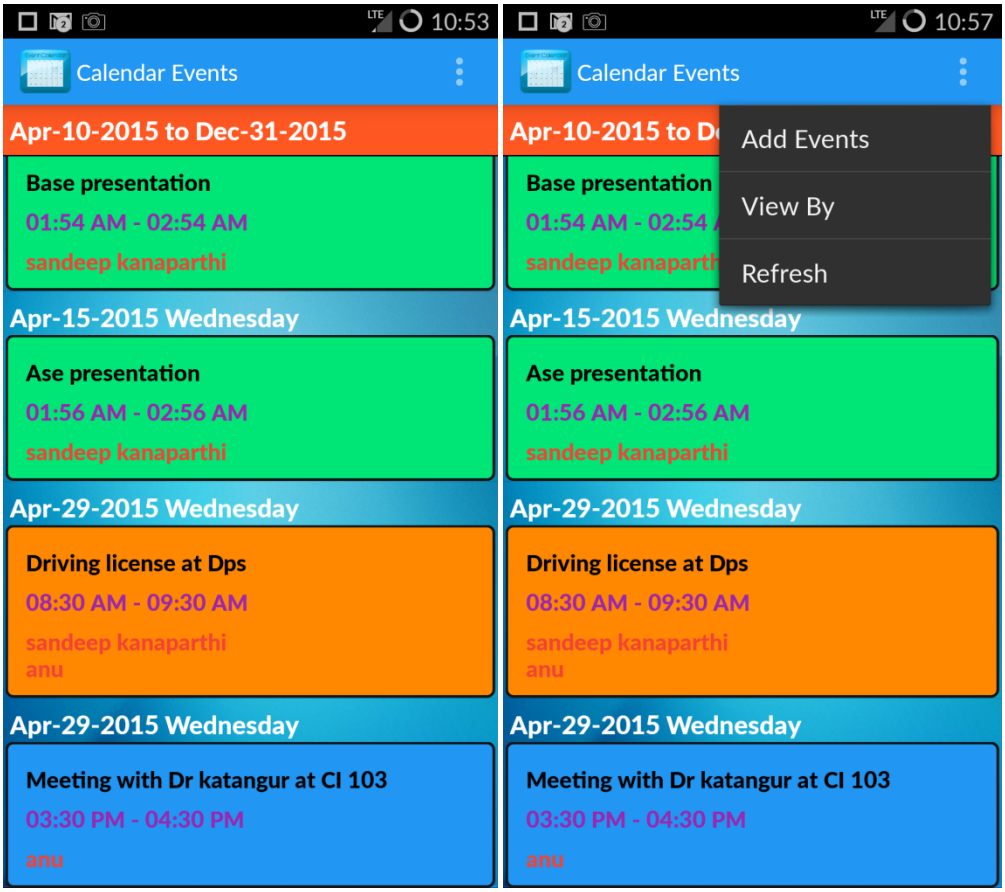


Figure 4.15: List view and List view with options

List activity has following functionalities.

1. Add events
2. Update/Delete events
3. View by Date/Week/Month and Year
4. Merging Unmerging events
5. Refresh

4.4.1 Add Events:

This screen allows user to add new event from this app. User will go to add events screen through an intent as shown in figure 4.16 using username and selected date.

```
Intent intent = new Intent(ListEventsActivity.this,AddEventsActivity.class);
                intent.putExtra(Constants.USER_NAME, userName);
                intent.putExtra(Constants.DATE, selectedDate);
                startActivity(intent);
```

Figure 4.16: Code snippet for navigation to add event screen

User can set event title, location, attendees, start date/time and end date/time. As soon as the user saves the event, the event will be added to Google calendar and Parse server. Event is added to Google calendar as shown in Figure 4.17.

```
GoogleCalendarProvider calendarService = new GoogleCalendarProvider(
                                         AddEventsActivity.this, userName, requestData,
                                         getContentResolver());
                                          calendarService.execute(); }   }); }
```

Figure 4.17: Code snippet for adding event to Google calendar

After adding to Google calendar, the event needs to be added to the Parse Web Server, so that other users in the group can see the newly created event. The event is added to the Parse Web Server as shown in figure 4.18.

```
public static void submitEventData(final UserEventVO userEventVO,
                    final SubmitDataCallback submitUserCallback) {
          ParseObject post = new ParseObject("GroupEvents");
          post.put("user ID", userEventVO.getUser ID());
          post.put("group ID", userEventVO.getGroup ID());
          post.put("eventId", userEventVO.getEventId());
          post.put("eventSummary", userEventVO.getEventSummary());
          post.put("eventLocation", userEventVO.getEventLocation());
          post.put("attendees", userEventVO.getAttendees());
          post.put("startDate", new Date(userEventVO.getStartDate()));
          post.put("endDate", new Date(userEventVO.getEndDate()));
          if (userEventVO.getReminder() != null) {
                    post.put("reminder", userEventVO.getReminder());
          }
          post.saveInBackground(new SaveCallback() { ……});}
```

Figure 4.18: Code snippet for adding event to the Parse web server

4.4.2 Update /Delete Events

The user can only update or delete his/her events. To update or delete an event the user should click on the event, which is basically either green or orange in color. The update and delete event is similar to the add event which is discussed in section 4.4.1. In the case of add event, all the fields of an event are newly created and all the fields are used in *RequestData* class model but in the case of update event , only updated field is saved along with the event ID. Similarly, for deleting events the *RequestData* class model stores the event ID of the event which is need to be deleted. Later, using this data model, the events are updated / deleted in Parse Web Server and also in Google calendar as shown in figure 4.19.

```
ParseQuery<ParseObject> query = ParseQuery.getQuery("GroupEvents");
query.whereEqualTo("eventId", eventId);
query.findInBackground(new FindCallback<ParseObject>() {
          for (ParseObject message : list) {
                    try {
                              message.delete();
                    } catch (ParseException e) {
                              submitUserCallback.onFail(e.getMessage());
                                        } }}       });
```

Figure 4.19: Code snippet to delete events

4.4.3 View By

This menu has 4 sub menus such as view by date, view by week, view by month and view by year as shown in figure 4.20. The end date for event selection range is calculated based on this option.

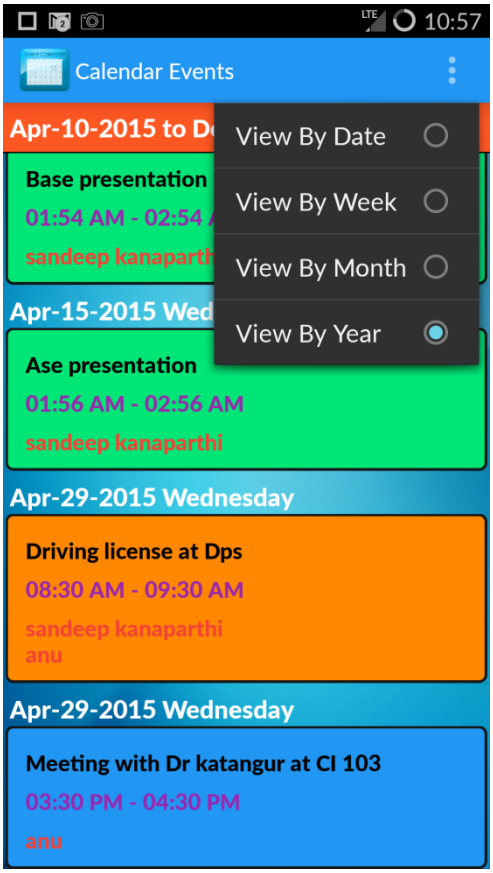


Figure 4.20: Viewing events by day/week/month and year

View by Date: Start date is selected date and End date is one day after the selected date.

View by Week: Start date is selected date and End date is the current weeks last day.

View by Month: Start date is selected date and End date is the current months end date.

View by Year: Start date is selected date and End date is Dec 31 of the current year.

Figure 4.21 shows the code for viewing the events by day.

```
public static  long getEndDate(long startDate, VIEWBY viewBy) {
                long endDate = startDate;
                Calendar cal = Calendar.getInstance();
                cal.setTimeInMillis(endDate);
                cal.set(Calendar.HOUR_OF_DAY, 0);
                cal.set(Calendar.MINUTE, 0);
                cal.set(Calendar.SECOND, 0);
                cal.set(Calendar.MILLISECOND, 0);
                if (viewBy == VIEWBY.VIEW_BY_DATE) {
                        cal.add(Calendar.DAY_OF_YEAR, 1);
                        endDate = cal.getTimeInMillis();
                }
```

Figure 4.21: Code snippet for selecting dates for view by functionality

4.4.4 Merging and Unmerging events

Events of different users in the local database will be compared against each other using an algorithm which compares the event title, location, and date-time range. The algorithm merges events if it has the same event title and same event location and the time overlap is greater than 30 minutes. Importantly, the algorithm is designed in such a way that it ignores the case sensitivity, periods and comas in the event title and event location. For instance, if one user enters the event as *"Graduation Day at Downtown"* and other user enters as *"graduation day at down town"* the algorithm understands that both events are the same and merges them.

The event when merged turns into orange color as shown in figure 4.15 and when it is unmerged they change to their respective colors depending upon who they belong to. When events are merged, only the event of current user is shown in the list. Other users' events are not added to list.

The merging/unmerging algorithm is shown in figure 4.22. All the users' events are stored in the local database. *CommonUtils.iCompare(event1, event2)* performs comparison between each and every events in the local database, and merges all the events which matches the algorithms.

```
                if (!user1.equalsIgnoreCase(user2)) {
                        String title1 = userEvent1.getEventSummary().replaceAll(" ", "")
                                        .replaceAll(",", "");
                        String title2 = userEvent2.getEventSummary().replaceAll(" ", "")
                                        .replaceAll(",", "");
                        if (title1.equalsIgnoreCase(title2)) {
                                String location1 = userEvent1.getEventLocation();
                                String location2 = userEvent2.getEventLocation();
                                if (location1 != null && location2 != null
                                                && location1.equalsIgnoreCase(location2)) {
                                        long startDate1 = userEvent1.getStartDate();
                                        long startDate2 = userEvent2.getStartDate();
                                        long endDate1 = userEvent1.getEndDate();
                                        long endDate2 = userEvent2.getEndDate();
                                        if ((startDate1 <= endDate2 && startDate2 <= endDate1)
                                                        || (startDate2 <= endDate1 && startDate1 <=
endDate2)) {
                                                long range = Math.min(endDate1, endDate2)
                                                                - Math.max(startDate1, startDate2);
                                                return (range >= 1800000);
                                        }
                                }
                        }
                }
                return false;
                        }
```

Figure 4.22: Code snippet for merging / unmerging algorithm

4.4.5 Refresh

Whenever the uses launches the List activity or clicks on “Refresh”, it gets the most updated data from the Google calendar, the Parse Web Server and local database. In the list view user will see all the events belong to the user and the group. It will also merge/unmerge the events based on the algorithm which is discussed earlier in the section 4.4.4.

First and foremost, the app collects all the events from the Google calendar and saved in *RequestData* class. All the events from Google calendar are fetched as in figure 4.23. By doing so, the application can get the events if the user adds/updates or deletes the events directly from the default calendar on the device.

```
RequestData request = new RequestData();
request.setType(RequestData.REQUEST_EVENTS_LIST);
request.setStartDate(selectedDate);
request.setViewBy(viewBy);
GoogleCalendarProvider calendarService =
new GoogleCalendarProvider ListEventsActivity.this,userName, request,getContentResolver());
calendarService.execute();
```

Figure 4.23: Code snippet for fetching events from Google calendar

The application has user events from his/her calendar, and the events of other users in the group are fetched from Parse Web Server and stored in the local database. But before that, all the previous data is cleared from the local database to just avoid duplicate and irrelevant events if any users are deleted from the group. The events from Parse server are fetched and local database is filled with Parse Web Server events as shown in figure 4.24.

```
private void updateEvents(DBHelper dbHelper, UserVO userVO,
                List<UserEventVO> eventListFromParse) {
        clearEventListFromDB(dbHelper, userVO);
        List<UserEventVO> combinedList = CommonUtils.compareEventsData(
                eventListFromParse, eventListFromDB, userVO, true);
        for (UserEventVO event : combinedList) {
                if (event.getFlag() == Constants.NEW_DATA) {
                        dbHelper.insertEvent(event);
                } else if (event.getFlag() == Constants.UPDATE_DATA) {
                        dbHelper.updateEvent(event);
                }
        }
        updateEventListFromDB(dbHelper, userVO);
        updateListView();       }
```

Figure 4.24: Code snippet for fetching events from Parse server to local database

At this point, the app has Google calendar events and the events from the Parse. But, from the current user prospective, since the Google calendar events are more updated than Parse events, we compare both for any new/updated or deleted events and changes in local database and Parse database.

To accomplish that, we need to check if there are any new/updated/deleted events from Google calendar in two phases. Phase 1: Compare for deleted events and delete them from Parse and local database. Phase 2: Compare for new/updated events in Parse Web Server and local database.

First the multi access calendar compares the events and finds deleted events. Figure 4.25 shows the code snippet to get the information about deleted events. Later this information is added to *RequestData* class model and the events are deleted from the Parse Web Server and also from local database.

```
List<UserEventVO> prevEventList, List<UserEventVO> currEventList) {
        List<String> eventIdList = new ArrayList<String>();
        for (UserEventVO event : currEventList) {
                eventIdList.add(event.getEventId());
        }
        List<String> prevEventIdList = new ArrayList<String>();
        for (UserEventVO event : prevEventList) {
                prevEventIdList.add(event.getEventId());
        }
        List<String> deletedEvents = new ArrayList<String>();
        for (String eventId : prevEventIdList) {
                if (!eventIdList.contains(eventId)) {
                        deletedEvents.add(eventId);
                }
        }
        return deletedEvents;
}
```

Figure 4.25: Code snippet for getting details about deleted events

If there are no deleted events, the application can directly go to phase 2 i.e., checks for new/updated events, otherwise events are deleted from Parse Web Server. In phase 2 the application compares for new/updated data. Based on current data in local database, the new and modified events are separated and are stored in *newEventList* array and *updateEventList* array. Here *newEventList* contains all new events to be submitted to Parse Web Server and *updateEventsList* has events to be updated in Parse Web server as shown in figure 4.26. After updating the Parse Server, the application also updates the local database.

```
        List<UserEventVO> newEventsList = new ArrayList<UserEventVO>();
        List<UserEventVO> updateEventsList = new ArrayList<UserEventVO>();
        for (UserEventVO event : combinedList) {
                if (event.getFlag() == Constants.NEW_DATA) {
                        dbHelper.insertEvent(event);
                        newEventsList.add(event);
                } else if (event.getFlag() == Constants.UPDATE_DATA) {
                        dbHelper.updateEvent(event);
                        updateEventsList.add(event);}}
```

Figure 4.26: Code snippet to get new and updated events

Once all calendar events are updated to the Parse Web Server, the application re-fetch all events from the database on Parse Web Server, for the reason that other users in the group may have created/updated/deleted events in the meanwhile. Later, the application saves the entire fetched events in local database. Now the local database will have the most updated data from Parse Web Server. After doing so, the application invokes merging / unmerging algorithm on local database and the list of events are displayed. To sum up, the pseudo code shown in the figure 4.27 explains the list activity.

```
CalendarService.Excute #to get calendar events which are not updated in the Parse
getEventFromParse       #to get events of others' events which are not updated in local database
UpdateLocalDB           #compares step 1, step 2 and updates in the local database
SubmitDataToParse       #update the changes in the parse Database
getEventFromParse       #in case if anyone in the group created/updated/deleted events
UpdateLocalDB           # update local data base with data from 5. It has most updated events.
Merging/Unmerging algorithm   # runs algorithm on the data in step 6.
View events
```

Figure 4.27: Pseudo code of the list activity

5. TESTING AND EVALUATION

The app has to be downloaded and installed on the android device. Once the app is installed it can be tested for the following functionalities.

1. Login
2. Registration
3. View My Profile
4. View Group / Add Member to the group
5. List events
6. Merging / Unmerging Events
7. Contacts Synchronization
8. Add/Update/Delete Events
9. Logout

The test cases are shown in the order mentioned above. As, this application uses Google calendar the user must add the Google account and should also synchronize Google account to the calendar app.

The procedure to add Google account in the device is as follows

1. Go to settings
2. Under Personal, select Accounts
3. Select Add Account and then Select Google
4. Give username and password, check the contacts and calendar boxes, select synchronize option

5.1 Login

Figure 5.1(a) shows the login screen. In the case of absence of any filed, a toast will be displayed as shown in figure 5.1(b). If user email address is not synchronized with default calendar app on the device it displays as a toast as in figure 5.1(c). Even if the email address is synchronized the user should be registered into the system. Otherwise a dialog alert will be displayed as shown in figure 5.1(d)

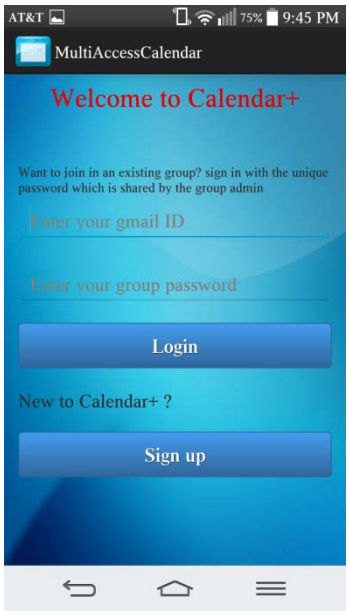
5.2 Registration

Figure 4.2(a) shows the Registration page. In case of absence of any field, a toast will be displayed as shown in the figure 5.2(b). User cannot signup if the Google account is not synchronized to the default calendar app on the device. If user tries to sign up a toast will be displayed as shown in figure 5.2(c). If the user already exists, he/she cannot be added again to the group, as this creates duplicate events in the group. If the user tries to do so a toast is displayed as shown in figure 5.2(d).

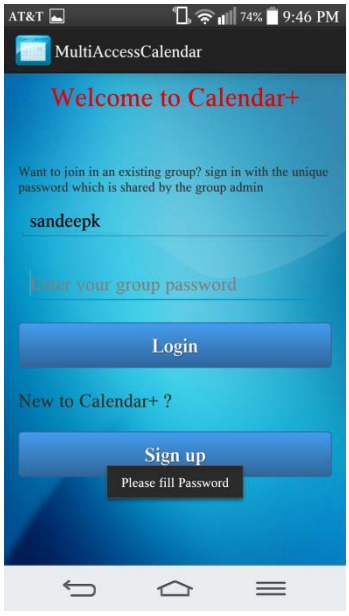
5.3 View My Profile

After successful login or registration, the Dashboard screen appears. This screen consists of calendar and options menu. “My profile” is the first option in the menu. The test case for this module is that user has to see his details correctly. Additionally, the user must be able to edit his name and the changes made should be saved to his profile.

Figure 5.3(a) shows the user data (at 7:25). User changes his name (at 7:29) as shown in figure 5.3(b) and clicks on save button. Figure 5.3(c) shows a toast displaying member data updated (at 7:30). Figure 5.3(d) shows updated user data (at 7:30)

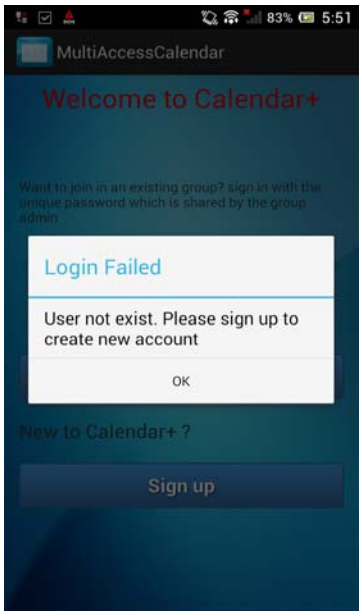


(a)



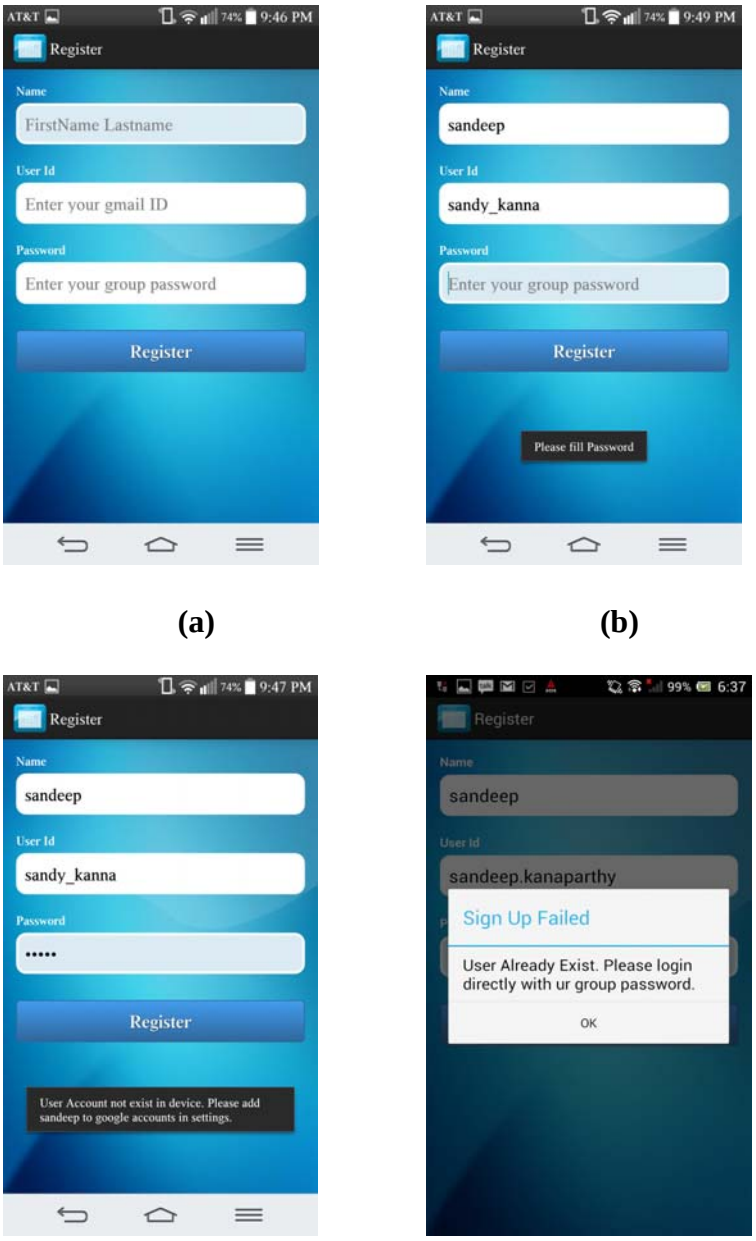
(b)

(c)



(d)

Figure 5.1: Testing for the Login Screen



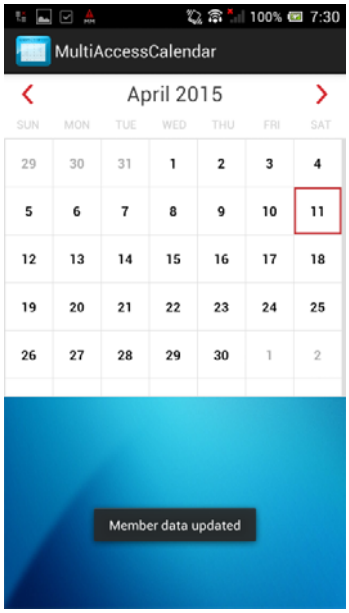
(a) **(b)**

(c) **(d)**

Figure 5.2: Testing for the Registration Screen

(a)

(b)



(c)

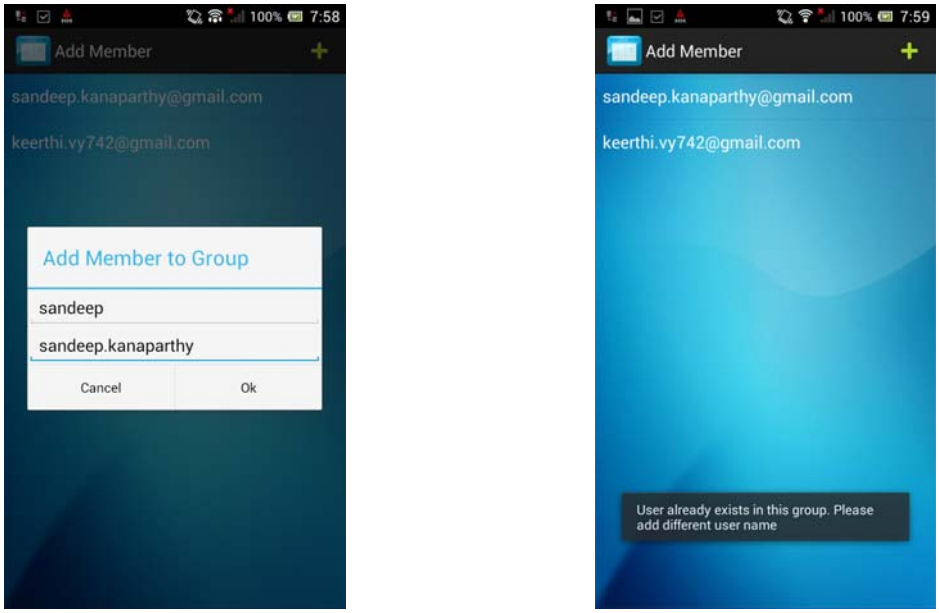
(d)

Figure 5.3: Testing for My Profile Screen

5.4 View Group

1. User can view the group
2. Delete the member
3. Add the member

The test cases here are user cannot delete his own account. In *view_group_members* screen the current user is disabled. Hence he/she cannot delete their account. Furthermore, he/she can't add himself to the group again which creates duplicate events in the Parse Web Server. If user tries to add himself to the group a toast is displayed as shown in figure 5.4(b)



a b

Figure 5.4: Testing for Group members

5.5 List Events

In order to see others' events, user has to invite them to the group. The test case here is user must not see his invitee events until he/she accepts the invitation. The app checks if the invitee accepted the invitation or not. In order to accept the invitation, he/she should sign in at least once. Otherwise, other users are not allowed to see his invitee events. This procedure is considered as accepting the invitation and giving others permission to view his/her events.

And the other test case here is that the user is not allowed to edit/delete others events. The app fetches the events from the Parse server and does not allow user to edit others events.

5.6 Merging and Unmerging events

As we discussed in section 4.4.4, similar events gets merged based on event title, location and overlap period of 30 minutes as shown in figure 4.15. If user changes the event details, the event gets unmerged.

5.7 Contacts Synchronization

The application synchronizes contact with the Google account present in the default calendar app on the device. To avoid blank screen and abrupt application termination a progress dialog box showing the contacts are getting synchronized is displayed as shown in figure 5.5

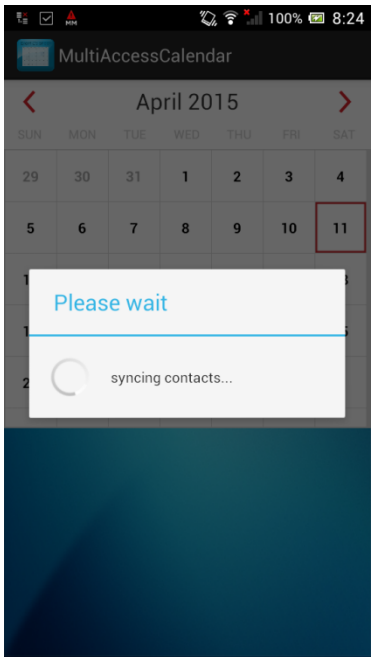


Figure 5.5: Testing for Contact Synchronization

5.8 Add/ Update and Delete data

User can add/ update and delete events which belong to the user only. The user is not allowed to modify the data of others events. The test case here is showing adding a new event, updating and deleting the existing events.

Figure 5.6 shows the test case for adding an event. User selects the “Add Event” option as shown in figure 5.6(a). The application launches *AddEvent* activity where user can add the event as shown in figure 5.6(b). However, if the user adds an event in which the starting time is greater than the ending time, a toast is displayed as shown in figure 5.6(c). When the user gives all the information about the event and presses the save button the events gets added as shown in figure 5.6(d)

Similarly, when the user wants to update an existing event, they should click on the event which they want to update as shown in figure 5.7(a). The application launches

EditEvent activity as shown in figure 5.7(b). As discussed in the above paragraph, the user cannot update the event with starting time greater than the ending time, if user tries to do so a toast is displayed as shown in figure 5.7(c). After updating an event, user should press the save button and the event details gets updated as shown in figure 5.7(d).

Deleting an event is similar to updating an event. Here, the user needs to select the event to be deleted as shown in figure 5.8(a), it navigates to the *EditEvent* activity as shown in figure 5.8(b). On pressing the delete event, the event gets deleted as shown in figure 5.8(c).

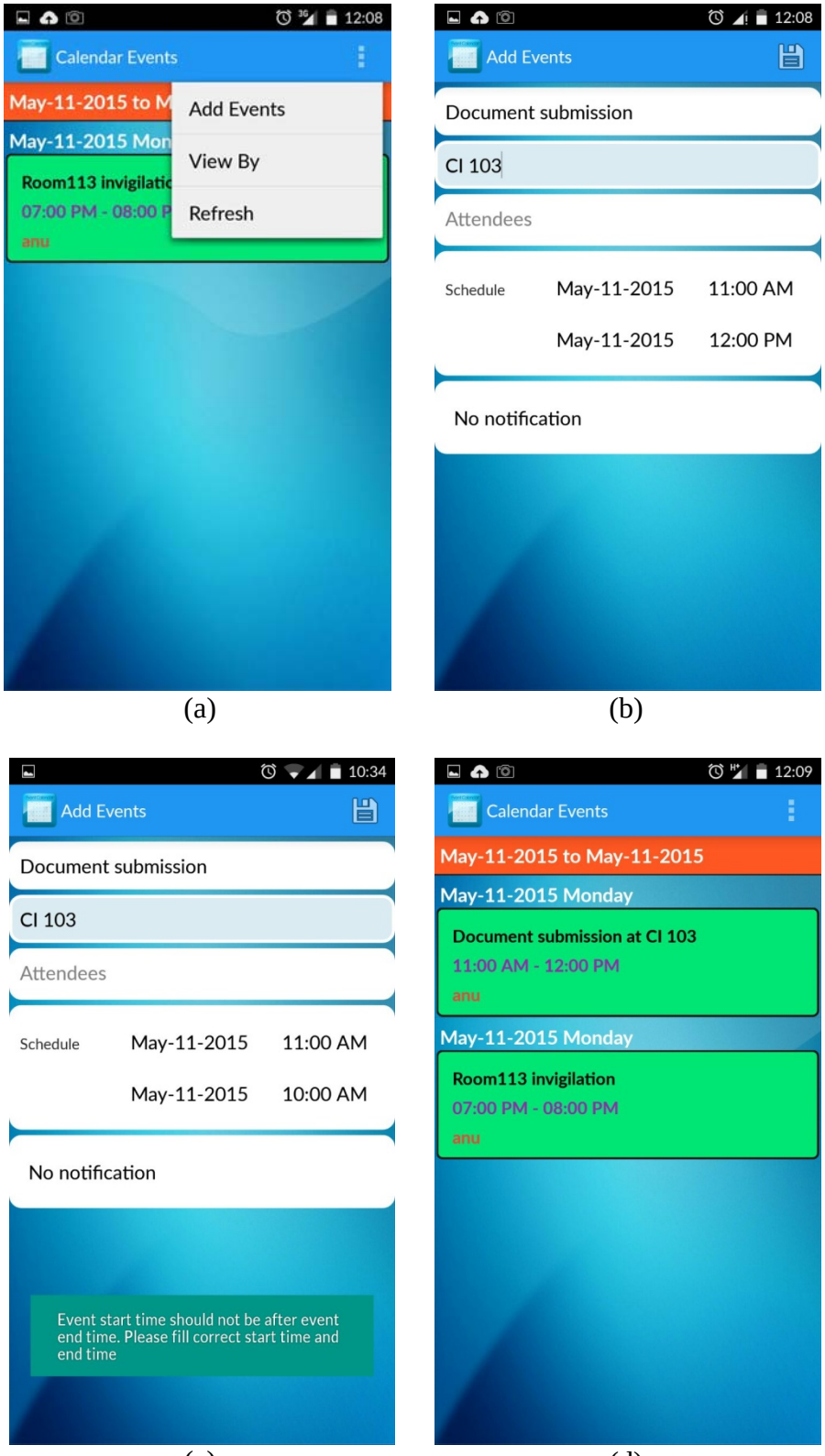


Figure 5.6 Testing for adding an event

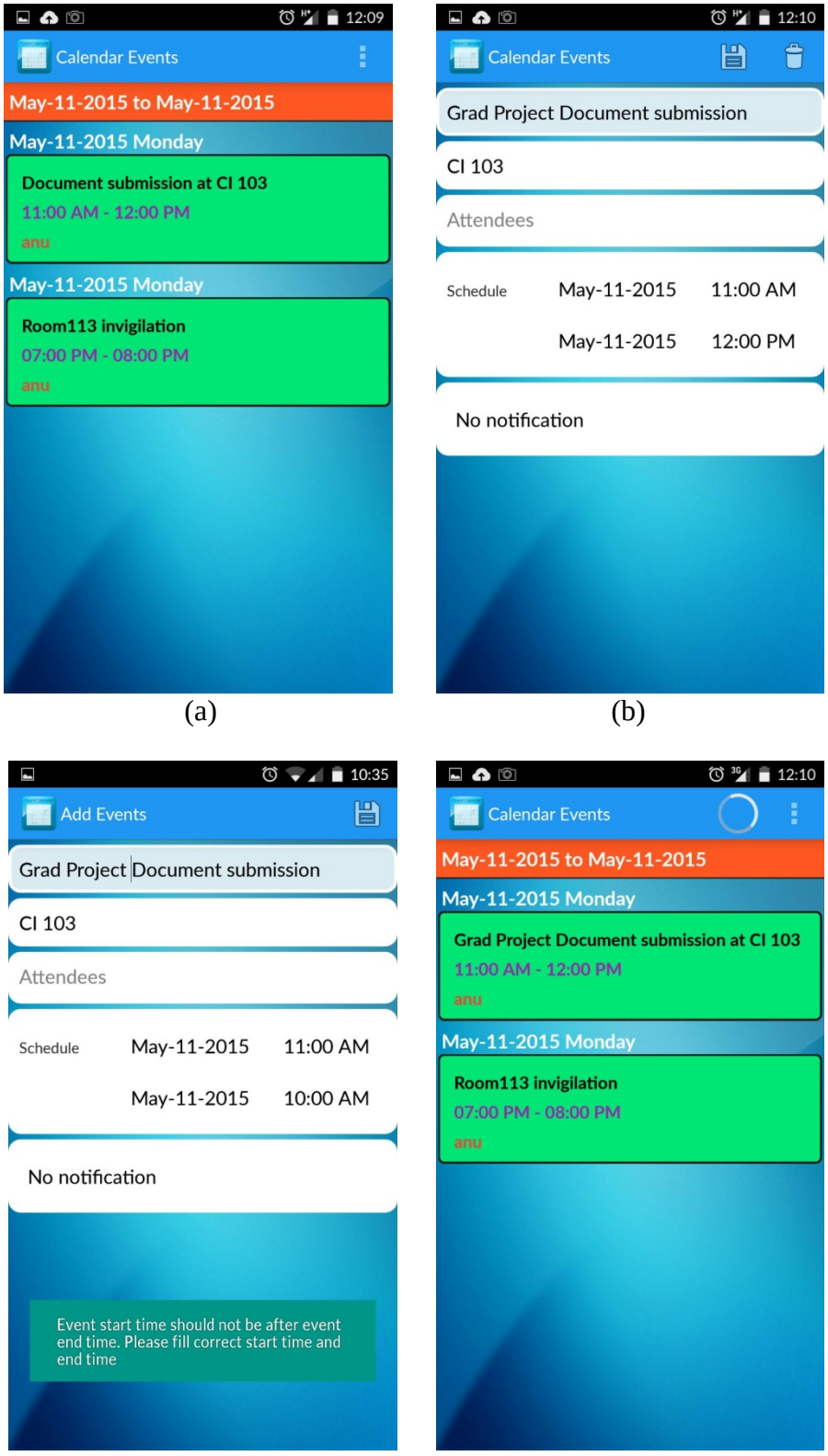


Figure 5.7 Test cases for updating an event

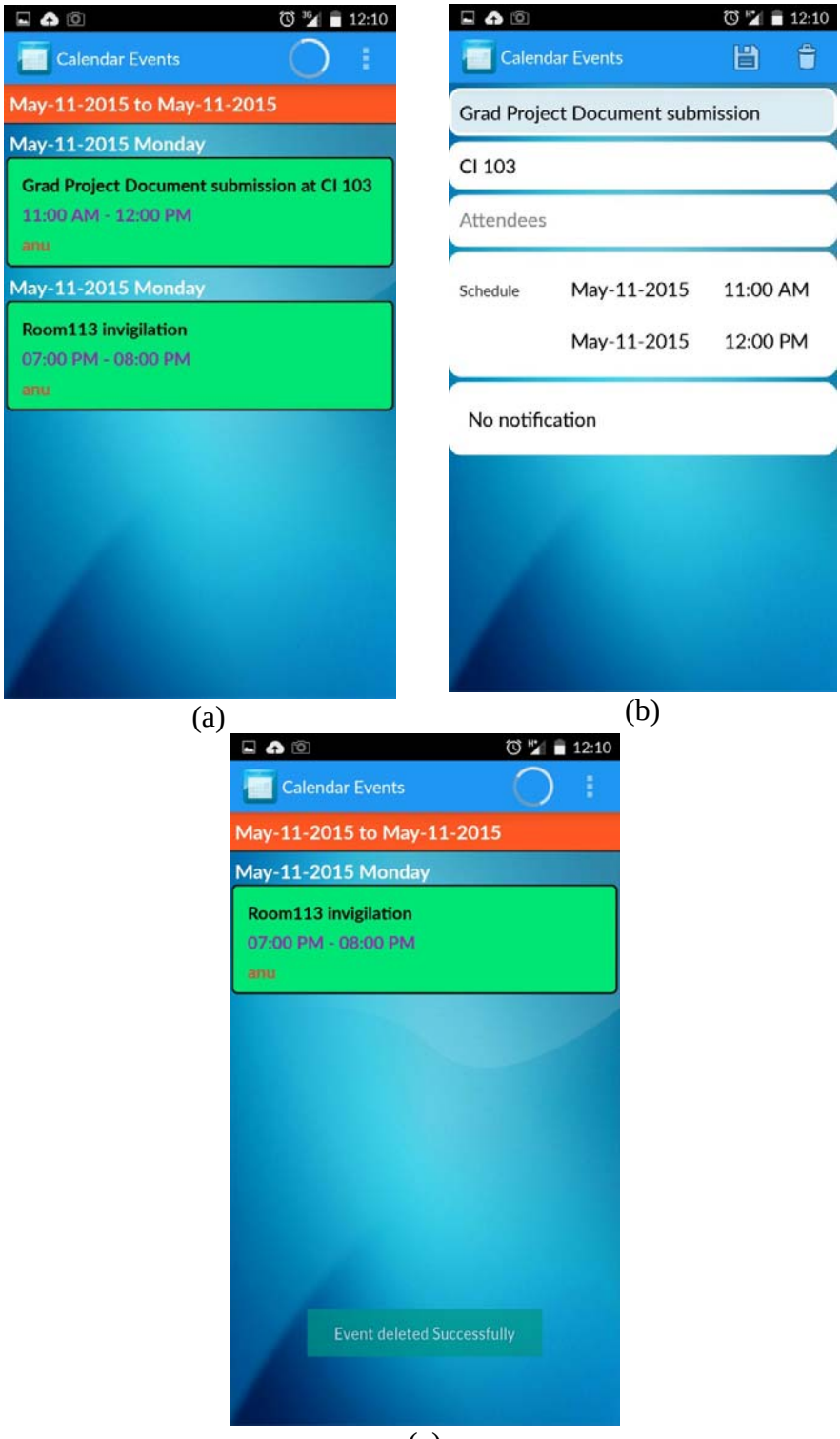


Figure 5.8 Test cases for deleting an event

5.9 Logout

The app does not allow the user from navigating to login page after he enters the dashboard. If user presses the back button on the home, the logout dialog box is displayed asking the user whether he/she wants to logout or cancel as shown in figure 5.9.

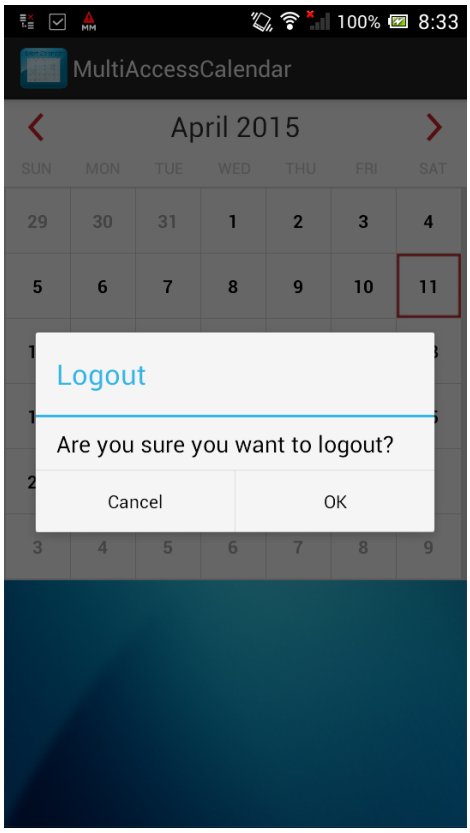


Figure 5.9: Testing for back navigation

6. CONCLUSION AND FUTURE WORK

Calendar+ application provides a shared password to access and store other users' event details which can be shared by email or by other forms of communication. This application stores all the users' events in Parse Web Server. SQLite is used as a backend to store the information about event and user details in the local database before adding to the Parse Web Server. There is no limit on the number of user accounts for accessing the calendar. User can add, update and delete event details in the application. Related events are merged/unmerged using an algorithm. Events can be seen in a customized view format like day/week/month and year. This application will automatically synchronize phone book contacts from mobile phone.

In the future, by using two-step verification more security can be provided for Calendar+ shared password which improves data security levels. Multiple accounts for a user can be implemented. On the other hand, merging/unmerging algorithm which is used to compare the events can use artificial intelligence and thus merging/unmerging can be done on more parameters. Additionally we can attract more users to use Calendar+ by providing better interface with advanced functionalities. More importantly, this app can be extended to fetch events not only from Google calendar but also from other available calendars as well.

7. BIBLIOGRAPHY AND REFERENCES

[1] AFP RELAXNEWS. (2013, March 22). *Smart phones on the verge of taking over the world. astro AWANI.* Retrieved from http://english.astroawani.com/lifestyle/smartphones-verge-taking-over-world-10503

[2] Pew Research Center. (2014, January). *Mobile technology fact sheet*. Retrieved from http://www.pewinternet.org/fact-sheets/mobile-technology-fact-sheet/

[3] Edwards, J. (2014, May 31). *The iPhone 6 had better be amazing and cheap, because Apple is losing the war to Android.* Retrieved from http://www.businessinsider.com/iphone-v-android-market-share-2014-5

[4] *Google Play*. (2015, May 5). Retrieved from http://en.wikipedia.org/wiki/Google_Play

[5] *Smartphone*. (2015, May 5). Retrieved from http://en.wikipedia.org/wiki/Smartphone

[6] *Sunrise Calendar*. (2015, April 14).Retrieved from http://en.wikipedia.org/wiki/Sunrise_Calendar

[7] *Android guide*. (n.d.). Retrieved from https://www.parse.com/docs/android/guide#objects

[8] Dao, T. (n.d.). *Caldroid*. Retrieved from https://github.com/roomorama/Caldroid