Integrated Group Project

NA3

Charlie Howes (CH), Lewis Allen (LA),
Adam Howes (AH), Ben Ashing (BA),
Constantinos Ioannou (CI)

March 13, 2017

Contents

1 Planning

1.1 Gantt Chart

1.1.1 Proposed

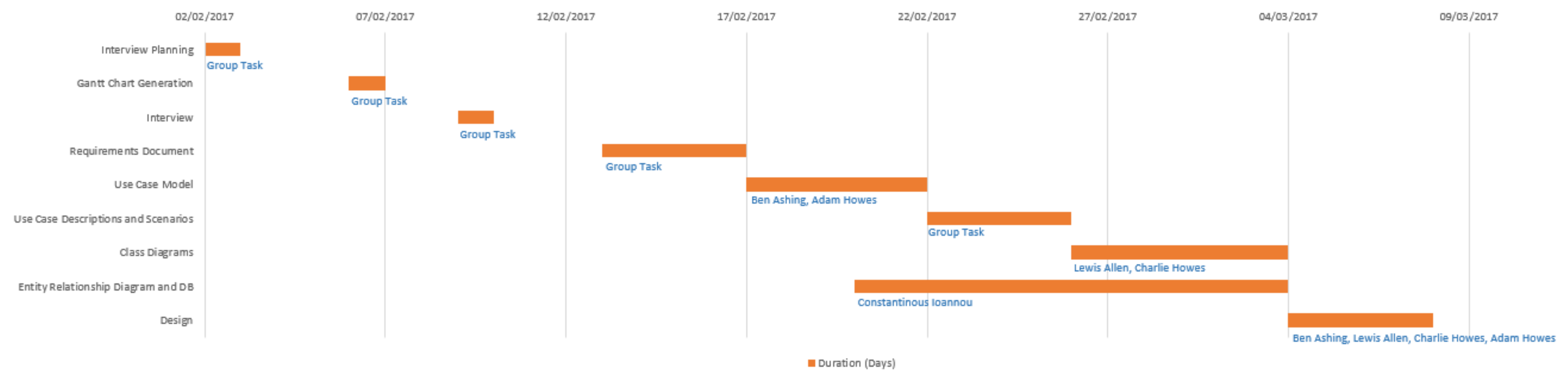

Figure 1: Proposed Gantt Chart

1.1.2 Actual

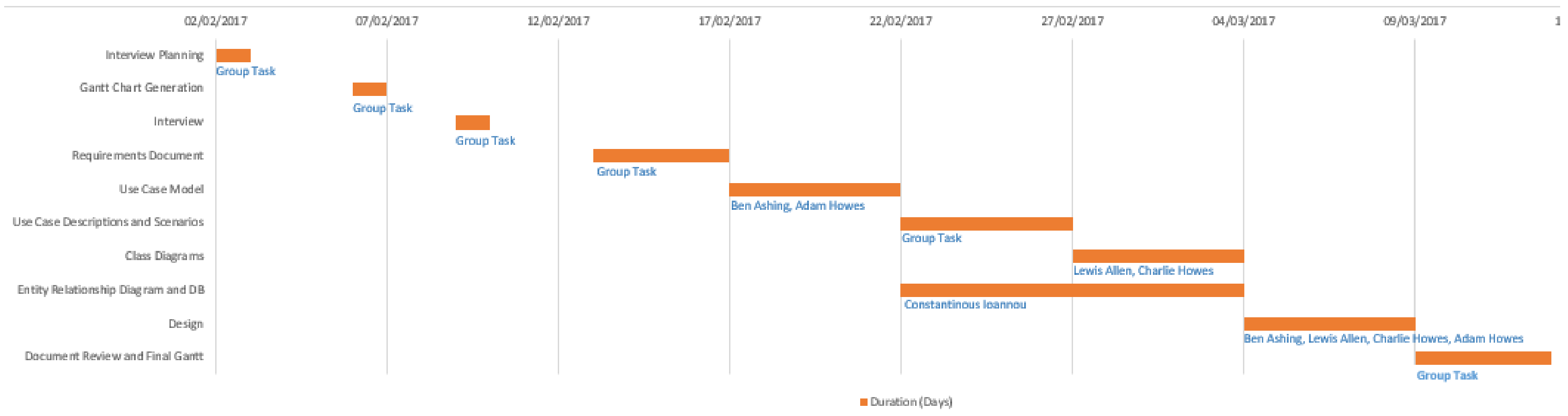

Figure 2: Actual Gantt Chart

1.2 Minutes

2 Requirements

2.1 Document

1. Business Requirements
   - B1. The Planning should be completed by March 13th
   - B2. The finished product should be delivered in May.
   - B3. The software will be adopted and used by all staff.
2. User Requirements
   - U1. Users must be able to log in using a user name and password.
   - U2. Users must be able to log out
   - U3. The user must be able to personalize their view.
   - U4. The user must able to easily view calendars for daily, monthly and yearly schedules.
   - U5. The user must be able to set recurring appointments.
   - U6. Staff must be able to create groups.
   - U7. Staff must be able to modify groups
   - U8. Staff must be able to add other staff/groups to events.
   - U9. The user must be able to cancel appointments at any time within a session.
   - U10. The user must be able to use a search feature to find other users.
   - U11. The user should be able to make event requests.
     - U11.1. Users should be able to receive event requests.
     - U11.2. Users should be able to accept event requests.
     - U11.3. Users should be able to decline event requests.
3. Quality Requirements
   - Q1. The application must use symbols to clearly show changes in events.
   - Q2. The application should notify the user in any changes to events such as cancellations.
   - Q3. The application must implement optimized loading times.
4. Functional Requirements
   - F1. Only members of staff should be assigned accounts.
   - F2. The architecture of the project should support different platforms.
   - F3. A method must exist which allows staff to be given administration rights to form an administrative team.
   - F4. Administrators must be able to verify accounts.
5. Non-Functional Requirements
   - NF1. The user should be able to complete any single task with a minimum of eight actions. (LA)
   - NF2. The code needs to be easily maintainable by keeping the code organized, well written, documented and simple. (CH)
   - NF3. The project must be easily scaled to implement additional features and a larger user base. (CH)

NF4. The program must be executable on the following Operating Systems: (AH)

- Windows 8, 8.1 & 10
- Mac OS
- Linux Ubuntu

NF5. The minimum specifications to run the program should be: (AH)

- Processor: Pentium 4
- RAM: 500MB
- Disk Space: 100MB

NF6. The code must also be able to run on both 32 and 64-bit Operating Systems and be able to view events when not connected to the Internet. (AH)

NF7. All personal data must be fully secure through encryption and hashing. (BA)

NF8. The project must be adaptable in the future for mobile implementations. (CI)

2.2 Stakeholder Diagram

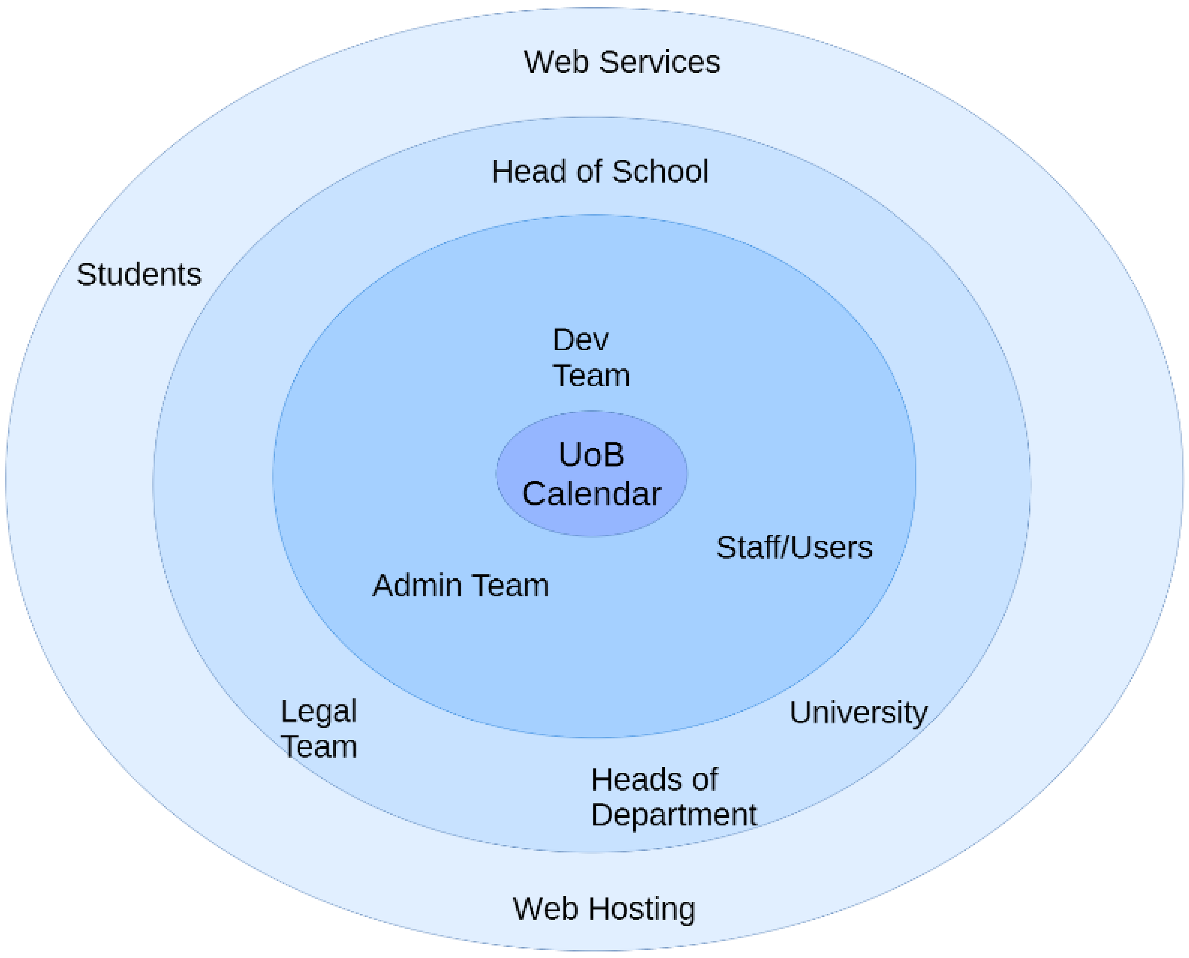

Figure 3: Stakeholder Diagram

3 Use Case Model

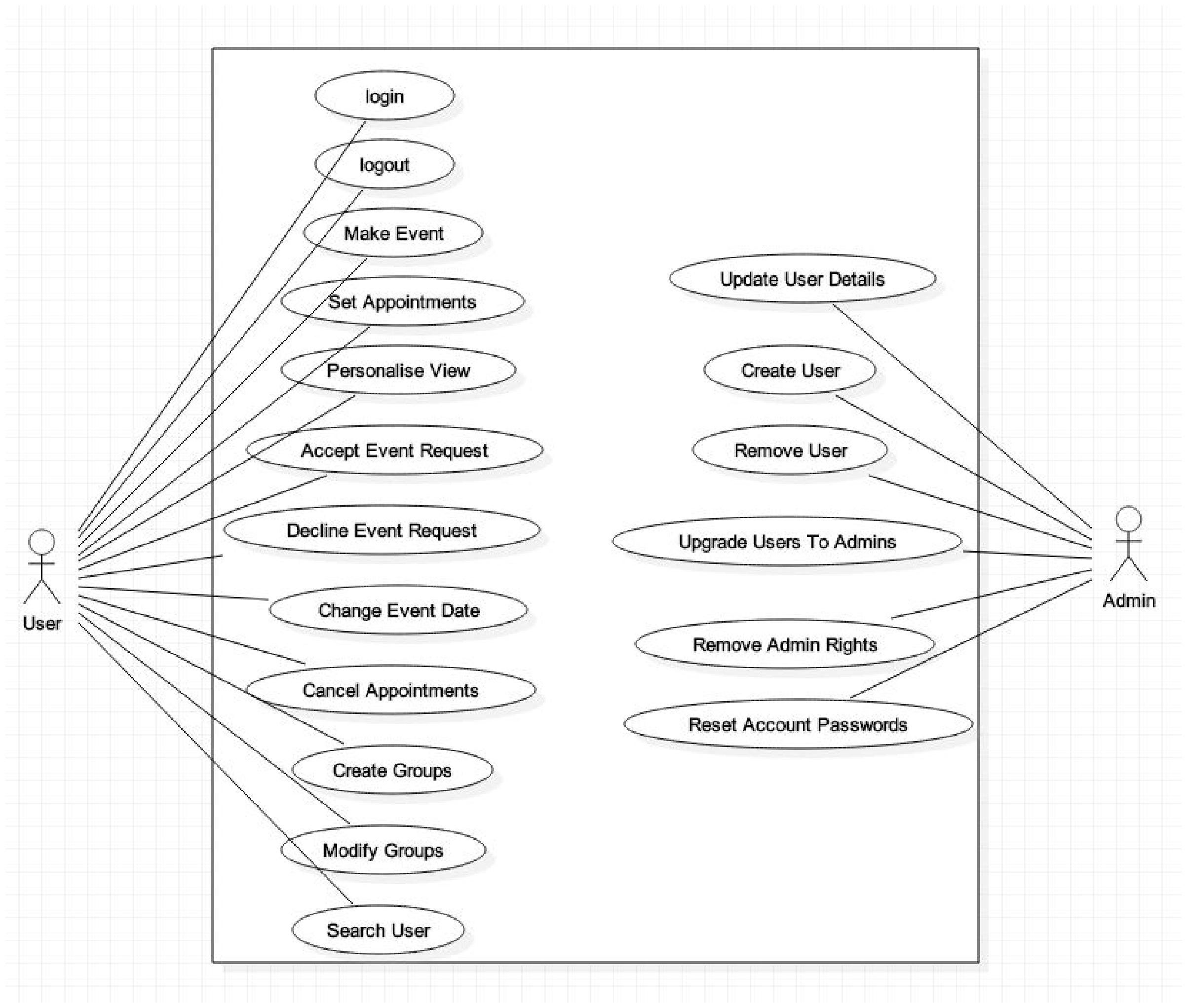

Figure 4: Use Case Diagram

3.1 Case Descriptions

3.1.1 Login - CI

Use case: Login

Actors: User (Primary)

Precondition: Time, Date & Location from customer (via phone).

Success postcondition: The user fills in their login credentials successfuly and logs into the system.

Failure postcondition: The user does not fill the correct login credentials.

Trigger: The personal calendar of the user is displayed

Main success scenario:

1. User wants to login
2. User inserts credentials into the system
3. User clicks the login button
4. System checks the input credentials.

5. System fetches information and displays their personal calendar

Extensions:

3a User entered invalid data

   1. System asks the user to input correct data
   2. Restart from 2

3.1.2 Creating an Event - LA

Use case: Creating an Event

Actors: User (Primary)

Precondition: The User has logged into their account.

Success postcondition: The User has the created event displayed on their calendar.

Failure postcondition: The User was unable to create the event.

Trigger: The User clicks the 'add event' button.

Main success scenario:

1. The user clicks the 'add event' button.
2. The User specifies the time, day and duration of the event.
3. The user names the event and gives it a description.
4. The user clicks 'Confirm'.
5. The System adds the event to the user's calendar.

Extensions:

2a The user has input an event time/date and there is a conflicting schedule. (The user already has an event booked for that duration)

   1. The System indicates that the user can no longer click confirm.
   2. The System informs the user that there is already an event planned for that duration.
   3. The user is returned to the event creation screen and is given the option to change to event time/day, along with the other details if required.
   4. The user modifies the time/day to a free period.
   5. Continue on to step 3.

3.1.3 Edit Event - BA

Use case: Editing the Details of Events

Actors: User (Primary)

Precondition: The user is logged in and has an event to edit.

Success postcondition: The User has successfully edited the details of their event.

Failure postcondition: The User is unable to edit the details of their event

Trigger: The User clicks on the event edit button.

Main success scenario:

1. The User will navigate to the event in the calendar
2. The User will select the event they wish to edit

3. The User will click on the edit button
4. The User will be able to change any detail of the even
5. The User will click on the save button to save any changes

Extensions:

- 3a The User is not the host of the event
    1. An edit button won't be displayed to the user if they don't have the right change it
- 4a The User makes an illegal change to the event e.g. changing the date to a date in the past
    1. The User is notified that the changes made cannot be saved
    2. The User Changes the data so that it is valid
    3. Proceed to step 5
- 5a The User closes the calendar without saving the changes that have been made
    1. The User is notified that changes won't be saved
        - a
            1. The user clicks cancel
            2. Proceed to step 5
        - b
            1. The user ignores the notification and navigates away from the page
            2. Failure postcondition is met

3.1.4 Creating a Group - CH

Use case: Creating a Group

Actors: User (Primary)

Precondition: The User has logged into their account.

Success postcondition: The User creates a group to use in events.

Failure postcondition: The User was unable to create a group

Trigger: The User clicks the "Add Group" button.

Main success scenario:

1. The User inputs the name of the group
2. The User inputs the description of the group
3. The User clicks on the "Add Member" button and adds a member.
4. The User clicks "Create".
5. The System adds the group to the user.

Extensions:

- 1a The user didn't input the name of the group
    1. The program disabled the "Create" button
    2. The user is informed that they are required to input a name.
        - a The user clicks cancel
            1. The program goes back to the previous screen.
            2. The failure postcondition is met.
        - b The user inputs the name
            1. The program enables the "Create" button

        2. Continue from step 2.

4a The user clicks "Cancel"

   1. The program goes back to the previous screen.
   2. The failure postcondition is met

4 Class Diagram

4.1 Diagram

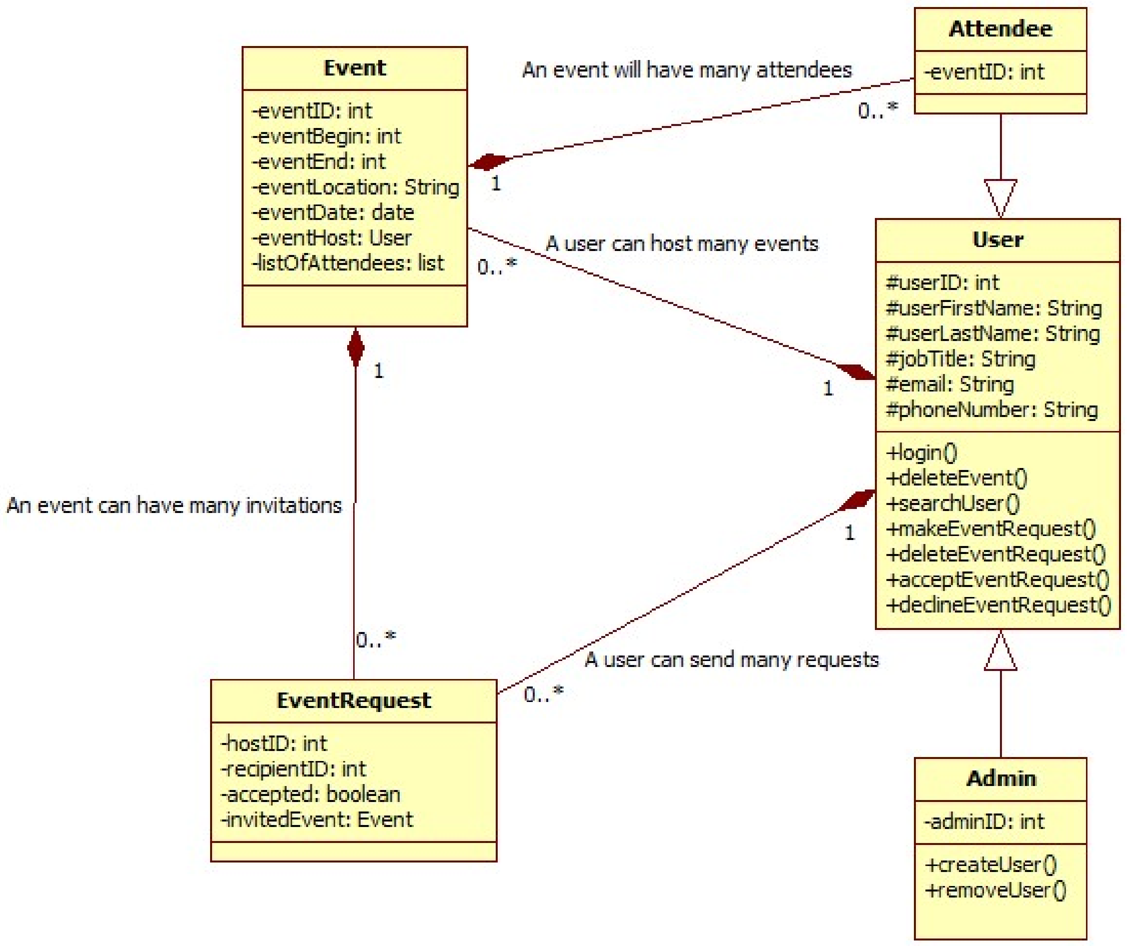

Figure 5: Class Diagram

4.1.1 Notes

The Classes The class diagram consists of four main classes:

- **User:** The user will do the majority of the operations involving the other classes and will allow for event creation and deletion as well as sending, receiving and accepting invitations. The variables inside this class are protected as the class Admin will inherit from this class and an admin must have all the functionality of user and more.
- **Attendee:** A problem we faced was distinguishing between a host and an attendee when creating events. We solved this by having two separate classes to be able to tell between hosts and attendees. An attendee is a user who is attending an event, and contains the event ID of the event to be attended.
- **Event:** The event class will store the details for a single event. Information such as dates, times and locations will be stored in this class through the use of the database.
- **EventRequest:** The Event Request class allows the User class to send out requests specifying the event being invited to as well as the recipient to the request.

- **Admin:** The administrator Class is a user with special privilages which allow for the creation and removal of new and existing accounts. As the Admin class will extend the User class, it will have all the functionality of a User plus more. This authorization will only be given to a select few.

The Relationships

- **User - Event:** The relationship between these classes is composition, as an event can no longer exist without the host who created it. The multiplicity is one-to-many, as the user can create many events.
- **Attendee - Event:** The relationship between attendee and event is to allow the class to distinguish between who is the host of an event, and who is simply an attendee. This is to ensure events are destroyed when host accounts are deleted, but not if attendee accounts are deleted.
- **Attendee - User:** An attendee is a user who is attending an event. The attendee has all the attributes of a user but also contains the event ID for the event he/she is attending.
- **User - EventRequest:** The relationship between these classes is aggregation, as even if the user who send the request is deleted, the request can still exist as it links directly to the event. The multiplicity is one-to-many as a user can send many requests.
- **Event - EventRequest:** The relationship between these classes is composition, as if the event a request points to is removed, any requests relating to it must also be removed. The multiplicity is one-to-many as an event can have many event requests.
- **User - Admin:** The relationship between these classes is generalisation, as the class Admin inherits from the class User. This is to ensure the class Admin has all the functionality of a User class aswell as further behaviour such as being able to create accounts.

5 Database

5.1 Documentation

5.1.1 Entities

Event:
This entity will store all the information about the events. The unique ID for the entity is the *Event_ID* and it will be used to determine an event. The entity can hold all the necessary information such as date duration and location. The event is created by a user so the *Host_ID* is used to identify the user that create the event. The option to keep the event private is also available.

Event Request:
The Event Request entity is used store the information about an event request. The unique ID is the foreign key from the Event entity. It stores the response of user to the request as well as date invited and if the user seen the request or not.

User:
All the data of a user are store in the User entity. The unique *User_ID* is used as a primary and is used to log in in to the system with the required password. The basic personal data of the user are also store such as first name, last name, position, email and phone number. An Admin attribute is used to identify if a user is an admin or a normal user.

Attendee:
The attendee entity holds the information about the attendance of the event. The combination of the two attributes *User_ID* and *Event_ID* will give a combine key for the table.

5.1.2 Tables

Table 1: Event

Attributes	PK/FK	Data Type	Constraints
Event_ID	PK	VarChar(10)	Unique, Not Null
User_ID	FK	VarChar(10)	Not Null
Host_ID		VarChar(10)	Not Null
Event_Description		VarChar(30)	Not Null
Event_Start_Date		Date	Not Null
Event_Duration		Integer	
Room_No		VarChar(10)	
Location		VarChar(15)	
Private		Bit	
Comment		VarChar(100)	

Table 2: Attendee

Attributes	PK/FK	Data Type	Constraints
Event_ID	FK	VarChar(10)	Unique, Not Null
User_ID	FK	VarChar(10)	Not Null

Table 3: User

Attributes	PK/FK	Data Type	Constraints
User_ID	PK	VarChar(10)	Unique, Not Null
Admin		Bit	Not Null
Password		VarChar(20)	Not Null
First_Name		VarChar(15)	Not Null
Last_Name		VarChar(15)	Not Null
Position		VarChar(30)	
Email		VarChar(30)	
Phone_Number		VarChar(20)	Not Null

Table 4: Event Request

Attributes	PK/FK	Data Type	Constraints
Event_ID	FK	VarChar(10)	Unique, Not Null
Response		Bit	Not Null
Seen		Bit	Not Null
Date_Invited		Date	Not Null

5.2 Entity Relationship Diagram

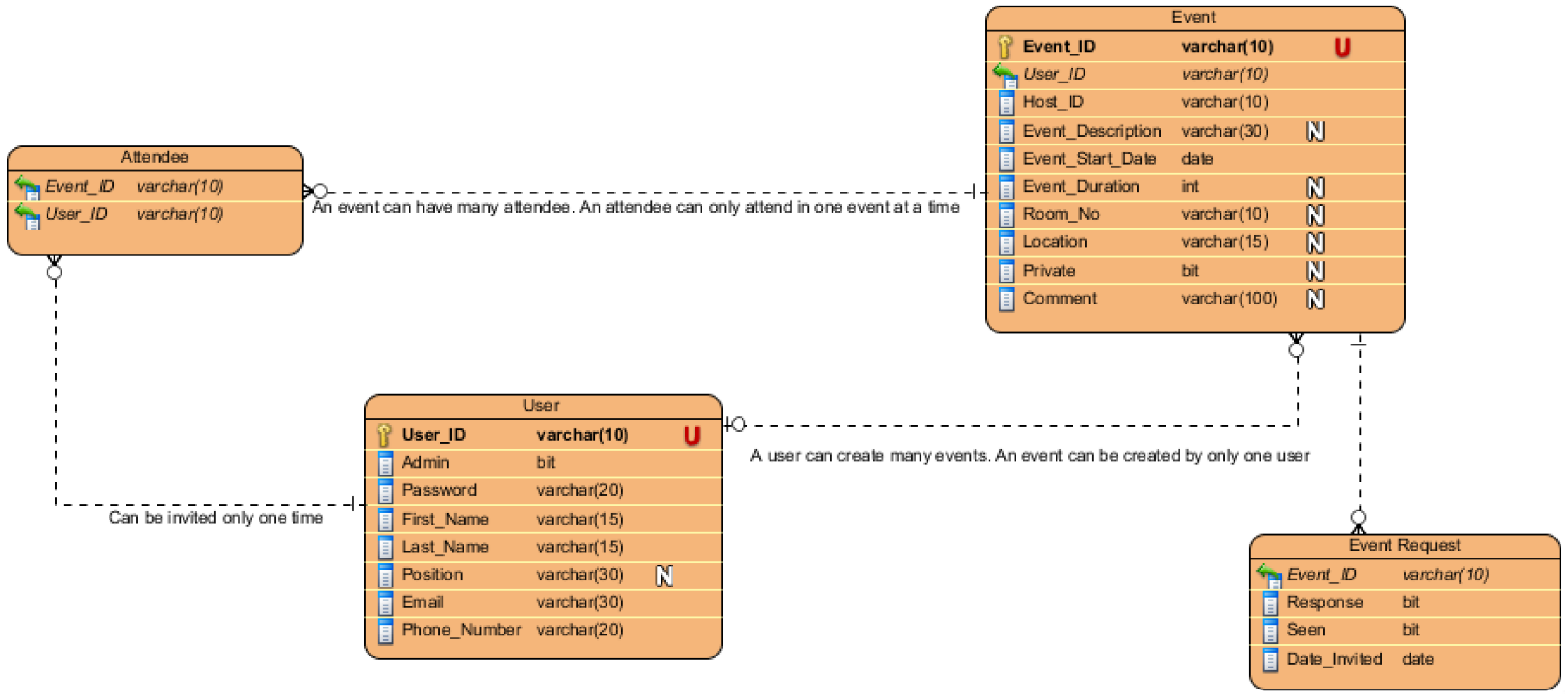

Figure 6: Entity Relationship Diagram

6 Design

6.1 High-Level Architecture Diagram

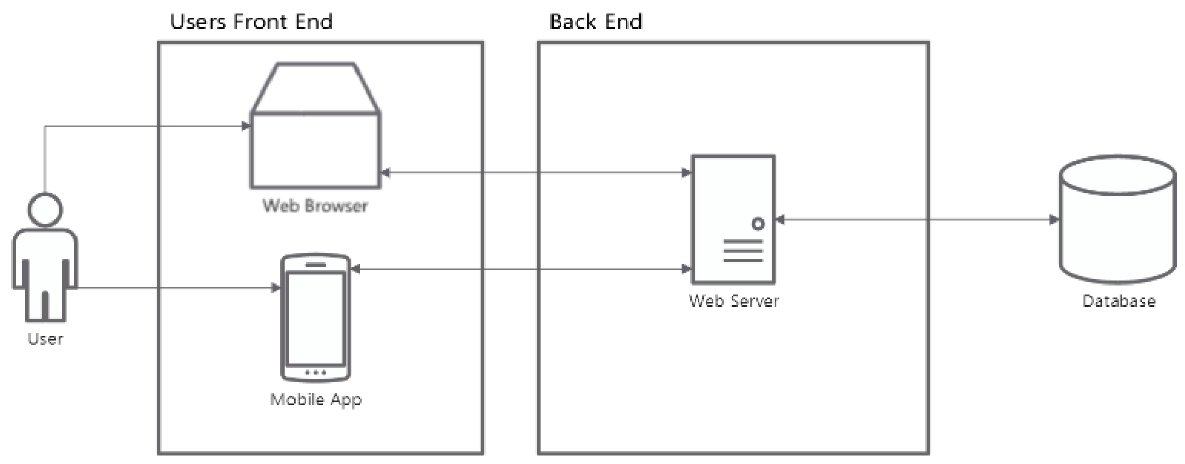

Figure 7: High-Level Architecture Diagram

6.2 Model View Controller

6.2.1 Design

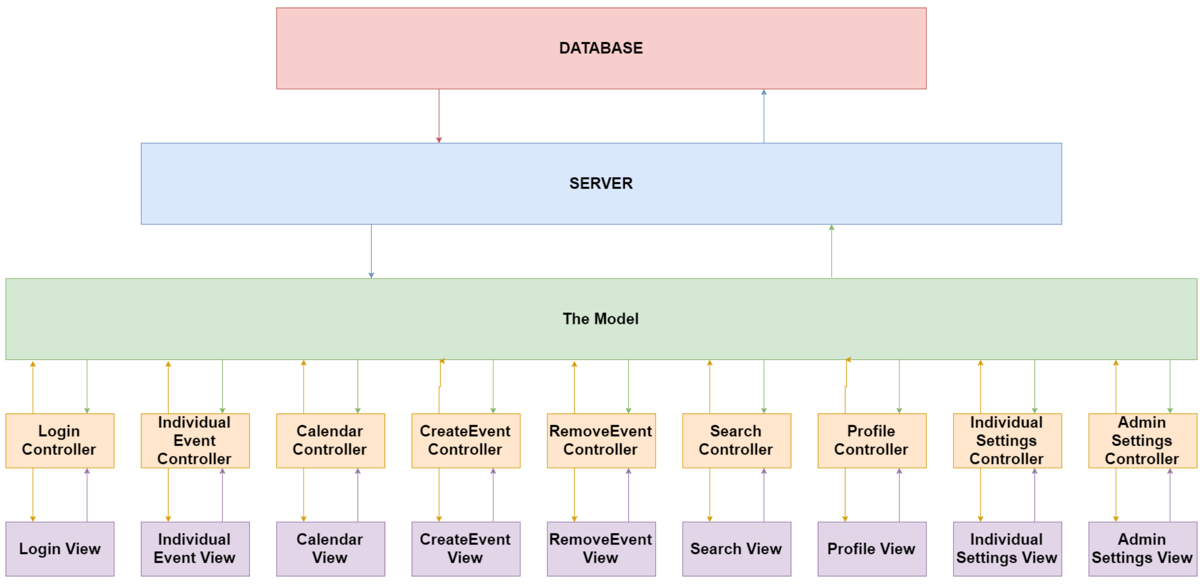

Figure 8: MVC Diagram

Diagram Notes The Model-View-Controller design application is made up of several main aspects. It consists of:

- **The View Classes**
  Every view in the application will have its own View class. This class handles the updating and gathering of information through the user interface.

- **The Controller Classes**
  Each respective View has its own Controller. This controller class handles communicating

the information from the View to the Model class.

- **The Model Class**
  The Model class is where all the respective information will be stored and all logical calculations will take place. This model is shared amongst all controller classes, but its possible that this large Model class will be split up into smaller individual model classes.

- **The Server**
  The Server will handle communications between the database and the Model class.

- **The Database**
  The database is where all the data in the application will be stored using appropriate encryption and hashing depending on the confidentiality of the data stored.

6.2.2 Explanation, Rationale, Benefits and Disadvantages

The Model-View-Controller (MVC) is a design pattern which focuses on providing a versatile user interface. It separates internal representations of information from the parts of the program which present this information to the user. This section will describe our rationale behind the decision to use the MVC. It will also outline how the use of the MVC design pattern within our application will offer benefits not only to the user, but also to the developers whilst creating the application. Despite this however, its usage also comes with a few disadvantages. These will also be outlined and discussed below.

Rationale
The rationale for the use of the MVC pattern within our application is formed from our analysis of the requirements. We needed a design pattern that is both versatile and maintainable whilst remaining uncomplicated to allow other development teams to extend the project in the future. The project needs to be able to perform the basic tasks of a calendar system such as modifying views and updating data. The Model-View-Controller pattern provides the tools to do this in an efficient manner through its relatively simple framework.

Benefits
The MVC design pattern comes with a variety of benefits, such as:

- **Separation of Logic and View**
  The separate Model and View classes will allow the code for the logic to be separated from the code for the views. This is useful as it promotes good code organisation, making it easier to pinpoint where problems occur and as a result bugs easier to fix. It also allows new features to be implemented more efficiently as well as existing features to be modified effectively.

- **Simultaneous Views**
  The nature of the MVC's specified encapsulation allows the program to display multiple views based off the same information. This will be useful when we implement the different types of calendars such as the Daily, Monthly and Yearly views.

- **Parallel Development**
  The loose coupling provided by the MVC pattern will allow separate members of the development team to be working on different aspects of the application at the same time. For example, One member could be working on the logic within the model whilst another on the physical representation of the view to the user. This is an advantage when considering both development speed and team-member co-operation.

- **Strong Cohesion**
  Whilst the code regarding different aspects of the application will remain separate in different classes, the pattern relies on relationships between these classes, giving the application an

essence of strong cohesion. An example of this is how the controller will communicate between the model and the view to perform actions.

Disadvantages
Whilst the MVC pattern comes with a variety of benefits, it also has several disadvantages. These include:

- **Keeping things Consistent**
  As we will be splitting code in to several different classes instead of keeping it form within one class, it will require the maintaining of several classes at once. This can be dangerous as forgetting to update one of these classes could cause bugs and other unforeseen problems. This can be avoided by careful planning and implementation.

- **Complexity of Framework and Navigation within Code**
  Whilst the MVC provides good organisation and encapsulation, it will require us to adapt the planning steps we made during the planning phase to function correctly with the MVC. This adds new layers of abstraction to the process, making it more complex and difficult to navigate.

7 Appendix

7.1 Scenarios

Scenario

By Constantinos Ioannou

Persona

Name	Stavros Poseidias	
Age	26	
Occupation	PHD in Computing at the University of Brighton	
Place of Residence	Brighton	
Family Members	Wife	
Technical Skills	An enthusiastic computer scientist with good knowledge of programming language. Good communication and problem solving skills. Patience and multi-tasking.	
Personality	• Good Learner • Highly motivated • Very organized person • Hard working person.	
What does the he need from the System?	• Wants to include personal details of the users so that he can easily communicate with other users. • A mobile version should be available so that he accesses the calendar at any given time. • Able to see the calendar of other user. (if it is not private) • Assign fixed events on his calendar. • The system must be able to search for certain events or persons	

Scenario

Stavros is doing his PHD at University of Brighton. He is responsible to give lectures to second year students. He has a really busy schedule because of the PHD studies and the preparation of the lectures and the course work for the students that he is responsible. Stavros uses his smartphone in order to set reminders and meeting. Every month he assigns the fixed events of the month in his personal diary. During the week he set meeting with the course tutor by sharing their calendars to find an empty slot to set the meeting.

During his free time Stavros, he enjoys playing football in a local football team. Every day he spent time to go to the gym where some of his workout classes have fixed time that he has to attend. Due to his busy schedule he would really like if the application sends him notifications about upcoming events so that he can easily be organised. As this application is something really personal and is going to be used every day, he would really appreciate if he is able to personalise his calendar.

Scenario

By Lewis Allen

Persona

Name	Alex Barker	
Age	32	
Occupation	Senior Lecturer in Computing at the University of Brighton	
Place of Residence	Hove	
Family Members	Wife and Daughter	
Technical Skills	Experienced with Computing. Skilled Communicator. Good level of managerial skills – manages several staff. Regularly delivers presentations. Multi-level communicator. Logistical planning.	
Personality	• Highly Organised • Very Attentive to Detail • Enjoys Efficiency • Hates disorganisation. • Regularly plans the day ahead of time. • Conscientious. Wants to do a good job.	
What does the he need from the System?	• Efficient and fast viewing of schedule. • Easily modifiable diaries for changes to his day. • Would like to see schedules of other staff in order to organise meetings. • Easy to add staff to events. • Clear contact details of people he wants to communicate with. • Reminders of events. • Notifications to changes.	

Scenario

Alex Barker wakes up at 7:00am and gets ready for work. He then proceeds to log in to the calendar application on his computer in order to check his schedule for the day. After seeing a free period in his day, Alex decides that he would like to organise a meeting with several of his staff to check the progress on a recent project. He quickly navigates to each members profile and takes a look at their contact details. Alex checks each staff members schedule in order to decide for a time. He creates an event for his meeting and quickly sends a meeting request to each of his staff members.

The majority his staff accept the event request. Unfortunately, halfway through the work day one staff member rejects the request due to an illness. The application sends Alex a notification regarding this through his email as he has this option turned on in his settings. This staff member is critical to the meeting so Alex modifies the event and makes the meeting take place during the next week. The application notifies each member of staff of this change in time.

During the next week all members of staff manage to accept the event request. The application notifies Alex and his staff an hour before the meeting occurs and reminds them of the location where the meeting will occur. Alex and his staff then attend the meeting at the designated time.

Scenario

By Ben Ashing

Persona

Name	Charles Xavier	
Age	35	
Occupation	Lecturer in Psychology at the University of Brighton	
Place of Residence	Kemptown	
Family Members	None	
Technical Skills	• Can read peoples thoughts • Good organisational skills • Fluent programmer in many languages • Expert in many scientific fields	
Personality	• Strives to serve the greater good • Diligent and well-motivated	
What does the he need from the System?	• Needs to be able to keep a track of his deadlines and meetings • Needs to be able to invite to events and meetings • Needs to receive notifications by email	

Scenario

Charles wakes up around 6am and switches on his pc before taking a shower. After making some breakfast he sits down at his machine and logs into his calendar to check his commitments for the day. Spotting a clash in his timetable, he edits one of his reminders and pushes it back an hour he has free. He then logs off and heads to work.

Whilst on his way to work he receives an email notification from a colleague inviting him to a meeting later in the week.

When Charles arrives at work he logs on to his computer and opens his calendar. He then accepts the meeting invitation he received earlier, adding it to his calendar. He then goes to give his lectures. Throughout the day he receives reminders for the events he has coming allowing him to easily keep track of his day without having to keep logging in to his computer.

7.2 Screen Designs

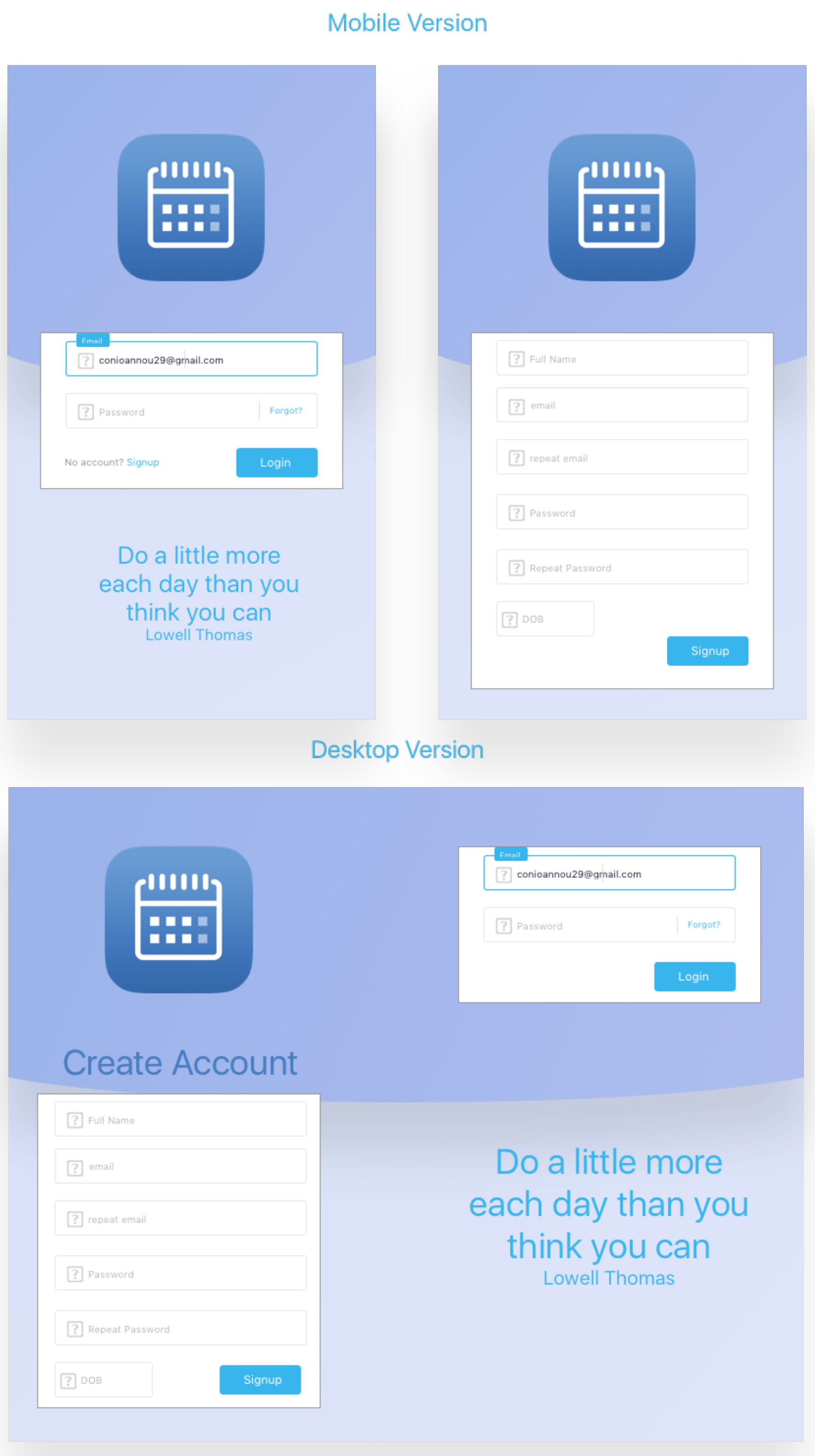

Figure 9: Login - CI

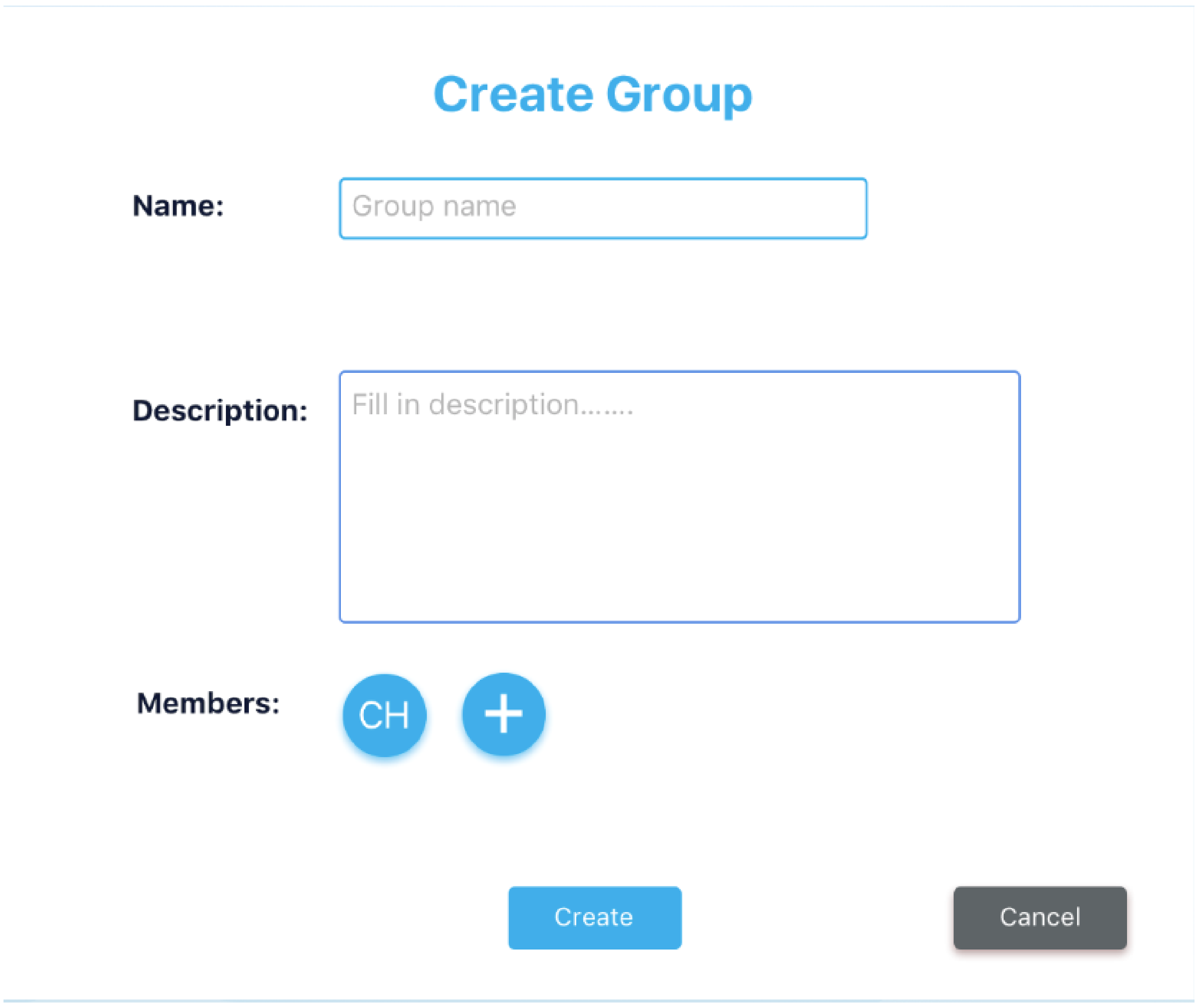

Figure 10: Create Group - CH