# Team Chennai Project Report

Mark Willett, Karolina Pakenaite, Natalie McDonnell,Tomaz Deregowski, Swetha Charles

# Project Description:

## Outline

We aim to create a distributed calendaring system, designed for use within organisations in order to create and manage meetings.

Users of the system will have a personal account with their own login details, and will be able to view global meetings shared to all users and personal meetings.

A key requirement of the system is to provide all users with a synchronised and homogeneous view of the meetings stored in calendars.

User and appointment information will be stored in a PostgreSQL database, and the user will interact with the system through a Java-based client.

## The Interface

Every user will have a profile of basic identifying information that they will be able to view and edit their details.

The users of the system will be able to view calendars via one primary interfaces, the list view and the day view.

The list view will present the user with a written description of the meetings that have booked in the upcoming days. The day view will allow the users to view their day as a timeline, giving a more focussed top down view of a particular day's schedule.

## Possible extensions

This project allows for a variety of possible extensions to functionality to enhance the user experience. These include, but aren't limited to:

- Monthly view, the ability to see a user's calendar from a monthly perspective.
- Allow users to have contacts so users can share calendars only to specific users.
- Create meetings which can include these contacts.
- Private meetings. Meetings that will show on your calendar marked as *Personal*, but no information other than its timing will be available to any other users (even if they have shared with another user). This will allow users to use their calendar for personal scheduling information, meaning that others will have a more complete picture of their schedule without having access to personal information.

- Meeting postings. Each meeting could have a forum style comment wall that would allow for the discussion of the meeting before or after the event.
- Accept or decline a meeting with an accompanying message. An inbox could be implemented that means that as users accept or decline meetings there are able to establish discourse between themselves and the meeting creator.

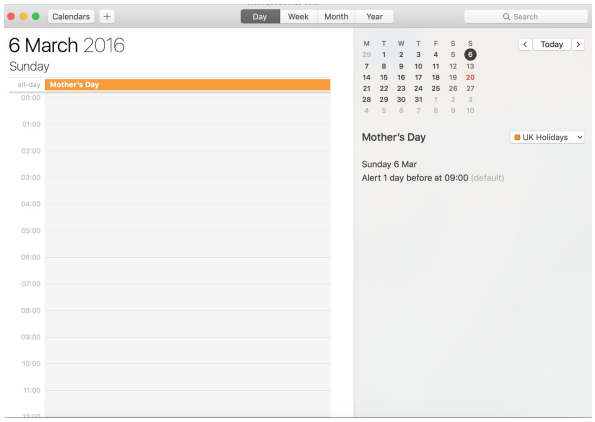
## Project Development:

### UI

To create a successful UI it was essential to discover the functionalities the user will need while also keeping it simple to understand how to use.
Many of the functionalities include:

- Be able to create an account,
- Be able to log in,
- Be able to create events and have these displayed clearly,
- Be able to edit and delete events,
- Be able to create, edit and delete global events for every user to see,
- Be able to view user details,
- Be able to edit user details including password,
- Be able to logout.

To understand better about successful calendar applications we looked at google calendar and apple calendar. Both of these contained a day view which displayed results in a list style which we believed would be the most achievable and clear way to display our events. Their calendars also contained a month view which we thought would also be useful, however it would be something to leave until after the list view was implemented. Unfortunately we did not have time to implement this idea.




Apple calendar on Mac

We also found that the essential information the user would want for their events are the name, date, start time, end time, location and a description. To be able to create an event the user would need to fill in all the information except the location or notes.

The appearance of the UI has been kept simple to make it easy for the user to view and input data. For example we have used pop up frames to create since we believed for our use it was ideal as it is clearer for the user and grabs their attention to the task at hand. We used the colours grey, white and red since they make everything stand out and be clear to the user.

**Uses Cases**

The following detailed use cases are within our design.

| | |
|---|---|
| Sign up to the service | On the main screen select the ‘Sign up’ button<br>Fill out data<br>If the data is not filled in correctly warnings will appear<br>Select submit button<br>Pop should appear to show successful sign up |
| Login to the service and create an event | On main screen fill in login details and select ‘Login’<br>On the home screen select the ‘+’ button<br>Fill in event details out in pop up window<br>Select ‘submit’<br>Any invalid details will show warnings<br>Pop should appear to show successful event creation |
| Edit an event | On the home screen select the ‘Edit event’ from the event wanting to be changed<br>Pop up window should show event details<br>Change any details<br>Select ‘submit’<br>Any invalid details will show warnings<br>Pop should appear to show successful event edit |
| Delete an event | On the home screen select the ‘Delete event’ from the event wanting to be changed<br>Pop up window should show confirmation<br>Select ‘okay’<br>Pop should appear to show successful event deletion |
| View profile and edit | Select the ‘Profile’ option from the menu bar<br>Profile information is displayed<br>Select the ‘Edit details’ button<br>Make any changes and enter password<br>Select submit<br>Any invalid details will show warnings<br>Pop should appear to show successful edit |
| Change password | Select the ‘Profile’ option from the menu bar<br>Select the ‘Change password’ button<br>Enter old password then new password<br>Select ‘submit’ |

| | |
|---|---|
| | Any invalid details will show warnings<br>Pop should appear to show successful edit |

### Communication between Client & Server: “ObjectTransferrable”

To standardise communication between Client and Server the team decided to create an abstract class called “Object Transferrable” to act as a supertype. All objects that are passed to and from the server must extend this supertype abstract class ObjectTransferrable. All Object Transferrables also have a String field variable called OpCode. Opcode help differentiate one object type from another. They are used by the protocols on both the client and server to decide what the object contains and how to process its contents. A list of the objects, their opcodes, and their objectives are found below( note that are ordered in a fashion that groups them by use cases):

| Name | opCode | Objective |
|---|---|---|
| OTUsernameCheck | 0001 | Sent to the server to check if a given username is already registered. Sent back to the client to indicate true or false to this check |
| OTEmailCheck | 0002 | Sent to the server to check if a given email is already registered. Sent back to the client to indicate true or false to this check |
| OTRegistrationInformation | 0003 | Sent to the server. Contains the registering users details |
| OTRegistrationInformationConfirmation | 0006 | Sent to the client to indicate the success or failure (with reasons) of registration request |
| OTExitGracefully | 0005 | Sent to the server when the client wishes to close the connection |
| OTErrorResponse | 0007 | Sent to the client when there was an error handling their sent object. It contains details of the error and how it occurred |
| OTRequestMeetingsOnDay | 0008 | Sent to the server to request meetings for a given day |
| OTReturnDayEvents | 0009 | Sent to the client, containing an ArrayList of Events for the given day |
| OTCreateEvent | 0010 | Sent to the server to create an event in the database |
| OTCreateEventSucessful | 0011 | Sent to the client to indicate the success or failure of event creation |

| | | |
|---|---|---|
| OTLogin | 0012 | Sent to the server to request the hashed version of the user's password |
| OTHashToClient | 0015 | Sent back to the client, containing the hashed version of their password |
| OTLoginSuccessful | 0013 | Sent back to the server to indicate the success of their login attempt (no object sent if it failed) |
| OTLoginProceed | 0016 | Sent back to the client to let them know that the server has successfully made note of their login attempt and logged their username against their login session |
| OTHeartBeat | 0014 | The heartbeat is sent to and from the server to indicate the continued presence of the link between the server and the client |
| OTUpdateEvent | 0017 | Sent to the server to update and event |
| OTUpdateEventSuccessful | 0018 | Sent the the client to indicate the success or failure of the update event request |
| OTDeleteEvent | 0019 | Sent to the server to delete an event |
| OTDeleteEventSuccessful | 0020 | Sent to the client to indicate the success or failure of deleting the event |
| OTUpdateUserProfile | 0021 | Sent to the server to update the user’s profile |
| OTUpdateUserProfileSuccessful | 0022 | Sent to the client to indicate the success or failure of up profile update |
| OTUpdatePassword | 0023 | Sent to the server to update a user’s password |
| OTUpdatePasswordSuccessful | 0024 | Sent to the client to indicate the success or failure of a password update |

## Client

The client class represents the user side of the client-server relationship. This portion will outline client-side design decisions and provide justifications for them.

The client uses Java sockets to communicate with the server. Once started, the client spawns three classes:

1. The model
2. The view (UI)
3. A heartbeat thread

The first two classes follow the the Model-View-Controller architecture to run an interactive UI. The user interacts with the view, this in turn changes the model. Finally, the model uses the client to communicate with the server.

To elaborate on the communication aspect, the client has two crucial methods for talking to the server. These are:

1. writeToServer()
2. readFromServer()

Communication between the client and server happens via objects. An abstract super class called Object Transferrable was created for this sole purpose (Please refer to the Object Transferrables section above) The client writes to the server either request information (e.g. asking for the hashed password of a user) or to provide information to the server (e.g. information needed to create a new user). After writing, the client immediately waits to receive a response. Importantly, the above two methods are synchronized using the “synchronized” Java keyword. This makes certain that after writing the server, the client is busy waiting for a response. Until it receives a response, the client will do nothing else ( including writing to the server again). Therefore, communication between client and server always follows a synchronized back and forth pattern. A possible negative is that in case of an unresponsive server, the client could wait for a long time. Our solution to this problem was to set a time limit on this wait. The client will wait for 5 seconds for server to respond before closing down the socket. 5 seconds is a significantly long time to wait for a non I/O operation.
More commonly, the client will receive a reply object from the server. It then calls the runObjectTransferrableReceivedFromServer() method. This method reads the object received and performs a suitable action.

Finally, the HeartBeat thread exists to prompt the client to send a heartbeat every second. The rest of the time, the thread sleeps. To elaborate on the heartbeat mechanism, the client sends a heartbeat object. It then waits for 200 milliseconds for the server to respond. If nothing is returned from the server, the client assumes the connection is dead and runs the cleanUpAndPromptUserToRestart() method. This method displays a suitable error message

and prompts them to check their internet. A “restart” button is also displayed. If clicked, the button will attempt to reopen a connection with the server.

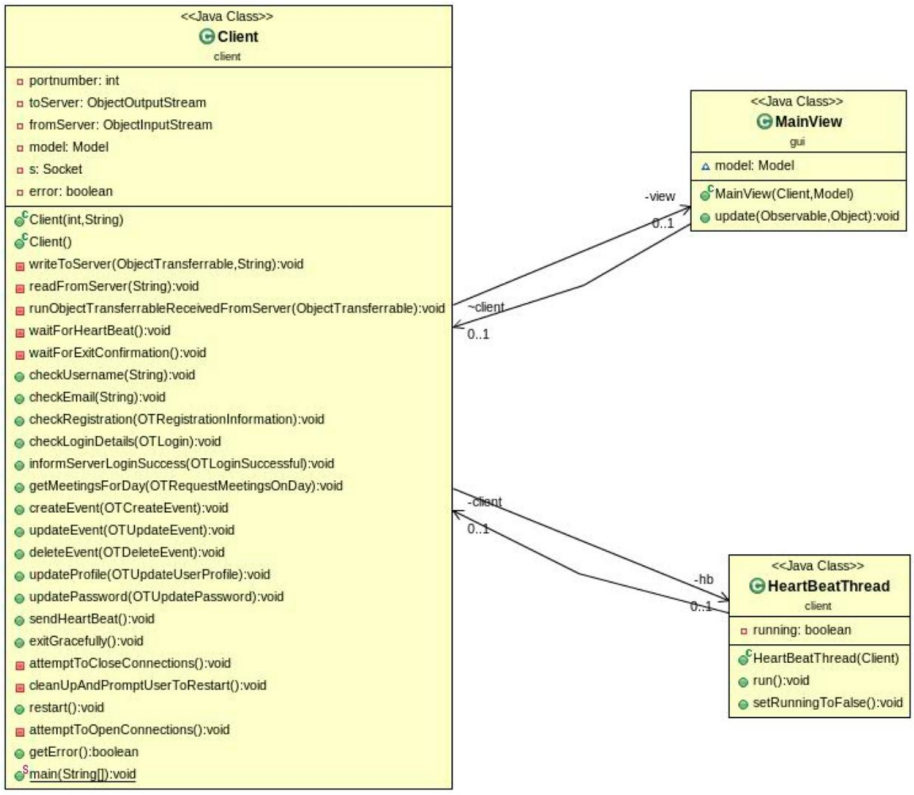
The above paragraphs capture the key aspects of the client. Other minor aspects are outlined below:

- Model States: In our program, the model is given a “state” (a custom Enum) that represents where the user is in the flow of events. The changeCurrentState() method (class: Model) is used to change the state of the model. This in turn affects the view the user sees.

- Exiting: A special object called “OTExitGracefully” is used to signal exit of the client. The server confirms the exit by sending the same object class back. Client waits a maximum of 300 milliseconds for this confirmation. Either way, the socket is shutdown from the client side after 300 milliseconds have passed.

- Password storage: The client/model does not store the user’s password as plain text. Rather a hashed version of the password is stored locally.

Class Diagram for Client Side:
NB: The model is left out for brevity’s sake.



## Database

This project uses a PostgreSQL database to store user and meeting data. It consists of two tables, the user table and the meetings table. They are structured as follows:

| TABLE: users | |
|---|---|
| Column name | Characteristics |
| userName | varchar(20) primary key |
| password | varchar(60) NOT NULL |
| firstName | varchar(30) NOT NULL |
| lastName | varchar(30) NOT NULL |
| userEmail | email UNIQUE |

| TABLE: meetings | |
|---|---|
| Column name | Characteristics |
| meetingID | serial primary key |
| creatorID | varchar(20) REFERENCES users(userName) ON DELETE CASCADE |
| meetingDate | date NOT NULL |
| meetingTitle | varchar(50) NOT NULL |
| meetingDescription | varchar(200) |
| meetingLocation | varchar(200) |
| meetingStartTime | time NOT NULL |
| meetingEndTime | time NOT NULL |
| lockVersion | integer NOT NULL |

Each user can be the creator of many meetings, creating a one to many relationship between these tables.

The user ‘global’ is reserved as it will be used to stored global events that are visible to all users.

The emails that are stored in the database are checked to ensure they adhere to a regular expression that loosely enforces an email structure, given here:

```
CREATE DOMAIN email AS TEXT CHECK (VALUE ~
'^((?:(?:[A-Za-z0-9])+((?:(?:(?:\.)|(?:_))){1}[A-Za-z0-9]+)*)@(?:(?:[a-z0-9
]+\.)(?:(?:[a-z0-9])+\.)*(?:[a-z0-9])+))$');
```

Any email that is supplied to the database will have to have passed identical verification by the client.

## Server

The server serves as infrastructure that supplies client updates and queries to the database, and outputs the results back to the client. The interactions are handled by a strictly designed protocol of the receipt and broadcasting of objects between the client and the server. The objects used are all of the super-type ObjectTransferrable (more details in the ObjectTransferrable section of this report).

All client/server interactions are strictly one-to-one including disconnection, with any object being received by the server resulting in a response back to the client.

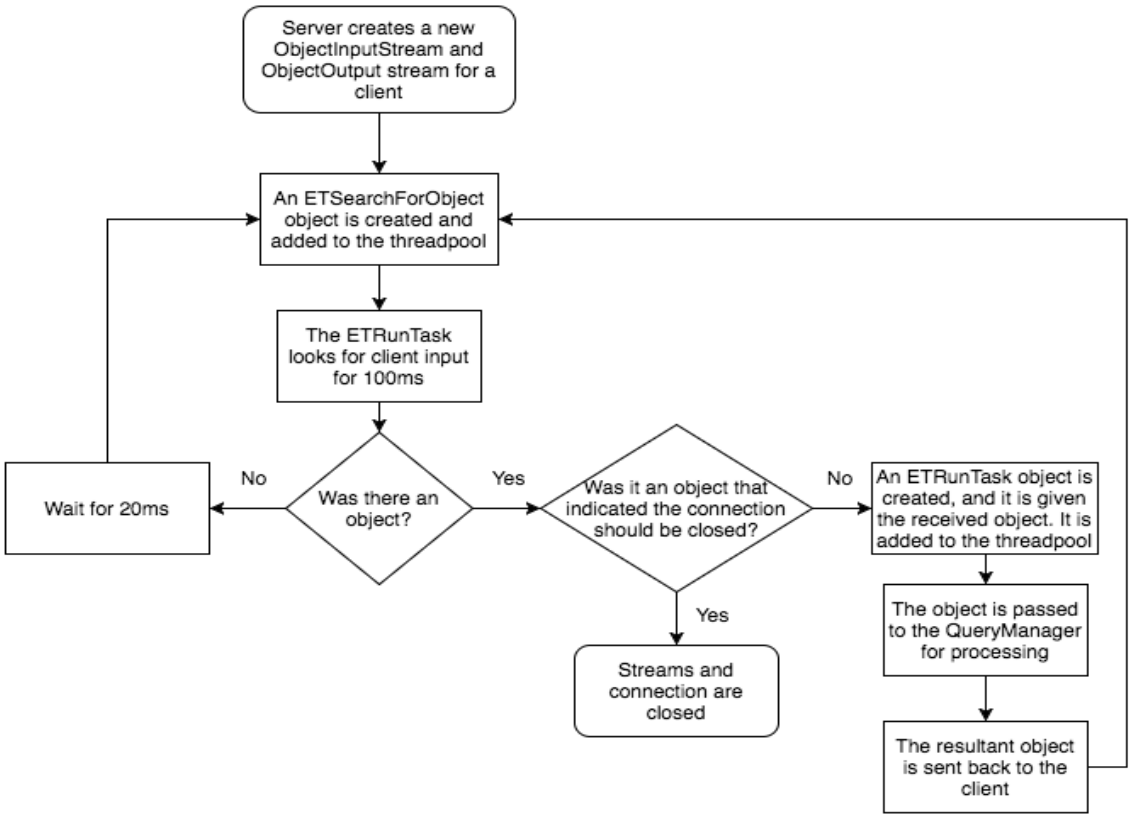
The server has four main components:

1. An ArrayList of client connections
2. An executor service that manages a threadpool of 4 threads, which will be used to look for, execute, and reply to client input
3. The query manager that defines the protocol for dealing with the various different ObjectTransferrables that the server could receive
4. A GUI (designed with a model-view-controller architecture) that displays new connections, allows for managed disconnection with clients, and the displaying of important communication information (such as erroneous events)

The process that the server uses to manage client connections can be broadly described in five steps:

1. A client requests a connection with the server
2. The server accepts the connection, and creates a ClientInfo object which is unique to that client. It constructs ObjectOutputStreams and ObjectInputStreams in that will be used to handle the receipt and sending of ObjectTransferrables.
3. This ClientInfo object is stored in an ArrayList
4. The server then uses the thread pool to periodically check the input stream for objects to process and reply to.
5. When the thread pool processes an object that indicates the client is disconnecting, the streams and connection are closed.

The threadpool will only have one of two objects in its queue, both of which extend ExecutableTask. These objects and the way they interact are best described by the following flow chart:



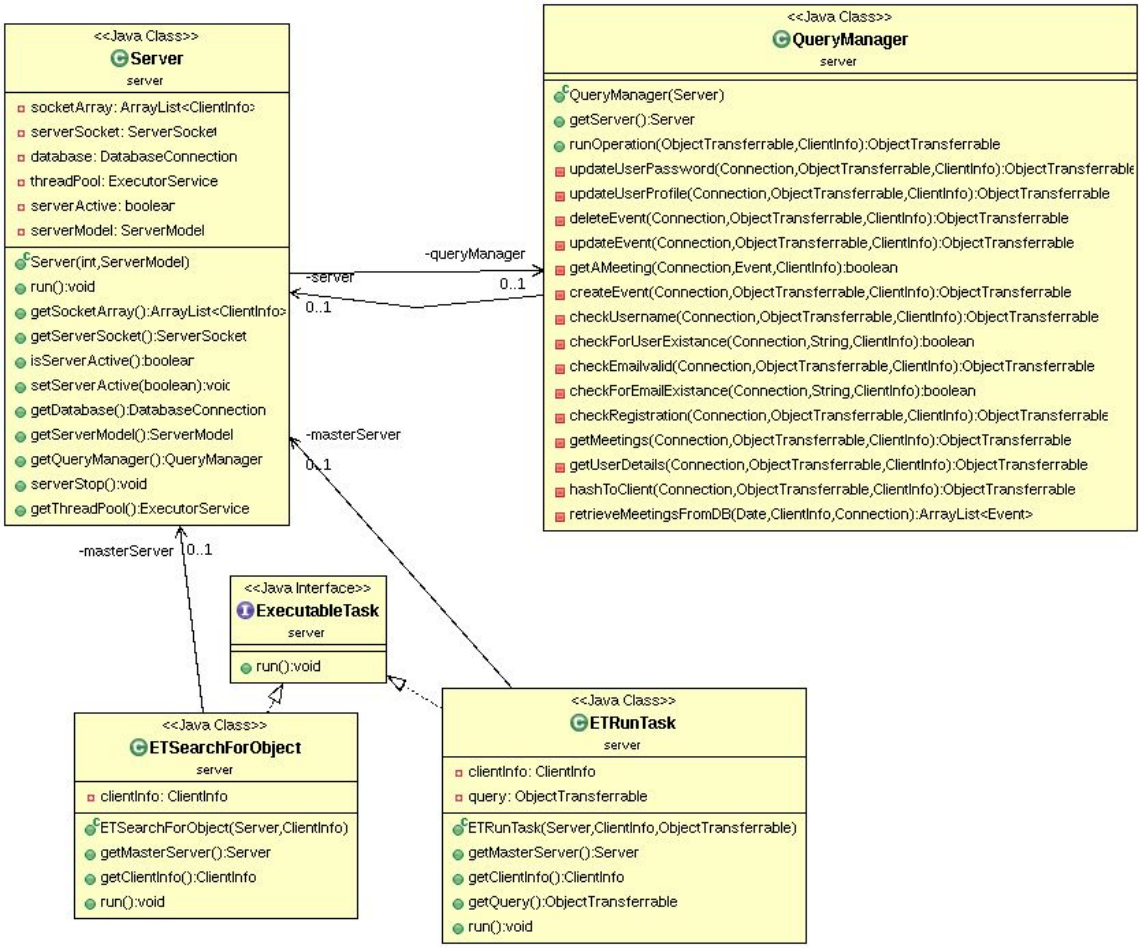

The query manager categorises the ObjectTransferrable that has been received, constructs the relevant query or update to be passed to the JDBC database connection. The results of this are then wrapped in an ObjectTransferrable to be passed back to the client.

Most of the tasks completed by the query manager can be done without having to worry about thread interference, but there are two areas where special consideration had to be made. These are account creation, and the editing of global events.

By synchronising the methods that relate to account creation, and ensuring that the method checks the username and email for uniqueness before attempting to commit to the database, the issues of thread interference are overcome.

The other key area, editing and deleting global events, required a specialised approach. The methods that handles these tasks are synchronized, but it is also important to ensure that the client that isn't trying to delete or edit an out of date version of the event. In order to ensure that this doesn't happen, the lockVersion column of the meeting table is used. When an event is initially created, it is given a lock version of one, and this is incremented with each subsequent edit of the meeting. When a client attempts to edit or delete an event, the server ensures that the version that they are trying to edit or delete has a matching lock version number with the live

database. If it doesn't, the edit or delete will fail and a relevant message will be sent to the client to display to the user.
Class diagram for server-side:




## Test Plan

The following document outlines the planned testing of the calendar system developed by our team, the tests involve both JUnit tests where applicable and functional testing. The system has also been designed with some internal testing and error reporting in mind, particularly in the remote communication elements. Which would be hard to test without this functionality.

**Client side:**
The internal testing of the client side implementation is focused on the client side model and the client class, which store much of the information relative to the client and connection.

**Server side:**

The internal testing of the server side implementation is focused on the server side model, these have functions which can be remotely tested.

**Object transfer system:**
The object transfer system has its own related exception created to make testing easier, there is also an error response object to be used in the event that an error need to be sent over the remote communication. It is unnecessary to test many of the object transfer objects themselves as they are only a collection of private variables and their relevant getter methods.

**Functionality requirements:**
The functionalities of the system as a whole which must be tested are

- The client must be able to create an account which is added to the server database
- The client must be able login which will check credentials with the database server and will only allow the login state if these credentials match a registered account.
- The client must be able to create events and have these added to the database.
- The client must be able to have any events they are associated with retrieved and displayed correctly in the gui.
- The client must be able to log out successfully back to the login screen, which will close any active connection to the server.
- The client must be able to connect to a server on a different IP address
- If the connection to the server and or client are lost, or shutdown unexpectedly. The opposing connection should be closed in a timely manner.
- If the client is closed without logging out, the server should close the connection.
- There must be global events which will be returned to every user when their events on that day are requested.
- The user must be able to request a refresh from the server, the client sends a new request and the server returns relevant events.
- The client must be able to delete events to remove them from the database
- The client must be able to change their password in the system
- The client must be able to change their email address once registered

**Testing and Results:**
Since testing the GUI part of the system and server interactions could not be efficiently JUnit tested, they could only be performed through functional testing.
The functionalities mentioned above were tested almost completely successfully. There was one problem with the server not closing connections correctly if the client was forcibly and suddenly stopped which could not be easily resolved. This was because of varied responses from the socket server side in this case, one of which was a normal response during operation, made solving this problem difficult.

SQL injection was attempted on fields which would be formatted into queries and inserts to the database to check that the prepared statement system was functioning correctly. The prepared statements functioned as expected.
Long entries for fields for events and profile data were tested to make sure they were limited and entered correctly into the database and displayed correctly by the client GUI. There was a minor problem with word wrapping when the event descriptions and locations were first displayed on logging in but navigating to any other page put the wrapping back correctly. This will hopefully be corrected with a minor fix.

The sanity checks were tested by trying to enter invalid data including

- Invalid email formats
- Invalid character lengths
- Invalid number entry of greater length allowed
- Invalid hours greater than or equal to 24
- Invalid minutes greater than or equal to 60
- Invalid event timings where the start is after the end time
- Entries longer than the SQL database storage should allow.

All of these checks were made across every data entry point in the program to which these checks should apply. The majority passed successfully, however it was discovered that the character limits and email validation were not applied properly to the change profile data, and a change in password was not properly recognised by the client. There were also some pop up message problems discovered in this area.

A list of the states which the client and server could be in were assembled, for the server this was basically singular for any given connected client. However there are multiple possible states for the client. The states, and the ones that would be testable could be identified by examining the ModelState enumeration file for the GUI, general operations such as checking how the heartbeat was functioning and how the system coped with shutdown requests at the client side. This testing identified a couple of minor problems such as the client not being able to be closed whilst the edit event window is open. If the server is shut down in this state it causes an incorrect message to be displayed but does not break the program.

One of the other tests conducted was trying to click each button multiple times in rapid succession to check for unwanted behavior, this identified a problem with how certain operations were handled.

Over the course of this testing many debug outputs were used by the console to examine that responses were as expected at various stages of the operation. Many of these were later removed, with hindsight these debug messages could have been tied to an overall debug variable allowing them to be switched on and off as necessary.

**Conclusions:**

Overall by taking a methodical approach to testing and analysing the possible inputs, states, and actions. Looking at how they should be limited, and their expected responses, a relatively thorough testing could be carried out. This is despite the fact much of the system cannot be unit tested by conventional means. The mixture of using functionality requirements and edge conditions provided a lot of useful analysis to the function of the program, and although some problems were identified that have yet to be fully fixed. The system as a whole is generally functional and achieves its functionality requirements, dealing with most problems well.

## Evaluation:

**Team evaluation**
Our aim was to create a well developed calendar desktop application with features for adding contacts and viewing other users' calendars. In hindsight, these aims were quite far-fetched. We were also, due to unforeseen circumstances, working with reduced manpower. However, the product we did produce can be seen as a functional calendar prototype.

Moving on to our thoughts on what went well: the team was well organized and supportive throughout. A facebook group chat was used to by most members to keep the team informed. It was also a place to air ideas and concerns.

Interestingly, the group project taught us that early and clear communication is important. It was important for the team to meet in person at least three times a week with all members. This did not happen in the beginning. Were we to do this project again, these aspects would be changed. Moreover, greater thought would go into the planning stage.

If we were tasked with extending the project, the team would focus on implementing a "Forgotten password" feature. This currently does not exist. We would also like to add thematic wallpaper for each month. Another area to extend would be to view other members' personal calendars. If we were to implement a large piece of extended functionality, it would be to create meeting with listed attendees, that are shared amongst the users.

**Software evaluation**
*Usability* - The calendar aims to inform users of system status. To this end, clear message pop-ups are displayed after a significant action is carried out e.g. log-in provides a "log-in successful" popup. Incorrect username login provides an "incorrect username" popup etc.…
*Usability* - The calendar has a minimalist design. Irrelevant and confusing information is not shown to the user.
*Usability* -The calendar helps users diagnose and recover from errors. In case of internet connection failure, the system shows a helpful error message prompting the user to reconnect. It also provides a restart button.
*Testing* - The calendar has Unit tests. Scripts could be written to for carrying out GUI testing.

*Testing* - A large area of testing that, given more time, would have been a worthwhile investment, is load testing the server in an automated fashion. This would have allowed us to see how the server dealt with a large volume of clients providing a large volume of traffic.

## Project Diary:

| Meeting Date | Time | Attendees | Agenda |
|---|---|---|---|
| 23/02/16 | 1h | Everyone | Decided on project |
| 25/02/16 | 30mins | Everyone but Tom | Decided on roles |
| 26/02/16 | 1h | Everyone | Updated each other on progress |
| 01/03/16 | 1h | Everyone but Tom | Updated each other on progress<br>Rearranged roles<br>Decided on using transferable object |
| 02/03/16 | 1h | Mark, Swetha, Natalie | Updated each other on progress<br>Worked on our roles<br>Database running |
| 03/03/16 | 1h | Mark, Swetha, Natalie | Updated each other on progress<br>Worked on our roles<br>Client built, transferable objects made |
| 04/03/16 | 2h | Mark, Swetha, Natalie | Updated each other on progress<br>Worked on our roles<br>Server working and linked to database |
| 08/03/16 | 30mins | Swetha, Natalie | Updated each other on progress<br>Worked on our roles |
| 09/03/16 | 1h | Everyone | Updated each other on progress<br>Worked on our roles |
| 11/03/16 | 2h | Mark, Swetha, Natalie | Updated each other on progress<br>Worked on our roles |
| 13/03/16 | 7h | Mark, Natalie, Tom (4h), Swetha (6h, working remotely) | Updated each other on progress<br>Worked on our roles |
| 15/03/16 | 4.5h | Mark, Swetha (3h) Natalie | Updated each other on progress<br>Worked on our roles |
| 16/03/16 | 4.5h | Mark, Swetha, Natalie | Updated each other on progress<br>Worked on our roles |
| 17/03/16 | 4h | Mark, Natalie, Swetha, Tom (2h) | Updated each other on progress<br>Worked on our roles<br>Whole thing running, only few changes needing to be made |
| 18/03/16 | 3h | Mark, Swetha, Natalie, Tom | Updated each other on progress |

| | | | |
|---|---|---|---|
| | | | Made small changes |
| 20/03/16 | 5h | Mark, Natalie (3h), Swetha, Tom | Worked on report<br>Made the last few small changes |

## Team Organisation Report:

The initial roles we decided on were as follows:

- Mark - Database
- Swetha - Server, taking minutes
- Natalie - GUI, taking minutes
- Tom - Client
- Caroline - Server, contacting tutor

Due to unfortunate circumstances Caroline admitted that she would be unable to help out with the code as she has been in and out of hospital. She contacted welfare and spoke to our tutor so we arranged that she would organise the group, prepare the report and do some testing. Unfortunately this was not able to happen so we had to arrange otherwise.

Very shortly afterwards Tom was taken by a personal issue which again was taken up with welfare and our tutor. Fortunately Tom was able to join in near the end of the project and produced the test plan while also creating transferable objects and helping out with the last small changes. Hence this is the actual roles the group had:

- Mark - Database, Server, Transferable Objects, Client-Side receiving and sending of Object Transferrables
- Swetha - Client, Taking minutes, Model
- Natalie - GUI, Model, Taking minutes
- Tom - Transferable Objects, Various small additions, Testing and Test Plan
- Caroline - N/A

Mark produced the report on the database, server and object transferables; Swetha on the client, Natalie on the UI and Tom on the testing.
As a group we feel we dealt with these issues well and produced a good final product under the circumstances.

## References:

(1) "One Account. All of Google." *Google Calendar*. N.p., n.d. Web. 19 Mar. 2016. <https://calendar.google.com/calendar/render?pli=1#main_7%7Cday>.
(2) "Java Software." *Java Software*. N.p., n.d. Web. 1 Mar. 2016. <https://www.oracle.com/java/index.html>.
(3) "Stack Overflow." *Stack Overflow*. N.p., n.d. Web. 1 Mar. 2016.

(4) "Software Product Evaluation." *Software Process Improvement* (2011): 1-6. Web. 19 Mar. 2016.