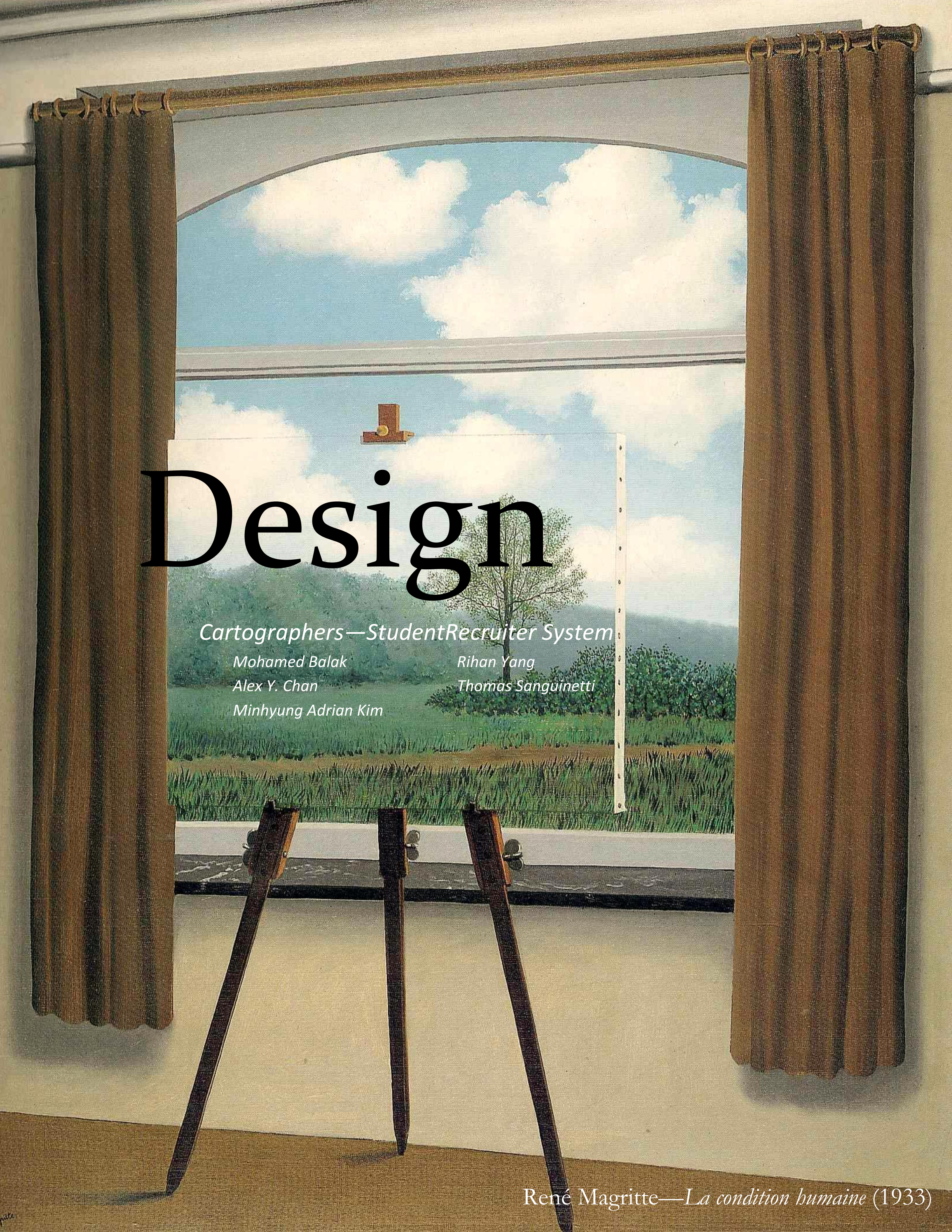


Design

Cartographers—StudentRecruiter System

Mohamed Balak
Alex Y. Chan
Minhyung Adrian Kim
Rihan Yang
Thomas Sanguinetti

René Magritte—*La condition humaine* (1933)

`

Table of Contents

`

`

`

1. Introduction

1.1 Purpose

The purpose of this design document is to provide a description of the *StudentRecruiter System* (SRS) in order to guide the development of it into a working software product. The SRS is designed to manage the organisation of visits by schools and students to university open days and events.

1.2 Scope

This document describes the implementation details of the SRS. The SRS will consist of three main functions: 1) to facilitate the planning, publishing and marketing of university events, 2) to enable registered users to book to attend events (including scheduling concerns), 3) to collect and meaningfully present useful data (such as feedback data, demographic data) from users about university events to university staff.

The SRS will assist departments within universities to organise and communicate event information to schools

1.3 Document overview

This document includes graphical, technical, and narrative documents such as the entity-relationship diagram, algorithm design (pseudocode), a class diagram, prototypes of the user interface, and sequence diagrams which aim to detail and guide the eventual implementation of the system's specified requirements. This document specifies evaluation and validation techniques which will help the developer determine how the system should work.

This document provides a high-level description of the SRS that aims to be accessible to:

- Software developers
- IT staff and advisers
- Computer Science lecturers

It is assumed that the target audience will have experience and technical knowledge of software design and development, including industry standards such as UML.

1.4 Definitions

SRS – StudentRecruiter System

1.5 System environment

Development: Eclipse
Diagrams: LucidChart, ArgoUML, MySQLWorkbench

`

Database: MySQL 5

1.6 Design approach

The design approaches we have used are as follows:

1.6.1 Data flow design

The data flow of the SRS is mostly internet-based. Data is sent to and retrieved from a MySQL database via the user’s interaction with the application layer.

1.6.2 Architecture design

We use a three-tier architecture in which our application will divide into three layers: the client, the application server and the database.

1.6.3 User interface design

We employ a Model-View-Controller (MVC) architecture in our design and implementation of the user interface layer. This architecture separates the user’s actions and what they see from the data that is held in the system.

1.6.4 User interface view design

The design of the user interface (UI) is based on aspects of the page schematic approach. Drawings of the pages that users will encounter emphasise the navigability of the application, and how the user will interact with it to input and view the relevant data required to complete specified requirements (eg. pg ‘use case: view events’).

`

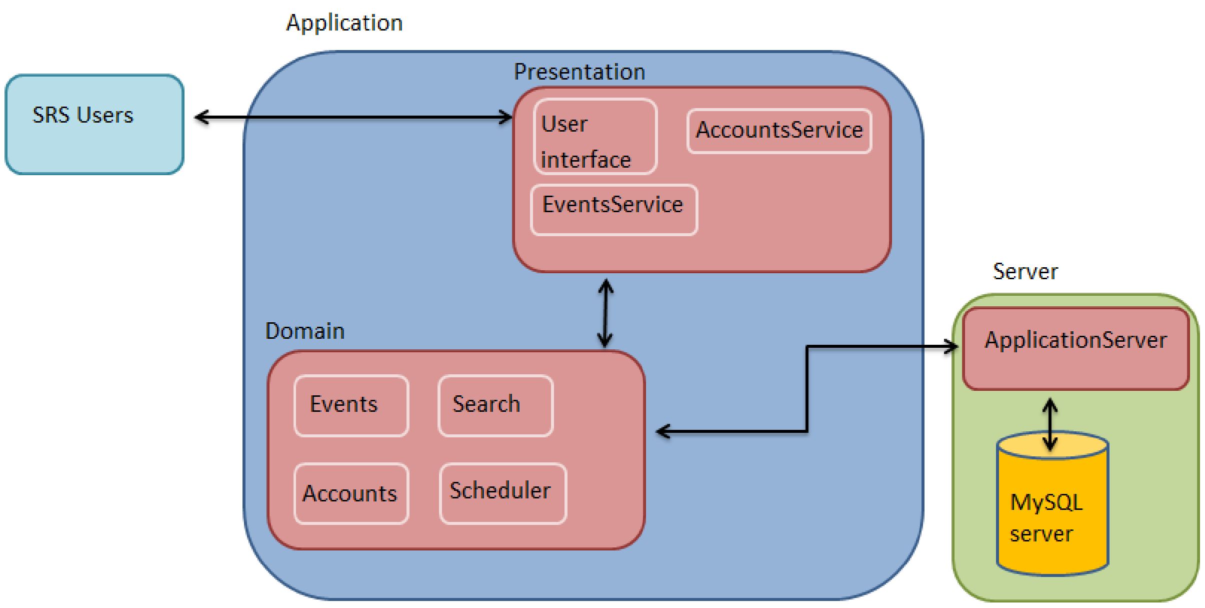
2. Requirements traceability

Requirement	Description	Design reference
REQ1, 2, 3, 4	Register and login	§3.5.1, §3.6.1, §5.4.1, §5.4.4, §5.4.5, §5.4.6, §6, §6.2.3, §6.3.1, §7, §8.1
REQ5, 6, 7, 8, 9	Viewing events, applying to attend event	§3.5.2, §3.6.2, §4.1, §4.2, §4.3, §5.5.2, §5.4.8, §6, §6.2.1, §6.2.2,
REQ10, 11, 12	Creating an event	§4.4, §5.5.1, §5.4.7, §5.4.10, §5.4.11
REQ13, 14, 15, 16	Admin functions	§3.5.3, §5.4.3, §8
REQ17	Feedback	§5.4.12, §7

`

3. Architectural design

3.1 System boundary diagram



3.2 System architecture

We employ a Model-View-Controller (MVC) architecture in our design and implementation of the user interface layer. This architecture separates the user's actions and what they see from the data that is held in the system.

User actions, such as listing available events or searching for an event, are handled by a corresponding controller. The view is what the user will see from either a Java client or a web page. The model retrieves data from the server and parses it into the correct format for display.

We make use of an Object-Oriented approach to identify existing inheritances inside the system to facilitate reuse of code and reduce repetition and redundancy. All interface controllers will be designed as subclasses of `ApplicationController`, so common actions like user authentication can be shared by every controller.

3.3 Three-tier architecture

We use a three-tier architecture in which our application will divide into three layers: the client, the application server and the database. The client is our presentation tier, the application server the domain logic tier, and the database the data storage tier.

`

The advantage of using sSuch an architecture is that our system becomes flexible and reusable. For example: we will create different client applications to access the same database; we are able to change different parts of the system without reworking the entire application.

- Presentation tier: Java client and a website for students
- Logical tier: PHP / Node.js / Java
- Data tier: MySQL server

The Java client can get the event list through the API by sending the following HTTP request to the server: `GET/events/list`

The server will then return a JSON array back to the client.

3.4 High level components

Splitting the system into "components" allows us to view and model how the system will operate and behave with different areas of the system all interacting.

The different components of our system will not be differentiated by the programming language used to develop but by how the different components are used.

With our system having a number of different elements interacting with each other, we have split it up into three components described below.

3.5 Components

3.5.1 Registration and Accounts

This component of our system deals with account details and registration of all users as well as logging in and out of the system.

3.5.2 Events

This part of the system deals with events and any event related activity that happens. This could be from viewing detailed event information or admin users applying for an event to be created.

3.5.3 Administrator

Finally the administrator, as well as being a particular type of user of the system, will be a title for the part of the system that deals with event application details and changes as well as the scheduling of events and changes to account details.

`

3.6 Components Interfaces

3.6.1 Registration and Accounts

Account registration:

This interface comprises a simple registration form for the users to fill in. In doing so they will be authorised to log into the system using their account number and password.

Update account details

Similar account registration, updating account details will have a clear interface with form fields containing their own account information. These form fields will be editable in order for users to amend their own account details and change their password should they wish.

3.6.2 Events

Main user interface

The main screen of our system will provide a number of features and system functionality for our users, centring around displaying available events In the middle of the page. Users will able to navigate easily around this interface to find more detailed information about the displayed events.

View events

A separate user interface that displays all events on our system for users to browse. From here users can easily events using a quick book button or find more details on each event by clicking on the event itself.

Event information

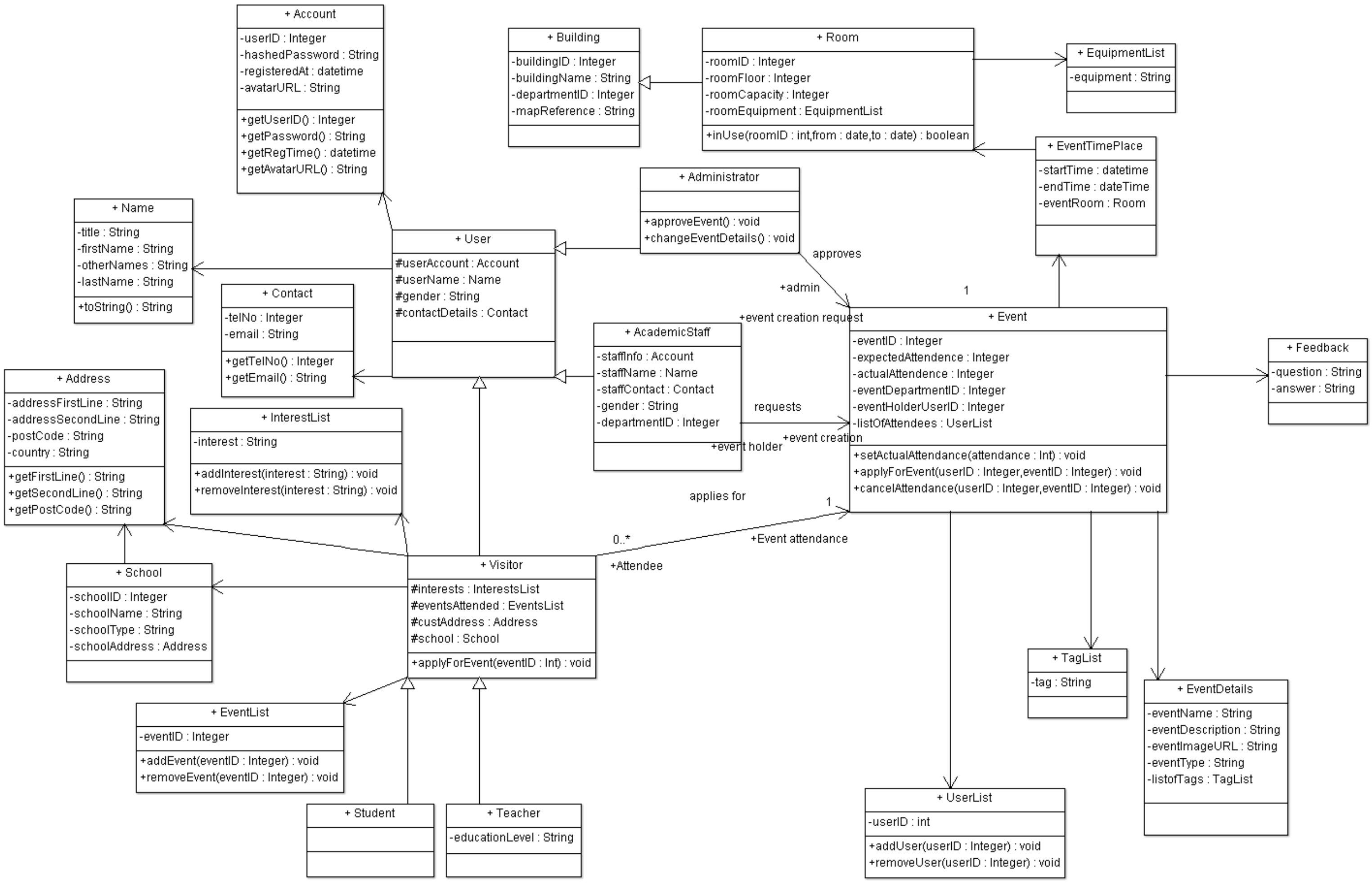
An interface used for displaying full details about each particular event. There will not be much user interaction with this interface as its main purpose will be for displaying information and graphics.

Book event

Booking a particular event will be done on a simple user interface. It will contain only a small amount of components in order to guide the user into booking without any distractions. Only a few fields will be needed, as the system will use the users previously entered account information.

`

3.7 Class diagram



`

4. Algorithm design

4.1 Description and interest matching (brute force method)

Each user will be able to select from a checklist a list of their fields of `interest` (e.g. medicine, careers, concerts) on registering for an account as well as on their account details page. On creation, each event will have a blurb which describes the event, `description`. To work out which events to recommend to the user, one factor of relevancy would be to find which events have the most frequent use of the `interest` string in their `description`.

```
String description
String interest
Integer count = 0
do
    // from the first char of the description up to the last char
    less the number of chars in the interest
    for( i = 0; i < len(description) - len(interest); i++ )
        // if current description char is equal to the first char
        of the interest
        if( description[i] == interest[0] )
            // search the next chars in the description until non-
            matching char is reached or complete match with
            interest is found, return true if complete match
            found, return false otherwise
            for( j = 1; j < len(interest); j++ )
                while( description[i + j] == tag[j] )
                    ++count
```

4.2 Tags and interest matching (brute force method)

Each user will be able to select from a checklist a list of their fields of `interest` (e.g. medicine, careers, concerts) on registering for an account as well as on their account details page. On creation, each event will include a list of `tag`s which is a list of keywords about the event. To work out which events to recommend to the user, one factor of relevancy would be to find which events use the `interest` string as a `tag`. That is, look for a match in the list of `tag`s and `interest`s

```
String tag
String interest, firstInterest, lastInterest
Integer count = 0
Boolean match = false
do
    for(interest = firstInterest to interest = lastInterest)
        while( match = (interest != tag) )
            interest.nextInterest()

    return match
```

`

4.3 Search and Recommender Engine

The SRS allows user to search through a variety of event descriptions and the system also has the capability to recommend events to the user based on his profile or browsing history. Thus a precise search and recommender system is required. We will adopt a simple but powerful algorithm for this system. TF-IDF (Term Frequency-Inverse Document Frequency) is applied to do the text indexing to evaluate the importance of each word occurrence (Manning, Raghavan, and Schütze 2008). Following that, VSM (Vector Space Model) will be used to compute the similarity between the query and the text (Lops, De Gemmis, and Semeraro 2011).

4.3.1 Weighting

Let $\mathrm{D} = \{\mathrm{d}_1, \mathrm{d}_2, \ldots, \mathrm{d}_\mathrm{N}\}$ denotes the set of documents (in our system is descriptions of events). A set $\mathrm{T} = \{\mathrm{t}_1, \mathrm{t}_2, \ldots, \mathrm{t}_\mathrm{n}\}$ contains the words from English dictionary. The importance of a word is weighted by the product of TF (term frequency) and IDF (inverse document frequency). That is to say, a word is likely to be important if it appears many times in one particular document (high TF value) but few times in other documents (low IDF value). The following formula computes TF-IDF:

$$\mathrm{TFIDF}(\mathrm{t_k}, \mathrm{d_j}) = \mathrm{TF}(\mathrm{t_k}, \mathrm{d_j}) \cdot \log \frac{\mathrm{N}}{\mathrm{n_k}}$$

Which can be further normalized to in order to make the weight a value between 0 and 1

$$\mathrm{w_{k,j}} = \frac{\mathrm{TFIDF}(\mathrm{t_k}, \mathrm{d_j})}{\sqrt{\sum_i \mathrm{TFIDF}(\mathrm{t_i}, \mathrm{d_j})^2}}$$

4.3.2 Scoring

The similarity between two documents (or between query and document) can be measured by vector inner product. The cosine similarity is much easier to compute than any other form:

$$\mathrm{similarity}(\mathrm{d_i}, \mathrm{d_j}) = \frac{\mathrm{d_i} \cdot \mathrm{d_j}}{||\mathrm{d_i}|| \cdot ||\mathrm{d_j}||} = \frac{\sum_k w_{ki}^2 \cdot w_{kj}^2}{\sqrt{\sum_k w_{ki}^2} \cdot \sqrt{\sum_k w_{kj}^2}}$$

`

4.4 Room scheduling

We anticipate that the university staff expect that once an event and a room are timetabled together, the time and place of an event should be somewhat inflexible in order to reduce confusion for visitors. Exceptions to this include accessibility issues, priority scheduling, higher than expected demand, equipment mismatch (e.g. a projector has become broken), among others. Hence, the scheduling of events will be mostly based on a first come first serve (FCFS) basis. If the desired room is not available, other rooms in the same building should be checked.

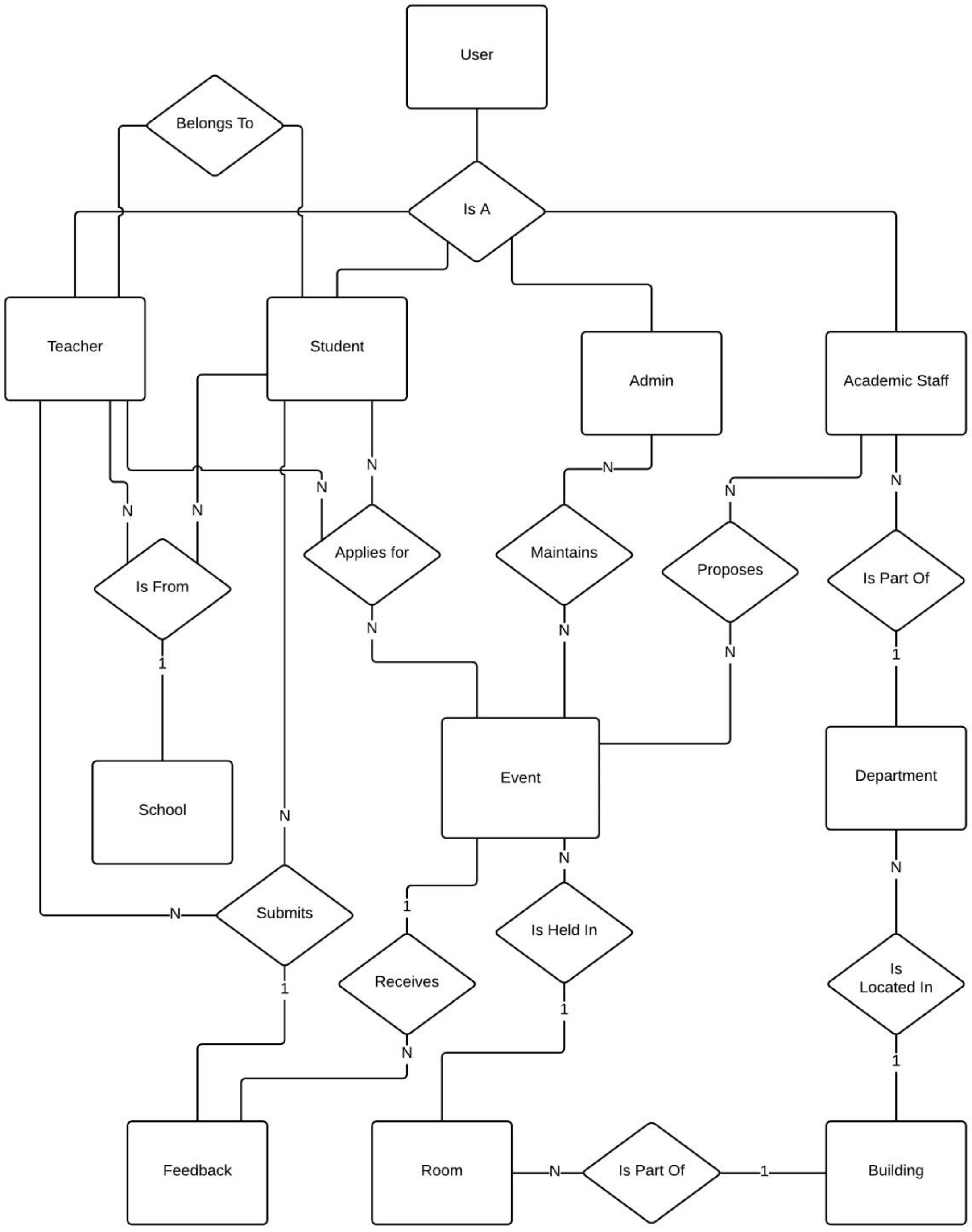
```
do
  // if room capacity exceeds expected attendance, and room is not
  already scheduled to hold an event, add event and room bind to
  schedule return true, else do nothing and return false
  if( room.capacity() > event.expectedAttendance() &&
  room.isEmpty(event.startTime(), event.endTime() )
       scheduleEvent()
       return true

  else

       return false
```

`

5. Database design

5.1 Entity relationship model

We first chose several entities that are required to implement in a database system from the use case diagram. Then following a standard database design process, the relationship between the entities are identified through an ER Diagram as shown below.



`

5.2 Logical table design (normalisation)

Next, we went through a standard normalisation process. Applying these rules will help the system to be organized in a more logical way and to reduce the possibility of data duplication. At this stage, we first decomposed many-to-many relationship into two one-to-many relationships. For example, for students applying for events, an intermediate table, `applications`, with two foreign keys pointed towards `students` and `events` is created. We applied the normalisation process to turn the schema into 3NF.

Third Normal Form (3NF) is a table that is already in 1NF and 2NF, and in which the values in all non-primary-key columns can be work out from only the primary key column(s) and no other columns.

Please refer to the Data Dictionary (4.4) to see the final schema of the database.

5.3 Physical table design

5.3.1 Overview

The physical database design comes after the logical design. The database of the SRS will be hosted on a MySQL Database System on a Linux machine. We will make use of a several useful features from MySQL Database including indexing and foreign key constraints to keep the data integrity and give a better performance.

5.3.2 Database Indicies

Turing on the index can allow the database system to locate the information faster. Primary keys will be indexed by MySQL automatically but we will also need to turn on unique index and un-clustered index to deliver a better performance.

5.3.2.1 Unique Index

The unique index is used to ensure the uniqueness on a column. Unique index is applied on column `email` of table `users` to prevent duplicate email address in our system.

5.3.2.2 Un-clustered Index

Turning on un-clustered index on frequently queried columns can speed up the database system. In our system, all the foreign keys are indexed since they will be used very often when performing join operations. Several datetime fields are also indexed by us because the administrators may search through a data range frequently.

All the indicies done manually are listed in the table below.

`

Table	Column
users	email
events	start_time
events	proposed_at

5.3.3 Foreign Key Constraints

Foreign key constraints are used to ensure the referential integrity. In our system, SET NULL, CASCADE and RESTRICT are used. Depending on the relationship, different options are enabled. For example, in table events we have the following constraints:

```
CONSTRAINT `fk_events_room_id`
FOREIGN KEY (`room_id`)
REFERENCES `mydb`.`rooms` (`id`)
ON DELETE SET NULL
ON UPDATE CASCADE,
CONSTRAINT `fk_events_proposed_by`
FOREIGN KEY (`proposed_by`)
REFERENCES `mydb`.`users` (`id`)
ON DELETE SET NULL
ON UPDATE CASCADE,
CONSTRAINT `fk_events_approved_by`
FOREIGN KEY (`approved_by`)
REFERENCES `mydb`.`admins` (`user_id`)
ON DELETE SET NULL
ON UPDATE CASCADE
```

`

5.4 Data Dictionary

Table definitions for every entity in our database are shown below including the column names, data types and a brief description.

5.4.1 User

`users` defines all attributes for the all types of users including login credentials and user profiles.

Column	Data Type	*NULL*	Description
id	INTEGER	No	Primary key
email	VARCHAR(90)	No	Unique email address of the user
hashed_password	VARCHAR(90)	No	Hashed password
firstname	VARCHAR(45)	No	First name
lastname	VARCHAR(45)	No	Last name
gender	TINYINT	Yes	Gender
dob	DATE	Yes	Date of birth
avatar	VARCHAR(180)	Yes	URL to the avatar of the user
registered_at	DATETIME	No	Default: Current timestamp

5.4.2 University staff

`staffs` defines staff user type.

Column	Data Type	*NULL*	Description
user_id	INTEGER	No	Primary key references user
department_id	INTEGER	Yes	Foreign key references department
phone	VARCHAR(45)	Yes	Phone number of the staff member

5.4.3 Admin.

`admins` defines admin user type.

Column	Data Type	*NULL*	Description
user_id	INTEGER	No	Primary key references user
phone	VARCHAR(45)	Yes	Phone number of the admin

`

5.4.4 Teacher

`teachers` defines teacher user type.

Column	Data Type	*NULL*	Description
user_id	INTEGER	No	Primary key references user
school_id	INTEGER	Yes	Foreign key references school
phone	VARCHAR(45)	Yes	Phone number of the teacher

5.4.5 Student

`students` defines student user type.

Column	Data Type	*NULL*	Description
user_id	INTEGER	No	Primary key references user
school_id	INTEGER	Yes	Foreign key references school
teacher_id	INTEGER	Yes	Foreign key references teacher
address_line1	VARCHAR(90)	Yes	The address line 1
address_line2	VARCHAR(90)	Yes	The address line 2
address_line3	VARCHAR(90)	Yes	The address line 3
postcode	VARCHAR(45)	Yes	The postcode for the address

5.4.6 School

`schools` defines the structure of a school.

Column	Data Type	*NULL*	Description
id	INTEGER	No	Primary key
name	VARCHAR(90)	No	Name for the school
school_type	VARCHAR(45)	Yes	Type of the school
address_line1	VARCHAR(90)	Yes	The address line 1
address_line2	VARCHAR(90)	Yes	The address line 2
address_line3	VARCHAR(90)	Yes	The address line 3
postcode	VARCHAR(45)	Yes	The postcode for the address
tel	VARCHAR(45)	Yes	The contact number of the school

`

5.4.7 Event

`events` defines attributes of an event.

Column	Data Type	*NULL*	Description
id	INTEGER	No	Primary key
title	VARCHAR(180)	No	Title for the event
description	TEXT	No	Full description for the event
tags	VARCHAR(180)	Yes	Comma separated tags for recommender system
start_time	DATETIME	Yes	Start time of the event
end_time	DATETIME	Yes	End time of the event
proposed_at	DATETIME	No	Time when the event is proposed for review
proposed_by	INTEGER	Yes	Foreign key references staff
approved_at	DATETIME	Yes	Time when the event is approved by admin
approved_by	INTEGER	Yes	Foreign key references admin
status	VARCHAR(45)	Yes	The current status of the event
applicants	INTEGER	No	Number of applications
facebook_link	VARCHAR(180)	Yes	Facebook link for this event
twitter_link	VARCHAR(180)	Yes	Twitter link for this event

5.4.8 Application

`applications` is used for students to apply for events. It decomposes the many to many relationship as an intermediate table between user and event.

Column	Data Type	*NULL*	Description
id	INTEGER	No	Primary key
student_id	INTEGER	Yes	Foreign key references student
event_id	INTEGER	Yes	Foreign key references event
created_at	DATETIME	No	Default: Current timestamp

`

5.4.9 Department

`departments` supplies information for departments.

Column	Data Type	*NULL*	Description
id	INTEGER	No	Primary key
name	VARCHAR(90)	No	Name of the department
building_id	INTEGER	No	Foreign key references building
address_line1	VARCHAR(90)	Yes	The address line 1
address_line2	VARCHAR(90)	Yes	The address line 2
address_line3	VARCHAR(90)	Yes	The address line 3
postcode	VARCHAR(45)	Yes	The postcode for the address
tel	VARCHAR(45)	Yes	The contact number of the department

5.4.10 Building

`buildings` defines attributes of a building.

Column	Data Type	*NULL*	Description
id	INTEGER	No	Primary key
name	VARCHAR(45)	No	Name of the room
map_no	INTEGER	Yes	Number of the campus map

5.4.11 Room

`rooms` records room name, location and facilities.

Column	Data Type	*NULL*	Description
id	INTEGER	No	Primary key
room_name	VARCHAR(45)	No	Name of the room
room_no	INTEGER	Yes	Number of the room (if any)
building_id	INTEGER	No	Foreign key references building
size	INTEGER	No	Maximum capacity of the room
has_projector	BOOLEAN	No	The room has a working projector

`

5.4.12 Feedback

`feedbacks` collects feedbacks from the students.

Column	Data Type	*NULL*	Description
id	INTEGER	No	Primary key
student_id	INTEGER	Yes	Foreign key
comment	TEXT	Yes	Comments for students
submit_time	DATETIME	Yes	Default: current timestamp

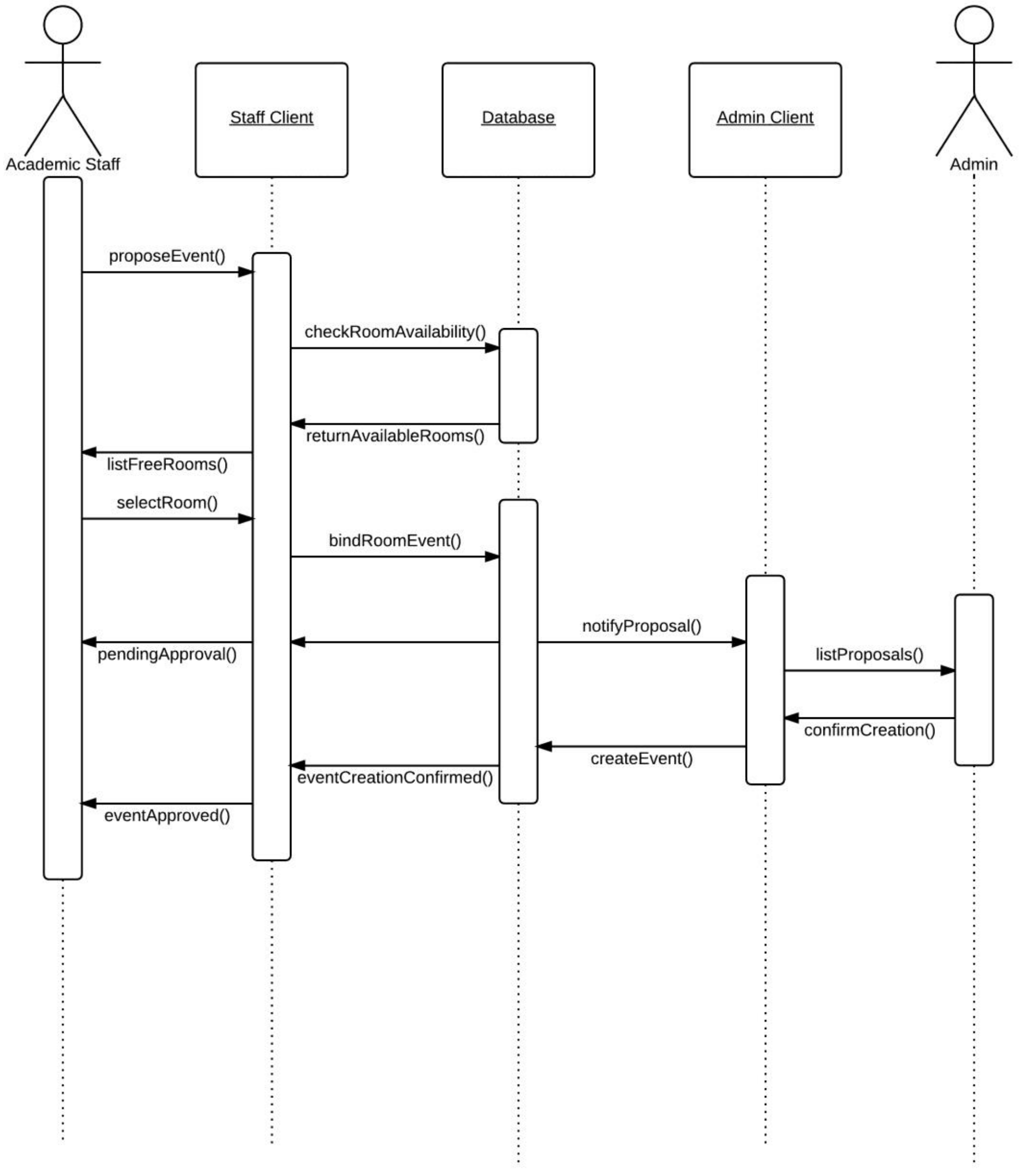
`

5.5 Sequence diagrams

The following UML sequence diagrams show the flow of activity for two use cases: the creation of an event, and booking an event when it is full.

5.5.1 Creating an event

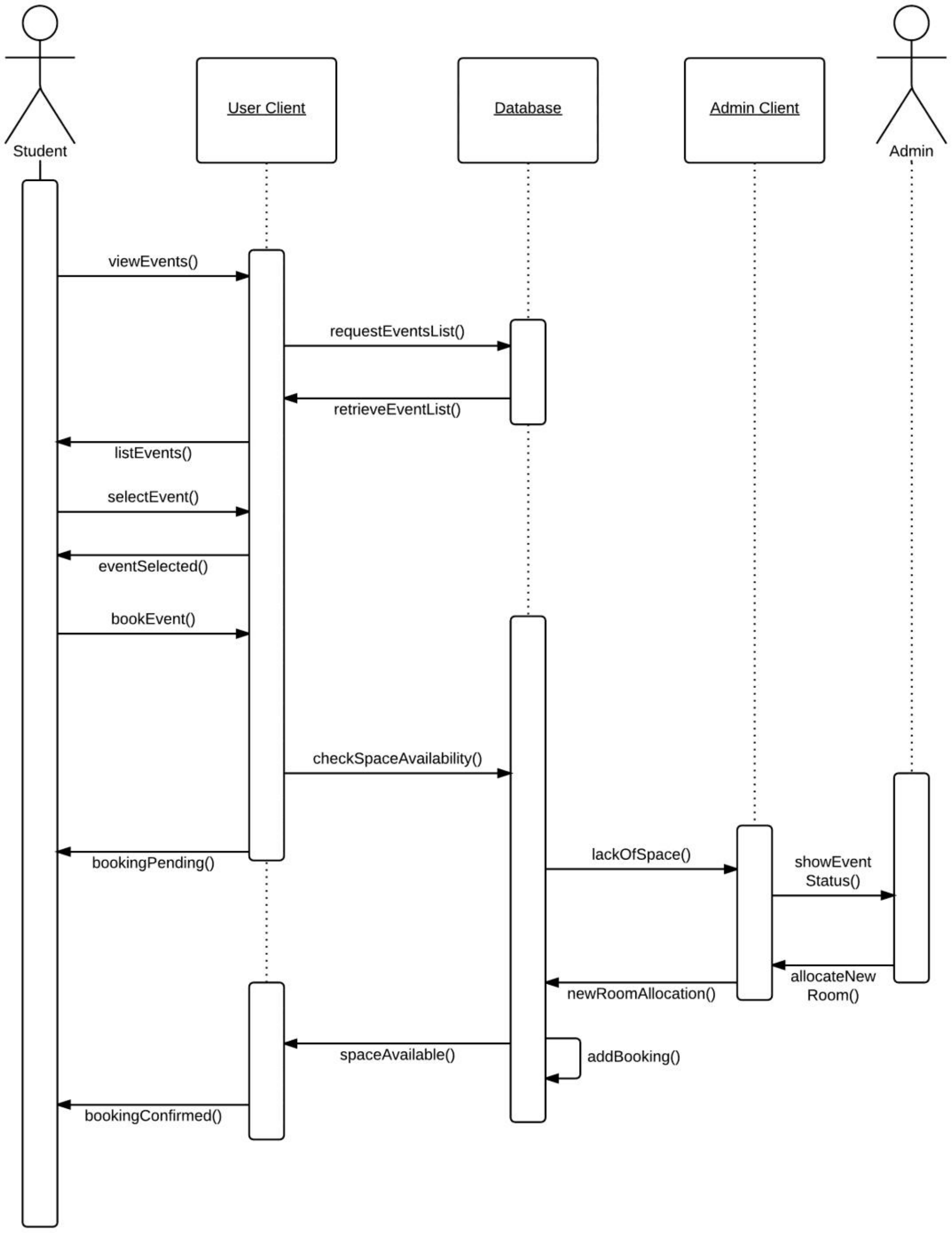
The diagram below shows the process of event creation. Two users: academic staff and admin, and three objects: the client for each user and the database are involved. The staff first proposes an event creation by submitting a form and selecting a room via the staff client, which will in turn access the database to retrieve necessary information. When the room has been selected, the admin is informed of this and can confirm the event creation which will finally notify the staff of the approval.



`

5.5.2 Booking an event which is full

The diagram below shows the case when a student attempts to book an event which is full. Instead of the booking immediately becoming successful, the booking will go into a pending state (the client notifies the student of this via a `bookingPending()` method.) The administrator is notified of the problem and allocates a bigger room for the event, creating space for the user. The database is updated and the user client shows the booking confirmation to the student.

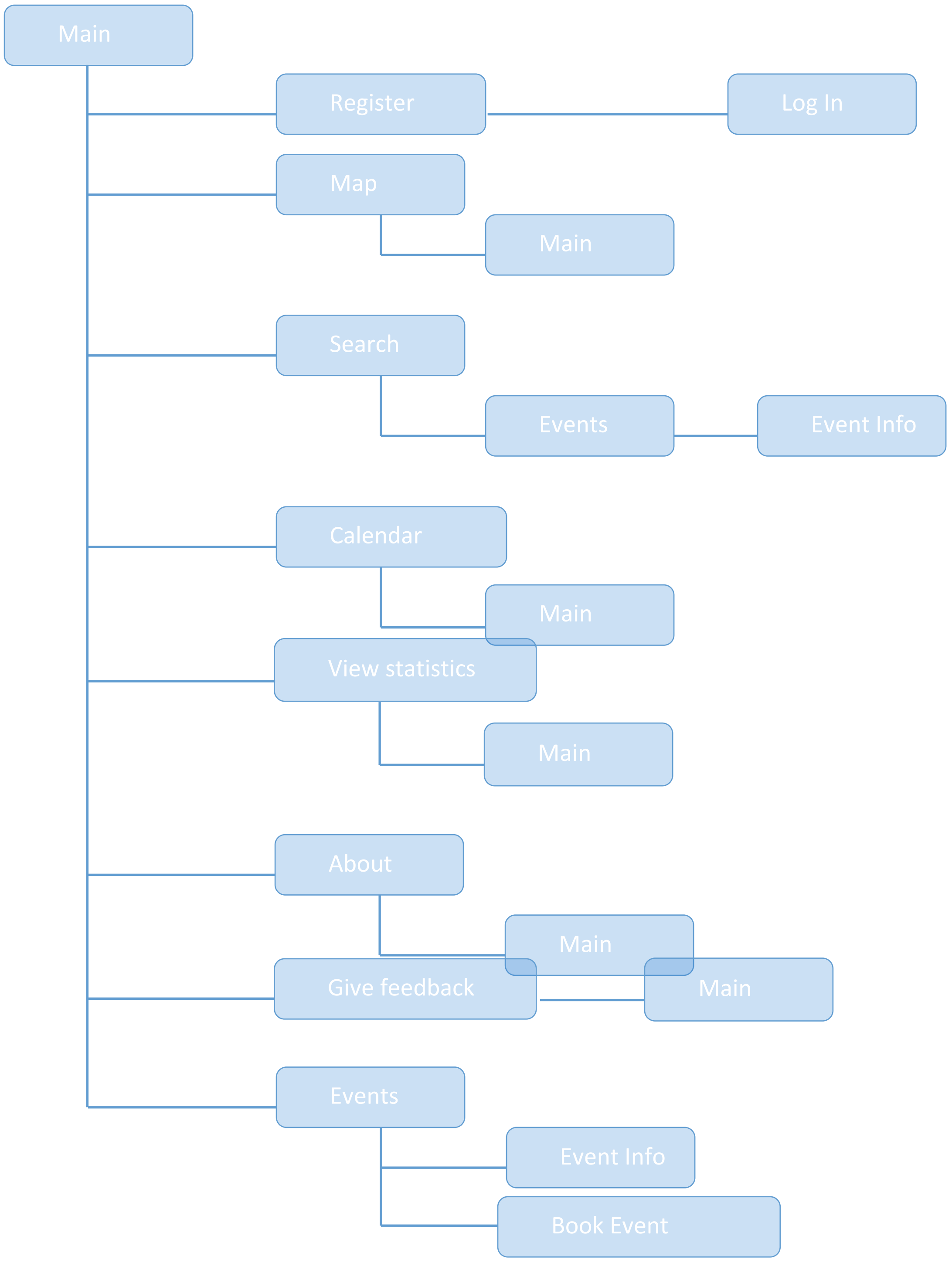


`

6. User interface design

6.1 Structure and navigation

Below is a simple diagram outlining the structure and navigation of our system. It highlights which pages are accessible from other pages and shows how users would find certain areas of the system.



`

6.2 Example user interfaces for use cases

In this section of the document, we present three example user interfaces from our system relating to different use cases: *view events*, *view single event*, *register*. Although only mock-up designs, they allow us to represent how the system is going to look and highlights the positioning of buttons, panels and text areas in our Java system.

Each user interface we have displayed in this section is accompanied with a description and navigational details of each interface as well a list of the Java Swing elements used to create the interface.

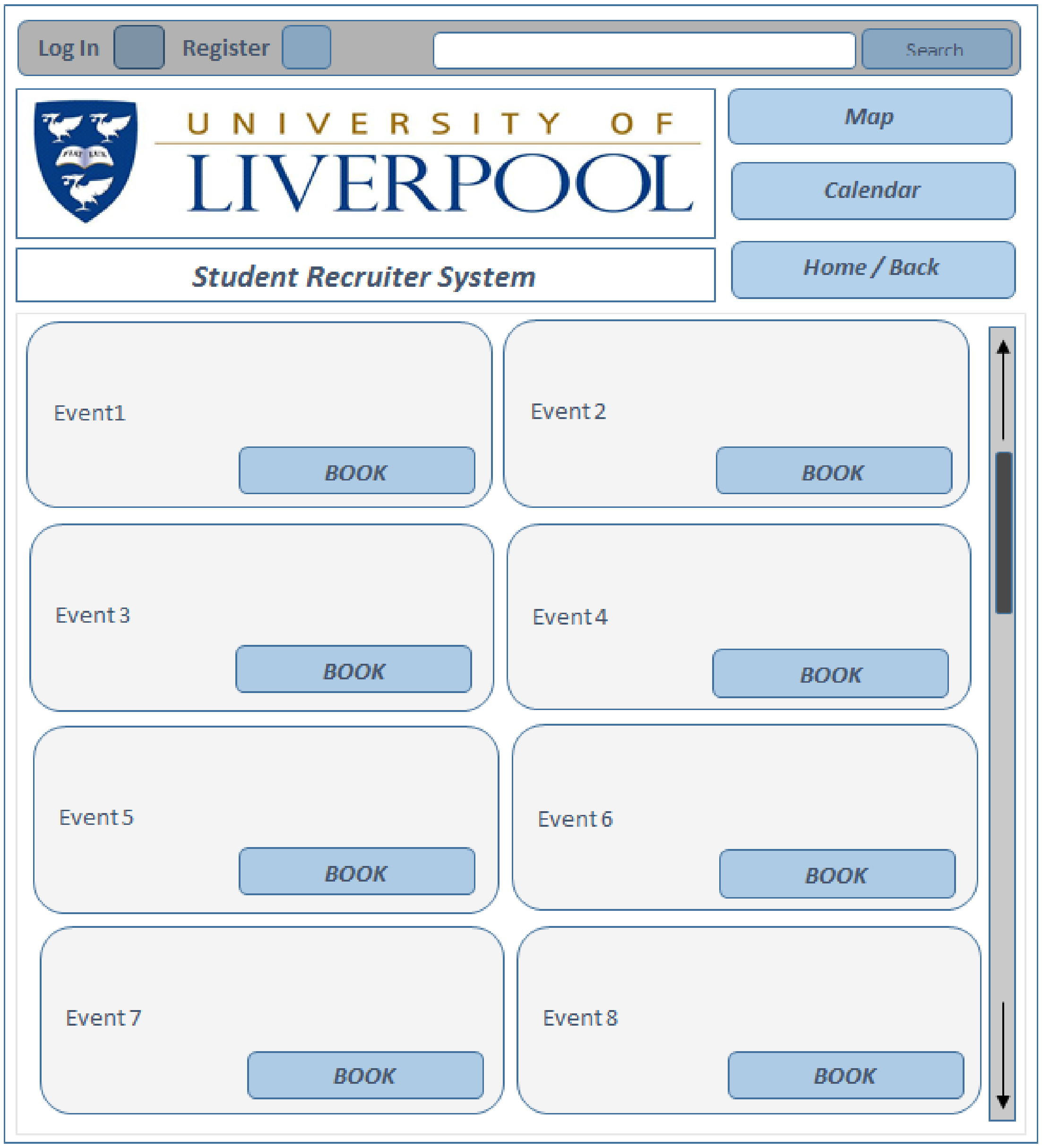
`

6.2.1 Use Case: View events

6.2.1.1 Description

This mock-up shows a page that allows users to view all the available events currently on the events listings. Users can view information on events from this screen and can also navigate the system to a number of pages using the buttons provided.

6.2.1.2 UI Prototype



`

6.2.1.3 Navigation

Navigation around our system is simple and consistent throughout most screens and pages. Users will be able to use the toolbar, found at the top of each screen, to either login, logout or register using the buttons provided as well as use the search bar in the top right corner.

Three buttons, at the right hand side of the page, are used for navigation around the system to useful areas which users will want access to from wherever they are on the system. Map, Calendar and Home/Back labels on these buttons show the user where these buttons lead.

Each event is displayed as a panel with an image which will be used as a button. Clicking the image will lead the user to the information page about that particular event. A Book button will be attached to each event which navigates users to an event page, with more detailed information about that event.

6.2.1.4 Issues

A suspected issue with this page is that gathering all the events and displaying them on the system in this way might cause us problems as if we try and make each event panel too complicated, using images with buttons layered on top. Java's GUI development restrictions might lead to a compromise with fewer images and more of `Swing`'s inbuilt components.

6.2.1.5 Elements

Login / Logout	-	JLabel / JButton
Search Bar	-	JTextField / JButton
Uni Logo	-	JLabel
Map Button	-	JButton
Calender Button	-	JButton
Home / Back	-	JButton
Event Panels	-	JPanel / JIcon / JIcon / JButton
Scroll Bar	-	JScrollBar or JScrollPane

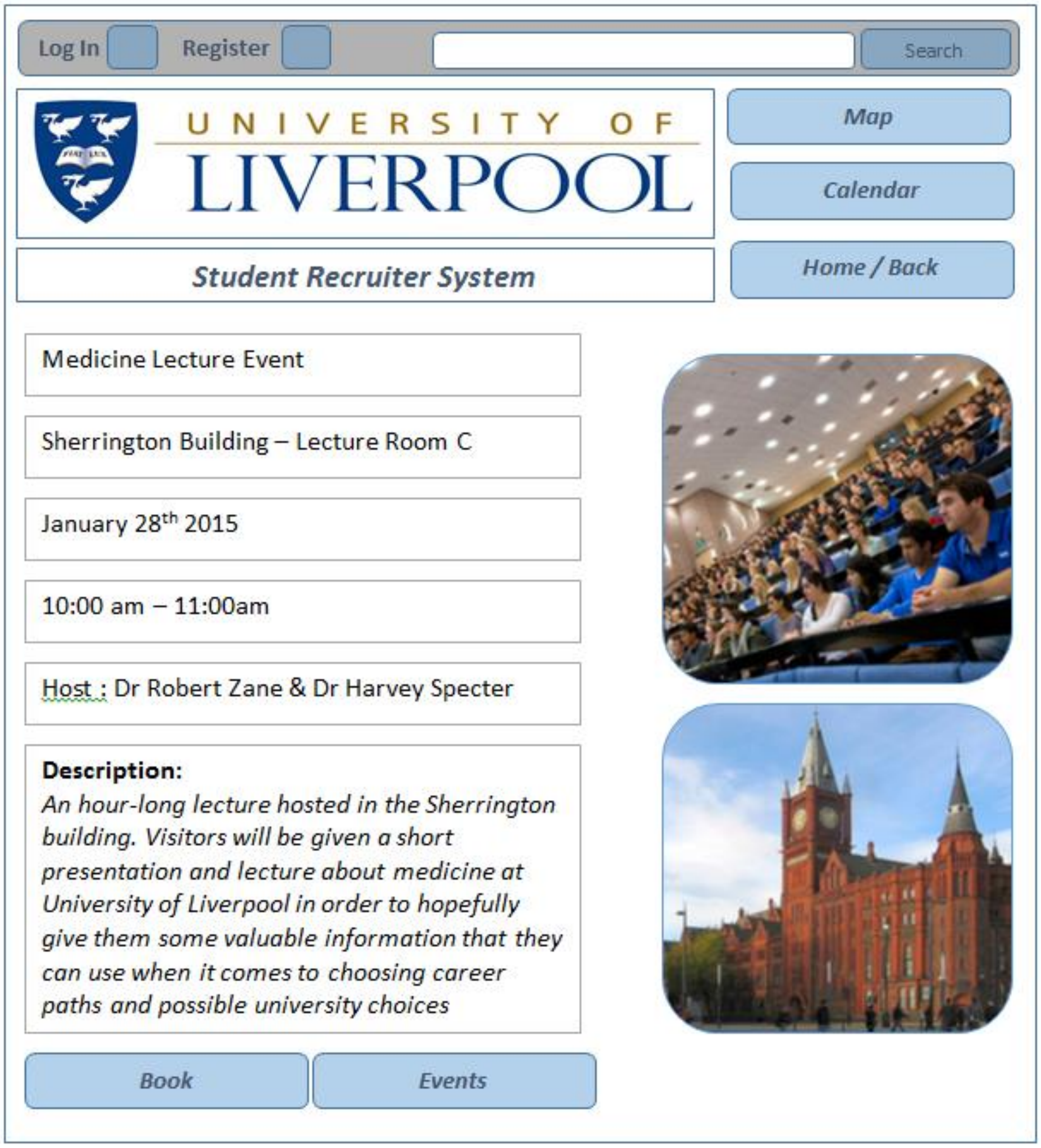
`

6.2.2 Use Case: View single event

6.2.2.1 Description

This simple mock-up shows a page or screen of our system where users will be viewing more detailed information about the events that are taking place. This screen will appear when a user clicks on an event. It displays some details regarding the event as well as a two related images. Every event will have a page or screen like this in our system.

6.2.2.2 UI prototype



Medicine Lecture Event

Sherrington Building – Lecture Room C

January 28th 2015

10:00 am – 11:00am

Host : Dr Robert Zane & Dr Harvey Specter

Description:
An hour-long lecture hosted in the Sherrington building. Visitors will be given a short presentation and lecture about medicine at University of Liverpool in order to hopefully give them some valuable information that they can use when it comes to choosing career paths and possible university choices

`

6.2.2.3 Navigation

Navigation around our system is simple and consistent throughout most screens and pages. Users will be able to use the “toolbar”, found at the top of each screen, to either login/ out or register using the buttons provided as well as use the search bar in the top right corner.

Three buttons, at the right hand side of the page, are used for navigation around the system to useful areas which users will want access to from wherever they are on the system. Map, Calendar and Home/ Back labels on these buttons show the user where these buttons lead.

This page is mainly used for displaying information so will have limited extra navigation options. Two buttons found at the bottom of the page will lead the user to a booking page or back to the events list page respectively.

6.2.2.4 Issues

No suspected issues with the development of this screen

6.2.2.5 Elements

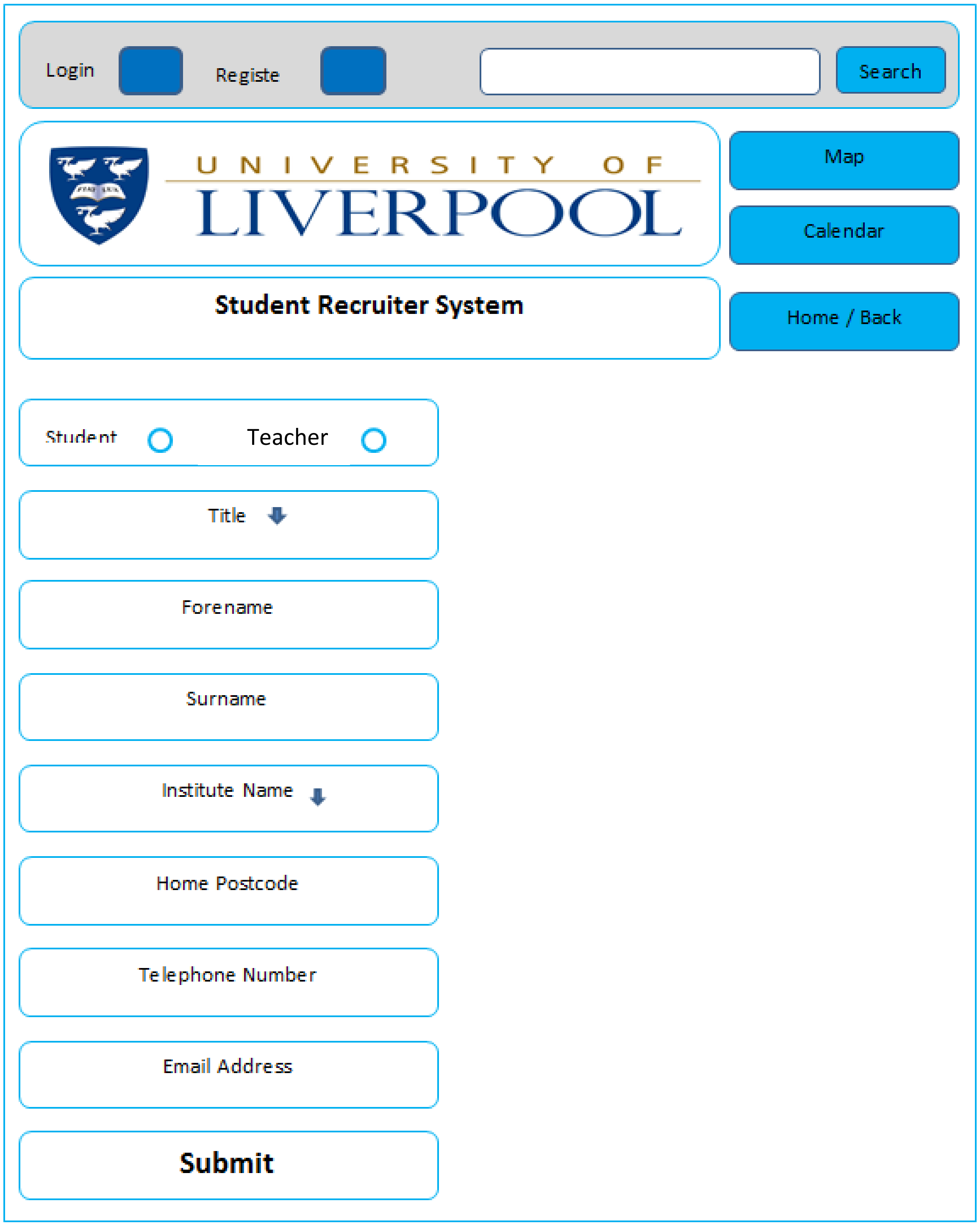
Login / Logout	-	JLabel / JButton
Search Bar	-	JTextField / JButton
Uni Logo	-	JLabel
Map Button	-	JButton
Calender Button	-	JButton
Home / Back	-	JButton
Event Info	-	JTextField and JTextArea
Book Button	-	JButton
Events Button	-	JButton

`

6.2.3 Use case: Register

6.2.3.1 Description

The Register page allows students and teachers to register for an account on the system. It is a standard registration form asking the user for their details. User input validation methods (see 5.4) are used to prevent invalid data entries.



`

6.2.3.2 Navigation

Navigation to this page is done by clicking on the Register Account button from the main page. The search can be used to navigate to other pages. The Map, Calendar, Home / Back button can be used to navigate away to the corresponding areas.

6.2.3.3 Issues

Home Postcode, Telephone number and Email address must be entered in the correct format, however we have included input validation methods and user hints to prevent this being a serious issue. With regards to the institute name, this should be implemented as a dropdown list of predefined entries which greatly reduce user input problems.

6.2.3.4 Elements

Login / Logout	-	JLabel / JButton
Search Bar	-	JTextField / JButton
University logo1	-	JLabel
Map	-	JButton
Calendar	-	JButton
Home / Back	-	JButton
SRS	-	JLabel
Student / Teacher	-	JRadioButton
Title	-	JLabel / JComboBox
Forename	-	JTextField
Surname	-	JTextField
Institute Name	-	JComboBox
Home Postcode	-	JTextField
Telephone Number	-	JTextField
Email Address	-	JTextField
Submit	-	JButton

`

6.3 Error messages warnings and support

6.3.1 Error Messages & Warnings

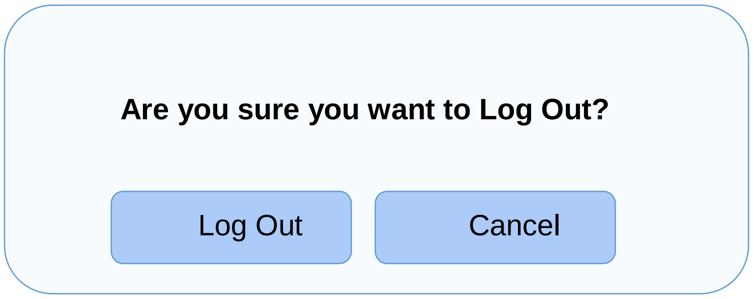
Our system will use simple and informative error messages consistently throughout in order to give the user information about mistakes during certain operations. All the dialog boxes will follow a certain format in order to keep the presentation and consistency at a high level. Keeping the error messages and warnings short and simple allows the users to quickly identify the problem the have made with hopes of fixing.

Warnings and error messages will appear in the same format, however any warnings displayed will contain a label indicating to the user what type of problem occurred

Example : User clicks Log Out

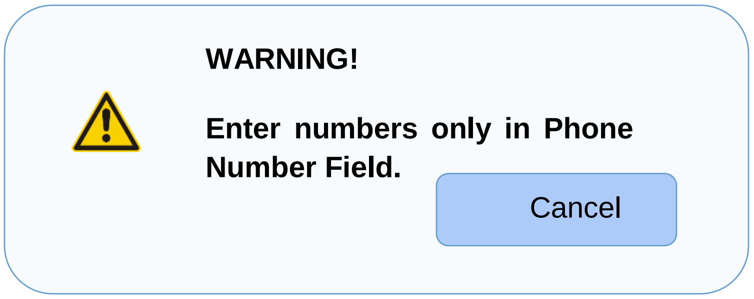
A dialog box will open asking the user if they are sure they want to log out in order to prevent users logging out accidently. The box will have two options Log Out and Cancel as well as a clearly displayed message "Are you sure you want to log out"

Below is an example dialog box that would open in the case of this example. This mock-up shows the simplicity of our warning and error message boxes. The user has two clear options: Log Out or Cancel



Are you sure you want to Log Out?

Example warning message:



WARNING!

Enter numbers only in Phone Number Field.

`

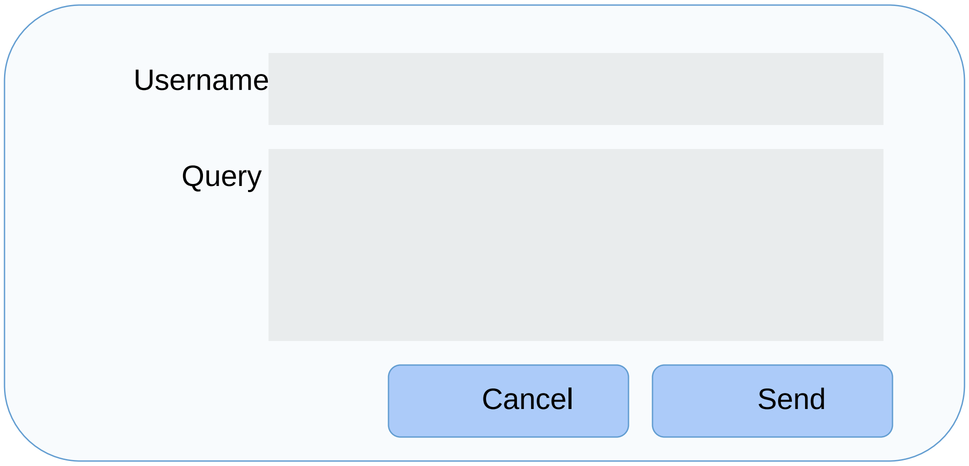
6.4 System Support and Help

System support will be available throughout the system and accessible from most of the user interfaces.

When users click on the "HELP" button, they will be directed to an interface which will display a series of information on system features and helpful advice about how to use the system properly. F.A.Q's will be included which will hopefully answer any quick queries users may have. This would be a useful feature for users with very low IT experience.

On this page there will also be a feature allowing users to send a queries to the system administrator via the help page. This will require the user to be logged in. The user will type their query into a text field and send it to the admin using a dialog box that will appear. This will then be forwarded to the system admin who will respond with an answer as soon as possible.

Example System Support Dialog Box:



`

6.5 User input validation methods

Throughout the system, there will be many occasions when user input will have to be validated in order to keep the data clean. Keeping erroneous data out of the database will keep the system efficient.

We will be using some existing Java methods to validate any data as well. Using these methods will allow us to check whether the users inputted data matches pre-defined criteria.

Below we have highlighted a few examples which use both existing and our own Java methods.

Example 1 – "Only numbers to be entered in Phone Number field"

Validation Rule: Users must enter numbers only in numeric fields
Validation Method: Java Method - isNumbers()

Using an existing Java method "isNumbers", allows us to check whether the user has inputted a number or not. This method will return true if a number is found or false if any miscellaneous characters are entered.

Example 2 – "Entered Phone Numbers must be exactly 11 digits"

Validation Rule: Users must enter exactly 11 digits in the phone number field.
Validation Method: Java Method countNumbers(int Number)

Using the Java method shown below we are able to count the amount of digits in a integer value.

```
static int countNumbers(int number)
{
   int total = 0;
   do
   {
      number = number / 10;
      numdgits++;
   } while (number > 0);
   return total;
}
```

`

7. Business rules and policies

- Feedback must be anonymous
- Only students and teachers from approved schools can make accounts on the system
- Highly information about students (such as address) should be highly secure to protect them from child abusers
- The SRS should comply with EU standards with respect to cookies, web components must undergo a cookie audit
- The SRS must comply with the UK Equality Act (2010) with respect to accessibility issues

8. Evaluation and testing

8.1 Usability testing

Once we have completed the implementation of our software system, we will conduct a series of usability tests in order to measure how efficiently and easily different users can complete tasks on our system. This should give us an idea of any design flaws or features that could be improved.

We will set up these usability tests by carefully creating custom scenarios, hopefully replicating realistic situations, in which different types of users will carry out tasks and instructions under observation. Observing users and measuring them against certain criteria will allow us to measure different levels of usability eg. *"Time taken to complete a task".*

After our usability tests are complete we can gather feedback from the users carrying out the tests in the form of data collection and questionnaires, again helping up improve any areas of the system if needs be. Gathering this kind of qualitative data will give us more descriptive and personal feedback about the usability of the system

Below we have outlined an example usability tests that we will be conducting during the testing stage of our system development.

Nota bene: criteria used to measure our test subjects will be defined at a later stage in development when we have a used the system ourselves and we can make realistic assumptions about time and speed of use etc.

8.1.1 Usability Test Example – Book Event (Pre-registered user)

8.1.1.1 Description

In this simple test users will be presented with the main user interface of our system and asked to log in and book a specified event

`

Users will already be registered therefore; the registration process will not be included in this usability test.

8.1.1.2 Tasks

Login to system
Book specified event *(e.g. Medical Lecture in Duncan Building)*

8.1.1.3 Start

Users will begin this test (not logged in) on the main interface of our system.

8.1.1.4 Finish

The test will be complete once the user has completed the event booking and the confirmation dialog box appears on screen.

8.1.1.5 Criteria and Feedback:

We will assess the users during this test on the time taken to complete the activities listed. The system will then gather feedback from the users in order to gather data about their experiences with the system.

Example Question – "Did you encounter any difficulties completing any tasks?"

`

8.2 Software Verification Testing

Once we have built our software we will run it through a series of tests to identify whether the system satisfies the requirements we have outlined during the first stage of development.

Verifying our software system in this way will help us identify any problems with the software. As we will be building and testing as a continuous process throughout the implementation stage of the development, we will always be keeping track of any problems and errors and fixing them on the move. This should hopefully lead to solid and robust system at the end. This kind of testing will be essential before delivering the final working build.

We will carry out a number of verification tests and checks once we have completed the implementation, following our use cases and requirements as a guideline in order to check our system completes operations that we have previously defined.

Below we have outlined a provisional verification test plan that the developers should conduct. It specifies which services our system should complete.

Operation	Expected result	**Test**
Register account	Successful	Register a new user account on system
Registration denied	Unsuccessful	Attempt to register account with unverified school
Log in	Successful	Log into system using valid account details
Unsuccessful log in	Unsuccessful	Attempt to log in using invalid username
View Events	Display a list of all events	Open events user interface
Apply for event	Successful	Book a place at an event with valid user details
Create event	Successful	Academic department creates a new event in available location
Change event details	Successful	Administrator changes time and location of a specified event
Remove event	Successful	Administrator removes an event from the system

`

8.3 Software Validation Testing

Validation is defined as the process of ensuring that the conceptual model is sufficiently accurate to represent the real world from the perspective of its intended uses. In other words, it has the physical concern of building the right model. It is applied only after the verification has been done.

This type of testing ensures that a product meets the specifications and needs of customers. However, in this project we do not have a customer as we will be defined our own user requirements and decided ourselves what the system should do.

It also allows us to see if our system behaves as intended when deployed in an appropriate environment, i.e. on computers running our Java program. Validation testing, against our original system specification, will show us if we have built our system correctly, efficiently and as intended.

Once we have built the system, we will carry out a series of validation tests in order to detect problems and bugs in the software.

Tests will be carried out in three ways:

- Testing whole system
- Integration testing (different components integrating)
- Unit testing (individual methods and classes)

We will write a series of test suites in the Java testing framework, Junit. This allows us to control our tests as well as conduct accurate and repeated test. Automatic testing is essential as manual testing is prone to human error.

We will write our test cases and test fixtures throughout the development stage once a clear outline of the different methods in our Java code comes to the fore.

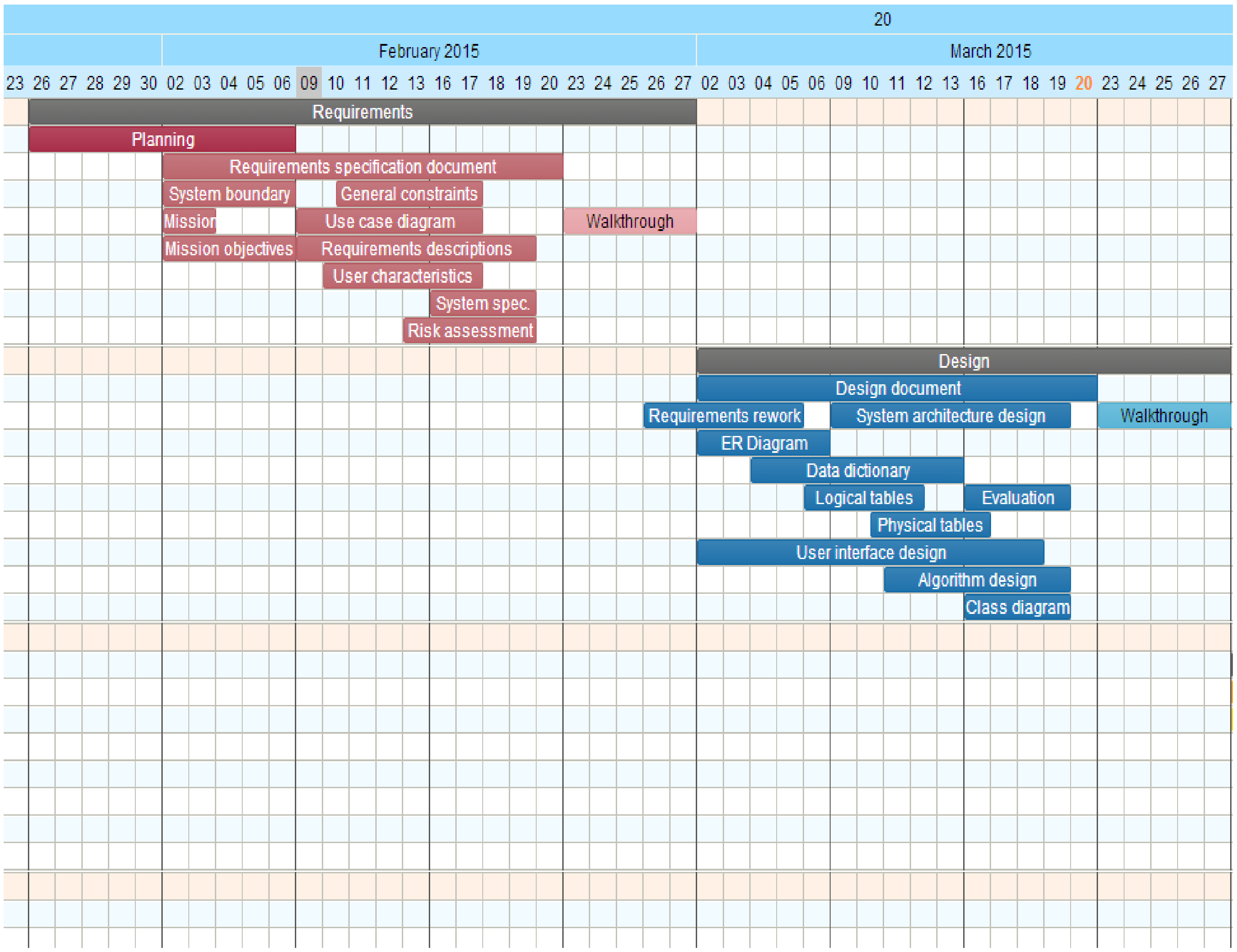
Test	Expected outcome	Description	Junit test methods used
actionListener test on Calendar Button	Successful	A test to test whether the actionListener method attached to our calendar button works as expected. This method call should open another user interface on our system when the button is clicked.	assertTrue()

`

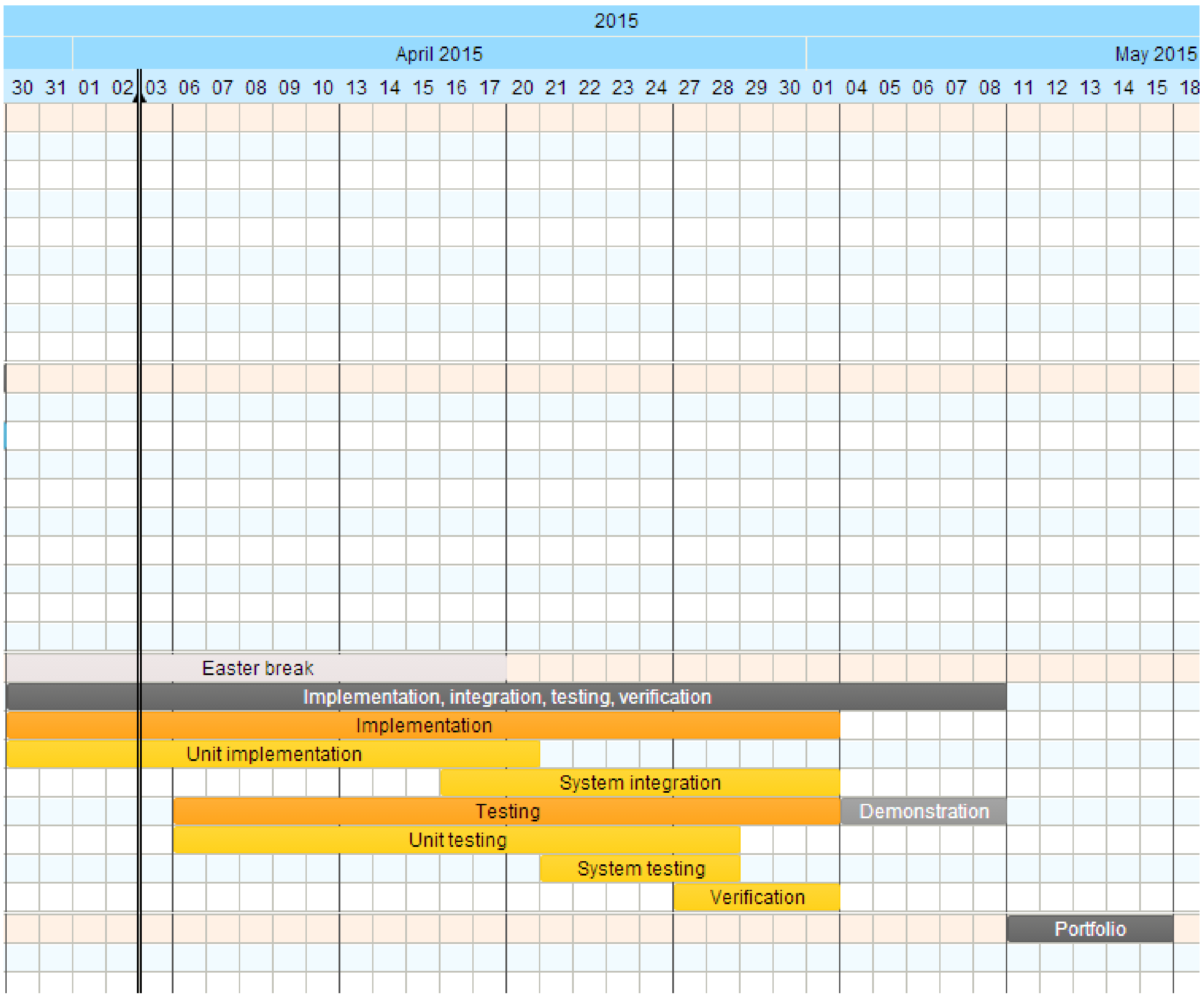
9. Planning

9.1 Gantt chart

The Gantt chart below (split over two pages) shows the progress of the project thus far and the tasks that will be done in the future. Thus far, the requirements and design stage of the software development process have been completed. This leaves implementation, integration, testing verification.



`


2015
April 2015
May 2015
30 31 01 02 03 06 07 08 09 10 13 14 15 16 17 20 21 22 23 24 27 28 29 30 01 04 05 06 07 08 11 12 13 14 15 18
Easter break
Implementation, integration, testing, verification
Implementation
Unit implementation
System integration
Testing
Demonstration
Unit testing
System testing
Verification
Portfolio

`

10. Bibliography

Cartographers (2015), StudentRecruiter System Requirements Specification

Connolly T.M. and Begg C.E. (2004). *Database Solutions: A step-by-step guide to building database*, 2nd edn. Pearson, Addison Wesley

Elmasri R. and Navathe S.B. (2011). Database Systems: Models, languages, design, and application programming, 6th edn. Pearson

Lops, Pasquale, Marco De Gemmis, and Giovanni Semeraro. 2011. "Content- Based Recommender Systems: State of the Art and Trends." In Recommender Systems Handbook, 73–105. Springer.

Manning, Christopher D, Prabhakar Raghavan, and Hinrich Schütze. 2008. Introduction to Information Retrieval. Vol. 1. Cambridge university press Cambridge.

Zito M. (2015). General Purpose Course Notes

Zito M. (2015). Group Software Project Lecture Material