

AQA Computing Unit 4 — School Room Booking System

Full Documentation

**Adam Matthew Blakey**, 7402

**Kings Priory School**, 39327

Pages

This document: <https://goo.gl/ILdhx7>

Live system backups: <http://goo.gl/iiHVq7>

Contents

[A: Analysis Section 10](#_Toc448907957)

[[A.1] Identification of the Problem 10](#_Toc448907958)

[[A.2] Description of the Current System 10](#_Toc448907959)

[[A.3] Identification of the Prospective Users 17](#_Toc448907960)

[[A.4] Interview with the Prospective Administrator 17](#_Toc448907961)

[[A.5] Questionnaire from the Prospective Users 19](#_Toc448907962)

[[A.6] Identification of User Needs 23](#_Toc448907963)

[[A.7] Acceptable Limitations 24](#_Toc448907964)

[[A.8] Data Sources and Destinations 24](#_Toc448907965)

[[A.9] Data Volumes 25](#_Toc448907966)

[[A.10] Analysis Data Dictionary 25](#_Toc448907967)

[[A.11] Data Flow Diagrams for the Proposed System 27](#_Toc448907968)

[[A.12] General Objectives of the Project 29](#_Toc448907969)

[[A.13] Specific Objectives of the Project 29](#_Toc448907970)

[[A.14] Realistic Appraisal of the Feasibility of Potential Solutions 30](#_Toc448907971)

[[A.14.1] Realistic Solutions 30](#_Toc448907972)

[[A.14.2] Justification of Chosen Solution 30](#_Toc448907973)

[[A.15] SWOT Analysis 31](#_Toc448907974)

[[A.16] SMART Targets 31](#_Toc448907975)

[[A.17] Comments from the Client 31](#_Toc448907976)

[B: Design Section 33](#_Toc448907977)

[[B.1] Overall System Design 33](#_Toc448907978)

[[B.1.1] System Outline Charts 33](#_Toc448907979)

[[B.1.2] Description of Modular Structure of the System 33](#_Toc448907980)

[[B.2] Validation Required 36](#_Toc448907981)

[[B.3] Database Design including Normalised Relations with Example Data 36](#_Toc448907982)

[[B.3.1] UNF (Un-normalised Normal Form) 36](#_Toc448907983)

[[B.3.2] 1NF (First Normal Form) 39](#_Toc448907984)

[[B.3.3] 2NF (Second Normal Form) 42](#_Toc448907985)

[[B.3.4] 3NF (Third Normal Form) 45](#_Toc448907986)

[[B.3.5] BCNF (Boyce-Codd Normal Form) 45](#_Toc448907987)

[[B.3.6] 4NF (Fourth Normal Form) 45](#_Toc448907988)

[[B.4] Definition of Data Requirements, Entity Relationships and Entity Descriptions 45](#_Toc448907989)

[[B.5] Description of Record Structure 48](#_Toc448907990)

[[B.6] The ‘openDB’ Class 51](#_Toc448907991)

[[B.7] Main Algorithm from the viewAllEvents.php page 52](#_Toc448907992)

[[B.8] Main Algorithm from the addEventImport.php Page 52](#_Toc448907993)

[[B.9] Sample of Planned SQL Queries 53](#_Toc448907994)

[[B.10] Identification of Storage Media 54](#_Toc448907995)

[[B.11] Identification of Suitable Algorithms for Data Transformation in Pseudo-Code 57](#_Toc448907996)

[[B.12] Human-Computer Interface (HCI) Design Rationale 57](#_Toc448907997)

[[B.13] User Interface (UI) Sample of Planned Data Capture and Entry Designs 59](#_Toc448907998)

[[B.14] UI Sample of Planned Valid Output Designs 60](#_Toc448907999)

[[B.15] Description of Measures Planned for Security and Integrity of Data 61](#_Toc448908000)

[[B.16] Description of Measures Planned for System Security 61](#_Toc448908001)

[[B.16.1] MD-5[http://security.blogoverflow.com/2013/09/about-secure-password-hashing/] 61](#_Toc448908002)

[[B.16.2] Salting and Hashing 61](#_Toc448908003)

[[B.17] Description of Login System 62](#_Toc448908004)

[[B.18] Description of Viewing Events 62](#_Toc448908005)

[[B.19] Description of Managing Events 64](#_Toc448908006)

[[B.20] Description of Managing Users 64](#_Toc448908007)

[[B.21] Detailed Description of Each File 64](#_Toc448908008)

[[B.21.1] addEvent.php [2] 65](#_Toc448908009)

[[B.21.2] addEventImport.php [2] 65](#_Toc448908010)

[[B.21.3] addEventSingle.php [2] 66](#_Toc448908011)

[[B.21.4] addHolidaySingle.php [3] 67](#_Toc448908012)

[[B.21.5] addRoomImport.php [3] 67](#_Toc448908013)

[[B.21.6] addRoomSingle.php [3] 68](#_Toc448908014)

[[B.21.7] addUserImport.php [3] 68](#_Toc448908015)

[[B.21.8] addUserSingle.php [3] 68](#_Toc448908016)

[[B.21.9] adminPanel.php [3] 69](#_Toc448908017)

[[B.21.10] editEventSingle.php [2] 69](#_Toc448908018)

[[B.21.11] editRoomSingle.php [3] 70](#_Toc448908019)

[[B.21.12] editUserSingle.php [3] 71](#_Toc448908020)

[[B.21.13] index.php [0] 72](#_Toc448908021)

[[B.21.14] login.php [|0|] 72](#_Toc448908022)

[[B.21.15] logout.php [1] 72](#_Toc448908023)

[[B.21.16] roomBookingFile.csv [N/A] 72](#_Toc448908024)

[[B.21.17] roomFile.csv [N/A] 72](#_Toc448908025)

[[B.21.18] staffBookingFile.csv [N/A] 73](#_Toc448908026)

[[B.21.19] staffFile.csv [N/A] 73](#_Toc448908027)

[[B.21.20] userFile.csv [N/A] 73](#_Toc448908028)

[[B.21.21] viewAllEvents.php [1] 73](#_Toc448908029)

[[B.21.22] viewAllHolidays.php [3] 74](#_Toc448908030)

[[B.21.23] viewAllRooms.php [3] 74](#_Toc448908031)

[[B.21.24] viewAllUsers.php [3] 74](#_Toc448908032)

[[B.22] Overall Test Strategy 75](#_Toc448908033)

[[B.22.1] Testing Strategy 75](#_Toc448908034)

[[B.23] Testing Plan 76](#_Toc448908035)

[[B.24] Comments from the Client 89](#_Toc448908036)

[C: Technical Solution Section 92](#_Toc448908037)

[[C.1] Included Files (as of 17/3/16) 92](#_Toc448908038)

[[C.1.1] class.openDB.inc.php 92](#_Toc448908039)

[[C.1.2] createTables.php 106](#_Toc448908040)

[[C.1.3] functions.php 110](#_Toc448908041)

[[C.1.4] header.php 115](#_Toc448908042)

[[C.1.5] loginInformation.php 116](#_Toc448908043)

[[C.1.6] menu.php 117](#_Toc448908044)

[[C.1.7] SQLDetails.php 119](#_Toc448908045)

[[C.1.8] styles.css 120](#_Toc448908046)

[[C.1.9] titleForFilenames.php 127](#_Toc448908047)

[[C.2] Page Files (as of 17/3/16) 129](#_Toc448908048)

[[C.2.1] addEvent.php 129](#_Toc448908049)

[[C.2.2] addEventImport.php 131](#_Toc448908050)

[[C.2.3] addEventSingle.php 142](#_Toc448908051)

[[C.2.4] addHolidaySingle.php 152](#_Toc448908052)

[[C.2.5] addRoomImport.php 156](#_Toc448908053)

[[C.2.6] addRoomSingle.php 161](#_Toc448908054)

[[C.2.7] addUserImport.php 165](#_Toc448908055)

[[C.2.8] addUserSingle.php 170](#_Toc448908056)

[[C.2.9] adminPanel.php 175](#_Toc448908057)

[[C.2.10] editEventSingle.php 178](#_Toc448908058)

[[C.2.11] editHolidaySingle.php 188](#_Toc448908059)

[[C.2.12] editRoomSingle.php 192](#_Toc448908060)

[[C.2.13] editUserSingle.php 197](#_Toc448908061)

[[C.2.14] index.php 204](#_Toc448908062)

[[C.2.15] login.php 206](#_Toc448908063)

[[C.2.16] logout.php 210](#_Toc448908064)

[[C.2.17] viewAllEvents.php 211](#_Toc448908065)

[[C.2.18] viewAllHolidays.php 221](#_Toc448908066)

[[C.2.19] viewAllRooms.php 225](#_Toc448908067)

[[C.2.20] viewAllUsers.php 229](#_Toc448908068)

[[C.3] Conversion Plan 234](#_Toc448908069)

[[C.4] Comments from the Client 234](#_Toc448908070)

[D: System Testing 236](#_Toc448908071)

[[D.1] Alpha: Included Files Testing 236](#_Toc448908072)

[[D.1.1] Alpha: Included Files Testing Plan 236](#_Toc448908073)

[[D.1.2] Outcomes from Alpha: Included Files Testing 244](#_Toc448908074)

[[D.2] Alpha: Viewable Pages Testing 247](#_Toc448908075)

[[D.2.1] Alpha: Viewable Pages Testing Plan 247](#_Toc448908076)

[[D.2.2] Outcomes from Alpha: Viewable Pages Testing 255](#_Toc448908077)

[[D.3] Comments from the Client 277](#_Toc448908078)

[E: System Maintenance 280](#_Toc448908079)

[[E.1] System Overview 280](#_Toc448908080)

[[E.2] Difficult Aspects of the System 280](#_Toc448908081)

[[E.3] A Sample of Detailed Algorithm Design 281](#_Toc448908082)

[[E.4] Procedure and Variable Description List 281](#_Toc448908083)

[[E.4.1] Variable Lists for The OpenDB Class 281](#_Toc448908084)

[[E.4.2] Variables Lists for Critical Pages to Functionality 284](#_Toc448908085)

[[E.5] Annotated Listings of Program Code and Tailoring 290](#_Toc448908086)

[[E.6] Comments from the Client 290](#_Toc448908087)

[F: Appraisal 292](#_Toc448908088)

[[F.1] User Feedback (and Beta Testing) 292](#_Toc448908089)

[[F.2] Analysis of User Feedback 296](#_Toc448908090)

[[F.3] Comparison of Project Performance Against Numbered General and Specific Objectives 302](#_Toc448908091)

[[F.4] Possible Extensions 304](#_Toc448908092)

[[F.4.1] System Security 304](#_Toc448908093)

[[F.4.2] Other Additions 305](#_Toc448908094)

[[F.5] Comments from the Client 305](#_Toc448908095)

[G: Notes 308](#_Toc448908096)

[[G.1] Spellings and Other Terms 308](#_Toc448908097)

[[G.2] Comments in Code 309](#_Toc448908098)

[H: User Manual b](#_Toc448908099)

[[H.1] A Brief Introduction and Installation Instructions b](#_Toc448908100)

[[H.1.1] Installing the Hardware and Software b](#_Toc448908101)

[[H.1.2] Setting up the Booking System e](#_Toc448908102)

[[H.2] Detailed Description of the Use of the Full System r](#_Toc448908103)

[[H.2.1] Types of Error Messages r](#_Toc448908104)

[[H.2.2] Recovery Procedures y](#_Toc448908105)

[[H.2.3] Points of Data Entry y](#_Toc448908106)

[[H.3] Comments from the Client ee](#_Toc448908107)

Analysis Section

A



# Analysis Section

## Identification of the Problem

Kings Priory School, located in Newcastle, currently uses a series of paper-based and computer-based room booking systems which regularly become out-of-sync with each other. This means that administrators have to spend hours each day updating and unifying the systems.

The data manager, Mr Jacobs, recently approached me to discuss ways to make a more concise, unified system; knowing that I was interested in computing, he thought that undertaking the project may be of interest to me. He believes that the way forward is to create a web-based application, so that staff and students can access the data wherever and whenever it is necessary. This will also mean that all the data is available instantly from one system.

Currently, all changes have to be compiled and added manually by Mr Jacobs, which is very time-consuming for him. On some days, 20–30 requests may be sent to Mr Jacobs to amend room bookings.

## Description of the Current System

Currently, Mr Jacobs is emailed by teachers, as modelled in the data flow diagram below, and then has to spend, potentially, hours emailing staff, updating systems and confirming with the original teacher about a room booking. Therefore, Mr Jacobs would like a system which will be able to be viewed by teachers (so that the emailing stage is eliminated) and will be able to hold data about all rooms on the school site (to eliminate the multiple-system checking stage).

The current system comprises of 4 sub-systems: Nova-T (for booking permanent rooms), SIMS (for booking temporary rooms), Outlook Calendar (for booking all rooms in the building, Tynemouth House) and a paper-based system (for out-of-period bookings [e.g. lunchtime, after school]).

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | |  |  | | *Nova-T showing an individual room’s repeated bookings.* | Nova-T showing repeated bookings for the entire site. | |  |  | | *Nova-T showing an individual teacher’s repeated bookings.* | *Nova-T showing a list of all repeated bookings for each room in a week.* | |
| *System 1 — The current Nova-T system (for booking permanent room bookings).* |

|  |
| --- |
| delete.png |
| *System 2 — The current SIMS system (for temporary room bookings).* |

|  |
| --- |
|  |
| *System 3 — The current Outlook Calendar system (for booking rooms in Tynemouth House).* |

|  |
| --- |
|  |
| *System 4 — The current paper-based system.* |

|  |
| --- |
| Current System Flowchart - New Page |
| *Overall system — A data flow diagram of the processes involved in the current system.* |

## Identification of the Prospective Users

The prospective users will be all pupils, staff and administrators of Kings Priory School — all will use the system, whether it be to view, request or amend events. Administrators will be able to update and amend events which affect all pupils and staff, whereas staff will only be able to amend events that affect themselves. Students will only be able to view the events.

The primary user will be Mr Jacobs (as the data manager) who will use the system to collate all data about room bookings into one system. After discussing the system with Mr Jacobs, the system will also be used by other users who will have varying levels of privileges: administrators should be able to amend rooms; teachers should be able to request to book rooms; and students should be able to view the room bookings.

This could be modelled by assigning **administrators** a user level of ‘**3**’, **teachers** a user level of ‘**2**’ and **students** a user level of ‘**1**’; in this model, users will be able to do what the levels below can do, but not what the levels above can do. This will be discussed in greater detail in the design section (section [B](#_Design_Section), page 33).

## Interview with the Prospective Administrator

To fully understand how the current system works and how the new system should work, I interviewed Mr Jacobs; I have typed out a transcript of the interview below:

* What is the current system?
  + We currently use a timetabling system called Nova-T that then links to another system called SIMS. Permanent rooms are booked using Nova-T; temporary rooms are booking using SIMS. Rooms in some buildings are booked using Outlook Calendar.
* What are the problems with the current system?
  + Only 2 admins that can book and change (Mr Jacobs and Mr Oldham).
* What are the beneficial features of the current system?
  + If room bookings are changed in SIMS, the booking also appears on the teacher’s timetable.
* What are the possible solutions?
  + Bespoke software.
  + Outlook Calendar (particularly used already).
  + SIMS.
  + Nova-T.
* What are the problems with implementing each of the possible solutions?
  + Training for staff may be time-consuming and difficult for bespoke software
  + Double bookings can occur in Outlook Calendar.
  + Bookings can be made for the wrong day very easily in SIMS.
  + The graphical user interface (GUI) in Outlook Calendar is not optimised for use in a business or school context.
  + Only 2 administrators can add and amend events in Nova-T and SIMS.
* What form will the output take?
  + It will be on a website where users can login and see their timetables and can book rooms.
* How often is the output required?
  + Continuously — it will automatically be generated for that specific user at that specific time.
* Will a printed output be required?
  + Yes, for timetables for teachers and pupils. Also, cover teachers will need their own temporary timetables and logins (also, temporary teachers’ logins should have an expiry date).
* What processes need to be performed on the data?
  + Overview of all rooms will need to be generated and exported at the start of each year (or each time that a teacher’s timetable is updated).
* How much data is there?
  + All data is held in plain text in a database — database size is not going to be bigger than 1MB.
* How often does the data change?
  + At the start of term, the data changes on a daily basis daily; also, during the building work, the rooms also change daily. When there is no building works and it is in the middle of a term, it is unlikely that any changes will be made.
* Are the changes to the data to be made at once, or can they be done in batches?
  + The data should be made at once and in batches (to save time, if needs be, but also to edit individual room bookings, if necessary).
* Is the system to be secure?
  + Yes, with passwords.
* Should there be limits to areas of the system for different users?
  + Yes, admins should be able to access more than teachers or students.
* Will any personal data be entered into the system?
  + School email addresses will be the only personal data.
* What hardware/software does the user have?
  + Web browser on school network computers.
* What hardware/software does the user need?
  + Web browser on school network computers.
* How much money is available for the new system?
  + Around £100–£150 is available for the purchasing of a Ubuntu Server running a LAMP (Linux, Apache, MySQL, PHP) server.
* Do you have any additional ideas for the system?
  + Requests generated from users are then approved by the network administrator.
  + Graphical system (highlighting red) for classrooms that are selected by a user.
  + Application programming interface (API) for calendars (implementation with Google Calendar, iCal and Outlook Calendar).
  + Routes generated for guests to get to classrooms.
  + Rooms should have a capacity parameter.
  + Lift access for disabled people.
  + Specific resources needed for the room should be able to be requested.
  + Whether or not multimedia devices are available in the room (for example, projectors).
  + Is the room set-out for a meeting or a lesson?
  + Buffer for out-of-period room bookings (so that they don’t overlap with adjacent room bookings).

## Questionnaire from the Prospective Users

I created and sent a questionnaire to some of the prospective users (the administrators and teaching staff). Below is a copy of the questionnaire and the responses.

|  |
| --- |
|  |
|  |
|  |
| *Questionnaire questions that were sent to some of the teachers at Kings Priory School.* |

|  |  |  |  |
| --- | --- | --- | --- |
| **Timestamp** | **How easy is it to create a TEMPORARY room booking, currently?** | **How easy is it to create a PERMANENT room booking, currently?** | **How frequently do you need to make/amend a room booking?** |
| 22/09/2015 15:26:05 | 1 | 1 | Once/twice per month |
| 23/09/2015 10:56:14 | 4 | 1 | Once/twice per month |
| 23/09/2015 11:03:23 | 2 | 1 | Several times per week |
| 23/09/2015 12:01:05 | 2 | 2 | Once/twice per month |
| 23/09/2015 13:45:31 | 2 | 0 | Once/twice per month |
| 23/09/2015 16:05:30 | 5 | 5 | Once/twice per month |
| 01/10/2015 09:41:57 | 2 | 3 | Several times per week |
| *See up-to-date responses:* [*https://goo.gl/3Pu33k*](https://goo.gl/3Pu33k) | | | |

## Identification of User Needs

Overall, Mr Jacobs would like a single web-based system that teaching staff can use to view and amend room bookings specific to them and students can use to view room bookings. Administrators should also be able view and amend all events.

The system needs to be available across the school network (and possibly the wider internet so that users can make adjustments from home, although it will be ultimately up to the network administrator to port-forward the appropriate ports). All user accounts should be secured using passwords with a suitable hash function. The system should also be very easy for the administrator to update and data must be able to be manipulated in batches. Additionally, the database used for the system should contain no redundant data.

Within the system, there should be the ability for teachers to submit a request for a room booking. Teachers should be able to recognise, clearly, the difference between proposed bookings from other teachers and bookings that have been approved by an administrator.

## Acceptable Limitations

Limitations:

* The system will not be able to account for events ran by 2 teachers (if a room booking is to be shared by 2 teachers or classes).
* The system will not be able to account for multiple-capacity rooms (where the capacity of a room may change [for example, in the main hall, where seats may or may not be laid out making 0 or 800 seats]).

## Data Sources and Destinations

Here’s a table demonstrating the data sources and destinations in the system:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Data Source** | **Process** | **User** | **Output** | **Destination** |
| SIMS | Export teacher timetable with room bookings (column 1 is teacher-code and the other columns are different periods throughout the week) as a CSV. | Administrator | Ultimately, read from database to see the room bookings. | Room booking database as a permanent booking. |
| Nova-T | Export teacher timetable with room bookings (column 1 is teacher-code and the other columns are different periods throughout the week) as a CSV. | Administrator | Ultimately, read from database to see the room bookings. | Room booking database as a permanent booking. |
| Outlook Calendar | Export teacher timetable with room bookings (column 1 is teacher-code and the other columns are different periods throughout the week) as a CSV. | Administrator | Ultimately, read from database to see the room bookings. | Room booking database as a permanent booking. |
| Teacher | Request a room for a specific time on the graphical interface of the system. | Teacher | “Request has been successfully sent to the administrator”. | The administrator(s). |
| Student | View the system from the graphical interface. | Student | Output an overview of rooms which the student is booked to be on and an overview of which rooms are free. | The student. |

This process can happen at any time and, if the system is implemented fully, every member of the school could be using the system several times per day.

## Data Volumes

When storing data in plain text into a database, relatively little amounts of data are produced. It is expected that the database size will not exceed 1MB for the length of a year (because the database on SIMS was only 1MB for an entire school year [2014–2015]). This approximation may change, but, unless photographs are implemented at a later date, very little storage will be required.

## Analysis Data Dictionary

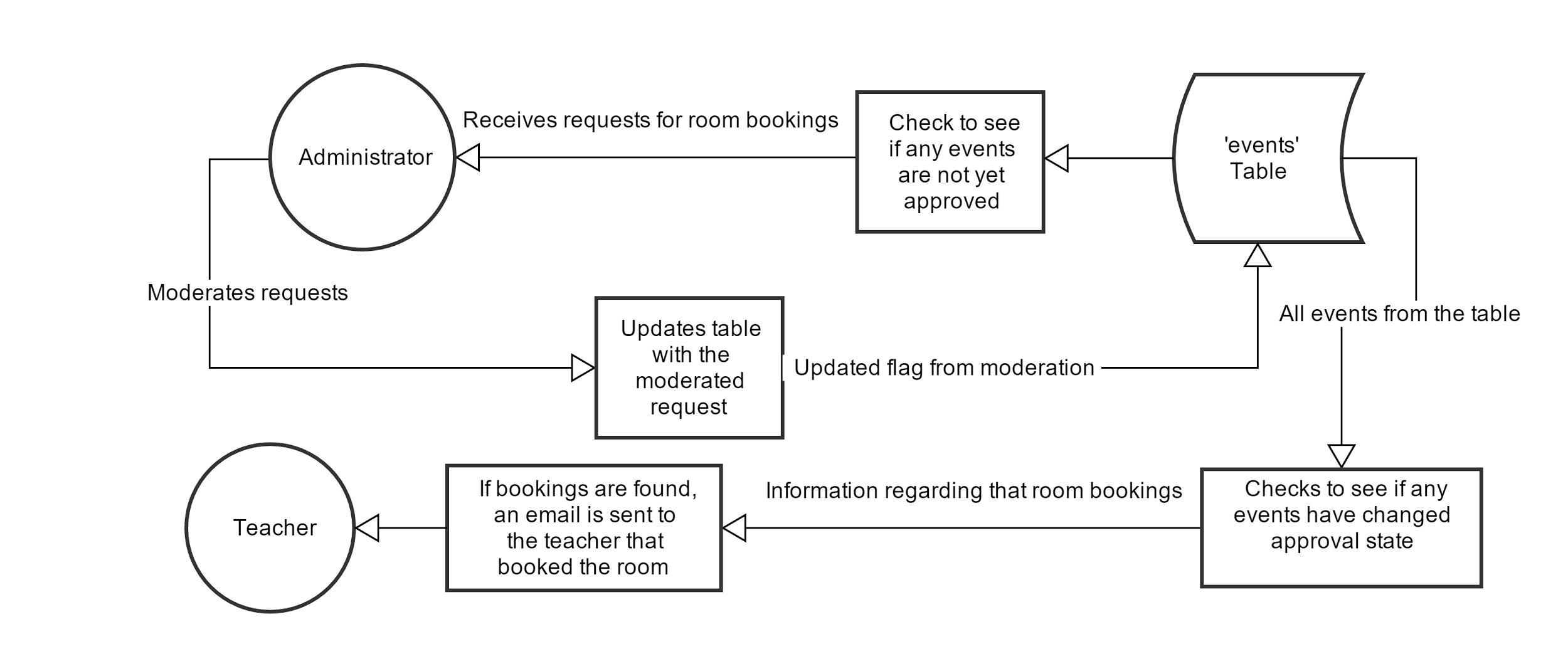
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **ownerName** | **ownerEmailAddress** | **ownerPassword** | **ownerUserID** | **ownerDateOfBirth** | **eventStartTime** | **eventName** | **eventDescription** | **eventApproved** | **eventModeratedBy** | **roomName** | **roomCapacity** | **attendeeNames** |
| Jeanette Patterson | jeanette.patterson@kps.woodard.co.uk | password123 | JPA | 23/4/1970 | 08:55, Monday, 16th of Novemeber 2015 | A2 Computing | The upper sixth computing class studying for AQA computing. | Yes | Jon Jacobs | C14 | 25 | Adam Blakey, Puya Mirkarini, Lauren Scott, Brandon Moss, Jack Myers |
| Judith Wills | judith.wills@kps.woodard.co.uk | hello789 | JWI | 14/2/1987 | 12:00, Thursday, 19th of November 2015 | Year 9a Mathematics | The top set year 9 class for maths studying for AQA maths. | Yes | Steve Oldham | CO23 | 20 | Billy Mason, Jon Grange, Sandra Harrington, Fred Pointon, Billy Smith |
| Philip Sanderson | philip.sanderson@kps.woodard.co.uk | philips4 | PSA | 12/11/1935 | 12:55, Wednesday, 18th of November 2015 | Additional Music Session | An additional music session run at lunchtimes to improve students’ learning. | No | Jon Jacobs | Seminar Room | 10 | Adam Blakey, Rhiannon Soulsby, Sophie Smith, Alicia Dodds, Elliot Gray |

## Data Flow Diagrams for the Proposed System

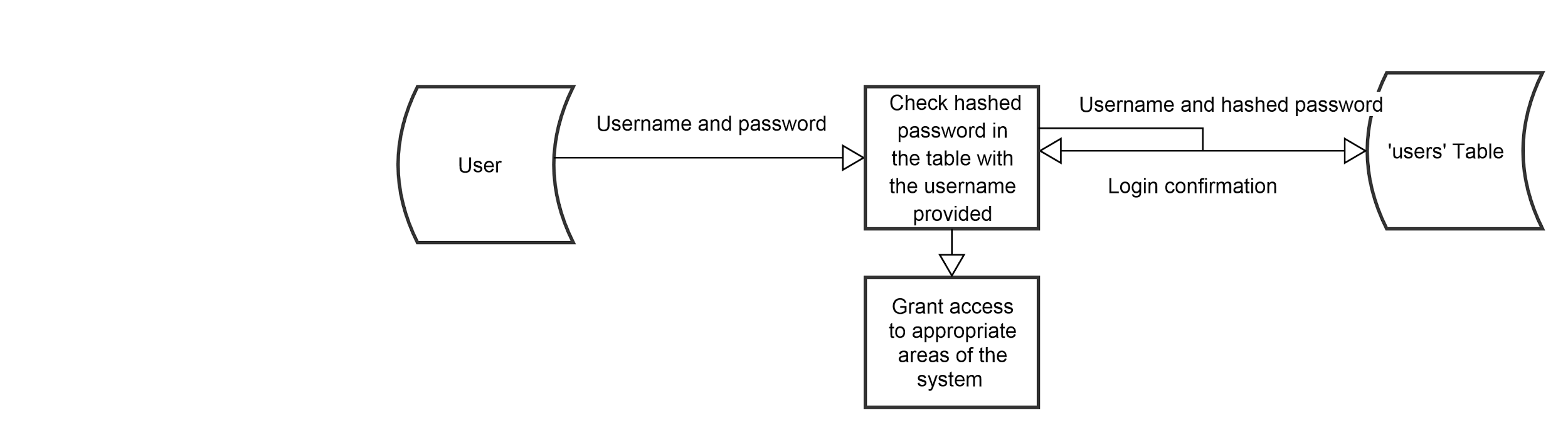
#### Context Diagram



#### Batch Data Entry



#### User Login System



## General Objectives of the Project

Mr Jacobs has asked me to create a school room booking system for Kings Priory School, which will allow access to students, teachers and administrators only; the system must be secured with a mechanism to only allow authorised users to access the system. The system should allow: Students to log into the system and view events for teachers; Teachers to log into the system and add events for themselves; Administrators to log into the system to add events, add rooms, add users and add holidays. The system should also allow users that are permitted to add events to add them by bulk-import (so each event doesn’t have to be added manually).

## Specific Objectives of the Project

Specifically, the new system will include:

1. The ability for timetabled lesson rooms to be exported from SIMS into the new system.
2. The ability for timetabled lesson rooms to be altered.
3. The ability for teachers to request new bookings rooms.
4. The system needs to be web-based.
5. Administrators should be the only people able to approve room bookings.
6. Administrators should be able to upload CSV files (or similar) to update data in batches.
7. The system should be secure, and only people with the correct authority should be able to view certain parts of the system.
8. The administrators should be able to edit individual users’ details and permissions for the site.
9. Should automate the importing of repeated bookings.
10. Should accept single and imported CSVs entries to be imported to the database.
11. Passwords should be necessary for accessing the system.
12. Passwords should not be stored in plain-text.
13. The database used for storing the data should be fully normalised.
14. When adding events, you should be able to pick a period (not a time).
15. The system should be intuitively laid out.
16. The system should be appealing to the eye.

## Realistic Appraisal of the Feasibility of Potential Solutions

### Realistic Solutions

#### Paper-based

Whilst time consuming, it is very easy for teachers to use, amend and understand; it also requires no power, so, if there were some sort of power-cut, the room bookings would still be accessible. Also, in this system, there is no way of the data being lost unless the paper is taken or destroyed by a fire (or equivalent) and will be available in a power shortage or when the intranet/internet is offline.

#### Pre-built Software (SIMS and Nova-T)

Alternatively, pre-existing software could be implemented that is already designed by professionals to deliver a solution that is made for the room bookings. Using a pre-built system would increase the likeliness of the system being more secure and may have been more vigorously tested in comparison to other systems.

#### A Bespoke Solution in PHP

A solutions that meets the client specification exactly and can be tailored to their needs. Also, with PHP, the system will be online and accessible to anyone with an internet connection.

### Justification of Chosen Solution

A solution in PHP has been chosen for:

* Versatility:
  + PHP a dynamic language that can have many different functions and can change according to various variables and conditions;
* Complexity:
  + PHP can deal with complicated expressions and supports object-oriented programming and MySQL connections;
* Accessibility:
  + PHP outputs HTML and so can be viewed on any web browser with an HTTP connection to the PHP server.

## SWOT Analysis

* Strengths:

Meets all of the client specification.

* Weaknesses:

Be aware that bespoke software developed by 1 person cannot be maintained if the person leaves.

Won't have been tested as rigorously as other solutions.

* Opportunities:

Allows for the possibility of rooms to be booked outside timetables for paid hiring (therefore generating a small profit for the school).

* Threats

Be aware that bespoke software developed by 1 person cannot be maintained if the person leaves.

The system doesn’t function in the same way as the previous system did.

## SMART Targets

SMART targets have been set throughout the objective sections (sections [A.12](#_General_Objectives_of) and [A.13](#_Specific_Objectives_of), page 29), where all of the time is within the time given for this project.

## Comments from the Client

The client has been in communication with me throughout the analysis section and has advised me on some fields that have been removed due to time constraints and also some data fields that were completely unnecessary.

**Mr Jacobs**: *“For security reasons it would be beneficial to have logins with passwords as security is more important than ease of access. This will mean that people will only have access to the system if I give them access by creating a login for them.”*



Design Section

B



# Design Section

## Overall System Design

The following tables describe which parts of the system will be involved with the inputs. This gives us a greater understanding of what tables are going to need to be connected to insert or update data in the database. The output column also gives us an idea of what kind of data would be usefully outputted from the data input.

### System Outline Charts

Processes concerning database communication are discussed in detail in the detailed description of each file section (section [B.21](#_Detailed_Description_of), page 64).

### Description of Modular Structure of the System

The system will be modularised primarily by splitting each module in different files (or pages, as often referred to in this documentation). Below is a structure chart displaying the overall structure of the system.

|  |
| --- |
| comp4_site_structure (1).png |
| *Structure chart displaying the overall structure of the system.* |

A key aspect that will be prevalent throughout the entire system will be the idea of users having user levels and this will determine which users can access different parts of the system. It will be split into 4 different categories, as shown by the table:

|  |  |  |
| --- | --- | --- |
| **User Level** | **Title** | **Description** |
| 0 | Guest | A user level created for users that are not logged in or do not have any permissions. |
| 1 | Student | A user level created for students that are able to only view events. |
| 2 | Staff | A user level created for staff that are able to request bookings of events. |
| 3 | Administrator | A user level created for admins that are able to approve requests for bookings. |

Each of these user levels will be assigned to a user. The guest user level will not necessarily be applied to an individual user (unless an administrator decides to allocate a user that user level [perhaps if the administrator don’t want someone to use the system but also don’t want to delete the user record]) but will mostly be used to describe users that are not logged in.

The structure of the system will differ for the type of person that is logged in. The menu structure for each is below with the user level in brackets. It should be noted that the ‘login’ button changes to ‘logout’ when a user is logged in and also that users with a higher user level than the one previous to them can access every item that they can but additional items too.

* **Guest (0)**
  + Home | Login
* **Student (1)**
  + Home | View All Events | Logout
* **Teacher (2)**
  + Home | View All Events | Add Events | Logout
* **Administrator (3)**
  + Home | View All Events | Add Events | Admin Panel | Logout

## Validation Required

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Validation Type** | **Validation Rule** | **Error Message** |
| Email address | Type check | Must contain: @ and a valid domain, although this validation is handled by HTML. | You must enter a valid email address |
| All fields | Range check | All fields must restrict the size of the values entered so that they do not exceed the field size in the database. | Please enter text that is less than [NUMBER] characters long. |

## Database Design including Normalised Relations with Example Data

Continuing from the record structure section above, we end up at a fully normalised data base in fifth normal form through various stages.

Below are sections describing the different normal forms (NF).

An underlined heading represents a key field.

### UNF (Un-normalised Normal Form)

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Owner Name** | **Owner Email Address** | **Owner User ID** | **Owner Date of Birth** | **Event Start Time** | **Event Name** | **Event Description** | **Event Approved** | **Event Moderated By** | **Room Name** | **Room Capacity** | **Attendee Names** |
| Jeanette Patterson | jeanette.patterson@kps.woodard.co.uk | JPA | 23/4/1970 | 08:55, Monday, 16th of November 2015 | A2 Computing | The upper sixth computing class studying for AQA computing. | Yes | Jon Jacobs | C14 | 25 | Adam Blakey, Puya Mirkarini, Lauren Scott, Brandon Moss, Jack Myers |
| Judith Wills | judith.wills@kps.woodard.co.uk | JWI | 14/2/1987 | 12:00, Thursday, 19th of November 2015 | Year 9a Mathematics | The top set year 9 class for maths studying for AQA maths. | Yes | Steve Oldham | CO23 | 20 | Billy Mason, Jon Grange, Sandra Harrington, Fred Pointon, Billy Smith |
| Philip Sanderson | philip.sanderson@kps.woodard.co.uk | PSA | 12/11/1935 | 12:55, Wednesday, 18th of November 2015 | Additional Music Session | An additional music session run at lunchtimes to improve students’ learning. | No | Jon Jacobs | Seminar Room | 10 | Adam Blakey, Rhiannon Soulsby, Sophie Smith, Alicia Dodds, Elliot Gray |

We first need to add a unique ID for the event. We will do this by assigning each event an incrementing numerical value.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Event ID** | **Owner Name** | **Owner Email Address** | **Owner User ID** | **Owner Date of Birth** | **Event Start Time** | **Event Name** | **Event Description** | **Event Approved** | **Event Moderated By** | **Room Name** | **Room Capacity** | **Attendee Names** |
| 1 | Jeanette Patterson | jeanette.patterson@kps.woodard.co.uk | JPA | 23/4/1970 | 08:55, Monday, 16th of November 2015 | A2 Computing | The upper sixth computing class studying for AQA computing. | Yes | Jon Jacobs | C14 | 25 | Adam Blakey, Puya Mirkarini, Lauren Scott, Brandon Moss, Jack Myers |
| 2 | Judith Wills | judith.wills@kps.woodard.co.uk | JWI | 14/2/1987 | 12:00, Thursday, 19th of November 2015 | Year 9a Mathematics | The top set year 9 class for maths studying for AQA maths. | Yes | Steve Oldham | CO23 | 20 | Billy Mason, Jon Grange, Sandra Harrington, Fred Pointon, Billy Smith |
| 3 | Philip Sanderson | philip.sanderson@kps.woodard.co.uk | PSA | 12/11/1935 | 12:55, Wednesday, 18th of November 2015 | Additional Music Session | An additional music session run at lunchtimes to improve students’ learning. | No | Jon Jacobs | Seminar Room | 10 | Adam Blakey, Rhiannon Soulsby, Sophie Smith, Alicia Dodds, Elliot Gray |

### 1NF (First Normal Form)

From the unnormalised data above, we can see that there are definitely repeating groups of data in some of the fields.

Also, after discussion with Mr Jacobs, he has said that some of the fields are irrelevant and aren’t needed (so the date of birth field has been removed and the owner name has been split up into first name and second name [you will see that each of the attendees’ names will be split up later too]).

In the tables that will follow, passwords have been assigned to users with the intention that they will be able to log into the system.

Additionally, I thought that it was better to convert the dates and times to a unix timestamp so that all the data for the time is kept in one property.

**Event Bookings:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Event ID** | **Owner First Name** | **Owner Last Name** | **Owner Email Address** | **Owner Password** | **Owner User ID** | **Event Start Time** | **Event Name** | **Event Description** | **Event Approved** | **Event Moderated By** | **Room Name** | **Room Capacity** | **Attendee Names** |
| 1 | Jeanette | Patterson | jeanette.patterson@kps.woodard.co.uk | password123 | JPA | 1447664100 | A2 Computing | The upper sixth computing class studying for AQA computing. | Yes | Jon Jacobs | C14 | 25 | Adam Blakey |
| 1 | Jeanette | Patterson | jeanette.patterson@kps.woodard.co.uk | password123 | JPA | 1447664100 | A2 Computing | The upper sixth computing class studying for AQA computing. | Yes | Jon Jacobs | C14 | 25 | Puya Mirkarini |
| 1 | Jeanette | Patterson | jeanette.patterson@kps.woodard.co.uk | password123 | JPA | 1447664100 | A2 Computing | The upper sixth computing class studying for AQA computing. | Yes | Jon Jacobs | C14 | 25 | Lauren Scott |
| 1 | Jeanette | Patterson | jeanette.patterson@kps.woodard.co.uk | password123 | JPA | 1447664100 | A2 Computing | The upper sixth computing class studying for AQA computing. | Yes | Jon Jacobs | C14 | 25 | Brandon Moss |
| 1 | Jeanette | Patterson | jeanette.patterson@kps.woodard.co.uk | password123 | JPA | 1447664100 | A2 Computing | The upper sixth computing class studying for AQA computing. | Yes | Jon Jacobs | C14 | 25 | Jack Myers |
| 2 | Judith | Wills | judith.wills@kps.woodard.co.uk | hello789 | JWI | 1447934400 | Year 9a Mathematics | The top set year 9 class for maths studying for AQA maths. | Yes | Steve Oldham | CO23 | 20 | Billy Mason |
|  | Etc. for other pupils. | | | | | | | | | | | | |
| 3 | Philip | Sanderson | philip.sanderson@kps.woodard.co.uk | philips4 | PSA | 1447851300 | Additional Music Session | An additional music session run at lunchtimes to improve students’ learning. | No | Jon Jacobs | Seminar Room | 10 | Adam Blakey |
|  | Etc. for other pupils. | | | | | | | | | | | | |

### 2NF (Second Normal Form)

We will now split the data into relations where the data relies only on dependant keys. However, ultimately, the data will be linked together into single events (and so each entity will eventually need to relate to an event record).

I will also only pick a couple records for each entity to demonstrate the normalisation process so that the point is still clear whilst not taking up an unnecessary amount of page space.

**Teachers:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **First Name** | **Last Name** | **Email** | **Password** |
| JPA | Jeanette | Patterson | jeanette.patterson@kps.woodard.co.uk | password123 |
| JWI | Judith | Wills | judith.wills@kps.woodard.co.uk | hello789 |

**Students:**

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **First Name** | **Last Name** | **Email** |
| 03PointonF | Fred | Pointon | 03pointonf@kps.woodard.co.uk |
| 02BlakeyA | Adam | Blakey | 02blakeya@kps.woodard.co.uk |

One thing to notice about the two tables above: They’re very similar! In the next normalisation stage, we shall group these tables together and use a user level to differentiate between teachers and students. We’ve also used the student’s ID and email address in the format as the one that they use on school computers (so that they can easily be found on other systems as well as this one). We will also be allowing students to log into the system to view events (so they will also need a password field). The updated table will look something like:

**Users:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **First Name** | **Last Name** | **Email** | **Password** | **User Level** |
| JPA | Jeanette | Patterson | jeanette.patterson@kps.woodard.co.uk | password123 | 2 |
| 02BlakeyA | Adam | Blakey | 02blakeya@kps.woodard.co.uk | password | 1 |

**Events:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **ID** | **Name** | **Start Time** | **Description** | **Room ID** | **Owner ID** | **Approved** | **Moderated By** |
| 1 | A2 Computing | 1447664100 | The upper sixth computing class studying for AQA computing. | C14 | JPA | Yes | JonJacobs |
| 2 | Year 9a Mathematics | 1447934400 | The top set year 9 class for maths student for AQA maths. | CO23 | JWI | Yes | SteveOldham |

Also notice that administrators could fit into the above user table, and let’s assign administrators with a user level of 3 (to tell them apart from other users).

**Users:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **First Name** | **Last Name** | **Email** | **Password** | **User Level** |
| JonJacobs | Jon | Jacobs | jon.jacobs@kps.woodard.co.uk | jonspassword | 3 |
| 02BlakeyA | Adam | Blakey | 02blakeya@kps.woodard.co.uk | password | 1 |

**Rooms:**

|  |  |  |
| --- | --- | --- |
| **ID** | **Name** | **Capacity** |
| C14 | C14 | 25 |
| CO23 | CO23 | 20 |

**Attendees:**

|  |  |  |
| --- | --- | --- |
| **ID** | **UserID** | **EventID** |
| 1 | 02BlakeyA | 1 |
| 2 | 03PointonF | 2 |

### 3NF (Third Normal Form)

There are no transitive dependencies, so the data is in third normal form, although further revisions to this model will be made as the design of the system is realised.

### BCNF (Boyce-Codd Normal Form)

The normalisation passes this normal form as no data depends on 2 properties that exist in another relation.

### 4NF (Fourth Normal Form)

The normalisation passes this normal form as there are no multivalued dependencies.

## Definition of Data Requirements, Entity Relationships and Entity Descriptions

Here’s an entity-relationship diagram used for modelling the database structure in the system. In the square brackets, you can see the data type assigned to each entity property.

|  |
| --- |
| comp4_entity_relationship_diagram.png |
| *An entity relationship diagram for the proposed system. (V43)* |

Whilst the entity relationships above would have been ideal, it is immediately clear that the system will not be developed in the correct length of time if the above relationships are to be used in the proposed system. I have discussed this with the end-user, Mr Jacobs, and he has allowed for me to simplify the relationships by removing the attendees’ relation and the data fields that concerned approving of events.

Below, there is a simpler entity relationship diagram which will do the majority of the tasks that the end-user, Mr Jacobs, needs and wants for his system.

|  |
| --- |
|  |
| *A simpler entity relationship diagram for the proposed system.* |

As it can be immediately seen, the simpler entity relationship diagram consists of fewer entities, fewer attributes in entities and fewer relationships between existing and current entities. This will create the basis for a suboptimal, but still useful, system.

## Description of Record Structure

Below, underlined property names indicate that the field is a primary key. A property in italics indicates that the property is optional.

|  |  |  |  |
| --- | --- | --- | --- |
| **Property Name** | **Description** | **Data Type** | **Justification for Data Type** |
| **General** | - | - | - |
| ID (all automatic IDs used) | Takes an incrementing numerical value. | Integer(10) | It is unlikely that there will ever be more than 9,999,999,999 records in any given table. |
| **Users** | - | - | - |
| userID | Will take a userID code of JPA (for teachers), for example, or 02BlakeyA (for students). | String(20) | The user IDs are unlikely to exceed 20 characters. |
| *emailAddress* | Takes a string, which should be pre-processed to ensure that the email address contains a syntactically valid domain name and an ‘@’ character. | String(254) | Email addresses can only contain a maximum of 254 characters. |
| password | Stores a 64 byte ‘salted’ password hash (with the salt being the userID). | String(64) | The hash function only returns a 64-byte string. |
| userLevel | Will hold a user level of 0, 1, 2 or 3 and will signify some hierarchical structure (the bigger, the more privileges). | Integer(1) | The user level only needs to be 1 digit in length because |
| *firstName* | Will store the user’s first name. | String(35) | The user is unlikely to have a first name longer than 35 characters. |
| *lastName* | Will store the user’s last name. | String(50) | The user is unlikely to have a surname longer than 50 characters. |
| **UserLevels** | - | - | - |
| userLevel | An integer representing a different type of user for each value. | Integer(1) | For the current system, only the numbers: 0, 1, 2 and 3 are needed; even in the future, it is unlikely that the user level will need more than 10 unique types. |
| title | A title to describe the current user level (e.g. ‘Student’ or ‘Administrator’). | String(40) | Future titles for user levels (if there are any in the future) are unlikely to exceed 40 characters. |
| description | A description to describe the current user level (e.g. ‘A user level created for students that are able to only view events.’ or ‘A user level created for admins that are able to approve requests for bookings.’). | String(200) | User level descriptions are unlikely to exceed 200 characters. |
| **Events** | - | - | - |
| ID | Will store the unique ID of the event, and will be an auto-incrementing value. | Integer(10) | It is unlike that there will be more than 9,999,999,999 events on the system. |
| name | Stores the name of the event. | String(10) | It is unlikely that the name of an event is going to exceed 10 characters. |
| ownerID | Stores the owner of the event’s ID | String(20) | The maximum allowed for a user ID in the Users table is 20 characters, so that has been kept consistent here. |
| startTimestamp | Will hold a 64-byte unix timestamp. | Integer(64) | A unix timestamp will only ever be a maximum of 64 bytes in size. |
| roomID | Holds the ID of the room that is booked. | String(10) | Kept consistent with the room ID’s maximum length in the Rooms definition. |
| **Rooms** | - | - | - |
| roomID | Holds the ID of the room. | String(10) | It is unlikely that the ID of the room will be longer than 10 characters long. |
| *name* | Holds the name of the room. | String(40) | It is unlikely that the name of a room is going to be longer than 40 characters. |
| capacity | Holds the capacity of the room. | Integer(4) | The room is unlikely to have a capacity of more than 9,999. |
| **Periods** | - | - | - |
| periodNumber | Holds the number of the period in the day. | Integer(2) | In the current system, there are only 7 periods, but in the future there is unlikely to ever be more periods than what 2 digits can accommodate for. |
| startTime | Holds the start time of the period. | Integer(4) | Is held as an integer so that comparisons between earlier and later times can be made easily later. |
| endTime | Holds the end time of the period. | Integer(4) | Is held as an integer so that comparisons between earlier and later times can be made easily later. |
| **Holidays** | - | - | - |
| ID | Holds the unique ID of the holiday. | Integer(10) | There is unlikely to be more than 9,999,999,999 holidays in the system. |
| *label* | Holds the label of the holiday (what the holiday is described as). | String(50) | The label of a holiday is unlikely to exceed 50 characters. |
| startTimestamp | Holds the start timestamp of the holiday. | Integer(64) | A unix timestamp will only ever be a maximum of 64 bytes in size. |
| endTimestamp | Holds the end timestamp of the holiday. | Integer(64) | A unix timestamp will only ever be a maximum of 64 bytes in size. |

## The ‘openDB’ Class

I designed the ‘openDB’ class to simplify connections to the database. I felt that splitting up the SQL queries into different functions and parameters made the code much more readable and easier to maintain. It also made each of the functions a lot easier to test individually and meant that, once I’d written it well once, I didn’t have to write it again! You can see more of it in the following sections (sections [C.1.1](#_class.openDB.inc.php) and [E.4.1](#_Procedure_and_Variable), pages 92 and 281, respectively)

## Main Algorithm from the viewAllEvents.php page

|  |
| --- |
| *fetch all events into an array*  *referenceTime ← the unix timestamp of the start of the Monday of the week selected*  *for (dayCounter ← 0 to 4)*  *print day of the week for dayCounter, and print the date for given reference time*  *for (periodCounter ← 1 to 7)*  *timeToCheck ← finds the time for the given period number in periodCounter*  *foreach (event result fetched as singleResult)*  *if an event exists:*  *fullSearchTime ← referenceTime + dayCounter\*(60\*60\*24) + timeToCheck*  *print details of the event name and either the room name or the teacher name (depending if the event was to be filtered by teacher or room).*  *else:*  *print a “Book now” button.* |

## Main Algorithm from the addEventImport.php Page

|  |
| --- |
| the user inputs:   * if the event should repeat weekly   + if the event should repeat into holidays   + what date the event should repeat up until   + and the imported file, seen below:   *importedFile ← an imported file as a two-dimensional array*  *foreach (importedFile as row)*  *foreach (row as cell)*  *find roomID, ownerID and eventName from the cell string*  *insert a record into the Rooms and Users tables if a room or user with the found ID does not exist*  *loop (through each week that the event may appear until the chosen end date has been reached)*  *insert into the Events table will all of the data that has been collected (including the calculating of the timestamp for insert), unless there is a holiday during the time* |

## Sample of Planned SQL Queries

The built class above will deal with the creation of SQL queries, but a query that may take place may be:

|  |
| --- |
| SELECT \* FROM Users WHERE UserID = `02BlakeyA` |

or

|  |
| --- |
| SELECT \* FROM Rooms WHERE Capacity > 20 |

The class, unless using a direct query (where the query is typed directly), just returns all columns from the table in order to make queries more simplistic. After using the query from the class, the programmer can then specify, using a relational array, which column they want from the result.

An example of this may be like so: If I wanted to get the first name of the user with the user ID ‘JPA’, I’d do something like below (see the description of the class in the system maintenance section (section [E.4](#_Procedure_and_Variable), page 281) for more details on how this works):

|  |
| --- |
| $bookingSystem->queryTable(“Users”, array(“userID”=>”JPA”), 1, “userID”)[‘firstName’] |

However, if I wanted to do a query for some more advanced function using the created database connection class, I’d have to use the direct flag and type out the query(although this doesn’t happen frequently). There is an example below where I get the event name with a given room ID (C14) which also has a start timestamp between 2 timestamps (again, see the description of the class in the system maintenance section (section [E.4](#_Procedure_and_Variable), page 281) to see how this works):

|  |
| --- |
| $bookingSystem->queryTable(NULL, "SELECT \* FROM Events WHERE `roomID` 'C14' AND `startTimestamp` BETWEEN 1457308800 AND 1457913600 ORDER BY startTimestamp ASC", NULL, NULL, NULL, false, true)[‘name’] |

## Identification of Storage Media

The table below is a copy of the one above from the description of record structure section (section [B.5](#_Description_of_Record), page 48).

|  |  |  |  |
| --- | --- | --- | --- |
| **Property Name** | **Estimated Maximum Number of Records** | **Maximum Size per Record (B)** | **Total Size of Records (B)** |
| **Users** | **-** | **-** | **-** |
| userID | 1,500 | 20 | 30,000 |
| *emailAddress* | 254 | 381,000 |
| password | 64 | 96,000 |
| userLevel | 1 | 1,500 |
| *firstName* | 35 | 52,500 |
| *lastName* | 50 | 75,000 |
| **TOTAL** | | | **636,000** |
| **UserLevels** | **-** | **-** | **-** |
| userLevel | 4 | 1 | 4 |
| title | 40 | 160 |
| description | 200 | 800 |
| **TOTAL** | | | **964** |
| **Events** | **-** | **-** | **-** |
| ID | 100 (100 teachers)\*6 (6 lessons per day)\*5 (5 days per week)\*39 (39 weeks per academic year) = 117,000 (estimated number of records per year). | 10 | 1,170,000 |
| name | 50 | 5,850,000 |
| ownerID | 20 | 2,340,000 |
| startTimestamp | 64 | 7,488,000 |
| roomID | 10 | 1,170,000 |
| **TOTAL** | | | **18,018,000** |
| **Rooms** | **-** | **-** | **-** |
| roomID | 150 | 10 | 1,500 |
| *name* | 40 | 6,000 |
| capacity | 4 | 600 |
| **TOTAL** | | | **8,100** |
| **Periods** | **-** | **-** | **-** |
| periodNumber | 7 | 2 | 14 |
| startTime | 4 | 28 |
| endTime | 4 | 28 |
| **TOTAL** | | | **70** |
| **Holidays** | **-** | **-** | **-** |
| ID | 20 (each year) | 10 | 200 |
| *label* | 50 | 1,000 |
| startTimestamp | 64 | 1,280 |
| endTimestamp | 64 | 1,280 |
| **TOTAL** | | | **3,760** |
| **COMPLETE SYSTEM TOTAL** | | | **18,666,894 (18.7MB)** |

## Identification of Suitable Algorithms for Data Transformation in Pseudo-Code

Data will be transformed in relatively simple ways. The extent of the most complicated data transformation algorithm will parse a CSV file. See the algorithm of the addEventImport.php page.

See the design of some of the algorithms above.

## Human-Computer Interface (HCI) Design Rationale

For the HCI of the system, I have chosen a standardised colour system, involving a minimal set of colours. The consistency of the design has been kept because of the use of a central cascading-style sheet (CSS).

After some user feedback, it was noticed that the adminPanel.php page was uneasy to navigate and wasn’t visually appealing. Below is a table containing a screenshot before and after the changes were made. During this process, duplication of text that links to the various pages was also removed.

|  |  |
| --- | --- |
|  |  |
| *Screenshot of system before design change.* | *Screenshot of system after design change.* |

It was thought that exit/back buttons were unnecessary as the user is only ever ‘one-level deep’ (so they can navigate back by using the menu).

No shortcuts have been made, but the tab key can be used to cycle through the fields on data entry pages.

Error messages have been carefully thought about and are talked about in detail in the system testing section (section [D](#_System_Testing), page 236).

## User Interface (UI) Sample of Planned Data Capture and Entry Designs

|  |  |
| --- | --- |
|  |  |
| *Planned UI of the Add Event: Import page, for example.* | *Planned UI of the Add Event: Single page, for example.* |

The other data entry pages, besides the ones above, will have a very similar layout.

## UI Sample of Planned Valid Output Designs

|  |  |
| --- | --- |
|  |  |
| *The output from the View All Events page will look something similar to the design above.* | *Even though some of the data fields are empty, users may look like this when displayed to the administrator on the View All Users page.* |

## Description of Measures Planned for Security and Integrity of Data

Integrity of data will be maintained by keeping the database in a high normal-form, with no non-key dependencies; everything will be kept to a unique key, which will not only help for keeping the integrity of the data, but also makes it far easier when searching for data from the database.

## Description of Measures Planned for System Security

To ensure maximum system security, I looked at several different hashing algorithms for storing passwords:

* MD-5
* SHA
  + SHA-1
  + SHA-256
* KECCAK (SHA-3)

### MD-5[<http://security.blogoverflow.com/2013/09/about-secure-password-hashing/>]

MD-5 is a hashing algorithm that is widely used, despite it being prone to collisions (where multiple inputs will return the same value from an MD5 function). In 2004, a theoretical attack was produced which allowed for weakening the pre-image resistance property of MD-5. In practice, the attack is way too slow to be useful. A more recent common use for MD-5 is, not for passwords, but for checking file integrity. This still does pose a small threat (because an attacker could replace a file with the same MD-5 hash value), although it is very uncommon that a ‘clean’ useful file will have a useful, ‘unclean’ counterpart.

### Salting and Hashing

Salting and hashing is a technique used by many login systems as well as other applications which rely on hashing. It is almost the same as regular hashing, but you add a known ‘salt’ to the string before you hash it, making the input string longer and more complicated, which in turn then makes it a lot harder for someone to hack into someone’s account by knowing only the hashed password string.

Something that has been available to hackers for a long time is rainbow tables for hashing algorithms. Rainbow tables are tables that contain relationships between a string and its hash value; this could mean that a hacker could reverse-lookup a hash value to find the initial string that would’ve been used, and enter the system by entering the user’s password. Salting and hashing helps to avoid this situation as, by adding a salt you don’t only create a unique string to hash, but you also create a hash value from a string which you would be unlikely to come across.

For example, if a user’s password is “password” and the user’s user ID is “02BlakeyA” (and we use “02BlakeyA” as the salt”), we can immediately see that “02BlakeyApassword” is a much less likely string to be hashed than “password” and so would be less likely to be found in a rainbow table.

## Description of Login System

The login system will be very secure and will use the salting and hashing technique to hash the passwords. It will take the user’s userID and password and generate an SHA256 hash from those inputs conjugated together into a single string.

The only disadvantage of this technique, is that the password will no longer be valid if the userID changes, although Mr Jacobs says that this won’t be a problem as it won’t happen very often. However, if it does need to happen, the password hash will have to be changed to something else (either a new user-defined password or a default password, such as: “changeme” or “password”).

## Description of Viewing Events

The view events page will work by, first, asking the user if they want to view events by room or by teacher and then will give them a drop-down box of either teachers or rooms depending on what the user clicked to begin with. Once an individual teacher or room has been selected, a timetable will then appear with events corresponding to that room or teacher.

By default, the week shown will be the current week, but will be able to select other weeks from a HTML5 week input box. The data concerning the week number and year will then be stored as a $\_GET variable and will be computed into a unix-timestamp when the reference time for a week is calculated.

All of the events will work off a given ‘reference time’, which will be calculated as midnight on the Monday of that week’s beginning. Only 1 event-based SQL query will be executed, but this result will then be stored locally and with then be searched through many times (one for each period of each day), probably using 2 nested for-loops. This will significantly improve performance on the site as the page will not have to communicate large queries several times.

|  |
| --- |
| referenceWeek ← a provided year and week number  referenceTime ← convertToUnixTimestamp(referenceWeek)  start ← get the timestamp of the Monday before the referenceTime at 00:00  end ← get the timestamp of the Friday at 23:59 (because that’s when the school week finishes)  SQLQuery ← SELECT \* FROM Events Where RoomID = [the one chosen by the user] AND startTimestamp is BETWEEN start and end |

The algorithm above works out, given a starting reference time (which will always be 00:00 on the Monday of a week), what events to show from the database. It first gathers all matching records from the database within the timeframe of 00:00 on the Monday through to 23:59 on Friday (the school week). This can be done relatively easily because the times and dates are all stored in the Unix Timestamp format.

When building the algorithm to carry out this task, I thought about many approaches. One approach was to ‘hard-code’ days of the week into the program and to call a subroutine for each period of each day to check if there was a resulting event at that time. I however felt that this approach would’ve created unnecessary duplication of code.

My next approach was to use 2 nested for loops to loop through, firstly, the days of the week. It would then loop through each period within that day. It was safe to say that the ‘for’ loop could be hard-coded with 5 days and 7 periods because the number of lessons in a week is not going to change. However, it would be relatively easy to change this should the situation arise (where all school weeks have been elongated).

Amongst these ideas I chose to go with the 2 nested for loops idea because of the ability to reuse code and also because the code appears to be very concise.

The above algorithms only deal with a single room for a week. I now had to think about how a teacher’s timetable could be composed and also how a full list of teachers and, separately, a list of rooms for a week could be composed. I then went about doing this by creating a large table with the columns as the periods and days of the week and the rows as teachers (or rooms, as appropriate).

An additional feature that will need to be added will be clickable ‘book now’ button. This will be available to administrators when viewing bookings by both rooms and teachers, but teachers won’t be able to book rooms on another teacher’s timetable but still will be able to book a room when viewing events by rooms.

## Description of Managing Events

One method of editing single events in the system will be to click on the event’s name in the view events page. This will then take you to a separate page which will populate fields with the values corresponding to that particular event. Clicking the ‘Save Details’ button, or similar, will then query the database to update that event with its corresponding event ID.

One problem that may occur whilst using this method is that there is very few ways of redirecting the user back to the page they were on previously to clicking on the event. One way of getting round this, would be to build an array with all of the $\_GET variables and then serialize the array into a new $\_GET variable. This could then be passed to the edit page and then unserialized when the user is redirected back to the initial page. This ensures that the user will have a continuant experience and won’t get frustrated with having to find the week number and room ID that they were on previously.

## Description of Managing Users

Users will only be able to change others’ data if they are an administrator. Also, an administrator will be the only type of user that will be able to view a list of all users on the system and are able to change their details. This especially helps if a users’ email address changes or if someone forgets their login password and need it to be reset to something else. On the page where all users are viewed, there will be an edit button to change a single users’ data.

## Detailed Description of Each File

Before I detail the functionality of each page, it should be noted that the numbers in the square brackets after each page name will represent the minimum user level needed to view that page. Any number encapsulated in ‘the modulus function’ (or |\_|) will be represent a page where the content is restricted to only that user level.

### addEvent.php [2]

This page will work as an intermediate page for accessing the add event by single and import pages.

### addEventImport.php [2]

This page is arguably the most data-processing-intensive on the entire system. The code on this page will require the following data:

* The date that the events should start from.
* If the event should repeat weekly.
  + The date that the event should repeat up until.
  + If the repetition should continue into holidays.
* The CSV file to be parsed by the system.

The code on the page will work out:

* What name to call the event (according to the CSV file).
* Who the owner of the event is and adds their userID to the booking (according to the CSV file) and if the user doesn’t exist in the User table, adds the user.
* What the starting timestamp of the event is (according to the CSV file). For this to be calculated:
  + The date will be calculated (according to the inputted dates by the user) along with a recurring weekly addition added to the event’s insert time (if the event repeats).
  + The time at which a particular period will start (according to a database table containing the relationships between period numbers and start times).
* What roomID of the event should be (according to the CSV file) and adds the room to the Rooms table if it doesn’t exist.

Once these data items have been inputted or calculated, the events will be inserted into the database. It is evident that the input size to this page could be enormous, and so insertions to the database may take a long time to execute (depending upon the host server’s hardware). Additional insertions to the database may be made beyond the events (for example, if a room doesn’t exist in the Rooms table, it will be added [as explained above]).

The output of the insertion will be visible by: Viewing the contents of the database directly; viewing the events on the viewAllEvents.php page (to be explained later); and by viewing the success messages below the form for insertion, which I will talk a little about now:

* The number of successful events will be shown.
  + These will then be listed below the count as the data in the field that the page was provided with in the CSV file (so that the user knows which strings from the CSV file have been imported successfully).
  + Events that have been inputted multiple times (over many weeks) will have their strings grouped together so that is easy to check if a range of dates have inserted successfully.
* The number of unsuccessful events will be shown.
  + These will then be listed below the count as the data in the field that the page was provided with in the CSV file (so that errors in syntax can be seen immediately).

### addEventSingle.php [2]

This page will be used for inserting individual events to the system.

The page will take the following as inputs:

* Event name.
* Teacher ID (from a drop-down of existing teachers).
* Insertion date.
* Period number for insertion.
* Room ID (from a drop-down of existing rooms).

As well as being able to type in the individual data fields manually, there is a circumstance where the data fields will be partially completed; from the viewAllEvents.php page, there will be a ‘book now’ button available to all teachers and administrators which allow them to book the selected slot (a given period and date) and the current room ID or teacher ID will be passed through so that can be partially completed, too.

For the partial completions, the timestamp of selected slot for insertion will be passed through; the page will then convert this timestamp to a separate date and time, and then will calculate what the correct period should be for that time. This will then be converted back to a timestamp when inserted to the database. Although this seems wasteful, it allows the user to clearly see which date and period they have chosen for their event.

Once the inputs have been inputted, the software will have to process the data:

* Period number and date will have to be converted to a single timestamp, involving a lookup for the insert time for the given period number.

No output will be provided, besides a success message, but the user may then be redirected back to the addEvent.php page (if there was no timestamp or serialised GET variables passed into the page) or the user may be redirected back to the viewAllEvents.php page (if there was a timestamp and a set of serialised GET variables). The serialised GET variables are the ones that were stored in the $\_GET array on the previous page and are used here to correctly redirect the user to the correct week and view on the viewAllEvents.php page.

### addHolidaySingle.php [3]

This page takes the following as inputs:

* Holiday label.
* Holiday start date.
* Holiday end date.

These dates are then converted to a timestamp and are inserted into the database.

Once the insert has completed, the user will be redirected to the viewAllHolidays.php page.

### addRoomImport.php [3]

This page will take a single uploaded CSV file as its input; the CSV should contain each of these columns in this order:

* Room name.
* Room ID.
* Room capacity.

A row will be created in the database table for each row in the CSV file.

### addRoomSingle.php [3]

This page will take the following as inputs:

* Room ID.
* Room name.
* Room capacity.

A row will be inserted into the database table “Rooms” for the details provided in the data fields. Once the insert has completed successfully, the user will be redirected to viewAllRooms.php.

### addUserImport.php [3]

This page will take a single uploaded CSV file as its input; the CSV should contain each of these columns in this order:

* First name.
* Last name.
* User ID.

The userLevel will be calculated from the syntax of the user ID and the password will automatically be set to password for the imported users containing the details above. The password (“password” at this stage) will be salted and hashed with the salt being the user’s user ID.

### addUserSingle.php [3]

This page will take the following as inputs:

* User ID.
* Email Address.
* Password.
* Password confirmation.
* First name.
* Last name.
* User level.

A row will be inserted into the database table “Users” for the details provided in the data fields. Once the insert has completed successfully, the user will be redirected to viewAllUsers.php.

### adminPanel.php [3]

This page does no processing, and instead just holds links as a quick route for administrators to get into different pages to do administrative tasks.

There are the following links on this page:

* View All Users [viewAllUsers.php].
* Add Users [no link].
  + By Import [addUserImport.php].
  + Manually [addUserSingle.php].
* View All Rooms [viewAllRooms.php].
* Add Rooms [no link].
  + By Import [addRoomImport.php].
  + Manually [addRoomSingle.php].
* View All Events [viewAllEvents.php].
* Add Events [addEvent.php].
  + By Import [addEventImport.php].
  + Manually [addEventSingle.php].
* View All Holidays [viewAllHolidays.php].
* Add Holidays [no link].
  + Manually [addHolidaySingle.php].

### editEventSingle.php [2]

This page will take the following as editable inputs, some of which will already be partially entered (as you are editing the event):

* Event name.
* Owner ID.
* Insert timestamp.
  + Insert date.
  + Insert period.
* Room ID.

For this page to function, there must also be an event ID specified as part of the $\_GET variables; this will be used to gather the data for auto-filling the data fields from the database table.

Once the event has been sent to be updated in the database, an update query will be sent which will update the columns with the new values where the event ID is kept consistent to the one provided as a $\_GET variable.

Alternatively, the user will be able to delete the event by clicking the button beside the update button; this will delete the current event from the database table by using its ID.

The user will then be redirected to the viewAllEvents.php page by making use of the serialised GET variables from the previous page.

### editRoomSingle.php [3]

The following data fields are taken as editable inputs for this page, some of which will already be partially entered (as you are editing the room):

* Room name.
* Room capacity.

For this page to function, there must also be a room ID specified as part of the $\_GET variables; this will be used to gather the data for auto-filling the data fields from the database table.

Once the room has been sent to be updated in the database, an update query will be sent which will update the columns with the new values where the room ID is kept consistent to the one provided as a $\_GET variable.

Alternatively, the user will be able to delete the room by clicking the button beside the update button; this will delete the current room from the database table by using its room ID.

The user will then be redirected to the viewAllEvents.php page by making use of the serialised GET variables from the previous page.

### editUserSingle.php [3]

The following data fields are taken as editable inputs for this page, some of which will already be partially entered (as you are editing the user):

* Email address.
* Password.
* Password confirm.
* First name.
* Last name.
* User level.

For this page to function, there must also be a user ID specified as part of the $\_GET variables; this will be used to gather the data for auto-filling the data fields from the database table.

Once the user has been sent to be updated in the database, an update query will be sent which will update the columns with the new values where the user ID is kept consistent to the one provided as a $\_GET variable.

Alternatively, the user will be able to delete the user by clicking the button beside the update button; this will delete the current room from the database table by using its room ID.

The user will then be redirected to the viewAllUsers.php page by making use of the serialised GET variables from the previous page.

Additionally, if you are logged in as an administrator and you try to change your user level from administrator, the system won’t allow you to do so. This is a security feature so that the system can’t be in a state of being inaccessible (unless, of course, the administrator forgets their password).

### index.php [0]

This page is a welcome page that will be displayed to all users, whether or not they are logged in. It is used only as a welcome screen for then being used to navigate to other more functional parts of the site.

### login.php [|0|]

This page is arguably the page that has to be the most secure (as it creates and destroys the cookies that allow access to the site).

The login page asks for 2 pieces of data:

* User ID.
* Password.

To help make it easier for the user, the user will be told when logging in if the user ID exists and whether or not the password is valid. Before the password is hashed and validated, the password must first be salted (due to the nature of the security system in place [salting and hashing security]); this means that the user ID must first be appended to the password string before the entire string is hashed and compared to the hashed password string in the database.

### logout.php [1]

The logout page will destroy the cookies (userID and userPassword cookies) by setting the expiry date of the cookies to 3 days in the past (so no sort of error could possibly occur with time-zone issues).

### roomBookingFile.csv [N/A]

This is a file used for testing which contains data for use on the addEventImport.php page and lists the data with respect to rooms. The data will be listed in columns ranging from Monday, period 1 to Friday, period 7. The rows will each contain bookings regarding a different room. The sorts on the rooms can be ignored, however, as each individual cell redefines which room is concerned.

### roomFile.csv [N/A]

This is a file used for testing which contains data for use on the addRoomImport.php page, listing data in the form:

* Room name.
* Room ID.
* Room capacity.

### staffBookingFile.csv [N/A]

This is a file used for testing which contains data for use on the addEventImport.php page and lists the data with respect to staff. The data will be listed in columns ranging from Monday, period 1 to Friday, period 7. The rows will each contain bookings regarding a different teacher ID. The sorts on the teacher ID can be ignored, however, as each individual cell redefines which teacher is concerned.

### staffFile.csv [N/A]

This is a file used for testing which contains data for use on the addUserImport.php page, listing data only from teachers in the form:

* First name.
* Last name.
* Teacher ID (or user ID).

### userFile.csv [N/A]

This is a file used for testing which contains data for use on the addUserImport.php page, listing data only from students in the form:

* First name.
* Last name.
* User code (or user ID).

### viewAllEvents.php [1]

This page will ultimately pull together all aspects of the system into this one page. This will more than likely be the most useful page of the site to any user, especially because the page is accessible for all logged-in users.

First, the user will be asked if they want to view events involving teachers or rooms. Then, depending on their selection, will select either a single teacher or a single room for which the events which are viewable will relate to the option chosen (if “[JPA] Jeanette Patterson” is chosen, it will show all events for that particular teacher, and if “SH5” is chosen, it will show all events for that particular room).

The default view for the room will be for the current week of events (if today’s date occurs on the first week of the year, all events for the selected teacher or room will be shown for the first week of the year, and the same is true for every other week). A week selection box will be used to select the different weeks for viewing.

### viewAllHolidays.php [3]

This page will display a table containing all holidays with the following column headings:

* Select (used for selecting multiple holidays).
* Label.
* Holiday start.
* Holiday end.
* Edit (links each row to editHolidaySingle.php with the appropriate holiday ID passed through).

There will also be an operations heading below with a “delete selected” button, allowing for selected holidays to be deleted easily.

### viewAllRooms.php [3]

This page will display a table containing all rooms with the following column headings:

* Select (used for selecting multiple holidays).
* Room ID.
* Name.
* Capacity.
* Edit (links each row to editRoomSingle.php with the appropriate room ID passed through).

There will also be an operations heading below with a “delete selected” button, allowing for selected rooms to be deleted easily.

### viewAllUsers.php [3]

This page will display a table containing all users with the following column headings:

* Select (used for selecting multiple holidays).
* User ID.
* User level.
* Email address.
* First name.
* Last name.
* Edit (links each row to editUserSingle.php with the appropriate user ID passed through).

There will also be an operations heading below with a “delete selected” button, allowing for selected users to be deleted easily.

## Overall Test Strategy

### Testing Strategy

The entire testing strategy mainly focuses on bottom-up testing (as each module is usually tested and used before the final system is tested and used). Also, the documented tests are nearly all black box tests (as they reflect how the user would interact with the modules in a real situation). White box testing was obviously carried out but it seemed unnecessary to document this as a lot of the boundaries of the data input usually fitted with the routes in the code.

#### Functional Testing

This will involve me testing the files that are included on every page; this will include the following files:

* class.openDB.inc.php
* createTables.php
* functions.php
* header.php
* loginInformation.php
* menu.php
* SQLDetails.php
* styles.css
* titleForFilenames.php

Some other pages will also ultimately involve functional testing, but these pages will actually always use the included pages above.

#### System Testing

This will involve me testing each page, page by page, on the system to make sure that all functionality functions as expected.

#### End User Testing

This will involve the end user, Mr Jacobs (and possibly his colleagues), testing the system in ways that satisfy their needs of the system.

## Testing Plan

Here, each page or function has at least one test so that every functionality of the system has been exploited. The first, second and third rows for each test ID represents: Typical, erroneous and extreme data, respectively.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test ID** | **Description** | **TEX (Typical, Erroneous, Extreme)** | **Reason for Test** |
| **Included Files Tests** | **-** | **-** | **-** |
| class.openDB.inc.php | **-** | **-** | **-** |
| class.openDB.inc.php.\_\_CONTRUCT | To test that the properties of the instance of the class are set correctly. | Construct a database class with a valid host, username, password and database name. | To test that the class can be successfully instantiated. |
| Construct a database class with no details for each of the parameters. | To check that the class can’t be instantiated without parameters for the database and MYSQL connection. |
| N/A | N/A |
| class.openDB.inc.php.createDB() | To test that the database of name specified is created correctly. | Create a database with a valid name. | To make sure that a database with a valid name can be created. |
| Create a database with an invalid name. | To make sure that a database with an invalid name can’t be created. |
| N/A | N/A |
| class.openDB.inc.php.connect() | Test that a connection to the database can be made successfully. | Connect to the database. | To make sure that the system can connect to the database. |
| N/A | N/A |
| N/A | N/A |
| class.openDB.inc.php.disconnect() | Test that a connection can be disconnected from the database. | Disconnect from the database. | To make sure that the system can disconnect from the database when a connection is no longer required. |
| N/A | N/A |
| N/A | N/A |
| class.openDB.inc.php.deleteFromTable() | To test that a record can be deleted from the table successfully. | Delete a record from a database table. | Make sure that a record can be deleted from a table. |
| Don’t specify any data for the deletion from the table. | To make sure that an error message is shown when parameters to the function are blank. |
| N/A | N/A |
| class.openDB.inc.php.createTable() | To test that tables can be created in the database successfully. | Create a table in the database. | To make sure that a table can be created in the database. |
| N/A | N/A |
| Create a table without any data fields. | To make sure that a table can still be made without any data fields in the table. |
| class.openDB.inc.php.queryTable() | To test that tables can be queried successfully, including use of the direct parameter. | Query a database table. | To make sure that a table can be queried. |
| Attempt to query a database table which doesn’t exist. | To make sure that errors when trying to query a table which doesn’t exist are handled correctly. |
| Query a table with no records. | To make sure that no errors occur when querying a table with no records. |
| class.openDB.inc.php.insertTable() | To test that records can be inserted into a specified table successfully. | Test that records can be inserted into the database table. | To test that records can be inserted into database tables. |
| Attempt to insert into a table without any data. | To make sure that blank rows that are attempted at being inserted are dealt with appropriately. |
| N/A | N/A |
| class.openDB.inc.php.updateTable() | To test that records can be updated in a particular table, given a where clause as a reference point for the record to be changed. | Test that a record can be successfully updated. | To make sure that rows from tables can be updated according to a reference field. |
| Test for when no data is specified. | To make sure that errors when no data is specified are handled correctly. |
| Test for where there is only 1 element of data in the “data” and “where” arrays. | To make sure that boundary amounts of data in the arrays are coped with. |
| createTables.php | **-** | **-** | **-** |
| createTables.php.1 | Check that all tables are created with the correct data types for each field. | Make sure that all tables are included in the file and are created correctly in the database. | To make sure that all tables are made without an issue. |
| N/A | N/A |
| N/A | N/A |
| functions.php | **-** | **-** | **-** |
| functions.php.getUserLevel() | Check that correct user levels are returned from the function. | See what the user level is for each type of user logged in with set cookies. | To make sure that only users of a high enough user level can access a page. |
| N/A | N/A |
| N/A | N/A |
| functions.php.formatTime() | Check that the time is formatted correctly. | Input sample times that vary in character length. | To make sure that sample times can be formatted without an issue. |
| N/A | N/A |
| N/A | N/A |
| functions.php.serializeGetVariables() | Check that the $\_GET variables are serialised correctly, even when there is just one $\_GET variable. | Test for 2 or more $\_GET variables. | To make sure that a selection of $\_GET variables can be serialised correctly. |
| Test for 0 $\_GET variables. | To make sure that a specified $\_GET array with no elements should be dealt with appropriately. |
| Test for when there is only 1 $\_GET variable. | To make sure that the $\_GET array with only one element is coped with. |
| functions.php.destroyCookies() | Test that cookies are destroyed, with no remnants of valid data left. | Valid cookies being destroyed. | To make sure that cookies can be destroyed without an issue. |
| N/A | N/A |
| Function attempts to destroy cookies that don’t exist. | To make sure that cookies that don’t exist can be deleted (by deleted, I mean that the expiry time is set in the past and the content of the cookie is removed). |
| functions.php.returnUserLevelError() | Test to see if the correct error message is shown. | Test for when a user isn’t able to see a page. | To make sure that a suitable message is given to the user trying to access the page, including a statement telling the user what sort of users can access the page. |
| Test for when no parameter value for the user level needed is provided. | To make sure that an error message is given when there is no parameter specified. |
| Test for when a user is able to see a page. | To make sure that users who are intended to see the content of the page can actually see the content of the page. |
| functions.php.saltAndHashPassword() | Test to see if the salt and hash function works, given a user ID and a password. | Test to see if a valid user ID and password is provided. | To make sure that passwords are correctly salted and hashed. |
| Test if no user ID is provided. | To make sure that passwords can’t be hashed without a user ID specified. |
| Test if no password is provided. | To make sure that no passwords is accepted by the function. |
| functions.php.isCurrentPage() | Test for checking if the current file is the same as the one passed in as a parameter. | Test for the page being and not being the one sought. | To make sure that pages can be correctly identified as being the one sought or not. |
| N/A | N/A |
| N/A | N/A |
| functions.php.generateTitle() |  | Test to see if the correct title is shown for a given page. | To make sure that the correct titles are shown for the pages. |
| See how the function behaves when there are two entries of a title for a given filename. | To make sure that erroneous duplications of filenames can be handled. |
| Test if the filename doesn’t exist in the array containing all known filenames related to a title. | To make sure that a default title is used instead of a unique title if there is no title specified. |
| header.php | **-** | **-** | **-** |
| header.php.1 | Test to see if the user login is checked, cookies are destroyed (if the user no-longer exists) and if the title is added correctly. | Test for the user ID and password combination being valid. | To make sure that valid ID and password combinations do not result in the cookies being destroyed. |
| N/A | N/A |
| Test for the user ID and password combination being invalid. | To make sure that the cookies are destroyed if the user ID and password combination is invalid. |
| loginInformation.php | **-** | **-** | **-** |
| loginInformation.php.1 | Test to see if the correct user ID and user level is shown at the top of each page (as long as the user is logged in). | If the user is logged in and if the user isn’t logged in. | To make sure that the correct login information is shown for users. |
| N/A | N/A |
| N/A | N/A |
| menu.php | **-** | **-** | **-** |
| menu.php.1 | Test to see if the correct menu is shown for the user logged in. | Test with a user with each user level. | To make sure that only the appropriate buttons for the user with a given user level are shown. |
| N/A | N/A |
| N/A | N/A |
| SQLDetails.php | - | - | - |
| SQLDetails.php.1 | Test that the openDB class is instantiated correctly. | Attempt to instantiate the class. | To make sure that the class instantiates with given details correctly. |
| N/A | N/A |
| N/A | N/A |
| styles.css | - | - | - |
| styles.css.1 | Check that there are no contradicting style statements. | Look at all places where the styles defined in this file are used. | To make sure that all statements are unique and reflect how the system should be styled. |
| N/A | N/A |
| N/A | N/A |
| titleForFilenames.php | - | - | - |
| titleForFilenames.php.1 | Test to make sure that all pages are associated with a title. | Test that all pages have a title. | Make sure that all pages have a title. |
| A file has more than one title associated with it (it isn’t unique). | Make sure that erroneous duplication of files is dealt with appropriately. |
| N/A | N/A |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test ID** | **Description** | **TEX (Typical, Erroneous, Extreme)** | **Reason for Test** |
| **Individual Page Tests** | **-** | **-** | **-** |
| addEvent.php | - | - | - |
| addEvent.php.1 | Make sure that the links to addEventSingle.php and addEventImport.php work correctly. | The user clicks the button. | To ensure that all buttons on the page link the user to where they need to go. |
| N/A | N/A |
| N/A | N/A |
| addEventImport.php | - | - | - |
| addEventImport.php.1 | Make sure that all buttons and pickers work correctly on the form, including validation. | The user enters the data for the import correctly. | To make sure that all of the inputs work correctly on the page. |
| The user enters the end date before the start date. | To make sure that an invalid date range can’t be entered. |
| N/A | N/A |
| addEventImport.php.2 | Make sure that the correct events are inserted for the options chosen on the form. | A date is selected for repeat for a set number of days, including exclusion on holidays. | To make sure that only the correct events are inserted to the database table. |
| An end date that is before the start date. | To make sure that erroneous date ranges are dealt with. |
| Insert events for one day only. | To make sure that events for one day are inserted correctly. |
| addEventSingle.php | - | - | - |
| addEventSingle.php.1 | Make sure that the form submission is rejected when incomplete fields have been submitted. | Fill in the details fully, including selection of all select fields. | To test that the form submits successfully. |
| Haven’t filled in any fields. | To test that one or more error message may be shown. |
| N/A | N/A |
| addEventSingle.php.2 | To test that inputs from other pages work. | Fill in the details fully, including selection of all the remaining select fields. | To test that the correct details are being filled in and that fields that aren’t already filled in can now be filled in, |
| Haven’t filled in any fields. | To test that one or more error message may be shown from partially filled in forms. |
| N/A | N/A |
| addHolidaySingle.php | - | - | - |
| addHolidaySingle.php.1 | Test to make sure that there is a validation check to see if the end date is before the start date. | Enter a valid start and end date for a holiday. | To test that valid dates are accepted. |
| Enter an end date before the start date. | To make sure that invalid date ranges are rejected. |
| Enter a start date and end date for the same date. | To make sure that single day holidays are allowed. |
| addRoomImport.php | - | - | - |
| \*addRoomImport.php.1 | Make sure that the form submits correctly and handles errors correctly. | A valid CSV file is uploaded. | To test that valid CSV files can be uploaded and processed without an issue. |
| Make sure that the form can’t be submitted without an uploaded CSV file. | To test that no errors will occur when the form is submitted without a CSV file. |
| A CSV file containing only 1 row is uploaded. | To test that a CSV file with only 1 row will upload and process without an issue. |
| addRoomSingle.php | - | - | - |
| addRoomSingle.php.1 | To test that the details can be entered and inserted to the database table successfully. | All fields are filled in with valid data. | To make sure that valid values are accepted. |
| The capacity field is filled in with non-numeric characters. | To make sure that invalid values for the capacity |
| When a capacity of 0 is entered. | To make sure that boundary values are accepted. |
| addUserImport.php | - | - | - |
| addUserImport.php.1 | Make sure that the form submits correctly and handles errors correctly. | A valid CSV file is uploaded. | To test that valid CSV files can be uploaded and processed without an issue. |
| Make sure that the form can’t be submitted without an uploaded CSV file. | To test that no errors will occur when the form is submitted without a CSV file. |
| A CSV file containing only 1 row is uploaded. | To test that a CSV file with only 1 row will upload and process without an issue. |
| addUserSingle.php | - | - | - |
| addUserSingle.php.1 | To check if password and password confirm are compared and hashed correctly when sent to the database. | Passwords that match. | To make sure that a row with a valid password combination is inserted to the database table. |
| Passwords that don’t match and passwords that are both empty (match, but are still invalid). | To make sure that a row with an invalid password combination are not inserted to the database table, and an error is given instead. |
| Passwords that are 1 or 32 characters long. | To make sure that passwords that are either long or short still successfully get entered to the database table. |
| adminPanel.php | - | - | - |
| adminPanel.php.1 | To test that all links to administrator-only pages work. | The user clicks the buttons. | To make sure that the users are linked to the correct places. |
| N/A | N/A |
| N/A | N/A |
| editEventSingle.php | - | - | - |
| editEventSingle.php.1 | To test that all data fields are filled in correctly. | The user clicks on an event from viewAllEvents.php and is directed to the edit page containing that event with all of the details filled in. | To make sure that the right data for the event is being shown on the edit page. |
| Wrong event ID is passed through. | To test that the edit page can deal with erroneous event IDs. |
| N/A | N/A |
| editRoomSingle.php | - | - | - |
|  | To test that all data fields are filled in correctly, including the disabled use of changing the room ID. | The user clicks on the edit button on viewAllRooms.php and is directed to the edit page containing details specific to the room ID specified. | To make sure that the right data for the event is being shown on the edit page. |
| The wrong room ID is passed through. | To test that the edit page can deal with erroneous event IDs. |
| N/A | N/A |
| editUserSingle.php | - | - | - |
|  | To test that all data fields are filled in correctly, including the disabled use of changing the user ID. | The user clicks on the edit button on viewAllUsers.php and is directed to the edit page containing details specific to the user ID specified. | To make sure that the right data for the event is being shown on the edit page. |
| The wrong user ID is passed through. | To test that the edit page can deal with erroneous user IDs. |
| N/A | N/A |
| index.php | - | - | - |
|  | To test that the text that tells you your user level works. | Log in as different levelled users and see if the appropriate text for each is shown. | To make sure that the welcome screen functions in the correct manner. |
| N/A | N/A |
| N/A | N/A |
| login.php | - | - | - |
|  | Make sure that cookies for users are only created for the correct user ID and user password combinations. | Log in as a user with the correct user ID and user password. | To make sure that valid logins are recognised and the system creates cookies successfully. |
| Log in with a valid user ID and invalid user password, and log in with an invalid user ID. | To make sure that the appropriate error messages are shown for each of these actions. |
| N/A | N/A |
| logout.php | - | - | - |
|  | Make sure that cookies are destroyed completely (the expiry date is set to the past AND the content of the cookie is deleted). | The user uses the logout.php page. | To make sure that the cookies no longer contain the user’s login details. |
| The user attempts to use logout.php without being logged in. | To make sure that the erroneous use is handled correctly. |
| N/A | N/A |
| viewAllEvents.php | - | - | - |
| viewAllEvents.php.1 | Make sure that all buttons work correctly for selecting the view for events and also for editing events. | For created events, rooms and teachers. | To test that the correct events are shown for what the user selects. |
| N/A | N/A |
| For 0 or 1 created events, rooms and teachers. | The system to cope and there to be an empty selection box for when there are no teachers or rooms. |
| viewAllEvents.php.2 | Make sure that the correct events are displayed for the given room ID or teacher ID. | Display a sample of events for given room IDs or teacher IDs. | To make sure that only the correct events for a given room ID or teacher ID are shown. |
| Don’t provide a valid room ID or teacher ID. | To make sure that an error message stating the problem is shown. |
| N/A | N/A |
| viewAllHolidays.php | - | - | - |
| viewAllHolidays.php.1 | Make sure that all records display correctly. | User views all holidays. | To make sure that all details for each holiday appears correctly. |
| A non-admin user tries to view the page. | To make sure that only administrators can view the page. |
| N/A | N/A |
| viewAllRooms.php | - | - | - |
| viewAllRooms.php.1 | Make sure that the “delete selected” operation functions correctly. | The user selects a set of 2 or more rooms. | To test that non-patterned selections can be made. |
| The user selects no rooms before operating the delete function. | To test that all of the rooms remain after the “delete selected” command is clicked. |
| The user selects only 1 or all of the rooms. | To test that a single or all of the entries can be selected at once. |
| viewAllUsers.php | - | - | - |
| viewAllUsers.php.1 | Make sure that clicking on the edit button and the email address takes the user to the correct places. | The user clicks the buttons/links. | To test that the links work sufficiently well. |
| N/A | N/A |
| N/A | N/A |

The test plan will be executed in full in the testing section (section [D](#_System_Testing), page 236).

## Comments from the Client

Mr Jacobs has been in communication with me throughout the design section (section [B](#_Design_Section), page 33) and has advised some changes that have been reflected in this section (as he suggested some of the tests and other minor changes).

**Mr Jacobs**: “*It is important that the database is normalised so that data becomes less redundant. I also spoke to Adam about simplifying some of the properties of the relations and decided that it would be best to go with the simpler design of the entities.*

*The HCI is very important; it makes it accessible to all of the user’s visibility needs. It could be improved that the blue and the black in the menu bar could be more contrasting colours so that users with visual impairments can easily read the menu text.*

*It is great that such a secure password-holding method has been incorporated into the system (salting and hashing). This means that the user’s passwords are much safer in the database than if they were store as plain-text or as a regular hash.*

*Adam also looked through the documentation for the existing systems (Nova-T in particular) to get a real understanding of the existing systems.”*



Technical Solution

C



# Technical Solution Section

## Included Files (as of 17/3/16)

Note: These pages do not have a screenshot as they do not necessarily produce a visual output (or change depending on where the pages are called from).

### class.openDB.inc.php

<?php

**class** **openDB**

{

*// Defines each of the properties in the class*

**private** $host; *// The host address of the MYSQL server*

**private** $username; *// The username of the login details of the MYSQL server*

**private** $password; *// The password of the login details of the MYSQL server*

**private** $database; *// The database name for this particular class instance*

**private** $openMYSQL; *// Variable used for holding the returned values of 'mysqli\_connect()' upon connection to the database*

*// Method used for assigning values to properties in the class and initially creating the database*

**public** **function** \_\_CONSTRUCT($host = "localhost", $username = "root", $password = "root", $database = "bookingSystem")

{

$this->host = $host;

$this->username = $username;

$this->password = $password;

$this->database = addslashes($database);

$this->createDB();

}

*// Method used for creating to the database*

**private** **function** createDB()

{

*// This method needs a local MYSQL connection because there is no database created yet*

$localMYSQL = mysqli\_connect($this->host, $this->username, $this->password);

*// Creating the database*

$query = "CREATE DATABASE ".$this->database;

*// Execute database creation query*

**if** (mysqli\_query($localMYSQL, $query))

{

**return** **true**;

}

**else**

{

**return** **false**;

}

mysqli\_close($localMYSQL);

}

*// Method for creating a MYSQL connection*

**private** **function** connect()

{

*// Connects to the database and then sets the default database to '$this->database'*

$this->openMYSQL = mysqli\_connect($this->host, $this->username, $this->password, $this->database);

mysqli\_select\_db($this->openMYSQL, $this->database);

*// If there is an error in connecting to the database, the function will return the appropriate value*

**if** (mysql\_error())

{

**return** **false**;

}

**else**

{

**return** **true**;

}

}

*// Method for closing a MYSQL connection*

**private** **function** disconnect()

{

$this->openMYSQL = mysqli\_close($this->openMYSQL);

*// If there is an error in disconnecting to the database, the function will return the appropriate value*

**if** (mysql\_error())

{

**return** **false**;

}

**else**

{

**return** **true**;

}

}

*// Method for deleting a record from a database table*

**public** **function** deleteFromTable($table, $data)

{

$this->connect();

*// Builds the query using a foreach loop to cycle through all the columns and values in '$data'*

$query = "DELETE FROM `**{**$table**}**`";

$foreachCount = 1;

**foreach**($data **as** $column=>$value)

{

*// If it's the first time through the loop, a WHERE is added, otherwise an AND is added*

**if** (($foreachCount == 1) **AND** (count($data) > 0))

{

$query .= " WHERE ";

}

**else**

{

$query .= " AND ";

}

$query .= "`**{**$column**}**` = '**{**$value**}**'";

$foreachCount++;

}

*// Executes query*

**if**(mysqli\_query($this->openMYSQL, $query))

**return** **true**;

**else**

**return** **false**;

$this->disconnect();

}

*// Method for creating a database table*

**public** **function** createTable($table, $data = **NULL**)

{

$this->connect();

*// Creates named table and automatically creates a column named 'ID' which is the primary key and will auto increment for each record inserted to the table*

$createTableQuery = "CREATE TABLE IF NOT EXISTS **{**$table**}** (ID INT(10) NOT NULL AUTO\_INCREMENT, PRIMARY KEY (ID))";

mysqli\_query($this->openMYSQL, $createTableQuery);

*// Data is executed in two halves so that an existing table may still have missing rows added*

**if** ($data != **NULL**)

{

*// Queries row-by-row so that missing rows can be added regardless of whether other rows exist*

**foreach**($data **as** $column)

{

*// Initialise variables to avoid errors*

$fullColumn[0] = "";

$fullColumn[1] = "";

*// Checks if the column to be added is a primary key column and splits the string up if it is*

**if**(strpos($column, "PRIMARY KEY"))

{

$fullColumn = explode(",", $column, 2);

$column = $fullColumn[0];

$fullColumn[1] = str\_replace(", ", "", $fullColumn[1]);

}

*// Adds the column to the database*

$alterTableQuery = "ALTER TABLE **{**$table**}** ADD **{**$column**}**";

mysqli\_query($this->openMYSQL, $alterTableQuery);

*// Checks if the column to be added is a primary key column drops the other primary column if it is*

**if**(strpos($fullColumn[1], "PRIMARY KEY"))

{

mysqli\_query($this->openMYSQL, "ALTER TABLE **{**$table**}** DROP PRIMARY KEY, CHANGE ID ID INT(10) NOT NULL");

mysqli\_query($this->openMYSQL, "ALTER TABLE **{**$table**}** DROP `ID`");

mysqli\_query($this->openMYSQL, "ALTER TABLE **{**$table**}** **{**$fullColumn[1]**}**");

}

}

}

$this->disconnect();

}

*// Method to query a database table*

**public** **function** queryTable($table, $data = "\*", $limit = **NULL**, $order = "ID", $sort = "ASC", $debug = **false**, $direct = **false**)

{

$this->connect();

*// If direct is true, we run a direct SQL query from the data variable*

**if** ($direct)

{

**if** ($debug)

**echo** $data;

$result = mysqli\_query($this->openMYSQL, $data);

**if** (!$result)

{

**return** **false**;

}

**else** **if** ($result->num\_rows == 1) *// Returns a single array if there is only one record returned*

{

**return** mysqli\_fetch\_array($result);

}

**else** **if** ($result) *// Returns a multi-dimensional array when there's 2 or more records*

{

**while**($row = mysqli\_fetch\_array($result))

{

$output[] = $row;

}

**if** (isset($output))

**return** $output;

**else**

**return** **false**;

}

}

**else**

{

*// Assigning default values to variables if they are passed in as NULL*

**if** ($data == **NULL**)

$data = "\*";

**if** ($order == **NULL**)

$order = "ID";

**if** ($order == **NULL**)

$order = "ASC";

$query = "SELECT \* FROM **{**$table**}**";

$foreachCount = 1;

*// If $data is "\*", it will return all data*

**if** ($data != "\*")

{

**foreach**($data **as** $column=>$value)

{

**if** (($foreachCount == 1) **AND** (count($data) > 0))

{

$query .= " WHERE ";

}

**else**

{

$query .= " AND ";

}

$query .= "`**{**$column**}**` = '**{**$value**}**'";

$foreachCount++;

}

}

*// Adds orders and sorts to the query*

$query .= " order by **{**$order**}**";

$query .= " **{**$sort**}**";

**if** ($limit != **NULL**)

$query .= " limit **{**$limit**}**";

*// If $debug is true, shows the query:*

**if** ($debug)

**echo** $query;

$result = mysqli\_query($this->openMYSQL, $query);

**if** (!$result) *// Returns false if there have been no records found*

{

**return** **false**;

}

**else** **if** ($result->num\_rows == 1) *// Returns a single array if there is only one record returned*

{

**return** mysqli\_fetch\_array($result);

}

**else** **if** ($result) *// Returns a multi-dimensional array when there's 2 or more records*

{

**while**($row = mysqli\_fetch\_array($result))

{

$output[] = $row;

}

*// Returns the output*

**if** (isset($output))

**return** $output;

**else**

**return** **false**;

}

}

$this->disconnect();

}

*// Method for inserting a record into a database table*

**public** **function** insertTable($table, $data)

{

$this->connect();

$query = "INSERT INTO **{**$table**}**";

*// COLUMN section*

$foreachCount = 1;

**foreach**($data **as** $column=>$value)

{

**if** ($foreachCount == 1)

{

$query .= " (";

}

**else**

{

$query .= ", ";

}

$query .= addslashes($column);

$foreachCount++;

}

$query .= ")";

*// VALUES section*

$foreachCount = 1;

**foreach**($data **as** $column=>$value)

{

**if** ($foreachCount == 1)

{

$query .= " VALUES (";

}

**else**

{

$query .= ", ";

}

$query .= "'".addslashes($value)."'";

$foreachCount++;

}

$query .= ")";

**if**(mysqli\_query($this->openMYSQL, $query))

**return** **true**;

**else**

**return** **false**;

$this->disconnect();

}

*// Method for updating a table*

**public** **function** updateTable($table, $data, $where, $debug=**false**)

{

$this->connect();

$query = "UPDATE **{**$table**}**";

*// COLUMN section*

$foreachCount = 1;

**foreach**($data **as** $column=>$value)

{

**if** ($foreachCount == 1)

{

$query .= " SET ";

}

**else**

{

$query .= ", ";

}

$query .= "`**{**$column**}**`='**{**$value**}**'";

$foreachCount++;

}

*// WHERE section*

$foreachCount = 1;

**foreach**($where **as** $whereColumn=>$whereValue)

{

**if** ($foreachCount == 1)

{

$query .= " WHERE ";

}

**else**

{

$query .= " AND ";

}

$query .= "`**{**$whereColumn**}**`='**{**$whereValue**}**'";

$foreachCount++;

}

*// If we're debugging, the query will be shown*

**if** ($debug)

**echo** $query;

**if**(mysqli\_query($this->openMYSQL, $query))

**return** **true**;

**else**

**return** **false**;

$this->disconnect();

}

}

?>

### createTables.php

<?php

**include\_once** "./includes/SQLDetails.php"; *// Includes the instantiated variable '$bookingSystem' for connections to the database.*

*// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*// This block of code creates database tables and inserts any data into tables which are static and are used for reference throughout the system*

*// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

$bookingSystem->createTable("Users", **array**("userID VARCHAR(20) NOT NULL, ADD PRIMARY KEY(userID)",

"emailAddress VARCHAR(254)",

"password VARCHAR(64) NOT NULL",

"userLevel INT(1) NOT NULL DEFAULT 1",

"firstName VARCHAR(35)",

"lastName VARCHAR(50)")

);

$bookingSystem->createTable("UserLevels", **array**("userLevel INT(1), ADD PRIMARY KEY(userLevel)",

"title VARCHAR(40) NOT NULL",

"description VARCHAR(200) NOT NULL")

);

$bookingSystem->insertTable("UserLevels", **array**("userLevel"=>0,

"title"=>"Guest",

"description"=>"A user level created for users that are not logged in or do not have any permissions.")

);

$bookingSystem->insertTable("UserLevels", **array**("userLevel"=>1,

"title"=>"Student",

"description"=>"A user level created for students that are able to only view events.")

);

$bookingSystem->insertTable("UserLevels", **array**("userLevel"=>2,

"title"=>"Staff",

"description"=>"A user level created for staff that are able to request bookings of events.")

);

$bookingSystem->insertTable("UserLevels", **array**("userLevel"=>3,

"title"=>"Administrator",

"description"=>"A user level created for admins that are able to approve requests for bookings.")

);

$bookingSystem->createTable("Events", **array**("name VARCHAR(50) NOT NULL",

"ownerID VARCHAR(20) NOT NULL",

"startTimestamp INT(64) NOT NULL",

"roomID VARCHAR(10) NOT NULL")

);

$bookingSystem->createTable("Rooms", **array**("roomID VARCHAR(10) NOT NULL, ADD PRIMARY KEY(roomID)",

"name VARCHAR(40)",

"capacity INT(4) DEFAULT 0")

);

$bookingSystem->createTable("Periods", **array**("periodNumber VARCHAR(40) NOT NULL, ADD PRIMARY KEY(periodNumber)",

"startTime INT(4)",

"endTime INT(4)")

);

$bookingSystem->insertTable("Periods", **array**("periodNumber"=>1,

"startTime"=>855,

"endTime"=>950)

);

$bookingSystem->insertTable("Periods", **array**("periodNumber"=>2,

"startTime"=>950,

"endTime"=>1045)

);

$bookingSystem->insertTable("Periods", **array**("periodNumber"=>3,

"startTime"=>1105,

"endTime"=>1200)

);

$bookingSystem->insertTable("Periods", **array**("periodNumber"=>4,

"startTime"=>1200,

"endTime"=>1255)

);

$bookingSystem->insertTable("Periods", **array**("periodNumber"=>5,

"startTime"=>1255,

"endTime"=>1350)

);

$bookingSystem->insertTable("Periods", **array**("periodNumber"=>6,

"startTime"=>1350,

"endTime"=>1445)

);

$bookingSystem->insertTable("Periods", **array**("periodNumber"=>7,

"startTime"=>1445,

"endTime"=>1540)

);

$bookingSystem->createTable("Holidays", **array**("label VARCHAR(50)",

"startTimestamp INT(64) NOT NULL",

"endTimestamp INT(64) NOT NULL")

);

?>

### functions.php

<?php

*//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*// PHP FUNCTIONS*

*//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*// Returns the user level according to what's stored in the cookie*

**function** getUserLevel($bookingSystem, $userID)

{

**return** $bookingSystem->queryTable("Users", **array**("userID"=>$\_COOKIE['userID']), 1, "userID", "ASC", **false**)['userLevel'];

}

*// Formats an input time with a colon*

**function** formatTime($inputTime)

{

*// This will create a string which will be 5 characters long -- 4 numerical characters and a colon.*

**return** rtrim(chunk\_split(str\_pad($inputTime, 4, "0", STR\_PAD\_LEFT), 2, ":"), ":");

}

*// Serialises the GET variables*

**function** serializeGetVariables()

{

$getVariables = array\_keys($\_GET);

**foreach**($getVariables **as** $singleGet)

{

$getVariablesWithValues[$singleGet] = $\_GET[$singleGet];

}

**return** serialize($getVariablesWithValues);

}

*// Destroys any user-login-related cookies*

**function** destroyCookies()

{

setcookie("userID", "", time() - 60\*60\*24\*3, "/");

setcookie("userPassword", "", time() - 60\*60\*24\*3, "/");

}

*// Checks to make sure that the user's details are still relevant*

**function** checkUserLogin($bookingSystem, $userID, $userPassword)

{

**if** ($bookingSystem->queryTable("Users", **array**("userID"=>$userID, "password"=>$userPassword), 1, "userLevel", "ASC", **false**))

**return** **true**;

**else**

**return** **false**;

}

*// Returns an error message containing the name of the user level that a user must be to view the page*

**function** returnUserLevelError($bookingSystem, $userLevelNeeded)

{

**echo** $userLevelTitle = $bookingSystem->queryTable("UserLevels", **array**("userLevel"=>$userLevelNeeded), 1, "userLevel", **false**, **false**)['title'];

**if** (in\_array(substr($userLevelTitle, 0, 1), **array**("A", "E", "I", "O", "U"))) *// Checks to see if the first letter is vowel*

$aOrAn = "an";

**else**

$aOrAn = "a";

**return** "You need to be ".$aOrAn." '".$userLevelTitle."' to view this content.";

}

*// Salts and hashes the password, with the salt being the user ID (capitalised)*

**function** saltAndHashPassword($userID, $password)

{

**return** hash("sha256", strtoupper($userID).$password);

}

*// Checks to see if the current page is the page sought ($filename)*

**function** isCurrentPage($filename)

{

$serverURL = $\_SERVER['SCRIPT\_FILENAME'];

*// If the filename is contained within the returned current page from the server*

**if** (strpos($serverURL, $filename))

**return** **true**;

**else**

**return** **false**;

}

*// Generates the title for the page*

**function** generateTitle()

{

*// A large relational array for holding relationships between pages to titles*

**include\_once** "./includes/titleForFilenames.php";

$basetitle = "KPS | RBS";

$sep = "|";

*// Adds appropriate formating to each filename and returns the appropriate result*

**foreach** ($titleForFilenames **as** $t => $filename)

{

*// Adds preceeding slash to filename*

$filename = "/".$filename;

*// If the current filename is the same as the file that is currently being accessed*

**if** (isCurrentPage($filename))

{

**if** ($t == "NOTITLE")

{

*// Returns an empty string as this page doesn't require a title*

**return** "";

}

**elseif** ($t == "")

{

*// Returns just the basetitle*

**return** $basetitle;

}

**else**

{

*// Returns title after the basetitle*

**return** $basetitle." ".$sep." ".$t;

}

}

}

}

?>

### header.php

<?php

*// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*// This file will be included on every page*

*// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

**include\_once** "./includes/createTables.php";

**include\_once** "./includes/SQLDetails.php";

**include\_once** "./includes/functions.php";

*// Checks to see if the user is logged in (but doens't check on the login page), and destroys the cookies if they're not logged in*

**if** (!checkUserLogin($bookingSystem, $\_COOKIE['userID'], $\_COOKIE['userPassword']) **AND** !strpos($\_SERVER['SCRIPT\_FILENAME'], "login.php"))

{

destroyCookies();

}

**echo** "<title>".generateTitle()."</title>";

?>

### loginInformation.php

<?php

**include\_once** "./includes/functions.php";

*// If the user ID cookie is set:*

**if**(isset($\_COOKIE['userID']))

{

*// Finds the user's user ID (to make sure that it's valid) and user level*

$userIDResponse = $bookingSystem->queryTable("Users", **array**("userID"=>$\_COOKIE['userID']), 1, "userID");

$userLevelResponse = $bookingSystem->queryTable("UserLevels", **array**("userLevel"=>$userIDResponse['userLevel']), 1, "userLevel");

*// Outputs a message to tell the user their user ID and user level*

**echo** "<div id='headerUsername' name='headerUsername'><i>You are logged in as: <strong>".$\_COOKIE['userID']."</strong> (<strong>".$userLevelResponse['title']."</strong>)</i></div>";

}

?>

### menu.php

<?php

*// Builds the list of buttons in the first array, with a list of what user level is needed in the second array*

$buttons = **array**('Home', 'View All Events', 'Add Event', 'Admin Panel', 'Login');

$userLevelForButtons = **array**('Home'=>0, 'View All Events'=>1, 'Add Event'=>2, 'Admin Panel'=>3, 'Login'=>0, 'Logout'=>1);

**include** "./includes/titleForFilenames.php";

*// The heading that will be shown at the top of every page*

**echo** "<h1 align='center'>Kings Priory School Room Booking System</h1>";

*// Builds the menu inside of this container*

**echo** "<menu id='headerMenu' name='headerMenu'>";

**foreach**($buttons **as** $buttonItem)

{

*// Changes login to logout if the user isn't logged in*

**if** ($buttonItem == 'Login')

{

**if**(getUserLevel($bookingSystem, $\_COOKIE['userID']) < 1)

{

**echo** "<a id='headerMenuLogin' href='".$titleForFilenames['Login']."'>Login</a>";

}

**else**

{

**echo** "<a id='headerMenuLogout' href='".$titleForFilenames['Logout']."'>Logout</a>";

}

}

*// If the level needed is less than the user that is logged in then the button is shown*

**else** **if**($userLevelForButtons[$buttonItem] <= getUserLevel($bookingSystem, $\_COOKIE['userID']))

{

**echo** "<a id='headerMenu".str\_replace(" ", "", $buttonItem)."' href='./".$titleForFilenames[$buttonItem]."'>".$buttonItem."</a>"; *// Builds the button with an appropriate element ID, URL and text*

}

}

**echo** "</menu>";

*// Includes the login information, which tells the user who they are logged in as and what type of user they are*

**include\_once** "./includes/loginInformation.php";

*// A break to separate the menu from the content underneath*

**echo** "<br />";

?>

### SQLDetails.php

<?php

**include\_once** "class.openDB.inc.php";

*// Here, you will need to create a new instance of the class 'openDB'. There's an example setup below:*

*// $bookingSystem = new openDB("localhost", "rootUser", "rootPass", "bookingSystem");*

*// Below makes the instantiation process explicit:*

*// $bookingSystem = new openDB({HOST}, {USERNAME}, {PASSWORD}, {DATABASENAME});*

$bookingSystem = **new** openDB("localhost", "root", "root", "bookingSystem");

?>

### styles.css

*/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/*

*/\* Styles that are used system-wide \*/*

*/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/*

**body** {

**width**: 1000px;

**margin**: **auto**;

**font-family**: Arial;

}

**h1**, **h2**, **h3**, **h4**, **h5**, **h6** {

**font-size**: 1.4em;

}

**h1** {

**text-decoration**: **underline**;

}

**h3**, **h4**, **h5**, **h6** {

**font-weight**: **normal**;

}

**a** {

**color**: **inherit**;

**text-decoration**: **none**;

}

**a**:hover {

**color**: **inherit**;

**text-decoration**: **underline**;

}

**a**:visited {

**color**: **inherit**;

}

**a**:active {

**color**: **inherit**;

}

#bookNow **a** {

**color**: #CCCCCC;

}

#errorMessage {

**color**: red;

**padding**: 10px;

**font-weight**: **bold**;

}

#successMessage {

**color**: green;

**padding**: 10px;

**font-weight**: **bold**;

}

#loginFormFieldSet {

**width**: 500px;

**margin**: **auto**;

}

#loginFormFieldSet **label** {

**padding-left**: 10px

}

#loginFormFieldSet **input** {

**display**: **block**;

**width**: 400px;

**padding**: 10px;

**margin**: 10px;

}

**.userFormFieldSet** {

**width**: 500px;

**margin**: **auto**;

}

**.userFormFieldSet** **label** {

**padding-left**: 10px

}

**.userFormFieldSet** **input** {

**display**: **block**;

**width**: 400px;

**padding**: 10px;

**margin**: 10px;

}

**.editFieldSet** {

**width**: 500px;

**margin**: **auto**;

}

**.editFieldSet** **label** {

**display**: **block**;

**padding-left**: 10px

}

**.editFieldSet** **input**, **select** {

**display**: **block**;

**width**: 400px;

**padding**: 10px;

**margin**: 10px;

}

**.eventHideableFields** {

**padding-left**: 50px;

}

#headerMenu > **a** {

**position**: **relative**;

**font-size**: 14pt;

**color**: #000000;

**border**: 2px **solid** #7a7979;

**background-color**: #448aff;

**height**: 20px;

**width**: 140px;

**display**: **inline**-**block**;

**margin-left**: 10px;

**margin-right**: 10px;

**border**-radius: 5px;

**font-weight**: 500;

**padding**: 2px;

**text-decoration**: **none**;

**text-align**: **center**;

**font-weight**: **bold**;

}

#headerMenu {

**text-align**: **center**;

}

**table** {

**border-collapse**: **collapse**;

}

#usersTable > **table**, **th**, **td** {

**border**: 1px **solid** black;

}

#usersTable > **td** {

**padding**: 15px;

}

**.viewAllUsersEdit** {

**position**: **relative**;

**font-size**: 12pt;

**color**: #000000;

**border**: 2px **solid** #999999;

**background-color**: #E6E6E6;

**height**: 10px;

**width**: 80px;

**margin-left**: 10px;

**margin-right**: 10px;

**border**-radius: 3px;

**font-weight**: 500;

**padding**: 0px;

**text-decoration**: **none**;

}

#eventTable **td** {

**width**: 100px;

**height**: 80px;

}

### titleForFilenames.php

<?php

*// 'NOTITLE' removes title*

$titleForFilenames = **array**(

"Add Event"=>"addEvent.php",

"Add Event: Import"=>"addEventImport.php",

"Add Event: Single"=>"addEventSingle.php",

"Add Holiday: Single"=>"addHolidaySingle.php",

"Add Room: Import"=>"addRoomImport.php",

"Add Room: Single"=>"addRoomSingle.php",

"Add User: Import"=>"addUserImport.php",

"Add User: Single"=>"addUserSingle.php",

"Admin Panel"=>"adminPanel.php",

"Edit Event: Single"=>"editEventSingle.php",

"Edit Holiday: Single"=>"editHolidaySingle.php",

"Edit Room: Single"=>"editRoomSingle.php",

"Edit User: Single"=>"editUserSingle.php",

"Home"=>"index.php",

"Login"=>"login.php",

"Logout"=>"logout.php",

"View All Events"=>"viewAllEvents.php",

"View All Holidays"=>"viewAllHolidays.php",

"View All Rooms"=>"viewAllRooms.php",

"View All Users"=>"viewAllUsers"

);

?>

## Page Files (as of 17/3/16)

### addEvent.php

|  |
| --- |
|  |

<!DOCTYPE html>

<html>

<head>

<!-- **This** includes the stylesheet **for** the system -->

<link rel="stylesheet" href="./includes/styles.css" type="text/css" />

<!-- Includes the header, which includes other useful files, checks user login details **and** generates an appropriate title -->

<?php **include** "./includes/header.php"; ?>

</head>

<body>

<?php

*// Includes the menu for navigating the site, which will be generated differently for each user*

**include** "./includes/menu.php";

*// Checks to make sure that only users of a sufficient user level can view the content of the page*

$userLevelNeeded = 2;

**if**(getUserLevel($bookingSystem, $\_COOKIE['userID']) >= $userLevelNeeded)

{

?>

<!-- Links to the add event pages -->

<h1>Add Event</h1>

<h3><a href="addEventSingle.php">Add Event: Single</a></h3>

<h3><a href="addEventImport.php">Add Event: Import</a></h3>

<?php

}

**else**

{

*// If the user doesn't have the correct user level, they are shown an error message*

**echo** "<p>".returnUserLevelError($bookingSystem, $userLevelNeeded)."</p>";

}

?>

</body>

</html>

### addEventImport.php

|  |
| --- |
|  |

<?php

**include\_once** "./includes/SQLDetails.php";

$errorMessage = "";

**if** (isset($\_POST['importRoomBookingFormSubmit']))

{

*// Need to create error messages for empty fields*

**if** ($\_FILES['importRoomBookingFormCSVUpload'] == **NULL**)

$errorMessage .= "Please upload a CSV file.<br />";

**if** ($\_POST['importRoomBookingFormInsertDate'] == **NULL**)

$errorMessage .= "Please select an insert date.";

**if** ($\_POST['importRoomBookingFormRepeatEvent'])

{

**if** ($\_POST['importRoomBookingFormInsertDate'] > $\_POST['importRoomBookingFormRepeatToDate'])

$errorMessage .= "The repeat-to date is earlier than the insert date.";

}

}

?>

<!-- A JS function for hiding and showing repeat functions on the form -->

<script>

function hideRepeats()

{

if (document.getElementById('importRoomBookingFormRepeatEvent').checked)

{

document.getElementById('importRoomBookingFormRepeatEventHideableFields').style.display = 'block';

}

else

{

document.getElementById('importRoomBookingFormRepeatEventHideableFields').style.display = 'none';

}

}

</script>

<!DOCTYPE html>

<html>

<head>

<!-- This includes the stylesheet for the system -->

<link rel="stylesheet" href="./includes/styles.css" type="text/css" />

<!-- Includes the header, which includes other useful files, checks user login details and generates an appropriate title -->

<?php **include** "./includes/header.php"; ?>

</head>

<body>

<?php

*// Includes the menu for navigating the site, which will be generated differently for each user*

**include** "./includes/menu.php";

*// Verifies that the current user has the right user level*

$userLevelNeeded = 3;

**if**(getUserLevel($bookingSystem, $\_COOKIE['userID']) >= $userLevelNeeded)

{

?>

<h1>Add Event</h1>

<h2>Add Event: Import</h2>

<!-- Some messages to warn the user about upload times -->

<p>After submitting the upload file, the upload process will start. <strong>Be patient</strong>, especially for large inserts.</p>

<!-- <p>The events will start inserting the day after the one that you select.</p> -->

<form id="importRoomBookingForm" name="importRoomBookingForm" action="<?php **echo** $\_SERVER['PHP\_SELF']; ?>" method="post" autocomplete="off" enctype="multipart/form-data">

<fieldset name="importRoomBookingFormFieldSet" id="importRoomBookingFormFieldSet" class="editFieldSet">

<legend>Add Event: Import</legend>

<?php

*// A message to the user to let them know that the events are inserting as soon as the CSV has been uploaded*

**if** ((isset($\_FILES['importRoomBookingFormCSVUpload'])) **AND** ($errorMessage == ""))

**echo** "<div id='successMessage' name='successMessage'>Inserting... See insertion details below.</div>";

?>

<div id="errorMessage" name="errorMessage"><?php **echo** $errorMessage; ?></div>

<!-- The date that the events should start from - if the form has already been sumitted with an insert date, that will be the default; if not, today's date will be used -->

<label for="importRoomBookingFormInsertDate">The date that the events should start from</label>

<input type="date" id="importRoomBookingFormInsertDate" name="importRoomBookingFormInsertDate" value="<?php **if**(isset($\_POST['importRoomBookingFormInsertDate'])) { **echo** $\_POST['importRoomBookingFormInsertDate']; } **else** { **echo** date("Y-m-d", time()); } ?>" />

<!-- Calls the JS function below once the button has been clicked; this displays the other data fields concerned with repeating -->

<label for="importRoomBookingFormRepeatEvent">Repeat event weekly</label>

<input type="checkbox" id="importRoomBookingFormRepeatEvent" name="importRoomBookingFormRepeatEvent" onClick="hideRepeats()" />

<!-- These hideable fields are shown when the repeat event weekly checkbox is checked -->

<div id="importRoomBookingFormRepeatEventHideableFields" style="display:none" class="eventHideableFields">

<!-- The date that the events should repeat up to -->

<label for="importRoomBookingFormRepeatToDate">The date that the event should repeat up until</label>

<input type="date" id="importRoomBookingFormRepeatToDate" name="importRoomBookingFormRepeatToDate" />

<!-- If this is checked, the events will repeat even when there are holidays -->

<label for="importRoomBookingFormRepeatInHolidays">Continue to repeat in holidays</label>

<input type="checkbox" value="true" id="importRoomBookingFormRepeatInHolidays" name="importRoomBookingFormRepeatInHolidays" />

</div>

<!-- Allows for a CSV for parsing to be uploaded -->

<label for="importRoomBookingFormCSVUpload">Upload a user CSV containing: Mon:1; Mon:2; ...; Fri:7, listed either with staff or rooms.</label>

<input type="file" accept=".csv" name="importRoomBookingFormCSVUpload" id="importRoomBookingFormCSVUpload" />

<!-- Submits the form -->

<input type="submit" name="importRoomBookingFormSubmit" />

<br />

<br />

<?php

**if** ((isset($\_FILES['importRoomBookingFormCSVUpload'])) **AND** ($errorMessage == ""))

{

$fullImportedCSV = file($\_FILES['importRoomBookingFormCSVUpload']['tmp\_name']);

$CSVHeaders = explode(",", $fullImportedCSV[0]); *// This keeps the headings in an array called $CSVHeaders*

$importedCSV = $fullImportedCSV;

array\_shift($importedCSV); *// This removes the headings element of the array*

*// For each $importedCSV array\_shift() ? This should remove teacher and room headings*

$successful = **array**(); *// An array to store a list of successful imported items*

$unsuccessful = **array**(); *// An array to store a list of unsuccessful imported items*

**if**($\_POST['importRoomBookingFormRepeatInHolidays'] == "true")

$repeatInHolidays = **true**;

**else**

$repeatInHolidays = **false**;

*// Loops through each row of the CSV, row by row*

**foreach**($importedCSV **as** $row)

{

$row = explode(",", $row); *// Put in here a splitter on ',' to break up the string into elements of an array (turning Comma Separated Values into an array)*

$daysOfTheWeek = **array**("Mon", "Tue", "Wed", "Thu", "Fri");

$dayPeriodCounter = -1; *// Starts at -1 so that the first column is ignored*

*// Loops through each element of row (so each value) and assigns it to cell*

**foreach**($row **as** $cell)

{

*// Resets values used for each import*

$roomID = "";

$ownerID = "";

$eventName = "";

$insertDay = 0;

$insertPeriod = 0;

*// Need to calculate day and period of the event*

$insertDay = (int) ($dayPeriodCounter / 7);

$insertDay = $daysOfTheWeek[$insertDay]; *// Figures out which day of the week it is*

$insertPeriod = $dayPeriodCounter % 7 + 1; *// Figures out which period it is for insert*

$insertDate = strtotime($\_POST['importRoomBookingFormInsertDate']); *// Turns the date format into a unix timestamp*

*// If the terminate date isn't set, then use the insert date as the terminate date*

**if** (isset($\_POST['importRoomBookingFormRepeatEvent'])) *// If the repeat checkbox is not checked, then set the terminate date to the insert date, otherwise set the terminate date to the date selected*

$terminateDate = strtotime($\_POST['importRoomBookingFormRepeatToDate']);

**else**

$terminateDate = $insertDate;

*// Parses the cell string to see what the room name is*

**if** (strpos($cell, "#"))

$roomID = substr($cell, strpos($cell, "#") + 1, strpos($cell, ")") - strpos($cell, "#") - 1); *//only count for the length between # and 1 before )*

*// Parses the cell string to see what the ownerID is*

**if** (strpos($cell, "$"))

$ownerID = substr($cell, strpos($cell, "$") + 1, 3); *//start 3 after where $ is found*

*// Parses the cell string to see what the event name is*

**if** (strpos($cell, "$"))

$eventName = substr($cell, 0, strpos($cell, "$") - 1); *//stop 1 before where $ is found*

*// If the RoomID contains letters and numbers we can presume that the room name is the same as the roomID*

**if** (preg\_match('/[A-Z]+[0-9]+/', $roomID))

$roomName = $roomID;

**else**

$roomName = "";

**if** ((!$bookingSystem->queryTable("Rooms", **array**("roomID"=>$roomID), "roomID")) **AND** (strlen($roomID) > 0) **AND** ($roomID != " ")) *// Checks to see if the room exists and, if not, adds it (as long as the roomID isn't blank)*

$bookingSystem->insertTable("Rooms", **array**("roomID"=>$roomID, "name"=>$roomName));

*// Figures out what the time for the period for insert is*

$insertPeriodTime = $bookingSystem->queryTable("Periods", **array**("periodNumber"=>$insertPeriod), 1, "periodNumber")['startTime'];

$insertPeriodTimeFormatted = formatTime($insertPeriodTime); *// Formats the time with a colon, according to the function's definition in ./includes/functions.php*

**if** ((!$bookingSystem->queryTable("Users", **array**("userID"=>$ownerID), "userID")) **AND** (strlen($ownerID) > 0) **AND** ($ownerID != " ")) *// Checks to see if the room exists and, if not, adds it (as long as the roomID isn't blank)*

$bookingSystem->insertTable("Users", **array**("userID"=>$ownerID, "userLevel"=>2, "password"=>saltAndHashPassword($ownerID, "password")));

*// I NEED TO TEST THIS*

**if** ((strlen($cell) > 0) **AND** (strlen($eventName) > 0) **AND** (strlen($roomID) > 0) **AND** (strlen($insertPeriodTimeFormatted) > 0) **AND** (strlen($ownerID) > 0))

{

*// Need to calculate when to import the event*

**while** ($insertDate <= $terminateDate) *// Need to loop through all of the weeks until the specified date*

{

*// Need to calculate the timestamp of each iteration*

$insertTimestamp = strtotime(date("Y-m-d**\T**H:i", $insertDate - 24\*60\*60)." next ".$insertDay." ".$insertPeriodTimeFormatted); *// Makes sure that the insert timestamp for this insert is calculated for the day specified, not the day after (that's why is goes back and forwards a day, whilst avoiding weekend-based errors)*

*// If the event is not set to repeat, it is inserted once:*

**if** (!$\_POST['importRoomBookingFormRepeatEvent'])

{

*// Attempts to insert the event*

**if**($bookingSystem->insertTable("Events", **array**("name"=>$eventName, "roomID"=>$roomID, "startTimestamp"=>$insertTimestamp, "ownerID"=>$ownerID)))

$successful[] = $cell." {<i>".date("d/m/Y", $insertTimestamp)." Period".$insertPeriod."</i>}";

**else**

$unsuccessful[] = $cell." {<i>".date("d/m/Y", $insertTimestamp)." Period".$insertPeriod."</i>}";

}

**else**

{

*// If the event is set to repeat in holidays and the repeat setting has been set (the event will be inserted even if the event falls in a holiday date range):*

**if**($repeatInHolidays)

{

*// Attempts to insert the event*

**if**($bookingSystem->insertTable("Events", **array**("name"=>$eventName, "roomID"=>$roomID, "startTimestamp"=>$insertTimestamp, "ownerID"=>$ownerID)))

$successful[] = $cell." {<i>".date("d/m/Y", $insertTimestamp)." Period".$insertPeriod."</i>}";

**else**

$unsuccessful[] = $cell." {<i>".date("d/m/Y", $insertTimestamp)." Period".$insertPeriod."</i>}";

}

**else**

{

*// If the event is set to repeat and there are no holidays set*

**if** (**empty**($bookingSystem->queryTable(**NULL**, "SELECT \* FROM Holidays WHERE ".$insertDate." BETWEEN `startTimestamp` AND `endTimestamp`", **NULL**, "ID", "ASC", **false**, **true**)))

{

*// Attempts to insert the event*

**if**($bookingSystem->insertTable("Events", **array**("name"=>$eventName, "roomID"=>$roomID, "startTimestamp"=>$insertTimestamp, "ownerID"=>$ownerID)))

$successful[] = $cell." {<i>".date("d/m/Y", $insertTimestamp)." Period".$insertPeriod."</i>}";

**else**

$unsuccessful[] = $cell." {<i>".date("d/m/Y", $insertTimestamp)." Period".$insertPeriod."</i>}";

}

**else**

{

*// No events will be inserted if there are holidays set*

}

}

}

$insertDate += 60\*60\*24\*7; *// Adds on one week to the last insert date*

}

}

**else**

{

*// If the cell's contents resembles a teacher ID or a room ID:*

**if** (!(($bookingSystem->queryTable(**NULL**, "SELECT \* FROM `Rooms` WHERE `roomID` LIKE '%**{**$cell**}**%'", **NULL**, **NULL**, **NULL**, **false**, **true**))) **OR** ($bookingSystem->queryTable(**NULL**, "SELECT \* FROM `Users` WHERE `userID` LIKE '%**{**$cell**}**%'", **NULL**, **NULL**, **NULL**, **false**, **true**))) *// If it is a teacher or a room as the cell's value, then ignore and don't add to $unsuccessful[]*

{

*// If the cell string has a valid make-up, or if it contains LUNCH, LEAD or PPA, then it is ignored*

**if** ((strlen($cell) > 0) **AND** ($cell != " ") **AND** (strlen(trim($cell)) != 0) **AND** ($cell != **NULL**) **AND** (!strpos($cell, "LUNCH")) **AND** (!strpos($cell, "LEAD")) **AND** (!strpos($cell, "PPA")))

$unsuccessful[] = $cell." {<i>Missing (or too much) data</i>}";

}

}

$dayPeriodCounter++;

}

}

*// Displays the number of successful and unsuccessful inserts*

**echo** "<p>".count($successful)." items inserted successfully.</p>";

**echo** "<p>".count($unsuccessful)." items failed to insert.</p>";

**echo** "<p>See details below.</p>";

*// Displays all succesful inserts*

**if** (count($successful) > 0)

{

**echo** "<h3>Successful Items</h3>";

**echo** "<ul>";

**foreach** ($successful **as** $item)

{

**echo** "<li>".$item."</li>";

}

**echo** "</ul>";

}

*// Displays all unsuccessful inserts*

**if** (count($unsuccessful) > 0)

{

**echo** "<h3>Unsuccessful Items</h3>";

**echo** "<ul>";

**foreach** ($unsuccessful **as** $item)

{

**echo** "<li>".$item."</li>";

}

**echo** "</ul>";

}

}

?>

</fieldset>

</form>

<?php

}

**else**

{

*// An error message is shown if the user isn't allowed to view the page due to their user level*

**echo** "<p>".returnUserLevelError($bookingSystem, $userLevelNeeded)."</p>";

}

?>

</body>

</html>

### addEventSingle.php

|  |
| --- |
|  |

<!DOCTYPE html>

<html>

<head>

<!-- **This** includes the stylesheet **for** the system -->

<link rel="stylesheet" href="./includes/styles.css" type="text/css" />

<!-- Includes the header, which includes other useful files, checks user login details **and** generates an appropriate title -->

<?php **include** "./includes/header.php"; ?>

</head>

<body>

<?php

*// Includes the menu for navigating the site, which will be generated differently for each user*

**include** "./includes/menu.php";

*// These variables store the messages that will be shown after the form has been submitted*

$successMessage = "";

$errorMessage = "";

*// This page requries the user to be a teacher*

$userLevelNeeded = 2;

**if**(getUserLevel($bookingSystem, $\_COOKIE['userID']) >= $userLevelNeeded)

{

?>

<h1>Add Event</h1>

<h2>Add Event: Single</h2>

<?php

*// If the form has been submitted:*

**if** (isset($\_POST['addEventSingleFormSubmit']))

{

**if** (strlen($\_POST['addEventSingleFormEventName']) < 1)

$errorMessage .= "Please enter an event name.<br />";

**if** (strlen($\_POST['addEventSingleFormTeacherID']) < 1)

$errorMessage .= "Please enter a teacher ID.<br />";

**if** (strlen($\_POST['addEventSingleFormDate']) < 1)

$errorMessage .= "Please enter a date.<br />";

**if** (strlen($\_POST['addEventSingleFormPeriod']) < 1)

$errorMessage .= "Please enter a period.<br />";

**if** (strlen($\_POST['addEventSingleFormRoom']) < 1)

$errorMessage .= "Please enter a room.<br />";

*// Builds an insert array for each of the columns in the table*

$insertArray['name'] = $\_POST['addEventSingleFormEventName'];

$insertArray['ownerID'] = $\_POST['addEventSingleFormTeacherID'];

$insertArray['roomID'] = $\_POST['addEventSingleFormRoom'];

*// The code below figures out what the start timestamp for the event will be*

$periodTime = $bookingSystem->queryTable("Periods", **array**("periodNumber"=>$\_POST['addEventSingleFormPeriod']), 1, "periodNumber")['startTime'];

$insertPeriodTimeFormatted = formatTime($periodTime); *// Formats the period time with a colon*

$insertArray['startTimestamp'] = strtotime(date("Y-m-d", strtotime($\_POST['addEventSingleFormDate']))."T".$insertPeriodTimeFormatted); *// Formats the date and time for the period into the correct format then to a unix timestamp*

*// If there are no error messages and the insert was successful*

**if**(($errorMessage == "") **AND** ($bookingSystem->insertTable("Events", $insertArray)))

{

$successMessage = "Event inserted successfully.";

*// If there was a get string provided*

**if** (strlen($\_POST['addEventSingleFormReturnGET']) > 0)

{

*// Unserialises the GET variables and returns the user to viewAllEvents.php with what ever the parameters were set as before the user wanted to add an event*

$returnGET = unserialize($\_POST['addEventSingleFormReturnGET']);

$successMessage .= "<META HTTP-EQUIV=REFRESH CONTENT='1; URL=./viewAllEvents.php?view=".$returnGET['view'];

**if** (isset($returnGET['roomID']))

$successMessage .= "&roomID=".$returnGET['roomID'];

**else** **if** (isset($returnGET['teacherID']))

$successMessage .= "&teacherID=".$returnGET['teacherID'];

**if** (isset($returnGET['referenceWeek']))

$successMessage .= "&referenceWeek=".$returnGET['referenceWeek'];

$successMessage .= "'>";

}

**else**

{

*// Redirects the user to addEvent.php if there were no GET variables set*

$successMessage .= "<META HTTP-EQUIV=REFRESH CONTENT='1; URL=./addEvent.php'>";

}

}

**else**

{

*// Error message if the event fails to insert*

$errorMessage .= "Event failed to insert.";

}

}

**if** (isset($\_GET['time']))

{

*// If the time is set, we need to figure out what period the event would have been for*

$explodedTimestamp = explode("T", date("Y-m-d**\T**Hi", $\_GET['time']));

$eventDate = $explodedTimestamp[0];

$periodTime = $explodedTimestamp[1];

$eventPeriod = $bookingSystem->queryTable("Periods", **array**("startTime"=>$periodTime), 1, "periodNumber")['periodNumber'];

}

**else** **if**(isset($\_POST['addEventSingleFormSubmit']))

{

*// Sets all the variables if the form has been submitted*

$eventName = $\_POST['addEventSingleFormEventName'];

$eventTeacherID = $\_POST['addEventSingleFormTeacherID'];

$eventDate = $\_POST['addEventSingleFormDate'];

$eventPeriod = $\_POST['addEventSingleFormPeriod'];

$eventRoom = $\_POST['addEventSingleFormRoom'];

}

**if** (isset($\_GET['serializedGetVariables']))

{

*// Finds out what the values of each of the variables in the GET array were*

$returnGET = unserialize($\_GET['serializedGetVariables']);

**if** (isset($returnGET['teacherID']))

$eventTeacherID = $returnGET['teacherID'];

**elseif** (isset($returnGET['roomID']))

$eventRoom = $returnGET['roomID'];

}

?>

<form id="addEventSingleForm" name="addEventSingleForm" action="<?php **echo** $\_SERVER['PHP\_SELF']; ?>" method="post" autocomplete="off">

<fieldset name="addEventSingleFormFieldSet" id="addEventSingleFormFieldSet" class="userFormFieldSet">

<legend>Add Event: Single</legend>

<!-- Shows the error and success messages -->

<div id="errorMessage" name="errorMessage"><?php **echo** $errorMessage; ?></div>

<div id="successMessage" name="successMessage"><?php **echo** $successMessage; ?></div>

<label for="addEventSingleFormEventName">Event Name:</label>

<input type="text" name="addEventSingleFormEventName" maxlength="50" autofocus value="<?php **echo** $eventName; ?>" />

<label for="addEventSingleFormTeacherID">Teacher ID:</label> <!-- Different for administrators and teachers -->

<select name="addEventSingleFormTeacherID">

<option value="" disabled selected>Select a teacher</option>

<?php

$returnedTeacherList = $bookingSystem->queryTable("Users", **array**("userLevel" => 2), **NULL**, " userID, lastName, firstName", "ASC"); *// Picks all teachers*

*// Creates a multidimensional array for the foreach loop*

**if** (isset($returnedTeacherList['userID']))

$teacherResult[0] = $returnedTeacherList;

**else**

$teacherResult = $returnedTeacherList;

**foreach** ($teacherResult **as** $teacher)

{

*// If the current user is one of the teachers in the list, select them as default*

**if** (isset($eventTeacherID) **AND** ($teacher['userID'] == $eventTeacherID))

$selected = " selected";

**else**

$selected = "";

**echo** "<option value='".$teacher['userID']."'".$selected.">[".$teacher['userID']."] ".$teacher['firstName']." ".$teacher['lastName']."</option>";

}

?>

</select>

<!-- If the date isn't set, then use today's date by default -->

<?php **if**(isset($eventDate)) { $defaultDateValue = $eventDate; } **else** { $defaultDateValue = date("Y-m-d"); } ?>

<label for="addEventSingleFormDate">Date:</label>

<input type="date" name="addEventSingleFormDate" value="<?php **echo** $defaultDateValue; ?>" />

<label for="addEventSingleFormPeriod">Period:</label>

<select name="addEventSingleFormPeriod">

<option value="" disabled selected>Select a period</option>

<?php

$periodResult = $bookingSystem->queryTable("Periods", "\*", **NULL**, "periodNumber", "ASC"); *// Picks all periods*

**foreach** ($periodResult **as** $period)

{

*// If the period number sought is the same as the one in the loop, then set it as selected*

**if** (isset($eventPeriod) **AND** ($period['periodNumber'] == $eventPeriod))

$selected = " selected";

**else**

$selected = "";

**echo** "<option value='".$period['periodNumber']."'".$selected.">Period ".$period['periodNumber']." (".formatTime($period['startTime'])." to ".formatTime($period['endTime']).")</option>";

}

?>

</select>

<label for="addEventSingleFormFirstName">Room:</label>

<select name="addEventSingleFormRoom">

<option value="" disabled selected>Select a Room</option>

<?php

$returnedRoomList = $bookingSystem->queryTable("Rooms", "\*", **NULL**, "roomID", "ASC"); *// Picks all rooms*

*// Creates a multidimensional array for the foreach loop*

**if** (isset($returnedRoomList['roomID']))

$roomResult[0] = $returnedRoomList;

**else**

$roomResult = $returnedRoomList;

**foreach** ($roomResult **as** $room)

{

**if** (isset($eventRoom) **AND** ($room['roomID'] == $eventRoom))

$selected = " selected";

**else**

$selected = "";

**if** ($room['capacity'] == "")

$room['capacity'] = "Unknown capacity"; *// If there is no capacity specified, unknown is listed next to the room.*

**echo** "<option value='".$room['roomID']."'".$selected.">".$room['name']." (".$room['capacity'].")</option>";

}

?>

</select>

<!-- A hidden field to ensure that we don't lose the GET variables from the previous page when we submit the form -->

<input type="hidden" value="<?php **echo** htmlspecialchars($\_GET['serializedGetVariables']); ?>" id="addEventSingleFormReturnGET" name="addEventSingleFormReturnGET" form="addEventSingleForm" />

<input type="submit" value="Add Event" id="addEventSingleFormSubmit" name="addEventSingleFormSubmit" form="addEventSingleForm" />

</fieldset>

</form>

<?php

}

**else**

{

*// Returns an appropriate error message if the user can't view the page*

**echo** "<p>".returnUserLevelError($bookingSystem, $userLevelNeeded)."</p>";

}

?>

</body>

</html>

### addHolidaySingle.php

|  |
| --- |
|  |

<?php

*// Builds the success and error messages*

$errorMessage = "";

$successMessage = "";

**if** ($\_POST['createSingleHolidayFormSubmit'])

{

**if** ($\_POST['createSingleHolidayFormLabel'] == "")

$errorMessage .= "Please enter a holiday label.<br />";

**if** ($\_POST['createSingleHolidayFormStart'] == 0) *// We can presume that 0 is an unfilled field*

$errorMessage .= "Please enter a holiday start.<br />";

**if** ($\_POST['createSingleHolidayFormEnd'] == 0) *// We can presume that 0 is an unfilled field*

$errorMessage .= "Please enter a holiday end.<br />";

**if** ($\_POST['createSingleHolidayFormStart'] > $\_POST['createSingleHolidayFormEnd'])

$errorMessage .= "The start date is after the end date of the holiday.";

}

?>

<!DOCTYPE html>

<html>

<head>

<!-- This includes the stylesheet for the system -->

<link rel="stylesheet" href="./includes/styles.css" type="text/css" />

<!-- Includes the header, which includes other useful files, checks user login details and generates an appropriate title -->

<?php **include** "./includes/header.php"; ?>

</head>

<body>

<?php

*// Includes the menu for navigating the site, which will be generated differently for each user*

**include** "./includes/menu.php";

*// The user must be an administrator*

$userLevelNeeded = 3;

**if**(getUserLevel($bookingSystem, $\_COOKIE['userID']) >= $userLevelNeeded)

{

?>

<?php

*// If the form has been submitted and there have been no errors:*

**if**((isset($\_POST['createSingleHolidayFormSubmit'])) **AND** ($errorMessage == ""))

{

*// Fills in the fields with the data that have been submitted*

$holidayLabel = $\_POST['createSingleHolidayFormLabel'];

$holidayStart = strtotime($\_POST['createSingleHolidayFormStart']);

$holidayEnd = strtotime($\_POST['createSingleHolidayFormEnd']);

*// Checks if the insert was successful or not*

**if**($bookingSystem->insertTable("Holidays", **array**("label"=>$holidayLabel, "startTimestamp"=>$holidayStart, "endTimestamp"=>$holidayEnd)))

{

$successMessage .= "Insert successful.";

$successMessage .= "<META HTTP-EQUIV=REFRESH CONTENT='1; URL=./viewAllHolidays.php'>";

}

**else**

{

$errorMessage .= "Insert for ".$holidayLabel." failed.<br />";

}

}

?>

<h1><a href="adminPanel.php">Admin Panel</a></h1>

<h2>Add Holidays</h2>

<h3><a href="addHolidaySingle.php">Add a Single Holiday</a></h3>

<form id="createSingleHolidayForm" name="createSingleHolidayForm" action="<?php **echo** $\_SERVER['PHP\_SELF']; ?>" method="post" autocomplete="off">

<fieldset name="createSingleHolidayFormFieldSet" id="createSingleHolidayFormFieldSet" class="userFormFieldSet">

<legend>Add a Single Holiday</legend>

<!-- Shows the success and error messages -->

<div id="errorMessage" name="errorMessage"><?php **echo** $errorMessage; ?></div>

<div id="successMessage" name="successMessage"><?php **echo** $successMessage; ?></div>

<label for="createSingleHolidayFormLabel">Holiday Label:</label>

<input type="text" name="createSingleHolidayFormLabel" maxlength="50" autofocus value="<?php **echo** $\_POST['createSingleHolidayFormLabel']; ?>" />

<label for="createSingleHolidayFormStart">Holiday Start:</label>

<input type="date" name="createSingleHolidayFormStart" value="<?php **echo** $\_POST['createSingleHolidayFormStart']; ?>" />

<label for="createSingleHolidayFormEnd">Holiday End:</label>

<input type="date" name="createSingleHolidayFormEnd" value="<?php **echo** $\_POST['createSingleHolidayFormEnd']; ?>" />

<input type="submit" value="Create Holiday" id="createSingleHolidayFormSubmit" name="createSingleHolidayFormSubmit" form="createSingleHolidayForm" />

</fieldset>

</form>

<?php

}

**else**

{

**echo** "<p>".returnUserLevelError($bookingSystem, $userLevelNeeded)."</p>";

}

?>

</body>

</html>

### addRoomImport.php

|  |
| --- |
|  |

<?php

**include\_once** "./includes/SQLDetails.php";

$errorMessage = "";

$successMessage = "";

**if** (isset($\_POST['importAddRoomFormSubmit']))

{

**if** (!isset($\_POST['importAddRoomFormCSVUpload']))

$errorMessage .= "Please upload a CSV file.<br />";

**else**

$successMessage .= "Inserting...";

}

?>

<!DOCTYPE html>

<html>

<head>

<!-- This includes the stylesheet for the system -->

<link rel="stylesheet" href="./includes/styles.css" type="text/css" />

<!-- Includes the header, which includes other useful files, checks user login details and generates an appropriate title -->

<?php **include** "./includes/header.php"; ?>

</head>

<body>

<?php

*// Includes the menu for navigating the site, which will be generated differently for each user*

**include** "./includes/menu.php";

$userLevelNeeded = 3;

**if**(getUserLevel($bookingSystem, $\_COOKIE['userID']) >= $userLevelNeeded)

{

?>

<h1>Admin Panel</h1>

<h2>Create Events</h2>

<h3>Add a Single Room</h3>

<?php

**if** (isset($\_FILES['importAddRoomFormCSVUpload'])) *// Can't get uploaded files to work*

{

*//var\_dump(file\_get\_contents($\_FILES['importAddRoomFormCSVUpload']['tmp\_name']));*

$fullImportedCSV = file($\_FILES['importAddRoomFormCSVUpload']['tmp\_name']);

$CSVHeaders = explode(",", $fullImportedCSV[0]); *// This keeps the headings*

$importedCSV = $fullImportedCSV;

array\_shift($importedCSV); *// This removes the headings element of the array*

**foreach**($importedCSV **as** $row)

{

$row = array\_map("trim", explode(",", $row)); *// Put in here a splitter on ',' to break up the string into elements of an array and removes any remaining whitespace (with the array\_map function)*

*// Inserts each room*

**if**($bookingSystem->insertTable("Rooms", **array**("name"=>$row[0], "roomID"=>$row[1], "capacity"=>$row[2])))

{

$successful[] = "The room '".$row[0]."' was added sucessfully.";

}

**else**

{

$unsuccessful[] = "There was an error when adding the room '".$row[0]."'. This may be because a room of the same room ID exists.";

}

}

}

?>

<form id="importAddRoomForm" name="importAddRoomForm" action="<?php **echo** $\_SERVER['PHP\_SELF']; ?>" method="post" autocomplete="off" enctype="multipart/form-data">

<fieldset name="importAddRoomFormFieldSet" id="importAddRoomFormFieldSet" class="createSinglEventFieldSet">

<legend>Add Room: Import</legend>

<!-- Success and error messages -->

<div id='errorMessage' name='errorMessage'><?php **echo** $errorMessage; ?></div>

<div id='successMessage' name='successMessage'><?php **echo** $successMessage; ?></div>

<label for="importAddRoomFormCSVUpload">Upload a room CSV containing: Room name; Room ID; and Room capacity.</label>

<input type="file" name="importAddRoomFormCSVUpload" id="importAddRoomFormCSVUpload" />

<input type="submit" name="importAddRoomFormSubmit" />

<br />

<br />

</fieldset>

</form>

<?php

*// Displays all succesful inserts*

**if** (count($successful) > 0)

{

**echo** "<h3>Successful Items</h3>";

**echo** "<ul>";

**foreach** ($successful **as** $item)

{

**echo** "<li>".$item."</li>";

}

**echo** "</ul>";

}

*// Displays all unsuccessful inserts*

**if** (count($unsuccessful) > 0)

{

**echo** "<h3>Unsuccessful Items</h3>";

**echo** "<ul>";

**foreach** ($unsuccessful **as** $item)

{

**echo** "<li>".$item."</li>";

}

**echo** "</ul>";

}

}

**else**

{

**echo** "<p>".returnUserLevelError($bookingSystem, $userLevelNeeded)."</p>";

}

?>

</body>

</html>

### addRoomSingle.php

|  |
| --- |
|  |

<!DOCTYPE html>

<html>

<head>

<!-- **This** includes the stylesheet **for** the system -->

<link rel="stylesheet" href="./includes/styles.css" type="text/css" />

<!-- Includes the header, which includes other useful files, checks user login details **and** generates an appropriate title -->

<?php **include** "./includes/header.php"; ?>

</head>

<body>

<?php

*// Includes the menu for navigating the site, which will be generated differently for each user*

**include** "./includes/menu.php";

*// The user needs to be an administrator to view the page*

$userLevelNeeded = 3;

**if**(getUserLevel($bookingSystem, $\_COOKIE['userID']) >= $userLevelNeeded)

{

*// Variables that contain error and success messages*

$errorMessage = "";

$successMessage = "";

**if** (isset($\_POST['addRoomFormSubmit']))

{

**if** ($\_POST['addRoomFormRoomID'] == "")

$errorMessage .= "Please enter a room ID.";

*// If the form has been submitted, use the posted values as the default values for the fields*

$addRoomFormRoomID = $\_POST['addRoomFormRoomID'];

$addRoomFormRoomName = $\_POST['addRoomFormRoomName'];

$addRoomFormCapacity = $\_POST['addRoomFormCapacity'];

}

*// If the form has been submitted and there are no errors:*

**if** ((isset($\_POST['addRoomFormSubmit'])) **AND** ($errorMessage == ""))

{

**if** ($addRoomFormCapacity == "")

$addRoomFormCapacity = 0;

*// Builds the insert array*

$insertArray = **array**("roomID"=>$addRoomFormRoomID,

"name"=>$addRoomFormRoomName,

"capacity"=>$addRoomFormCapacity

);

**if**($bookingSystem->insertTable("Rooms", $insertArray))

{

$successMessage .= "Insert successful.";

$successMessage .= "<META HTTP-EQUIV=REFRESH CONTENT='1; URL=./viewAllRooms.php'>";

}

**else**

{

$errorMessage .= "Insert for ".$\_POST['addRoomFormRoomID']." failed.";

}

}

?>

<h1>Add Event</h1>

<h2>Add Event: Single</h2>

<form id="addRoomForm" name="addRoomForm" action="<?php **echo** $\_SERVER['PHP\_SELF']; ?>" method="post" autocomplete="off">

<fieldset name="addRoomFormFieldSet" id="addRoomFormFieldSet" class="userFormFieldSet">

<legend>Add Event: Single</legend>

<!-- Displays the success and error messages -->

<div id="errorMessage" name="errorMessage"><?php **echo** $errorMessage; ?></div>

<div id="successMessage" name="successMessage"><?php **echo** $successMessage; ?></div>

<label for="addRoomFormRoomID">RoomID:</label>

<input type="text" name="addRoomFormRoomID" maxlength="10" autofocus value="<?php **echo** $\_POST['addRoomFormRoomID']; ?>" />

<label for="addRoomFormRoomName">Room Name:</label>

<input type="text" name="addRoomFormRoomName" maxlength="40" autofocus value="<?php **echo** $\_POST['addRoomFormRoomName']; ?>" />

<label for="addRoomFormCapacity">Capacity:</label>

<input type="Capacity" name="addRoomFormCapacity" maxlength="4" value="<?php **echo** $\_POST['addRoomFormCapacity']; ?>" />

<input type="submit" value="Add Room" id="addRoomFormSubmit" name="addRoomFormSubmit" form="addRoomForm" />

</fieldset>

</form>

<?php

}

**else**

{

*// The user is shown a suitable error message if they aren't eligible to view the page*

**echo** "<p>".returnUserLevelError($bookingSystem, $userLevelNeeded)."</p>";

}

?>

</body>

</html>

### addUserImport.php

|  |
| --- |
|  |

<?php

**include\_once** "./includes/SQLDetails.php";

*// Error messages for the data fields*

$errorMessage = "";

**if** ($\_POST['importAddUserFormSubmit'])

{

**if** (!isset($\_FILES['importAddUserFormCSVUpload']['tmp\_name']))

$errorMessage .= "Please upload a CSV.<br />";

}

?>

<!DOCTYPE html>

<html>

<head>

<!-- This includes the stylesheet for the system -->

<link rel="stylesheet" href="./includes/styles.css" type="text/css" />

<!-- Includes the header, which includes other useful files, checks user login details and generates an appropriate title -->

<?php **include** "./includes/header.php"; ?>

</head>

<body>

<?php

*// Includes the menu for navigating the site, which will be generated differently for each user*

**include** "./includes/menu.php";

$userLevelNeeded = 3;

**if**(getUserLevel($bookingSystem, $\_COOKIE['userID']) >= $userLevelNeeded)

{

?>

<h1>Admin Panel</h1>

<h2>Create Events</h2>

<h3>Add a Single User</h3>

<form id="importAddUserForm" name="importAddUserForm" action="<?php **echo** $\_SERVER['PHP\_SELF']; ?>" method="post" autocomplete="off" enctype="multipart/form-data">

<fieldset name="importAddUserFormFieldSet" id="importAddUserFormFieldSet" class="createSinglEventFieldSet">

<legend>Add User: Import</legend>

<!-- Error and success messages -->

<div id='errorMessage' name='errorMessage'><?php **echo** $errorMessage; ?></div>

<div id='successMessage' name='successMessage'><?php **echo** $successMessage; ?></div>

<label for="importAddUserFormCSVUpload">Upload a user CSV containing: First Name; Last Name; and User ID; </label> <!-- Need to change containing fields -->

<input type="file" name="importAddUserFormCSVUpload" id="importAddUserFormCSVUpload" />

<input type="submit" name="importAddUserFormSubmit" />

<br />

<br />

<?php

**if** (isset($\_FILES['importAddUserFormCSVUpload']))

{

*// Starts to extract the data from the file that has been uploaded*

$fullImportedCSV = file($\_FILES['importAddUserFormCSVUpload']['tmp\_name']);

$CSVHeaders = explode(",", $fullImportedCSV[0]); *// This keeps the headings*

$importedCSV = $fullImportedCSV;

array\_shift($importedCSV); *// This removes the headings element of the array*

**foreach**($importedCSV **as** $row)

{

$row = explode(",", $row); *// Put in here a splitter on ',' to break up the string into elements of an array*

$foreachCount = 0;

**foreach**($row **as** $rowItem) *// A loop to strip spaces, tabs and other dangerous characters*

{

$row[$foreachCount] = trim($row[$foreachCount]);

$foreachCount++;

}

$insertArray = **array**("firstName"=>$row[0], "lastName"=>$row[1], "userID"=>$row[2], "password"=>saltAndHashPassword($row[2], "password"));

**if** (preg\_match('~[0-9]~', $row[2])) *// If the user's code contains numbers then they are a student, else they are a teacher*

{

$insertArray["userLevel"] = 1;

$insertArray["emailAddress"] = $row[2]."@kps.woodard.co.uk"; *// You can figure out the school email address if they are a student, so we will add this now*

}

**else**

{

$insertArray["userLevel"] = 2;

$insertArray["emailAddress"] = $row[0].".".$row[1]."@kps.woodard.co.uk";

}

**if**($bookingSystem->insertTable("Users", $insertArray))

**echo** "The user '".$row[0]." ".$row[1]."' was added sucessfully.<br />";

**else**

**echo** "There was an error when adding the user '".$row[0]." ".$row[1]."'. This may be because a user of the same user ID exists.<br />";

}

}

?>

</fieldset>

</form>

<?php

}

**else**

{

**echo** "<p>".returnUserLevelError($bookingSystem, $userLevelNeeded)."</p>";

}

?>

</body>

</html>

### addUserSingle.php

|  |
| --- |
|  |

<?php

*// Includes other files needed for database connectivity and for system-wide functionality*

**include\_once** "./includes/functions.php";

**include\_once** "./includes/SQLDetails.php";

*// Sets the variable for storing error messages*

$errorMessage = "";

**if** ($\_POST['registerFormSubmit'])

{

**if** ($\_POST['registerFormUserID'] == "")

$errorMessage .= "Please enter a user ID.<br />";

**if** ($\_POST['registerFormEmailAddress'] == "")

$errorMessage .= "Please enter an email address.<br />";

**if** ($\_POST['registerFormPassword'] == "")

$errorMessage .= "Please enter a password.<br />";

**if** ($\_POST['registerFormPasswordConfirm'] == "")

$errorMessage .= "Please enter a password confirmation.<br />";

**if** ($\_POST['registerFormFirstName'] == "")

$errorMessage .= "Please enter a first name.<br />";

**if** ($\_POST['registerFormLastName'] == "")

$errorMessage .= "Please enter a last name.<br />";

**if** ($\_POST['registerFormPassword'] <> $\_POST['registerFormPasswordConfirm'])

$errorMessage .= "The password and confirm password do not match.";

}

**if** (($errorMessage == "") **AND** (isset($\_POST['registerFormSubmit'])))

{

**if**($bookingSystem->insertTable("Users", **array**("userID"=>$\_POST['registerFormUserID'], "emailAddress"=>$\_POST['registerFormEmailAddress'], "password"=>saltAndHashPassword($\_POST['registerFormUserID'], $\_POST['registerFormPassword']), "firstName"=>$\_POST['registerFormFirstName'], "lastName"=>$\_POST['registerFormLastName'], "userLevel"=>$\_POST['registerFormUserLevel'])))

{

*// Success message which will redirect the user back to the viewAllUsers.php page*

$successMessage = "Saving changes...";

$successMessage .= "<META HTTP-EQUIV=REFRESH CONTENT='1; URL=./viewAllUsers.php'>";

}

**else**

{

$errorMessage .= "There was a problem inserting the data to the table.";

}

}

?>

<!DOCTYPE html>

<html>

<head>

<<!-- This includes the stylesheet for the system -->

<link rel="stylesheet" href="./includes/styles.css" type="text/css" />

<!-- Includes the header, which includes other useful files, checks user login details and generates an appropriate title -->

<?php **include** "./includes/header.php"; ?>

</head>

<body>

<!-- Includes the menu for navigating the site, which will be generated differently for each user -->

<?php **include\_once** "./includes/menu.php";

$userLevelNeeded = 3;

**if**(getUserLevel($bookingSystem, $\_COOKIE['userID']) >= $userLevelNeeded)

{

?>

<h1>Admin Panel</h1>

<h2>Add Users</h2>

<h3>Manually</h3>

<form id="registerForm" name="registerForm" action="<?php **echo** $\_SERVER['PHP\_SELF']; ?>" method="post" autocomplete="off">

<fieldset name="registerFormFieldSet" id="registerFormFieldSet" class="userFormFieldSet">

<legend>Manually Register User</legend>

<!-- Displays the success and error messages -->

<div id="errorMessage" name="errorMessage"><?php **echo** $errorMessage; ?></div>

<div id="successMessage" name="successMessage"><?php **echo** $successMessage; ?></div>

<label for="registerFormUserID">User ID:</label>

<input type="text" name="registerFormUserID" maxlength="20" autofocus value="<?php **echo** $\_POST['registerFormUserID']; ?>" />

<label for="registerFormUserID">Email Address:</label>

<input type="email" name="registerFormEmailAddress" maxlength="254" autofocus value="<?php **echo** $\_POST['registerFormEmailAddress']; ?>" />

<label for="registerFormPassword">Password:</label>

<input type="password" name="registerFormPassword" maxlength="32" value="<?php **echo** $\_POST['registerFormPassword']; ?>" />

<label for="registerFormPasswordConfirm">Password Confirm:</label>

<input type="password" name="registerFormPasswordConfirm" maxlength="32" value="<?php **echo** $\_POST['registerFormPasswordConfirm']; ?>" />

<label for="registerFormFirstName">First Name:</label>

<input type="text" name="registerFormFirstName" maxlength="35" value="<?php **echo** $\_POST['registerFormFirstName']; ?>" />

<label for="registerFormLastName">Last Name:</label>

<input type="text" name="registerFormLastName" maxlength="50" value="<?php **echo** $\_POST['registerFormLastName']; ?>" />

<label for="registerFormUserLevel">User Level:</label>

<select name="registerFormUserLevel">

<?php

**foreach** ($bookingSystem->queryTable("UserLevels", "\*", **NULL**, "userLevel") **as** $result) *// Builds a drop down list with all user levels*

{

**if** ($registerFormUserLevel == $result['userLevel'])

$selected = " selected";

**else**

$selected = "";

**echo** "<option".$selected." value='".$result['userLevel']."'>".$result['title']."</option>";

}

?>

</select>

<input type="submit" value="Register User" id="registerFormSubmit" name="registerFormSubmit" form="registerForm" />

</fieldset>

</form>

<?php

}

**else**

{

**echo** "<p>".returnUserLevelError($bookingSystem, $userLevelNeeded)."</p>";

}

?>

</body>

</html>

### adminPanel.php

|  |
| --- |
|  |

<!DOCTYPE html>

<html>

<head>

<!-- **This** includes the stylesheet **for** the system -->

<link rel="stylesheet" href="./includes/styles.css" type="text/css" />

<!-- Includes the header, which includes other useful files, checks user login details **and** generates an appropriate title -->

<?php **include** "./includes/header.php"; ?>

</head>

<body>

<?php

*// Includes the menu for navigating the site, which will be generated differently for each user*

**include** "./includes/menu.php";

*// Only administrators can view the content of this page*

$userLevelNeeded = 3;

**if**(getUserLevel($bookingSystem, $\_COOKIE['userID']) >= $userLevelNeeded)

{

*// A hierarchical structure of the administrator-only pages*

**echo** "<h1>Admin Panel</h1>";

**echo** "<h2>Users</h2>";

**echo** "<h3><a href='viewAllUsers.php'>View All Users</a></h3>";

**echo** "<h3><a href='addUserImport.php'>Add Users: By Import</a></h3>";

**echo** "<h3><a href='addUserSingle.php'>Add Users: Single</a></h3>";

**echo** "<br />";

**echo** "<h2>Rooms</h2>";

**echo** "<h3><a href='viewAllRooms.php'>View All Rooms</a></h3>";

**echo** "<h3><a href='addRoomImport.php'>Add Rooms: By Import</a></h3>";

**echo** "<h3><a href='addRoomSingle.php'>Add Rooms: Single</a></h3>";

**echo** "<br />";

**echo** "<h2>Holidays</h2>";

**echo** "<h3><a href='viewAllHolidays.php'>View All Holidays</a></h3>";

**echo** "<h3><a href='addHolidaySingle.php'>Add Holidays: Single</a></h3>";

}

**else**

{

**echo** "<p>".returnUserLevelError($bookingSystem, $userLevelNeeded)."</p>";

}

?>

</body>

</html>

### editEventSingle.php

|  |
| --- |
|  |

<?php

**include\_once** "./includes/SQLDetails.php";

**include\_once** "./includes/functions.php";

*// Error message variable to store error messages*

$errorMessage = "";

**if** (isset($\_POST['editFormDelete']))

{

*// Query to delete the record*

**if** ($bookingSystem->deleteFromTable("Events", **array**("ID"=>$\_POST['editFormEventID'])))

{

*// Tells the user which event ID has been deleted and redirects them*

$successMessage = "Event ID ".$\_POST['editFormEventID']." has been deleted successfully.";

$successMessage .= "<br />Loading viewAllEvents.php...";

*// Unserialises the GET variables and returns the user to viewAllEvents.php with what ever the parameters were set as before the user wanted to add an event*

$returnGET = unserialize($\_POST['editFormReturnGET']);

$successMessage .= "<META HTTP-EQUIV=REFRESH CONTENT='5; URL=./viewAllEvents.php?view=".$returnGET['view'];

**if** (isset($returnGET['roomID']))

$successMessage .= "&roomID=".$returnGET['roomID'];

**else** **if** (isset($returnGET['teacherID']))

$successMessage .= "&teacherID=".$returnGET['teacherID'];

**if** (isset($returnGET['referenceWeek']))

$successMessage .= "&referenceWeek=".$returnGET['referenceWeek'];

$successMessage .= "'>";

}

**else**

{

$errorMessage = "Something went wrong when deleting the event.";

}

}

**else** **if** ($\_POST['editFormSubmit'])

{

*// Sets the POST variables into more conviniently named variables that can be assigned from any other method*

$editFormName = $\_POST['editFormName'];

$editFormOwnerID = $\_POST['editFormOwnerID'];

$editFormStartTimestamp = $\_POST['editFormStartTimestamp'];

$editFormRoomID = $\_POST['editFormRoomID'];

*// If the form has been submitted, we test to see if any of the fields are blank*

**if** ($editFormName == "")

$errorMessage .= "Please enter an event name.<br />";

**if** ($editFormOwnerID == "")

$errorMessage .= "There is no owner attached to this event.<br />";;

**if** ($editFormRoomID == "")

$errorMessage .= "There is no room attached to this.<br />";

*// Set variables to post values*

}

**elseif** (!isset($\_GET['eventID']))

{

*// If there is no userID speicifed by GET and there's no POST submission set, something has gone wrong and we tell the user this*

$errorMessage .= "Something has gone wrong; no event ID specified.<br />";

}

**else**

{

*// Else, everything is working and we assign the form variables the values of the userID taken from the database*

$eventResult = $bookingSystem->queryTable("Events", **array**("ID"=>$\_GET['eventID']), 1);

$editFormEventID = $\_GET['eventID'];

$editFormName = $eventResult['name'];

$editFormOwnerID = $eventResult['ownerID'];

$editFormStartTimestamp = $eventResult['startTimestamp'];

$editFormRoomID = $eventResult['roomID'];

*// This code figures out what the timestamp of the date and period is*

$explodedTimestamp = explode("T", date("Y-m-d**\T**Hi", $editFormStartTimestamp));

$eventDate = $explodedTimestamp[0];

$periodTime = $explodedTimestamp[1];

$editFormPeriod = $bookingSystem->queryTable("Periods", **array**("startTime"=>$periodTime), 1, "periodNumber")['periodNumber'];

}

*// If the form has been submitted and there have been no errors found*

**if** (($\_POST['editFormSubmit']) **and** ($errorMessage == ""))

{

$editFormEventID = $\_POST['editFormEventID'];

$editFormName = $\_POST['editFormName'];

$editFormOwnerID = $\_POST['editFormOwnerID'];

$editFormDate = $\_POST['editFormDate'];

$editFormRoomID = $\_POST['editFormRoomID'];

$editFormPeriod = $\_POST['editFormPeriod'];

$updateArray = **array**(

"name"=>$editFormName,

"ownerID"=>$editFormOwnerID,

"roomID"=>$editFormRoomID

);

*// The code below figures out what the start timestamp for the event will be*

$periodTime = $bookingSystem->queryTable("Periods", **array**("periodNumber"=>$editFormPeriod), 1, "periodNumber")['startTime'];

$insertPeriodTimeFormatted = formatTime($periodTime); *// Formats the period time with a colon*

$editFormStartTimestamp = strtotime(date("Y-m-d", strtotime($editFormDate))."T".$insertPeriodTimeFormatted); *// Formats the date and time for the period into the correct format then to a unix timestamp*

$updateArray['startTimestamp'] = $editFormStartTimestamp;

**if**($bookingSystem->updateTable("Events", $updateArray, **array**("ID"=>$editFormEventID)))

{

$errorMessage = "";

$successMessage = "Saving changes...";

*// If there was a get string provided*

**if** (strlen($\_POST['editFormReturnGET']) > 0)

{

*// Unserialises the GET variables and returns the user to viewAllEvents.php with what ever the parameters were set as before the user wanted to add an event*

$returnGET = unserialize($\_POST['editFormReturnGET']);

$successMessage .= "<META HTTP-EQUIV=REFRESH CONTENT='1; URL=./viewAllEvents.php?view=".$returnGET['view'];

**if** (isset($returnGET['roomID']))

$successMessage .= "&roomID=".$returnGET['roomID'];

**else** **if** (isset($returnGET['teacherID']))

$successMessage .= "&teacherID=".$returnGET['teacherID'];

**if** (isset($returnGET['referenceWeek']))

$successMessage .= "&referenceWeek=".$returnGET['referenceWeek'];

$successMessage .= "'>";

}

**else**

{

*// Redirects the user to addEvent.php if there were no GET variables set*

$successMessage .= "<META HTTP-EQUIV=REFRESH CONTENT='1; URL=./viewAllEvents.php'>";

}

}

**else**

{

$errorMessage = "There was a problem inserting the data to the table.";

}

}

?>

<!DOCTYPE html>

<html>

<head>

<!-- This includes the stylesheet for the system -->

<link rel="stylesheet" href="./includes/styles.css" type="text/css" />

<!-- Includes the header, which includes other useful files, checks user login details and generates an appropriate title -->

<?php **include** "./includes/header.php"; ?>

</head>

<body>

<!-- // Includes the menu for navigating the site, which will be generated differently for each user -->

<?php **include\_once** "./includes/menu.php";

$userLevelNeeded = 2;

**if**(getUserLevel($bookingSystem, $\_COOKIE['userID']) >= $userLevelNeeded)

{

?>

<h1><a href="./viewEvents.php">View Events</a></h1>

<h2>Edit Single Event</h2>

<form id="editForm" name="editForm" action="<?php **echo** $\_SERVER['PHP\_SELF']; ?>" method="post" autocomplete="off">

<fieldset name="editFormFieldSet" id="editFormFieldSet" class="userFormFieldSet">

<legend>Manually edit Event</legend>

<!-- Displays the success and error messages -->

<div id="errorMessage" name="errorMessage"><?php **echo** $errorMessage; ?></div>

<div id="successMessage" name="successMessage"><?php **echo** $successMessage; ?></div>

<label for="editFormName">Event Name:</label>

<input type="text" name="editFormName" maxlength="20" value="<?php **echo** $editFormName; ?>" />

<?php

*// This makes sure that only administrators or owners of the event can change the owner*

**if** ((getUserLevel($bookingSystem, $\_COOKIE['userID']) == 3) **OR** ($\_COOKIE['userID'] == $editFormOwnerID))

$disabled = "";

**else**

$disabled = " disabled";

?>

<!-- Hidden field for the event ID -->

<input type="hidden" name="editFormEventID" id="editFormEventID" value="<?php **echo** $editFormEventID; ?>" />

<label for="editFormOwnerID">Teacher ID:</label> <!-- Different for administrators and teachers -->

<select<?php **echo** $disabled; ?> name="editFormOwnerID">

<option value="" disabled selected>Select a teacher</option>

<?php

$returnedTeacherList = $bookingSystem->queryTable("Users", **array**("userLevel" => 2), **NULL**, " userID, lastName, firstName", "ASC"); *// Picks all teachers*

*// Creates a multidimensional array for the foreach loop*

**if** (isset($returnedTeacherList['userID']))

$teacherResult[0] = $returnedTeacherList;

**else**

$teacherResult = $returnedTeacherList;

**foreach** ($teacherResult **as** $teacher)

{

**if** (isset($editFormOwnerID) **AND** ($teacher['userID'] == $editFormOwnerID))

$selected = " selected";

**else**

$selected = "";

**echo** "<option value='".$teacher['userID']."'".$selected.">[".$teacher['userID']."] ".$teacher['firstName']." ".$teacher['lastName']."</option>";

}

?>

</select>

<label for="editFormDate">Date:</label>

<input type="date" name="editFormDate" value="<?php **echo** date("Y-m-d", $editFormStartTimestamp); ?>" />

<label for="editFormPeriod">Period:</label>

<select name="editFormPeriod">

<option value="" disabled selected>Select a period</option>

<?php

$periodResult = $bookingSystem->queryTable("Periods", "\*", **NULL**, "periodNumber", "ASC"); *// Picks all periods*

**foreach** ($periodResult **as** $period)

{

*// If the period number sought is the same as the one in the loop, then set it as selected*

**if** (isset($editFormPeriod) **AND** ($period['periodNumber'] == $editFormPeriod))

$selected = " selected";

**else**

$selected = "";

**echo** "<option value='".$period['periodNumber']."'".$selected.">Period ".$period['periodNumber']." (".formatTime($period['startTime'])." to ".formatTime($period['endTime']).")</option>";

}

?>

</select>

<label for="editFormRoomID">Room:</label>

<select name="editFormRoomID">

<option value="" disabled selected>Select a Room</option>

<?php

$returnedRoomList = $bookingSystem->queryTable("Rooms", "\*", **NULL**, "roomID", "ASC"); *// Picks all rooms*

*// Creates a multidimensional array for the foreach loop*

**if** (isset($returnedRoomList['roomID']))

$roomResult[0] = $returnedRoomList;

**else**

$roomResult = $returnedRoomList;

**foreach** ($roomResult **as** $room)

{

**if** (isset($editFormRoomID) **AND** ($room['roomID'] == $editFormRoomID))

$selected = " selected";

**else**

$selected = "";

**if** ($room['capacity'] == "")

$room['capacity'] = "Unknown capacity"; *// If there is no capacity specified, unknown is listed next to the room.*

**echo** "<option value='".$room['roomID']."'".$selected.">".$room['name']." (".$room['capacity'].")</option>";

}

?>

</select>

<!-- A hidden field to ensure that we don't lose the GET variables from the previous page when we submit the form -->

<input type="hidden" value="<?php **echo** htmlspecialchars($\_GET['serializedGetVariables']); ?>" id="editFormReturnGET" name="editFormReturnGET" form="editForm" />

<input type="submit" value="Save Edited Event" id="editFormSubmit" name="editFormSubmit" form="editForm" />

<input type="submit" value="Delete Event" id="editFormDelete" name="editFormDelete" form="editForm" />

</fieldset>

</form>

<?php

}

**else**

{

?>

<p><?php returnUserLevelError($bookingSystem, $userLevelNeeded); ?></p>

<?php

}

?>

</body>

</html>

### editHolidaySingle.php

|  |
| --- |
|  |

<?php

*// Builds the success and error messages*

$errorMessage = "";

$successMessage = "";

**if** ($\_POST['editSingleHolidayFormSubmit'])

{

*// Error messages for variables being 0 or empty*

**if** ($\_POST['editSingleHolidayFormLabel'] == "")

$errorMessage .= "Please enter a holiday label.<br />";

**if** ($\_POST['editSingleHolidayFormStart'] == 0) *// We can presume that 0 is an unfilled field*

$errorMessage .= "Please enter a holiday start.<br />";

**if** ($\_POST['editSingleHolidayFormEnd'] == 0) *// We can presume that 0 is an unfilled field*

$errorMessage .= "Please enter a holiday end.<br />";

*// Error message for the start timestamp being after the end timestamp*

**if** ($\_POST['editSingleHolidayFormStart'] > $\_POST['editSingleHolidayFormEnd'])

$errorMessage .= "The start date is after the end date of the holiday.";

}

?>

<!DOCTYPE html>

<html>

<head>

<!-- This includes the stylesheet for the system -->

<link rel="stylesheet" href="./includes/styles.css" type="text/css" />

<!-- Includes the header, which includes other useful files, checks user login details and generates an appropriate title -->

<?php **include** "./includes/header.php"; ?>

</head>

<body>

<?php

*// Includes the menu for navigating the site, which will be generated differently for each user*

**include** "./includes/menu.php";

*// This page's content can only be accessed by administrators*

$userLevelNeeded = 3;

**if**(getUserLevel($bookingSystem, $\_COOKIE['userID']) >= $userLevelNeeded)

{

?>

<?php

*// If the form has been submitted and there's no errors*

**if**((isset($\_POST['editSingleHolidayFormSubmit'])) **AND** ($errorMessage == ""))

{

$holidayLabel = $\_POST['editSingleHolidayFormLabel'];

$holidayStart = strtotime($\_POST['editSingleHolidayFormStart']);

$holidayEnd = strtotime($\_POST['editSingleHolidayFormEnd']);

*// Checks if the insert was successful or not*

**if**($bookingSystem->insertTable("Holidays", **array**("label"=>$holidayLabel, "startTimestamp"=>$holidayStart, "endTimestamp"=>$holidayEnd)))

{

$successMessage .= "Insert successful.";

$successMessage .= "<META HTTP-EQUIV=REFRESH CONTENT='1; URL=./viewAllHolidays.php'>";

}

**else**

{

$errorMessage .= "Insert for ".$holidayLabel." failed.<br />";

}

}

?>

<h1><a href="adminPanel.php">Admin Panel</a></h1>

<h2>Edit Holiday</h2>

<h3><a href="addHolidaySingle.php">Edit a Single Holiday</a></h3>

<form id="editSingleHolidayForm" name="editSingleHolidayForm" action="<?php **echo** $\_SERVER['PHP\_SELF']; ?>" method="post" autocomplete="off">

<fieldset name="editSingleHolidayFormFieldSet" id="editSingleHolidayFormFieldSet" class="userFormFieldSet">

<legend>Add a Single Holiday</legend>

<!-- Shows the user the success and error messages -->

<div id="errorMessage" name="errorMessage"><?php **echo** $errorMessage; ?></div>

<div id="successMessage" name="successMessage"><?php **echo** $successMessage; ?></div>

<label for="editSingleHolidayFormLabel">Holiday Label:</label>

<input type="text" name="editSingleHolidayFormLabel" maxlength="50" autofocus value="<?php **echo** $\_POST['editSingleHolidayFormLabel']; ?>" />

<label for="editSingleHolidayFormStart">Holiday Start:</label>

<input type="date" name="editSingleHolidayFormStart" value="<?php **echo** $\_POST['editSingleHolidayFormStart']; ?>" />

<label for="editSingleHolidayFormEnd">Holiday End:</label>

<input type="date" name="editSingleHolidayFormEnd" value="<?php **echo** $\_POST['editSingleHolidayFormEnd']; ?>" />

<input type="submit" value="edit Holiday" id="editSingleHolidayFormSubmit" name="editSingleHolidayFormSubmit" form="editSingleHolidayForm" />

</fieldset>

</form>

<?php

}

**else**

{

**echo** "<p>".returnUserLevelError($bookingSystem, $userLevelNeeded)."</p>";

}

?>

</body>

</html>

### editRoomSingle.php

|  |
| --- |
|  |

<?php

**include\_once** "./includes/SQLDetails.php";

**include\_once** "./includes/functions.php";

*// Error message variable*

$errorMessage = "";

*// Code to be executed if the room is to be deleted*

**if** (isset($\_POST['editFormDelete']))

{

**if** ($bookingSystem->deleteFromTable("Rooms", **array**("roomID"=>$\_POST['editFormRoomID'])))

{

$successMessage = $\_POST['editFormRoomID']." has been deleted successfully.";

$successMessage .= "<META HTTP-EQUIV=REFRESH CONTENT='1; URL=./viewAllRooms.php'>";

}

**else**

{

$errorMessage = "Something went wrong when deleting the room.";

}

}

**else** **if** ($\_POST['editFormSubmit'])

{

$editFormRoomID = $\_POST['editFormRoomID'];

$editFormRoomName = $\_POST['editFormRoomName'];

$editFormCapacity = $\_POST['editFormCapacity'];

*// If the form has been submitted, we assign the values of the POST variables into the form variables*

*//$editFormUsername = $\_POST['editFormUsername'];*

}

**elseif** (!isset($\_GET['roomID']))

{

*// If there is no userID speicifed by GET and there's no POST submission set, something has gone wrong and we tell the user this*

$errorMessage .= "Something has gone wrong; no room ID specified.<br />";

}

**else**

{

*// Else, everything is working and we assign the form variables the values of the userID taken from the database*

$roomResult = $bookingSystem->queryTable("Rooms", **array**("roomID"=>$\_GET['roomID']), 1, "roomID");

$editFormRoomID = $\_GET['roomID'];

$editFormRoomName = $roomResult['name'];

$editFormCapacity = $roomResult['capacity'];

}

*// If the form has been submitted and there have been no errors found*

**if** (($\_POST['editFormSubmit']) **and** ($errorMessage == ""))

{

$updateArray = **array**(

"roomID"=>$editFormRoomID,

"name"=>$editFormRoomName,

"capacity"=>$editFormCapacity

);

**if**($bookingSystem->updateTable("Rooms", $updateArray, **array**("roomID"=>$editFormRoomID), **true**))

{

$errorMessage = "";

$successMessage = "Saving changes...";

$successMessage .= "<META HTTP-EQUIV=REFRESH CONTENT='1; URL=./viewAllRooms.php'>";

}

**else**

{

$errorMessage = "There was a problem inserting the data to the table.";

}

}

?>

<!DOCTYPE html>

<html>

<head>

<!-- This includes the stylesheet for the system -->

<link rel="stylesheet" href="./includes/styles.css" type="text/css" />

<!-- Includes the header, which includes other useful files, checks user login details and generates an appropriate title -->

<?php **include** "./includes/header.php"; ?>

</head>

<body>

<!-- Includes the menu for navigating the site, which will be generated differently for each user -->

<?php **include\_once** "./includes/menu.php";

$userLevelNeeded = 3;

**if**(getUserLevel($bookingSystem, $\_COOKIE['userID']) >= $userLevelNeeded)

{

?>

<h1><a href="./admin.php">Admin Panel</a></h1>

<h2><a href="./viewAllRooms.php">View All Rooms</a></h2>

<h3><a href="<?php **echo** $\_SERVER['PHP\_SELF']."?roomID=".$\_GET['roomID']; ?>">Edit Single Room</a></h3>

<form id="editForm" name="editForm" action="<?php **echo** $\_SERVER['PHP\_SELF']; ?>" method="post" autocomplete="off">

<fieldset name="editFormFieldSet" id="editFormFieldSet" class="userFormFieldSet">

<legend>Manually edit Room</legend>

<div id="errorMessage" name="errorMessage"><?php **echo** $errorMessage; ?></div>

<div id="successMessage" name="successMessage"><?php **echo** $successMessage; ?></div>

<label for="editFormRoomID">Room ID:</label>

<input type="text" name="editFormRoomIDDisabled" maxlength="10" disabled value="<?php **echo** $editFormRoomID; ?>" />

<input type="hidden" name="editFormRoomID" value="<?php **echo** $editFormRoomID; ?>" />

<label for="editFormRoomName">Room Name:</label>

<input type="text" name="editFormRoomName" maxlength="40" autofocus value="<?php **echo** $editFormRoomName; ?>" />

<label for="editFormCapacity">Room Capacity:</label>

<input type="number" name="editFormCapacity" maxlength="4" value="<?php **echo** $editFormCapacity; ?>" />

<input type="submit" value="Save Edited Room" id="editFormSubmit" name="editFormSubmit" form="editForm" />

<input type="submit" value="Delete Room" id="editFormDelete" name="editFormDelete" form="editForm" />

</fieldset>

</form>

<?php

}

**else**

{

?>

<p><?php returnUserLevelError($bookingSystem, $userLevelNeeded); ?></p>

<?php

}

?>

</body>

</html>

### editUserSingle.php

|  |
| --- |
|  |

<?php

**include\_once** "./includes/SQLDetails.php";

**include\_once** "./includes/functions.php";

$errorMessage = "";

*// Code to be run if the user is to be deleted, including a redirect if the deletion was successful*

**if** (isset($\_POST['editFormDelete']))

{

**if** ($bookingSystem->deleteFromTable("Users", **array**("userID"=>$\_POST['editFormUserID'])))

{

$successMessage = $\_POST['editFormUserID']." has been deleted successfully.";

$successMessage .= "<META HTTP-EQUIV=REFRESH CONTENT='1; URL=./viewAllUsers.php'>";

}

**else**

{

$errorMessage = "Something went wrong when deleting the user.";

}

}

**else** **if** ($\_POST['editFormSubmit'])

{

$editFormUserID = $\_POST['editFormUserID'];

$editFormEmailAddress = $\_POST['editFormEmailAddress'];

$editFormPassword = $\_POST['editFormPassword'];

$editFormFirstName = $\_POST['editFormFirstName'];

$editFormLastName = $\_POST['editFormLastName'];

$editFormUserLevel = $\_POST['editFormUserLevel'];

*// If the form has been submitted, we test to see if any of the fields are blank*

**if** ($editFormUserID == "")

$errorMessage .= "Please enter a user ID.<br />";

**if** ($editFormEmailAddress == "")

$errorMessage .= "Please enter an email address.<br />";

**if** ($editFormFirstName == "")

$errorMessage .= "Please enter a first name.<br />";

**if** ($editFormLastName == "")

$errorMessage .= "Please enter a last name.<br />";

*// If the form has been submitted, we assign the values of the POST variables into the form variables*

$editFormUserID = $\_POST['editFormUserID'];

}

**elseif** (!isset($\_GET['userID']))

{

*// If there is no userID speicifed by GET and there's no POST submission set, something has gone wrong and we tell the user this*

$errorMessage .= "Something has gone wrong; no user ID specified.<br />";

}

**else**

{

*// Else, everything is working and we assign the form variables the values of the userID taken from the database*

$userResult = $bookingSystem->queryTable("Users", **array**("userID"=>$\_GET['userID']), 1, "userID");

$editFormUserID = $\_GET['userID'];

$editFormEmailAddress = $userResult['emailAddress'];

$editFormPassword = "";

$editFormFirstName = $userResult['firstName'];

$editFormLastName = $userResult['lastName'];

$editFormUserLevel = $userResult['userLevel'];

}

*// If the form has been submitted and there have been no errors found*

**if** (($\_POST['editFormSubmit']) **and** ($errorMessage == ""))

{

$updateArray = **array**(

"userID"=>$editFormUserID,

"firstName"=>$editFormFirstName,

"lastName"=>$editFormLastName,

"emailAddress"=>$editFormEmailAddress,

"userLevel"=>$editFormUserLevel

);

**if** ($editFormPassword != "")

$updateArray['password'] = saltAndHashPassword($editFormUserID, $editFormPassword); *// If the password field isn't blank, we insert the new hashed password to the update array for the database*

*// The record is attempted to be updated:*

**if**($bookingSystem->updateTable("Users", $updateArray, **array**("userID"=>$editFormUserID)))

{

$errorMessage = "";

$successMessage = "Saving changes...";

$successMessage .= "<META HTTP-EQUIV=REFRESH CONTENT='1; URL=./viewAllUsers.php'>";

}

**else**

{

$errorMessage = "There was a problem inserting the data to the table.";

}

}

?>

<!DOCTYPE html>

<html>

<head>

<!-- This includes the stylesheet for the system -->

<link rel="stylesheet" href="./includes/styles.css" type="text/css" />

<!-- Includes the header, which includes other useful files, checks user login details and generates an appropriate title -->

<?php **include** "./includes/header.php"; ?>

</head>

<body>

<!-- Includes the menu for navigating the site, which will be generated differently for each user -->

<?php **include\_once** "./includes/menu.php";

$userLevelNeeded = 3;

**if**(getUserLevel($bookingSystem, $\_COOKIE['userID']) >= $userLevelNeeded)

{

?>

<h1><a href="./admin.php">Admin Panel</a></h1>

<h2><a href="./viewAllUsers.php">All Users</a></h2>

<h3><a href="<?php **echo** $\_SERVER['PHP\_SELF']."?userID=".$\_GET['userID']; ?>">Edit Single User</a></h3>

<form id="editForm" name="editForm" action="<?php **echo** $\_SERVER['PHP\_SELF']; ?>" method="post" autocomplete="off">

<fieldset name="editFormFieldSet" id="editFormFieldSet" class="userFormFieldSet">

<legend>Manually edit User</legend>

<div id="errorMessage" name="errorMessage"><?php **echo** $errorMessage; ?></div>

<div id="successMessage" name="successMessage"><?php **echo** $successMessage; ?></div>

<label for="editFormUserID">User ID:</label>

<input type="text" name="editFormUserID" maxlength="20" disabled value="<?php **echo** $editFormUserID; ?>" />

<label for="editFormUserID">Email Address:</label>

<input type="email" name="editFormEmailAddress" maxlength="254" autofocus value="<?php **echo** $editFormEmailAddress; ?>" />

<label for="editFormPassword">Password:</label>

<input type="password" name="editFormPassword" maxlength="32" placeholder="If left blank, password is unchanged" value="<?php **echo** $editFormPassword; ?>" />

<label for="editFormFirstName">First Name:</label>

<input type="text" name="editFormFirstName" maxlength="35" value="<?php **echo** $editFormFirstName; ?>" />

<label for="editFormLastName">Last Name:</label>

<input type="text" name="editFormLastName" maxlength="50" value="<?php **echo** $editFormLastName; ?>" />

<label for="editFormUserLevel">User Level:</label>

<select name="editFormUserLevel">

<?php

*// This stops an administrator changing their user level so that they can't be "locked-out" of the system*

**if** ($\_GET['userID'] == $\_COOKIE['userID'])

{

$disabled = " disabled";

$disabledMessage = "You can't change your own user level from administrator.";

}

**else**

{

$disabled = "";

$disabledMessage = "";

}

**foreach** ($bookingSystem->queryTable("UserLevels", "\*", **NULL**, "userLevel") **as** $result) *// Builds a drop down list with all user levels*

{

**if** ($editFormUserLevel == $result['userLevel'])

$selected = " selected";

**else**

$selected = "";

**echo** "<option".$selected.$disabled." value='".$result['userLevel']."'>".$result['title']."</option>";

}

?>

</select>

<?php **echo** $disabledMessage; ?>

<input type="submit" value="Save Edited User" id="editFormSubmit" name="editFormSubmit" form="editForm" />

<input type="submit" value="Delete User" id="editFormDelete" name="editFormDelete" form="editForm" />

</fieldset>

</form>

<?php

}

**else**

{

?>

<p><?php returnUserLevelError($bookingSystem, $userLevelNeeded); ?></p>

<?php

}

?>

</body>

</html>

### index.php

|  |
| --- |
|  |

<!DOCTYPE html>

<html>

<head>

<!-- **This** includes the stylesheet **for** the system -->

<link rel="stylesheet" href="./includes/styles.css" type="text/css" />

<!-- Includes the header, which includes other useful files, checks user login details **and** generates an appropriate title -->

<?php **include** "./includes/header.php"; ?>

</head>

<body>

<?php

**include\_once** "./includes/createTables.php";

*// Includes the menu for navigating the site, which will be generated differently for each user*

**include** "./includes/menu.php";

*// Test login details are added to the database here. These are used for debugging purposes only and should NOT be uncommented for general use. For each user, the password and user ID are the same*

*/\*$bookingSystem->insertTable("Users", array("userID"=>"3", "emailAddress"=>"3@kps.woodard.co.uk", "password"=>"c6f3ac57944a531490cd39902d0f777715fd005efac9a30622d5f5205e7f6894", "userLevel"=>3, "firstName"=>"Thirty", "lastName"=>"Three"));*

*$bookingSystem->insertTable("Users", array("userID"=>"2", "emailAddress"=>"2@kps.woodard.co.uk", "password"=>"785f3ec7eb32f30b90cd0fcf3657d388b5ff4297f2f9716ff66e9b69c05ddd09", "userLevel"=>2, "firstName"=>"Twenty", "lastName"=>"Two"));*

*$bookingSystem->insertTable("Users", array("userID"=>"1", "emailAddress"=>"1@kps.woodard.co.uk", "password"=>"4fc82b26aecb47d2868c4efbe3581732a3e7cbcc6c2efb32062c08170a05eeb8", "userLevel"=>1, "firstName"=>"Onety", "lastName"=>"One"));\*/*

?>

<p>Welcome to the Kings Priory School Room Booking System (KPS RBS).</p>

<?php

*// Displays a welcome message, depending on whether or not the user is logged in and also depending on what user level their user account is*

**if** (getUserLevel($bookingSystem, $\_COOKIE['userID']) > 0)

**echo** "Your user level is: <strong>".$bookingSystem->queryTable("UserLevels", **array**("userLevel"=>getUserLevel($bookingSystem, $\_COOKIE['userID'])), 1, "userLevel", **false**, **false**)['title']."</strong>. This means that you can view events and access other features using the main menu.";

**else**

**echo** "<p>You must <a href='./login.php'>login</a> to be able to view content.</p>";

?>

</body>

</html>

### login.php

|  |
| --- |
|  |

<?php

**include\_once** "./includes/SQLDetails.php";

**include\_once** "./includes/functions.php";

*// Initialise the success and error messages*

$successMessage = "";

$errorMessage = "";

*// Sets $\_POST['loginFormSubmit'] to false if it isn't set*

**if** (!isset($\_POST['loginFormSubmit']))

$\_POST['loginFormSubmit'] = **false**;

**if** ($\_POST['loginFormSubmit'])

{

*// This lets the user know if it is their user ID or their password that is the issue*

$userIDCheckResponse = $bookingSystem->queryTable("Users", **array**("userID"=>$\_POST['loginFormUserID']), 1, "userID");

**if**($userIDCheckResponse)

{

$loginResponse = $bookingSystem->queryTable("Users", **array**("userID"=>$\_POST['loginFormUserID'], "password"=>saltAndHashPassword($\_POST['loginFormUserID'], $\_POST['loginFormPassword'])), 1, "userID");

**if**($loginResponse)

{

setcookie("userID", $loginResponse['userID'], time() + (60\*60\*24\*7\*2), "/"); *// Sets login details for 2 weeks*

setcookie("userPassword", $loginResponse['password'], time() + (60\*60\*24\*7\*2), "/"); *// Sets login details for 2 weeks*

$successMessage = "Logging in...";

$successMessage .= "<META HTTP-EQUIV=REFRESH CONTENT='1; URL=./index.php'>";

}

**else**

{

$errorMessage = "Incorrect password.";

}

}

**else**

{

$errorMessage = "The user does not exist.";

}

}

**else**

{

*// Defining variables that will be used later, even if they're empty*

$errorMessage = "";

*// This is a security measure - it helps to prevent user ID and password boxes being autofilled*

$\_POST['loginFormUserID'] = "";

$\_POST['loginFormPassword'] = "";

}

?>

<!DOCTYPE html>

<html>

<head>

<!-- This includes the stylesheet for the system -->

<link rel="stylesheet" href="./includes/styles.css" type="text/css" />

<!-- Includes the header, which includes other useful files, checks user login details and generates an appropriate title -->

<?php **include** "./includes/header.php"; ?>

</head>

<body>

<!-- Includes the menu for navigating the site, which will be generated differently for each user -->

<?php **include** "./includes/menu.php";

*// If the user cookies don't exist:*

**if** (!checkUserLogin($bookingSystem, $\_COOKIE['userID'], $\_COOKIE['userPassword']))

{ ?>

<form id="loginForm" name="loginForm" action="<?php **echo** $\_SERVER['PHP\_SELF']; ?>" method="post">

<fieldset name="loginFormFieldSet" id="loginFormFieldSet">

<legend>Login</legend>

<div id="successMessage" name="successMessage"><?php **echo** $successMessage; ?></div>

<div id="errorMessage" name="errorMessage"><?php **echo** $errorMessage; ?></div>

<label for="loginFormUserID">User ID:</label>

<input type="text" name="loginFormUserID" maxlength="254" autofocus value="<?php **echo** $\_POST['loginFormUserID']; ?>" />

<label for="loginFormPassword">Password:</label>

<input type="password" name="loginFormPassword" maxlength="32" value="<?php **echo** $\_POST['loginFormPassword']; ?>" />

<input type="submit" value="Login" id="loginFormSubmit" name="loginFormSubmit" form="loginForm" />

</fieldset>

</form>

<?php

}

**else**

{ ?>

<p>You are already logged in. Please <a href="./logout.php">logout</a> to use the login page.</p>

<?php

}

?>

</body>

</html>

### logout.php

|  |
| --- |
|  |

<?php

**include\_once** "./includes/functions.php";

destroyCookies();

?>

<!DOCTYPE html>

<html>

<head>

<!-- This includes the stylesheet for the system -->

<link rel="stylesheet" href="./includes/styles.css" type="text/css" />

<!-- Includes the header, which includes other useful files, checks user login details and generates an appropriate title -->

<?php **include** "./includes/header.php"; ?>

</head>

<body>

<!-- Includes the menu for navigating the site, which will be generated differently for each user -->

<?php **include\_once** "./includes/menu.php"; ?>

<p>Logging out...</p>

<META HTTP-EQUIV=REFRESH CONTENT='1; URL=./index.php'>

</body>

</html>

### viewAllEvents.php

|  |
| --- |
|  |

<!DOCTYPE html>

<html>

<head>

<!-- **This** includes the stylesheet **for** the system -->

<link rel="stylesheet" href="./includes/styles.css" type="text/css" />

<!-- Includes the header, which includes other useful files, checks user login details **and** generates an appropriate title -->

<?php **include** "./includes/header.php"; ?>

</head>

<body>

<?php

*// Includes the menu for navigating the site, which will be generated differently for each user*

**include** "./includes/menu.php";

*// Any logged in user can view this page*

$userLevelNeeded = 1;

**if**(getUserLevel($bookingSystem, $\_COOKIE['userID']) >= $userLevelNeeded)

{

*// If the view has been set*

**if** (isset($\_GET['view']))

{

*// Checks if only the roomID or only the teacherID has been set*

**if** ((isset($\_GET['roomID'])) **XOR** (isset($\_GET['teacherID'])))

{

**if** (isset($\_GET['referenceWeek'])) *// Checks if the GET is set and sets the referenceWeek to the GET value, and if not uses the current week as the default referenceweek value*

$referenceWeek = $\_GET['referenceWeek'];

**else**

$referenceWeek = date("Y-\WW", time());

*// Returns the timestamp of the week provided*

$referenceTime = date("U", strtotime($referenceWeek));

*// Update week boxes for either a teacher or a room -- this needs to be different for the hidden fields in each as they will either have a teacher ID or room ID*

**if** (isset($\_GET['teacherID']))

{

*// Fetches details from the database used for gathering the different elements needed for displaying the teacher's name and ID*

$teacherName = $bookingSystem->queryTable("Users", **array**("userID"=>$\_GET['teacherID']), 1, "userID", "ASC"); ?>

<!-- Outputs the teacher's name and ID -->

<h1><?php **echo** $teacherName['firstName']." ".$teacherName['lastName']." (".$teacherName['userID'].")"; ?></h1>

<!-- Outputs the part that deals with the week chooser -->

<form name="referenceTimeChooser" action="<?php **echo** $\_SERVER['PHP\_SELF']; ?>" method="get" autocomplete="off">

<input name="referenceWeek" type="week" value="<?php **echo** $referenceWeek; ?>" />

<!-- The hidden fields ensure that the $\_GET values are still passed through when a new week is chosen -->

<input type="hidden" name="view" value="<?php **echo** $\_GET['view']; ?>" />

<input type="hidden" name="teacherID" value="<?php **echo** $\_GET['teacherID']; ?>" />

<!-- The title attribute gives a hint to the user as to what the button does. -->

<input title="Update to the selected week." type="submit" value="Update">

</form>

<?php

}

**elseif** (isset($\_GET['roomID']))

{

?>

<!-- Outputs the room name -->

<h1><?php **echo** $bookingSystem->queryTable("Rooms", **array**("roomID"=>$\_GET['roomID']), 1, "roomID")['name']; ?></h1>

<!-- Outputs the part that deals with the week chooser -->

<form name="referenceTimeChooser" action="<?php **echo** $\_SERVER['PHP\_SELF']; ?>" method="get" autocomplete="off">

<input name="referenceWeek" type="week" value="<?php **echo** $referenceWeek; ?>" />

<!-- The hidden fields ensure that the $\_GET values are still passed through when a new week is chosen -->

<input type="hidden" name="view" value="<?php **echo** $\_GET['view']; ?>" />

<input type="hidden" name="roomID" value="<?php **echo** $\_GET['roomID']; ?>" />

<!-- The title attribute gives a hint to the user as to what the button does. -->

<input title="Update to the selected week." type="submit" value="Update">

</form>

<?php

}

*// Table that is created to hold all of the events*

**echo** "<table class='eventTable' id='eventTable'>";

**if** (isset($\_GET['teacherID']))

{

*// A query build to gather all events between the start and end of the week for a given teacher ID*

$eventResult = $bookingSystem->queryTable(**NULL**, "SELECT \* FROM Events WHERE `ownerID` = '".$\_GET['teacherID']."' AND `startTimestamp` BETWEEN ".strtotime("last Monday 00:00", $referenceTime)." AND ".strtotime("next Friday 23:59", $referenceTime)." ORDER BY startTimestamp, ID ASC", **NULL**, **NULL**, **NULL**, **false**, **true**);

**if** (!is\_array($eventResult[0])) *// If the array isn't multidimensional, make it a multidimensional array so that the foreach loop doesn't fail later*

$eventResult[0] = $eventResult;

}

**elseif** (isset($\_GET['roomID']))

{

*// A query build to gather all events between the start and end of the week for a given room ID*

$eventResult = $bookingSystem->queryTable(**NULL**, "SELECT \* FROM Events WHERE `roomID` = '".$\_GET['roomID']."' AND `startTimestamp` BETWEEN ".strtotime("last Monday 00:00", $referenceTime)." AND ".strtotime("next Friday 23:59", $referenceTime)." ORDER BY startTimestamp, ID ASC", **NULL**, **NULL**, **NULL**, **false**, **true**);

}

*// If it is a single array, then add as an element to another array*

**if** (isset($eventResult['ID']))

$eventResult = **array**($eventResult);

*// Displays column headings*

**echo** "<tr>";

**echo** "<th>Day</th>";

**echo** "<th>Period 1</th>";

**echo** "<th>Period 2</th>";

**echo** "<th>Period 3</th>";

**echo** "<th>Period 4</th>";

**echo** "<th>Period 5</th>";

**echo** "<th>Period 6</th>";

**echo** "<th>Period 7</th>";

**echo** "</tr>";

*// A counter for each day*

**for** ($dayCounter = 0; $dayCounter <= 4; $dayCounter++)

{

**echo** "<tr>";

**echo** "<td><strong>".jddayofweek($dayCounter, 1)."</strong> (".date("d/m/Y", strtotime(jddayofweek($dayCounter, 1), $referenceTime)).")</td>"; *// Displays the days of the week from Monday to Friday due to the for loop above*

*// A counter for each period of each day*

**for** ($periodCounter = 1; $periodCounter <= 7; $periodCounter++)

{

**echo** "<td>";

*// Finds the start time associated with the given period number*

$timeToCheck = $bookingSystem->queryTable("Periods", **array**("periodNumber" => $periodCounter), 1, "periodNumber")['startTime'];

*// Pads $timeToCheck with zeros at the front until it is 4 characters long*

**while** (strlen($timeToCheck) < 4)

{

$timeToCheck = "0".$timeToCheck;

}

*// These variables limit the number of a events that can appear in an event (even if they're double booked, the first wil be shown)*

$resultCount = 0;

$resultLimit = 1; *// This can be changed to allow for more events to be shown in each cell, though it is not recommended*

**foreach**($eventResult **as** $singleResult) *// A foreach loop to go through each event returned for the week*

{

**if** ($resultCount < $resultLimit) *// This limits the number of results that can be displayed per period*

{

*// The '1970-01-01' part of the convesion is so that the time converts into straight-forward seconds*

$fullSearchTime = $referenceTime + $dayCounter \* (60\*60\*24) + strtotime("1970-01-01 ".$timeToCheck);

**if**(array\_search(intval($fullSearchTime), $singleResult, **false**)) *// Sees if the current period number, once converted to a time, is found in the returned set of results*

{

*// Creates a link to edit the page if the name of the event is clicked on*

**echo** "<a href='./editEventSingle.php?serializedGetVariables=".serializeGetVariables()."&eventID=".$singleResult['ID']."'><strong>".$singleResult['name']."</strong></a>";

**echo** "<br />";

*// The part below adds information regarding the room or teacher, depending on what $\_GET variable has been set*

**if** (isset($\_GET['roomID']))

{

*// Gets the owner of the event's name and user ID*

$userResult = $bookingSystem->queryTable("Users", **array**("userID" => $singleResult['ownerID']), 1, "userID", "ASC", **false**);

**echo** "<i>".$userResult['firstName']." ".$userResult['lastName']." (".$singleResult['ownerID'].")"."</i>";

}

**elseif** (isset($\_GET['teacherID']))

{

*// Gets the room of the event's name*

$roomResult = $bookingSystem->queryTable("Rooms", **array**("roomID" => $singleResult['roomID']), 1, "roomID", "ASC", **false**);

**echo** "<i>".$roomResult['name']."</i>";

}

*// Increments the result count and doesn't let the number of results in a cell exceed $resultLimit, and is also used for the "Book now" button*

$resultCount++;

}

}

}

*// If there were no results found for the given day and period, a book now button is shown*

**if**($resultCount == 0)

{

*// Lets administrators always book and event*

**if**(getUserLevel($bookingSystem, $\_COOKIE['userID']) >= 3)

{

*// Fills in the correct $\_GET variables for the edit page*

**echo** "<div id='bookNow'><a href='addEventSingle.php?time=".$fullSearchTime."&serializedGetVariables=".serializeGetVariables()."'>Book now</a></div>";

}

*// Makes sure that teachers can only book when the view is as a room, or if the userID is their own*

**elseif** ((getUserLevel($bookingSystem, $\_COOKIE['userID']) >= 2) **AND** (($\_GET['view'] == "room") **OR** ($\_GET['teacherID'] == $\_COOKIE['userID'])))

{

*// Fills in the correct $\_GET variables for the edit page*

**echo** "<div id='bookNow'><a href='addEventSingle.php?time=".$fullSearchTime."&serializedGetVariables=".serializeGetVariables()."'>Book now</a></div>";

}

*// Notice that the button isn't ever shown to students*

}

**echo** "</td>";

}

**echo** "</tr>";

}

**echo** "</table>";

}

*// If the teacher ID XOR the room ID isn't set, then if the view is set to teacher:*

**elseif**($\_GET['view'] == "teacher")

{

**echo** "<h1>Choose a Teacher</h1>";

*// Returns a list of all teachers*

$returnedTeacherList = $bookingSystem->queryTable("Users", **array**("userLevel" => 2), **NULL**, " userID, lastName, firstName", "ASC");

**echo** "<form id='teacherChooser'>";

**echo** "<select name='teacherID'>";

*// Creates a multidimensional array for the foreach loop*

**if** (isset($returnedTeacherList['userID']))

$buildTeacherList[0] = $returnedTeacherList;

**else**

$buildTeacherList = $returnedTeacherList;

*// Builds a list of teachers as options in a select box on the form*

**foreach** ($buildTeacherList **as** $teacher)

{

**echo** "<option value='".$teacher['userID']."'>[".$teacher['userID']."] ".$teacher['firstName']." ".$teacher['lastName']."</option>";

}

**echo** "</select>";

**echo** "<input type='hidden' name='view' value='teacher' />"; *// Need to set the view to room for the next stage on the page*

**echo** "<input type='submit' value='View Events' />";

**echo** "</form>";

}

*// If the teacher ID XOR the room ID isn't set, then if the view is set to room:*

**elseif**($\_GET['view'] == "room")

{

**echo** "<h1>Choose a Room</h1>";

**echo** "<p>Note: The number in parentheses is the capacity of the room. </p>";

*// Returns a list of all rooms*

$returnedRoomList = $bookingSystem->queryTable("Rooms", "\*", **NULL**, "name", "ASC");

**echo** "<form id='roomChooser'>";

**echo** "<select name='roomID'>";

*// Creates a multidimensional array for the foreach loop*

**if** (isset($returnedRoomList['roomID']))

$buildRoomList[0] = $returnedRoomList;

**else**

$buildRoomList = $returnedRoomList;

*// Builds a list of rooms as options in a select box on the form*

**foreach** ($buildRoomList **as** $room)

{

**if** ($room['capacity'] == "")

$room['capacity'] = "Unknown capacity"; *// If there is no capacity specified, unknown is listed next to the room.*

**echo** "<option value='".$room['roomID']."'>".$room['name']." (".$room['capacity'].")</option>";

}

**echo** "</select>";

**echo** "<input type='hidden' name='view' value='room' />"; *// Need to set the view to room for the next stage on the page*

**echo** "<input type='submit' value='View Events' />";

**echo** "</form>";

}

}

*// If the view, teacher ID and room ID isn't set, then let the user select to view rooms by either teacher or room:*

**else**

{

**echo** "<h1>View All Events</h1>";

**echo** "<h3><a href='".$\_SERVER['PHP\_SELF']."?view=teacher'>By Teacher</a></h3>";

**echo** "<h3><a href='".$\_SERVER['PHP\_SELF']."?view=room'>By Room</a></h3>";

}

}

**else**

{

**echo** "<p>".returnUserLevelError($bookingSystem, $userLevelNeeded)."</p>";

}

?>

</body>

</html>

### viewAllHolidays.php

|  |
| --- |
|  |

<!DOCTYPE html>

<html>

<head>

<!-- **This** includes the stylesheet **for** the system -->

<link rel="stylesheet" href="./includes/styles.css" type="text/css" />

<!-- Includes the header, which includes other useful files, checks user login details **and** generates an appropriate title -->

<?php **include** "./includes/header.php"; ?>

</head>

<body>

<?php

*// Includes the menu for navigating the site, which will be generated differently for each user*

**include** "./includes/menu.php";

*// The user needs to be an administrator to view this page*

$userLevelNeeded = 3;

**if**(getUserLevel($bookingSystem, $\_COOKIE['userID']) >= $userLevelNeeded)

{

*// Notice that this deletion block of code is within the check for the user level -- maliciously posted data cannot cause a mis-delete*

*// If the form has been submitted and the value is "Delete selected"*

**if** (isset($\_POST['holidayCheckboxSubmit']) **AND** ($\_POST['holidayCheckboxSubmit'] == "Delete selected"))

{

*// Deletes all from the "Holidays" table where the ID is the one selected in the checkbox array*

**foreach** ($\_POST['checkbox'] **as** $singleCheckbox)

{

$bookingSystem->deleteFromTable("Holidays", **array**("ID"=>$singleCheckbox));

}

}

**echo** "<h1>Admin Panel</h1>";

**echo** "<h2>View All Holidays</h2>";

*// Finds all holidays and sorts them by their start timestamp and then their end timestamp*

$holidaysResponse = $bookingSystem->queryTable("Holidays", "\*", **NULL**, "startTimestamp, endTimestamp", "ASC");

*// If there are holidays in the table:*

**if** (!**empty**($holidaysResponse))

{

**if** (isset($holidaysResponse['label'])) *// If the label element exists, then there must only be one record found but we add this to a multidimensional array to ensure the foreach loop works later*

$holidaysResponse = **array**($holidaysResponse);

**echo** "<form id='holidayCheckbox' method='POST'>";

**echo** "<table class='holidaysTable' id='holidaysTable'>";

**echo** "<tr>";

**echo**"<th>Select</th>";

**echo** "<th>Label</th>";

**echo** "<th>Holiday Start</th>";

**echo** "<th>Holiday End</th>";

**echo** "<th>Edit</th>";

**echo** "</tr>";

*// Loops through each found holiday in the table*

**foreach**($holidaysResponse **as** $holiday)

{

**echo** "<tr>";

**echo** "<td><input value='".$holiday['ID']."' type='checkbox' id='checkbox' name='checkbox[]'></td>";

**echo** "<td>".$holiday['label']."</td>";

**echo** "<td>".date('d/m/Y', $holiday['startTimestamp'])."</td>";

**echo** "<td>".date('d/m/Y', $holiday['endTimestamp'])."</td>";

**echo** "<td><a href='editHolidaysSingle.php?holidayID=".$holiday['ID']."'>Edit</a></td>";

**echo** "</tr>";

}

**echo** "</table>";

*// Shows a button that can be used to delete the selected holidays*

**echo** "<h3>Operations</h3>";

**echo** "<input type='submit' name='holidayCheckboxSubmit' form='holidayCheckbox' value='Delete selected' />";

**echo** "</form>";

}

*// else there are no holidays*

**else**

{

**echo** "No holidays created. Create one <a href='./addHolidaySingle.php'>here</a>.";

}

}

**else**

{

**echo** "<p>".returnUserLevelError($bookingSystem, $userLevelNeeded)."</p>";

}

?>

</body>

</html>

### viewAllRooms.php

|  |
| --- |
|  |

<!DOCTYPE html>

<html>

<head>

<!-- **This** includes the stylesheet **for** the system -->

<link rel="stylesheet" href="./includes/styles.css" type="text/css" />

<!-- Includes the header, which includes other useful files, checks user login details **and** generates an appropriate title -->

<?php **include** "./includes/header.php"; ?>

</head>

<body>

<?php

*// Includes the menu for navigating the site, which will be generated differently for each user*

**include** "./includes/menu.php";

*// The user needs to be an administrator to view the content of the page*

$userLevelNeeded = 3;

**if**(getUserLevel($bookingSystem, $\_COOKIE['userID']) >= $userLevelNeeded)

{

*// Notice that this deletion block of code is within the check for the user level -- maliciously posted data cannot cause a mis-delete*

*// If the form has been submitted and the value is "Delete selected"*

**if** (isset($\_POST['roomCheckboxSubmit']) **AND** ($\_POST['roomCheckboxSubmit'] == "Delete selected"))

{

*// Deletes all from the "Rooms" table where the ID is the one selected in the checkbox array*

**foreach** ($\_POST['checkbox'] **as** $singleCheckbox)

{

$bookingSystem->deleteFromTable("Rooms", **array**("roomID"=>$singleCheckbox));

}

}

**echo** "<h1>Admin Panel</h1>";

**echo** "<h2>View All Rooms</h2>";

*// Returns all rooms*

$roomResponse = $bookingSystem->queryTable("Rooms", "\*", **NULL**, "roomID");

*// If some rooms exist in the table:*

**if** (!**empty**($roomResponse))

{

**echo** "<form id='roomCheckbox' method='POST'>";

**echo** "<table class='roomTable' id='allRoomTable'>";

**echo** "<tr>";

**echo** "<th>Select</th>";

**echo** "<th>Room ID</th>";

**echo** "<th>Name</th>";

**echo** "<th>Capacity</th>";

**echo** "<th>Edit</th>";

**echo** "</tr>";

*// If the 0th element does not exist, then it is not a multidimensional array*

**if** (!is\_array($roomResponse[0]))

{

*// Makes the array multidimensional*

$loopArray[0] = $roomResponse;

}

**else**

{

*// Just straight assigns the value of the array if it is already multidimensional*

$loopArray = $roomResponse;

}

*// Loops through each room, displaying its details*

**foreach**($loopArray **as** $room)

{

**echo** "<tr>";

**echo** "<td><input value='".$room['roomID']."' type='checkbox' id='checkbox' name='checkbox[]'></td>";

**echo** "<td>".$room['roomID']."</td>";

**echo** "<td>".$room['name']."</td>";

**echo** "<td>".$room['capacity']."</td>";

**echo** "<td><a class='viewAllRoomsEdit' href='editRoomSingle.php?roomID=".$room['roomID']."'>Edit</a></td>";

**echo** "</tr>";

}

**echo** "</table>";

*// A submit button used for deleting records that are selected*

**echo** "<h3>Operations</h3>";

**echo** "<input type='submit' name='roomCheckboxSubmit' form='roomCheckbox' value='Delete selected' />";

**echo** "</form>";

}

**else**

{

*// No rooms have been found in the database table:*

**echo** "No rooms created.";

}

}

**else**

{

**echo** "<p>".returnUserLevelError($bookingSystem, $userLevelNeeded)."</p>";

}

?>

</body>

</html>

### viewAllUsers.php

|  |
| --- |
|  |

<!DOCTYPE html>

<html>

<head>

<!-- **This** includes the stylesheet **for** the system -->

<link rel="stylesheet" href="./includes/styles.css" type="text/css" />

<!-- Includes the header, which includes other useful files, checks user login details **and** generates an appropriate title -->

<?php **include** "./includes/header.php"; ?>

</head>

<body>

<?php

*// Includes the menu for navigating the site, which will be generated differently for each user*

**include** "./includes/menu.php";

*// The user needs to be an administrator to view the content of the page*

$userLevelNeeded = 3;

**if**(getUserLevel($bookingSystem, $\_COOKIE['userID']) >= $userLevelNeeded)

{

*// If the submit has been sent and it is set to "Delete selected" then:*

**if** (isset($\_POST['userCheckboxSubmit']) **AND** ($\_POST['userCheckboxSubmit'] == "Delete selected"))

{

*// Loops through each ID selected and deletes them*

**foreach** ($\_POST['checkbox'] **as** $singleCheckbox)

{

$bookingSystem->deleteFromTable("Users", **array**("userID"=>$singleCheckbox));

}

}

**echo** "<h1>Admin Panel</h1>";

**echo** "<h2>View All Users</h2>";

*// Returns all users in the Users table*

$usersResponse = $bookingSystem->queryTable("Users", "\*", **NULL**, "userID");

**if** (!**empty**($usersResponse))

{

**echo** "<form id='userCheckbox' method='POST'>";

**echo** "<table class='usersTable' id='allUsersTable'>";

**echo** "<tr>";

**echo** "<th>Select</th>";

**echo** "<th>User ID</th>";

**echo** "<th>User Level</th>";

**echo** "<th>Email Address</th>";

**echo** "<th>First Name</th>";

**echo** "<th>Last Name</th>";

**echo** "<th>Edit</th>";

**echo** "</tr>";

*// If the 0th element does not exist, then we presume that it is not multidimensional*

**if** (!is\_array($usersResponse[0]))

{

*// Make the array multidimensional (so the foreach loop does not fail)*

$loopArray[0] = $usersResponse;

}

**else**

{

*// Assign the $loopArray to the value of the response as it is already multidimensional*

$loopArray = $usersResponse;

}

*// Loops through each user*

**foreach**($loopArray **as** $user)

{

*// Finds the user's user level*

$userLevel = $bookingSystem->queryTable("UserLevels", **array**("userLevel"=>$user['userLevel']), 1, "userLevel");

*// Outputs a new row to the table with all of the data from the table associated with that user*

**echo** "<tr>";

**echo** "<td><input value='".$user['userID']."' type='checkbox' id='checkbox' name='checkbox[]'></td>";

**echo** "<td title='".$user['firstName']." ".$user['lastName']."&apos;s unique school ID.'>".$user['userID']."</td>";

**echo** "<td title='".$userLevel['description']."'>".$userLevel['title']."</td>";

**echo** "<td title='".$user['firstName']." ".$user['lastName']."&apos;s email address.'><a href='mailto:".$user['emailAddress']."'>".$user['emailAddress']."</a></td>";

**echo** "<td title='The user&apos;s first name.'>".$user['firstName']."</td>";

**echo** "<td title='The user&apos;s last name.'>".$user['lastName']."</td>";

**echo** "<td title='A button to edit ".$user['firstName']." ".$user['lastName']."&apos;s details.'><a class='viewAllUsersEdit' href='editUserSingle.php?userID=".$user['userID']."'>Edit</a></td>";

**echo** "</tr>";

}

**echo** "</table>";

*// A delete selected button*

**echo** "<h3>Operations</h3>";

**echo** "<input type='submit' name='userCheckboxSubmit' form='userCheckbox' value='Delete selected' />";

**echo** "</form>";

}

*// If the array is empty, then no users are in the table*

**else**

{

**echo** "No users created.";

}

}

**else**

{

**echo** "<p>".returnUserLevelError($bookingSystem, $userLevelNeeded)."</p>";

}

?>

</body>

</html>

## Conversion Plan

The plan for the transition between the existing and proposed system would be to have parallel systems running for a short time, to make sure that all data has been carried across.

A major problem that may occur during the conversion is that staff may need new training on how to use the system. This would easily be overcome by reading the user manual for this system (section [H](#_User_Manual), page b); it may also be useful for the users to read the existing documentation too.

## Comments from the Client

**Mr Jacobs**: “*I’ve been in contact with Adam throughout the technical solution section; he has proposed several ideas that I have accepted and rejected. I am happy that a working solution is starting to evolve.*

*Adam has also asked me to do some basic alpha testing during development to make sure that functions work as I want them to.*”



System Testing

D



# System Testing

The tables are replicated using the same test ID from the table in the design section (section [B.23](#_Testing_Plan), page 76); this table in the design section explains the purpose of the test and this section concentrates on the actual outcomes of the testing.

## Alpha: Included Files Testing

### Alpha: Included Files Testing Plan

|  |  |  |  |
| --- | --- | --- | --- |
| **Test ID** | **Expected Outcome** | **Actual Outcome** | **Comments and Corrective Actions** |
| class.openDB.inc.php | **-** | **-** | **-** |
| class.openDB.inc.php.\_\_CONTRUCT | A database to be made with the specified name, and the properties of the class to be set. | The database name is created successfully. | No changes needed. |
| No database to be created and no connection. | There was no database created and the system can’t communicate with the MYSQL server. | No changes needed. |
| N/A | N/A | N/A |
| class.openDB.inc.php.createDB() | A database to be made with the specified name. | A database was made with whatever database name was chosen. | No changed needed. |
| A database not to be made with an error message. | There was no database created and the system can’t communicate with the MYSQL server. | No changes needed. |
| N/A | N/A | N/A |
| class.openDB.inc.php.connect() | Queries are able to be ran using the connection. | A connection is established. | No changes needed. |
| N/A | N/A | N/A |
| N/A | N/A | N/A |
| class.openDB.inc.php.disconnect() | Connection to be closed and no more queries are able to be requested from the MYSQL server. | The connection is broken and no queries can be executed. | No changes needed. |
| N/A | N/A | N/A |
| N/A | N/A | N/A |
| class.openDB.inc.php.deleteFromTable() | That the correct record is deleted from the correct database table. | The correct records are deleted from the correct table. | No changes needed. |
| The query isn’t carried out. | The query isn’t carried out. | No changes needed. |
| N/A | N/A | N/A |
| class.openDB.inc.php.createTable() | That a table with the correct details is created. | The tables are created without a problem. | No changes needed. |
| N/A | N/A | N/A |
| That a table with only an ID field is created. | Tables with only an ID field are created. | No changes needed. |
| class.openDB.inc.php.queryTable() | The correct query is executed. | The correct query is executed, even if the query is selected as a direct by the direct parameter. | No changes needed. |
| The query not to be executed. | No query was executed. | No changes needed. |
| Return an empty results set. | An empty result set was returned. | No changes needed. |
| class.openDB.inc.php.insertTable() | The record with all of the correct details to be inserted to the correct database table. | Records were inserted correctly with all of the columns. | No changes needed. |
| No data is inserted into any table. | No data was inserted into the table. | No changes needed. |
| N/A | N/A | N/A |
| class.openDB.inc.php.updateTable() | The correct record is updated with the new data. | The record’s columns were updated correctly, according to some given reference data (a column that remained unchanged in its value). | No changes needed. |
| The table isn’t updated. | The row in the table wasn’t updated. | No changes needed. |
| The correct record is updated with the new single items of data. | The array containing only 1 element of data was inserted without a problem. | No changes needed. |
| createTables.php | **-** | **-** | **-** |
| createTables.php.1 | The tables are created successfully. | All the tables were added without an issue, including the correct data types and fields that remain constant. | No changes needed. |
| N/A | N/A | N/A |
| N/A | N/A | N/A |
| functions.php | **-** | **-** | **-** |
| functions.php.getUserLevel() | The user level for the user ID in the cookie is returned. | The correct user level is returned for a variety of different users with different user levels. | No changes needed. |
| N/A | N/A | N/A |
| N/A | N/A | N/A |
| functions.php.formatTime() | A 3 or 4 digit time is formatted with a colon between the hour and the minute. | The time is formatted correctly with a colon. | No changes needed. |
| N/A | N/A | N/A |
| N/A | N/A | N/A |
| functions.php.serializeGetVariables() | The $\_GET variables are all serialised into a single string. | All of the $\_GET variables are serialised without an issue. | No changes needed. |
| An error occurs or the $\_GET variables just aren’t used in the next stage. | There is no $\_GET string provided. | No changes needed. |
| The $\_GET variables are all serialised into a single string. | The $\_GET variables are serialised correctly. | No changes needed. |
| functions.php.destroyCookies() | The cookies are destroyed to the point where they no longer contain any salvageable data. | Cookies are set with an expiry date in the past with blank content (hence leaving no salvageable data). | No changes needed. |
| N/A | N/A | N/A |
| A cookie is created with an expiry date in the past. | No problems whatsoever; the cookie is created with no content with an expiry date in the past. | No changes needed. |
| functions.php.returnUserLevelError() | Met with an appropriate message for not being able to view a page. | The message: “You need to be an 'Administrator' to view this content.” was shown when a student or a teacher tried to view adminPanel.php. Tests on other pages produced a similar result. | No changes needed. |
| The function fails to return the error message. | The function returns false and no text output is returned. | No changes needed. |
| A user should not be met with any sort of message and should instead be able to view the content of the page. | The message wasn’t shown for any page where a user should have had access. | No changes needed. |
| functions.php.saltAndHashPassword() | A hash value is returned according to the SHA-256 hash algorithm. | Valid hashes were returned. These were confirmed by being checked against online SHA256 converters with the conjugated string. | No changes needed. |
| The function fails to return the hash. | The function returns false and no hash value is returned. |  |
| A hash value is returned according to the SHA-256 hash algorithm, but will, of course, only be a hash value of the user ID. | The hash is returned successfully (but is actually only the hash of the userID [and this is allowed in this system]). | No changes needed. |
| functions.php.isCurrentPage() | Either a true or false return value depending on whether or not the current page is the page sought. | Returns true or false correctly when used on different pages. | No changes needed. |
| N/A | N/A | N/A |
| N/A | N/A | N/A |
| functions.php.generateTitle() | The correct title is shown on the tab or window of the page. | All of the pages return the correct title as given in ./includes/titleForFilenames.php. | No changes needed. |
| One or the other title is shown — not both. | The first title is shown. | No changes needed. |
| A standardised title is used. | The base title is shown: “KPS | RBS”, although this should only ever happen when new pages are added and the array in ./includes/titleForFilenames.php isn’t updated. | No changes needed. |
| header.php | - | - | - |
| header.php.1 | Cookies are not destroyed. | The cookies aren’t destroyed. | No changes needed. |
| N/A | N/A | N/A |
| Cookies are destroyed. | The cookies are destroyed using the destroyCookies() function. | No changes needed. |
| loginInformation.php | - | - | - |
| loginInformation.php.1 | The correct user ID and user level is shown for the user ID specified in the cookie. | The login information is only shown when a user is logged in. | No changes needed. |
| N/A | N/A | N/A |
| N/A | N/A | N/A |
| menu.php | - | - | - |
| menu.php.1 | An appropriate menu is shown for the logged in user. | The correct set of menu buttons are shown for users with each type of user level. | No changes needed. |
| N/A | N/A | N/A |
| N/A | N/A | N/A |
| SQLDetails.php | - | - | - |
| SQLDetails.php.1 | The class is instantiated. | The class instantiates correctly. | No changes needed. |
| N/A | N/A | N/A |
| N/A | N/A | N/A |
| styles.css | - | - | - |
| styles.css.1 | Appropriate styling is used consistently throughout the system. | A consistent style is kept throughout the entire system. | No changes needed. |
| N/A | N/A | N/A |
| N/A | N/A | N/A |
| titleForFilenames.php | - | - | - |
| titleForFilenames.php.1 | All pages do have a title. | Some pages didn’t have a title. | I added all of the titles to the array with the corresponding page name. |
| Either one of the titles are shown; not both. | Just the first title in the array was chosen. | Ensured that the array only contained one title for each page name. |
| N/A | N/A | N/A |

### Outcomes from Alpha: Included Files Testing

#### class.openDB.inc.php.createDB()

|  |
| --- |
|  |
| *Demonstration that a database with any name can be created.* |

#### createTables.php.1

As you can see in the test run from class.openDB.inc.php.createDB(), tables are created in the database successfully.

#### loginInformation.php.1

Please see menu.php.1 for login information for different types of users. Below is a screenshot of the login information when the user isn’t logged in.

|  |
| --- |
|  |
| *Showing that the login information doesn’t appear when the user isn’t logged in.* |

#### menu.php.1

Here’s a demonstration to show that different menus are shown for each type of user logged in.

|  |
| --- |
|  |
| *The menu shown to students.* |
|  |
| The menu shown to staff (or teachers) that are logged in. |
|  |
| *The menu shown to administrators that are logged in.* |

#### titleForFilenames.php.1

See the titleForFilenames.php section (section [C.1.9](#_titleForFilenames.php), page 127) in the technical solution section for the amended code that contains all of the viewable pages on the site with their corresponding titles.

|  |
| --- |
|  |
| *In the top left it can be seen that the correct title is shown on this example page, viewAllEvents.php.* |

## Alpha: Viewable Pages Testing

### Alpha: Viewable Pages Testing Plan

|  |  |  |  |
| --- | --- | --- | --- |
| **Test ID** | **Expected Outcome** | **Actual Outcome** | **Comments and Corrective Actions** |
| addEvent.php | - | - | - |
| addEvent.php.1 | The user is taken to the correct page. | The user is taken to the correct page. | No changes needed. |
| N/A | N/A | N/A |
| N/A | N/A | N/A |
| addEventImport.php | - | - | - |
| addEventImport.php.1 | The events in the CSV file are imported correctly. | The event imports successfully. | No changes needed. |
| An error message stating that the end date is before the start date is shown. | The event imports successfully for one week. | I needed to make a validation check to see if the end date was before the start date. |
| N/A | N/A | N/A |
| addEventImport.php.2 | Events are imported for the dates between the two dates, even if there are holidays between those dates. | Events inserted even when there were holidays set. | I had to alter the algorithm for the import to make sure that the events didn’t insert when there were holidays. See the outcomes below in the next section (section [D.2.2](#_Outcomes_from_Alpha:), page 255). |
| An error message stating that the end date is before the start date is shown. | With the changes made in the previous test, a relevant error message was shown. | Changes made as above. |
| The events are inserted for that single day. | The events are inserted for a week after the insert date. | I NEED TO ADD TO THIS |
| addEventSingle.php | - | - | - |
| addEventSingle.php.1 | The single event is inserted. | The single event is inserted without an issue. | No changes needed. |
| An error message telling the user which fields haven’t been completed. | Appropriate error messages for shown for each blank field. | No changes needed. |
| N/A | N/A | N/A |
| addEventSingle.php.2 | The event is inserted without an issue. | The event didn’t inserted correctly when a teacher tried to add an event. | I had to add a separate hidden field for when teachers are restricted to only adding events for themselves. |
| An error message telling the user which fields haven’t been completed. | The correct error messages were given. | No changes needed. |
| N/A | N/A | N/A |
| addHolidaySingle.php | - | - | - |
| addHolidaySingle.php.1 | The holiday is inserted without an issue. | The holiday was inserted. | No changes needed. |
| An error message stating that the end date is before the start date is shown. | An error message was shown. | No changes needed. |
| A single day holiday is added. | A single dated holiday was added. | No changes needed. |
| addRoomImport.php | - | - | - |
| addRoomImport.php.1 | The rooms contained in the CSV file are inserted into the database table. | The correct rooms were added to the database table. | No changes needed. |
| The form will ask for a CSV file to upload. | An appropriate error message was shown. | No changes needed. |
| The single room contained in the CSV file is inserted into the database table. | A single room inserts without an issue. | No changes needed. |
| addRoomSingle.php | - | - | - |
| addRoomSingle.php.1 | The room is inserted into the database table. | The room inserts into the database correctly. | No changes needed. |
| The text isn’t allowed to be entered. | Non-numeric data was allowed to be entered. | I changed the field type parameter from type=”text” to type=”number”. |
| The room will be inserted with a capacity of 0. | The capacity inserted successfully. | No changes needed. |
| addUserImport.php | - | - | - |
| addUserImport.php.1 | The users contained in the CSV file are uploaded. | The users are inserted without an issue. | No changes needed. |
| The form will ask for a CSV file to upload. | An error message appears | No changes needed. |
| The single user contained in the CSV file is inserted into the database table. | The user is inserted without an issue. | No changes needed. |
| addUserSingle.php | - | - | - |
| addUserSingle.php.1 | The user is inserted into the database table with a hash on the user ID and the password provided in both password fields. | The hash is generated correctly and is inserted with the remainder of the recorder’s details. | No changes needed. |
| An error message stating that the passwords are different is shown. | An error message is shown. | No changes needed. |
| The user is inserted into the database with the password specified, as long as the password has a maximum length of 32 characters. | Passwords of 1 or 32 characters are hashed without a problem. | No changes needed. |
| adminPanel.php | - | - | - |
| adminPanel.php.1 | The user is taken to the page described by the hyperlink’s text. | The user is taken to the correct page. | No changes needed. |
| N/A | N/A | N/A |
| N/A | N/A | N/A |
| editEventSingle.php | - | - | - |
| editEventSingle.php.1 | The correct fields are already completed. | The fields are already completed, ready for editing. | No changes needed. |
| No fields are completed (unless the wrong event ID actually exists as an event ID). | The wrong data is only shown if the wrong ID actually exists. | No changes needed. |
| N/A | N/A | N/A |
| editRoomSingle.php | - | - | - |
| editRoomSingle.php.1 | The correct details are filled in for the user. | The correct details are filled in. | No changes needed. |
| No fields are completed (unless the wrong room ID actually exists as a room ID). | No fields were completed unless the wrong room ID actually existed in the table. | No changes needed. |
| N/A | N/A | N/A |
| editUserSingle.php | - | - | - |
| editUserSingle.php.1 | The correct fields are already completed. | The correct fields are completed. | No changes needed. |
| No fields are completed (unless the wrong user ID actually exists as a user ID). | No fields were completed unless the wrong user ID actually existed in the table. | No changes needed. |
| N/A | N/A | N/A |
| index.php | - | - | - |
| index.php.1 | The correct user level is shown and greeted to the user. | The correct user level is shown for the logged in user. | No changes needed. |
| N/A | N/A | N/A |
| N/A | N/A | N/A |
| login.php | - | - | - |
| login.php.1 | A cookie is created with the user ID and user password and the user is redirected to index.php. | A cookie is created successfully and the user is redirected. | No changes needed. |
| The user is shown various error messages, depending on which piece of data is invalid. | The user is told whether it is the user ID or the password that is at fault. | No changes needed. |
| N/A | N/A | N/A |
| logout.php | - | - | - |
| logout.php.1 | The user’s cookies are destroyed. | The bookies are destroyed when the page is run. | No changes needed. |
| The user is shown a standardised error message, similar to a one used on other pages where a user isn’t allowed to view a page. | The page runs and attempts to destroy the cookies (which is fine). | No changes needed. |
| N/A | N/A | N/A |
| viewAllEvents.php | - | - | - |
| viewAllEvents.php.1 | The buttons allow the user to select teachers or rooms. | The correct data was shown. | No changes needed. |
| N/A | N/A | N/A |
| The buttons still allow the user to select teachers or rooms. | For the case where there was only 1 teacher, a strange thing happened as you can see in the following section (section [D.2.2.11](#_viewAllEvents.php.1), page 264). | I needed to create a multidimensional array from the result of the query if there was only 1 record returned. |
| viewAllEvents.php.2 | The events for the given room ID or teacher ID is shown. | The correct events were shown. |  |
| No events are shown. | No events are shown. | It would be better to show an error message that there were no records found for what has been requested. This is documented in the following section (section [D.2.2](#_Outcomes_from_Alpha:), page 255). |
| N/A | N/A | N/A |
| viewAllHolidays.php | - | - | - |
| viewAllHolidays.php.1 | All holidays are shown. | All holidays are shown. | No changes needed. |
| An appropriate error message is shown. | The user is given an error. | No changes needed. |
| N/A | N/A | N/A |
| viewAllRooms.php | - | - | - |
| viewAllRooms.php.1 | Only the selected rooms are deleted. | Only the selected rooms are deleted. | No changes needed. |
| No rooms are deleted. | No rooms are deleted. | No changes needed. |
| Only the selected rooms are deleted. | Only the selected rooms are deleted. | No changes needed. |
| viewAllUsers.php | - | - | - |
| viewAllUsers.php.1 | The user is directed to the edit screen with the correct parameters. | The user is taken to the correct edit screen. | No changes needed. |
| N/A | N/A | N/A |
| N/A | N/A | N/A |

### Outcomes from Alpha: Viewable Pages Testing

#### addEvent.php

|  |
| --- |
|  |
| *Hovering over the addEventImport.php page with the correct URL being shown in the corner of Google Chrome.* |

#### addEventImport.php.1

|  |
| --- |
| if (the event is set to repeat)  if (insert date) > (repeat to date)  error message ← “The repeat-to date is earlier that the insert date” |
| *The piece of pseudocode that deals with the validation.* |

addEventImport.php.2

|  |  |
| --- | --- |
|  |  |
| *The reported error for a teacher attempting to add an event for themselves.* | *The insertion now working.* |

|  |
| --- |
|  |
| *The error message that is now shown for an insert of 17th of March 2016 to 16th of March 2016, which is a boundary error.* |

#### addEventSingle.php

|  |  |
| --- | --- |
|  |  |
| *Adding the individual event.* | *The event that has now been added.* |

|  |
| --- |
|  |
| *List of error messages for fields that were left blank.* |

#### addHolidaySingle.php

|  |  |
| --- | --- |
|  |  |
| *Showing that a holiday for with the same start and end date successfully inserts the holiday, despite it being boundary data.* | *Screen showing that “Single Day Holiday” has inserted successfully.* |

|  |
| --- |
|  |
| *An error message is shown if the start date is after the end date.* |

#### addRoomImport.php

|  |
| --- |
|  |
| *The error message shown for no CSV file being chosen for upload.* |

#### addRoomSingle.php.1

|  |  |
| --- | --- |
|  |  |
| *Text being allowed in the capacity field.* | *Now only numbers are allowed in the capacity field.* |

|  |  |
| --- | --- |
|  |  |
| *Test to show that a room with a capacity of 0 can be inserted successfully, despite is being a boundary value.* | *Shows that the room is added to the database.* |

#### addUserImport.php.1

|  |  |
| --- | --- |
|  |  |
| *A successful message that is shown after the users have imported successfully.* | *A mixed list of messages for when a user of the ID specified in the CSV file already exists.* |

#### addUserSingle.php.1

|  |  |
| --- | --- |
|  |  |
| *As it can quite clearly be seen by the lengths of the passwords, the passwords do not match.* | *Showing that the form can be submitted when the passwords match. Here, the password was “pass”.* |

#### editEventSingle.php.1

|  |  |
| --- | --- |
|  |  |
| *When the event highlighted is clicked on, the screenshot at the right occurs.* | Screenshot to demonstrate that the fields are filled in when an event is clicked on from the viewAllEvents.php page. |

#### login.php.1

|  |  |
| --- | --- |
|  |  |
| The user is told that the user with that user ID does not exist. | The user is told that the password is the data item at fault (so the user ID exists). |

#### viewAllEvents.php.1

|  |
| --- |
|  |
| *The unusual result from there being only one teacher created.* |

|  |
| --- |
|  |
| *After the changes had been made.* |

|  |
| --- |
|  |
| *Showing that long lists of rooms, for example, also work [please ignore that the capacities are 0 here].* |

#### viewAllEvents.php.2

|  |  |
| --- | --- |
|  |  |
| *No events were shown, but also no error message was received.* | *An error message is now shown when a requested teacher ID or room ID doesn’t exist in the table.* |

#### viewAllHolidays.php.1

|  |
| --- |
|  |
| *The error message shown to a non-admin user trying to access the page.* |

#### viewAllRooms.php.1

|  |  |
| --- | --- |
|  |  |
| *A selection of rooms to be deleted.* | *Only the selected rooms have been deleted.* |

|  |  |
| --- | --- |
|  |  |
| *A single room selected for deletion.* | *The single room that has now been deleted.* |

|  |  |
| --- | --- |
|  |  |
| *All rooms are selected for deletion.* | *All rooms are deleted.* |

#### viewAllUsers.php.1

|  |  |
| --- | --- |
|  |  |
| *Clicking on the button to edit a particular user’s details.* | *Take to the correct edit screen.* |

## Comments from the Client

**Mr Jacobs**: “*I am very impressed with the extensiveness of the testing; I am confident that the system will be very robust.*

*Adam can improve by making every page consistent; on some of the pages, titles are not in the same format as the ones on the previous page. I can imagine this becoming annoying or distracting for some of the users.*”



Systems Maintenance

E



# System Maintenance

On the whole, the system is modularised into separate php documents. Each page usually has at least one major function that adds to the total functionality of the system. There are also pages that are ‘included’ on other pages; these are things like the class instance for connecting to the database and functions that are used and called globally throughout the system.

The entire functionality of the system is explained in detail in the testing

(section [D](#_System_Testing), page 236).

## System Overview

The room booking system I have developed for Kings Priory School will be used to book rooms for teachers throughout the school. It will also allow for administrators to manage users, by creating logins and passwords, and will allow for students to view events for a given teacher or room.

## Difficult Aspects of the System

*Some aspects of the system can be quite hard to understand. These aspects have been explained in detail below to ensure that readers fully understand how those aspects function.*

Before the system can actually be used, the site has to first be setup with an initial administrative user (details of how to do this can be found in the user manual section (section [H.1.2](#_Setting_up_the), page e)). From there, the administrative user can then add more users of one of the following types: Guest, Student, Teacher and Administrator; each of these types of users can perform different tasks (and more of this is explained through the design, technical solution, system testing, and user manual sections) and these different user levels are critical to the functionality of the system.

There are also four main data items that help the system function — these are: Events, users, rooms and holidays. Each of these have different add screens, and each of them have an addition by import or singly (with the exception of holidays, where events may only be added singly).

If an event that has been added by import contains any new teacher IDs, records corresponding to the newly added teacher IDs will be created in the users table. By default, the password for these users will be ‘password’; this should then be changed as soon as possible by an administrator.

Similarly, if any rooms are contained in the event import don’t already exist, they will be created with a corresponding room ID; however, the capacity will be set to 0 (since there is no data available to set them to anything else) so these should be updated when the administrator has the opportunity.

The ability to add holidays is also a big feature included in this system; this allows for an administrator to create repeated bookings without any bookings being created over a set period of dates (which are referred to as holidays in this system). Details of how to create repeated bookings that don’t occur on holidays, see the user manual section (section [H.2.3.1](#_By_Import), page aa).

## A Sample of Detailed Algorithm Design

Samples of some detailed algorithm design can be found

## Procedure and Variable Description List

### Variable Lists for The OpenDB Class

(C): Class, (F): Function; (A): Array; (V): Variable; within functions: (A)/(V) = {some value}, is the default value for that function

* **(C)** openDB // A class used for opening a database connection.
  + (**F**) \_\_CONSTRUCT((**V**) host = "localhost", (**V**) username = "root", (**V**) password = "root", (**V**) database = "bookingSystem")
    - Public.
    - Is automatically called on the creation of a new instance of the class.
    - It will set the class-level variables to the values of those provided as parameters in the creation of the instance of the class.
  + (**F**) createDB()
    - Public.
    - Creates a database named by the variable during construction.
    - Returns 'true' if creation is successful and 'false' if creation has failed with (**F**) returnLatestError returning error number and description.
    - This method takes no parameters because there is already a (**V**) database variable set for the entire class. This class should only be used for a single database connect; multiple database connections should create a new instance of the class.
  + (**F**) connect()
    - Private (is not needed by itself outside of the class).
    - Creates a connection to the database
    - Returns ‘false’ if the connection is unsuccessful and ‘true’ otherwise.
  + (**F**) disconnect()
    - Private (is not needed by itself outside of the class).
    - Closes the connection to the database.
    - Returns ‘false’ if the disconnection is unsuccessful and ‘true’ otherwise.
  + (**F**) returnLatestError()
    - Public.
    - Returns an array consisting of two elements: the latest error number; and the latest correction error description.
  + (**F**) deleteFromTable((**V**) table, (**A**) data)
    - Public.
    - Builds a query from the array, (**A**) data, by using a foreach loop. Inside the foreach loop, it will use the temporary variables: (**V**) column, as the data; and (**V**) value, as the key.
    - Returns false if the deletion was unsuccessful.
  + (**F**) createTable((**V**) table, (**A**) data = NULL)
    - Public.
    - First, creates a table from the parameter, (**V**) table; it then loops through the array, (**A**) data, adding each element as a column to the table from the array.
    - Returns false if the creation of the table was unsuccessful.
  + (**F**) queryTable((**V**) table, (**A**) data = “\*”, (**V**) limit = NULL, (**V**) order = “ID”, (**V**) sort = “ASC”, (**V**) debug = false, (**V**) direct = false)
    - Public.
    - Takes the data from the relational array (data) and builds and SQL query from that. The function then executes the query.
    - Returns an array of the rows applicable to the query.
    - (**V**) debug outputs the query to the screen which is especially useful when debugging the SQL query.
    - (**V**) direct allows for a pre-built SQL query to be supplied in the (**A**) data parameter which will then be run directly.
  + (**F**) insertTable((**V**) table, (**A**) data)
    - Public.
    - Takes the data from the relational array (data) and inserts the data into the database.
    - Returns false if the insertion was unsuccessful.
  + (**F**) updateTable((**V**) table, (**A**) data, (**A**) where, (**V**) debug = false)
    - Public.
    - Takes the where parameter and updates the table with the specified data where the is a match.
    - Returns false if the update was unsuccessful.
    - (**V**) debug outputs the query to the screen which is especially useful when debugging the SQL query.
* (**F**) getUserLevel((**C**) bookingSystem, (**V**) userID)
  + Takes the class as a parameter and queries the database for the user level, given the userID.
  + Returns the user level.
* (**F**) formatTime((**V**) inputTime)
  + Takes the input time as an input parameter — this will be a 3 or 4 digit string that gets imported.
  + Returns the input time with a colon in the correct place.
* (**F**) serializeGetVariables()
  + Takes no parameters, but gets the $\_GET variables of the page that the function is called from.
  + Returns a string which is a serialised version of all of the $\_GET variables.
* (**F**) destroyCookies()
  + Sets cookies for the userID and userPassword to expire 3 days in the past with new content of a blank string.
  + No returned values.
* (**F**) checkUserLogin((**C**) bookingSystem, (**V**) userID, (**V**) password)
  + Check if a user exists with the given userID and password combination (whilst also making a call to saltAndHashPassword() so that the password can be hashed before checking against the database table.
  + Returns true if a user exists, else the function returns false.
* (**F**) returnUserLevelError((**C**) bookingSystem, (**V**) userLevelNeeded)
  + Checks in the database for the name of the user level associated with the level number specified in userLevelNeeded.
  + Returns a message stating that you have to be a [whatever the user level title is] to view the page.
* (**F**) saltAndHashPassword ((**V**) userID, (**V**) password)
  + Performs a SHA256 hash on the user ID and password, whilst capitalising the user ID, and appending the password after the capitalised user ID before hashing.
  + Returns the hashed string.
* (**F**) isCurrentPage((**V**) filename)
  + Checks if the current page is the one specified in the filename parameter.
  + Returns true if the page sought it the current page, else returns false.
* (**F**) generateTitle()
  + Finds what the current page is, whilst making calls to the isCurrentPage function.
  + Returns a title based on what the file is and what’s stored in the titleForFilenames.php page array.

A detailed description of all viewable pages used in the system is in the design section (setion [B.21](#_Detailed_Description_of), page 64).

### Variables Lists for Critical Pages to Functionality

#### addEventImport.php

|  |  |  |
| --- | --- | --- |
| **Name** | **Item Type** | **Description** |
| $\_POST (pre-defined) | Array | Contains all of the values of the data fields that have been submitted by the post method. |
| $userLevelNeeded | Variable | Used for making the statement for who can view the page absolutely clear; it isn’t essential to the functionality of the page. |
| $\_FILES (pre-defined) | Array | Used for storing uploaded files from a form. |
| $errorMessage | Variable | Used for storing a list of error messages. |
| $fullImportedCSV | Array | Used for storing the full, opened file. |
| $CSVHeaders | Variable | Holds the contents of the first row of the file. |
| $bookingSystem | Class | Used for connecting to the database. |
| $importedCSV | Array | Holds the opened file, excluding the first column |
| $successful | Array | Holds all success messages for imports. |
| $unsuccessful | Array | Holds all unsuccessful import messages. |
| $repeatInHolidays | Variable | A Boolean that is set to see if the event should repeat in holidays or not. |
| $row | Array | A sub-array of $importedCSV and is used to hold each row of the CSV file whilst parsing the data in a foreach loop. |
| $daysOfTheWeek | Array | Holds the shortened, ordered days of the week. |
| $dayPeriodCounter | Variable | Counts through each of the days and periods (5 days and 7 periods per day [therefore the counter counts 49 elements {0 to 48}]). |
| $cell | Variable | A string that contains the contents of each cell in the CSV file and is used in a foreach loop; this variable is then split up further into other variables that will be discussed in this section. |
| $roomID | Variable | Stores the room ID, as parsed from the $cell variable. |
| $ownerID | Variable | Stores the owner ID, as parsed from the $cell variable. |
| $eventName | Variable | Stores the event ID, as parsed from the $cell variable. |
| $insertDay | Variable | Stores, as an integer (from 0 to 4), what day the insert is for. |
| $insertPeriod | Variable | Stores, as an integer (from 1 to 7 [and hence the +1 in the code]), what the period is. |
| $insertDate | Variable | Stores the date of the selected by the user that is to be used for import to start from. |
| $terminateDate | Variable | This stores the date that the single event should repeat up until. |
| $insertPeriodTime | Variable | Stores the start time for a given period number (as found in the database). |
| $insertPeriodTime | Variable | Is a formatted version of the variable above; contains a colon between the hours and minutes. |
| $insertTimestamp | Variable | Stores the unix timestamp of that particular instance of a single event (so it is specific to a particular date, rather than day); you can see how this is calculated in the technical solution section, under the addEventImport.php heading (section [C.2.2](#_addEventImport.php), page 131). |
| $item | Variable | Variables used in foreach loops at the end of the page; the variable just stores the individual items of the $successful and $unsuccessful variables as the foreach loop loops through. |

#### Login.php

|  |  |  |
| --- | --- | --- |
| **Name** | **Item Type** | **Description** |
| $successMessage | Variable | Stores, as a string, all of the successful messages (such as “Logging in…” or the HTTP refresh to redirect the user). |
| $errorMessage | Variable | Stores, as a string, all of the error messages that the page returns; this may be something like “Incorrect password” or “The user does not exist”. |
| $\_POST (pre-defined) | Array | Contains all of the values of the data fields that have been submitted by the post method. |
| $userIDCheckResponse | Array | Used to check if the |
| $bookingSystem | Class | Used for connecting to the database. |
| $loginResponse | Array | The returned value from the query to the database check with the given user ID and password. |

#### viewAllEvents.php

|  |  |  |
| --- | --- | --- |
| **Name** | **Item Type** | **Description** |
| $userLevelNeeded | Variable | The user level that is needed to access the content of the page. |
| $bookingSystem | Class | The instantiated class (or object) that is used to connect to the database. |
| $\_COOKIE | Array | A pre-defined array which holds all data concerning the web browser’s cookies. |
| $\_GET | Array | A pre-defined array which holds all data concerning the web browser’s ‘get’ variables |
| $referenceWeek | Variable | The week that is being viewed on the current instance of the page. |
| $referenceTime | Variable | The time that is being used as a reference on the page. |
| $teacherName | Variable | If a teacher is set in the get variables, the teacher name for that teacher’s ID is assigned into this variable. |
| $\_SERVER | Array | A pre-defined array which holds all data concerning the server’s settings and states. |
| $eventResult | Array | The results of all of the events stored within a given week. |
| $dayCounter | Variable | A counter used in a ‘for’ loop for counting days. |
| $periodCounter | Variable | A counter used in a ‘for’ loop for counting periods. |
| $timeToCheck | Variable | The time to check for the given day and period. |
| $resultCount | Variable | The number of results found. |
| $resultLimit | Variable | The limit of the number of results. |
| $singleResult | Array | Used in a ‘foreach’ loop as the value of each element in $eventResult. |
| $fullSearchTime | Variable | This is the $referenceTime + $dayCounter \* (60\*80\*24) + strtotime(“1970-01-01 “.$timeToCheck).  In other words, this is the reference time, plus the number of days, plus the time at which a period occurs on that day. |
| $returnedTeacherList | Array | Lists all teachers for selection when the user is selecting what to view. |
| $buildTeacherList | Array | The array used for the ‘foreach’ loop. See the implemented code section for more details (section [C.2.17](#_viewAllEvents.php), page 211). |
| $teacher | Array | The individual teacher used for displaying teacher names and other details concerning the teacher directly. |
| $returnedRoomList | Array | Lists all rooms for selection when the user is selecting what to view. |
| $buildRoomList | Array | The array used for the ‘foreach’ loop. See the implemented code section for more details. (section [C.2.17](#_viewAllEvents.php), page 211). |
| $room | Arrays | The individual room used for displaying room names and other details concerning the room directly. |

## Annotated Listings of Program Code and Tailoring

The program code is well commented in the technical solution section (section [C](#_Technical_Solution_Section), page 92).

## Comments from the Client

**Mr Jacobs**: “*The system is well documented for all user levels of the system. I am also very confident that another programmer could take the work from Adam and extend the functionality of the system in the future.*”



Appraisal

F



# Appraisal

## User Feedback (and Beta Testing)

I created a Google Form and sent it to a selection of students, teachers and administrators. A copy of the form and the responses is below:

|  |  |
| --- | --- |
|  |  |
| *The first section for the form.* | *The section for only teachers.* |
|  |  |
| *The section for only administrators.* | *The general notes section at the end of the form.* |

## Analysis of User Feedback

|  |
| --- |
|  |
|  |
|  |
|  |
|  |

I have gathered, from these user responses, that the majority of things must function as expected (since all of the comments about unexpected things happening have mostly explained that they haven’t come across anything unexpected). However, the user feedback collected does tell me that there are improvements to make.

It has been made clear to me that the users that were questioned said that these were the main issues:

* Selection of teachers and rooms on the view events page needs to be easier.
  + This could be achieved by adding a search box at the top of the drop-down box for selecting these items.
* The ability for administrators to approve events, rather than teachers just being able to book them whenever they like. This will create a more unified experience in the system.
  + This could be achieved by following the initially proposed relationship model in the design section (section [B.4](#_Description_of_Record), page 45).
* The ability for students to be able to view their own timetable with the events that they are tied to.
  + This could, again, be achieved by following the initially proposed relationship model in the design section (section [B.4](#_Description_of_Record), page 45).
* That it was difficult to recognise some of the teachers just by their initials (when there were no first or last names provided in the view events page.
  + This could be made easier by ensuring that the administrator always adds first and last names to the records concerning teachers.

## Comparison of Project Performance Against Numbered General and Specific Objectives

*Key:*

|  |  |
| --- | --- |
| **Symbol** | **Meaning** |
| ↑ | Objective exceeded. |
| ✓ | Objective met. |
| ⎼ | Objective partially met. |
| ⨯ | Objective not met. |

1. The ability for timetabled lesson rooms to be exported from SIMS into the new system.
   * A CSV containing events can be exported from SIMS and imported into the system.
2. The ability for timetabled lesson rooms to be altered.
   * Individual events can be edited by clicking on them in the viewAllEvents.php page.
3. The ability for teachers to request new bookings rooms.
   * Teachers can request new bookings, but has to be done through direct contact with the administrator (and not the system that has been developed).
4. The system needs to be web-based.
   * The system is web-based, making use of PHP and MYSQL and only viewable in a web browser.
5. Administrators should be the only people able to approve room bookings.
   * Administrators can’t “approve” room bookings as such; they can add a room booking after one has been requested by contact from a teacher.
6. Administrators should be able to upload CSV files (or similar) to update data in batches.
   * Data can be uploaded using CSV files to update rooms, users and events all in batches, rather than individually (although individual entries can be done).
7. The system should be secure, and only people with the correct authority should be able to view certain parts of the system.
   * This has been achieved by adding a validation check on each page for the logged in user to make sure that the user’s user level is sufficiently high.
8. The administrators should be able to edit individual users’ details and permissions for the site.
   * This has been achieved by the edit screens accessed from the viewAllUsers.php page.
9. Should automate the importing of repeated bookings.
   * Repeat bookings can be imported by using the addEventsImport.php page and setting the repeat date to as far as the user wants into the future.
10. Should accept single and imported CSVs entries to be imported to the database.
    * There are single and imported entries for the rooms, users and events for the system and also single entries available for holidays (an import isn’t available for adding holidays as the previous systems can’t export any data that regards holidays).
11. Passwords should be necessary for accessing the system.
    * This objective wasn’t met, but it was reconsidered part-way through development that the security of the data was more important to the client than ease of access.
12. Passwords should not be stored in plain-text.
    * The passwords are not only hashed, but they are salted too so that the change of guessing a password from its hash value and a rainbow table is significantly small.
13. The database used for storing the data should be fully normalised.
    * The database used for storing the data is fully normalised.
14. When adding events, you should be able to pick a period (not a time).
    * Periods are chosen instead of times when adding an event.
15. The system should be intuitively laid out.
    * The system is very easy to navigate (as backed up the user feedback).
16. The system should be appealing to the eye.
    * The HCI was thought about extensively, although a comment about the appearance of the site was mentioned in the user feedback.

From these criteria comparisons, it can be seen that the majority of the system functionality has been produced from the objectives that were agreed at the start of the project and is therefore an effective solution to the problem. Any major changes were also communicated with the end user, Mr Jacobs, so that he wasn’t unpleasantly surprised with the final system.

The system will be easily maintained (as the system maintenance section (section [E](#_System_Maintenance), page 280) is well documented and easy to read).

## Possible Extensions

### System Security

One feature that could’ve been added to the user-login system is a feature that would create a row in the database each time that the user logs in, creating a unique session ID; this unique session ID could’ve then been stored as a cookie is the user’s browser. This session ID would then be deleted from the table when the user’s browser session is destroyed or a time limit is exceeded. This would have insured that users could only ever gain access to the site by using the login.php page.

Logins could also generate server-side sessions that will expire after a certain length of time. The system would then check the user login cookies and the session ID to make sure that the login is still valid.

### Other Additions

Another feature that could be added is the addition of a search for when a user is looking for a teacher and room in the drop-down lists on the viewAllEvents.php page (as suggested in the user feedback section (sections [F.1](#_User_Feedback_(and) and [F.2](#_Analysis_of_User), starting page 292)). Also, a search feature could be added that will search ALL events (for when a specific occurrence of an event is needed quickly). The SQL queries for these wouldn’t be too tricky to program.

Also, a feature could be added that would allow for the tables of data on each page to be sorted by different fields by clicking the heading on each of the tables that would then sort the table by that heading.

It was noticed that the project was no longer able to cope with imports after 26/3/16 — luckily the project was completely documented by this point! It is believed that this is due to the time-zone changing (the clocks went forward by one hour); this should be amended if the future if the project is to be properly implemented. It should be a relatively easy fix concerning the reference time calculation on the viewAllEvents.php page.

## Comments from the Client

**Mr Jacobs**: “*Adam has clearly reflected on the user feedback and has suggested possible improvements if the project were to be continued.*

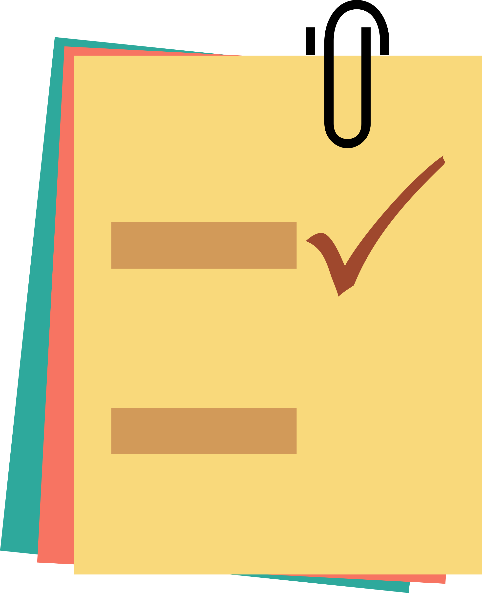
*It would’ve been best if Adam had used the more elaborate ERD for the database structure near the beginning of the project. This would have saved me time in the future with emailing teachers about event bookings and instead dealing with all requests through the booking system.*

*Adam also recently took on feedback regarding non-case sensitive user IDs for logging in. No two users are going to use the same user IDs with different cases and so this feature is very useful.*”



Notes

G



# Notes

## Spellings and Other Terms

Some spellings in the project may be inconsistent between American and English spellings (such as: serialise and serialize; and colour and color); this is because some functions and commands in the programming languages I’ve used use the American spelling of the words and I didn’t want to confuse the reader.

Also, some words will sometimes switch between the theoretical terms and their applied counterparts; for example, there may be a switch between ‘entities’ and ‘tables’ throughout my work.

Because PHP doesn’t require the programmer to define data types, there are some arrays that, under certain circumstances, will act as variables (and vice-versa). I will sometimes talk about the two interchangeably but only for this reason — I completely understand that arrays and variables are different. I also generally don’t specify a data type for the variables I talk about (unless I’m declaring them outside of PHP and in a MYSQL database, for example.

I also mention ‘foreach’ loops a lot; these are loops that are found in a lot of languages, but certainly not all. They’re basically used for looping through array elements, and work in a similar way to for loops but allow for the counter variable to be something more than just a number.

Calls to user-defined functions may contain significantly less parameters in some instances; don’t be alarmed. This is just because, for some function calls, default values in the function can be used instead. However, some functions may still contain default values explicitly. This is just because in the past, I may have used the debug feature or perhaps I actually needed a parameter in legacy code.

Room booking and event may also be used interchangeably; this is to help the transition between old and new terms. The new system uses “events” rather than “room bookings” so that “room bookings” and “rooms” couldn’t get mixed-up.

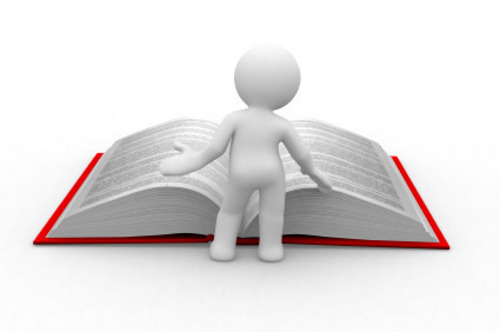
## Comments in Code

Between the different languages I’ve used, there are different delimiters for commenting; for example in PHP, they are delimited either by: “/\*[comment]\*/” or “//[comment]”, and in HTML they are delimited by “<!--[comment]-->”.

[END OF DOCUMENTATION]

User Manual

H



# User Manual

## A Brief Introduction and Installation Instructions

### Installing the Hardware and Software

The room booking system I have developed for Kings Priory School will be used to book rooms for teachers throughout the school. It will also allow for administrators to manage users, by creating logins and passwords, and will allow for students to view events for a given teacher or room.

The system will run on an Apache, MYSQL and PHP server. I recommend using a LAMP (Linux Apache MYSQL PHP) server (or WAMP [Windows Apache MYSQL PHP] for a Windows machine, although setup examples will only be given for Ubuntu Server 14.04 LTS).

I recommend using Ubuntu Server 14.04 LTS for the system to run on; the installation instructions for a LAMP server on Ubuntu Server 14.04 LTS can be found here: <https://help.ubuntu.com/community/ApacheMySQLPHP>. The system hardware specification is recommended to be:

* OS: **Ubuntu Server 14.04 LTS 64-bit**.
* Processor: **2.4GHz, quad-core**.
* RAM: **8GB, 1333MHz**.
* Hard disk (just for the booking system): **19MB** (including approximate data in the database for one year).
  + Note: The LAMP server will take hard disk space, and this needs to be accommodated for when choosing the hard disk for the server.
  + HDDs aren’t usually too expensive, so I used a 500GB disk for testing the system.
* Display: **Minimum resolution width of: 1000px** (on the client’s system).

The above system hardware specification was the specification that was used for testing. Lower specification hardware **may** work, but they haven’t been tested.

The zip file “bookingSystem.zip” contains all of the files for the booking system to function. These files should be extracted from the zip file and placed in “/var/www/html“. This will make the files visible locally at “<http://localhost/>” and publicly visible at your server’s IP address.

To avoid any problems with finding the server’s IP address, it is recommended that a static IP address is set for the server; instructions on how to set this for a Ubuntu Server can be found here: <https://help.ubuntu.com/lts/serverguide/network-configuration.html> under the “Static IP Address Assignment” heading.

Additionally, if the server is to be used outside of the LAN, the server’s HTTP port must be port-forwarded; you will first have to see if your router supports port-forwarding. If your router does support port-forwarding, you will have to lookup yourself specific instructions on how to port-forward your specific router. Here is instructions on how to port-forward a Virgin Media router: <http://help.virginmedia.com/system/selfservice.controller?CMD=VIEW_ARTICLE&ARTICLE_ID=27625&CONFIGURATION=1001&PARTITION_ID=1>.

Below is a selection of images which show how to access the port-forwarding menu on a Virgin Media Super Hub 2ac.

|  |  |
| --- | --- |
| Advanced-Settings-arrows.jpg | Port-Forwarding400.jpg |
| *Virgin Media Super Hub 2ac Advanced Settings page.* | *Virgin Media Super Hub 2ac Port Forwarding page.* |

Once you have set-up port-forwarding, it may also be useful to set-up a dynamic DNS to point to your dynamic public IP address. This can be done for free through: <http://www.noip.com/>.

If my dynamic DNS hostname was set to “<http://roombooking.no-ip.org>” then you would access the system through that address directly. If you are forwarding to a different port to 80, you will need to specify this when connecting. For example, if the forwarded port is port 4321, you would connect to the server by using: “<http://roombooking.no-ip.org:4321>”.

Of course, if the server is to be used only locally, the server’s address can be specific like this: “<http://192.168.0.242:4321>”, or similar.

### Setting up the Booking System

Now the booking system will need to be set up to communicate with the MYSQL server. When setting up the LAMP server earlier, you should’ve been asked for a username and password; if you can’t remember this password, don’t worry — you can reset it using the following commands: <https://www.howtoforge.com/setting-changing-resetting-mysql-root-passwords>.

Once you know the username and password, you will need to add this to the configuration file in the booking system. This will be located at “/var/www/html/includes/SQLDetails.php”; on this page, there is a page that will look like this:

<?php

**include\_once** "class.openDB.inc.php";

*// Here, you will need to create a new instance of the class 'openDB'. There's an example setup below:*

*// $bookingSystem = new openDB("localhost", "rootUser", "rootPass", "bookingSystem");*

*// Below makes the instantiation process explicit:*

*// $bookingSystem = new openDB({HOST}, {USERNAME}, {PASSWORD}, {DATABASENAME});*

$bookingSystem = **new** openDB("localhost", "root", "root", "bookingSystem");

?>

You need to edit the line which currently reads “$bookingSystem = **new** openDB("localhost", "root", "root", "bookingSystem");”, replacing “root” and “root” with your username and password for your MYSQL server. You can also optionally change the name of the database (to something other than “bookingSystem”, although this must be kept the same throughout use or a new database will be created [and you won’t be able to use the site with existing data!]). This should make the booking system communicate with the MYSQL server. If the system cannot communicate with the MYSQL server, you will get the following error message when you access “<http://localhost/index.php>”: “Failed to communicate with the database. Please check the MYSQL credentials.”. If you don’t receive this error message, then the booking system has successfully been able to communicate with the MYSQL server!

Next, you will have to set-up an administrator user to logon to the system to set everything up. You will first need to run “<http://localhost/index.php>” in a web browser once for the database for the system to be initialised.

You will now need to access the MYSQL server’s databases using a software package called: “PHPMyAdmin”. This can be accessed by visiting “<http://localhost/phpmyadmin>” (and replacing “localhost” with the appropriate address). You can log into this software by using the MYSQL username and password that you set up earlier.

|  |
| --- |
|  |
| *PHPMyAdmin welcome screen.* |

Once you have logged into PHPMyAdmin, you should be greeted with a screen looking something similar to the one below:

|  |
| --- |
|  |
| *Introductory screen after logging into PHPMyAdmin.* |

As you can see, on the left-hand side there is a database called “bookingSystem”; this is the one we created earlier by running the page “<http://localhost/index.php>”. If the database “bookingSystem” does not exist in this screen, please try checking your MYSQL details and going through the initialisation process of “/var/www/html/includes/SQLDetails.php”.

Don’t worry about the other databases; **leave them alone**.

If the database has appeared in PHPMyAdmin, click on “bookingSystem”; you should be greeted with a screen that looks like something similar to below:

|  |
| --- |
|  |
| *List of tables within the “bookingSystem” database.* |

You will now need to click on the “Users” table.

|  |
| --- |
|  |
| *After clicking on the “Users” table.* |

You will now need to insert a User (your user for administrative purposes). Click on “Insert” on the top menu. You should get a screen like the one below:

|  |
| --- |
|  |
| *Screen for Inserting into “Users”.* |

You will now need to fill in each data field in the top section of fields (only the top 6 boxes); if you want to, the emailAddress, firstName and lastName fields can be left blank (with the “Null” box checked), but the userID, password and userLevel fields must be filled in.

For the userID, pick something sensible that is between 1 and 20 characters long; I recommend a userID of at least 3 characters.

For someone called “Mickey Mouse”, a good userID may be something like “MMo” or “MiM” or “MMouse”.

The password is a bit trickier as it will have to involve you hashing a password. It is recommended that you use this website to hash your initial password: “<http://www.xorbin.com/tools/sha256-hash-calculator>”.

|  |
| --- |
| Capture.PNG |
| *Example hashed password if your userID was “MMouse” and your password was “password”.* |

Notice that the **user ID has to be capitalised**.

You will now need to copy the hashed password (the part highlighted in blue above) and paste it into the password field on PHPMyAdmin.

For the userLevel, you should enter the number “3”, without the quotes (as you will be an administrator).

Your details should look something similar to those below:

|  |
| --- |
| Capture.PNG |
| *Example inputted data.* |

You should now click the “Go” button that is directly below the data fields that you have just filled in (the one that is highlighted below).

|  |
| --- |
| Capture.PNG |
| *Zoomed in example of the filled in data fields.* |

After you have clicked the “Go” button, you should see a screen like below:

|  |
| --- |
| Capture.PNG |
| *Screen showing the success message after adding first user to the system.* |

You should now be able to login to the system by visiting “<http://localhost/login.php>” and entering “MMouse” as the user ID and “password” as the password.

|  |  |
| --- | --- |
|  |  |
| *Screens showing the login for “MMouse” and “password” working successfully.* | |

Once the first login has been made, it is very easy to add more users (see the next section (section [H.2.3](#_Points_of_Data), page y) for details on how to do this).

If you are still having difficulty with the login procedure, please email [adam@adamblakey.co.uk](mailto:adam@adamblakey.co.uk), and I will do my best to help.

## Detailed Description of the Use of the Full System

### Types of Error Messages

You may experience several errors when using the system on a day-to-day basis. One kind of error that you may experience is a set of errors like the one below. Notice also that the event failed to insert because an event name wasn’t given.

|  |
| --- |
| Capture.PNG |
| *Screen showing an error where the user hasn’t entered the event name.* |

You may also experience some more general errors with the system, such as the one below where some of the cells in the CSV file given for an event insert were not correctly formatted. Notice also that there is a success message for the successful entries and unsuccessful messages for the unsuccessful entries.

|  |
| --- |
|  |
|  |
| etc. |
|  |
| etc. |

You may also come across some validation errors too. For example, if you try to add a holiday where the end date is before the start date, you will get an error like the one below:

|  |
| --- |
|  |
| *Example of a validation error.* |

### Recovery Procedures

If you the database table becomes slightly corrupt (by some tables being accidently corrupted, for example), the original data can be restored without causing an issue to the rest of the site by just reimporting the data that has been corrupt. However, if the data has been automatically inserted for you (by importing an event and the user and room records were created automatically, then it is advised that you should clear all data in the database and start again). If the data that is corrupted was added manually, then this will need to be redone manually.

A good practice is to keep frequent backups of the database so that, if the data does somehow become corrupt, the system can be rolled back to a fairly recent backup. Please see this website, which goes through exporting data and reimporting it: <https://support.managed.com/kb/a2034/how-to-backup-and-or-restore-your-mysql-database-using-phpmyadmin.aspx>

### Points of Data Entry

Data can be entered to the system on one of the following places:

* Add Event.
  + - Add Event: Single.
    - Add Event: Import.
* Admin Panel.
  + Users.
    - Add Users: By Import.
    - Add Users: Manually.
  + Rooms.
    - Add Rooms: By Import.
    - Add Rooms: Manually.
  + Holidays.
    - Add Holidays: Manually.

Each page is fairly intuitive, so I will go through the two main types of insertion — by import and manually.

#### By Import

The Add Event: Import page is the most complex page out of the by import pages (mainly because of the number of options that can be chosen). Here is an example of some valid data being entered on this page:

|  |
| --- |
|  |
| *Valid data insertion for the Add Event: Import page.* |

There are some rules that you should bear in-mind when using this page in particular:

1. If the “Repeat event weekly” checkbox is left **unchecked**, then events will be inserted for a week after “The date that the events should start from”.
2. If the “Repeat event weekly” checkbox is **checked** and the “Continue to repeat in holidays” is left **unchecked**, then the events will be inserted after “The date that the events should start from” until “The date that the event should repeat up to”, but they **will not** be inserted for dates which are included in a holiday date range.
3. If the “Repeat event weekly” checkbox is **checked** and the “Continue to repeat in holidays” is **checked**, then the events will be inserted after “The date that the events should start from” until “The date that the event should repeat up to”, regardless of whether or not any dates are included in a holiday date range.

#### Manually

Manual data entry points are quite intuitive in comparison to the by import entries; each data field on the form will directly link to one piece of data displayed somewhere else.

An example page could be the addUserSingle.php page; this page asks for:

* User ID.
* Email address.
* Password.
* Password confirm.
* First name.
* Last name.

Each data field on this form clearly links to one piece of data (one characteristic about that particular user). The only piece of data that is not essential here is the password confirm, but that is included as a validation check to increase the likeliness of the user entered the password correctly.

#### Editing

Edit screens are also very intuitive as each of the edit screens on the site only ever concern a single record.

An example page could be the editEventSingle.php page. This page is accessed by clicking on an event’s name in the viewAllEvents.php page. The page is relatively intuitive and will already contain filled-in data fields for the event that you have clicked on. There is an example event below:

|  |
| --- |
|  |
| *The editEventSingle.php page showing an example page already filled in with details.* |

To change the event’s details, just edit the fields on the form and click “Save Edited Event”. You can also delete the event from this screen (which would be achieved by clicking the “Delete Event” button). Bear in mind that recovery of data after deleting it on this page is very unlikely; do not delete any event unless you really don’t want it!

Once the save or delete operation has been carried out, you will be redirected to the viewAllEvents.php page, and at the same place where it was before you clicked on the event’s name to edit it. If you’ve edited or deleted the event, you should see that the event’s details on this page have changed or disappeared.

## Comments from the Client

**Mr Jacobs**: “*The user manual is extensive and meets the needs of any eventual administrator. It is very easy to follow.*

*More run screens would make the manual slightly easier to follow.*”



[END OF USER MANUAL]