Advanced Web Techniques - 6G6Z1011
Online Website – 1CWK50
December 2014
Keith Yates

David M^c^Queen – 10153465

1. Design

1.1 Database

Using the specification provided for the NHS Shift Management system, the key aspects were extracted to assist in the design on an appropriate system.
The system needs to store multiple users, in which each user belongs to a specific level. The nature of the system is to manage users shifts; as such a table is required to store which shifts each staff member is working.
With this in mind, a database structure was developed which fulfils all of these storage requirements, using 3 separate tables. Figure 1 shows a visual representation of the database structure, whilst figure 2 shows a use case for the system.

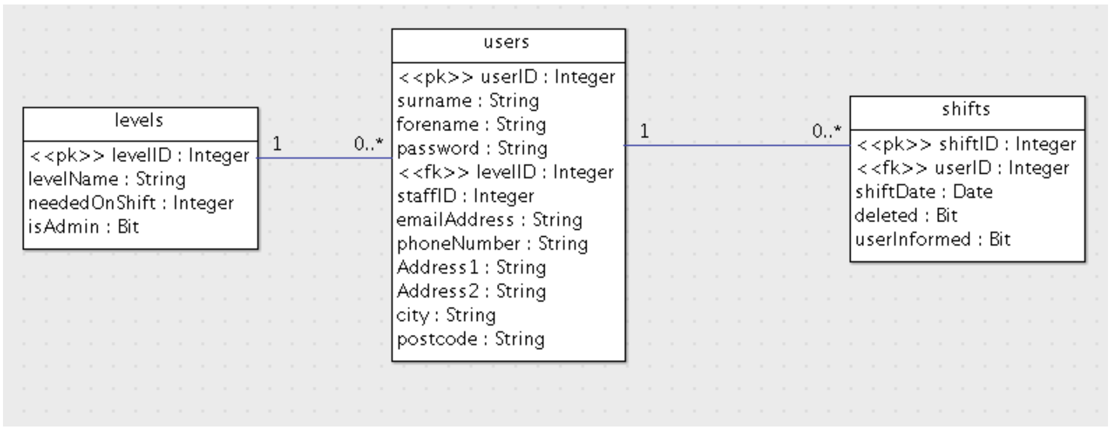


Figure 1 – ERD database design

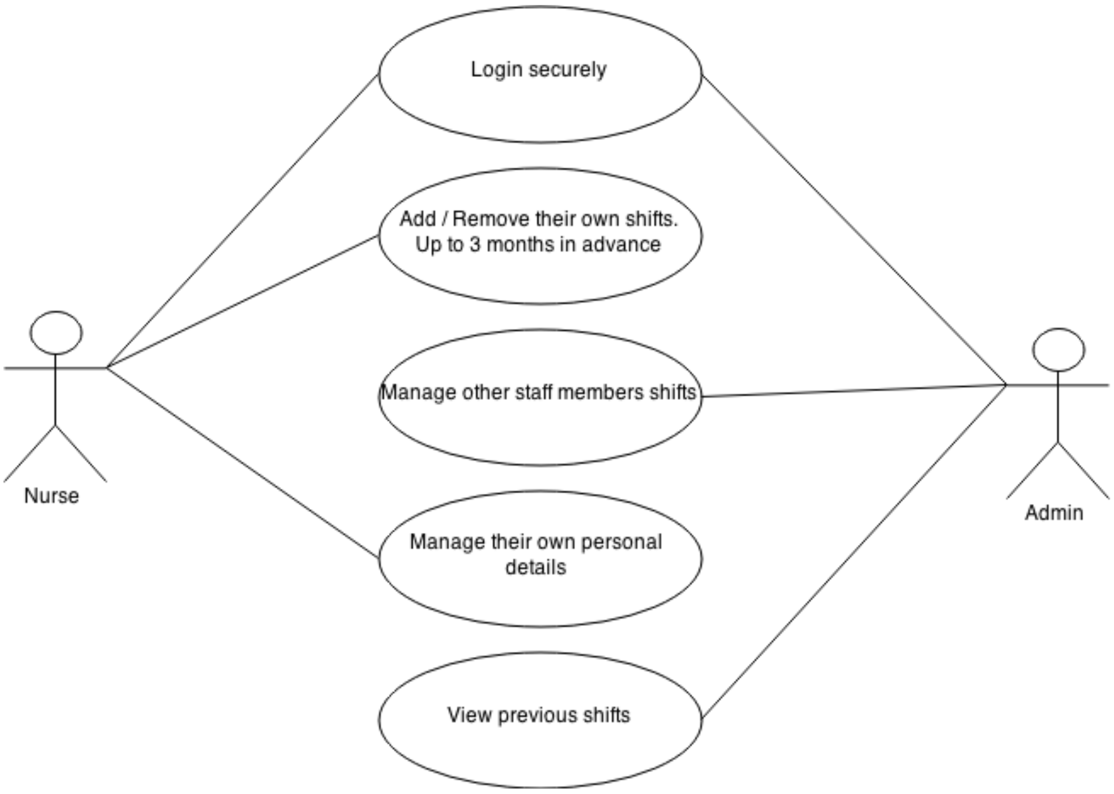

Login securely
Add / Remove their own shifts. Up to 3 months in advance
Manage other staff members shifts
Manage their own personal details
View previous shifts
Nurse
Admin

Figure 2 – Use case for the system

1.1.2 Database Tables

The users table (table 1) contains all of the user information, including the personal information, in one central location. There is a NHS staffID column that connects the staff member to their main NHS account. The levelID links the staff member to a specific staffing level.
The password field will store passwords that have been hashed, to prevent plain text passwords getting stored in the database.
The email field is the users personal email address, and bears no relation to the email that is used in order to log into the system.

Table 1 – Users Table Data Dictionary

Description	Column Name	Data Type	Length	Nullable	Unique	Key
The unique, auto incrementing userID	userID	int		False	True	Primary
Staff members surname	surname	varchar	200	False	False	
Staff members forename	forename	varchar	200	False	False	
Encrypted Password	password	varchar		False	False	
Staff levelID	levelID	int		False	False	Foreign (levels)
Staff NHS ID	staffID	int		False	True	
Personal email address	emailAddress	varchar	100	True	False	
Personal phone number	phoneNumber	varchar	14	True	False	
Personal address	Address1	varchar	100	True	False	
Personal address	Address2	varchar	100	True	False	
Personal address	City	varchar	100	True	False	
Personal address	postcode	varchar	9	True	False	

The levels table (table 2) is to contain information regarding each staff level. It is structured in a method that makes further development of the system easy, with

little to no modification of the existing system needed. One such development would be allowing for new staffing levels to be created (such as junior nurse). If a new level is created, this will be reflected in the relevant areas of the shift management tool, such as the minimum amount of staff needed per shift, per level (neededOnShift column).
The 'isAdmin' column determines if the staff level should be granted admin permission within the system, this allows for the possibility of multiple staffing levels having admin permission that could be of use if the system needs to be expanded, or linked to another system.
The levels table links to the users table on levelID.

Table 2 – Levels Table Data Dictionary

Description	Column Name	Data Type	Length	Nullable	Unique	Key
The unique, auto incrementing levelID	levelID	int		False	True	Primary
The level name	levelName	varchar	100	False	False	
How many staff of this level are needed as a minimum for a shift	neededOnShift	int		False	False	
Determine if the level is granted Admin privileges.	isAdmin	bit		False	False	

The shifts table (table 3) contains each and every shift, for all staff members. If a shift is deleted, then it is kept in the tables but is marked as deleted. This allows for deleted shifts to be reviewed, and it also allows for the deleted shift to be communicated to the staff member. The date format used to store the shift date is the international standard, ISO 8601 (ISO, 2014).
If any shift is created or deleted by admin, the userInformed column will allow the system to find these shifts, and then communicate the relevant message to the user.

Table 3 – Shifts Table Data Dictionary

Description	Column Name	Data type	length	Nullable	Unique	Key
The unique, auto incrementing	shiftID	bigint		False	True	Primary

shiftID						
The user which is working the shift	userID	Int		False	False	Foreign (users)
The date of the shift	shiftDate	Date (ISO 8601)		False	False	
If the shift has been deleted	Deleted	Bit		False	False	
If the user is aware of the shift (Admin created)	userInformed	Bit		False	False	

1.1.3 Stored Procedures

"*A stored procedure in SQL is a group of one or more SQL statements*" (Microsoft, 2014). These group functionality together, carrying out pre determined tasks, optionally accepting inputs and returning record sets. They are similar to functions in Object Orientated languages, and are useful to abstract database functionality away from the web server and keep it local to the database.
For the shift management system, all interaction with the database will be carried out using Stored Procedures. This drastically reduces the level of SQL code that is present on the server (section 1.4), and keeps all database functionality together. The only SQL code that is present on the server is the call to the relevant stored procedure.
As a minimum, stored procedures will be needed to carry out the following tasks:

- Create a user
- Read users details
- Update a users details
- Delete a user
- Create Shift
- Read Shifts
- Delete Shift
- Login

1.2 Server

The server section, which connects the database to the frontend display, will be created using the Model View Controller (MVC) pattern. This pattern separates each of the different sections out, so that each area is atomic, only being responsible for one specific task. The Database section (model) doesn't handle any logic or display; it is purely responsible for retrieving and updating the data, using stored procedures (section 1.2). The display section (View) simply displays information to the user, using data passed from the controller. The logic section (controller) connects the Model and View sections together. It performs all of the logic, calling the Model to perform any database functionality, and calling the View relevant for the information that needs to be displayed.

In order to implement the MVC pattern, the CodeIgniter framework will be utilised. "*CodeIgniter is a community-developed open source project*" (CodeIgniter 2014) "*with a very small footprint*". It allows for MVC to be implemented in an easy manor, allowing for security to be added preventing malicious attack.

1.3 Frontend

To achieve all of the functionality set out in the specification, a total of 3 pages will be available for the user.

- Login
- Calendar View
- Settings

1.3.1 Login

The login page will display a simple form, allowing the user to enter an email address and password. This will be the first page the user sees, and the page in which they are directed to if they are not logged in. Figure 3 shows the design of the login page, 2 input fields with a 'login' button.

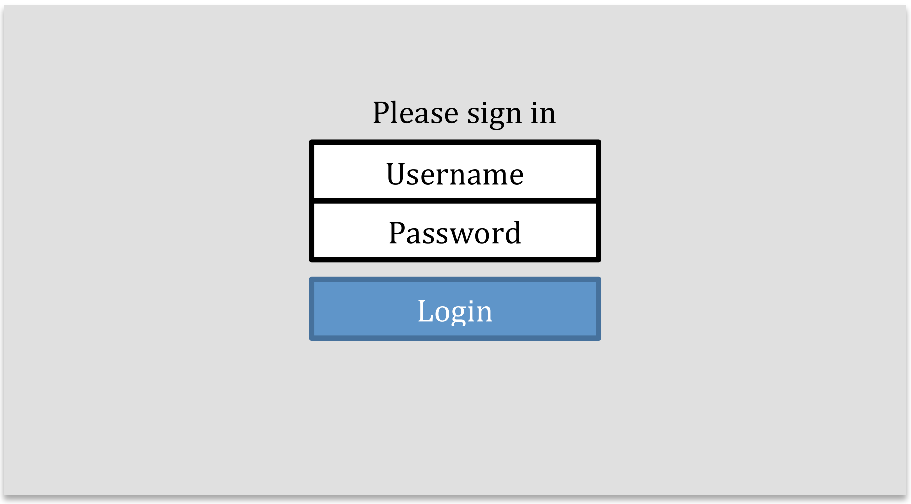


Figure 3 – Design of the login page

1.3.2 Calendar

The calendar view (Figure 4) will be the main view in which the user will use. This will display the calendar, which will allow the user to carry out all the shift management described in the specification.
Alongside displaying the calendar, messages will be displayed to the user (Figure 4, section 4.2). The messages will be; any modifications made by admin to the users shift; Instructions on how to use the calendar; a list of weeks which the user is below the shift level specified; Error messages and warning messages.

The calendar (Figure 4, section 4.3) aspect will be created using the fullcalendar.io library. This is a JQuery library released under the MIT license, meaning it can be used in almost any way so long as credit to the creator is left in tact.
There is one main benefit to using FullCalendar, over creating a new calendar system; there is no need to reinvent the wheel. Multiple aspects are modifiable enabling FullCalendar to fulfil all of the functionality set out in the specification., without the needed to create a new calendar. Through modifying different

parameters in the <script> section of the page, all functionality can be achieved, from changing the background colour of the events to handling the process of when a user clicks on a specific day.
As Fullcalender doesn't provide a 4-week view by default, it will be necessary to create a new view, in the core code, which fulfils the requirements of the spec; to show the next 4 weeks, with the first week being the next week.

At the top of the calendar view will be a navigation bar (Figure 4, section 4.1), this will display the users name along with navigation controls, allowing the user to access the settings page and logout. The Navigation Bar is present at the top of every page in which a logged in user sees.

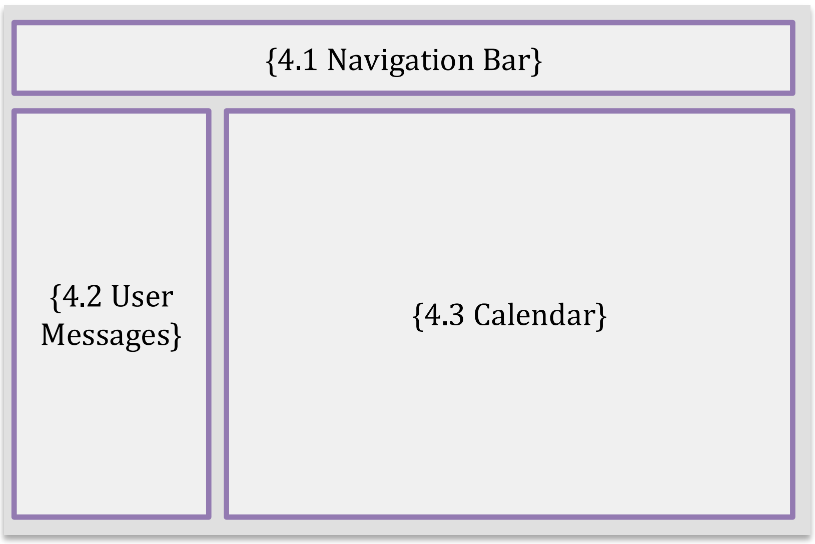


Figure 4 – Basic layout of the calendar view

1.3.2.1 Improvements

The standard user will only have one view available to them, as covered in section 1.4.2, however Admin will have 2 views available. These will be a month view, and a week view. The month view will allow admin users to view the entire month, previous months, and future months. Thus they will be able to review all previous months, and all staff shifts, providing a record of which shifts have been worked in the chance that this information is needed for administrative purposes.
The week view will provide more screen real estate in which to view the staff members that are working. This is essential due to the fact that each staff member is displayed, as opposed to the staff categories the standard user sees. Whilst this isn't a major problem with the current staff levels and staffing rules, if these were to change the system is able to accommodate for this increase and easily display all the staff that are working on specific days.

1.3.3 Settings

The settings page, figure 5, will display all of the current users personal details, which are in the system. This will allow the user to update their personal details as circumstances change. Further to being able to update persona details, users will be able to create a new password, which must meet the password requirements.

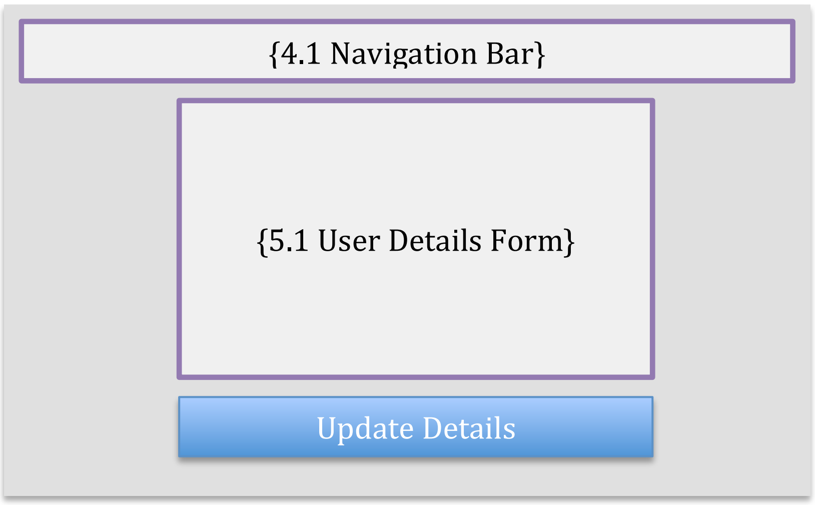


Figure 5 – Basic layout of the user settings page

1.3.4 Bootstrap

To assist in the presentation and style of the system, the Bootstrap framework will be utilised. "*Bootstrap is the most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web*" (Bootstrap, 2014), it is released under the MIT license, making it freely available to use.
Bootstrap provides an easy to use framework that can be applied to existing HTML in order to create beautiful, clean displays. As such it will be used to style display the system, being applied to each page.

2. Implementation
Throughout the development of the system, all of the code is checked into source control, using GitHub. Source control is an incredibly valuable tool, allowing for every stage of development to be reverted to, creating a checkpoint in case of any catastrophic failures in future development.

2.1 Database
The database is the initial section of the system to be created; it is the central hub for all of the data and the structure of the data can have a great impact on the rest of the system, affecting how the server requests the data, and the classes in which the data is based on.
Using a text editor (sublime), a database script was created which contained all the tables and stored procedures, along with dummy data inserted into the database. This allowed for easy transfer between database instances, such as the local development environment and the production environment. Creating the SQL code in such a way enables it to be stored in source control along with the rest of the development code.

MySQLWorkbench was used in order to connect to the database instances, further to this, it was used to write and execute additional SQL code throughout the development of the system. MySQLWorkbench proved to be a better portal for accessing the database instance than that of phpMyAdmin that ships with Xampp servers.
The SQL code that has been created for the system makes extensive use of various database operations, Including: Inner Joins, Left Joins, Aggregate functions, Group By, an Order By Statements. Making use of these operations allows for the creating of efficient SQL that performs all the necessary data modifications in one batch.

Alongside the stored procedures described in chapter 1.2, additional stored procedures were created to perform basic functionality that, although not necessary for the specified system, would be desirable for future development and ease of modification.
These stored procedures perform tasks such as:
- Create staff members
- Delete staff members
- Add new levels
- Delete levels
- Edit levels

If necessary, further development could be easily made to the system, allowing for these stored procedures to be utilised, thus providing extended functionality to the Admin level users.
The database script, created for all database operations, can be found on the server where the web application is hosted (CW1/Database/DatabaseSchema.sql).

2.1.1 Passwords

The Admin user that was defined in the specification had an incorrect password, and as such the password was changed to 'Organ1sed', the rest of the staff details have remained the same.

The passwords for each user are hashed to prevent plain text passwords being readable. In order to enhance the security of the passwords, each is 'salted' on either side of the user-defined password. This comprises of a Pre an Post salt. Essentially sandwiching the password between these salts, ensuring that no hashed password is the same as another hashed password, meaning the possibility f using hashing tables to attack the system is drastically reduced. The 'pre-salt' is a random string comprising of letters, numbers and symbols ('zhjbfvh56^%&'). The 'post-salt' is the users staff ID. Doing so ensures that the salt changes for each user. Using the admin timetable account as an example, which has the staff ID of '6189' and the password 'Organ1sed'. When this is concatenated with both of the salts results in a pre-hash password of 'zhjbfvh56^%&Organ1sed6189', which is then hashed with the SHA256 algorithm. If another user uses the same password, the resulting hash would be different as the post-salt is different.
In order to ensure that the password hashing worked fully, it was necessary to explicitly make the predicate use Binary comparison and explicitly set the collation, as shown in Code Example 1. Doing so ensures that when the database is deployed to a different database instance, the comparison would work as expected.

```
"BINARY u.password COLLATE utf8_general_ci = BINARY
SHA2(CONCAT('zhjbfvh56^%&' , CONCAT(password, u.staffID)),  256) COLLATE
utf8_general_ci;"
```
Code Example 1 – Password matching using Binary comparisons, and explicitly setting the collation.

2.2 Server

In order to develop the system locally, Xampp was installed on the development machine. This allowed for server side scripts (such as index.php) to run. Xampp was chosen as the specification stated the system needed to be developed in php and MySQL, both of which can run on a Xampp server.

As discussed in chapter 1.2, the MVC framework CodeIgniter was used for the development of the server side code. After configuring the framework to connect to the database instance, work commenced on the database access functions, which are present in the Model section. Comprising of 2 models, 1 for the shift functionality and a 2nd for the user functionality, each of the functions call the relevant stored procedure in the database, and then pass the result set back to the controller as a result array.

To perform the server side logic, 3 Controllers were created; One each for shift functionality and user functionality, and a 3rd specifically for verifying the users login credentials. The controller for verifying user credentials was kept separate from the other 2 controllers because although it was handling user functions, the user is not logged into the system at this point so it was prudent to keep logged in functionality and non-logged in functionality separately. Alongside the controller deciding which Model to call, it was also responsible for deciding which View to show to the user. This is relatively straight forward, with a single controller generally having a single associated view (other than the common header and footer files). With the main calendar, there could be 2 different level of user; Admin or Standard. As such different functionality is needed for each user. In this instance, the controller makes a decision on which view to load depending on if the user is marked as admin or not.

2.2.1 Server Side Validation

The controller also performs server side validation, which is essential to work as support for client side validation ensuring that the data input by the user is a) valid data for what has been requested, and b) non malicious, such as SQL injection. Server side validation is essential as client side validation can't always be trusted; JavaScript might be displayed on the clients machine, or the client might intentionally bypass the validation, both of which could cause problems in the logic and database sections of the system. It is not possible to bypass validation on the server, due to the fact that clients don't have access to the server itself, thus they don't have access to the validation code.

Although it is possible to perform validation server side, it is not a replacement for client side validation. Client side validation is essential for providing the user with a better experience using the system. They can be alerted of any issues with their inputted data without leaving the page. This has multiple benefits: If the data is invalid then it will need to be re-entered, if validation solely takes place on the server then when the page reloads the user will need to re-enter all of the data they have previously entered. Further to this, the process is creating unnecessary communication between the client and the server, which could have adverse affects if the user is on a slow connection, such as a mobile device.
Code Example 2 shows the server side validation that is applied to the user updating their personal details. In this example an array is being created with the new details, before the array is passed to the model that will perform the update in the database. Multiple operations are taking place here, for each of the values. The first is a ternary operator, checking the length of the new value. If it is less than 0, an empty sting is inserted into the database. However if there is a length greater than 0, the new value is used. In using this new value, it is limited to the max data size that the table can hold, to ensure that trying to insert a string that is too long generates no database errors. Using ternary operators in this situation removes the need for IF blocks for each new value, that would take up extra room and duplicated code, as the ternary operators are simple to understand and can be condensed onto a single line.

```
$userSettings = array(
'userID' => $userID,
'password' => $newSettings['password'],
'forename' => (strlen($newSettings['forename']) > 0 ? substr($newSettings['forename'], 0, 200) : ''),
'surname' => (strlen($newSettings['surname']) > 0 ? substr($newSettings['surname'], 0, 200) : ''),
'email' => (strlen($newSettings['emailAddress']) > 0 ? substr($newSettings['emailAddress'], 0, 100) :
''),
'phone' => (strlen($newSettings['phoneNumber']) > 0 ? substr($newSettings['phoneNumber'], 0, 14)
: ''),
'address1' => (strlen($newSettings['address1']) > 0 ? substr($newSettings['address1'], 0, 100) : ''),
'address2' => (strlen($newSettings['address2']) > 0 ? substr($newSettings['address2'], 0, 100) : ''),
'city' => (strlen($newSettings['city']) > 0 ? substr($newSettings['city'], 0, 100) : ''),
'postcode' => (strlen($newSettings['postcode']) > 0 ? substr($newSettings['postcode'], 0, 9) : '')
);
```
Code Example 2 – Server side validation on updating the users settings

The security that was implemented on the system comprises of storing the users details in a session, which is destroyed when the user logs out. When the user logs in, by providing valid credentials, the server retrieves specific user information from the database, including their unique user ID, and their name. With this information stored in a session, a check is performed at the beginning of each function performing sensitive tasks. If the user is logged in, then they are allowed to proceed. However if they are not logged in, then the user is redirected to the login page and prompted to login with valid credentials.
Performing security in this method has similar benefits to those discussed for server side validation; the client is unable to bypass the security checks, as they don't have access to the server.

2.3 Frontend

The front end, visual aspect, of the system was developed using the 3 main web development technologies; HTML, CSS, and JavaScript. All of these languages are essential in creating a website that is required to display content to the user. Alongside the main languages, several libraries and frameworks were used.
Bootstrap (chapter 1.4.4) was used to style the system, making use of both the display elements and the JavaScript functionality to enhance the display elements by using transition elements.
FullCalendar (chapter 1.4.2) was used to display and handle all the frontend calendar functionality. FullCalendar is a library that is specifically designed for displaying and handling calendar events, the platform it provides for creating custom functionality was utilised heavily for the purpose of this system. Event handlers, such as 'DayClick', 'EventClick', 'ViewRender', and 'eventAfterAllRender' were all utilised in the development of the front end of the system. With the different event handlers, it was easy to separate the different segments of code, keeping functionality separate and increasing code maintainability and readability. The specific code for each section was placed inside the relevant handlers, with some code being abstracted out into a separate function, which is then called from inside the handler, to increase code reusability.
As different functionality is needed for both the standard user and the admin user, 2 separate scripts were created, with one or the other being loaded depending on the user logged in. The decision on which script to load is made inside the controller, server side (as described in chapter 2.2). Once the relevant script has been loaded, the user is able to use the functionality in the calendar, whilst only having access to the functionality in which they are meant to have access to. This increases security, as non-admin users are unable to use the admin functionality, however this would also be prevented server side, and increases code maintainability as there is a different file for the different functionality. When making modifications that only apply to admin users, then changes only need to be made to the admin script file.

An example of the functionality that Fullcalendar is responsible for is displaying, to admin users, which days are understaffed. This is achieved by counting the events that are on a specific day, Code Example 3, and if there are not enough, highlight the relevant day, Image 1. This differentiates between standard nurse, and senior nurse, highlighting the days if either of these are below the required amount.

```
//Get all events for the day
    dayEventsCalculate = $('#calendar').fullCalendar('clientEvents', function (event) {
      return moment(event.start).isSame(nextDay, 'day');
    });

    dayShiftMissingCounter = 0;
   seniorMissingCounter = 0;
    standardMissingCounter = 0;
    dayEventsCalculate.forEach(function (event) {
      //Count up all of the shifts the user is working for the week
      if (event.onShift == 1) {
        dayShiftMissingCounter += 1;
        if (event.levelID == 2){
          standardMissingCounter += 1;
        }else if(event.levelID == 3){
          seniorMissingCounter += 1;
        }
      }
    });
```
Code Example 3 – Counting the staff on each day.

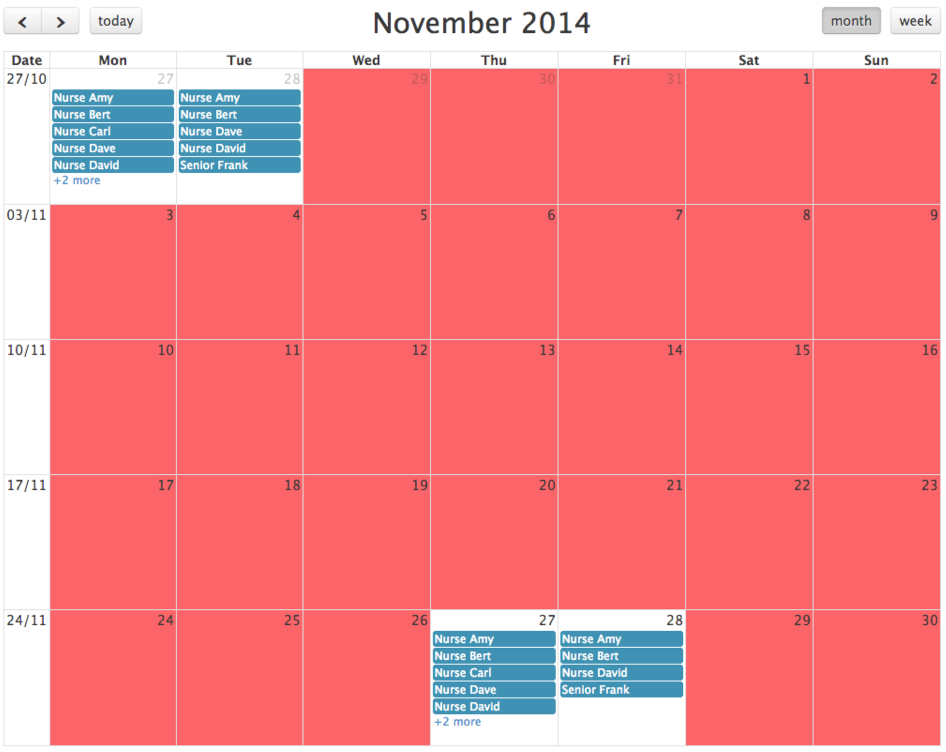


Image 1 – Understaffed shifts highlighted in red.

Whilst developing functionality for Fullcalendar, which uses moment.js for the dates, some expected functionality was uncovered. When assigning an existing moment date to a new variable using the standard assignment operator (=) a reference to the original moment was maintained. This caused issues that when the new data was modified; the original date was also modified, causing the calendar to have undesired functionality. The solution to this was to 'Clone' the original moment, Code Example 4, which creates a new moment with no reference to the original.

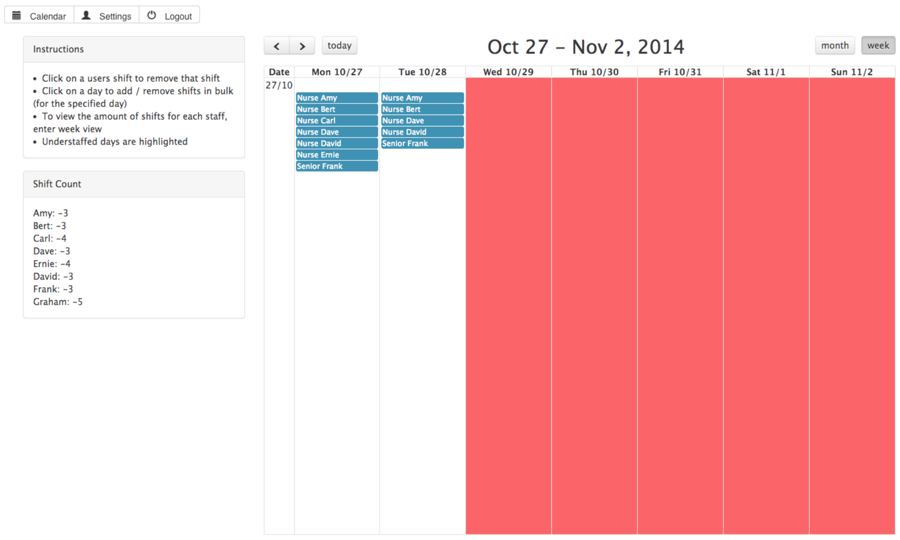
"nextWeek = moment(view.start);"
Code Example 4 – Cloning a moment

Ajax technologies have been used within the implementation of the calendar: when the user clicks on a day to add a shift, or clicks on a shift to remove it, an Ajax call is made to the server. The server then performs the server side validation, section 2.2.1, and informs the calling method if it has been successful or not.

FullCalendar does not come with the required view (4 weeks, commencing next week) 'out the box'. In order to achieve this functionality, it was necessary to create a custom view in the FullCalendar.js source code. Using the Month view as a template, and making modifications to achieve the desired functionality achieved this. The month view was left in tact, as this is still required, rather a new view was added. Appendix 2 shows the complete code for the custom view, whilst the entire code starts at line 7817 of the file fullcalendar.js.

2.3.2 Admin Functionality

When the user logs into the system, they are able to view specific instructions (for their staff level) alongside the calendar. This provides basic information into how the user should add and remove shifts. When logged in as admin, and viewing the calendar in week view, a list of each staff member and the amount of shits they are working, is displayed next to the calendar, as shown in Image 2.



Instructions

- Click on a users shift to remove that shift
- Click on a day to add / remove shifts in bulk (for the specified day)
- To view the amount of shifts for each staff, enter week view
- Understaffed days are highlighted

Shift Count

Amy: -3
Bert: -3
Carl: -4
Dave: -3
Ernie: -4
David: -3
Frank: -3
Graham: -5

Oct 27 – Nov 2, 2014

Date	Mon 10/27	Tue 10/28	Wed 10/29	Thu 10/30	Fri 10/31	Sat 11/1	Sun 11/2
27/10	Nurse Amy Nurse Bert Nurse Carl Nurse Dave Nurse David Nurse Ernie Senior Frank	Nurse Amy Nurse Bert Nurse Dave Nurse David Senior Frank	🟥	🟥	🟥	🟥	🟥

Image 2 – Admin week view interface

As per the specification, admin users are able to modify staff shifts. This is a simple process for admin users to take:

1. Click on a day they want to modify shifts for.
2. A menu is presented with all staff members on, those already working have a checked tick box, and those not working have an unchecked tick box, as shown in image 3
3. Checking a tick box next to a user will add that user to the day's shifts, unchecking a tick box will remove the selected user from that days shift.

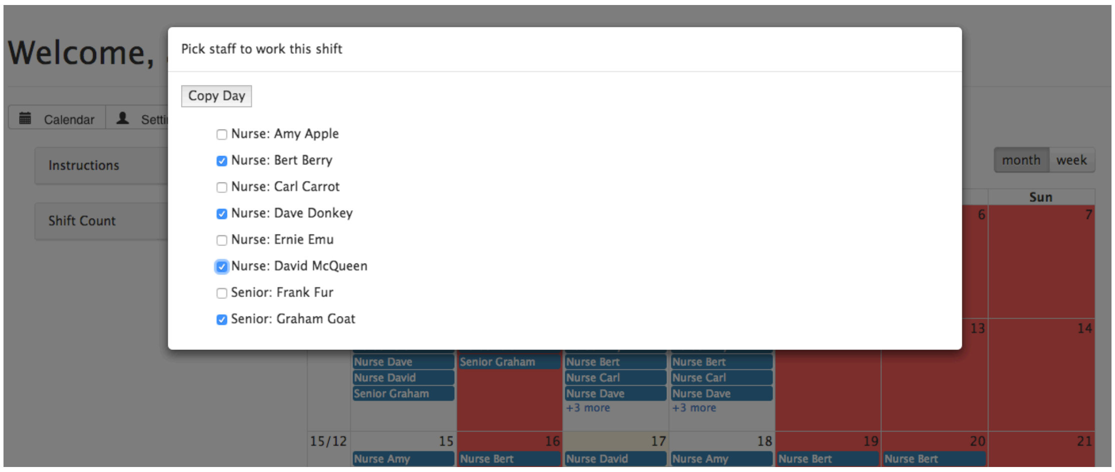
There is also the option for Admin to Copy & Paste specific days:

1. After steps 1 & 2 are completed above to display the menu with staff shifts on.
2. There is a Copy button, which will copy the shift setup for the current day
3. Upon clicking on another day, the admin user will be able to Paste the original day shifts; only, and all, of those staff that were on the original day will be working.

This allows for bulk management of shifts, allowing admin users to easily and quickly make modifications. This functionality is achieved through the use of Ajax, when a tick box is modified, at step 3, an Ajax process communicates this with the database and makes the necessary changes without having to reload the page, as show with code example 5.

```
if(element.checked){
    console.log('AddingShift');
    $.ajax({
      url: "<?php echo base_url(); ?>index.php/shift/addShift",
      dataType: 'json',
      data: {
        userID: userID,
        start: dayClicked.format()
      },
      success: function (result) {
        newShifts.push(userID);
        $('#calendar').fullCalendar('refetchEvents');
        countStaffShifts($('#calendar').fullCalendar('getView'));
        setTimeout(function () {
          highlightUnderstaffed($('#calendar').fullCalendar('getView'));
        }, 500);
      },
      error: function () {
        console.log("Something went wrong!");
      }
    });
  }
```

Code example 5 – Ajax functionality allowing admin users to add shifts for users.



Pick staff to work this shift

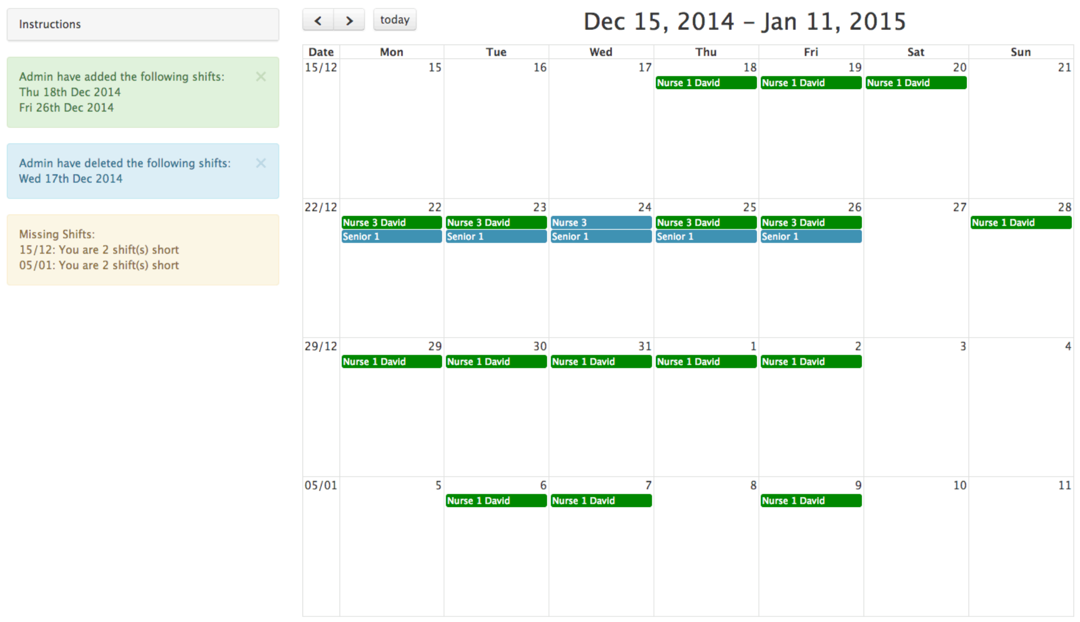
Copy Day

- [ ] Nurse: Amy Apple
- [✓] Nurse: Bert Berry
- [ ] Nurse: Carl Carrot
- [✓] Nurse: Dave Donkey
- [ ] Nurse: Ernie Emu
- [✓] Nurse: David McQueen
- [ ] Senior: Frank Fur
- [✓] Senior: Graham Goat

Image 3 – Menu allowing for admin users to bulk manage shifts

2.3.4 Standard User functionality

After admin have made modifications to the calendar, staff members are able to see which shifts have been modified via the messages section, as shown in Image 5. After reading this message, users are able to clear them, which prevents them from appearing again. The calendar is displayed as per the specification, the next 4 weeks are displayed, with the first being the next week, as shown in image 4. Users are able to modify their shifts by clicking on a day to add a new shift, or clicking on a current shift to remove that shift. These instructions are communicated to the user via the 'Instructions' box in the messages section, visible in image 4.
When the user adds or removes a shift, validation takes place to ensure that they are able to make those modifications, and is then communicated with the server via Ajax. If a user tries to make modifications to their shifts that would break the shift guidelines, then a warning message is displayed in the message section (left side of the calendar) and upon expanding the message the user is able to view the shift guidelines explaining the restrictions in place, as shown in image 6.
The user views the calendar as per the specification, displaying the amount of staff that is working and the staff level, on a specific day. If the current user is working, then their name is displayed next to it and the shift is also displayed in green, to make it clear to the user which days they are already working.


Instructions

Admin have added the following shifts:
Thu 18th Dec 2014
Fri 26th Dec 2014

Admin have deleted the following shifts:
Wed 17th Dec 2014

Missing Shifts:
15/12: You are 2 shift(s) short
05/01: You are 2 shift(s) short

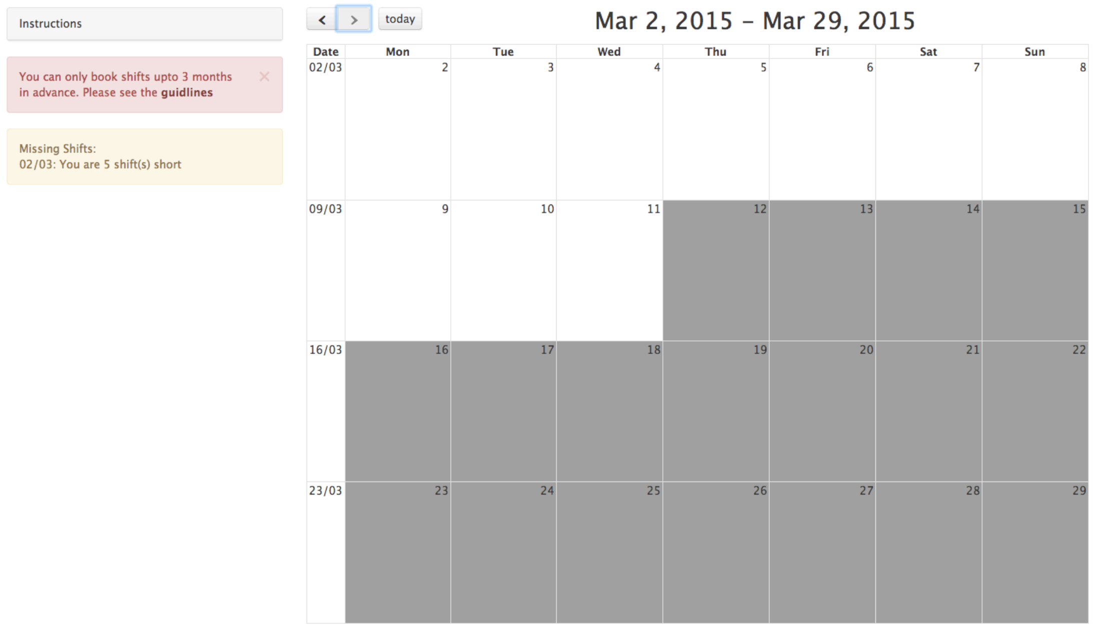
today

Dec 15, 2014 – Jan 11, 2015

Date	Mon	Tue	Wed	Thu	Fri	Sat	Sun
15/12	15	16	17	18 Nurse 1 David	19 Nurse 1 David	20 Nurse 1 David	21
22/12	22 Nurse 3 David Senior 1	23 Nurse 3 David Senior 1	24 Nurse 3 Senior 1	25 Nurse 3 David Senior 1	26 Nurse 3 David Senior 1	27	28 Nurse 1 David
29/12	29 Nurse 1 David	30 Nurse 1 David	31 Nurse 1 David	1 Nurse 1 David	2 Nurse 1 David	3	4
05/01	5	6 Nurse 1 David	7 Nurse 1 David	8	9 Nurse 1 David	10	11

Image 4 – The standard users interface, displaying messages next to the calendar with the next 4 weeks displayed.

If a standard user attempts to view more than 3 months in the future, from the current date, then they are prevented from doing do and a message is displayed explaining that they are unable to do so, Image 5.


Instructions

You can only book shifts upto 3 months in advance. Please see the **guidlines**

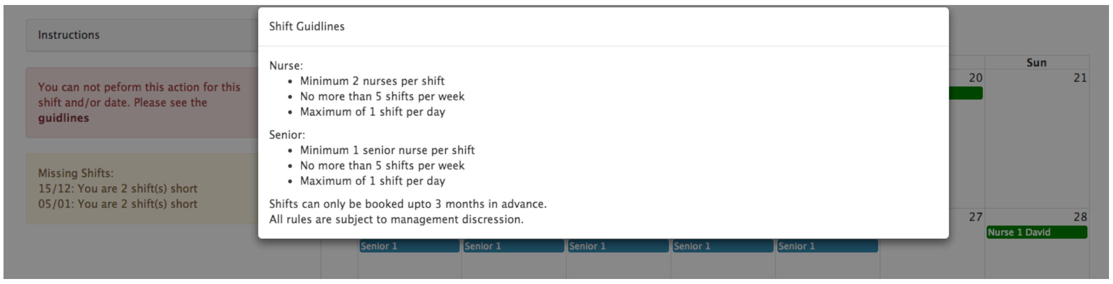
Missing Shifts:
02/03: You are 5 shift(s) short

today

Mar 2, 2015 – Mar 29, 2015

Date	Mon	Tue	Wed	Thu	Fri	Sat	Sun
02/03	2	3	4	5	6	7	8
09/03	9	10	11	12	13	14	15
16/03	16	17	18	19	20	21	22
23/03	23	24	25	26	27	28	29

Image 5 – Standard user, not being able to view more than 3 months in the future.


Instructions

You can not peform this action for this shift and/or date. Please see the **guidlines**

Missing Shifts:
15/12: You are 2 shift(s) short
05/01: You are 2 shift(s) short

Shift Guidlines

Nurse:
- Minimum 2 nurses per shift
- No more than 5 shifts per week
- Maximum of 1 shift per day

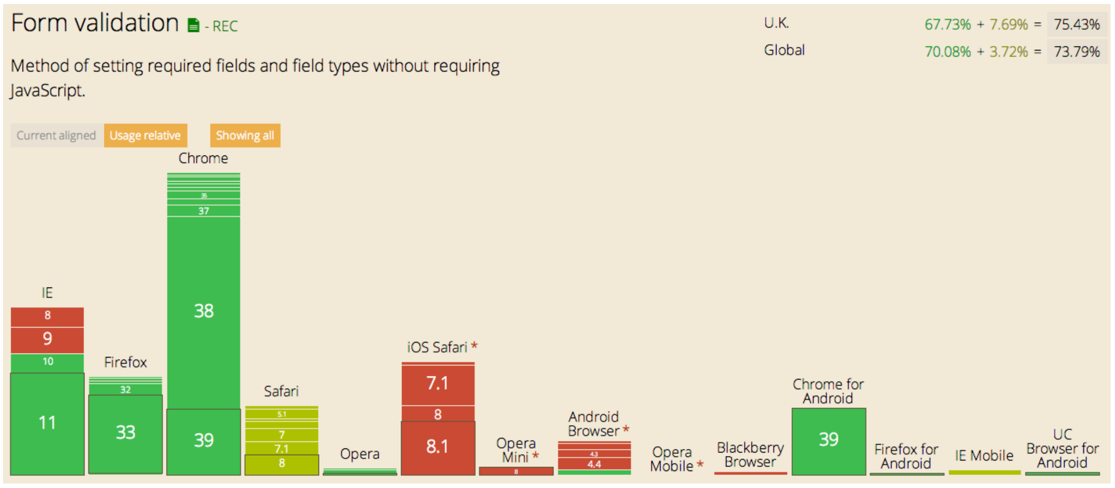
Senior:
- Minimum 1 senior nurse per shift
- No more than 5 shifts per week
- Maximum of 1 shift per day

Shifts can only be booked upto 3 months in advance.
All rules are subject to management discression.

Image 6 – The error message and shift guidelines popup displayed to the user when trying to modify shifts that would break the shift guidelines.

2.3.5 Client side validation

Further to the server side validation, chapter 2.2, client side validation is also implemented to provide the user with a better experience, preventing pages reloading when server side validation is failed and the user needs to re-enter all their information. This was achieved by using HTML5 form validation, which is supported by 73.79% Globally, and 75.43% UK, of browser versions used by Internet users (Can I Use, 2014), as show in figure 6. In addition to HTML form validation, JavaScript validation is also implemented in order to accommodate for the 26.21% of users that are unable to use HTML form validation. This validation is applied to the required inputs on the login form, and also the settings form, preventing the form from being submitted if the user has not entered the required information.


Form validation - REC

Method of setting required fields and field types without requiring JavaScript.

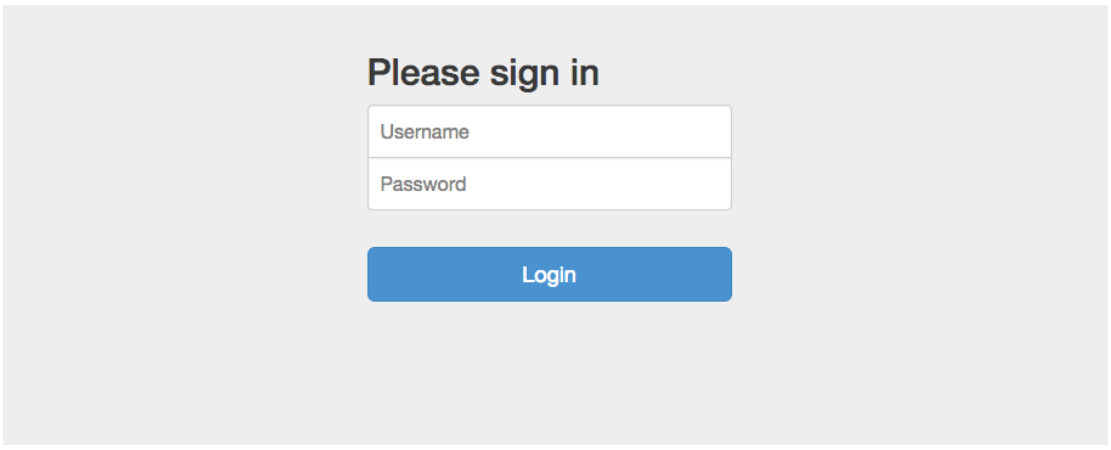
U.K. 67.73% + 7.69% = 75.43%

Global 70.08% + 3.72% = 73.79%

Figure 6 – Browser versions supporting Form Validation (Can I Use, 2014)

2.3.5.1 Login Form

The username and password inputs are validated to ensure that they have both being completed, and that the username input passes the required email format. As the user needs to enter their username in a specified format, it is essential that this be checked before sending the form to the server. However the password input is only checked that a password has been entered, and not that the password meets the password policy. This is for 2 reasons, 1) validating a password on the publicly available login form means that the regex pattern is publicly available. This could leave vulnerabilities as potential attackers would be know the password format, thus have a starting point for attack. 2) The user might have an old password that has been imported into the system, and doesn't meet the password policy. If this were to be validated on the login form, the user would be unable to access their account, however when creating a new password, the user is forced to conform to the new password policy, a covered in chapter 2.3.5.2. Image 7 shows the Login page.


Please sign in

Username

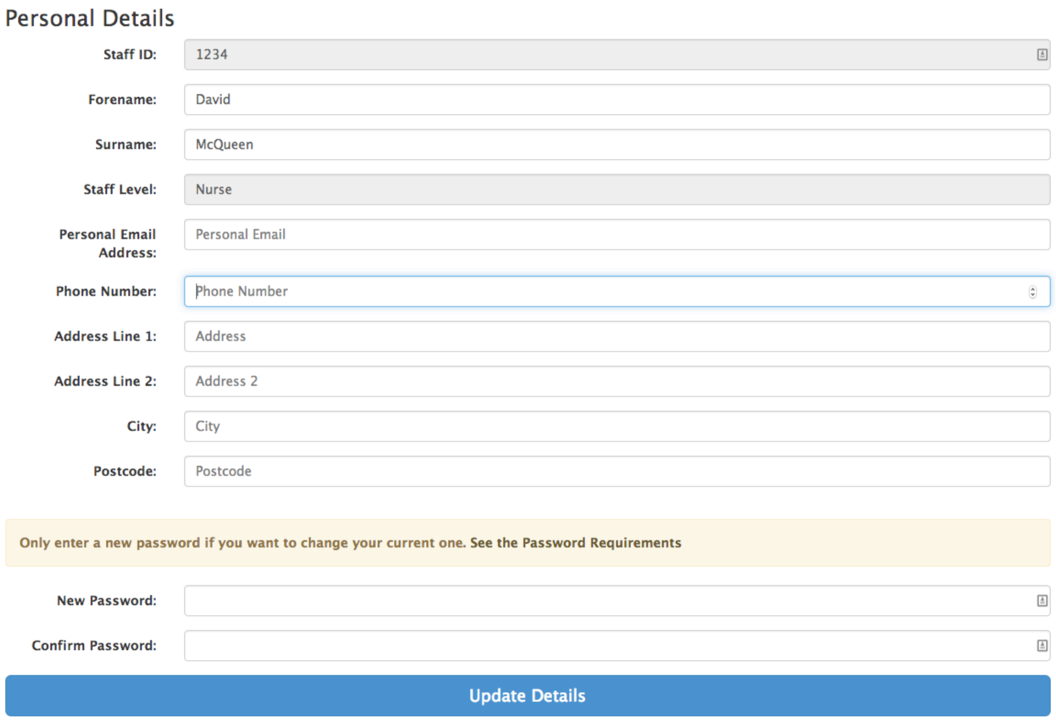
Password

Login

Image 7 – The login page

2.3.5.2 Settings Form

When the user is modifying their personal details, on the settings form, several inputs are validated. These are the required fields; Forename, Surname, and password. The rest of the information is not required, so is not validated beyond the HTML form validation checking that the input is the correct data type. The user is able to change their password on the settings page, and as such the password policy is enforced, unlike the login page (as discussed in chapter 2.3.5.1). However it is not necessary for the user to change their password when making other modifications to their personal details. As such, the password is only changed and the password validation is enforced, if the user fills in the new password field.
Users are also able to see their staff level, and their staff ID from the setting page, however they are unable to modify them. Image 8 shows the settings page


Personal Details

Staff ID:	1234
Forename:	David
Surname:	McQueen
Staff Level:	Nurse
Personal Email Address:	Personal Email
Phone Number:	Phone Number
Address Line 1:	Address
Address Line 2:	Address 2
City:	City
Postcode:	Postcode

Only enter a new password if you want to change your current one. See the Password Requirements

New Password:	
Confirm Password:	

Update Details

Image 8 – The user settings page

2.4 Files created

The website files that were created for the development of this system are as follows:

- Controllers/shift.php
- Controllers/user.php
- Controllers/verifyLogin.php
- Models/shift_model.php
- Models/user_model.php
- Views/css/templates/footer.css
- Views/css/templates/userBar.css
- Views/css/user/calendar.css
- Views/css/user/login.css
- Views/css/user/settings.css
- Views/templates/footer.php
- Views/templates/header.php
- Views/templates/userBar.php
- Views/user/calendar.php
- Views/user/calendarAdmin.php
- Views/user/calendarAdminScript.php
- Views/user/calendarStandardUser.php
- Views/user/login.php
- Views/user/settings.php

3. Testing & Evaluation

3.1 Test Plan

Throughout the implementation of the system, a test plan was created which was used for testing the system. The test plan (Appendix 1) consists of multiple tests, each covering a specific area or functionality of the system. Each test describes specific inputs, and expected outputs. With this test plan it is possible for multiple users to test the system to the same standards, and ensure that all bugs are identified and fixed. It also means that any future development can still use the same test plan, ensuring that bugs have not been introduced into pre-existing areas.

The test plan was created as implementation of the system progressed, to ensure that tests were created for each section of the system as the section was developed. Further to this, once development of the system was complete, the test plan was reviewed to ensure that the entire test covers in full the section of the system in which it was intended. The test plan is flexible and allows for more tests to be created, as more functionality needs to be tested.

"Test-driven development is an evolutionary approach to development which combines test-first development where you write a test before you write just enough production code to fulfil that test" (Agile Data, 2013). Test Driven Development (TDD) is usually accomplished with automated testing. However even without the use of automated testing the system was tested following the methodologies of Test Driven Development. Tests were created as development progressed, just before code was written for specific functionality.

Whilst it is possible to run automated tests in CodeIgniter, it was decided that these would not be used. This decision was mostly due to the fact that the majority of functionality is dependent on the data inside the database. As such automated tests would not be reliable, with tests passing in some situations and not others. Such as adding a new shift; if the user is not already working, then the shift will be added. However if the user is already working then the shift will not be added. The automated test would fail, however the code and functionality is working correctly.

If more server side logic were present, which was not dependent on data inside the database, then automated unit testing would be utilised to ensure that functionality works in the desired manor.

In replace of automated testing, manual testing was used to complete the test plan and ensure that everything was working as expected. This consisted of manually entering data into input boxes, to ensure that the validation was working correctly. It also involved manually adding and removing shifts for multiple members of staff, ensuring that all restrictions performed correctly, and also making modifications with the Admin user to multiple staff members shifts, ensuring that these persisted to the database correctly and the messages were relayed back to the staff member correctly. This method follows the principals of Black Box Testing, "*Black Box Testing is a software testing method in which the internal structure/design/implementation of the item being tested is not known to the tester*" (Software Testing Fundamentals, 2012). The test plan allows for the tests to be completed in full following the provided instructions, without the tester needing to know how the underlying system functions.

3.2 Evaluation

The system has been developed in a manor that allows for future expansion and possible integration into pre-existing systems. A popular data structure (MVC) has been implemented on the server, using a common framework that is openly available (CodeIgniter). Each core element of the server is segregated into its own area, ensuring each segment completes only the one specific task in which it is designed for, as discussed in chapter 2.2.
With this structure on the server it is easy for other developers to understand the process that is being taken at each stage, and it is easy for the developers to write their own code to enhance the system.

The database (chapter 2.1) makes extensive use of stored procedures, keeping SQL code away from the server to enhance maintainability. In addition to having all functional SQL code away from the server, stored procedures work similar to procedural functions in that a stored procedure can be called to perform a certain task and returns the relevant record sets. Each stored procedure is responsible for one job, and when this is required, the server calls the stored procedure without having to worry about the database structure. This itself also for expansion of the system, in that other systems can request data from the database without needed to know of the underlying structure.
Further to this, the SQL code developed in the stored procedures uses extensively SQL operations such as Join and Aggregate functions, ensuring efficiency in processing of SQL batches.

The frontend (chapter 2.3) uses a publicly available JavaScript library (FullCalendar) that allows for creation of custom calendars with extensive functionality. This was utilised to create a calendar that is easy to use for both the standard users (Nurses) and the admin users, which required different functionality. Further to this, Bootstrap was used to apply clean style to the system, and also to provide extra functionality in the form of Modals and the messages described in chapter 2.3.

Each area is developed in a method that means it doesn't need to know what the other areas of the system are doing. Simply that the task it has requested completes and returns either successfully, or returns an error. Doing so would allow for multiple developers to work on the development of the system, if expansion was required. Each developer could take a different area, such as 1 developer on the Frontend, 1 on the Server, and 1 on the Database, and each develop the functionality for their area.

3.2.1 Further Expansion

There are multiple expansion avenues that could be implemented in the system, a few examples are:

- An area where admin users are able to create and delete users from the system
- New staff levels added to the system, such as Junior Nurse

    - Minimal changes would need to be implemented as the database structure and some logic is already implemented to accommodate for this

References

Microsoft (2014) *Stored Procedures (Database Engine)* [Online]
http://msdn.microsoft.com/en-us/library/ms190782.aspx
[Accessed 24th November 2014]

Agile Data (2013) *Introduction to Test Driven Development* (TDD) [Online]
http://agiledata.org/essays/tdd.html
[Accessed 4th December 2014]

ISO (2014) *ISO 8601* [Online]
http://www.iso.org/iso/home/standards/iso8601.htm [Accessed 8th December 2014]

Can I Use (2014) *Can I Use Form Validation?* [Online]
http://caniuse.com/#feat=form-validation
[Accessed 8th December 2014]

Software Testing Fundamentals (2012) *Black Box Testing* [Online]
http://softwaretestingfundamentals.com/black-box-testing/ [Accessed 17th December 2014]

Bibliography

Bootstrap (2014) [Online] http://getbootstrap.com/
[Accessed 24th November 2014]

CodeIgniter (2014) [Online] http://www.codeigniter.com/community
[Accessed 24th November 2014]

Moment.JS (2014) [Online] http://momentjs.com/docs/
[Accessed 24th November 2014]

Microsoft (2006) *Guidelines for Test Driven Development* [Online]
http://msdn.microsoft.com/en-us/library/aa730844%28v=vs.80%29.aspx
[Access 4th December 2014]