

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.

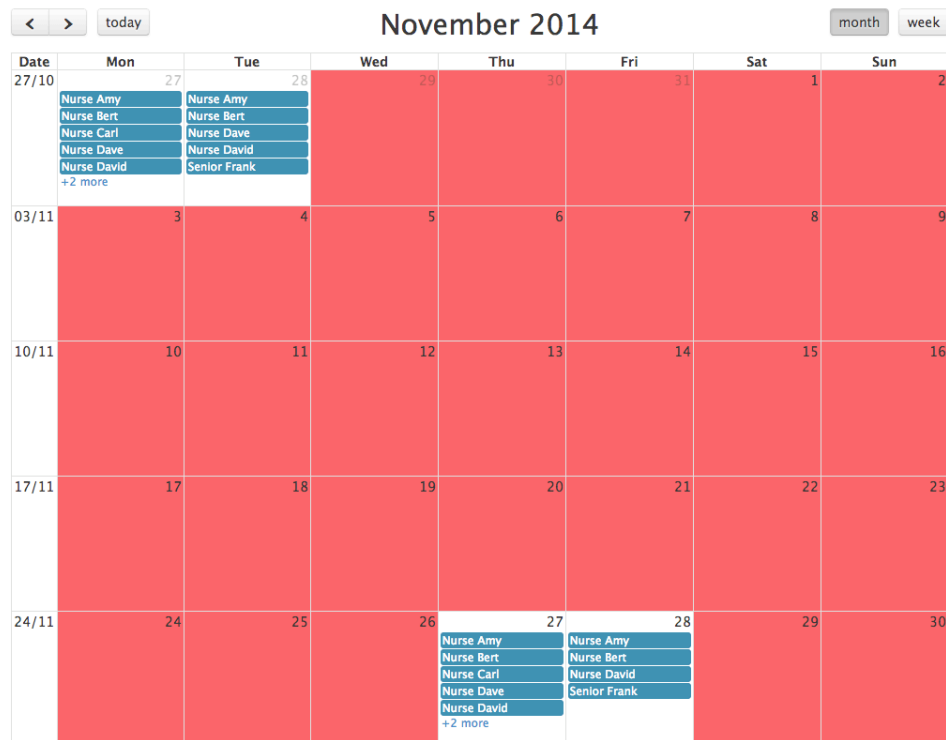


Image 1 – Understaffed shifts highlighted in red.

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 shifts they are working, is displayed next to the calendar, as shown in Image 2.

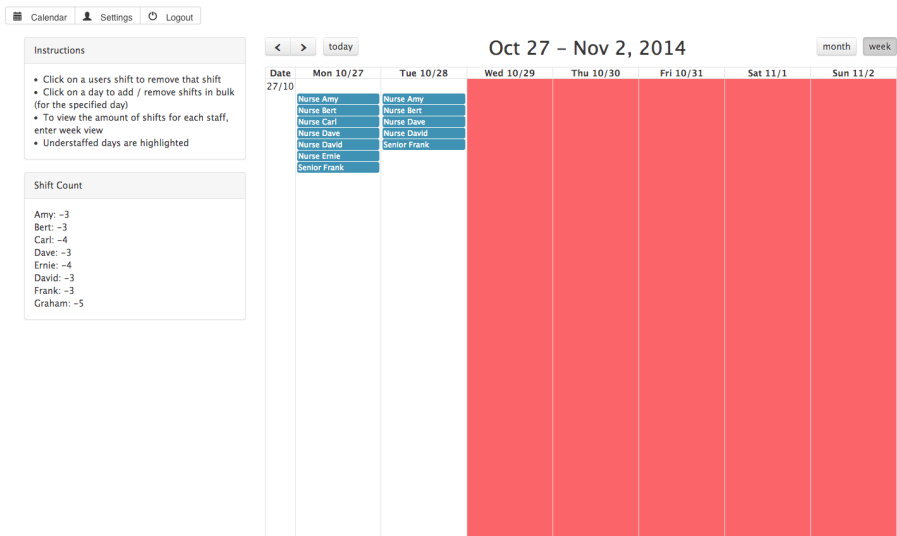


Image 2 – Admin week view interface

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.

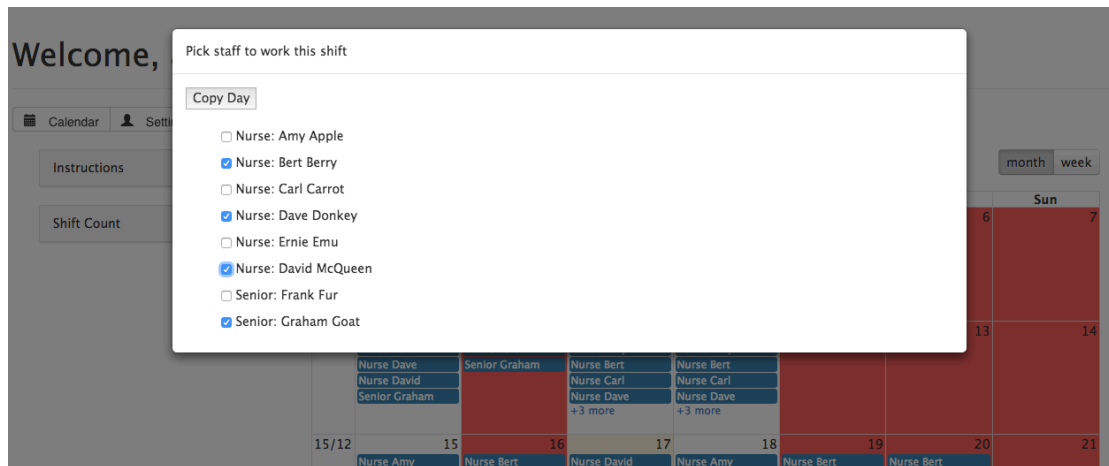


Image 3 – Menu allowing for admin users to bulk manage shifts

### 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.

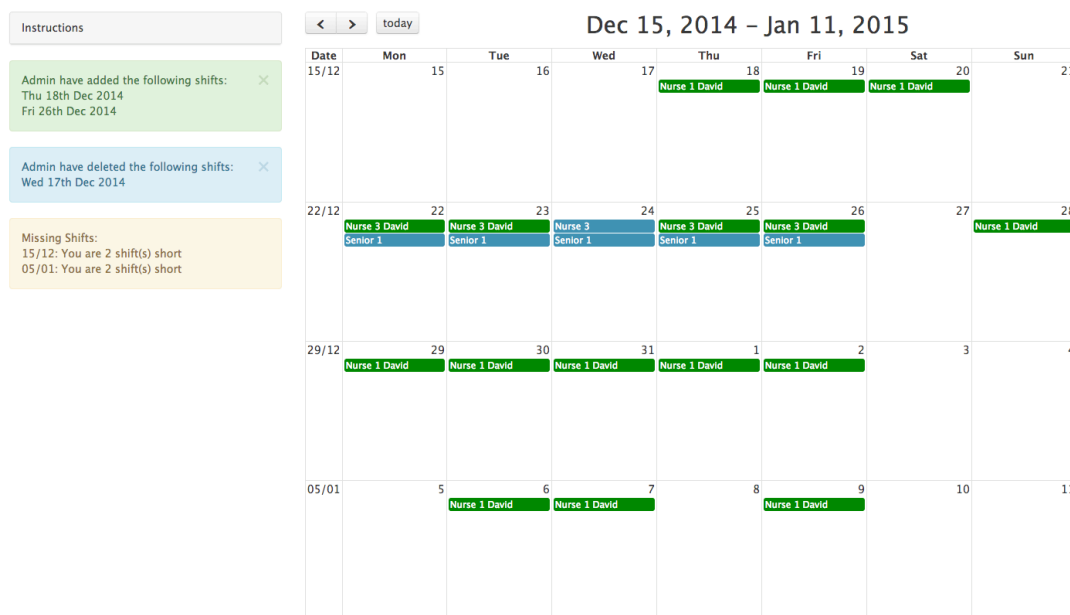


Image 4 – The standard users interface, displaying messages next to the calendar with the next 4 weeks displayed.

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

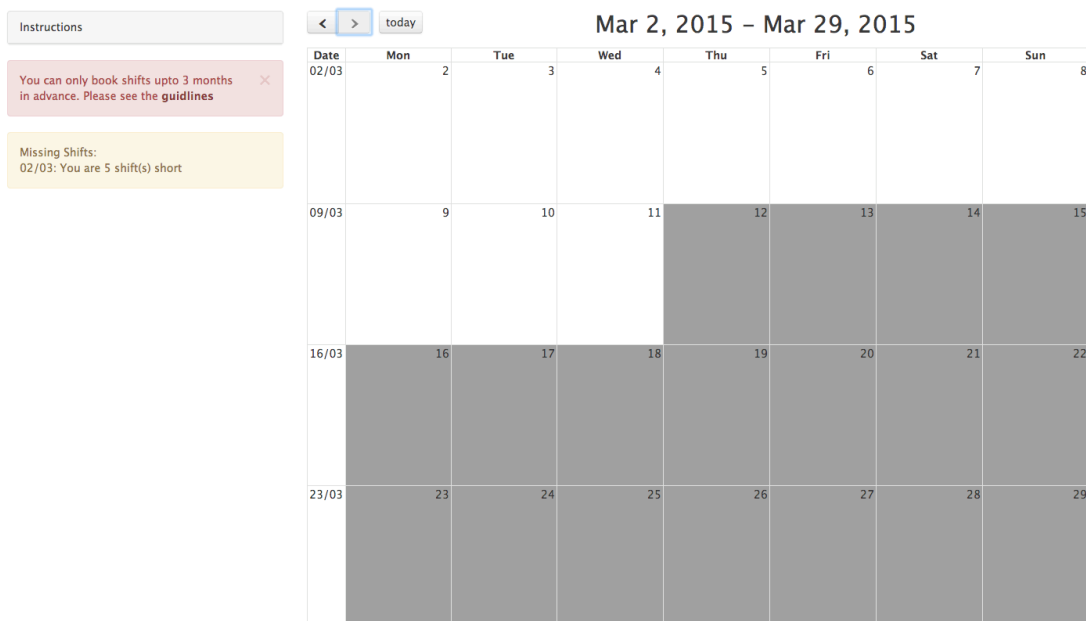


Image 5 – Standard user, not being able to view more than 3 months in the future.

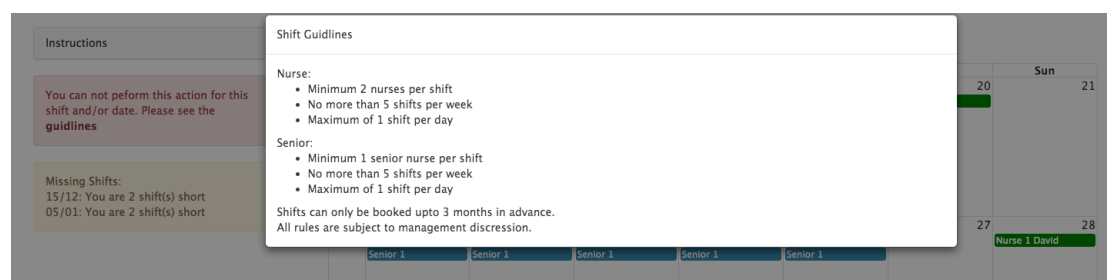


Image 6 – The error message and shift guidelines popup displayed to the user when trying to modify shifts that would break the shift guidelines.

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.

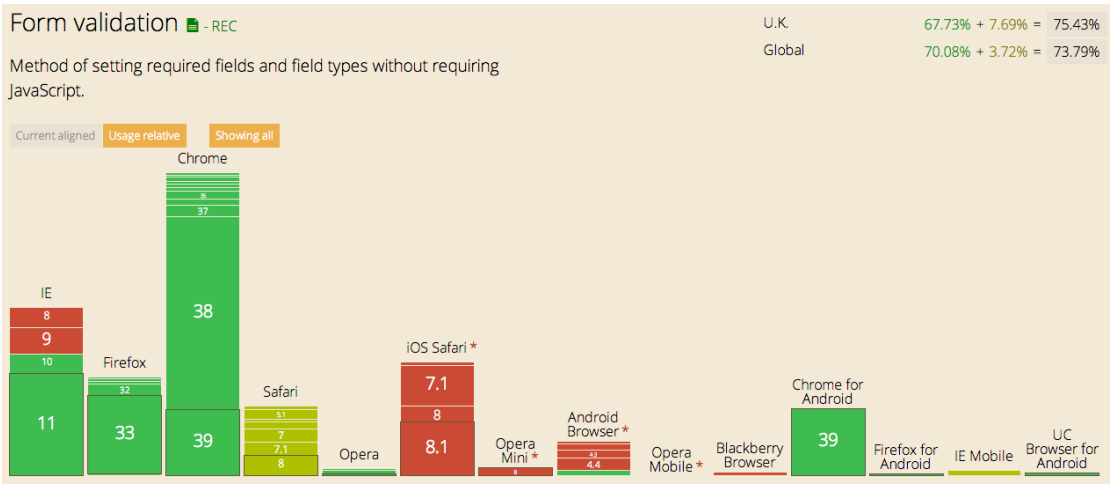
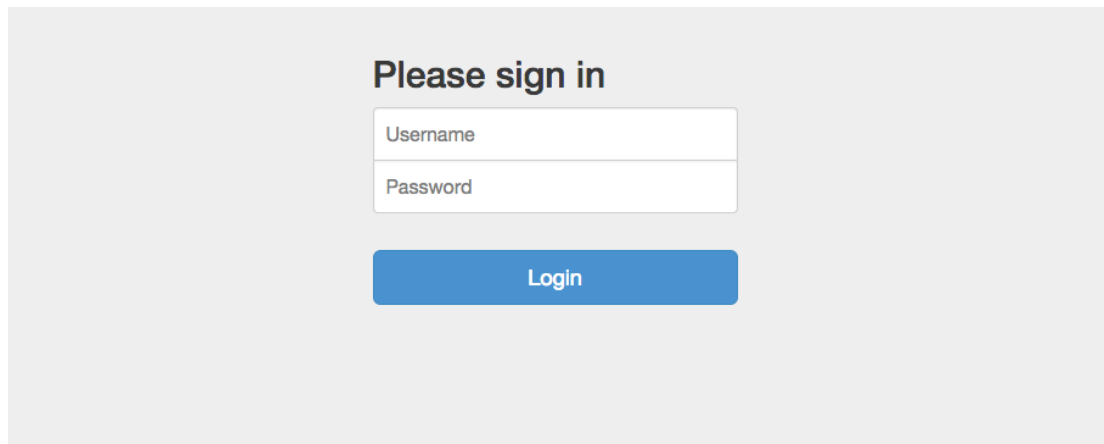


Figure 6 – Browser versions supporting Form Validation (Can I Use, 2014)

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

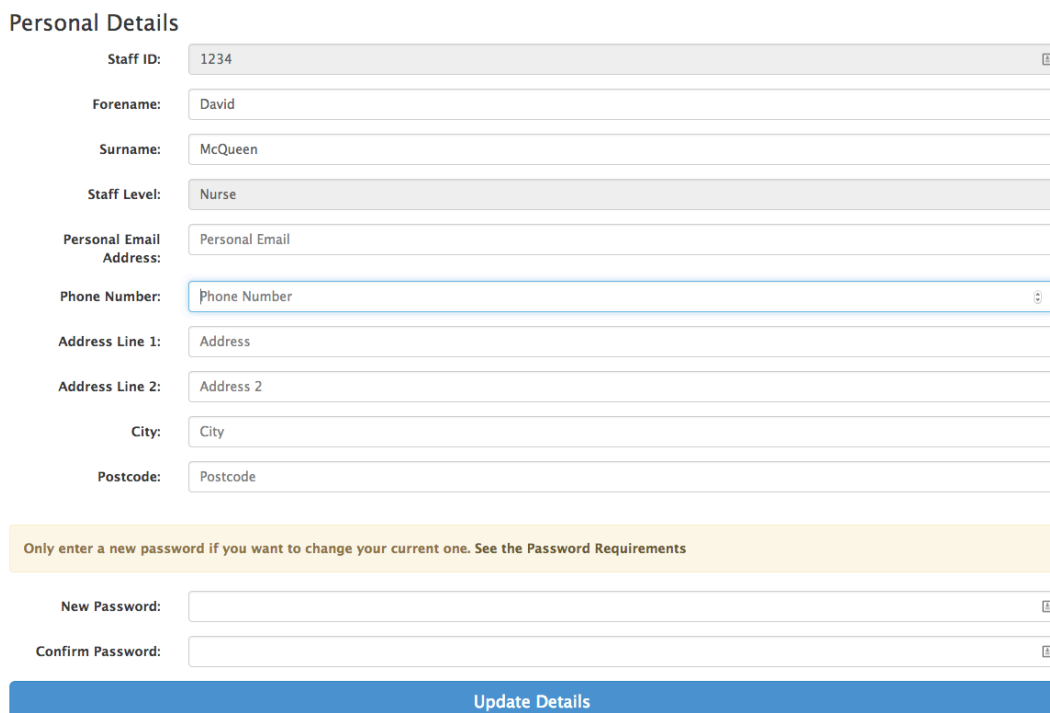
Login

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 24<sup>th</sup> November 2014]

Agile Data (2013) *Introduction to Test Driven Development (TDD)* [Online]

<http://agiledata.org/essays/tdd.html>

[Accessed 4<sup>th</sup> 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 8<sup>th</sup> December 2014]

Software Testing Fundamentals (2012) *Black Box Testing* [Online]

<http://softwaretestingfundamentals.com/black-box-testing/> [Accessed 17<sup>th</sup> December 2014]

## Bibliography

Bootstrap (2014) [Online] <http://getbootstrap.com/>

[Accessed 24<sup>th</sup> November 2014]

CodeIgniter (2014) [Online] <http://www.codeigniter.com/community>

[Accessed 24<sup>th</sup> November 2014]

Moment.js (2014) [Online] <http://momentjs.com/docs/>

[Accessed 24<sup>th</sup> November 2014]

Microsoft (2006) *Guidelines for Test Driven Development* [Online]

<http://msdn.microsoft.com/en-us/library/aa730844%28v=vs.80%29.aspx>

[Access 4<sup>th</sup> December 2014]