MyCare Records

Megan Patterson – B00582737

BSc Interactive Multimedia Design 2015

Peer Group: 2

Mentor: Dr Peter Nicholl

Acknowledgements

I would like to take this opportunity to thank everyone who has helped and supported me throughout the course of the MyCare Records project.

I would especially like to thank Dr. Peter Nicholl for all his help, guidance and support.

I would also like to thank Dr. George Moore and Mr. Christopher Murphy for their coaching and explanations.

I would like to extend a special thanks to Noel and Heather Elkin for proofreading this document, and to those who agreed to test the completed project and provide me with honest feedback, comments and suggestions.

The author

Table of Contents

1. Introduction

The MyCare Records system is a web-based application developed with the idea of allowing a patient to record readings relevant to the conditions they suffer from. The system aims to provide an interface, which allows the patient to not only make these readings, but to also present the recorded data in a clear and concise way. It is hoped that by doing this, patients will be able to better understand their conditions and potentially learn how to manage them more effectively.

Additionally, the system also provides healthcare professionals (HCP) with the readings made by their patients in an attempt to help them better assess the patient's progress between appointments.

It is hoped that the system will be able to act as the main reference point for a patient in regard to the various aspects surrounding their conditions. This includes elements such as their medication for each condition and appointments with healthcare professionals. It is also hoped that healthcare professionals will benefit from having this kind of information for all of their patients in one central location.

1.1 Aims & Objectives

The main aim of the project was to develop a system which allows patients and healthcare professionals to manage a patient's unique conditions effectively.

In order to create a system which could fulfill this aim, several objectives would need to be completed.

1. **Each user will have their own account for accessing the site.** This allows the content to be personalised to each individual user. A patient's account will differ in content from a HCP's account.
2. **The account will be split into different sections, or modules, for each condition.** There will be a general module which will appear on all accounts, and then only the modules applicable to the patient will be included in their account. An HCPs account will have modules for viewing patients, managing accounts and creating appointments.
3. **A patient will be able to record readings about the conditions they suffer from.** The types of readings will differ from condition to condition.
4. **Readings recorded by a patient will be displayed in a graphical format.** This will allow the patient to view their readings in a way which is easy to

understand. It will also help HCPs in identifying patterns in readings over a period of time.

5. **A patient's medication and appointments will be listed for each condition.** This will allow the patient to view all the information about their condition in one place.

The rest of this document outlines the processes carried out into the creation of the system. This includes aspects such as; idea generation, defining the requirements, initial paper prototyping, methodology selection, designing the UX and the system, creating the system and testing it, evaluating the system and the process used to create it, and finally drawing conclusions about the experience.

2. Concept Definition

This section describes how the concept for the MyCare Records system was defined. It outlines the idea generation process that was carried out to develop the concept of the MyCare Records system. The requirements, which the completed system must fulfill, are described to explain in more detail what the system needs to do. The paper prototyping process shows the thought process when designing the layout of the system. The feasibility of the project is discussed to show how thought was put into eliminating any potential risks and ensuring a key part of the system could be successfully created. Finally, the potential methodologies that could have been used to complete the creation of the system have been outlined and explained. The reason for the chosen methodology has also been made clear.

2.0.1 Naming Conventions and Terminology

There are several naming conventions and pieces of terminology which have been used, both in the creation of the system and its usage. These are defined below:

ECR: Electronic Care Record.
Module: a section of the system concerning one particular condition e.g. diabetes.
Tab: A sub-section within each module.
HCP: Healthcare Professional.

2.1 Idea Generation

An initial idea was presented to create an interactive system that will educate children about coping with diabetes (particularly type 1 diabetes). The system would allow users to

create their own account. They would then be taken through several different sections that would describe various aspects of coping with diabetes. There would be information and interactive sections allowing the children to educate themselves. Finally, there would be a section, which allowed the user to record their blood glucose readings. This would hopefully encourage children to try and take regular readings, and to balance them as much as possible. However, due to ethical issues with creating a system like this for children, the idea could not be continued.

However, the idea of creating a system to record readings was something that could be developed upon. Electronic Care Records (ECR) provide a digital version of a person's healthcare record (Harper, R. O'Loan, D. 2011). This means that all their medical information is stored in one central location. After looking into the implementation of the ECR, the concept of creating a recording system, which would link into the ECR was developed. This account would allow the patient to record readings about their conditions. These readings would then feed back into their ECR to be viewed by their HCPs. The idea was expanded upon further, by displaying the readings in a graphical format for the patient to view. This would potentially allow the patient to better understand their condition. By adding in information about the medication and appointments for each condition, the patient could be provided with a central hub to manage their conditions from.

2.2 Requirements Specification

The requirements can be split into two groups: functional requirements (i.e. things that the system must do), and non-functional requirements (dealing with issues such as usability, integrity, privacy and so on). Each requirement has its own unique number, a description, rationale describing the reason for the requirement, any dependencies the requirement has, and a priority number (1 being of low importance and 5 being of high importance). Several of the key requirements are listed below. All other requirements can be found in Appendix 1.

2.2.1 Functional Requirements

Requirement #: 1

Description: The system will have a login page

Rationale: This will allow the user to log into their individual account and view their own information (or in the case of the HCP, the information of the patients under their care)

Dependencies: None

Priority: 5

Requirement #: 4
Description: The system will allow an HCP to set up an account for a patient.
Rationale: This will involve choosing a patient, setting up the modules that should appear in the account by choosing the condition (or conditions) the patient suffers from, and adding information about the medications prescribed to the patient.
Dependencies: None
Priority: 5

Requirement #: 5
Description: Each patient's account will be divided into modules.
Rationale: These modules will split up all the information into the various conditions suffered by the patient. This will help to keep all readings, appointments etc. organized.
Dependencies: None
Priority: 5

Requirement #: 8
Description: A user will be able to record readings for a particular condition.
Rationale: In order to help keep track of their condition, the patient will be able to log into their account and record readings they take on a regular basis. For example, if the user suffered from diabetes, they would need to take readings of their blood glucose level several times a day.
Dependencies: #5, #6
Priority: 5

Requirement #: 9
Description: The readings made by a patient will be displayed in a graphical format.
Rationale: In order to help the patient better understand the readings they are taking, all readings should be displayed as a graph or chart. This will help them to tell if their readings are within an acceptable range.
Dependencies: #7
Priority: 5

2.2.2 Non-Functional Requirements

The non-functional requirements cover areas not directly linked with the way the system works. These requirements are concerned with the usability, accuracy and overall integrity

of the system. Below is an example of an access requirement. All other non-functional requirements can be viewed in Appendix 1.

Requirement #: 15

Description: Patients will not be allowed access to their full ECR

Rationale: Due to data protection issues, the patient account will only allow the patient to record and view readings, and view appointments and medication for each module.

Dependencies: None

Priority: 5

2.3 Paper Prototyping

The paper prototyping stage was used to explore the various layouts which could be used within the system. This started with creating some initial sketches of layout ideas, creating a 6-up of the graphical output the patient would view, before developing a final prototype for the system layout.

2.3.1 Initial Sketches

Several sketches were drawn to show the layouts of the various pages within the system.

The Fig. 2.1 shows the generic layout for all the pages. It highlights how the conditions are split into separate modules and how tabs are used within each module to display the various sub-sections. This layout would apply to both patient and HCP accounts.

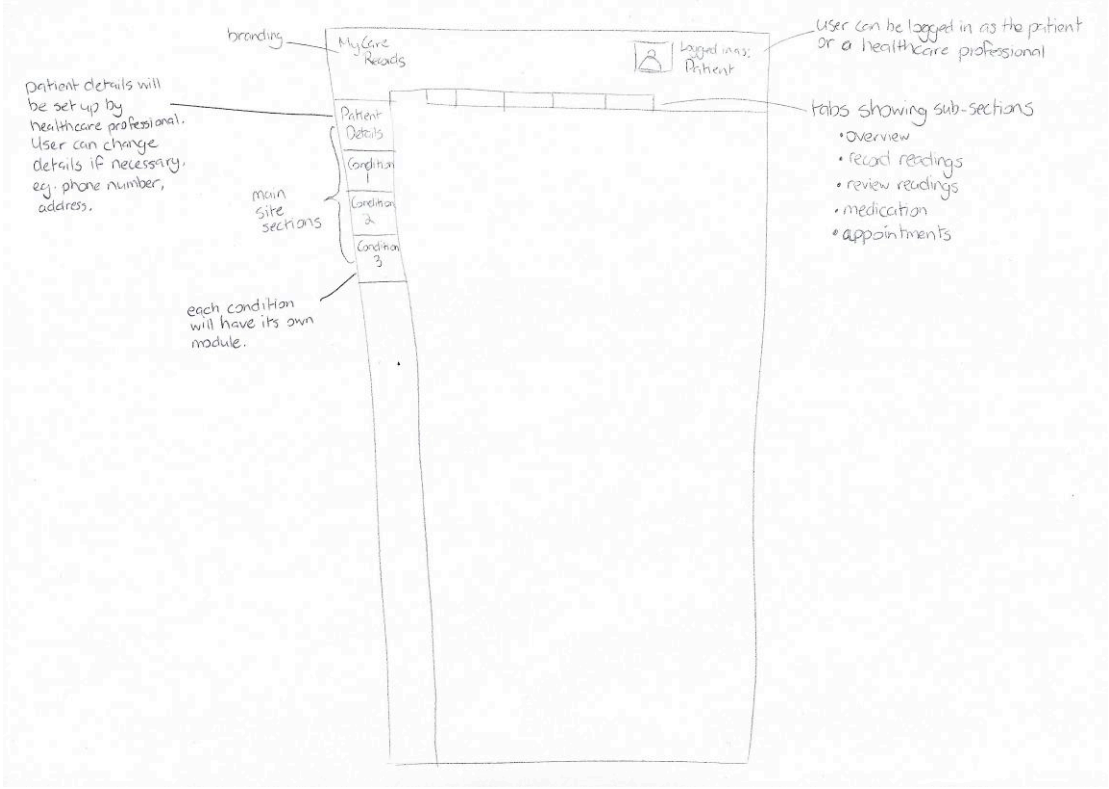


Fig. 2.1

Fig. 2.2 – Fig. 2.4 show how consideration was given to the layout of the various sub-sections within a patient account.

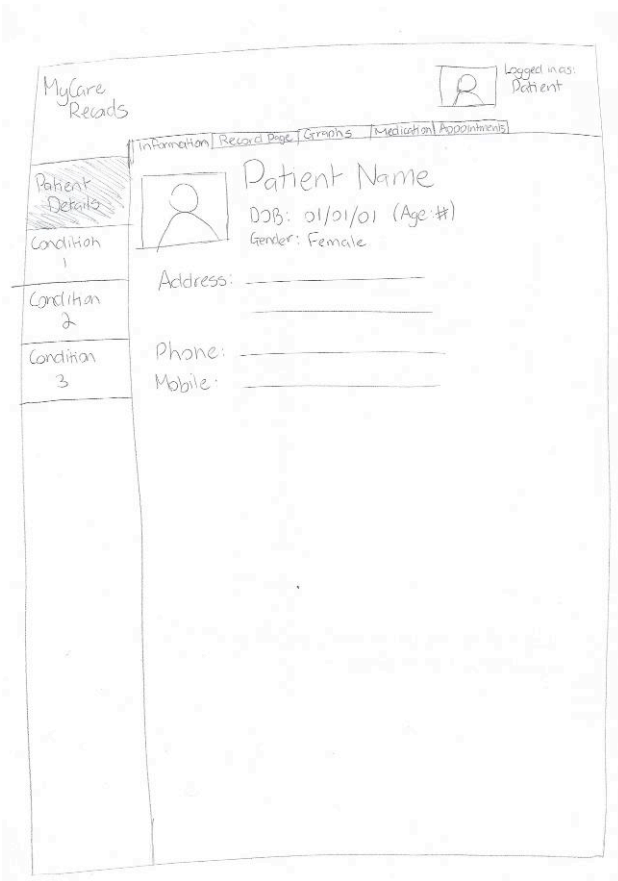


Fig. 2.2

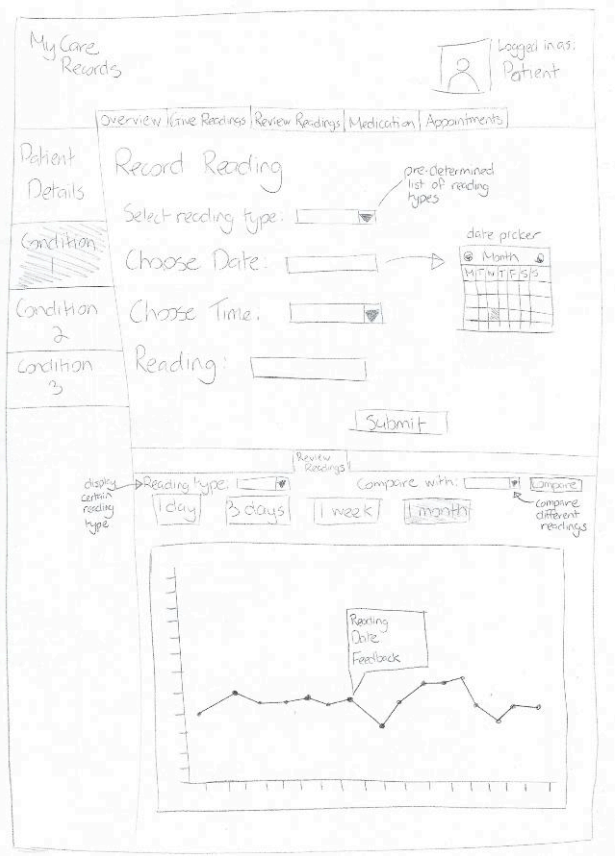


Fig. 2.3

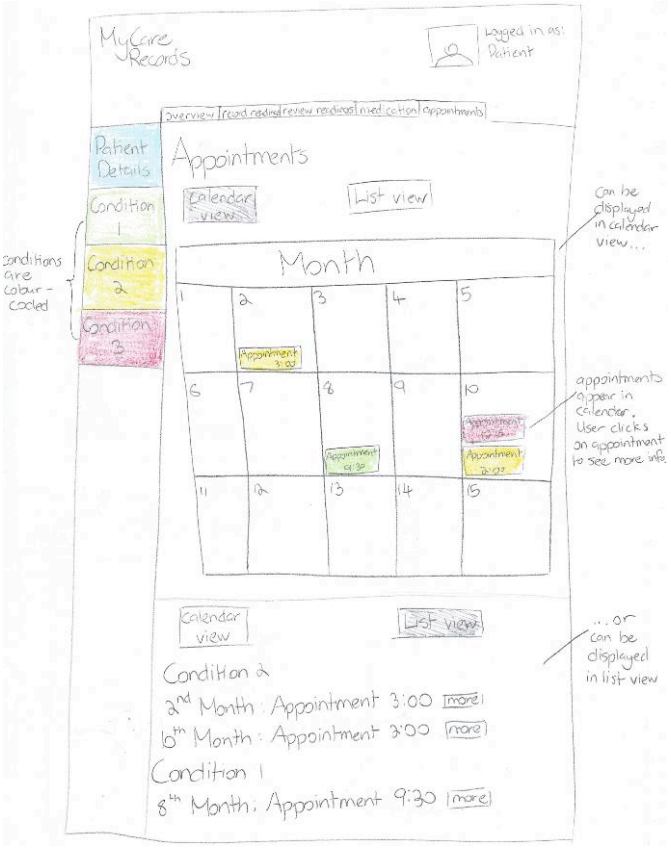


Fig. 2.4

The HCP account layout was also considered. Fig. 2.5 shows the layout of the Add Patient page.

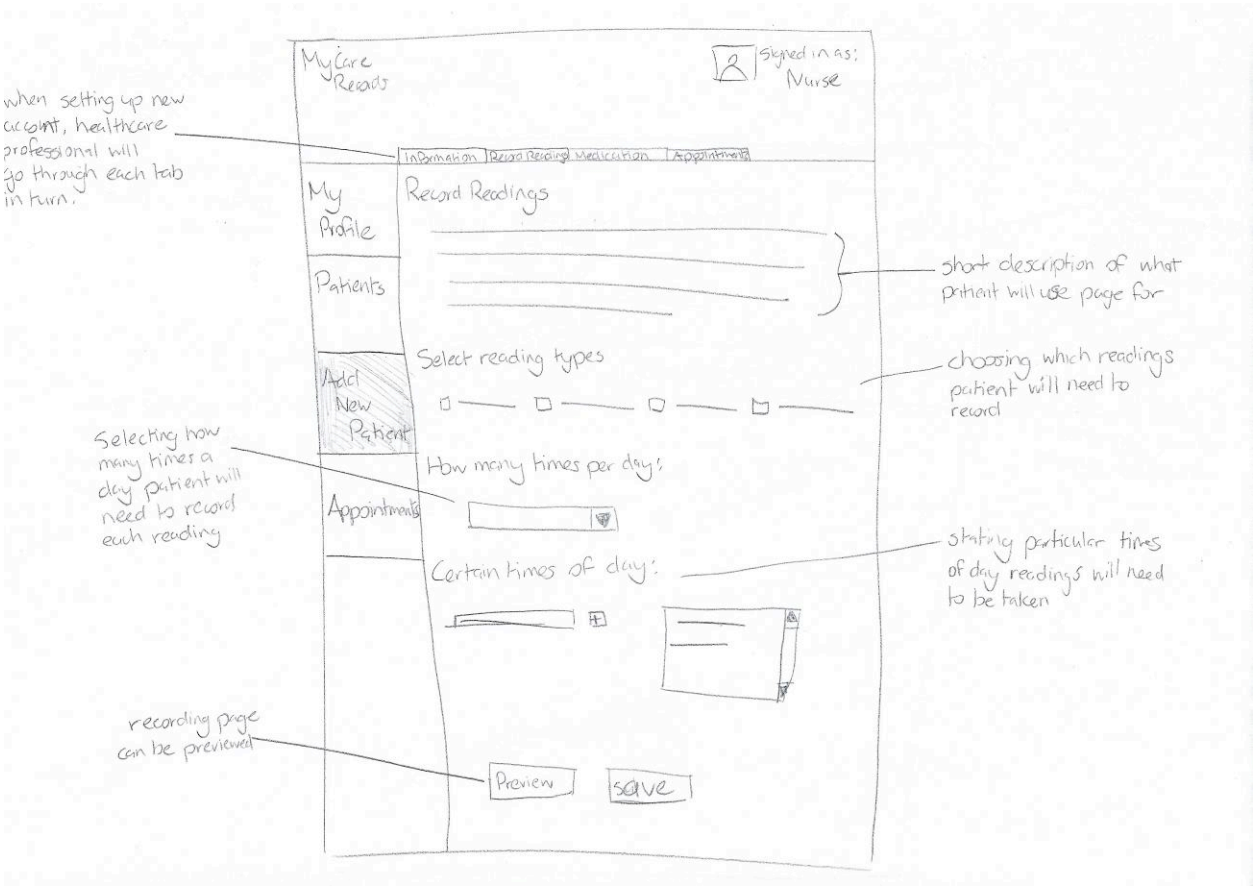


Fig. 2.5

As the graphical output was one of the key features of the MyCare Records system, it was important to pay considerable attention to its layout. Fig. 2.6 shows a 6-up which was created to experiment with different layouts. Fig. 2.7 shows the chosen design as a final prototype. All other prototyping material can be viewed in Appendix 2.

It was interesting to note that the completed project still stuck to these original designs, with only little variation on some of the pages. This variation is particularly evident in the HCP account, where the creation of a new patient account changed considerably. However, the entire project still maintained the layout set in Fig. 2.2.


MyCare Records
Review Readings
Reading type:
Reading Type
1 day
3 days
1 week
Review Readings
Reading type:
2nd October 7:30am
Reading 5.8
Feedback: Too low, reduce insulin intake
2nd October 12:15pm
Reading 7.1
Feedback: Normal reading
Review Readings
Date:
Reading Type:
Time
Reading
Feedback
7:30am 5.8
12:15pm 7.1
4:30pm 6.2
7:40pm 10.3
Review Readings
Reading Type:
1st October
2nd October
7:30am 5.8
Too low, reduce insulin
12:15pm 7.1
Normal Reading
3rd October
4th October
Reading 1: too low 5.8
Reading 2: 7.1
Reading 3: 6.2
Reading 4: dangerous level 10.3

Fig. 2.6

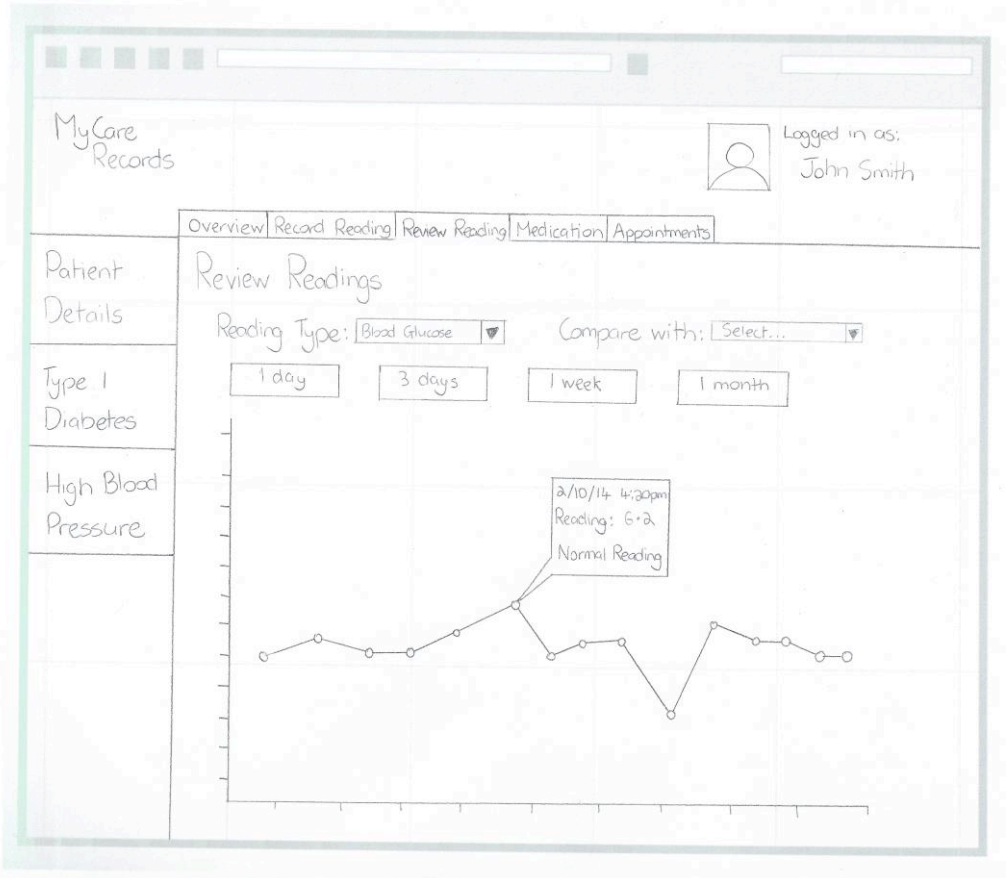

MyCare Records
Logged in as:
John Smith
Overview
Record Reading
Review Reading
Medication
Appointments
Patient Details
Type 1 Diabetes
High Blood Pressure
Review Readings
Reading Type: Blood Glucose
Compare with: Select...
1 day
3 days
1 week
1 month
2/10/14 4:30pm
Reading: 6.2
Normal Reading

Fig. 2.7

2.4 Feasibility Testing

In order to ensure the project would be feasible, several areas were investigated. A risk analysis was carried out to plan for any potential problems. Finally, a functional prototype of a key aspect of the project was created.

2.4.3 Risk Analysis

A risk analysis was carried out in order to investigate any potential threats to the completion of the project as outlined in Table 1. Each was given a level of significance, i.e. how much this risk could impact the success of the project (with 1 being low impact, and 5 being high impact). A possible solution to combating the risk was suggested, as well as a contingency plan in the event that the risk could not be resolved.

Table 1

Risk	Level	Possible solution	Contingency plan
Construction of account layouts	4	The jQuery.load() function can be used to load separate HTML pages into a container on the page. These can be activated by clicking links on the page.	Using sets of horizontal tabs within a set of vertical tabs could create the desired layout.
Creating graphical outputs	5	AmCharts, a JavaScript library, would be ideal for the creation of any graphical outputs needed within the system, as it allows for the creation of many different types of interactive graphs.	There are other JavaScript libraries which could have been used to generate static, non-interactive graphs and charts.
Construction of patient accounts by HCP	5	Creating a form on a single page, which the HCP can fill out to create an account and information for one specific condition. Other HCPs could then add or modify conditions on an existing account.	It may be necessary to create the account in stages using a multi-page approach.

2.4.4 Functional Prototype

In order to test whether the risks outlined in the risk analysis could be overcome, a functional prototype was created in attempt to solve one. As it would be one of the key features of the system, the challenge of producing a graphical representation of a patient's readings was tested. This involved creating a simple database of reading values and attempting to produce some kind of interactive graph. Not only that, but a method needed to be developed to allow the user to view different kinds of readings. A simple table was created within the database to hold a series of dates, values and reading types (Fig. 2.10).

reading	reading_time	reading_type
12.7	2015-01-01 08:01:12	blood_glucose
8.2	2015-01-01 08:36:17	insulin_intake
12.2	2015-01-01 12:02:21	insulin_intake
8.5	2015-01-01 12:20:39	blood_glucose
8.6	2015-01-01 17:34:30	blood_glucose
11.1	2015-01-02 07:31:49	blood_glucose
8.3	2015-01-02 08:09:31	insulin_intake
14.7	2015-01-02 13:13:34	blood_glucose
12.5	2015-01-02 13:41:13	insulin_intake
17.0	2015-01-02 17:57:49	insulin_intake
11.1	2015-01-03 08:32:16	blood_glucose

Fig. 2.10

The AM Charts JavaScript library was to be used in the creation of the graphs. Many examples of the charts and graphs which can be created with the library are present on their website, along with the source code. However, the examples all used hardcoded JSON to plot the data points on their graphs. The MyCare Records system would require taking the values from a database. In order to do this, an AJAX call was used to retrieve the necessary data as JSON, and use it with the library.

The issue of selecting a reading type was solved using a select box on the HTML page. Each option within the select box was given a value. The onchange() method was used to pass the value of the chosen option to a function within the jQuery. This value was used in the AJAX call to fetch the reading information for a particular reading type. The resulting JSON was then passed in as the data source for the graph. This resulted in a line graph plotting the various data points. When the reading type was changed, the graph automatically updated (Fig. 2.11).

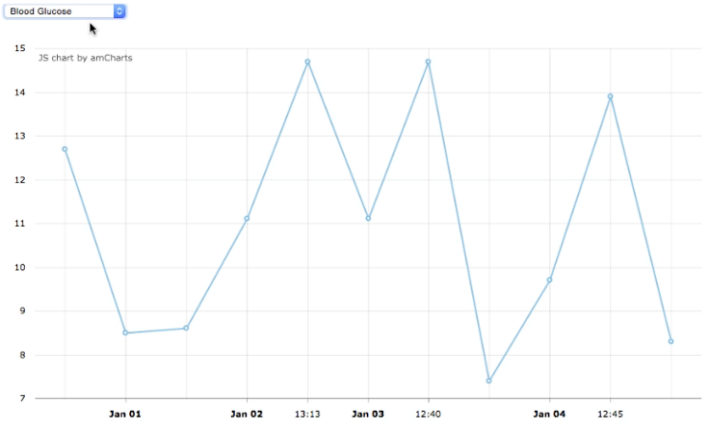


Fig. 2.11

The creation of this prototype proved that one of the main risks to the project could be overcome, meaning that the entire project was feasible and could be developed further.

2.5 Methodology Selection

In order for the project to meet its aim, a project management methodology was used. There were several methodologies to choose from. The three main choices are described below.

1. Waterfall/Modified Waterfall

Waterfall is a sequential, step-by-step methodology, which moves from one stage of the process to the next in a logical order. Starting from the initial ideas for the project, the model moves through the design, construction and testing processes. The model ensures each stage is complete before progressing to the next one. However, this can present a problem if something needs to be changed. The Modified Waterfall model simply allows for movement back through the various stages if necessary. This has the advantage of allowing the stages to overlap if required, meaning if circumstances change, modifications could be easily made.

2. Prototyping

The prototyping methodology consists of creating a series of prototypes, testing them, modifying them as necessary, and retesting the new prototype. This cycle continues until it is felt the product has reached a satisfactory level of quality. This model directly involves users in the development process and can easily identify any missing functionality. The system can however, result in a product which is never truly completed and can end up becoming overly complicated.

3. Agile

Agile is a methodology, which involves splitting a project into several sprints or iterations. At the end of each of these iterations, a minimum viable product will have been produced. This is a complete working prototype with limited functionality. Each iteration builds on the previous one, resulting in a completed product after several iterations. This provides the ability for adapting to new circumstances throughout the project life cycle. This methodology is not very suitable for single-developer projects.

Chosen Methodology

The methodology that was chosen for this project was the modified waterfall. This was mainly due to the structured step-by-step approach the methodology allows. The Modified Waterfall approach was chosen over the standard Waterfall due to its flexibility with returning to previous stages in order to make modifications. As this project was a single-

developer project, it was often necessary to return to the design stage to make minor modifications.

3. Design

In order to ensure that the system would fully meet all of its requirements, the system needed to be carefully designed. This involved looking into the visual design in more depth, as well as looking at which system architecture would be used and how the various technologies would fit into it.

3.1 UX Design Evolution

Working on from the initial sketches of the paper prototyping, the User Experience (UX) design of the system was considered. This was carried out in two stages; the initial UX design, which was built upon the paper prototypes, and the refined UX design, which examined the UX in greater detail. The design transition between the two stages is shown below and has been divided into the patient account and the HCP account. All additional UX design plans can be found in Appendix 2.

3.1.1 Patient Account

The UX design for the patient account dealt with the look of each sub-section within the condition module. These would be displayed in a tabular format within the main container of the page. The initial design showed that each condition would have an overview page, detailing some information about that condition. However, it was decided during the refinement stage to remove this, as it was felt the page did not add to the user's overall experience (Fig. 3.1).

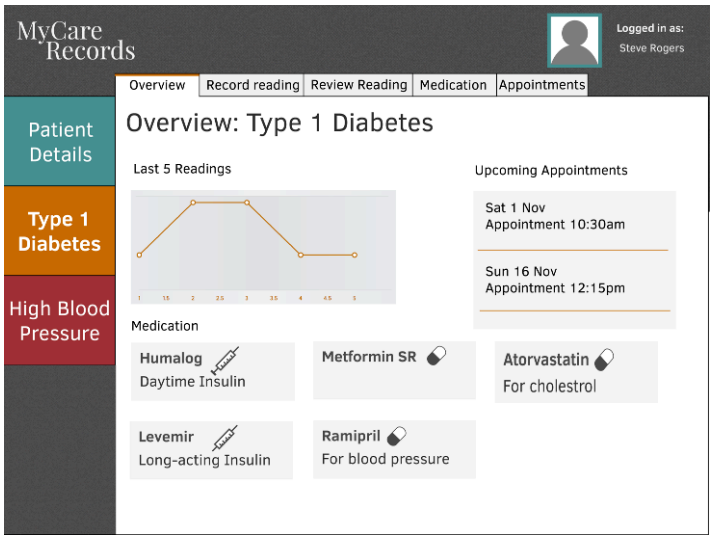


Fig. 3.1

The initial design for reviewing readings featured buttons, allowing the user to change the period of time they were viewing, by using these. This was removed as the AM Charts library used to create the functional prototype allows the user to pan up and down the graph and to alter the timeframe being viewed. The 'Compare with…' drop down was also removed, as many of the readings taken could not be compared effectively on one single axis (Fig. 3.2 and Fig. 3.3).

Fig. 3.2

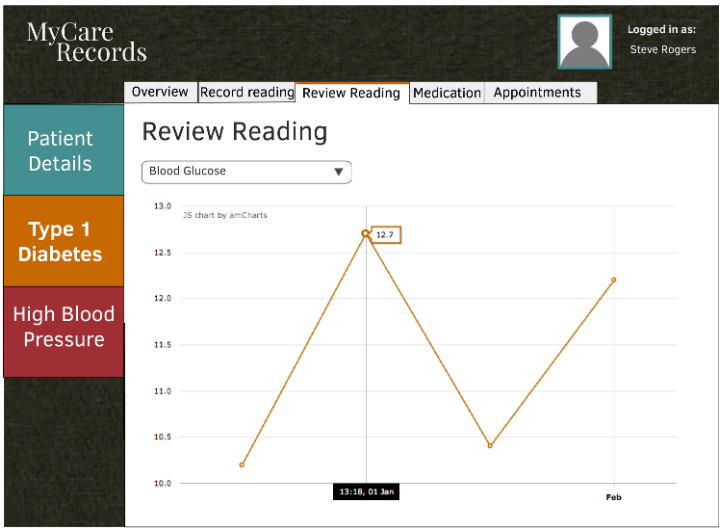


Fig. 3.3

The layout of the medication page was changed slightly from the initial design. This was done to simplify the layout, as well as simplifying the database structure. Rather than displaying all the times the medication needed to be taken, a single statement was displayed showing the dosage and frequency in which the medication needs to be administered. The medication is colour-coded to correspond with the various conditions, and small icons have been used to indicate the way each medication is administered (Fig. 3.4 and Fig. 3.5).

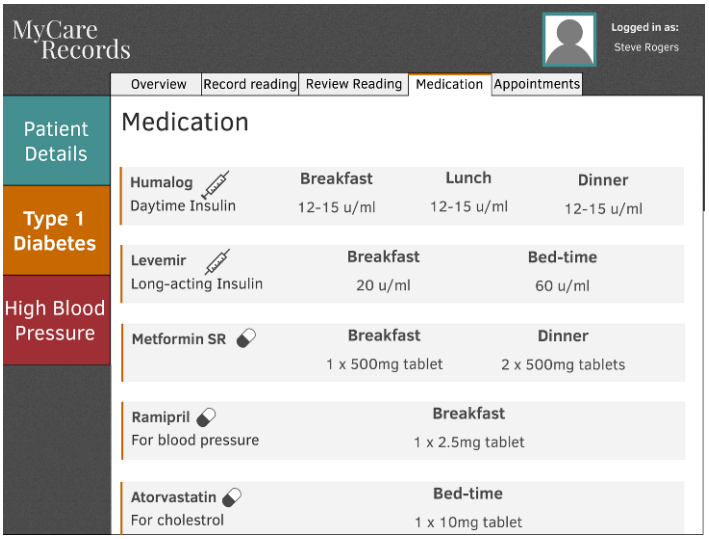


Fig. 3.4

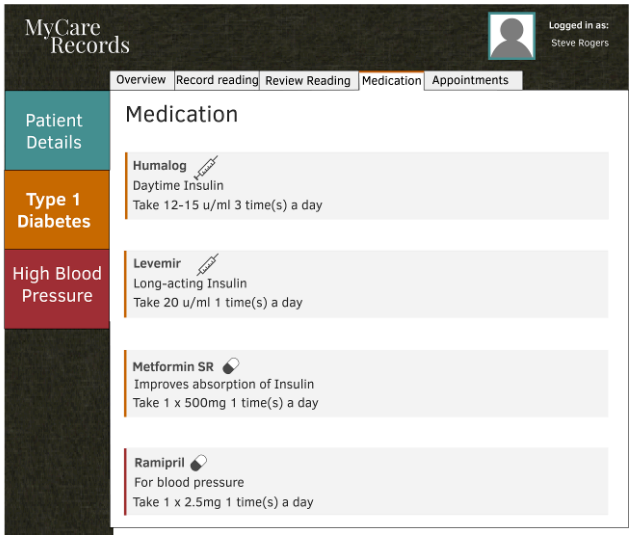


Fig. 3.5

3.1.2 HCP Account

The biggest change from the initial UX design to the final UX design for the HCP account, was the ‘Add Patient’ form.

Initially, the form was designed to be split into four sections. The HCP would select which patient they wished to create an account for by pulling the information from the database (in a real-world situation, the data would be pulled from the patient’s ECR) (Fig. 3.6). They would then specify the conditions they wanted to add to that patient’s account (Fig. 3.7). From there, they could edit the details of that condition; the readings the patient would take, and details about the medication being administered to them (Fig. 3.8). Finally, they would be able to preview the account before approving it (Fig. 3.9).

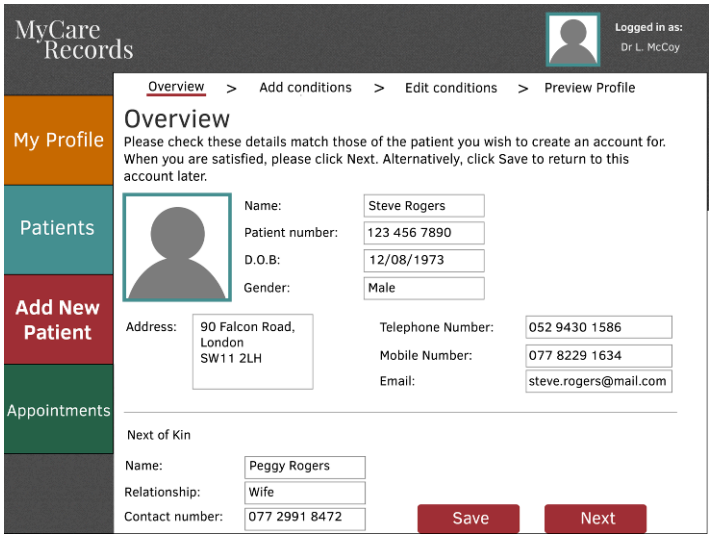


Fig. 3.6

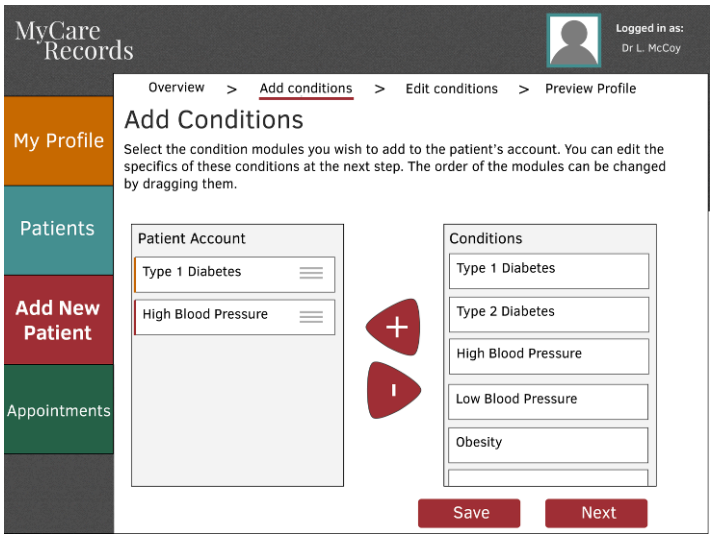


Fig. 3.7

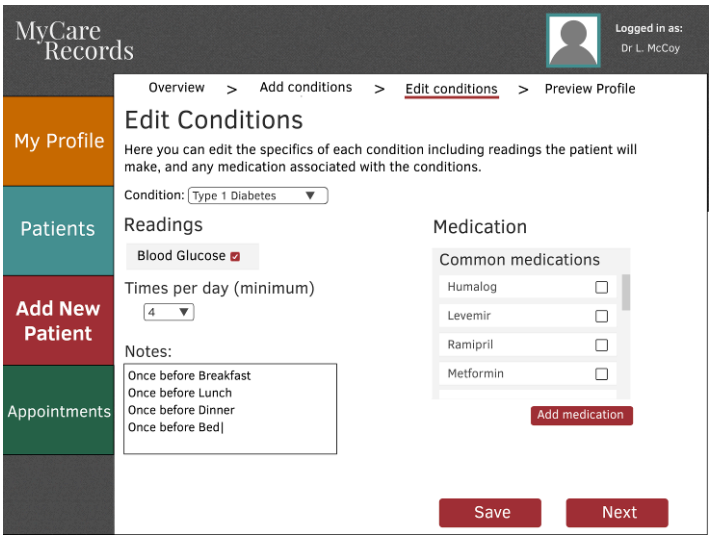


Fig. 3.8

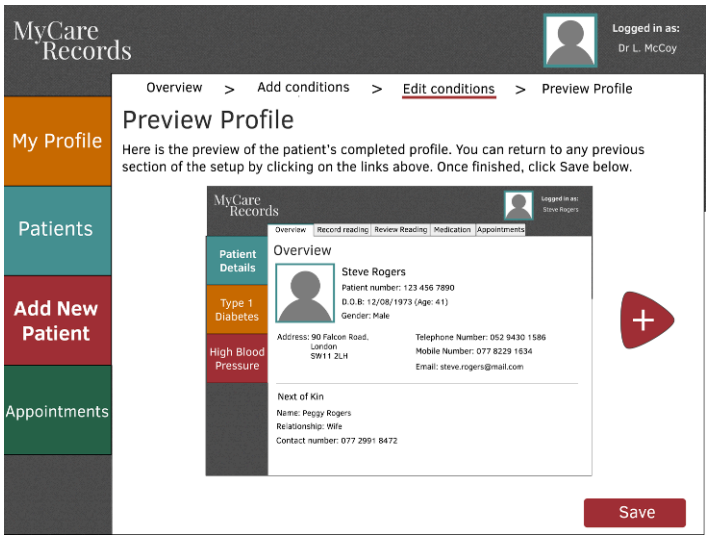


Fig. 3.9

The main issue with this initial approach was determining what would happen if the HCP did not complete the entire process. Would the information be partially saved, or would they need to start from scratch? These issues were over-complicating the process. In the refined UX design, the process was condensed into a single form (Fig. 3.10). All the above components were still incorporated, but were altered slightly to fit with the new layout.

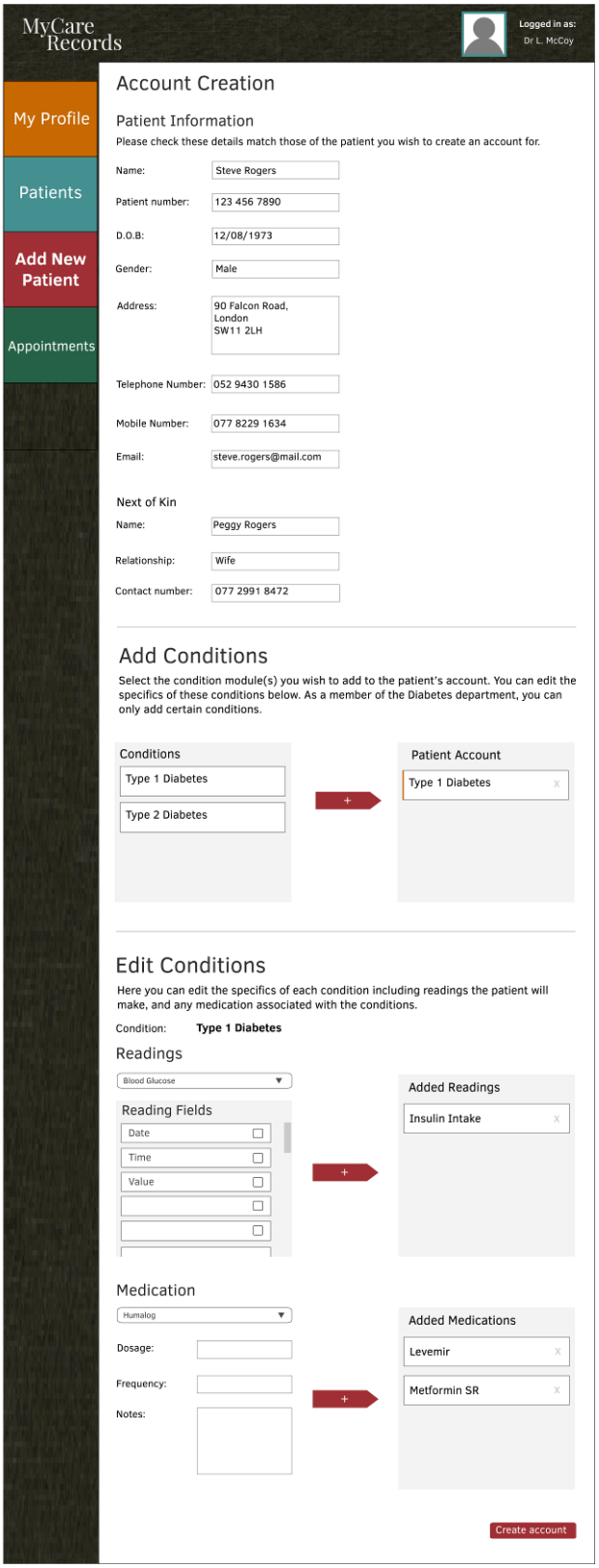


Fig. 3.10

3.2 System Design

When designing the system, the main factors, which needed to be considered were; which system architecture would best suit the project, what technologies would be needed and how would all these technologies fit into that architecture.

3.2.1 System Architecture

There are several different system architectures, which could have been used in the creation of the MyCare Records system. Each architecture was carefully considered before a choice was made.

Client-Server Model

One of the most basic ways of displaying the structure of a system is the Client-Server Model. This model shows the relationship between the client-side of a system and the server-side. The system also shows the various components present on each side of the model (Fig. 3.11).

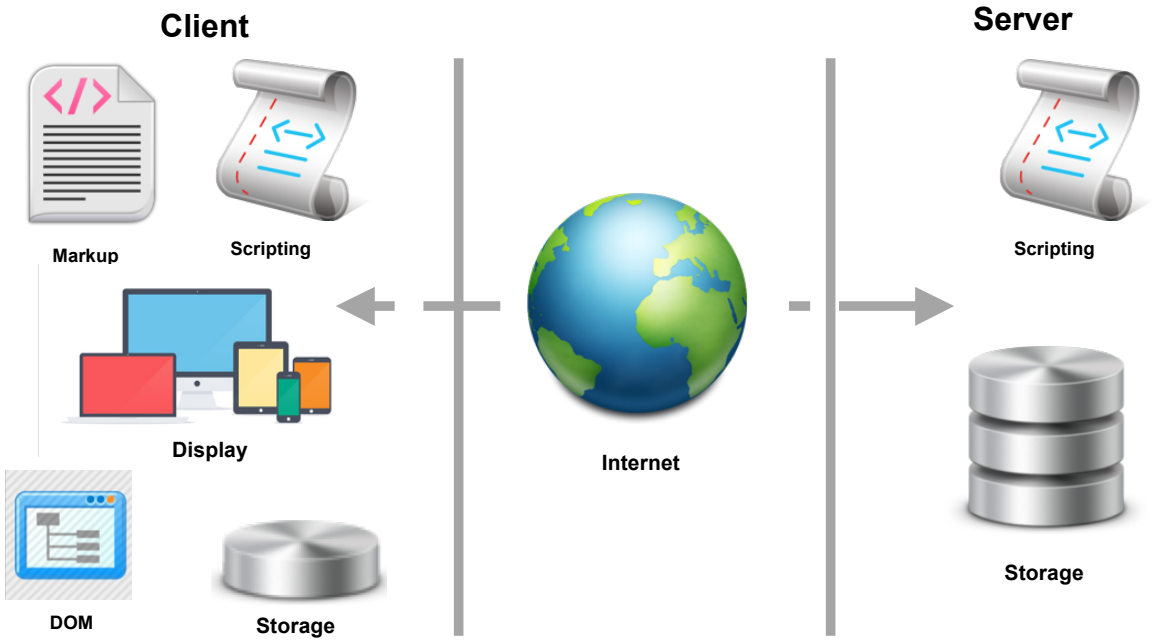


Fig. 3.11

Client-Side

1. **Markup**: Markup is used to express the processing, definition and presentation of text on a web page using languages such as XML, XHTML, and HTML (the latest version being HTML5).
2. **Scripting (client-side)**: A scripting language is often used on the client-side to add functionality to a web page by affecting the data the user sees in their browser. Common client-side scripting languages are JavaScript, Dart, VBScript (although this can also be used server-side), and Python.

3. **Display**: When creating a web application, it is important to consider what type of display a user will be using in order to ensure that content displays correctly. Types of display include desktops, tablets and smartphones.
4. **DOM (Document Object Model)**: The DOM is "*a platform-neutral and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents*"(W3C. 2005). Being able to control the DOM is key to creating dynamic web pages.
5. **Client-Side Storage**: While not as extensive as server-side storage, small amounts of data can be stored client-side. The most common way has been the use of cookies. With the introduction of HTML5, other methods of client-side storage have been developed. These include methodologies such as Session Storage, Local Storage, and even Database Storage. However, these storage methods are only available on the most up to date versions of browsers.

Server-side

1. **Scripting (server-side)**: Server-side scripting is often used to provide an interface for the user to access databases or file storage systems held on the server. There are many server-side scripting languages such as PHP, ASP.NET, Ruby, and even server-side JavaScript (such as Node.js).
2. **Storage**: Data is stored on a server using a database. There are different types of databases which can be used, depending on what way they are queried. For example, MySQL is an open source relational database management system, which uses SQL (Structured Query Language) to retrieve information from the database. Another open source database, MongoDB, does not use SQL, but rather uses JSON-style documents to retrieve information.

Three-tier architecture

Three-tier architecture divides the various components into three distinct sections:

1. **The Presentation Tier**: This is the top level and deals with the user interface. Its main task is to present the information in a way users will understand. This is done using markup languages such as XML, XHTML, HTML, and HTML5. Another language used in this tier would be CSS in order to style the content.
2. **Logic Tier**: This layer controls the functionality of the system by moving and processing data between the Presentation Tier and Data Tier, performing calculations and processing commands. This would be done using scripting

languages like JavaScript or Python, as well as languages such as PHP, ASP, ASP.NET, and Ruby to interpret information from the Data Tier.

3. **Data Tier**: This tier holds all the data needed within the system. This is typically stored in a database or file system. It will receive requests from the Logic Tier, and send back the results of these requests.

The three-tier architecture is a useful model to use as technology is always progressing rapidly. Using the architecture means that only one tier may need to be changed in order to take advantage of a new technology or programming language, while the other tiers remain the same. This stops the entire system needing to be rebuilt.

Model-View-Controller

Model-view-controller is used when generating user interfaces. Like the three-tier architecture, it is split into three sections. Each section is responsible for one aspect of the user interface (Fig. 3.12).

Model

The model represents knowledge. This knowledge is made up of the data and logic that models the knowledge. For example, in a simple website this would be the HTML. It provides the markup, giving the text structure and meaning.

View

The view is the visual representation of the model. It will fetch the necessary data from the model by asking questions, and update the data by sending messages back to the model. An example of this would be CSS. CSS is used to give HTML a particular look and feel. It is also possible to swap between CSS files without altering the HTML in any way.

Controller

The controller is the link between the user and the system. Its main task is to manage the delivery of views to the user. It never modifies the structure of the view. It provides a way for the user to input commands and data to the system. The controller will take the commands and relay them to the viewer. The browser often acts as the controller. It is responsible for taking CSS and HTML and combining them into one web page, that the user can understand and interact with.

Some say there are other components which should be present within the model-view-controller architecture.

- **User:** The user is the one interacting with the system and without them the system would not function correctly.
- **Router**: The router would be a component sitting outside of the functionality of the system, and would be responsible for ensuring messages reached the correct component. It would also state which component sent the message.

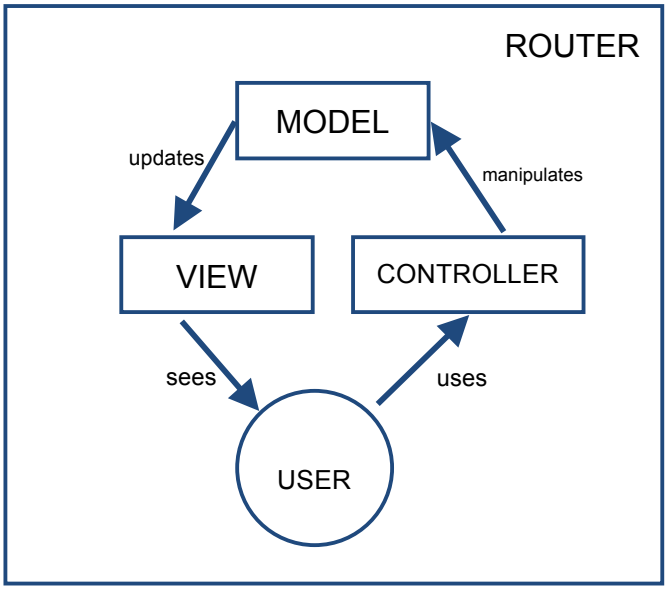


Fig. 3.12

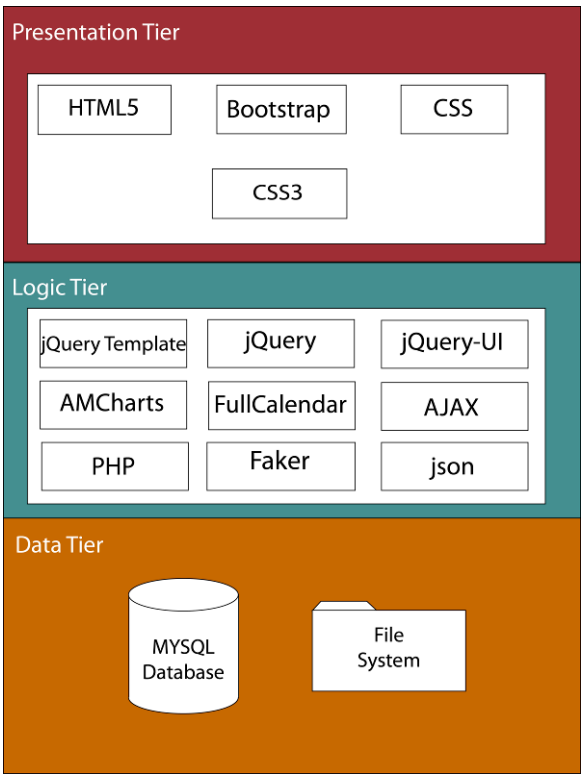


Fig. 3.13

Chosen Architecture

The architecture, which was chosen to develop the MyCare Records system, was the three-tier architecture. This architecture is able to combine the various components that make up the client-server model, while still maintaining segmented levels. As stated above, this allows for new technologies to be implemented easily. The architecture also allows for crossover between the tiers in a way the Model-View-Controller does not. For example, an element of the logic tier can have the ability, to a certain extent, to modify the structure of a file within the presentation tier. This allows for greater flexibility in creating a dynamic web application. As shown in Fig. 3.13, the technologies used within the project can be placed effectively into the structure of the architecture.

3.3 Presentation Tier

The presentation tier of the system uses several different technologies in order to display information to the user.

HTML5

HTML (Hypertext Markup Language) is the standardised markup language for creating web pages. HTML consists of a series of tags, which are used to give web pages structure, and may also contribute to some styling aspects of the page such as headings, paragraphs, lists and so on.

HTML5 is the latest version of HTML and was officially completed 24th October 2014. It still consists of tags and is used to provide structure to web pages. However, there are many new features which were not present in previous versions of HTML. Several of these features have been used to create the MyCare Records system.

- **New data types**: several new data types have been added to the input field. Several of these have been used in creating the various forms within the system. These data types include: `email`, which validates the text input within some browsers, and `tel`, which is used for telephone numbers.
- **Placeholders**: placeholders have been used within the system to give users an example of the kind of input to be placed in an input field. For example, if a patient was recording a reading about their blood pressure, the placeholder would say, 'e.g. 120/80'.
- **Semantic header**: rather than using `<div id="header">`, HTML5 has introduced a new `<header>` tag. This has been used to create the header banner containing the branding, the user's profile image and name, and the log out link.

Bootstrap

Bootstrap is a front-end framework, which allows for the implementation of a grid system when designing page layouts. This grid system is fluid; meaning that it, and the page content, are able to adapt to the size of the device the web page is being viewed on. The framework was developed with a collection of re-usable components such as buttons, navigation tabs, progress bars, warning messages and more. It also provides several JavaScript components in the form of jQuery plugins, allowing the implementation of dialog boxes, tooltips, and other features. When downloaded, the framework provides a series of CSS, JavaScript, and font files, which implement the grid system.

Two other front-end frameworks were considered; Foundation and Skeleton. All three frameworks provided the ability to create responsive web applications. Skeleton was not used, as it was the most basic package of the three, not providing any reusable

components. It was also decided that Foundation would not be used, as at the time of research, there were compatibility issues with HTML5.

CSS and CSS3

CSS files control the appearance of the application, including the layout, typography, colour scheme and so on. Certain CSS3 elements are also used within the system.

- **Media queries**: Media queries act as the break points for screen sizes. These allow the site to be responsive. The majority of the media queries are incorporated within the Bootstrap framework. However, several adjustments had to be made to these in order to ensure the site displayed correctly on certain devices.
- **Borders**: CSS3 brought the ability to alter borders much more extensively. This included creating rounded borders on input fields. This particular feature has been used throughout the MyCare Records system as part of the visual design.

3.3 Logic Tier

The logic tier of the system features many different technologies; from various JavaScript and PHP libraries, to plugins and APIs.

jQuery

JavaScript is a scripting language, which allows functionality to be added to web pages. JavaScript is essential to create pages with dynamic content by changing HTML elements and attributes, changing CSS, and validating data. These tasks are often carried out by manipulating the DOM.

jQuery is a library which makes DOM navigation and manipulation simpler. It allows elements such as animation, AJAX, and event handling to be used with the implementation of a simple API. jQuery has been used extensively throughout the application. In order to try and ensure the jQuery library will always be available to the user, links to both a local version and web version of the library have been added to the system.

jQuery-UI

jQuery-UI is a library which it built upon the main jQuery library. It provides an impressive number of components, which can be incorporated to enhance the interactivity of a site. These components include tabs, accordions, tooltips, progress bars and more. While Bootstrap also provides many of these features, jQuery-UI allows for better control over the components, as well as an easier level of customisation. In the case of MyCare

Records, the main component that was used from the library was tabs. These were used to divide the modules into various sections easily and clearly.

jQuery Templates

A jQuery template is a plugin, which allows for repeated information to be rendered easily. The data is passed to the template using a jQuery statement before being appended to a target element, such as a div or form. In the MyCare Records system, examples where jQuery templates were used within the system were to render; lists of medication on patient's profiles, the patient list of an HCP, and the overview information of both HCPs and patients. The templates were useful in keeping the data, logic and presentation tiers separate.

AMCharts

AMCharts is an API consisting of a series of JavaScript libraries. These libraries can be used to generate charts of many different types. Each type of chart can be customized further, ensuring that information can be displayed exactly as required. Additionally, all the charts have multiple levels of interactivity and animations. For example, the value of data points can be displayed when the user rolls their mouse over them. This API is key to generating visual outputs of user's information within the MyCare Records system. The API was used to create both line charts for many of the readings, as well as candle graphs for displaying blood pressure readings.

FullCalendar

FullCalendar is an API, which creates a fully functional calendar. The calendar allows for several different views; monthly, weekly and daily. The API allows appointments and readings to be added to the calendar with a specified date and time, as well as any other additional information. These were added using JSON, which was retrieved directly from the database using AJAX. The API also allows for a click function to be added to appointments. This was combined with the module component from Bootstrap to create a dialog box showing all the appointment or reading information.

PHP

PHP is a server-side scripting language primarily designed for use in web development. PHP also runs on most servers and is compatible with various platforms. It was used extensively throughout the creation of the system for various purposes.

- **Collecting form data**: PHP was used to collect the data from various forms within the application, configure the data, and pass it into the database.
- **Querying the database**: as the application needed to be dynamic, there were many AJAX calls made to the database. PHP was used to query the database for the required information and return it to the AJAX call in the required format.
- **Sending and receiving cookies**: cookies were used for several purposes within the system. PHP was used to create and check the cookies.

As the information being used within the system is of a sensitive nature, it was important to try and prevent it from being susceptible to SQL injection attack. PDO was used within the PHP files that accessed the database in order to try and prevent attacks. SQL statements were used in order to query the database.

Cookies

Cookies were used within the system to in order to make important information available. Once a user logs into the system, the following cookies are created:

- **User**: This cookie held the user's ID. It was used to check that the user had logged in to the system. It was also used in several of the PHP files to run queries to the database.
- **Type**: the type cookie indicated whether the user was a patient or HCP. This would ensure they were redirected to the correct account type.
- **Department**: In the case of a user being an HCP, a cookie was also created to hold the name of the department they belonged to. This was used when creating the 'Add Patient' form to ensure they could only add information about conditions they were capable of treating.

Each cookie was given an expiry time of 30 minutes. This timeframe was chosen for several reasons. Firstly, as the system holds sensitive personal information, it was important to keep the expiry time short to limit the risk of a user leaving their account logged in and information being stolen. Secondly, in the cases of some conditions, a user will need to record one type of reading, and then another type a short time later. The 30 minute timeframe should allow most users to make both readings without needing to log into the system more than once.

A cookie to hold the password of the user was not created, as it would provide an unnecessary security risk.

3.4 Data Tier

The data for the MyCare Records system is held within a relational database. This database contains several tables, which hold all the necessary information in order for the system to operate. When designing the data, an ER diagram was created in order to determine what tables would be needed and how they would link together. The cardinality relationships between the tables were then researched and schema for the tables was outlined.

3.4.1 ER Diagram

An Entity Relationship Diagram shows all the tables which will be present in the database and how they are related to each other. There are three types of relationship: one-to-one, one-to-many and many-to-many. The ER diagram for the MyCare Records system is shown in Fig. 3.14.

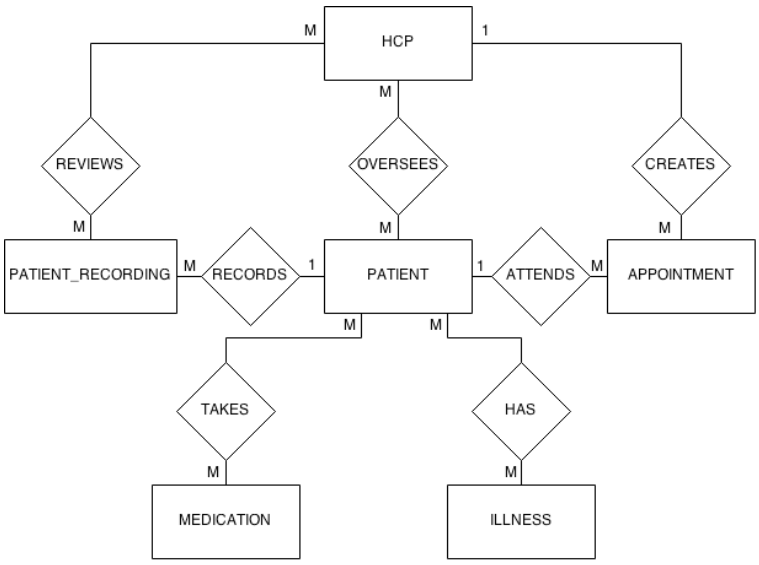


Fig. 3.14

3.4.2 Cardinality Relationships

Cardinality relationships explain the relationships between the tables in simple words. This helps to fully understand the structure of the database.

PATIENT to PATIENT_RECORDING is 1:M because:

- One PATIENT records many PATIENT_RECORDINGS
- One PATIENT_RECORDING is made by one PATIENT

PATIENT to MEDICATION is M:M because:

- One PATIENT may take more than one MEDICATION
- One MEDICATION can be taken by many PATIENTS

PATIENT to APPOINTMENT is 1:M because:

- One PATIENT can attend many APPOINTMENTS
- One APPOINTMENT is attended by one PATIENT

PATIENT to HCP is M:M because:

- One PATIENT can be overseen by more than one HCP
- One HCP can oversee more than one PATIENT

PATIENT to ILLNESS is M:M because:

- One PATIENT can have more than one ILLNESS
- One ILLNESS can affect more than one PATIENT

HCP to APPOINTMENT is 1:M because:

- One HCP can create many APPOINTMENTS
- One APPOINTMENT is created by only one HCP

HCP to PATIENT_RECORDING is M:M because:

- One HCP can review many PATIENT_RECORDINGS
- One PATIENT_RECORDING is reviewed by more than one HCP

In the case of several of the cardinality relationships, many to many relationships are present. In these cases, third normal form rules can be applied in order to resolve issues with these many-to-many relationships. This involves creating a link table, which can act as a connection between the two. Typically, this is done by creating a new table and creating a concatenated primary key using the two primary key of the original tables. Separately, each primary key can also act as a foreign key.

3.4.3 Database Schema

APPOINTMENT(appointment_id, start, end, title, location, clinic, patient_id, professional_id, condition_id, color)

HCP(professional_id, firstname, surname, hcp_title, telephone_number, mobile_number, email, department, img, password)

ILLNESS (condition_id, condition_name, filename)

ILLNESS_READING (condition_id*, reading_id*)

MEDICATION (medication_id, name, type, use, condition_id*)

PATIENT (patient_id, firstname, surname, DOB, gender, address1, address2, postcode, telephone_number, mobile_number, email, img, nok_name, nok_relationship, nok_number, password, has_account)

PATIENT_CONDITION (condition_id*, patient_id*, professional_id*)

PATIENT_MEDICATION (patient_id*, medication_id*, dosage, frequency)

PATIENT_RECORDING (reading_time, start, reading, reading_id, title, patient_id*, condition_id*, open, close)

READING (reading_id, reading_type, unit)

Key:
Underlined field indicates a primary key
* indicates a foreign key

3.4.4 File System

The MyCare Records system is comprised of a number of files of various types. In order to keep these files organized, a file structure has been used.
All the CSS files have been placed within a CSS folder. The img folder contains all the images used within the system, including those of patients and HCPs. The js folder holds all JavaScript files and libraries including jQuery, moment.js, jQuery-UI, and all the JS files for AMCharts and Fullcalendar. The script file holds the various PHP files, as well as the files for Faker and Carbon. Finally, all other files, such as the HTML files, are held within the main MyCare Records folder (Fig. 3.15).

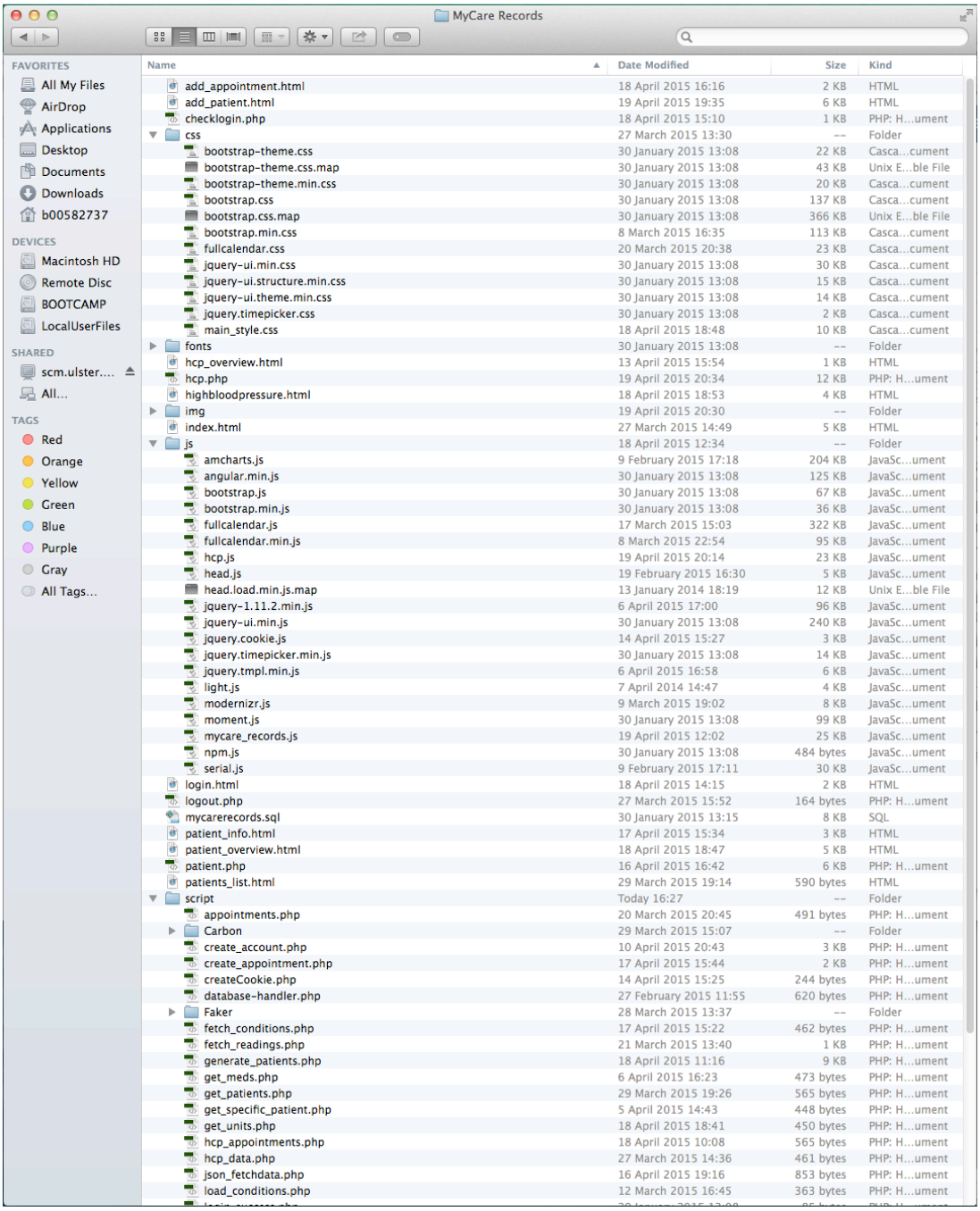


Fig 3.15

4. Implementation

When creating the MyCare Records system, the development was divided into several parts; the creation of the login system, development of a basic template for both account types, the development of the patient accounts, and the development of the HCP accounts. Additionally, during the development some elements of testing were carried out. These included unit tests and failure tests. The explanation and results of these tests can be found in Chapter 5.

4.1 Development of Login

In order to allow both patients and HCPs access their accounts, a login system needed to be developed. The login uses a form, which takes the users ID number and password. This is sent to a PHP file, which compares the values provided with those held in the database. This is done by running two queries; one, which checks the username and password against the patient table, and another which checks them against the HCP table. A row count is performed to see which query brought back a result (Fig. 4.1).

```
$myusername=$_GET["userID"];
$mypassword=$_GET['password'];

$query_patient="SELECT patient_id, password FROM patient WHERE patient_id='$myusername' and
password='$mypassword';";

$query_hcp="SELECT professional_id, password FROM hcp WHERE professional_id='$myusername' and
password='$mypassword';";

$result_patient = $dbh->query($query_patient);
$result_hcp = $dbh->query($query_hcp);

$count_patient = $result_patient->rowCount();
$count_hcp = $result_hcp->rowCount();
```

Fig. 4.1

If the credentials are correct, two if statements are used to determine which cookies are created. These provide authentication to the system on behalf of the user, allowing them to be redirected to the correct account type, and for their personal details to be retrieved. The user is then directed to either a patient account or HCP account (Fig. 4.2). On both accounts, a PHP statement is used to check the user cookie exists. If not, the user is redirected back to the login screen (Fig 4.3).

```
if($count_patient==1){
    setcookie("User", $myusername, time()+1800);
    setcookie("Type", "patient", time()+1800);
    header('Location: patient.php');
}

else if($count_hcp==1) {
    setcookie("User", $myusername, time()+1800);
    setcookie("Type", "HCP", time()+1800);
    header('Location: hcp.php');
}
```

Fig. 4.2

```
<?php
    if (!isset($_COOKIE['User'])) {
        header('Location: login.html');
    }
?>
```

Fig. 4.3

If the user's credentials do not match those held in the database, a dialog box asks the user to re-enter their details. A redirect was echoed from the PHP file to send the user back to the login page, rather than leading them to a blank screen (Fig. 4.4 and Fig. 4.5).

```
else{
    echo "<script> alert('Incorrect username or password');
window.location = 'login.html'</script> ";
}
```

Fig. 4.4

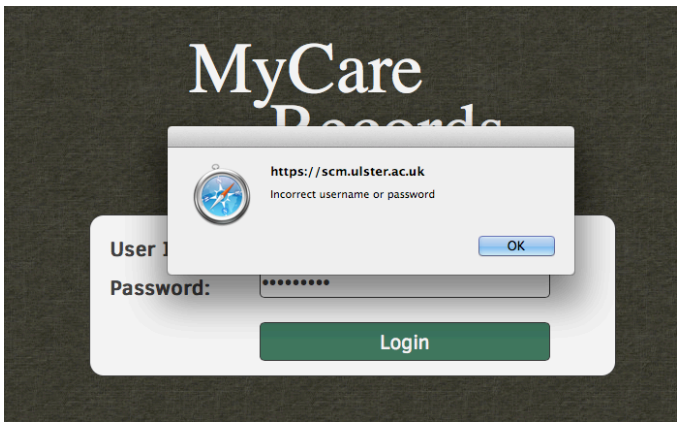


Fig. 4.5

4.2 Development of Basic Account Layout

As shown in the designs for the system, both types of account have the same basic layout. The main consistency was the use of a header. This holds the branding for the system, along with the user's avatar and name, as well as the logout link. This was created by using the new HTML5 header tag and three Bootstrap columns. The first column holds the branding for the site, which was added using SVG code (Fig. 4.6).

```
          <div id="title" class="col-md-3">

            <svg version="1.1" id="Layer_1" xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink" x="0px" y="0px"
                 width="200px" height="100px" viewBox="0 0 200 100" enable-background="new 0 0
200 100" xml:space="preserve">
            <rect y="11.53" fill="none" width="200" height="100"/>
            <text transform="matrix(1 0 0 1 0 42.897)"><tspan x="0" y="0" fill="#F2F2F2" font-
family="'PlayfairDisplay-Regular'" font-size="40">MyCare </tspan><tspan x="50" y="35"
fill="#F2F2F2" font-family="'PlayfairDisplay-Regular'" font-size="40">     Records</tspan></text>
            </svg>

          </div><!--closes title div-->
```

Fig. 4.6

The second column was used as a separator. The third column contains the user's avatar and name. An image tag was added to hold the avatar, and a div to hold the user's name. A link was also added to allow the user to log out of their account (Fig. 4.7).

```
<div class="col-md-6"></div><!--separator div-->

<div id="avatar" class="col-md-3">

    <img class="avatar" id="header_avatar" src="img/avatar.png" width="100" height="100"/>

    <p>Logged in as:</p>

    <p class="user_name" id="header_name"></p>

    <a id="logout" href="logout.php">Log Out</a>

</div>
```

Fig. 4.7

For both patient and HCP accounts, the User cookie is used to make an AJAX call to retrieve the user's information. Once the information has been returned as JSON, the attribute method was used to insert the avatar URL as the image source. The user's name was also appended to the div. If the user was an HCP, their title would also be appended (Fig.4.8 and Fig.4.9).

```
function loadHeader(json){

    $("#header_avatar").attr("src", json[0]["img"]);
    $("#header_name").append(json[0]["hcp_title"] + " " + json[0]["firstname"] + " " + json[0]
["surname"]);

}
```

Fig. 4.8

Fig. 4.9

Within the rest of the body of the page, two divs were created. The first was to hold the menu. This would list all the modules within the account as coloured 'buttons'. In the HCP account, all the modules were hard-coded, whereas in the patient account, only the Patient Overview was hard-coded. The second div was the content div. The information within each module would appear within this div (Fig. 4.10).

```
<div id="conditions" class="col-md-2">

    <ul id="menu">

        <li id="patient_overview" class="blue">
            <span id="patientoverview_link">Patient Overview</span>
        </li>

    </ul>

</div><!--closes conditions sidebar-->

<div id="content" class="col-md-10">

</div><!--closes content div-->
```

Fig. 4.10

4.3 Development of Patient Accounts

In order to create a patient account, several tasks needed to be completed and various components created. The functions, which control much of the layout and functionality of the patient account can be found in the main mycare_records.js file.

4.3.1 Condition Sections

Beginning with the basic account layout in the patient.php file, an individual patient's conditions needed to be added. As every patient's account would have slight variations,

the conditions relevant to a particular patient would need to be added dynamically. There were two approaches to achieving this that were considered.

The first consisted of creating a series of horizontal tabs nested within a set of vertical tabs. The second was using a series of spans to act as 'buttons', which the user would use to access the various condition modules. When clicked, a $.load function would be used to load an HTML page containing all the tabs for that module. While both methods were valid, it was decided to use the $.load function method, as it was felt this would be the cleaner approach.

When the patient.php file initially loads, a function is used to send an AJAX request, passing in the patient's ID as a piece of data. The patient ID can be passed into the function, or can be retrieved using a separate function, which obtains the value of the User cookie (Fig. 4.11). The PHP file queries the database, uses a while loop to fetch the column names of any conditions and place them in an array, before returning the results in JSON format (Fig. 4.12). This information would then be used to generate each condition module button.

```
if(str == undefined){
    var str = getUserCookie();
}
var url = "script/load_conditions.php";

$.ajax({
    type: "GET",
    data: "q=" + str,
    dataType: "json",
    url: url,
    success: generateTabs,
    error: errorMessage
});
```

Fig. 4.11

```
require_once("database-handler.php");

$patient = $_GET["q"];

$query = "SELECT condition_id FROM patient_condition WHERE patient_id=".$patient."";

$results = $dbh->query($query);

$details;

while($row = $results->fetchAll(PDO::FETCH_COLUMN)){
    $details=$row;
}

$dbh=null;

echo json_encode($details);
```

Fig. 4.12

A for loop was used to pass each condition ID into another AJAX call. This call queries the database and retrieves the name and filename for that condition as an associative array, before returning it as JSON (Fig. 4.13). This information was used to generate a module button for each condition.

```
var info = json;

for(i=0; i<info.length; i++){
    var str = info[i];

    $.ajax({
        type: "GET",
        data: "q=" + str,
        dataType: "json",
        url: "script/make_tab.php",
        success: makeTab,
        error: errorMessage
    });

}//closes loop
```

```
require_once("database-handler.php");

$condition=$_GET["q"];

$query = "SELECT condition_id, condition_name, filename FROM illness WHERE
condition_id=".$condition.";";

$results = $dbh->query($query);

    $details;

    while($row = $results->fetchAll(PDO::FETCH_ASSOC)){
        $details=$row;
    }

$dbh=null;

echo json_encode($details);
```

Fig. 4.13

The module buttons were generated by using a $.each loop. For every item, a span contained within a list item was appended to the menu div (Fig. 4.14). These spans would act as the 'buttons' the patient would use to access the sections for each condition.

```
                $.each(result, function(i, item) {
                    $("#menu").append("<li id='"+item.filename+"' class='col-md-12'><span
id='"+item.filename+"_link'>" + item.condition_name + "</span></li>");

                });
```

Fig. 4.14

Click events for these dynamically generated buttons were also added. These click events would run a series of functions to load the page for that condition and enter any relevant information, such as appointment and reading data. They also specified alterations to the CSS, which would occur when that particular button was clicked (Fig. 4.15).

```
$(document.body).on("click", "#type1diabetes_link", function(event){
    event.preventDefault();
    $("#content").load("type1diabetes.html", function(){
        $("span").css("font-weight", "normal");
        $("span").css("font-size", "20px");
        $("#type1diabetes_link").css("font-weight", "bold");
        $("#type1diabetes_link").css("font-size", "23px");
        specificMedicationInfo(1);
        specificAppointments(1);
        recordReading(1);
    });
});
```

Fig. 4.15

4.3.2 Patient Information

When a patient first logs into their account, they are presented with an overview of their information, such as their contact details and next of kin. This information is retrieved from the database using the patient's ID and an AJAX call, much in the same way the condition

IDs for the condition buttons were retrieved. The PHP file retrieves all the patient's details and returns them as a JSON.

This JSON is then passed to a function, which firstly empties the target div (so information is not repeated). It then applies the JSON to a jQuery template using a $.each loop. The jQuery template then applies the JSON to a specific template, held within the header of the patient.php file, and displays the information within a specified div (Fig. 4.16 and Fig. 4.17).

```
var userData = json;

$("#overview_content").empty();

$.each(json, function(index, value){

    $("#patient-info-tmpl").tmpl(value).appendTo("#overview_content");

});
```

Fig. 4.16

```
<script id="patient-info-tmpl" type="text/x-jQuery-tmpl" >

<img src="${img}" id="patient_image" class="avatar" />

  <h2 class="user_name">${firstname} ${surname}</h2>

  <p class="id_number"><b>Patient Number:</b> ${patient_id}</p>
  <p class="dob"><b>D.O.B:</b> </p>
  <p class="gender"><b>Gender:</b> ${gender}</p>

  <div class="col-md-6" style="float: left; display: inline-block;">
      <p class="address"><b>Address:</b> ${address1} <br> ${address2} <br> ${postcode}</p>
  </div>

  <div class="col-md-6" style="float: left; display: inline-block;">
      <p class="telephone"><b>Telephone Number:</b> ${telephone_number}</p>
      <p class="mobile_number"><b>Mobile Number:</b> ${mobile_number}</p>
      <p class="email"><b>Email:</b> ${email}</p>
  </div>

  <hr style="clear: both; border: 2px solid #f2f2f2;">

  <h2>Next of Kin</h2>

  <p class="nok_name"><b>Name:</b> ${nok_name}</p>
  <p class="nok_relationship"><b>Relationship:</b> ${nok_relationship}</p>
  <p class="nok_number"><b>Contact Number:</b> ${nok_number}</p>

</script>
```

Fig. 4.17

Several lines of code are used to reformat the date into a more readable format, and a function is used to calculate their current age. These two elements are then appended to the page in the relevant location (Fig. 4.18, Fig. 4.19, and Fig. 4.20).

```
function getAge(dateString) {
    var today = new Date();
    var birthDate = new Date(dateString);
    var age = today.getFullYear() - birthDate.getFullYear();
    var m = today.getMonth() - birthDate.getMonth();
    if (m < 0 || (m === 0 && today.getDate() < birthDate.getDate())) {
        age--;
    }
    return age;
}
```

Fig. 4.18

```
var date = new Date(userData[0]["DOB"]);
var birthdate = date.getDate() + '/' + (date.getMonth() + 1) + '/' +  date.getFullYear();
var ageYears = getAge(userData[0]["DOB"]);

$(".dob").append(birthdate + " (Age: " + ageYears + ")");
```

Fig. 4.19

Patient Overview

Ryan Bennett

Patient Number: 1234567890

D.O.B: 15/6/1995 (Age: 19)

Gender: Male

Address: 56 Walden Road
Greenway
TA3 0UQ

Telephone Number: 02846622048

Mobile Number: 07988702263

Email: ryanbennett@rhyta.com

Next of Kin

Name: Judith Bennett

Relationship: Mother

Contact Number: 07019746873

Fig. 4.20

4.3.3 Recording a Reading

One of the fundamental pieces of functionality of the MyCare Records system is allowing a patient to record a reading. This is done using a simple form, which the user fills out with the appropriate information.

When the page containing the form is loaded, a function runs to activate the datepicker and timepicker on the relevant input fields. The current date and time are also generated and appended to these fields in order to help speed up the process of making a reading (Fig 4.21).

```
$('#reading_date').datepicker();

$('.time').timepicker({'timeFormat': 'H:i:s'});

var now = new Date();

var day = ("0" + now.getDate()).slice(-2);
var month = ("0" + (now.getMonth() + 1)).slice(-2);
var today = (day)+"-"+(month)+"-"+now.getFullYear();

var hours = ("0" + now.getHours()).slice(-2);
var mins = ("0" + now.getMinutes()).slice(-2);
var current_time = (hours)+":"+(mins);

$("#reading_date").val(today);

$("#reading_time").val(current_time);
```

Fig. 4.21

The user begins by selecting a reading type from a dropdown list. As the readings taken for each condition are often the same, the reading types were hardcoded into the select box. Depending on the reading chosen, the units for that reading are appended next to the value input. This was done by adding an onchange attribute to the select box, passing the current value into a function, and obtaining the correct units using an AJAX call. Once the correct unit was

```
$.ajax({
    type: "GET",
    url: "script/get_units.php",
    data: "q=" + value,
    datatype: "json",
    success: function(json) {
        var data = $.parseJSON(json);
        console.log(data);
        console.log(data[0]["unit"]);

        $("#units").empty();
        $("#units").append(data[0]["unit"]);
    },
    error: errorMessage
});
```

Fig. 4.22

returned, it was appended to a div beside the reading input field (Fig. 4.22).

They can then enter the value of their reading and submit the form. Validation is performed to ensure all the fields have been filled in correctly. This includes checking for blank fields using if statements, and ensuring only numbers, decimals and forward slashes (for the blood pressure readings) have been entered into the reading input field using a Regular Expression (Fig. 4.23).

```
var inputs = /(?:\d*\.*\/)?\d+/gm;

if(reading_type == "Select reading type..."){
    alert("Please select a reading type");
    return false;
}

else if( date === "" || time === "" || reading === ""){
    alert("Please fill in all fields");
    return false;
}

else if(!reading.match(inputs)){
    alert("Please enter a valid reading");
    return false;
}
```

Fig. 4.23

Once the validation checks have been completed, the values of each input field are passed to an AJAX call using a data string. This passes the information to a PHP file. This file formats all the inputs, such as taking the time and date values and turning them into a timestamp (Fig. 4.24). It then passes all the values into the database as a new entry. A success message is returned from the AJAX call and displayed to the patient as an alert (Fig. 4.25).

```
$date = new DateTime($reading_date);
$final_date = $date->format("Y-m-d");

$timestamp = $final_date." ".$reading_time;

$time = $timestamp;
```

Fig. 4.24

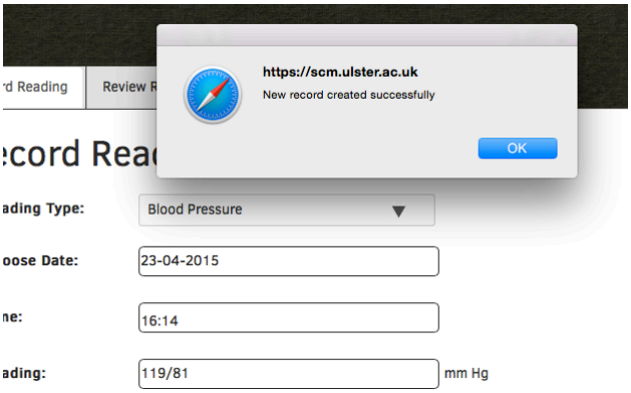


Fig. 4.25

If the reading type is blood pressure, a different query is used, as the parameters of blood pressure readings are slightly different from other reading types. Explode() needs to be used to separate the two values which make up the blood pressure reading (Fig. 4.26).

```
    $values = explode("/", $reading_value);
    $open = $values[0];
    $close = $values[1];

    $sql = "INSERT INTO patient_recording (reading_time, start, reading, reading_id,
title, patient_id, condition_id, open, close) VALUES (:timestamp, :time,
:reading,:reading_type, :title, :patient_id, :condition_id,:open,:close)";

    $stmt = $dbh->prepare($sql);
    $stmt->bindValue(':timestamp', $timestamp, PDO::PARAM_STR);
    $stmt->bindValue(':time', $time, PDO::PARAM_STR);
    $stmt->bindValue(':reading', $reading_value, PDO::PARAM_INT);
    $stmt->bindValue(':reading_type', $reading_type, PDO::PARAM_STR);
    $stmt->bindValue(':title', $reading_type, PDO::PARAM_STR);
    $stmt->bindValue(':patient_id', $patient_id, PDO::PARAM_STR);
    $stmt->bindValue(':condition_id', $condition_id, PDO::PARAM_STR);
    $stmt->bindValue(':open', $open, PDO::PARAM_STR);
    $stmt->bindValue(':close', $close, PDO::PARAM_STR);
    $stmt->execute();
    $affected_rows = $stmt->rowCount();

    if ($affected_rows == 1) {
        echo "New record created successfully";
    }
    else {
        echo "Error: " . $sql . "<br>" . $dbh->error;
    }
```

Fig. 4.26

4.3.4 Reviewing a Reading

In order to allow a patient to view their readings, a graph needed to be generated. This was done using AMCharts as previously mentioned in Chapter 2, as a simple prototype was built to determine the feasibility of the project.

The review readings tab contains a select box, which allows the patient to select the reading type, and a div where the generated chart is appended. Similarly to the select box on the Record Reading page, the onchange attribute is used to obtain the

```
function displayReadings(str) {
    var data = "q=" + str + "&id=" + userID;
    $.ajax({
        type: "GET",
        url: url,
        data: data,
        dataType: "json",
        beforeSend: function() { $('#wait').show(); },
        complete: function() { $('#wait').hide(); },
        success: determineJSON,
        error: errorMessage
    });
```

Fig. 4.27

value of the select box and pass it to a function. An AJAX call was used to retrieve the readings dependent on the value obtained from the select box and patient's ID. These results were returned as JSON. The beforeSend and complete parameters were used to show and hide a small loading animation. This was to provide visual feedback to the user to show them the graph was loading (Fig. 4.27).

The AMCharts library provides a high level of customization. For example, the position and display settings of the graph axis can be changed. The information displayed in the data balloon, which appears when the user rolls over the graph, can also be defined (Fig. 4.28). The ability to change the scale of the graphs was enabled to allow readings to be viewed for a particular time period (Fig. 4.29).

```
"chartCursor": {
    "cursorPosition": "mouse",
    "pan": true,
    "categoryBalloonDateFormat": "JJ:NN, DD MMM"
},
"categoryField": "reading_time",
"categoryAxis": {
    "minPeriod": "J",
    "parseDates": true,
    "equalSpacing": true,
    "minorGridEnabled": true,
    "position": "bottom"
},
```

Fig. 4.28

```
function zoomChart() {
    chart.zoomToIndexes(chart.dataProvider.length - 10, chart.dataProvider.length - 1);
}
```

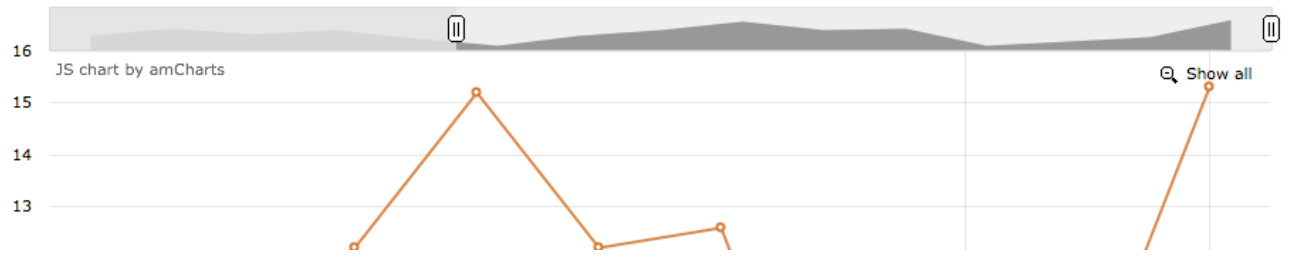


Fig. 4.29

As well as using line graphs, candle graphs were also used in order to display blood pressure readings. It was felt that these were better able to illustrate changes in blood pressure than the line graphs used for other readings (Fig. 4.30).

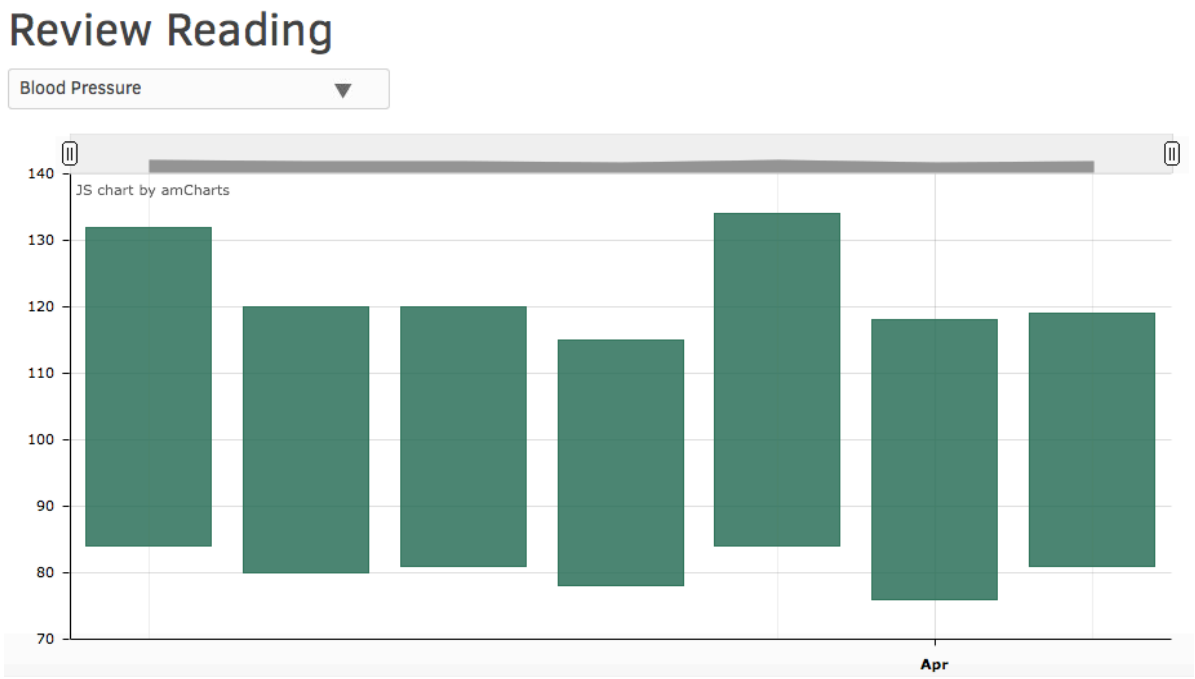


Fig. 4.30

4.3.5 Viewing Medication

The system allows a patient to view the medication they have been prescribed for their conditions. In the overview module, all their medication is listed and has been colour-coded to easily show which condition the medication is for. Iconography was also used to illustrate how the medication is administered, e.g. tablet or injection (Fig. 4.31).

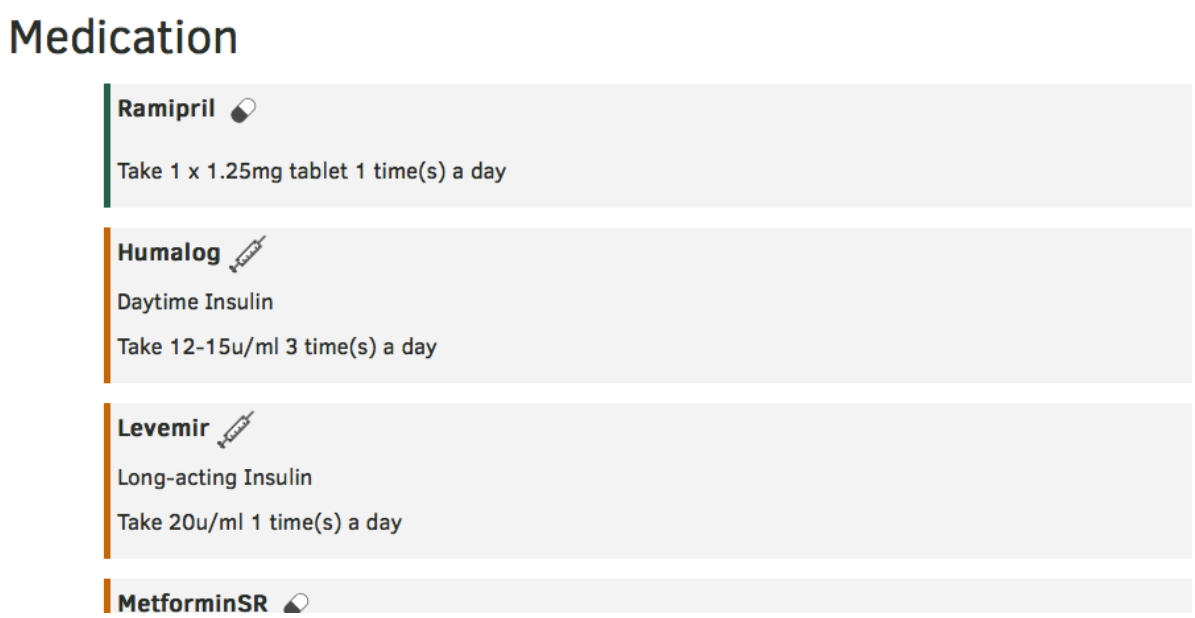


Fig. 4.31

When the patient.php page loads, an AJAX call is used to retrieve all the medication relevant to that particular patient, including the dosages and frequency of administration. The information is returned and passed as JSON to a jQuery template. A series of if statements are used within the template to determine what colour to code the div, and what icon needs to be shown (Fig. 4.32).

```
<script id="meds-info-tmpl" type="text/x-jQuery-tmpl">

    {{if condition_id == "1" || condition_id == "2"}}
        <div class='medication' id='medication${name}' style='border-left: 5px solid #c56801'>
    {{/if}}

    {{if condition_id == "3"}}
        <div class='medication' id='medication${name}' style="border-left: 5px solid #30664c">
    {{/if}}

    <p class='med_title'><b> ${name}</b>

        {{if type=="injection"}}
            <img src='img/injection.svg' width='30px' class='med_icon'/>
        {{/if}}

        {{if type=="tablet"}}
            <img src='img/tablet.svg' width='30px' class='med_icon'
        {{/if}}

    </p>

    <p class='med_description'>${use}</p>
    <p class='med_dosage'>Take ${dosage} ${frequency} time(s) a day</p></div>

</script>
```

Fig. 4.32

When the user accesses a particular condition module, a different function is used to make an AJAX call that will retrieve the medication specific to that condition. This is then passed to the same jQuery template to be displayed to the patient (Fig. 4.33).

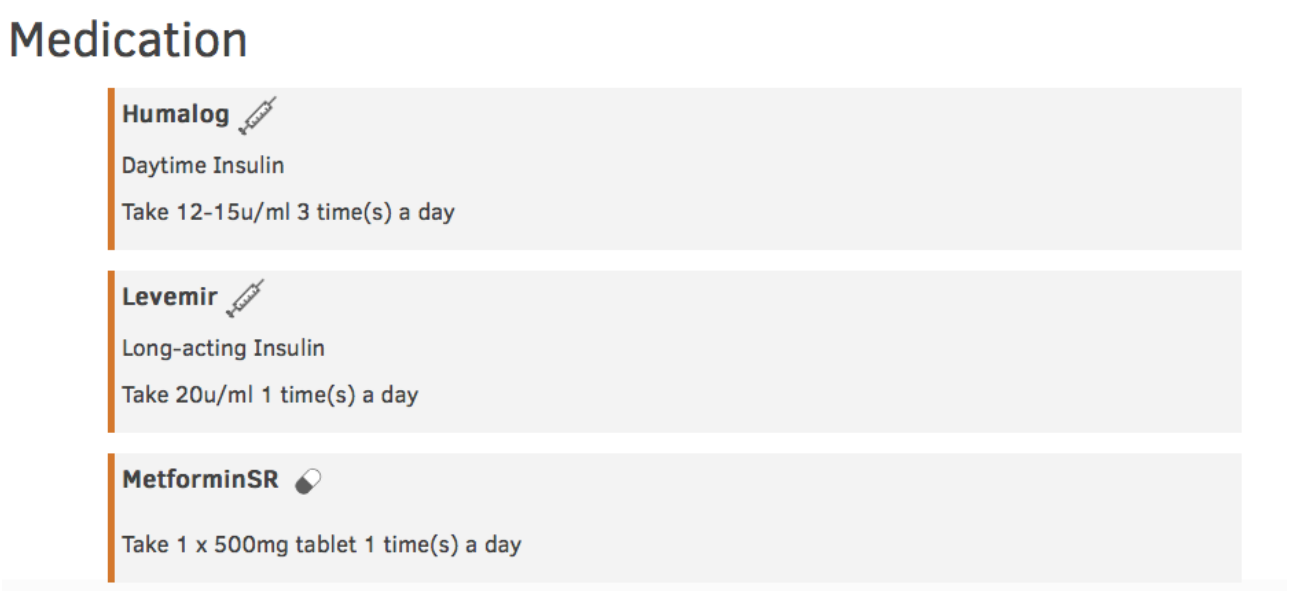


Fig. 4.33

4.3.6 Viewing Appointments

The FullCalendar API is used to generate a calendar to hold information about appointments belonging to the patient. Reading information is also added to the calendar. A function is used to run an AJAX call, which retrieves all the readings belonging to that specific patient. A $.each loop is used to check the title of each item of the returned JSON and assign a colour value. The JSON is then stored in a variable and another function is used to retrieve appointment data using the same method. The appointment data is also assigned to a variable (Fig. 4.34).

```
readingData = json;
$.each(readingData, function(i, item) {
    if(item.title == "blood_pressure"){
        item.color = "#43966e";
    }
    else if(item.title == "weight" || item.title == "steps"){
        item.color = "#498f8f";
    }

});
```

```
function determineAppointments(json){
    appointmentData = json;
    loadCalendar();
}
```

Fig. 4.34

The API is then used to create the calendar. The API allows different settings to be used, such as setting what the first day of the week should be, and determining which views can be used. The two variables, holding the reading and appointment data, are passed into the API as event sources (Fig 4.35).

```
firstDay: 1,//sets first day of week to Monday
header: {
        left: 'prev,next today',
        center: 'title',
        right: 'month,basicWeek,basicDay'
    },
eventLimit: true, // allow "more" link when too many events
eventSources: [readingData, appointmentData],
```

Fig. 4.35

The event click parameter allows for dialog boxes to be used to display more information about a particular event. A Bootstrap modal dialog box was used. As there are two types of events, if statements were used to specify what information would be displayed (Fig. 4.36 and Fig. 4.37).

```
    eventClick:  function(event) {

        if(event.title == "blood_glucose" || event.title == "blood_pressure" ||
event.title == "insulin_intake" || event.title == "weight" || event.title == "steps"){

            $('#modalTitle').html(event.reading_type);
            $('#modalBody').html("<p><b>Time:</b> " + event.reading_time + "</p><p>
<b>Value:</b> " + event.reading + " " + event.unit + "</p>");
            $('#fullCalModal').modal();

        }//closes if
```

Fig. 4.36

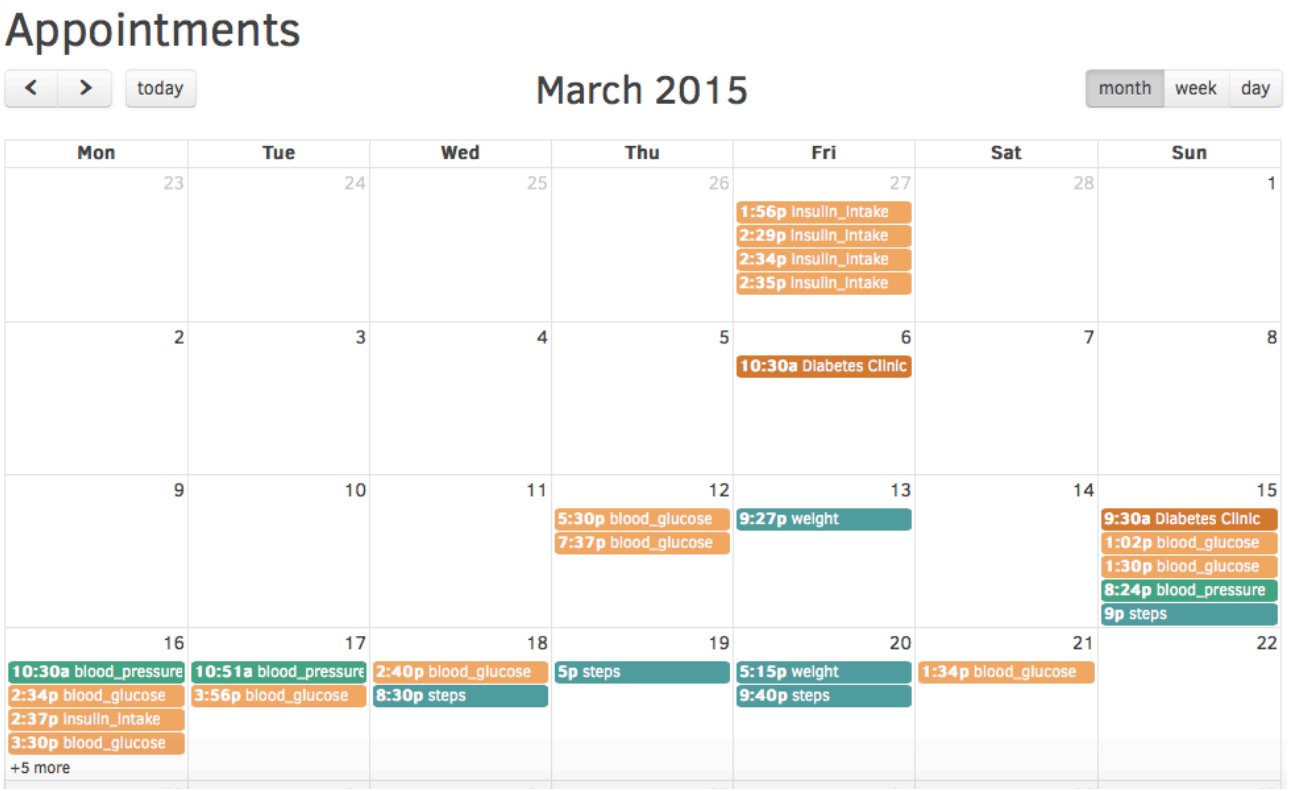


Fig. 4.37

There were also different appointment dialog boxes depending on whether the user is a patient or HCP. If the user is a patient, the time of the appointment, the department, the location, and the name of the HCP are displayed (Fig. 4.38 and Fig. 4.39).

```
    if(appointmentId == "patient"){

        var time = new Date(event.start);
        var starttime = (time.getHours() - 1) + ":" + time.getMinutes();
        $('#modalTitle').html(event.title);
        $('#modalBody').html("<p><b>Time: </b> " + starttime + "</p><p><b>Department:</b>
" + event.clinic + "</p><p><b>Location:</b> " + event.location + "</p><p><b>Healthcare
Professional:</b> " + event.hcp_title + " " + event.surname + "</p>");
        $('#fullCalModal').modal();

    }
```

Fig. 4.38

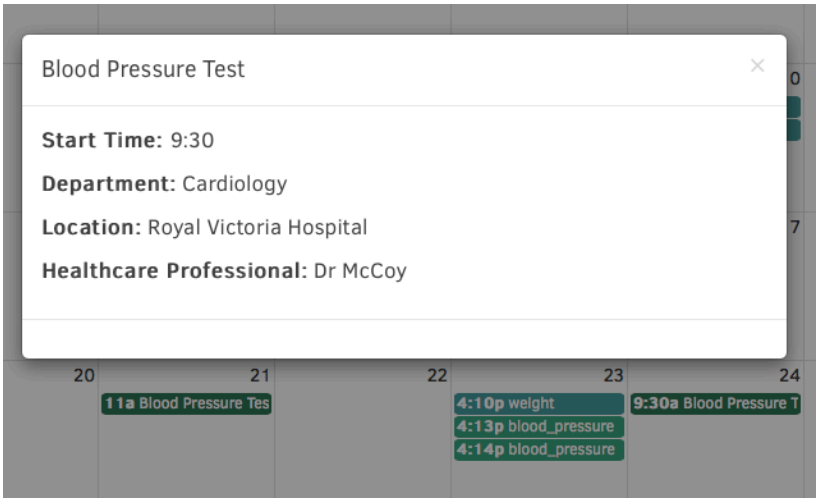


Fig. 4.39

For the calendars displayed within the condition modules, different functions were used to filter the data being used, much in the same way as the medication (Fig. 4.40).

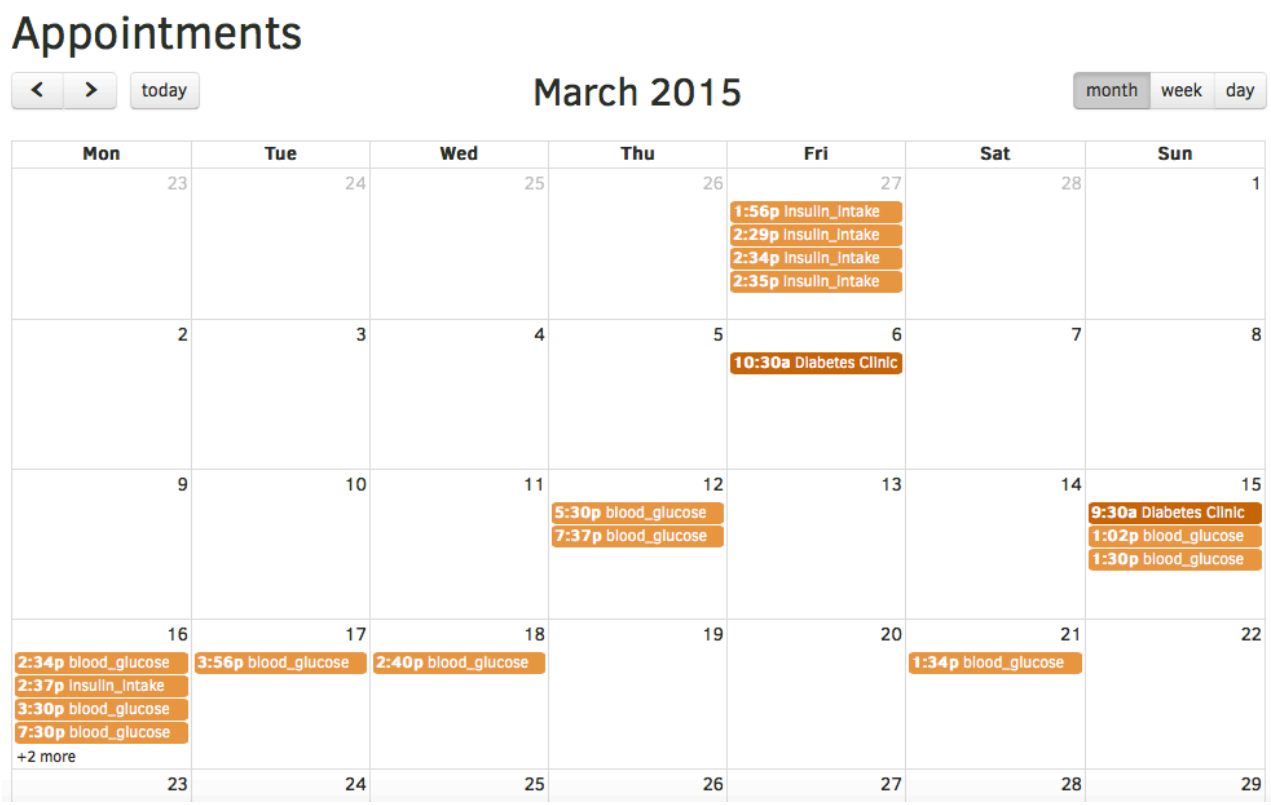


Fig. 4.40

4.4 Development of HCP Accounts

The creation of the HCP account makes use of many features created for the patient account, as well as other, unique features. In order to keep many of the functions present in the HCP accounts, only separate from the patient accounts, a separate hcp.js file was created.

4.4.1 HCP Overview

As with the patient account, when the HCP logs in to their account, they are presented with an overview module. This provides some basic contact information, as well as which department they belong to. This information is retrieved using the same method as was used to retrieve the patient details (Fig. 4.41).

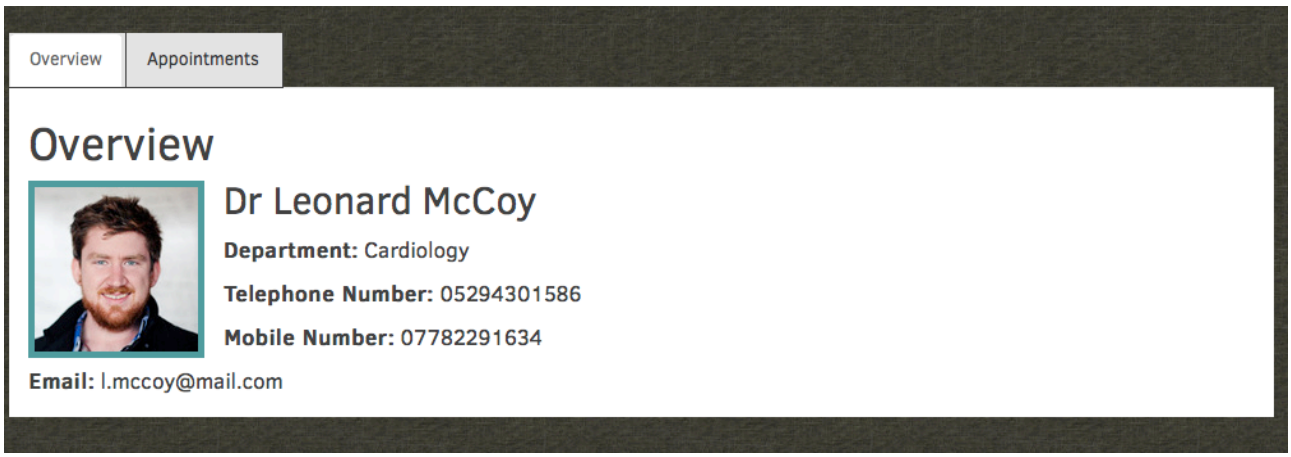


Fig. 4.41

There is also an appointment tab, displaying a calendar with the HCP's appointments. When one of the appointments is clicked, a dialog box appears detailing the time of the appointment, the patient's name and ID, as well as the location of the appointment (Fig. 4.42).

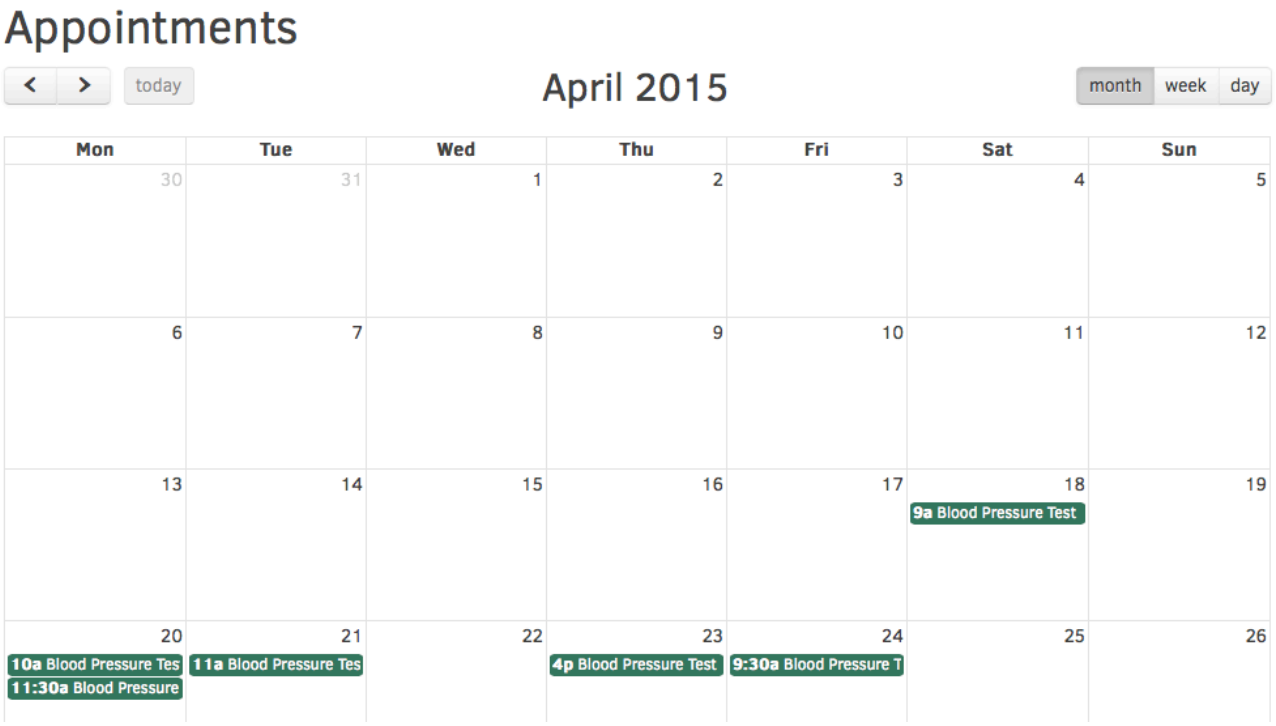


Fig. 4.42

4.4.2 Listing Patients

In order for an HCP to view all the patients in their care, a patient list was created within their account. This was done by using an AJAX call to retrieve all the patient information based on the HCP's ID number. The JSON was used in a for loop to generate a div for each patient. The div was then appended to the patient list (Fig 4.43).

```
    for(i=0; i<json.length; i++){

          var date = new Date(userData[0]["DOB"]);
            var birthdate = date.getDate() + '/' + (date.getMonth() + 1) + '/' +
date.getFullYear();

    $("#tab_patient_list").append("<div class='patient' id='patient"+ i +"'
value='"+userData[i]["patient_id"]+"'><img src='" + userData[i]["img"] + "'
class='list_avatar' ><h3>" + userData[i]["firstname"] + " " + userData[i]["surname"] + "
</h3><p id='patient_id'><b>Patient Number: </b>" + userData[i]["patient_id"] + "</p><p>
<b>Date of Birth: </b>" + birthdate + "</p></div>");


    }//closes for loop
```

Fig. 4.43

A click event was also added to each div so that, when clicked, that patient's information would be loaded into the content div (Fig. 4.44 and Fig. 4.45).

```
$(document.body).on("click", 'div[id^="patient"]', function(){

    patient_id = $(this).attr('value');
    $("#content").empty();
    $("#content").load("patient_info.html", function(){

        getReadings(patient_id);
        patientInfo(patient_id);
        medicationInfo(patient_id);
        readingTypes(patient_id);

    });
});
```

Fig. 4.44

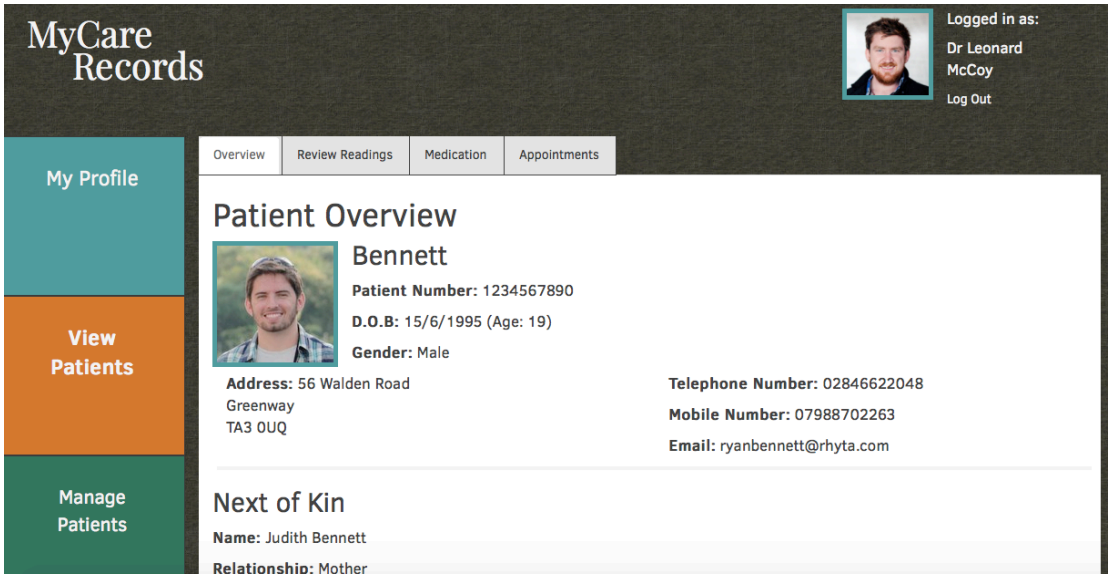


Fig. 4.45

4.4.3 Viewing Patient Information

The HCP view of a patient's information is very similar to the overview module of the patient's account, except the tab allowing the user to record readings has been removed.

Much of the information has been added to the account in the same way as the patient account, with the exception of the review readings section. A function is used to populate the select box with the reading types applicable to that patient. When the patient's details are loaded, an AJAX call is made to retrieve all the conditions, which apply to that patient, and return them as JSON.

Another function empties the select box, before adding the placeholder, weight, and steps options, as these are applicable for all patient accounts (Fig. 4.46).

```
$("#reading_selection").empty();
$("#reading_selection").append("<option value=''>Select a reading type...</option>");
$("#reading_selection").append("<option value='weight'>Weight</option>");
$("#reading_selection").append("<option value='steps'>Steps</option>");
```

Fig. 4.46

The $.each loop is then used, along with a series of if statements, to determine all other reading values which need to be appended to the select box (Fig. 4.47).

```
$.each(json, function(index, value){

    if(value["condition_id"] =="1"){
        $("#reading_selection").append("<option value='blood_glucose'>Blood Glucose</option>");
        $("#reading_selection").append("<option value='insulin_intake'>Insulin Intake</option>");
    }

    else if(value["condition_id"] == "2"){
        $("#reading_selection").append("<option value='blood_glucose'>Blood Glucose</option>");
    }

    else if(value["condition_id"] == "3"){
        $("#reading_selection").append("<option value='blood_pressure'>Blood Pressure</option>");
    }

});//closes $.each
```

Fig. 4.47

4.4.4 Adding Patient Account

As one of the main requirements for the system, it was important that an HCP has the ability to set up an account for a patient. This allows them to create an account for a patient, which includes all the relevant information about the particular condition(s) they are responsible for. For example, if the HCP was a member of the diabetic department, they can only create accounts with information relating to diabetes.

The form allows the HCP to select a patient using their patient ID. The select box is populated by using the onchange attribute to pass its value to a function. This function runs an AJAX call to a PHP file, which selects all the patients within the database who do

not have an account. A $.each loop is then used to append each patient ID to the select box.

Once an HCP selects a patient, an AJAX call is used to return all their details as JSON. This JSON is applied to a jQuery template and appended to the page under patient information. These details cannot be altered by the HCP, but are displayed to ensure the account is being created for the correct patient (Fig. 4.48 and Fig. 4.49).

```
<script id="account-tmpl" type="text/x-jQuery-tmpl">
<fieldset>

        <span class="field"><label for="name">Name: </label>
            <input type="text" class="patient_name" name="name" value="${firstname} ${surname}"
readonly></span>

        <span class="field"><label for="patient_number">Patient Number: </label>
            <input type="text" class="patient_number" name="patient_number" value="${patient_id}"
readonly></span>

        <span class="field"><label for="dob">DOB: </label>
            <input type="date" id="DOB" class="dob" name="dob" value="" readonly></span>

        <span class="field"><label for="gender">Gender: </label>
            <input type="text" class="gender" name="gender" value="${gender}" readonly></span>

        <span class="field"><label for="address">Address: </label>
            <input type="textarea" class="address" name="address" value="${address1}, ${address2}"
readonly></span>

        <span class="field"><label for="telephone">Telephone Number: </label>
            <input type="tel" class="telephone_number" name="telephone_number"
value="${telephone_number}" readonly></span>

        <span class="field"><label for="mobile">Mobile Number: </label>
            <input type="tel" class="mobile_number" name="mobile_number" value="${mobile_number}"
readonly></span>
```

Fig. 4.48

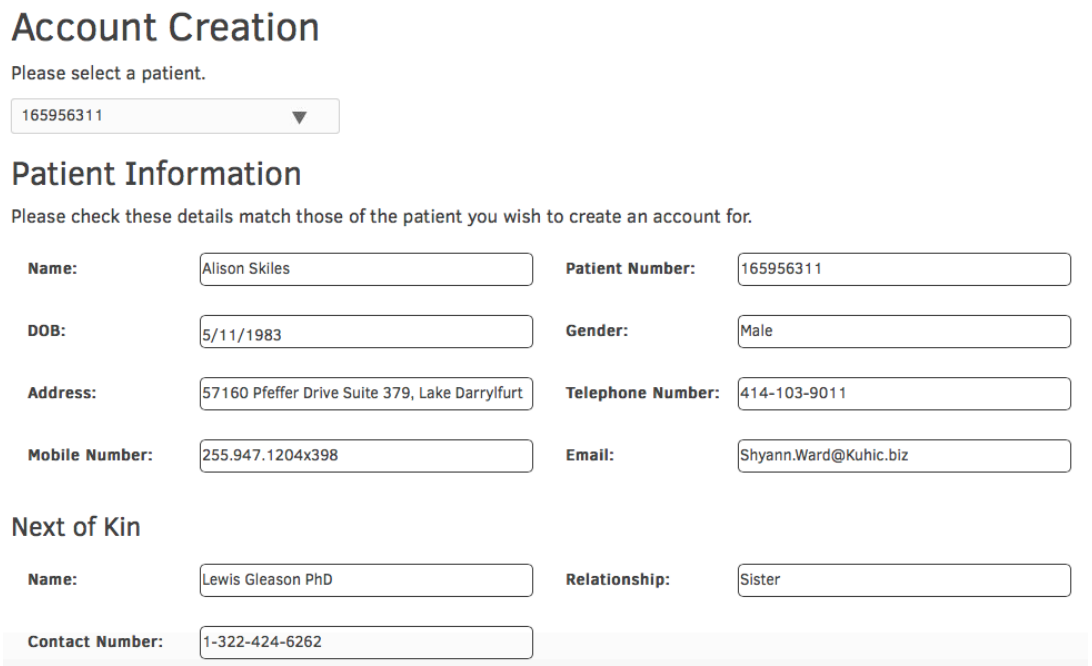


Fig. 4.49

Using radio buttons, the HCP can then choose what condition to assign to that patient. The radio buttons are appended using a template. The template used is dependent upon the department the HCP belongs to. The department is found using the Dept cookie and if statements are used to determine which template to use. A blank array needed to be passed into the template statement in order for it to display correctly (Fig. 4.50).

```
function determineConditions(){

    var cookie = getDeptCookie();

    if(cookie == "Cardiology"){
        $("#cariology-conditions-tmpl").tmpl({blank:[]}).appendTo(".condition_types");
    }

    else if(cookie == "Diabetes"){
        $("#diabetic-conditions-tmpl").tmpl({blank:[]}).appendTo(".condition_types");
    }

}//closes determineConditions
```

Fig. 4.50

The onclick attribute is added to the set of radio buttons and passes the value of the selected button to a function. This function uses a series of if statements to display the chosen condition and retrieve a list of the corresponding medication (Fig. 4.51).

```
if(info == "type1diabetes"){

    $(".chosen_condition").empty();
    $(".chosen_condition").append("Type 1 Diabetes");

    fetchMeds(1);//fetches medication for that condition

}//closes if

else if(info == "type2diabetes"){

    $(".chosen_condition").empty();
    $(".chosen_condition").append("Type 2 Diabetes");

    fetchMeds(2);

}//closes else if

else if(info == "highbloodpressure"){

    $(".chosen_condition").empty();
    $(".chosen_condition").append("High Blood Pressure");

    fetchMeds(3);

}//closes else if
```

Fig. 4.51

The condition ID passed into the fetchMeds() function is used within an AJAX call to retrieve all the medication that relates to that condition. The medications are returned as JSON, and are passed into a jQuery template to be listed as checkboxes. As well as the checkboxes, a div is created using the name of the medication as an ID (Fig. 4.52).

```
<script id="checkbox-medication-tmpl" type="text/x-jQuery-tmpl">

<input type='checkbox' id='${medication_id}' name='medication_type' value='${name}'
onclick='checkChecks(this.value)'><label>${name}</label><br><div id="${name}"
class="${name}"></div>

</script>
```

Fig. 4.52

An onclick attribute passes the value of the checkbox to a function when it is checked. This function first checks if the checkbox has been checked or not. If it has, it appends a jQuery template to the corresponding medication div. This template appends two input fields, which allow the HCP to enter the dosage and frequency for that medication. If the checkbox is not checked, the medication div is emptied (Fig. 4.53).

```
var c = $("input[value='" + value + "']").prop('checked');

if(c == true){

    //a blank data value needs to be passed in order for the template to work
    $("#medication-details-tmpl").tmpl({blank:[]}).appendTo("#" + value);

}

if(c == false){

    //removes additional fields
    $("#" + value).empty();

}
```

Fig. 4.53

Once all the relevant information has been filled in, the HCP can submit the form to create the account. All the data from the form is passed into a series of variables. In the case of the medication, several $.each loops are used to pass all the medication names, dosages, and frequencies into three separate arrays (Fig 4.50). All the values are then assigned to a data object. This data object is passed to a PHP file using AJAX. The data object needed to be stringified into JSON (Fig. 4.54 and Fig. 4.55).

```
$.ajax({
    type: "POST",
    url: "script/create_account.php",
    data: {"data": JSON.stringify(dataObj)},
    datatype: "html",
    success: function(html) {
      alert(html);
    },
    error: errorMessage
});
```

Fig. 4.54

```
event.preventDefault();
var patient_id= $(".patient_number").val();
var radio = $("input[name=condition]:checked").val();
var medication = [];
var dosage = [];
var frequency = [];
$(":checkbox:checked").each(function(i){
    medication[i] = $(this).val();
});

$(".dosage").each(function(i){
    dosage[i] = $(this).val();
});

$(".frequency").each(function(i){
    frequency[i] = $(this).val();
});

var dataObj = {};

dataObj.patient_number = patient_id;
dataObj.condition = radio;
dataObj.medication = medication;
dataObj.dosage = dosage;
dataObj.frequency = frequency;
```

Fig. 4.55

The data was decoded from JSON and each element was assigned to a variable. The HCP's ID was obtained from the User cookie (Fig. 4.56). A query was used to find the condition ID for the selected condition, as it was the ID, which needed to be added to the patient_condition table, along with the patient's ID and the HCP's ID (Fig. 4.57).

```
$data = json_decode($_POST['data']);

$professional_id = $_COOKIE['User'];

$medication = $data->medication;
$patient_id = $data->patient_number;
$dosage = $data->dosage;
$frequency = $data->frequency;
$condition = $data->condition;
```

Fig. 4.56

```
$find = $dbh->query("SELECT condition_id FROM illness WHERE filename='".$condition."';")
$fetch = $find->fetch();
$condition_id = $fetch["condition_id"];

$sql = "INSERT INTO patient_condition VALUES
('".$condition_id."','".$patient_id."','".$professional_id."');";

$run = $dbh->prepare($sql);
$run->execute();
```

Fig. 4.57

A foreach loop was then used in order to obtain all the medication ID's for the selected medication. These were stored in an array. A for loop was then used to add each medication to the patient_medication table, along with the patient's ID, and the dosage and frequency for that medication (Fig. 4.58).

```
foreach($medication as $value){

    $q = $dbh->query("SELECT medication_id FROM medication WHERE name='".$value."';");
    $f = $q->fetch();
    $result = $f["medication_id"];
    $medication_id[] = $result;
}//closes foreach

$length = count($medication_id);

for($i=0;$i<$length;$i++){

    $query = "INSERT INTO patient_medication VALUES ('". $patient_id."','".
$medication_id[$i]."','". $dosage[$i]."','". $frequency[$i]."');";
    $stmt = $dbh->prepare($query);
    $stmt->execute();
}//closes for
```

Fig. 4.58

A row count was made for the execution of both queries. An if statement was used to check if the value of both row counts was equal to one. If this was the case, the patient table was updated to set the has_account field for that patient to equal 1 (as they now had an account), and a statement was echoed to say the new account had been successfully created (Fig. 4.59). This statement would be displayed in an alert box to the HCP (Fig. 4.60).

```
$affected_rows_meds = $stmt->rowCount();
$affected_rows_condition = $run->rowCount();

if ($affected_rows_meds == 1 && $affected_rows_condition == 1) {
    $update = $dbh->query("UPDATE patient SET has_account=1 WHERE patient_id='".$patient_id."';");
    echo "New record created successfully";
}
else {
    echo "Error: " . $sql . "<br>" . $dbh->error;
}
```

Fig. 4.59

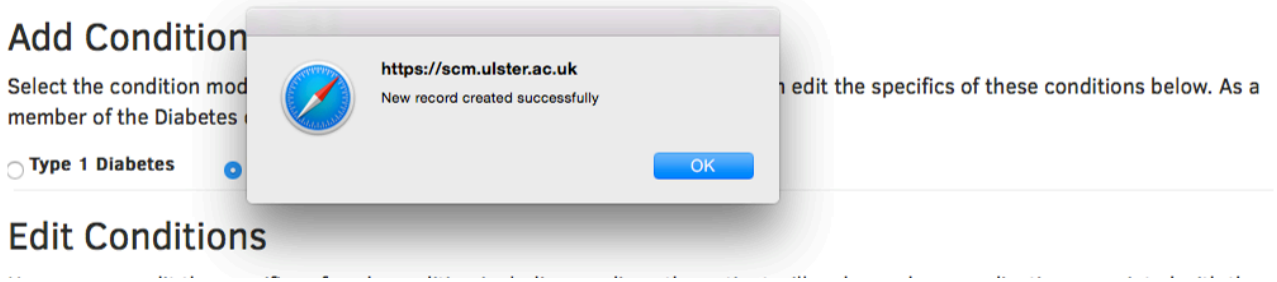


Fig. 4.60

4.4.5 Adding Appointments

The system allows HCPs to create appointments for patients by filling out a form. Similarly to the Add Account module, a PHP file is used to populate the select box. In this case, however, the PHP file returns the same patients, which appear in the Patient List module (Fig. 4.61).

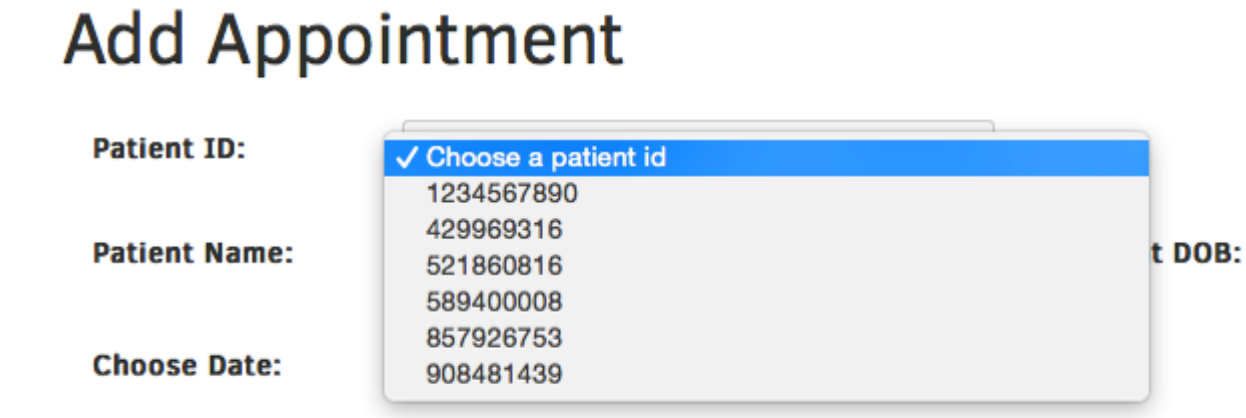


Fig. 4.61

When a patient ID is selected, the onchange attribute passes the value to a function. The function uses this to make an AJAX call which retrieve the patient's name and date of birth. These are appended to the form within the correct input fields. As in the Add Patient form, these values cannot be changed, but are used to identify the correct patient has been chosen (Fig. 4.62).

Fig. 4.62

Similarly to the Record Reading forms in the patient accounts, the current time and date are appended to their respective fields along with a time and date picker respectively. The department the HCP belongs to is also appended to the clinic input field, although this can be changed if necessary.

Once the HCP submits the form, validation is carried out to ensure all the fields have been filled in and an AJAX call sends the information to a PHP file. This file adds the information to the database and returns a confirmation message via the AJAX call, which is alerted to the HCP (Fig. 4.63).

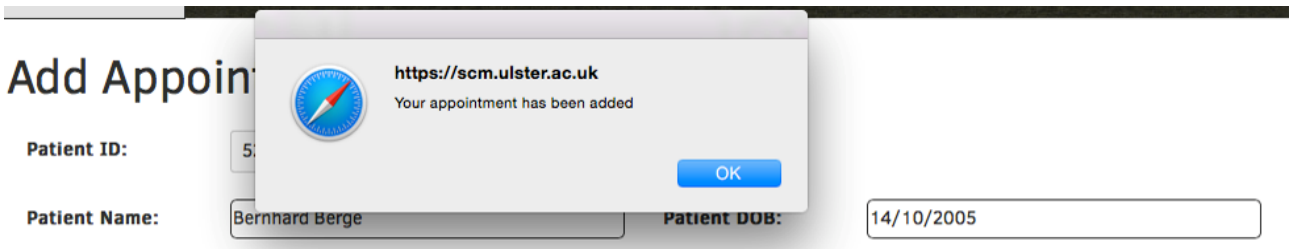


Fig. 4.63

5. Testing

In order to ensure the system behaves as expected, testing needed to be carried out. This was done using two different testing methods: white box testing, and black box testing.

5.1 Testing Approach Selection

5.1.1 White Box Testing

White-box testing is described as "*testing that takes into account the internal mechanism of a system or component*" (IEEE, 1990). A document created by Open Seminar outlines the different types of white-box testing (Williams, L. 2008). There are three main methods of white-box testing, which have been applied to the MyCare Records system.

Unit Testing

Unit testing allows various components, or units, of code to be tested. Typically, systems are built of many different units all connected together. If one of the units does not function correctly, it can often mean that most, if not all, of the system will not work as intended.

Unit tests are a form of testing often used in Behaviour Driven Development (BDD) programming. The process involves firstly writing the test and making it fail. The code is then written in order to make the test pass. This method ensures the code behaves as expected.

When carrying out unit tests on the MyCare Records system, the Jasmine library was used. This library is specifically tailored for unit testing JavaScript.

Basis Path Testing

Basis path testing takes a scenario within the system and ensures all the independent paths through the system have been considered. The easiest way to determine all the paths have been considered is to draw a flow diagram with each module on the diagram being numbered. The paths through the flow diagram can then be given, using a string of numbers. Once the paths have been determined, test cases can be written to ensure each of the paths is tested at least once.

Failure Test Cases

Failure test cases involve considering all the possible ways the users may break the program. This type of testing is particularly important, as the scenarios may not be generated using other methods. These test cases often involve looking at what the system

would do if a user entered an incorrect value into an input field, or if they try to complete tasks in a different order than intended. In order to generate the test cases, the various processes of both the patient account and the HCP account were completed with erroneous inputs entered and forms not correctly completed.

Once the test cases were generated, they were placed within a table. Each test case is given an id, a description, what the expected result is, and a space for the actual result (Table 2).

Table 2

Test ID	Description	Expected Results	Actual Results

5.1.2 Black-Box Testing

Black-box testing is concerned with the functionality of a system, rather than the internal workings. It "*focuses solely on the outputs generated in response to selected inputs and execution conditions*" (IEEE, 1990).

The main method of black-box testing used with the MyCare Records system was usability testing. As the system is a medical application, the usability had to be carefully considered, as medical errors are responsible for many deaths and injuries every year (Bond, RR. et al. 2014). It is, therefore, vitally important that the system is tested to ensure it is easy to use.

5.2 Testing Process

This section describes the process of carrying out the various testing methods described above.

5.2.1 White-Box Testing

Unit Testing

Several units of jQuery were tested using Jasmine. The main units which were tested, were AJAX calls, as they are an integral part of the system for sending information to and retrieving information from the database. Table 3 shows the tests to be carried out. Code for the unit tests can be found in Appendix 3.

Table 3

Test ID	Description	Expected Results
UT1	Check getReadings() AJAX	Pass

	is called with the correct settings	
UT2.1	determineReadings() should return item.color = #43966e when title = blood_pressure	Pass
UT2.2	determineReadings() should return item.color = #498f8f when title = weight	Pass
UT2.3	determineReadings() should return item.color = #498f8f when title = steps	Pass
UT3	Check patientInfo() AJAX is called with the correct settings	Pass
UT4	Check medicationInfo() AJAX is called with the correct settings	Pass

Basis Path Testing

Fig. 5.1 shows the various paths that can be taken through the system when recording a new reading. The paths have been outlined below using the numbers to indicate the path taken. Each path has been tested in turn and the results recorded.

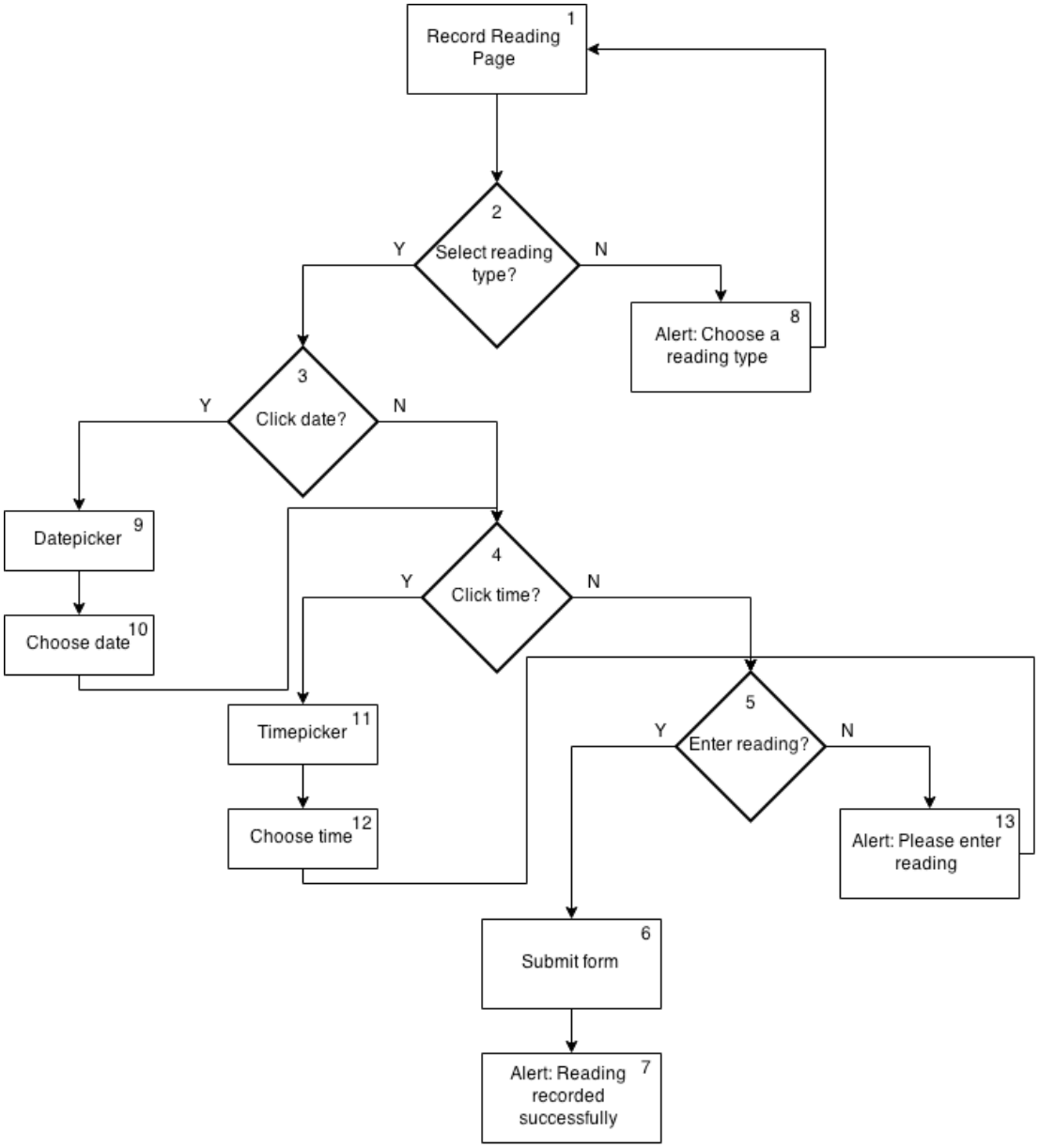


Fig. 5.1

Paths through the system

1. (1,2,3,4,5,6,7) – user fills out form correctly using pre-entered date and time
2. (1,2,3,9,10,4,5,6,7) – user fills out form correctly and chooses different date
3. (1,2,3,9,10,4,5,13) – user fills out form, chooses different date, but does not fill in reading value
4. (1,2,3,9,10,4,11,12,5,6,7) – user fills out form correctly and chooses different date and time
5. (1,2,3,9,10,4,11,12,5,13) – user fills out form, chooses different date and time, but does not fill in reading value
6. (1,2,8) – user does not select a reading type
7. (1,2,3,4,5,13) – user fills out form, but does not fill in reading value

Failure Test Cases

After going through both accounts within the system, the test cases shown in Table 4 were created.

Table 4

Test ID	Description	Expected Results
FTC1 Direct URL: https://scm.ulster.ac.uk/~B00582737/workspace/mp/hcp.php	Attempt to access the hcp.php page using a direct URL without logging in to the system	The system should redirect to the login page.
FTC2 Username = 12345 Password = foobar	Enter the incorrect username and password when attempting to log in.	An alert should appear: "Incorrect username or password". Redirect back to the login page
FTC3	Attempt to create a new account without selecting a patient	An alert should appear: "Please select a patient"
FTC4 Patient = 101596638 Condition = high blood pressure Medication = Ramapril Frequency = 2	Attempt to create a new account without specifying a dosage for a medication	An alert should appear: "Please fill in all the fields"
FTC5 Patient = 101596638 Condition = high blood pressure Medication = Ramapril Dosage = 1 x 150mg	Attempt to create a new account without specifying a frequency for a medication	An alert should appear: "Please fill in all the fields"
FTC6	Attempt to create an appointment without selecting a patient	An alert should appear: "Please select a patient id"
FTC7 Patient_id = 165956311 Time = 12:30:00 Location = Royal Victoria	Attempt to submit appointment with an alphabetic value in the date field	An alert should appear: "Please enter a valid date"

Hospital Clinic = Cardiology		
FTC8 Patient_id = 165956311 Date = 15/06/2015 Location = Royal Victoria Hospital Clinic = Cardiology	Attempt to submit appointment with an alphabetic value in the time field	An alert should appear: “Please enter a valid time”
FTC9 Patient_id = 165956311 Date = 15/06/2015 Time = 12:30:00 Clinic = Cardiology	Attempt to submit form without filling in all fields	An alert should appear: “Please fill in all the fields”
FTC10 Direct URL: https://scm.ulster.ac.uk/~B00582737/workspace/mp/patient.php	Attempt to access the patient.php page using a direct URL without logging in to the system	The system should redirect to the login page.
FTC11	Attempt to record a reading without selecting a reading type	An alert should appear: “Please select a reading type”
FTC12 Reading type = steps Date = 15/06/2015 Time = 12:30:00 Reading = foobar	Attempt to submit reading with a non-numeric value in the reading field	An alert should appear: “Please enter a valid reading”
FTC13 Reading type = steps Date = 15/06/2015 Time = 12:30:00	Attempt to submit a form without fully completing it.	An alert should appear: “Please fill in all fields”

5.2.2 Black Box Testing

The usability tests were carried out using a small number of test users. The usability test was influenced by an article by Jakob Nielsen, which states that "*The best results come from testing no more than 5 users and running as many small tests as you can afford*" (Nielsen, J. 2000).

The main methods of usability testing used were, concurrent think aloud, concurrent probing and retrospective probing. A script was written to talk each tester through the usability test by explaining its purpose and the process that would take place. This script can be viewed in Appendix 3.

The patient account and HCP account were tested separately by asking the testers to complete a series of tasks. While completing these tasks, the users were asked to describe their thinking process aloud as they worked. This helped to better understand their thinking process. Once the tasks were completed, they were asked several follow up questions about their experience using the system. All the tasks and questions asked can be viewed in Appendix 3.

5.3 Test Results

The following sections present the results of the various testing techniques carried out above.

5.3.1 White Box Testing

Unit Testing

Fig. 5.2 shows the unit tests carried out and the results that were obtained. As shown, the code successfully passed all the unit tests. The full test results can be viewed in Appendix 3.

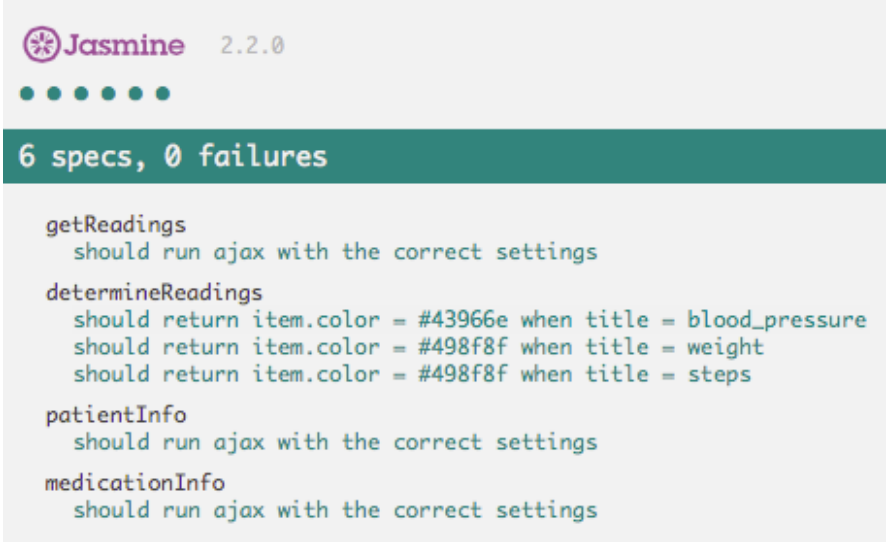


Fig. 5.2

Basis Path Testing

The various paths through the system were identified. Table 5 shows each path being tested, a description of the test, the expected results and what the actual results were.

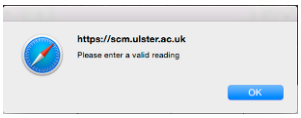
Table 5

Test ID	Description	Expected Results	Actual Results
BPT1	User fills out form correctly using pre-entered date and time	When user submits form, alert says: Reading recorded successfully	Reading recorded successfully
BPT2	User fills out form correctly and chooses different date	When user submits form, alert says: Reading recorded successfully	Reading recorded successfully
BPT3	User fills out form, chooses different date, but does not fill in reading value	When user submits form, alert says: Please enter a reading	Please enter a reading
BPT4	User fills out form correctly and chooses different date and time	When user submits form, alert says: Reading recorded successfully	Reading recorded successfully
BPT5	User fills out form, chooses different date and time, but does not fill in reading value	When user submits form, alert says: Please enter a reading	Please enter a reading
BPT6	User does not select a reading type	When user submits form, alert says: Choose a reading type	Choose a reading type
BPT7	User fills out form, but does not fill in reading value	When user submits form, alert says: Please enter a reading	Please enter a reading

Failure Test Cases

Each of the failure test cases were tested using both account types within the system. Table 6 shows a small selection of the results. The full set of results and the referenced figures can be viewed in Appendix 3.

Table 6

Test ID	Description	Expected Results	Actual Results
FTC4 Patient = 101596638 Condition = high blood pressure Medication = Ramapril Frequency = 2	Attempt to create a new account without specifying a dosage for a medication	An alert should appear: “Please fill in all the fields”	The alert box appeared 
FTC6	Attempt to create an appointment without selecting a patient	An alert should appear: “Please select a patient id”	The alert box appeared 
FTC7 Patient_id = 165956311 Time = 12:30:00 Location = Royal Victoria Hospital Clinic = Cardiology	Attempt to submit appointment with an alphabetic value in the date field	An alert should appear: “Please enter a valid date”	The alert box appeared 
FTC12 Reading type = steps Date = 15/06/2015 Time = 12:30:00 Reading = foobar	Attempt to submit reading with a non-numeric value in the reading field	An alert should appear: “Please enter a valid reading”	The alert box appeared 

5.3.2 Black Box Testing

The testing was completed with 5 users between the ages of 23 and 65, and with varying degrees of computer literacy. Three of the testers suffered from conditions outlined in the project (one type 1 diabetic, and two with high blood pressure). This helped to provide a wide demographic of users and members of the target audience. In each case, the site was tested using Google Chrome, and the accounts were reset for each tester (i.e. the readings made by the previous tester were removed from the database). Tables 8 and 9

outline some of the key tests, as well as the feedback from the testers. The full usability test results can be viewed in Appendix 3.

HCP Account

Table 7

Task	Results
Can you see more information about a particular appointment?	One tester did not know to click on appointment to view more information
Can you view the patient's calendar?	One tester did not complete, another took considerable period of time to complete
Can you set up an appointment for a patient…	All testers completed task. One tester typed in date rather than using the date picker

When asked to comment on their overall experience in using the Healthcare Professional account, all the testers stated that the system was clear, easy to use, logical and intuitive.

When asked to comment on any suggestions or improvements that could be made, a tester suggested a way of allowing the HCP to flag up any issues on a patient's account. For example, if the user had been taking medication for a certain length of time, a way of flagging to the patient they were due a review on that medication.

Patient Account

Table 8

Task	Results
Can you record a reading for the patient's weight…	All testers completed task. Two testers added units to reading
Can you tell me how many steps the patient recorded for the 10th April 2015?	One tester tried to use record reading form. Came up with figure of 152
Can you show details for a patient's appointment on 24th April 2015?	Two testers did not click appointment to view information
Can you record a blood glucose reading…	One tester made another weight reading. One tester typed in units. One tester had problems with the timepicker.
What was the patient's blood pressure on 16th March 2015 at 20:10?	One tester could not complete. Two testers attempted to use record reading form first.

When asked to comment on their overall experience in using the Patient account, again, all the testers stated that the system was clear, easy to use, logical and intuitive. All testers commented on the iconography used in the medication sections, and how clear it was in illustrating the type of medication.

When asked to comment on any suggestions or improvements that could be made, several of the testers suggested finding a clearer way to indicate which module the user was viewing. One suggested fading out the other modules in some way, while another suggested colour-coding the tabs within the module. One user commented that they found the blood pressure graph confusing and difficult to read.

6. Evaluation

With the project completed, there are several areas which require evaluation. This includes the results of the testing process, the project outcomes and the methodology.

6.1 Test Results

6.1.1 White Box Testing

The results of the unit testing shows how the units tested where of good quality as they successfully passed the tests. Only several units were tested due to the similarity in functionality between many of the functions in the system. However, the unit tests were useful for testing the more unique functions within the system, as shown in tests UT2.1, UT2.2, and UT2.3.

As shown in Table 6, all the paths through the record reading process behaved as expected. If the user entered incorrect information, the system indicated there was a problem, as shown in tests BPT3, BPT5, BPT6, and BPT7. The system successfully recorded readings when all the fields were completed correctly. Additionally, readings were also recorded successfully regardless of whether the date and/or time was the preloaded value, or specified by the user.

The failure test cases were an excellent way of checking the validation methods used within the system. As shown by the results (which can be viewed in Appendix 3), all the failure test cases behaved as expected. This shows that the integrity of the system has been taken into consideration, and that measures have been put in place to combat any errors or false readings.

6.1.2 Black Box Testing

The responses from the usability tests were very positive overall. The majority of the testers were able to complete the majority of the tasks with little to no issues. The issues that were flagged up seemed to be consistent from user to user. For example, when adding readings to the system, two of the testers also added in the units. This was unnecessary as the units appear to the right hand side of the input field. It also showed an issue in the validation process.

Overall, the users felt that the system was clear and easy to use. Three of the testers mentioned that the organization of the layout was logical and well thought out. One user stated that the system was very intuitive and had a good flow.

6.2 Project Outcomes

In order to ensure the project has met its overall aim, the project has been compared against the initially outlined objectives.

1. **Each user will have their own account for accessing the site.**

Due to the personalised nature of the system, it was important to provide every user with their own account. This account allows them access to their own personal information. The account is personalised to each individual, particularly in the case of patients, as only conditions relevant to them are present in their account. This means that only information which is necessary for that patient appears in their account.

For HCPs, their account also provides personal information about them, but also allows them to access the information belonging their patients. They can only view information about patients for whom they are responsible, meaning they only have access to the information they need.

2. **The account will be split into different sections, or modules, for each condition.**

Splitting the information into modules helps to divide up the large amount of information present in a user's account. In the case of a patient, they are able to view all the information about a particular condition all contained within one module. In the case of an HCP, the modules clearly divide the tasks they may wish to perform.

Similarly, dividing the modules into tabs helps to segment the information further. The layout of these tabs is consistent from module to module where possible. This helps the user to very quickly learn exactly where they need to go in order to obtain a certain piece of information or to complete a particular task.

3. **A patient will be able to record readings about the conditions they suffer from.**

As one of the fundamental pieces of functionality of the entire system, it was important that this process was clear and simple. The form the patient uses in order to make a reading was carefully thought out so that only the key pieces of information would need to be collected. It was attempted to make this process easier by providing a drop down to select the reading type, adding in the current date and time, and providing the units the reading would be measured in. Validation was also added to ensure that the readings made could be as accurate as possible, barring human error in entering an incorrect value.

4. **Readings recorded by a patient will be displayed in a graphical format.**

As another of the fundamental pieces of functionality, it was important that the graphical formats used were clear and easy for the user to read. Using a library, which provided a level of interactivity helps with reading the information, as simply moving over a data point brings up all the relevant information about that reading. It was important to try and make the interpretation of the readings as easy as possible, so the use of different graphs for different reading types was beneficial. It was also felt that the ability to alter the timeframe of the graphs would be helpful in identifying patterns in the readings.

5. **A patient's medication and appointments will be listed for each condition.**

It was felt that providing the patient's appointments and medication for each condition would be of benefit, as the patient would be able to use the system as a central hub for both managing and finding information about their conditions. Similarly, if the HCP was viewing the patient's account, it would be beneficial to them to also have information about their appointments and medication within the system. This would mean they would not need to log into another system to find that information.

After considering the initial objectives of the system, it is felt that the project has been successful in meeting the overall aim of developing a system, which allows patients and healthcare professionals to manage a patient's unique conditions effectively.

6.3 Methodology

The modified waterfall methodology was used to complete the project. It was felt that this methodology was effective in a project of this nature as it allowed for flexibility when moving through the various stages. For example, there were several occasions during the implementation stage of the project where the design needed to be modified for functionality reasons. The methodology effectively allowed for the need to return to a

previous stage. An aspect of the project could be reconfigured, and then be returned to the implementation stage of the model.

It is also worth noting that the prototyping methodology could have been used to manage the project. Creating a series of iterative prototypes, or minimum viable products, would have worked well in the development of the system.

7. Conclusion

The main aim of the MyCare Records system was to develop a web-based application, which would allow both healthcare professionals and patients to manage a patient's unique conditions.

The patient's conditions have been divided into separate modules to segment the information. The system allows patients to record readings for their various conditions using a simple form. These readings are then displayed in a graphical format in order to help the patient better understand their conditions. The medication prescribed for each condition, as well as the appointments with healthcare professionals are also displayed. This has been done in attempt to create a central location where patients can find information about their conditions.

The system also provides healthcare professionals with an interface allowing them to view their patient's information, as well as manage patient accounts, and create appointments. Using the MyCare Records system in tandem with a patient's ECR would help to provide a clear view of the patient's progress in coping with their conditions.

Future work to the project would include incorporating several of the features and improvements suggested in the usability testing, such as allowing a HCP to flag an element of the patient's account, which may need to be reviewed. It would also be of interest to include a way for a HCP to modify the content of a patient's account, such as changing the dosage of a type of medication, changing the type of reading types being taken, and perhaps to even remove a condition entirely.

8. References

Bond, RR. Et al. (2014) *A usability evaluation of medical software at an expert conference setting*. Computer Methods and Programs in Biomedicine, 113 (1). pp. 383-395.

Harper, R. O'Loan, D, 2011. NI Electronic Care Record (ECR), Healthcareisi, [ONLINE]. Available: http://www.slideshare.net/healthcareisi/ni-electronic-care-record-des-oloan [17/10/2014].

IEEE (1990). IEEE Standard 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology.

Nielsen, J. 2000. Why You Only Need to Test with 5 Users. [ONLINE] Available at: http://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/. [Accessed 20/04/2015].

W3C. 2005. Document Object Model (DOM). [ONLINE] Available at: http://www.w3.org/DOM/. [Accessed 15 November 14].

Williams, L. 2008. White Box Testing. [ONLINE] Available at: http://agile.csc.ncsu.edu/SEMaterials/WhiteBox.pdf. [20/04/2015].

Appendix 1: Additional Requirements

This appendix contains the requirements not present in the main report body.

Functional Requirements

Requirement #: 2

Description: The system will have a database

Rationale: This is needed to hold the information of each individual user, allowing them to log in and view their own information.

Dependencies: None

Priority: 5

Requirement #: 3

Description: Each type of user will have their own dashboard

Rationale: This will allow the user to access the various modules and tabs, which make up their account.

Dependencies: None

Priority: 5

Requirement #: 6

Description: Each module will be split into tabs.

Rationale: Each tab is a sub-section of the module. This will divide the module into sections such as appointments, medication, readings, and so on.

Dependencies: #5

Priority: 5

Requirement #: 9

Description: The graphical output can be changed to display different lengths of time.

Rationale: In order to better see patterns in readings over a period of time, the period of time the graphical output shows could be altered. For example, the graph could show a period of 1 day, 3 days, 1 week, 1 month etc.

Dependencies: #8

Priority: 3

Requirement #: 10

Description: The ECR will hold all the information known about one patient.

Rationale: Often, information about one patient can be held in several different locations. In this system, all these pieces of information will be held within the one ECR (and then split into the relevant modules).

Dependencies: None

Priority: 5

Non-functional Requirements

Usability Requirements

Requirement #: 11

Description: The system will be easy to use and understand after minimal training.

Rationale: It is important that both HCPs and patients will be able to easily use the system after being shown the basics.

Dependencies: None

Priority: 5

Requirement #: 12

Description: The system will use symbols and words that are understandable to all users

Rationale: This will ensure that the system is as easy to understand and use as possible

Dependencies: None

Priority: 3

Precision/Accuracy Requirements

Requirement #: 13

Description: All units used in the system will be accurate to at least one decimal place

Rationale: This will ensure that all readings recorded are as accurate as possible, and will be displayed accurately in any graphical outputs.

Dependencies: None

Priority: 2

Access Requirements

Requirement #: 14

Description: Only system administrators can authorise access to particular patient's account

Rationale: In order to add a patient to their list, the HCP must contact the system administrator in order to authorise the access.

Dependencies: None

Priority: 5

Integrity Requirements

Requirement #: 16

Description: All inputs will be validated before being accepted

Rationale: This will ensure that no incorrect data is being introduced into the system, potentially corrupting any recorded data.

Dependencies: None

Priority: 4

Waiting Room

The waiting room is an area where any ‘requirements in waiting’ can be stored. These are requirements which are outside the current scope of the project, but may be added in a later version of the system.

Requirement #: WR1

Description: The system will operate in local mode if disconnected from the server.

Rationale: If the patient is using the system to record readings, they could be using the system in an area where server access is not possible. They should still be allowed to make their readings and have them added to the server at a later date.

Requirement #: WR2

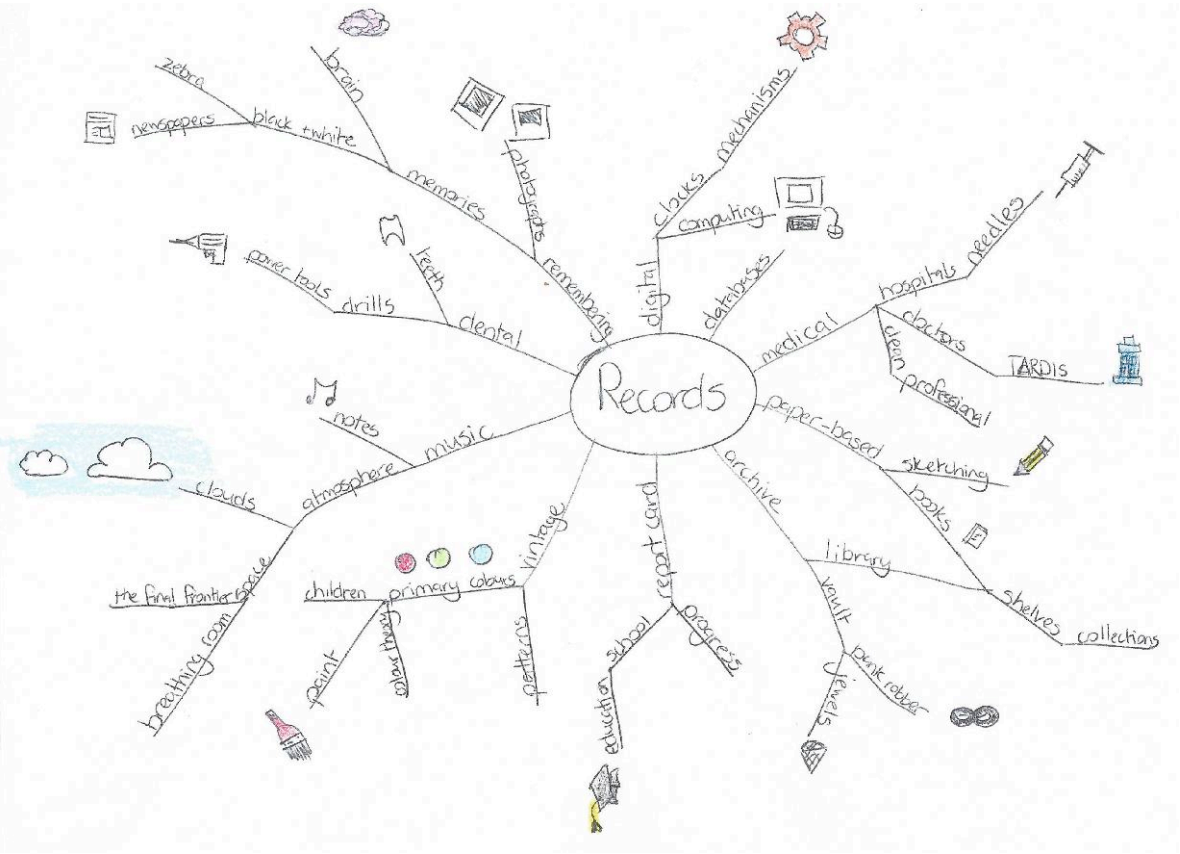
Description: The system allows a video chat between a patient and their HCP

Rationale: If there are concerns or issues that either the patient or the HCP would like to discuss, the system would allow a video chat to take place directly within the system itself.

Appendix 2: Additional Prototype materials

Mindmap

A mindmap was then created in order to explore various possibilities for the style and feel of the system. This is useful in diversifying the thinking around the system. The word 'Records' was placed in the centre of the mindmap, with 10 branches coming out from it.



Post-It Notes

The larger post-it notes represent the main sections of the site i.e. the Patient Details section, and the various Condition sections. The smaller notes represent the sub-sections within them, such as Recording Readings, Reviewing Readings, Medication etc.

This shows how the main categories could be arranged down the left hand side of the screen with the sub-categories arranged as tabs depending on which main category is active at the time.

This shows that each category could be arranged as a series of tabs, with the sub-sections arranged under each one (perhaps as an accordion).

The final arrangement shows a menu consisting of the main categories, with the relevant sub-categories appearing underneath.

Appendix 3: Testing

Unit Test Code

```
describe("determineReadings", function(){

    it("should return item.color = #43966e when title = blood_pressure", function(){

        var json = [{"title": "blood_pressure", "color": "#000000"}];

        determineReadings(json);

        expect(json).toEqual([{"title": "blood_pressure", "color": "#43966e"}]);

    });

        it("should return item.color = #498f8f when title = weight", function(){

        var json = [{"title": "weight", "color": "#000000"}];

        determineReadings(json);

        expect(json).toEqual([{"title": "weight", "color": "#498f8f"}]);

    });

            it("should return item.color = #498f8f when title = steps", function(){

        var json = [{"title": "steps", "color": "#000000"}];

        determineReadings(json);

        expect(json).toEqual([{"title": "steps", "color": "#498f8f"}]);

    });

});
```

```
describe("patientInfo", function(){

    it("should run ajax with the correct settings", function(){

        spyOn($, "ajax");

        var str = "foo";
        var url = "script/patient_data.php";

        patientInfo(str);

        expect($.ajax).toHaveBeenCalledWith({
            type: "GET",
            url: url,
            data: "q=" + str,
            dataType: "json",
            success: determinePatient,
            error: errorMessage
        });

    });

});
```

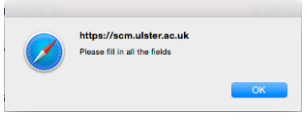
```
describe("medicationInfo", function(){

    it("should run ajax with the correct settings", function(){

        spyOn($, "ajax");

        var str = "foo";
        var url = "script/patient_med.php";

        medicationInfo(str);

        expect($.ajax).toHaveBeenCalledWith({
            type: "GET",
            url: url,
            data: "q=" + str,
            dataType: "json",
            success: determineMedication,
            error: errorMessage
        });
    });
});
```

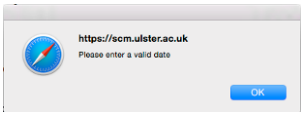
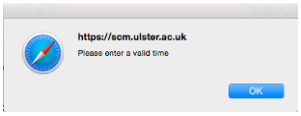
Unit Test Results

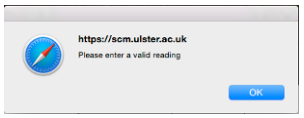
Test ID	Description	Expected Results	Actual Results
UT1	Check getReadings() AJAX is called with the correct settings	Pass	Pass
UT2.1	determineReadings() should return item.color = #43966e when title = blood_pressure	Pass	Pass
UT2.2	determineReadings() should return item.color = #498f8f when title = weight	Pass	Pass
UT2.3	determineReadings() should return item.color = #498f8f when title = steps	Pass	Pass
UT3	Check patientInfo() AJAX is called with the correct settings	Pass	Pass
UT4	Check medicationInfo() AJAX is called with the correct settings	Pass	Pass

Failure Test Cases

The table below shows the results of all the tests carried out.

Test ID	Description	Expected Results	Actual Results
FTC1 Direct URL: https://scm.ulster.ac.uk/~B00582737/workspace/mp/hcp.php	Attempt to access the hcp.php page using a direct URL without logging in to the system	The system should redirect to the login page.	The system redirected to the login page
FTC2 Username = 12345 Password = foobar	Enter the incorrect username and password when attempting to log in.	An alert should appear: “Incorrect username or password”. Redirect back to the login page	The alert box appeared and tester was redirected 
FTC3	Attempt to create a new account without selecting a patient	An alert should appear: “Please choose a patient”	The alert box appeared 
FTC4 Patient = 101596638 Condition = high blood pressure Medication = Ramapril Frequency = 2	Attempt to create a new account without specifying a dosage for a medication	An alert should appear: “Please fill in all the fields”	The alert box appeared 
FTC5 Patient = 101596638 Condition = high blood pressure Medication = Ramapril Dosage = 1 x 150mg	Attempt to create a new account without specifying a frequency for a medication	An alert should appear: “Please fill in all the fields”	The alert box appeared 

FTC6	Attempt to create an appointment without selecting a patient	An alert should appear: “Please select a patient id”	The alert box appeared 
FTC7 Patient_id = 165956311 Time = 12:30:00 Location = Royal Victoria Hospital Clinic = Cardiology	Attempt to submit appointment with an alphabetic value in the date field	An alert should appear: “Please enter a valid date”	The alert box appeared 
FTC8 Patient_id = 165956311 Date = 15/06/2015 Location = Royal Victoria Hospital Clinic = Cardiology	Attempt to submit appointment with an alphabetic value in the time field	An alert should appear: “Please enter a valid time”	The alert box appeared 
FTC9 Patient_id = 165956311 Date = 15/06/2015 Time = 12:30:00 Clinic = Cardiology	Attempt to submit form without filling in all fields	An alert should appear: “Please fill in all the fields”	The alert box appeared 
FTC10 Direct URL: https://scm.ulster.ac.uk/~B00582737/workspace/mp/patient.php	Attempt to access the patient.php page using a direct URL without logging in to the system	The system should redirect to the login page.	The system redirected to the login page
FTC11	Attempt to record a reading without selecting a reading	An alert should appear: “Please select a reading	The alert box appeared

	type	type”	
FTC12 Reading type = steps Date = 15/06/2015 Time = 12:30:00 Reading = foobar	Attempt to submit reading with a non-numeric value in the reading field	An alert should appear: “Please enter a valid reading”	The alert box appeared 
FTC13 Reading type = steps Date = 15/06/2015 Time = 12:30:00	Attempt to submit a form without fully completing it.	An alert should appear: “Please fill in all fields”	The alert box appeared 

Usability Testing: Questionnaire

The MyCare Records system allows both Healthcare Professionals and Patients to record and view readings relating to various conditions suffered by the patient. This testing will probably take no more than about 15 minutes. This questionnaire is anonymous and won't be used for any other purpose than for reporting in my assignment work. I'll be making notes as you go so I have a record of things that you found easy, hard, confusing, and so on. At end of our time I'll ask if you have any suggestions about how the site might be improved.

The web site is intended to be usable by anyone from secondary school age up. The site is just a prototype, but it contains features for recording and viewing readings, as well as viewing medication and appointments for separate conditions.

I want to make it clear right away that we are testing the usability of the web site, not you. You can't do anything wrong here. Nothing you do will be a 'mistake'. I need to hear what you think, so please don't worry about hurting my feelings. I want to know how to improve the site, so I need to know honestly what you think. I'm interested in anything that you find in some way awkward or confusing about the web site.

As we go along, I'm going to ask you to think out loud, to tell me what's going through your mind. This will help me to understand your thinking process.

If you have questions, just ask. I may not be able to answer them right away, since I'm interested in how people do when they don't have someone sitting next to them, but I will try to answer any questions you still have when we're done. Is that all ok?

We are going to start by looking at the Healthcare Professional's view. You will log in to the system using the credentials provided. I will then ask about your initial opinion of the site before asking you to complete a short series of tasks. Once that is complete, we will do the same again with the patient view.

Do you have any questions at this stage?

Let's begin.

Usability Test – Healthcare Professional Account

1. Can you start by logging in to the system using these details:
   Username: 0987654321
   Password: password
2. What are your initial impressions of the main site page? Where do you think you would click first?
3. Can you view a calendar showing the Healthcare Professional's appointments?
4. Can you see more information about a particular appointment?
5. Are you able to bring up a list of patients?
6. Can you view more detailed information about the first patient?
7. Can you bring up a graph showing the readings of any kind made by that patient?
8. How many types of medication does that patient take?
9. Can you view the patient's calendar?
10. Can you create a new account for a patient using the following criteria:
    Condition: Type 1 Diabetes
    Medication:
    - Humalog. **Dosage**: 15u/ml **Frequency**: 3
    - MetforminSR: **Dosage**: 1 x 150mg **Frequency**: 2
11. Can you set up an appointment for a patient using the following criteria:
    Patient ID: 1234567890
    Date: 7th May 2015
    Time: 2:00pm
    Location: Antrim Area Hospital
    Clinic: Diabetes

Thank you. What was your overall experience using the Healthcare Professional account? Do you feel there were improvements that could be made?

Thank you. We will now move on to testing the Patient account. Please log out of the Healthcare Professional Account.

Usability Test – Patient Account

1. Can you start by logging in to the system using these details:
   Username: 1234567890
   Password: password1
2. What are your initial impressions of the main site page? Where do you think you would click first?
3. Can you please name the patient's next of kin.
4. Can you record a reading for the patient's weight using the following criteria:
   Date: 4th May 2015
   Time: 10:30am
   Weight: 152 lbs
5. Can you tell me how many steps the patient recorded for the INSERT DATE?
6. What medication does that patient administer via injection?
7. Can you show details for a patient's appointment on INSERT DATE?
8. Can you record a blood glucose reading using the following criteria:
   Date: 4th May 2015
   Time: 7:45pm
   Reading: 9.2 mmol/L
9. What was the patient's insulin intake on INSERT DATE AND TIME?
10. What was the patient's blood pressure on INSERT DATE AND TIME?
11. What medication does the patient take for blood pressure?

Thank you. What was your overall experience using the Patient's account? Do you feel there were improvements that could be made?

Thank you for you help in completing the usability tests.

Usability Testing: Results

HCP Account

Task	Results
Where do you think you would click first?	All testers stated it was dependent on what task they wanted to perform. Some suggested View Patients, or the HCP appointments
Can you view a calendar showing the Healthcare Professional's appointments?	All testers completed task
Can you see more information about a particular appointment?	One tester did not know to click on appointment to view more information
Are you able to bring up a list of patients?	All testers completed task
Can you view more detailed information about the first patient?	All testers completed task
Can you bring up a graph showing the readings of any kind made by that patient?	All testers completed task
How many types of medication does that patient take?	All testers completed task
Can you view the patient's calendar?	One tester did not complete, another took considerable period of time to completed
Can you create a new account for a patient…	All testers completed task. One tester initially entered dosage and frequency in same input field
Can you set up an appointment for a patient…	All testers completed task. One tester typed in date rather than using the date picker

Patient Account

Task	Results
Where do you think you would click first?	All testers stated it was dependent on what task they wanted to perform. Some suggested Record Reading, a condition module, or Medication
Can you please name the patient's next of kin.	All testers completed task
Can you record a reading for the patient's weight…	All testers completed task. Two testers added units to reading
Can you tell me how many steps the patient recorded for the 10th April 2015?	One tester tried to use record reading form. Came up with figure of 152
What medication does that patient administer via injection?	All testers completed task.
Can you show details for a patient's appointment on 24th April 2015?	Two testers did not click appointment to view information
Can you record a blood glucose reading…	One tester made another weight reading. One tester typed in units. One tester had problems with the timepicker.
What was the patient's insulin intake on 27th February 2015 at 14:35?	All testers completed task
What was the patient's blood pressure on 16th March 2015 at 20:10?	One tester could not complete. Two testers attempted to use record reading form first.
What medication does the patient take for blood pressure?	All testers completed task.