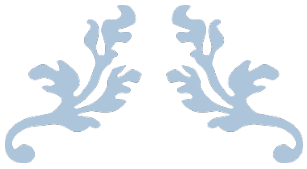


AN MDT WEB APPLICATION FOR UCLH

A PEACH Industry Project

AUTHOR:

JOSHUA HALE

SUPERVISOR:

DR. GRAHAM ROBERTS

SEPTEMBER 1, 2016

COMPUTER SCIENCE MSC SUMMER PROJECT

Abstract

The initial phase of this project was to design and create a library of components to be used by the PEACH team in development. The main phase of this project and bulk of this report is then to explore the concept of constructing a web application for use in multidisciplinary teams at UCLH to aid in the treatment of cancer patients. The MDT application is to be one of many applications built under the PEACH team and will be taken forward by future students at UCL.

The application back end was configured in Docker containers using OpenEMPI and OpenEHR servers. The application front end has been built using ReactJS, Redux, CSS3 and multiple open source JavaScript libraries. The author had not used any of these technologies before however has learnt a substantial amount during this project.

The design guide and components library was fully achieved. The MDT application has an authentication system along with referral form example to be taken forward to future students. A proof of concept has been developed and a discussion with how to take the project forward for future students has been held with the client.

Most of the aims and goals have been achieved during the project and can be considered a success.

Acknowledgements

I firstly want to thank my supervisor **Dr. Graham Roberts** for providing me with a great set of tools from my COMPGC22 class. These became invaluable during the project. Thank you for supporting me during my project and guiding me in the right direction.

I would like to thank **Alexandra Silver** for taking the time to read and mark this dissertation.

Dr. Dean Mohamedally, I would like to give thanks to you for persuading me to join and stick with the PEACH team. Without you I would not have been provided with such an opportunity.

I would also like to thank **Dr. Navin Ramachandran.** You have been such a fantastic client to work with. Your support and understanding throughout has been greatly appreciated.

I want to thank my **family** and **friends** for providing me with support and encouragement at times of need.

Finally, I want to thank my two dogs, **Digby** and **Charlie,** for keeping me company during the lonely, work-filled nights.

Table of Contents

1 Introduction

1.1 Problem Statement

Every cancer patient is discussed by a team of relevant specialists, to make sure that all available treatment options are considered for each patient [1]. This team of relevant specialists is known as a multidisciplinary team (MDT). Cancer MDT's were established to overcome many problems: (1) cancer treatments being delivered by generalists without the necessary knowledge and skills related to a specific cancer; (2) some factors relevant to decision making were being missed due to staff working in isolation; (3) information was not being collated thus hampering the onward flow of data to cancer registries [2].

As technology has advanced, applications to assist with the management of an MDT have been utilized. At UCLH, they currently use a system known as Infoflex[1] which is slow and complex. Whilst referring to a video demonstration of a more modern MDT application, Ben Goretzki, Cancer Service Project Manager at UCLH, mentioned that "…playing it back at 0.25x would still be a step up from what we currently have.".

Not only is the current system slow and outdated, the user interface is poorly designed. This poor design of UI leads to poor data input from the users and thus a lack of information to be used in making decisions. Constantine points out that the reality is that a good user interface allows people who understand the problem domain to work with the application without having to read the manuals or receive training [4]. Applying this quote to the current Infoflex system it can be seen that the user interface is clearly not acceptable. It is also important to remember that it won't matter how technically superior your software is or what functionality it provides, if your users don't like it they simply won't use it [5].

Thus the project is to redesign and build a new MDT application using more modern technologies. This will hopefully provide a system for the MDT whereby response time is diminished and input of *reliable* data is increased. These technologies will be discussed further in section 2.3.

This is a very exciting project to be a part of due to many reasons. Firstly, the redesign phase will provide an opportunity to create a consistent design guide booklet to be

[1] http://www.infoflex-cims.co.uk/

used by other PEACH sub-teams. The MDT team is part of PEACH which is a platform for e-health in collaboration with Microsoft and NHS. Secondly, opportunities to work in collaboration with these organizations are a rarity thus adding to the excitement of the project. And thirdly, this project is at the forefront of improving cancer patient treatment, something that affects nearly all of us either directly or indirectly. There will be 23.6 million new cases of cancer each year by 2030 worldwide (estimated) [6] and so it is enthusing to be part of a team whereby treatment could be enhanced by the work undertaken during this project.

It must be mentioned that the size and scale of the MDT application is extremely large. Throughout this project it must be assumed that the application is a proof of concept that will in turn be passed on to UCL students at the end of September 2016. The client is not expecting a fully functioning application and in fact in the time frame given this was determined as unfeasible.

1.2 Aims and Goals

1.2.1 Aims

During this project the author intends to achieve:

- Learn web development fundamentals
- Learn ES6 syntax, React, Redux, Webpack, Node (Express) in order to keep up with current hot trends in building web applications
- Understand Docker fundamentals
- Learn good design principles for UI/UX

1.2.2 Goals

During this project the author intends to deliver:

- An open source brand guidelines document to be used by sub-teams in MDT
- A library of components to be used by sub-teams in MDT
- A development environment to be used by future students in the development of MDT application
- Designs of user interface ideas for MDT application
- A mock authentication system using Node, Express and MongoDB
- A proof of concept referral form to refer patients to MDT meetings

- A document to be passed on to the client detailing future development

1.3 Overview of Approach

This project dedicated the initial 5 weeks on developing a user interface library to be used across all PEACH applications, along with a brand guidelines document. The client wanted this library developed to keep front-end applications appearing consistent under the PEACH brand.

"Design is not just what it looks like and feels like. Design is how it works."

-Steve Jobs

A brand guideline document was also an important requirement for the client and needed to be completed before moving onto the MDT application. As Steve Jobs mentioned, it is important to detail how the design works and we achieved this by providing these guidelines to the other PEACH teams.

We went about this by forming a team of 5 and had meetings every other day in order to design logos, choose fonts, select colours and publish a design guide. The team kept in close communication with the client via bi-weekly meetings and Slack. The approach taken in these initial 5 weeks was an iterative approach. The team would take into account current NHS design guidelines, build new components with these in mind and present them to professionals at UCLH. After deliberation, the team would decide to meet again and re-design components, logos, fonts and colours. This cycle repeated until the client was happy with the deliverable. The author then could focus on the main phase of his dissertation, MDT application.

The author applied a use case driven approach to developing the MDT application. This meant that the project team used the use cases to drive all development work, from initial gathering and negotiation of requirements through to implementation of code [7]. As outlined by Arlow and Neustadt [8] the five stages of the Unified Process are: (1) requirements; (2) analysis; (3) design; (4) implementation; (5) testing. The author adhered to these stages whilst undertaking the project as it allowed iterative practices to be used. This meant the author was able to learn one technology at a time and develop a working part. From here, the author could increment upon the working

section when another new technology was learnt. The client was happy with this approach as they were able to see incremental improvements in design and implementation.

1.4 Structure of Report

The first part of this report has introduced the problem, aims and goals of the author and what is to be achieved. An overview of how the project was handled has also been given.

The report then moves on to describing current systems at UCLH and how other companies have tackled the problem by building similar MDT web applications to the one built at PEACH. The author will go on to discuss which technologies were used and why they were chosen.

In the third section of this report the author moves on to requirements gathering and analysis of these requirements. Use cases will be referenced along with user stories and other analysis tools. Data modelling will be briefly touched upon as this was mainly undertaken by the UCLH themselves.

The largest section of the report is the Design section. This is where the author will describe how the application architecture looks and where the MDT application fits into the PEACH architecture as a whole. A detailed description of each component is then analysed. The storage representation of demographics and patient health information will be discussed. Separation of concern with regards to back-end and front-end will be touched upon along with a more detailed analysis of some interesting implementation details with the front-end stack. The author will spend some time reflecting on why these design choices were a good choice.

The penultimate section is dedicated to testing. The author is an avid tester and agrees with Jimmy Nilsson that TDD is a key approach to writing cleaner, simpler and more robust code [9]. The author will discuss a cutting edge approach used during testing known as time-travel debugging. An overview of a piece of middleware, written by the author, which provided logging information will be presented. Finally, the Mocha testing suite for unit tests will be presented along with some sample tests.

The last section is a conclusion and project evaluation. The author will be critical and focus on where improvements could be made. The author will demonstrate goals that

have been achieved along with some final thoughts on the overall project. A document will be produced which shall be handed to future developers in order to maintain a smooth transition of handover. This shall enable the future developers to commence with developing a lot quicker.

2 Context

Before starting to gather requirements, it is crucial to fully research the area in which this project is set. Since the application is in the e-health industry and will be gathering patient data the author initially decided to research around the information standards within the sector. This will be discussed in section 2.1 Information Standards. Research into e-health software that is available is discussed in section 2.2. In section 2.3 the author presents information on multidisciplinary team studies, the usefulness of MDT's and other applications that have been developed to assist MDT's. Section 2.4 will briefly touch upon why data collection, standardization across applications and analytics is crucial to the validity of e-health informatics. Sections 2.6 and 2.7 look more towards software trends, web architectures, design patterns and a thorough comparison of technologies. Introductions into chosen libraries and frameworks will be presented, along with reasoning and critical evaluations.

2.1 Information Standards

Information standards are an agreed set of rules, a consistent method or process for capturing, processing, managing and sharing of data and information [10]. The author had been presented current information standards used in UCLH by two professionals from the hospital. The reason that data must be standardized is extremely important in the current world as the ability to extract information from the data is less costly and more valid. Unfortunately, data has not always been standardized across all hospitals, this leads to multiple dictionaries being created and mappings between terms. An example of where information standards show their importance is as follows: A doctor in the Accident and Emergency (A&E) department of a hospital will share a patient's diagnosis of a fractured left femur with the patient's General Practitioner (GP) and perhaps a physiotherapist in another location. The doctor in A&E, the GP and the physiotherapist use different information technology systems, with different ways of recording this data. They all need to communicate securely, using a common terminology so that there is no misinterpretation of who the patient is and what the diagnosis and treatment were [10].

2.1.1 SNOMED CT

SNOMED is comprehensive, multilingual clinical healthcare terminology product, owned and distributed around the World by The International Health Terminology Standards Development Organisation (IHTSDO) [12]. The author felt that looking into SNOMED CT would prove to be a wise decision because it would enable the clinical health record of the individual to be benefitted by: (1) allowing accurate and comprehensive searches; (2) remove language barriers if database were to be used elsewhere; (3) enabling relevant clinical information to be recorded consistently. SNOMED CT would help when implementing the forms outlined in the requirements whereby patient information would be collected. It would help to keep consistency which is a primary requirement of all PEACH applications by the client. Due to SNOMED CT providing a standardization in terminology, this could help benefit the population too. The availability of standardized digital patient data is presenting unprecedented opportunities for the aggregation and mining of this data [11]. Thus by providing the ability in the front-end to capture correct, consistent and relevant patient data we could potentially use a single database to store patient data across PEACH applications.

2.1.2 Data Protection

It is worth noting that due to the privacy concerns of the project, the author was careful to partake in data protection courses online. After researching around the topic of dealing with private data in the NHS the decision was taken to complete multiple courses and gain certificates. The courses undertaken were as follows:

- Introduction to Information Governance [figure a.1]
- Access to information and information sharing in the NHS [figure a.2]
- Information Governance: The Beginner's Guide [figure a.3]
- Secure Handling of Confidential Information [figure a.4]
- Research Data and Confidentiality e-learning course [figure a.5]

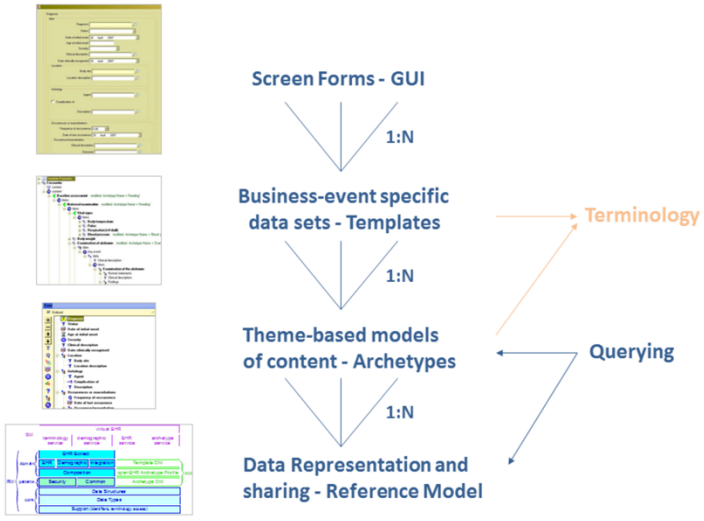
After completion of these courses, the author felt comfortable dealing with confidential data in UCLH.

2.2 E-health research

2.2.1 OpenEHR

OpenEHR is a virtual community working on means of turning health data from the physical form into electronic form and ensuring universal interoperability among all forms of electronic data. The primary focus of its endeavour is on electronic health records (EHR) and related systems [13]. OpenEHR became of interest to the team after being pushed to look into it by the client. The clients at UCLH wanted to use OpenEHR as the platform for developing the application as they had attended conferences and were avid supporters of the software.

The way OpenEHR works is a multi-level, single source modelling approach in which the models are built by domain experts. Models were created by the client and other medical professionals. These models were then delivered to us to be stored for later usage.



The figure to the left shows the multi-level approach to OpenEHR. This would be beneficial to use in development of the application because it allows us to let the domain experts (client) deal with the lower level querying and modelling. This enables us as developers to move forward with implementing the higher levels such as GUI once the client delivers us created templates.

Once comfortable that OpenEHR provided the team with all capabilities to meet user requirements in terms of data modelling and storing we moved ahead with using the OpenEHR platform.

2.2.2 Ethercis

OpenEHR is a very complex system that is difficult to learn given the timeframe we had. Thus we set out to find a platform to help flatten the learning curve. After much research, we discovered EtherCIS, an open source OpenEHR server [14]. EtherCIS is still a development in progress and is actually in BETA mode. We decided, along with the client, that this platform would be a good tool to use since it solved the issue of learning OpenEHR from scratch in a limited time frame. The Github repo was constantly updated so was an active project.

The design of EtherCIS allowed simple interactions with clients using a RESTful API. This was an additional bonus since the author had a small prior experience with using RESTful API's earlier in the year.

Clinical data would be exchanged using FLAT JSON which the whole team was comfortable with using. Data is also able to be persisted in a database and we decided to go ahead with PostgreSQL as the documentation for implementing this was more substantial.

2.3 MDT Research

The author decided to research why MDTs were so important in the treatment of cancer patients. By understanding this, the author felt it would benefit the overall design of the MDT application. This is because the problems that MDTs are solving could be enhanced directly by designing the application in the correct way. As discussed in the problem statement in section 1 there were many reasons for the need of MDTs.

2.3.1 Effective MDTs

If the author knows how an MDT can be made effective, then perhaps the application could benefit by designing these concepts directly into it. The author discovered that an effective MDT should result in: (1) patients being offered the opportunity to enter clinical trials; (2) patients being assessed and offered level of information and support needed to cope with their condition; (3) good data collection to benefit patient, audit and research. This list is not exhaustive however they apply directly to discussion held with the client.

Another PEACH team were developing a clinical trials application and so discussions were held to somehow integrate the MDT application into their clinical trials search. An additional PEACH team were developing a chat bot and so discussions were held with them regarding the help of patient wellbeing regarding some conversational UI. Finally, a good data collection to benefit the patient, audit and research was crucial. This meant UI had to be a key aspect in deliverables for the client. Capturing patient information and avoiding data 'holes' would prove a difficult task.

2.3.2 Other Applications

The Christie, based in Manchester are leading experts in cancer care, research and education. They have developed an MDT application to assist in the treatment pathway of their patients [see section B appendices].

This application comes with a step by step tutorial guide in webcasts. To understand more fully the goals of an MDT application this came in very handy. The application was essentially what the client wanted building, however, wanted a UI overhaul compared to The Christie application.

MDTech developed an MDT scheduler that helps maintain MDT meetings and schedules them. This application is a standalone application however the client mentioned wanting a scheduler built in to our MDT platform. This scheduler was not free and so unfortunately the author was unable to download and use the system to see what features could be implemented in their own platform.

SILVERLINK Software are a company that provide the NHS with patient administration systems (PAS). They are the largest provider in the NHS of these PAS and are currently deployed in 45 hospitals in the UK [15]. After looking into the products they create, one of them stood out that matched many requirements of solving the client's problem. This product is called Cancer Plan[2] and delivers a huge feature list that would solve the clients problem. Of course, integrating into own hospitals would take time and money. Developing a tailor made system would still incur these problems.

Thus the Cancer Plan software seems to be the best application available currently developed.

[2] http://www.cancerplan.co.uk/

2.3.3 Differentiation of PEACH MDT Application

It would be necessary to briefly touch upon why the author does not go with configuring one of the applications mentioned above into the UCLH. After talks with the client, an in-house developed MDT application was the best solution to the problems due to many reasons. The primary reason is the UI, branding and integration with other PEACH projects. It is all well and good installing and configuring a stand alone application into the UCLH like the aforementioned products, however this could have proved very difficult when hooking into the main PEACH architecture which will be looked at in a later chapter.

The PEACH MDT application was also to follow a brand guidelines booklet that had been developed by the author in an earlier phase. Thus by deciding to use a pre-existing application, work would have had to been carried out on existing code base. In previous projects, working on existing code base takes time especially when not developed by yourself, or if the code had not been documented cleanly.

The PEACH MDT application had a curated set of specific requirements and the applications mentioned in section 2.3.2 would have ticked these off. However, these applications have not been developed with PEACH or UCLH alone in mind and so may have too many or too few features. This could lead to paying for features of a product that are not going to be used by the users in any case.

2.4 Current System Overview

2.4.1 Infoflex

The author was invited into The MacMillan Cancer Center in Euston to review the current system in place. The presentation was given by the MDT Coordinator, Ben Goretzki. The current system was slow, outdated and in desperate need of an overhaul. It was at this presentation the author learnt that every trust is mandated to collect and submit to Public Health England. They also all mostly fail to do this, according to Ben. This means the current system in place is failing to capture data and valid information, thus leading to poorly collected data at Public Health England. Due to this, trends and patterns in national health cannot be reliably determined.

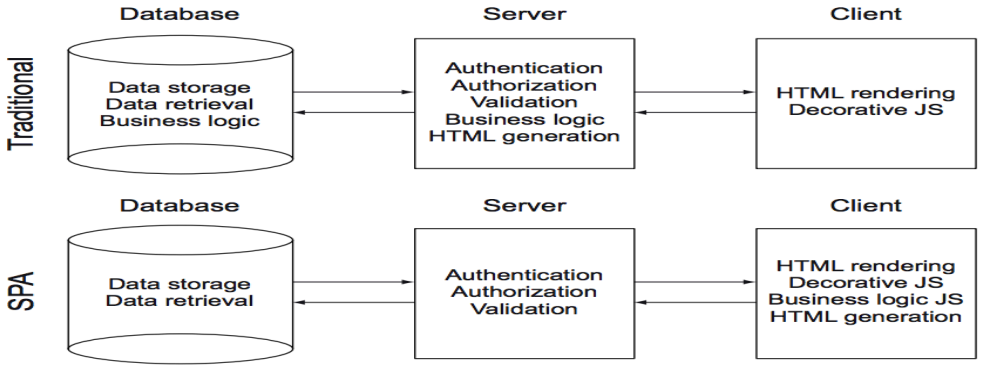
The author set out on the goal to understand *why* the data was not being captured by clinicians in the current Infoflex system.

Much of the data required for COSD will be either recorded or confirmed at the MDT meeting so use of the electronic system live at the meeting is key [16]. The current system only allowed the MDT Coordinator to add notes during a live MDT. This meant that huge amounts of data were being lost during the MDT presentations. Thus a live MDT input was needed to capture data on the fly. Infoflex UI was deemed extremely complex and it did not help that a lot of the clinicians are not willing to spend a long time learning the UI. This leads to user demotivation and thus failure to input all patient data was apparent. To solve this, UI was a key aspect of the project and redesigning forms for better data capture.

2.5 Current Web Technologies

2.5.1 Single Page Applications

Single page applications are web applications that load a HTML page once and from there they dynamically make updates to the apps pages as the user interacts with the app.



Traditional vs SPA architecture [17]

As shown in the image above, SPA place more emphasis of business logic and HTML generation on the client side. The author believes this architecture would be a good design choice for the MDT application to take load off servers in the UCLH and let browsers handle more of the processing. Also, with more emphasis of a push to develop SPA in the modern world the previously thought disadvantages are now less common. For example, a known disadvantage of SPA was a requirement of the client to enable JavaScript. In this modern day of computing, almost everybody enables JavaScript now and in UCLH as was confirmed. Having worked on a traditional application architecture before the author decided, along with the rest of the team, to

move forward in developing a SPA for the client. This would enable us to learn new technology design patterns too.

2.6 Tools Used and Why

Here the author will introduce the libraries and software used in developing the MDT application. A brief comparison of similar technologies will be presented along with a reasoning as to why the decision to use one over the other.

2.6.1 React, Angular and Elm

Having previous experience developing in Angular the team made a decision to use this project as an opportunity to learn a new framework. The author had never worked with React before, but after learning functional programming at UCL, had a keen interest in developing in React. This is because functional programming concepts are inherent in the development of React applications. Also, if an application is built from pure functions, this enables us to more easily reason about the behaviour of an application, and so making it less likely for unpredictable behaviour to occur. React will enable us to put together simple, easily understandable applications by creating composable sections to be passed down to future students.

The author looked into a much stricter functional framework called Elm which is again a front-end framework with cues taken from Haskell [18]. It would have been too niche a framework to develop the application. With lack of examples and tutorials available the team decided it be best to start development in React if we were to go with the functional style UI.

Angular was considered briefly however nobody was comfortable with the newer v2.0. Angular 2.0 actually stripped the two-way data binding in favour of the uni-directional approach so we could have gone with development in this. Angular is really a whole MVC framework whereas React is just the 'V' and so it was decided that if we went ahead with less to begin with we could slowly build up the libraries imported as needed. By using smaller libraries such as React we can essentially 'pick and mix' to create our own development environment. Although this is possible with Angular 2.0, extra baggage comes along with it that potentially we would not have grasped given the time frame.

2.6.2 Redux, Flux and MVC

Redux is a predictable state container for JavaScript applications. We decided to go ahead with using Redux as the application state container. The team could have just as easily decided to use Facebooks Flux however after some deliberation the reason for choosing Redux are as follows: (1) developer experience; (2) ecosystem; (3) simplicity. In Flux stores, the data and functions are both intertwined. This prevents being able to refresh the code on the fly without also refreshing the data, or state. However, after researching into Redux, we can see that the author, Dan Abramov has used the separation of concerns pattern and pulled the data and functions into separate sections. This allows for a very neat ability to alter reducer code on the fly without altering state of the application thus saving all important time on this project.
The ecosystem of Redux is growing at an extremely fast pace and thus many tools and libraries were developed for us to use throughout the project via open source projects. Research revealed that the reason Redux was growing so quickly was due to the ability of Redux to provide extension points such as middleware [20]. This ability to hook in and create own middleware proved invaluable to the author during implementation stages.
Finally, it was decided early on in the process to not follow the MVC design pattern and instead move forward with a more unidirectional data flow with composable user interfaces. The team believed this would help in determining the source of bugs in the application. Changes in models in MVC style can cause some undeterminable behaviour in the application and it is not unusual to spend an unreasonable amount of time trying to locate the source of the bug. However, the author feels with the functional style and unidirectional data flow this time can be significantly reduced.

2.6.3 Webpack, Gulp and Browserify

The team decided to use Webpack as the build tool for the application. Assets, JavaScript, images, fonts and CSS are all put into a dependency graph by Webpack. We decided to use Webpack because the application is at an industry scale and extremely complex thus the benefits would outweigh the complexity and time taken to learn it, especially for future developers. Also, Webpack is the main build tool implemented within the React community and so the team were able to find a significant amount of more help and documentation when using with React tools.

Gulp was considered however it is only a build tool at the end of the day. This meant that including plugins such as for JSX syntax, Babel to enable ES6 syntax and others would have been more difficult having to use transforms. Browserify also was not a viable option because we wanted to use the react-hot-loader[3] library to increase productivity in the implementation stages. This is only available with using Webpack for builds. It should be noted that LiveReactload[4] was an option to use with Browserify to achieve the same goals as react-hot-loader however a community wave was moving with React/Webpack and so we decided along that route.

2.6.4 ES5 and ES6

Since this project were to be handed down to future students we felt that all development achieved now should be with the interest of the future developers in mind. This meant that sacrificing our time to learn new ES6 syntax would be worth it in the long run for the client. Since future students will be more accustomed to ES6 and would almost expect to be developing new apps in ES6 syntax we decided to use this.

The team had never worked with ES6 before however it would also prove to benefit us in terms of improving our desirability in employment by learning the new syntax. Since not all browsers support the new syntax we had to find a way of transforming the written ES6 into ES5 to be executed by the browser. The author discovered a fantastic compiler called Babel which allowed the team to develop in ES6 and transpile down to ES5 using a Webpack loader called babel-loader[5].

During the development the author made use of a Babel REPL[6] to test ES6 syntax vs ES5 syntax. This proved to be an extremely helpful tool whilst learning the new syntax.

2.6.5 Peachstrap, Material-UI and Semantic-UI

Since a significant amount of time was spent developing the Peachstrap framework we decided to use this. The team were not sure of which technology was to be used in developing the MDT application until after the brand guidelines were finished. This

[3] https://github.com/gaearon/react-hot-loader
[4] https://github.com/milankinen/livereactload
[5] https://github.com/babel/babel-loader
[6] https://babeljs.io/repl/

was unfortunate because if we had known that React were to be used then the team would have preferred to use a Material-UI style library.

Peachstrap enabled the team to develop with responsive web design in mind. Peachstrap allows developers to utilize the grid based system in order to achieve a greater user experience on all devices. Wireframing tools such as Webflow[7] and PowerPoint seemed like viable options to mock up designs for the client. They enable users to create user interfaces without any logic being executed. This enables the user to get a feel for the application and was extremely important in the case of the MDT application.

2.7 Useful Libraries

During development the author spent a long time on deciding which libraries would help speed up development in the long run. Also which libraries would prove useful for meeting user requirements in a faster time. The development environment would be set up using a package.json file where the team could list dependencies, development dependencies, scripts among many other key values.

Below are just a select few of libraries chosen to help development and future students development of the MDT application.

2.7.1 React-Router-Redux

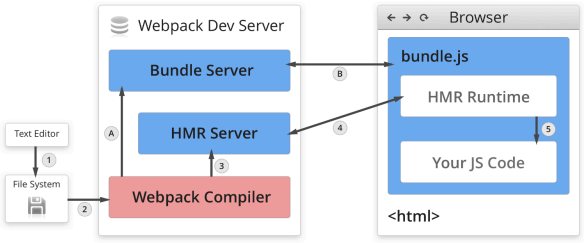
This library allowed the author to develop the single page application with react-router and Redux in sync. The use of react-router-redux allowed the author to implement time travel debugging (see section 6) whilst navigating between pages when actions were replayed. The author could have just stuck with react-router however when testing this additional feature proved extremely helpful.

2.7.2 Hot Module Replacement

This feature was implemented into the development environment using Webpack. This took a substantial amount of time to understand how to configure, however it will prove invaluable to the long term development of future students as productivity increases dramatically with this enabled. The prevention of having to reload pages on each save and refresh page losing state helps increase development efficiency, thus delivering the clients requirements in a shorter space of time on each iteration.

[7] https://webflow.com/

The figure below shows a quick overview of how Hot Module Replacement works.



Hot Module Replacement Architecture [20]

For a more detailed description of the above figure it is worth viewing the article "Understanding Webpack HMR" found online [20].

2.7.3 Redux-Thunk

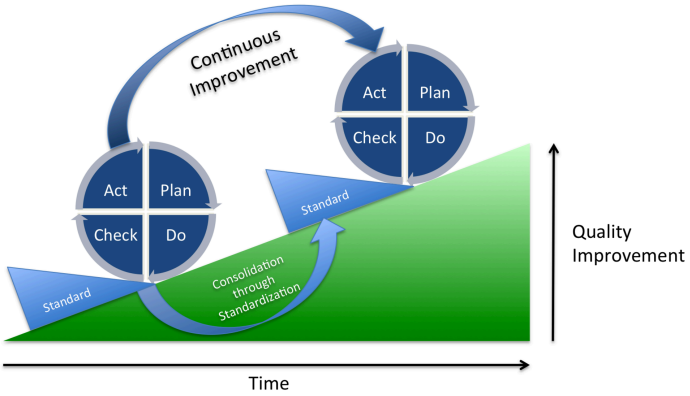
This middleware allows developers to write action creators that return a function instead of an action. Thunks in software engineering simulate lazy evaluation, a concept first encountered by the author in functional programming at UCL. They delay the function argument computation [21] which is a very useful feature in Redux especially for dispatching actions on conditional statements. The automation of dispatching is removed in redux-thunk thus allowing the author to wait for events such as asynchronous calls before dispatching a certain action. This library seemed crucial to use because it enabled us to gain access to the dispatch function and thus dispatch actions conditionally.

2.8 Shewhart Cycle (PDSA)

In terms of management methods used by the author, he felt it was extremely important to manage the project using the Shewhart Cycle [19]. W. Edwards Deming is the founder of the PDSA Cycle for Learning and Improvement whereby the PDSA stands for Plan-Do-Check-Act. It is a four-step iterative management method used by the author in managing his own goals and targets during the project.

Planning allowed the author to establish the objectives and processes necessary to deliver results in accordance with the expected goals. Following this, the author was able to implement on the plan and actually make the component. Studying the actual results using test and comparisons against expectations, the author was able to iterate on the component in the acting stage.

The cycle enables the author to improve quality over time continuously and the figure below visualizes this:



Continuous quality improvement with PDCA [19]

As shown here the standards of the author increased over time displayed by the quality improvement gradient. Continuous improvement was made by following the cycle during the project.

3 Requirements and Analysis

Before starting the design and implementation of the application it is important to understand the requirements fully. The approach taken to gather user requirements consisted of interviewing the client, meeting with users of the current application and developing use cases, user stories and personas. Due to the size and complexity of the application, an extremely thorough requirements gathering took place.

3.1 Detailed Problem Statement

A detailed problem statement has been presented in section 1. Upon reaching this stage of the project there had been no changes in the fundamental problem.

3.2 Structured List of Requirements

After meeting with the client the team developed a base set of requirements. These are shown in figure c.1. We developed this list of requirements by splitting the overall user requirement of an application into several sub-components. These components were comprised of:

- etherCIS
- openEMPI
- user authentication
- "MDT Hub" web app
- Patient Profile
- Meeting Scheduler
- Meeting Mode
- MDT Referral Form
- MDT Prep Form
- Form Creator

Within each sub-component the requirements were refined into smaller more manageable tasks. Using the MoSCoW method, the team developed an understanding with the client in what importance each deliverable had. This prioritization method was developed by Dai Clegg[21] and was used by the team to try and integrate our agile approach into a rapid application development approach too. This would be due to each sprint being devoted to a sub-component identified above and the client

would be able to see the application slowly come together. Below shows the ratio of requirement types:

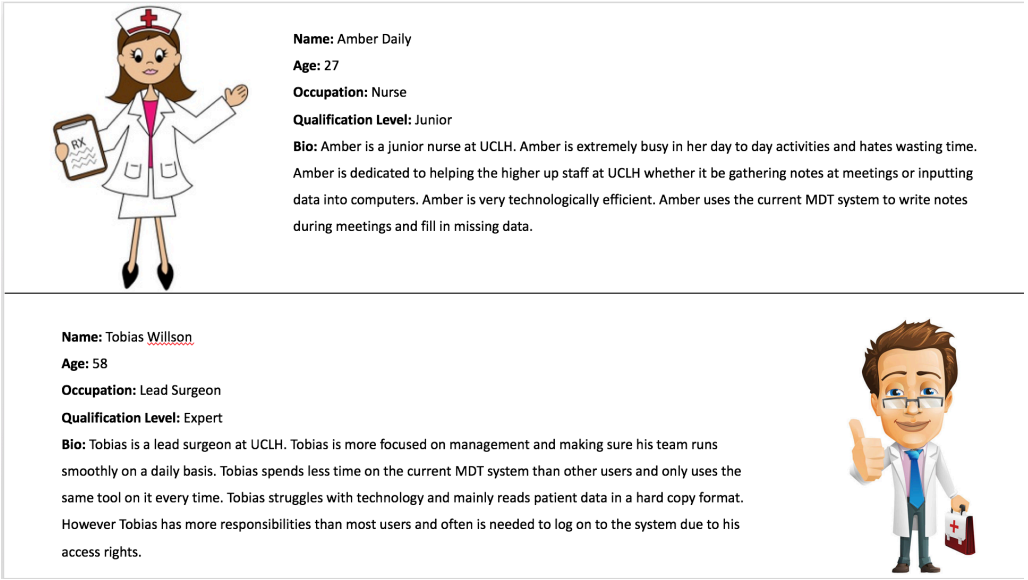
Category	Amount
Must Have	30
Could Have	6
Should Have	4

The requirements were finally curated into a document which explained each sub-component in a lot more detail. This document proved invaluable to the team because it enabled a finer understanding of each requirement before development. These can be found in figures c.2 in the appendices.

3.3 Personas

First introduced by Alan Cooper, a persona defines an archetypical user of a system [23]. The typical kind of user of the application will be outlined by the persona. This is extremely helpful prior to designing and developing the application especially in the case of this project. The reason for this is because understanding the roles of specialists such as Radiologists and MDT Coordinators is a difficult task without visualizing and creating these personas.

Unlike in section 3.4 these personas are not actors and in fact are archetypical instances of the actors presented there. These personas prove beneficial to the author as it was difficult at times to meet with the actual users themselves.



3.4 User Stories

It is difficult to understand as a developer the requirements of the MDT application without visually seeing an MDT in practice. Terminology in the medical profession is often complex and difficult to comprehend by someone that is not in the field. Thus the team decided to further abstract from use cases and develop user stories to understand the role of each user of the application. The roles were split into: (1) MDT Coordinator; (2) Referrer; (3) General Clinician; (4) MDT Chair; (5) Admin, each with different responsibilities and use cases.

Figure c.3 in the appendices shows how the team developed user stories by splitting the application into separate pages. Initially, we set the general principles out whereby the general workflow of the pages is presented. By using the Keep It Simple Stupid (KISS) [24] principle we developed an easily readable set of user stories that are crucial in design and development. These user stories really helped the author develop a solid understanding of what the client wanted and what the users needed.

3.5 Use Case Diagram

The author created a use case diagram to show different titled use cases and which actors played a part in them. A finer list of use cases is shown in the appendices.

3.6 Workflow Diagrams

Activity diagrams were created by the team in order to see the flow of the application. This really helped develop a finer understanding of activities performed by users and the system.

The first image shows a general flow of the application and how different roles are interacting with different parts of the system.

The second image shows another possible workflow whereby a decision node is shown which checks whether the patient is registered. No roles are depicted in this flow as any user will be capable of each node.

The final image depicts the workflow of creating an MDT meeting along with notifications being sent.

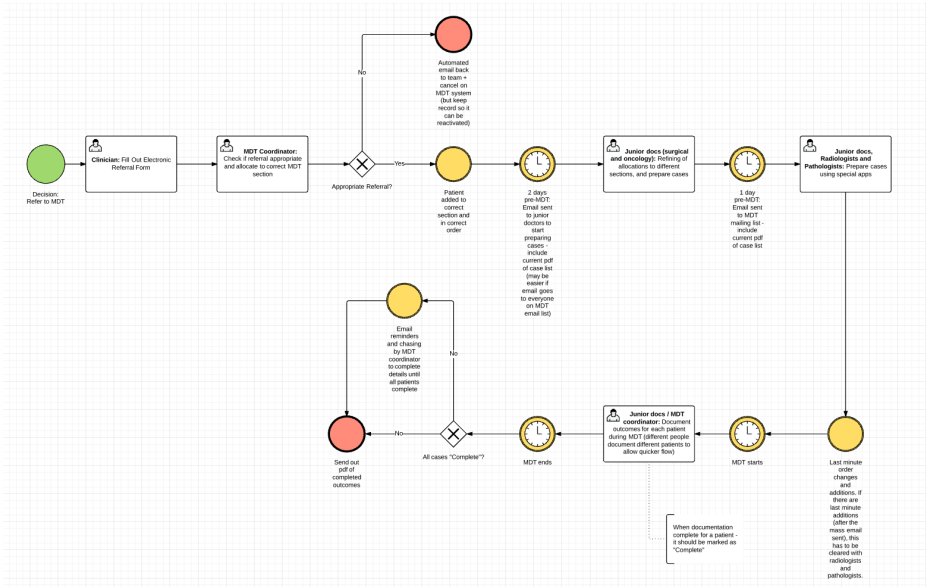


Figure showing general flow of application

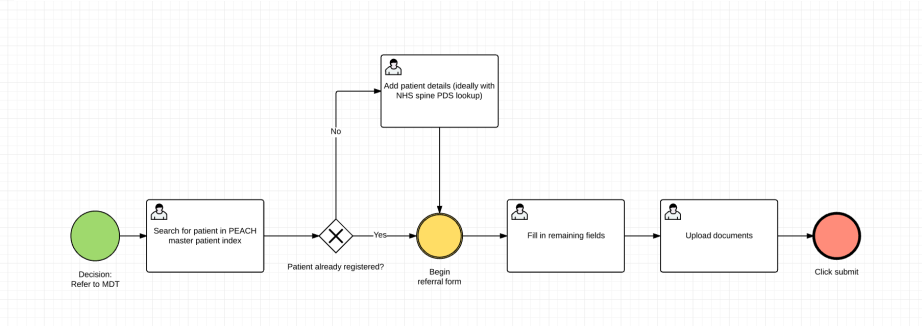


Figure showing general flow of MDT referral

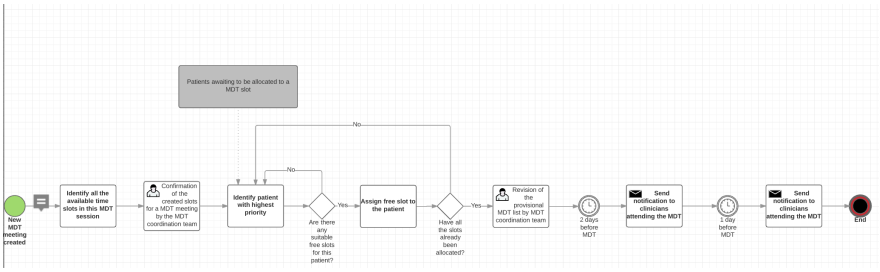


Figure showing creation of MDT meetings along with notification points

3.7 Results of Analysing Requirements

From analysing the requirements, mock-ups of user interface ideas were presented to the client. The team started to think about the overall architecture of the application, for example determining which databases would be needed to be implemented. Since the requirements were to use openEMPI as the server for patient *medical* data we still needed to think about implementing a database for demographics.

The requirements gathering stage was extensive on this project and were constantly changing. This was not a problem since the team were using the agile approach throughout and so constant communication with client was happening.

It was at this section of the project the author realised the scale and complexity of the application to be used at UCLH. We decided on the web views needed and these can be seen in the curated requirements document in the appendices figure c.2.

The team also developed some mock-ups to present to the client using the requirements given, these can be seen in figure c.4.

One issue the author could see developing from the requirements analysis is the fact that this is an industry scaled application and none of the team had worked on these before. Time was also running short for the project and so it soon came clear that a proof of concept was to be delivered and a more thorough development process would be tackled by future developers. The author saw fit to develop an authentication system and dynamic roles based routing from the requirements analysis. Also an attempt at hooking up an MDT referral form would be developed.

4 Design and Implementation

Since a significant portion of time was spent on developing a design guide and brand booklet for PEACH the author will firstly discuss the implementation details of this. After this, the high level architectures are discussed of PEACH and MDT application. The author will then discuss implementation and design decisions regarding development of the MDT application.

Finally, I will discuss why these design decisions proved to be good decisions and how they will benefit future developers of MDT application.

4.1 Design Guide

Since PEACH is a new project the client requested a new user interface and design guidelines booklet be developed and published. The author decided to take active involvement in this sub task.

The decision to build on top of Bootstrap[8] came about via many discussions between the UI team and the client. To make the library available along with files for fonts and icons the author packaged the components and published to Node Package Manager (NPM).

4.1.1 Logo

The logos for PEACH were developed through an iterative approach by the UI team. The team decided to meet every few days starting with a whiteboard, the team collaborated with each other sharing ideas for the PEACH platform.

A range of logo ideas can be seen in the appendices figure D.1. These logos were developed using tools such as GIMP and were presented to the client via face to face meetings. Eventually, the client decided to go with the logo shown below:

Figure showing current logos

[8] http://getbootstrap.com/

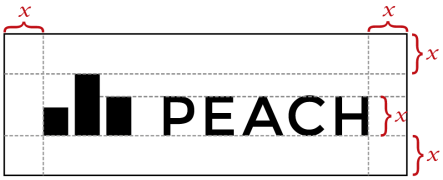


Figure showing clearspace rule design

4.1.2 Components

The components to be used throughout were created using SASS on top of Bootstrap v4.0. The colour schemes developed were coded using SASS preprocessor in order to reuse the colour variables for each component as needed:

```
//New dark colour scheme Peach
$peach-main-dark: #132F4D;
$peach-orange-dark: #D57939;
$peach-grey-dark: #363636;
$peach-navy-dark: #63717E;
$peach-purple-dark: #753F6E;
$peach-mint-dark: #6AB6AC;
$peach-sand-dark: #D4A480;
$peach-pink-dark: #C8696D;
$peach-blue-dark: #1F5893;
$peach-green-dark: #50864E;

//New medium colour scheme Pea
$peach-main-med: #BBC4CE;
$peach-orange-med: #E2C6C6;
$peach-grey-med: #C3C3C4;
$peach-navy-med: #C8D0E7;
$peach-purple-med: #CFB5D1;
$peach-mint-med: #BCD8D9;
$peach-sand-med: #EED1C6;
$peach-pink-med: #E4BCC4;
$peach-blue-med: #C2DAEE;
$peach-green-med: #C3E1CB;
```

Figure showing SASS variables used

```
.btn-peach-one-round{
  //border-radius: 15px !important;
  @include button-variant($peach-3, $peach-4, $peach-4,24px);
}
.btn-peach-two-round{
  //border-radius: 15px !important;
  @include button-variant($peach-4, $peach-3, $peach-3,24px);
}
```

Figure showing mixins being used to create new PEACH style buttons

Some mixins were altered in the .scss files to cater for more customization.

4.1.3 Open Source

The author decided to make the design guide an open source project. This meant that if there were use cases that had not been catered for in other teams they were able to contribute to the design guide.

The author achieved this by creating a UCLH GitHub profile and uploading the necessary files to the repository. From here, other PEACH team members were able to make a pull request and alter the design guide.

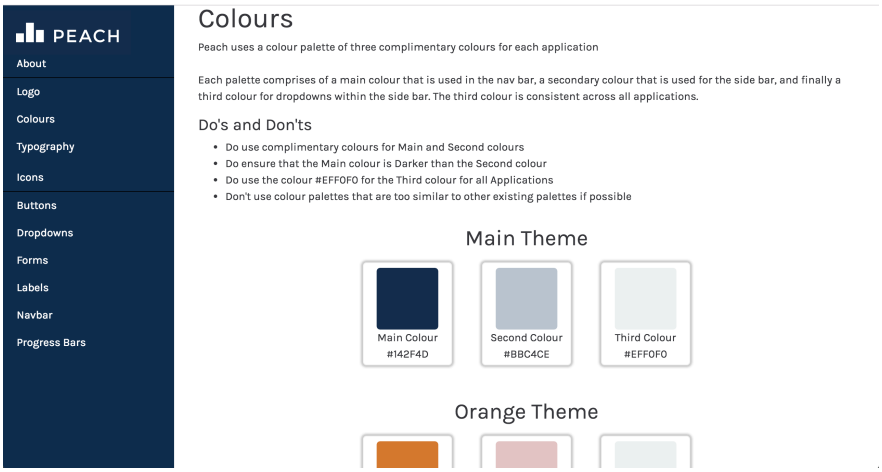


Figure showing design guide website

The author hosted the website using the UCLH Github pages feature. To visit the design guide please visit the following link:

https://peach-uclh.github.io/peach-design-guide/

More screenshots of the design guide can be found in the appendices figure d.2.

4.1.4 Publish to NPM

To make the library developed by the team available to all other PEACH sub-teams the author decided to publish the files to npm. This allowed for others to install the libraries using 'npm install peachstrap-ucl' command.

4.2 High Level Architectures

4.2.1 MDT Application

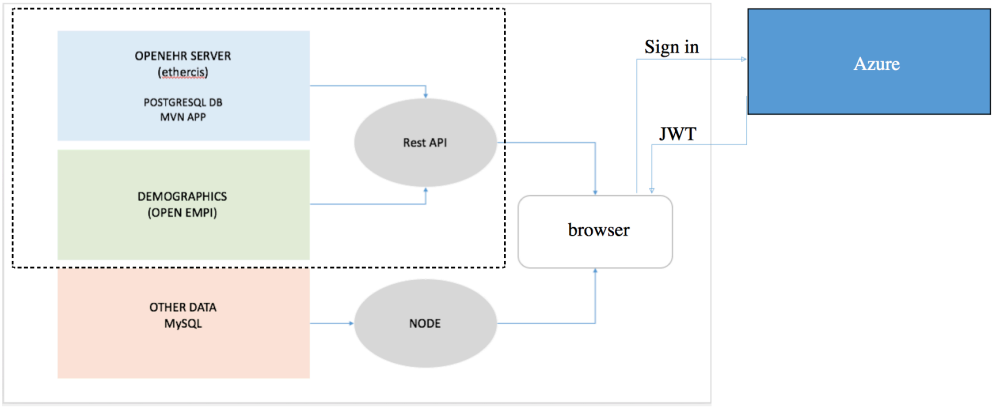


Figure showing high level architecture

The OpenEHR server and demographics server were developed in a Docker[9] container by one of the MDT team (Alessandro). This was at the request of the client in order to provide portability, efficiency and did not need the overhead that comes with running a virtual machine.

The Docker environment for the EtherCIS server can be found here:
https://github.com/alessfg/docker-ethercis

The Docker environment for the OpenEMPI server can be found here:
https://github.com/alessfg/docker-openempi

Zooming into the browser component in the architecture figure above the author will use the following design architecture [29]:

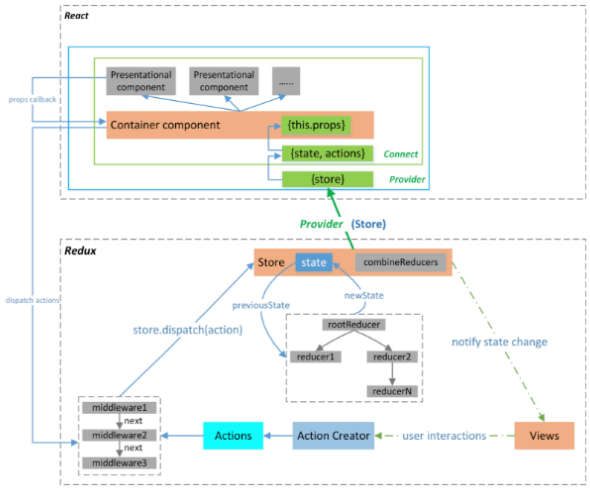


Figure showing architectural overview of front-end

This architectural figure shows the seperation of React and Redux. It also shows how data is passed from Redux to React via the Provider (see section 4.5.1) HOC pattern. The container components are wrapped in a connect container that has access to the store.

In the Redux side it shows where combining reducers occurs, which the author needed to use to combine authentication and form reducers. The middlewares are presented which is where the author hooks into the Redux application to write own logger middleware for debugging purposes (see section 5.3).

[9] https://www.docker.com/

4.3 Data Storage

The data that is to be stored on the servers is still to be determined by the clients. Due to the complexity of the data, this was not able to be achieved by the MDT software team.

Few weeks before the end of the project, the author was provided with an operational template of the data to be included in the referral form page. This can be found at the following link

https://github.com/freshehr/opencancer/tree/master/technical/operational.

Unfortunately, without these data templates it was deemed very difficult to work on developing the application to fit the data needs.

4.4 Environment Configuration

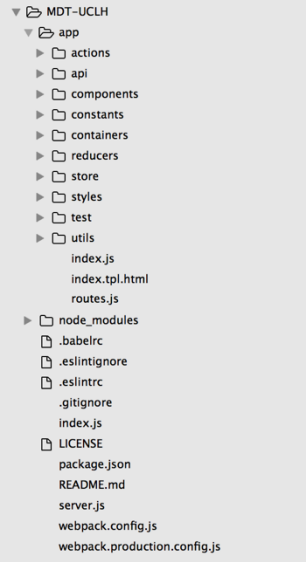


Figure showing development environment structure

The figure to the left displays the structure of development environment.

Package.json contains all configuration options for the Node application including developer dependencies, application dependencies and scripts.

Server.js set up the Webpack development server and hosted the web app on port 3000.

Webpack.config.js and Webpack.production.config.js are the configuration files for the Webpack bundler. The author uses two separate config files because the production file is used for building a production ready application. The two use cases are demonstrated in the scripts values in package.json file.

.babelrc is used to configure the Babel presets and plugins.

Finally, the .eslintrc is also a configuration file for eslinter to specify the set of rules that will throw errors.

The folders within the app directory are designed in a way that the author has grouped them by nature. This has been followed from the Redux book[10].

[10] http://redux.js.org/docs/advanced/ExampleRedditAPI.html

Actions are the only source of information for the store, they are payloads of information and describe the fact that something happened [25]. Components are the ReactJS components developed by the author that do not necessarily know about the store. They are functional UI rendering functions that act on props only. The containers are what the author refers to as smart components. These smart components actually talk to the state of the application using React bindings[11].
Reducers are a way to specify how the applications state changes in response to something happening [26]. The store is where the author initialized the store of the application. Own middleware was developed here and applied to the middleware layers in this directory, more can be seen in 4.8 about this.
Finally, the test directory allowed the author to apply unit tests using Mocha. More on this in section 5.

4.5 Routing and Dynamic Navigation

4.5.1 Routes Design Pattern

The author decided to implement a routing system for the MDT application. This was achieved using a library called react-router[12]. React-router is a routing library for React that is highly maintained and well documented.
To implement the routes into the application it was first necessary to create a root container. This container would return the Provider wrapper component and the Router component.

```
7  export default class Root extends Component {
8      render() {
9          const { store, history } = this.props;
10         return (
11             <Provider store={store}>
12                 <div>
13                     <Router history={history} routes={routes} />
14                     <DevTools />
15                 </div>
16             </Provider>
17         );
18     }
19 }
```

Figure showing Root container to render routes

Line 9 demonstrates using ES6 syntax to deconstruct the props object. This gives access to store and history which are passed down from the parent container as props.
The Router component seen on line 13 contains a routes property assigned to an imported routes variable. This routes variable contains the routes nested in more

[11] https://github.com/reactjs/react-redux
[12] https://github.com/reactjs/react-router

detail. The reason the author has decided to use this design pattern is that it keeps the separation of low level routing of components to higher level abstract parent containers.

4.5.2 History

Since the application is an SPA this model breaks down the browsers design for page navigation using the forward/back buttons [27]. Of course this caused issues because if the user were to try and navigate back using the back button then the expected behaviour would not occur. A solution to this is provided by the author by implementing the hash history feature of the react-router library. Hash history is a hash-based URL used in determining routing state in a SPA.

4.5.3 Navigation Bar

The navigation bar proved to be a difficult design due to each user having different routes to display. The author needed a way of determining which role the user logging in had and so instead of waiting for the Azure authentication to be completed, decided to create a mock authentication system shown in section 4.6.

The author wanted to create the components to be as reusable as possible. This meant that making sure the navigation bar acted as a dumb component and worked off props alone was the favoured design choice.

Since the navigation bar was to be present on all routes, the decision was made to render the Navbar component inside the main App component, however not injected as a child. From here, the author was able to render all children of the App component that were injected via the react-router.

The author had to find a way of keeping the state of the application in line with the rendering of Navbar components. This proved a complex issue however a solution was designed.

```
class NavbarContainer extends Component {

    render() {
      const { user } = this.props;
      console.log(this.props.onLogin)
        return (
      <NavbarType>
        <NavbarHeader />
          <NavbarCollapseWrapper>
            <NavbarListNavigation
                    role={user.role}
                    onLogin={this.props.onLogin}
                    onLogout={this.props.onLogout}
                  />
          </NavbarCollapseWrapper>
      </NavbarType>
      );
```

Figure showing Navbar container code

Here is the Navbar container which is the only component of the navigation bar that knows of the applications state, i.e. role of user logged in. A hierarchy of components has been created in order to abstract away much of the navigation html.

The author is passing the user object in as props to the container component in which the role is then passed into NavbarListNavigation component to dynamically generate a list of links to display in the navigation.

To pass the role from state to props the author has used a library called react-redux[13]. This allows for the use of two crucial functions: (1) mapStateToProps(); (2) mapDispatchToProps(). I am then able to export the 'decorated' smart component using the connect() function which will sync the container to the application state.

Another thing to notice is the author has decided to pass down onLogin and onLogout functions as props too. This allows for more reusable navigation components as if they were to be reused then developers can provide own functions as props to the components.

The functions that are passed in as props are shown below:

```
38  const mapDispatchToProps = (dispatch) =>{
39    return {
40      onLogin: (email, password) => dispatch(loginUser(email, password)),
41      onLogout: () => dispatch(logout())
42    }
43  }
```

Figure showing mapDispatchToProps() function

Using a middleware known as redux-thunk[14] the author is able to gain access to the dispatch function and fire it off when the user calls the onLogin function passed down as props.

NavbarListNavigation is where the dynamic generation of the navbar links occurs. The way the author has decided to render the dynamic list is by generating on the fly links via the role passed in.

```
37    render (){
38      const { role } = this.props;
39      return(
40      <ul className="nav navbar-nav pull-right">
41        {this.buildNav(role)}
42      </ul>
43      )
44    }
```

Figure showing render() function of NavbarListNavigation component

This is the render function of the NavbarListNavigation component. On line 38 shows deconstruction of the props and assignment of the role to a new variable. This role is then passed to a buildNav function that has been created inside the class.

[13] https://github.com/reactjs/react-redux
[14] https://github.com/gaearon/redux-thunk

The buildNav function below takes in a role and checks whether it is initialized, i.e. a user is logged in. If they are, then an array of items is created using the.map function available on all Array.prototypes. Simple regex is used to clean the label string of spaces before making it a link.

```
buildNav (role){
  let nav = [];
  if (role){
  nav = navbars[role].map((label)=>{
    label = label.replace(/\s/g, '');
    return (
        <li key={label}><Link className='nav-links' to={label} >{label}</Link></li>
      )
  })
    nav.push(<Logout key={'Logout'} logout={this.props.onLogout}/>)
  }
  else{
    nav.push(<Login key={'Login'} login={this.props.onLogin} />)
  }
  return nav;
}
```

Figure showing buildNav() function

The results of different users logging in can be seen below, the first is a clinician and the second is an admin:

Figure showing clinician navigation bar

Figure showing admin navigation bar

4.6 Mock Authentication System

4.6.1 Server Side

The author wanted to demonstrate how asynchronous server calls were made within an authentication process of React/Redux application. It was necessary to design a mock authentication API using Node, ExpressJS and MongoDB. A very simple Doctor model was created that had only email, password and role fields. This however was enough to demonstrate how the authentication system would work in the MDT application once the Azure was set up.

```
api.post('/login', function(req, res){
  User.findOne({
    username: req.body.username
  }).select('password').exec(function(err, user){

    if(err) throw err;

    if(!user){
      res.send({message: 'User doesnt exist'});
    } else if (user){
      var validPassword = user.comparePassword(req.body.password);

      if(!validPassword){
        res.send({message: 'Invalid password!'})
      } else {
         User.findOne({
          username: req.body.username
        }).exec(function(err, user){

        var token = createToken(user);

        res.json({
          success: true,
          message: "Successful login!",
          token: token
        });
      })
      }

    }

  });
```

Figure showing '/login' API endpoint

The API endpoint that was hit from the client side would be '/login'. In the request body the author would provide a username and password value. The username would be checked against the database and if a doctor was found with the same username then the encrypted passwords would be compared. Finally, if the passwords matched, a token would be created and signed which could then be passed back in the http response to be decoded and used on the client side. This token contained information such as email, username, password and most crucially for navigation, the role.

The author would run this express server on a localhost but with a different port to the MDT development server.

4.6.2 Client Side

The client side authentication flow starts off with the user clicking the login button in the Navbar. This Login button was separated into a separate component and would be dynamically generated depending on whether the user was set in application state:

```
      nav.push(<Logout key={'Logout'} logout={this.props.onLogout}/>)
    }
    else{
      nav.push(<Login key={'Login'} login={this.props.onLogin} />)
```

Figure showing insertion of Login Logout components with props

The login and logout functions have been passed in via props and are infact dispatch calls to the actions in order to change the state. This is a complex flow of data and so below is a diagram to help explain the development the author has achieved.

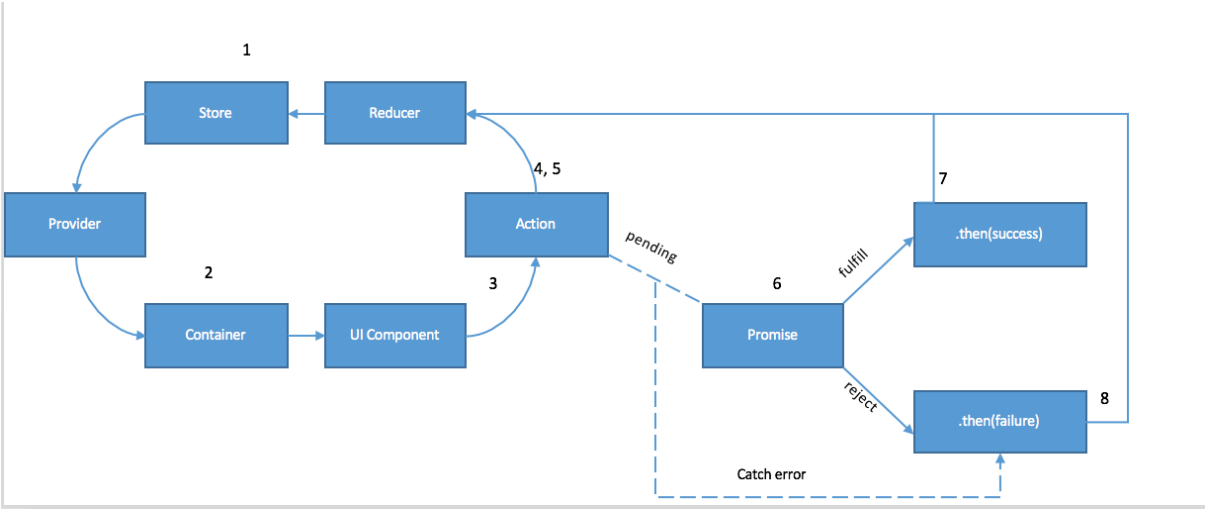


Figure showing flow of auth data through MDT application

The figure above should help in conveying abstractly how the auth system works.

1) Store initialised with authReducer
2) Map loginUser() action to LoginUI props in Login container using mapDispatchToProps()
3) User clicks login submitting email/password combo to loginUser() via this.props
4) Redux-thunk middleware gives access to the dispatch function in loginUser()
5) dispatch({'LOGIN_REQUEST'}) -> Thus updating state -> re-render occurs.
6) Return axios promise thus delaying dispatch and conditionally calling it.
7) dispatch({'LOGIN_SUCCESS', token})
8) dispatch({'LOGIN_FAILURE', err)}

To see this code in action, refer to the full code in the appendices.

To make the http request to the server the decision was made to use Axios[15] which is a promise based HTTP client for the browser. It allowed the author to demonstrate a proof of concept of how to integrate asynchronous data flow into the MDT application.

4.7 Referral Form

Development on the implementation of the referral form was commenced by the author. The author knew that the full implementation would not be able to be

[15] https://github.com/mzabriskie/axios

finished in time however enough was completed to demonstrate to future students how the form could be implemented in the application.

Using a library called redux-form[16] the author was able to write forms using the Higher Order Component design pattern. This allowed to abstract away form state which can hide complexities in dealing with forms in React/Redux.

Redux-form library consisted of four things [28]:

1. Redux reducer that listens to dispatchers of redux form actions
2. React component decorator (this is the HOC design pattern mentioned earlier)
3. Field component to connect store to field
4. Action creators to interact with form

For the MDT referral form, the author decided to use a multi page wizard form for ease of navigation. Each page was split up using the data requirement models provided earlier.

Each page is a component which is connected to the Redux state by decorating the exported component in the reduxForm HOC.

```
const validate = values => {
  const errors = {}
  if (!values.firstName) {
    errors.firstName = 'Required'
  } else if (values.firstName.length > 10) {
    errors.firstName = 'Must be 10 characters or less'
  }
  if (!values.lastName) {
    errors.lastName = 'Required'
  } else if (values.lastName.length > 25) {
    errors.lastName = 'Must be 25 characters or less'
  }
  return errors
}
```

Figure showing form validation

Validation client side could be done on the form too. The figure to the left demonstrates validation on the first page of the MDT referral form. Using the help of redux-form examples, the author was able to implement validation and provide a demonstration for future developers on how to write validation. An example of how the form looks is shown in the appendices figure d.3.

4.8 Middleware

Using the redux-thunk middleware the author was able to hook into the Redux code and write middleware. As shown in the figure below, one is able to write multiple middleware layers and compose them in a chain. In fact, using the thunk middleware

[16] https://github.com/erikras/redux-form

layer, the author was able to gain access to the dispatch function in Redux thus allowing for asynchronous API calls.

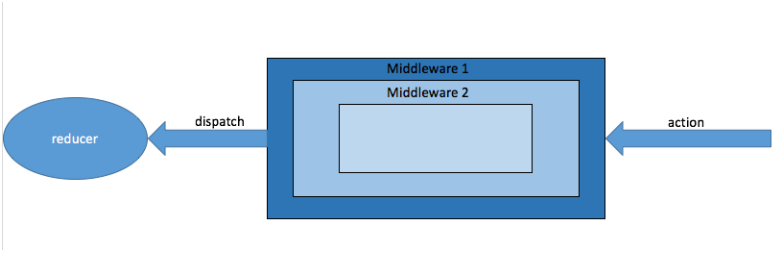


Figure showing how middleware layers fit into Redux

The author also wrote his own piece of middleware to help in the debugging stage of development. This is discussed further in section 5.3.

```
const enhancer = compose(
  applyMiddleware(thunk, logger),
  DevTools.instrument()
  )

export default function configureStore(initialState) {
    const store = createStore(
        rootReducer,
        initialState,
        enhancer
    );
    return store;
}
```

Figure showing configuration of Redux store

To compose the middleware layers a variable enhancer was created using the compose function from the Redux API. Passing the enhancer object as a third argument to the createStore function allowed redux to do the heavy lifting of creating the store for application state.

4.9 Why Good Design Choices

This section is to reflect on why the implementation and design choices were chosen. Firstly, knowing that the application would not be developed in the time frame given, the author decided to demonstrate a broad range of implementation choices. This was for the benefit of the client and future developers because they would be able to pick up the work and use examples from the authors code to continue development. Every aspect of the application has been implemented from logging, form control and validation, passing of state from redux to presentational components, middleware development and reusable components.

The author has implemented with scalability in mind for example using multiple reducers for each section of the state and combining the reducers in a separate file. This is a reducer composition design pattern and should help develop maintainable code when passed to future students.

5 Testing

Software testing is the process of purposely trying to find errors or bugs in the developed code by running the program against different cases. Trying to detect all the ways in which a piece of software can fail is a monumental task which practically is not possible. The author used an iterative process of software testing [29] which consisted of (1) designing test; (2) executing test; (3) identifying problems; (4) fixing problems.

Research showed that finding and fixing a software problem after delivery is often 100 times more expensive than finding and fixing it during the requirements and design phase [30]. It is for this reason the author spent a considerable amount of time testing the developed code. This was to ensure that passing of the code down to future developers would not lead to time being wasted on finding and fixing bugs.

5.1 Time Travel Debugging

One of the decisions to go ahead with building a Redux application was the time travel debugging feature that the author was able to configure. With Hot Module Replacement (see section 2.7.2) the application was able to keep state even after code had changed. This is conjunction with time travel debugging tools the author was able to go backwards to previous points in the applications state.

This is extremely useful because the author was able to test very specific parts of user interaction. The main advantage is the speed of which the author was able to test certain interactions. The author would interact with a certain part of the UI such as the login button. This would cause state to change and if the correct changes were not displayed the author was able to time travel back in time to before the interaction occurred. This allowed for a change in code and then to reapply the UI interaction.

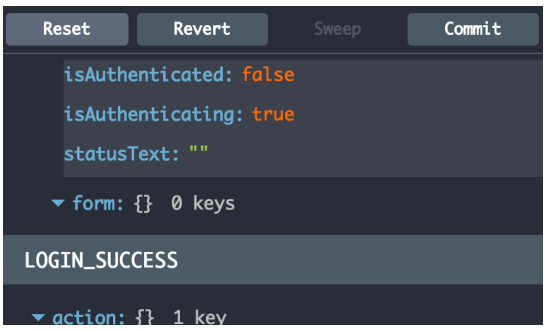


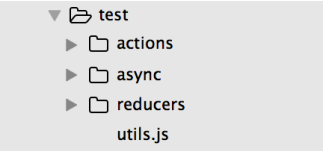
Figure showing the tool used to time travel

The tools were attached to the browser and state changes were observed during interactions and firing of actions. Using the 'Revert' button, interactions could be reversed and changes in code implemented. The author was able to then re apply the

interaction test and witness if fixed.

5.2 Unit Tests with Mocha

Most of the Redux code that was written by the author are functions. These are mainly actions and reducers. They are pure functions and so can be easily tested since the return value is only determined by its input value with no discernible side effects. The author placed all of the tests in a test directory (see below) and then split the tests into sub directories based on the nature of the object being tested.



Tests were run by writing a test script in the package.json file. This enabled the author to write 'npm test' which ran all the subsequent tests found in the test directory.

```
6   "scripts": {
7     "test": "mocha ./app/test --compilers js:babel-register --recursive",
```

5.2.1 Authentication Reducer Tests

The authentication reducer was built using Test Driven Development [31] (TDD) principles to encourage simple designs in the code. Using each iteration of designing the test, a fully function authentication reducer became apparent. The author had to mock some initial state of the authentication in the store to apply the tests to:

```
9   let initialState = {
10        uid: '',
11        role: '',
12        isAuthenticated: false,
13        isAuthenticating: false,
14        statusText: ''
15      }
```

Once this mock state was created the author could reduce dependencies on other code segments to allow pure testing of the reducer.

The list of tests applied to this reducer can be seen below:

```
authentication reducer
  ✓ should return the initial state
  ✓ should handle login request
  ✓ should handle successful login
  ✓ should handle unsuccessful login
  ✓ should handle logging out
```

The title of the test would be passed as the first argument into the it() function, followed by a function that handles the logic of the test.

```
41  it('should handle successful login', ()=>{
42    expect(authReducer(undefined, {
43      type: authConstants.LOGIN_SUCCESS,
44      payload: {
45        uid: '123abc',
46        role: 'clinician'
47      }
48    }))
49      .toEqual({
50        uid: '123abc',
51        role: 'clinician',
52        isAuthenticating: false,
53        isAuthenticated: true,
54        statusText: ''
55      })
56  })
```

This test is representing a successful payload return from the server in which a uid and role is received in an http response. This should then set the state to the state shown on lines 50-45.

The imports used in the authentication reducer test file could cause tests to incorrectly pass and so that leads me on to my next section.

```
1  import expect from 'expect'
2  import authConstants from '../../constants/auth'
3  import authReducer from '../../reducers/auth'
4  import * as actions from '../../actions/auth'
```

The author chose to rely on the constants written elsewhere in the code to use within the unit tests. These constants were created using a utility function and so the decision was made to create tests for this function.

5.2.2 Utility Tests

The function used by the author to create constants is a design choice to ensure constant objects are produced in the same manner every time. These constants are used throughout the application and thus the pure function that creates them needs to be tested.

```
1  export function createConstants(...constants) {
2      return constants.reduce((acc, constant) => {
3          acc[constant] = constant;
4          return acc;
5      }, {});
6  }
```

Running the npm test command shows that the constant creator function is running correctly.

```
util helper functions
  ✓ should create an object of constants given strings
```

The code for the constant creator is shown below.

```
1  import expect from 'expect'
2  import { createConstants } from '../utils'
3
4  describe('util helper functions', ()=>{
5
6    it('should create an object of constants given strings', ()=>{
7      const expectedObject = {
8        ADD_PATIENT: 'ADD_PATIENT',
9        REFER_PATIENT: 'REFER_PATIENT',
10       REMOVE_PATIENT: 'REMOVE_PATIENT'
11     }
12     expect(createConstants('ADD_PATIENT','REFER_PATIENT','REMOVE_PATIENT'))
13       .toEqual(expectedObject)
14   })
15
16 })
```

5.2.3 Mock Object for Store and Asynchronous Action Creators

The testing of the asynchronous behavior of the authentication process was a lot more complex. The author decided to use a library called redux-mock-store[17] to imitate the Redux store. This gave the ability to test asynchronous behavior in the actions. The way this works is by storing the dispatched actions in an array which can then be used in the tests. Since the action creators had been tested too this was deemed a safe test. The mock store is created in the following way:

```
16  const middlewares = [ thunk, logger ]
17  const mockStore = configureMockStore(middlewares)
```

Firstly, an array of middleware is created consisting of thunk and a logger. Then using the configureMockStore() function the author is able to pass in the middleware array. This ensured that the same middleware was being used in the mock store as had been used in the real Redux store.

```
34    const store = mockStore({ auth : {} })
```

The author then designed how the mock store object is to look by passing in a key of 'auth' and a value of an empty object.

The following code demonstrates how the mock store is used in conjunction with Mocha.

```
19  describe('async auth actions', ()=>{
20    it('creates LOGIN_SUCCESS when successful login has occured', ()=>{
21    //array of expected actions to be dispatched.
22    const expectedActions = [
23        { type: constants.LOGIN_REQUEST },
24        { type: 'LOGIN_SUCCESS',
25          payload: {
26            uid: '123abc',
27            role: 'clinician'
28    }}
29     ]
30    const store = mockStore({ auth : {} })
31
32    return store.dispatch(actions.loginUser('abc@123.com', 'password123'))
33    expect(store.getActions()).toEqual(expectedActions)
34    })
```

On line 32 the author dispatched the login user action with a valid email and password. This should pass the Login success constant along with the payload from the server.

From the figure below we can see that the test is successful.

```
async auth actions
  ✓ creates LOGIN_SUCCESS when successful login has occured
  ✓ creates LOGIN_ERROR when unsuccessful login has occured
```

A test was also created for the login error by passing in an incorrect email and password combination.

[17] https://github.com/arnaudbenard/redux-mock-store

5.3 Middleware Logger

To aid in the debugging process the author decided to utilize the fact that middleware layers can hook into the Redux code by writing a logger function. The logger function is curried and uses the concept of closure to implement lexically scoped name binding in order to log out the actions being dispatched to the console.

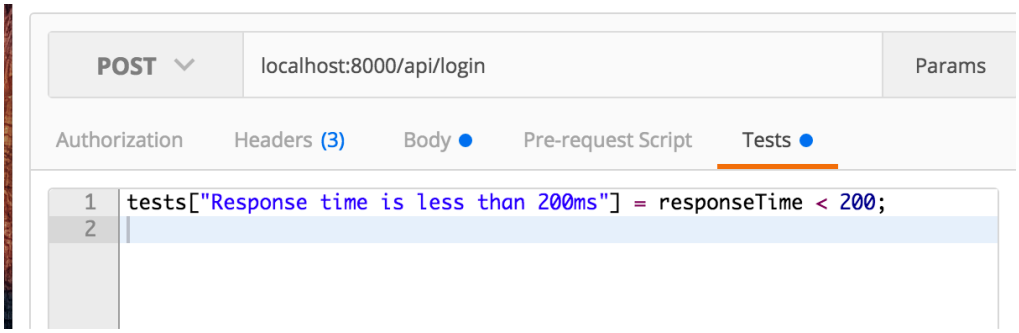
```
// //own middleware
const logger = store => next => action => {
  console.info('dispatching', action)
  let result = next(action)
  console.log('next state', store.getState())
  return result
}
```

Middleware developed for debugging purposes

5.4 Postman

In order to test that the authentication API was working successfully, the author decided to use Postman[18]. Test scripts could be written in the Postman application to generate API tests.

To test the response time of the API signup the following code is used:



[18] https://www.getpostman.com/

6 Conclusion

6.1 Goals Achieved

- *An open source brand guidelines document to be used by sub-teams in MDT*

  A full brand guidelines document has been produced and delivered to the client. The brand guidelines document was also presented to the PEACH team and has been hosted to be viewed publicly. The document has been deemed open source and account details handed over to the client.

- *A library of components to be used by sub-teams in MDT*

  A library of components has been created on top of Bootstrap v4.0. This library of components has been uploaded as an npm package allowing teams to download it through npm install command. The library of components is extensive and are customised to suite differing themes in the PEACH projects.

- *A development environment to be used by future students in the development of MDT application*

  A fully configured development environment has been created. The stack used in the development is React, Redux, Webpack, Node. Useful tools and libraries have been imported to aid in development processes such as linting tools. This environment has been hosted on Github and handed over to the client.

- *Designs of user interface ideas for MDT application*

  Mock-ups have been created in relation to general layout of navigation and colour schemes along with logo placement. However, since a lot of time in the early stages of the project was spent on UI/design of PEACH branding the author decided to not spend as much time on mocking up UI of the MDT

application. This decision was made to ensure enough time was focused on demonstrating programming ability.

- *A mock authentication system using Node, Express and MongoDB*

  A mock authentication system has been developed using Node, Express and MongoDB. This mock authentication system demonstrates to the client and future developers how an authentication system could be implemented in the MDT application with the integration of Redux states. The authentication will eventually be hooked up to a server with secure clinician data however this has not been provided as of yet.

- *A proof of concept referral form to refer patients to MDT meetings*

  A design for the referral of patients to MDT meetings has been developed using a wizard like form display. The form state is hooked up to Redux action creators and the store. Future developers can use this as an example form whereby it demonstrates how the forms can be implemented in the application.

- *A document to be given to client on future development in order for future students to start developing early*

  A document has been passed on to the client which explains the directory structure and how to continue with development. A meeting was organised with the client to discuss the plan for future developers.

6.2 Fulfilment of Personal Aims

- *Learn web development fundamentals*

  The project provided a fantastic opportunity to learn around the topic of web development. The author has learnt how to utilise open source libraries and

how to integrate them into his own applications. Getting involved in web development communities on Reddit, Github and Gitter board has been interesting and the community has shown they are a helpful crowd.

- *Learn ES6 syntax, React, Redux, Webpack, Node (Express) in order to keep up with current hot trends in building web applications*

  By writing the application in ES6 syntax the author has had the opportunity to explore the new syntax. The most interesting and useful syntax learnt in this project was the inclusion of object deconstruction. React and Redux have a strong following and using these technologies showed an insight into why that is. Building the backend for the mock authentication API provided the author with an opportunity to explore Node and Express along with MongoDB.

- *Understand Docker fundamentals*

  Unfortunately, the author did not get to spend much time working on the Docker containers. This is due to another team member taking on the task. This did not prevent the author from learning and reading fundamentals however no hands on coding experience in creating Docker containers was achieved.

- *Learn good design principles for UI/UX*

  Since the initial phase of the project was to design and develop branding for the new PEACH team, a lot of time was spent learning good design principles for UI/UX. Medium articles proved to be invaluable and bloggers from other sources also wrote posts that shared latest UI/UX designs. After reading and following these bloggers, the author developed a solid understanding of good UI/UX practices.

6.3 Critical Evaluation

A key criticism of this project is the way the management was handled. The lack of hierarchies made working alongside others extremely difficult. In a recent study [32] it was shown that ten characteristics underpinning effective interdisciplinary team work were identified. When comparing the ten characteristics with the characteristics of our own team it was evident that many aspects were missing. For example, (1) appropriate skill mix; (2) positive leadership and management; (3) training and development were just some of the traits lacking in our team.

Appropriate skill mix was a difficult characteristic to achieve. Essentially, people were placed in what matched their interests as best as possible. This unfortunately did not mean to say those placed had any experience in that field of interest. In the case of MDT, some team members were of extremely varying standards and so some of the team felt as though they were behind due to the complexity and progress of the others in the team with more experience.

The second point of positive leadership and management should have been in place from the beginning. Unfortunately, this was not the case and so confusion arose to who should develop which aspects of the application. With no management and leadership, developers in the team seemed to clutch at what they wanted leaving a first come first serve allocation system.

Finally, training and development was cut short due to the splitting of the project into two phases. The first phase solely focusing on gathering and implementing UI/UX requirements left the author with less time to train and develop skills needed to be able to code the MDT application.

6.4 Future Work

The work carried out by the author was a proof of concept and the application is not in a fully functioning state. Instead, the author has provided a range of examples of how the application can be developed by future students.

The authentication has been implemented using Node, Express and MongoDB however this will eventually be hooked up to the PEACH team Azure accounts using B2C.

The MDT application is likely to have multiple forms and so the referral form implemented by the author will provide examples on which future students can create these forms.

If more time was given, the author would be able to implement a fully working MDT referral system that would connect up to a scheduler that has been developed by another team member. Hopefully this can be implemented within the first month of development with the new students.

6.5 Final Thoughts

To wrap up, the author feels that the project has been successful. Although the MDT application has not been completed to a standard that is usable by the client, the author feels that enough guidance can be provided to future students such that it will not take them long to get started on development. The initial phase of the project was interesting and gave the author experience in UI/UX whereby a more creative side was shown. The second phase of the project allowed for hands on experience in a real world development team with a real world client. This has been an excellent involvement giving the author crucial skills for working in the tech industry.

Appendices

Section A

figure a.1

figure a.2

figure a.3

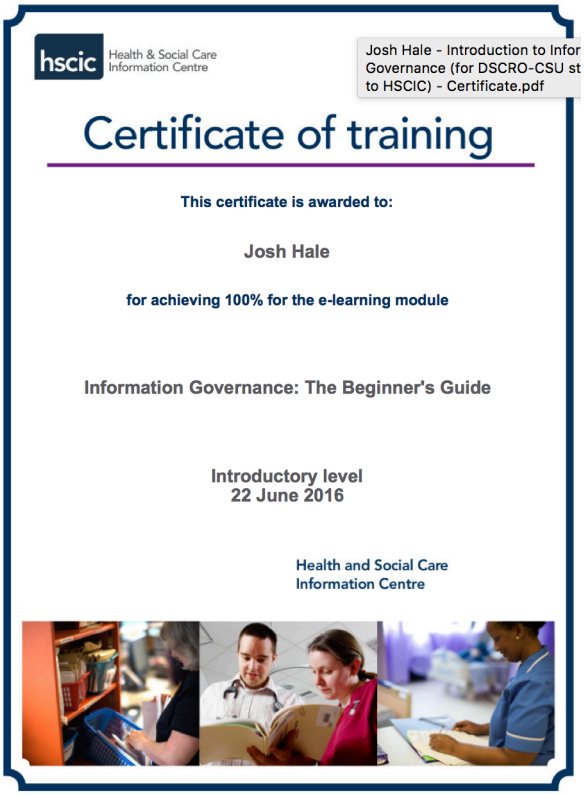
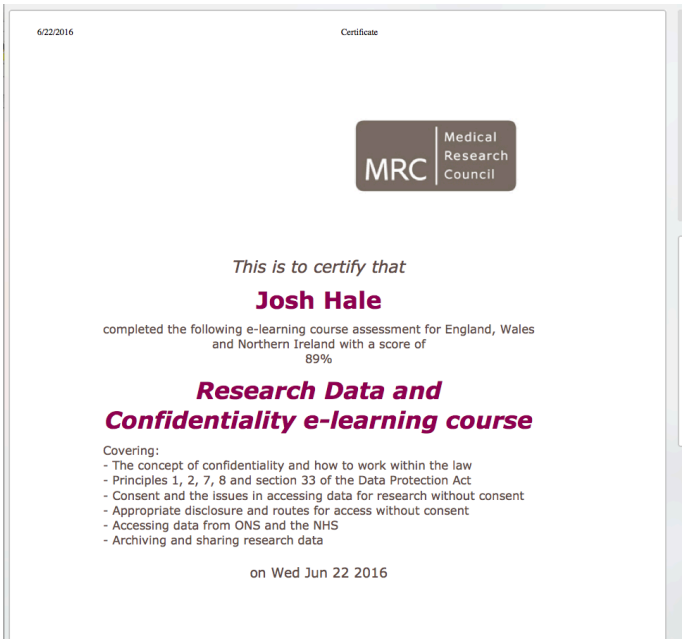


figure a.4

figure a.5



6/22/2016 Certificate

MRC Medical Research Council

This is to certify that

Josh Hale

completed the following e-learning course assessment for England, Wales and Northern Ireland with a score of
89%

Research Data and Confidentiality e-learning course

Covering:
- The concept of confidentiality and how to work within the law
- Principles 1, 2, 7, 8 and section 33 of the Data Protection Act
- Consent and the issues in accessing data for research without consent
- Appropriate disclosure and routes for access without consent
- Accessing data from ONS and the NHS
- Archiving and sharing research data

on Wed Jun 22 2016

Section B

figure b.1

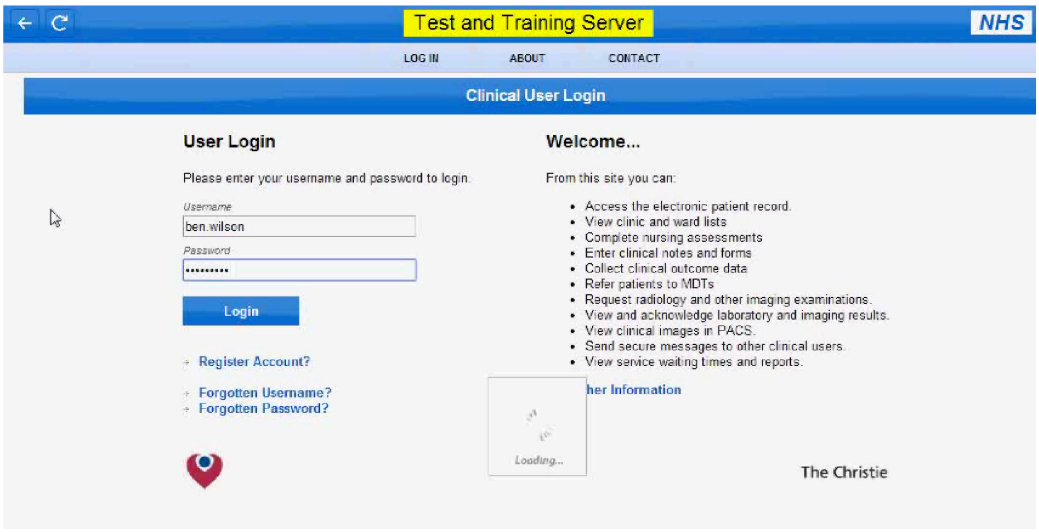


figure b.2

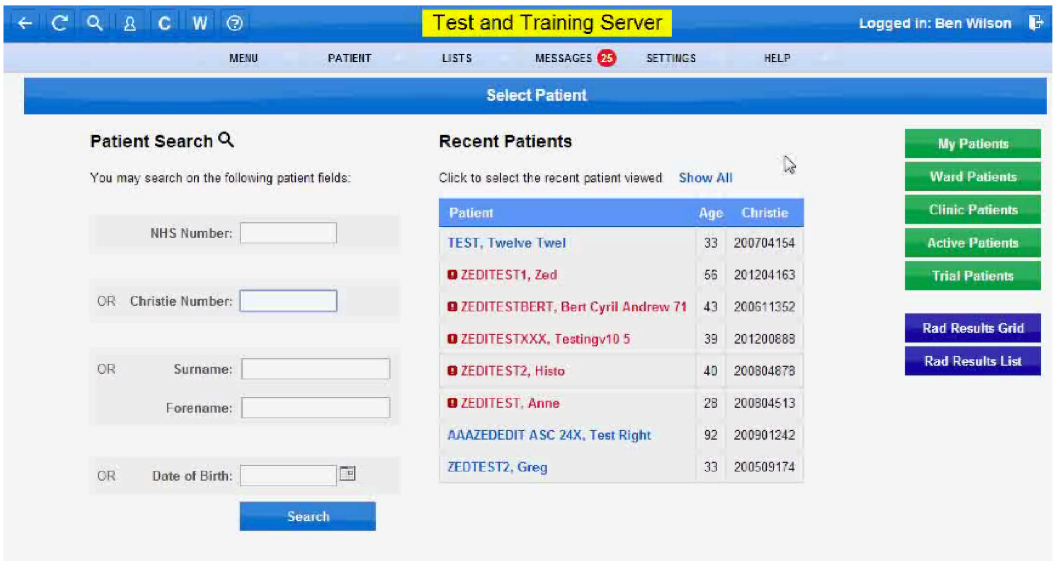
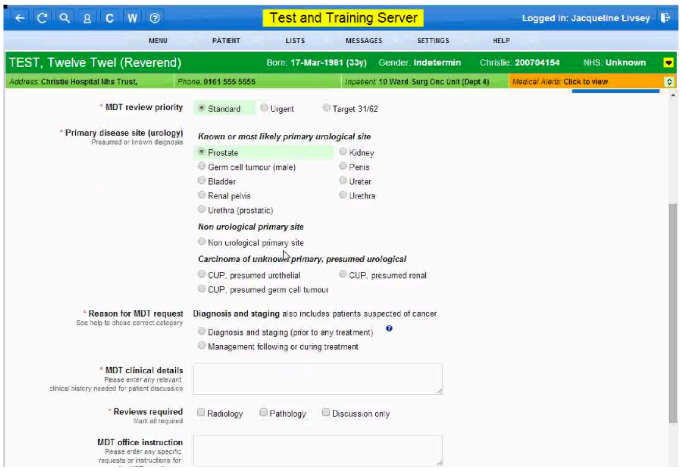


figure b.3



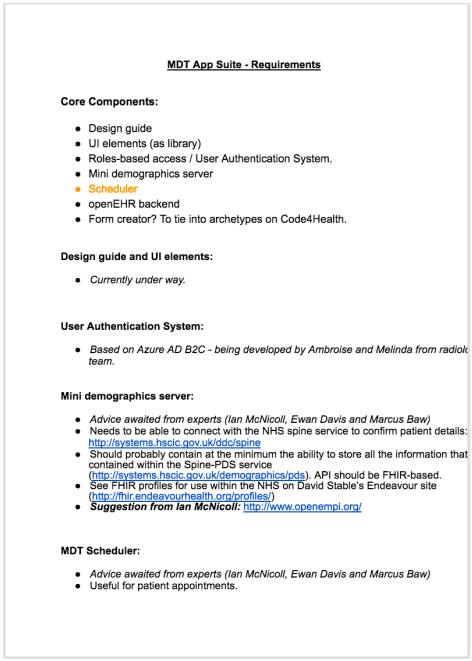
Section C

Figure c.1

	A	B	C	D
1	**Requirement ID**	**Functional Requirement**	**Priority**	**Users**
2		***BACKEND***		
3	***etherCIS***			
4	RQ1	Install and configure EtherCIS, subject to it being appropriate	Must	N/A
5	RQ2	Health related data stored via forms should be appropiately stored in etherCIS	Must	N/A
6	***openEMPI - Might not be necessary***			
7	RQ3	Users should be able to add new patients to the server	Must	Clinicians
8	RQ4	Users should be able to search for patients in the server	Must	Clinicians
9	RQ5	Create new ehrID in etherCIS upon registering new patient	Must	N/A
10	RQ6	Ability to store all the information that is contained within the Spine-PDS service	Could	N/A
11	RQ7	Ability to validate records against the Spine-PDS service	Could	N/A
12	***User Authentication System***			
13	RQ8	Authentication using Azure AD B2C	Must	All
14	RQ9	Role based authorisation using Azure AD B2C	Must	All
15		***FRONTEND***		
16	***"MDT Hub" Web App***			
17	RQ10	Hub view should changed based on user role (MDT Coordinator vs Clinician)	Must	All
18	RQ11	MDT Coordinator view list of new referrals	Must	MDT Coordinator
19	RQ12	MDT Coordinator can start meeting mode	Must	MDT Coordinator
20	RQ13	Clinician can view list of patients and patient records to be discussed in upcoming meeting	Must	Clinicians
21	RQ14	Ability to search for patients and load their profile	Must	All

	A	B	C	D
22	***Patient Profile***			
23	RQ15	Ability to view patient demographics	Must	All
24	RQ16	Ability to view patient past referrals	Must	All
25	RQ17	Ability to view patient EHR	Must	All
26	RQ18	Clinician can create new referral form for patient	Must	Clinicians
27	***Meeting Scheduler***			
28	RQ19	Meetings for different areas to be created automatically every week at their respective timeslot	Must	N/A
29	RQ20	Create a new meeting if necessary	Must	MDT Coordinator
30	RQ21	Display the different invited clinicians that are due to attend an MDT meeting	Must	N/A
31	RQ22	Display a list of the different patients that are going to be discussed in an MDT meeting	Must	N/A
32	RQ23	Add new patients to meeting schedule	Must	MDT Coordinator
33	RQ24	Send notification emails to both Junior and Senior MDT doctors 2 and 1 day before the meeting resp	Must	N/A
34	RQ25	View patient profiles with demographics and EHR data	Must	N/A
35	RQ26	Order patient discussions within a meeting according to how they will be discussed	Could	N/A
36	RQ27	Meeting list can then be saved as a .docx or .pdf file	Should	N/A
37	***Meeting Mode***			
38	RQ28	System should be able to keep a register of attendance	Must	N/A
39	RQ29	Users should be able to write public and/or private notes for each patient during a meeting	Must	All
40	RQ30	Users should be able to take pictures and/or videos and upload them for each patient during a meet	Should	All
41	RQ31	Users should be able to write using inking technology	Could	All
42	***MDT Referral Form***			
43	RQ32	Ability to add images and documents	Should	Clinicians
44	RQ33	Validate all fields	Must	N/A
45	RQ34	Patient demographic lookup	Must	N/A
46	RQ35	Clinician must fill out all fields specified at https://docs.google.com/spreadsheets/d/1Ljs2nwUCqortk	Must	N/A
47	RQ36	Referral form can then be saved as a .docx or .pdf file	Could	N/A
48	***MDT Preparation Forms***			
49	RQ37	Depending on the type of clinician they will be able to fill out different MDT preparation forms (Radio	Must	Clinicians
50	RQ38	Form must be able to take in text	Must	Clinicians
51	RQ39	Form must be able to take in files / images	Should	Clinicians
52	***Form Creator - Extension Goal***			
53	RQ40	User should be able to access a form builder that will automatically upload the template to etherCIS	Could	All

figure c.2



MDT App Suite - Requirements

Core Components:

- Design guide
- UI elements (as library)
- Roles-based access / User Authentication System.
- Mini demographics server
- Scheduler
- openEHR backend
- Form creator? To tie into archetypes on Code4Health.

Design guide and UI elements:

- *Currently under way.*

User Authentication System:

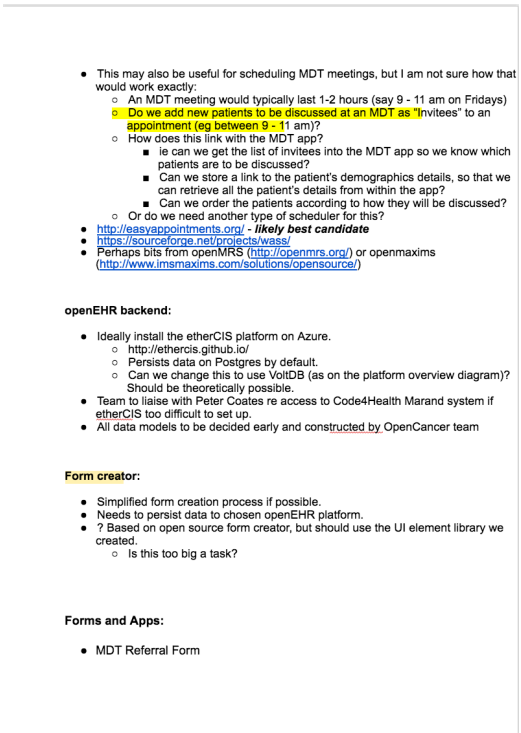
- *Based on Azure AD B2C - being developed by Ambroise and Melinda from radiol team.*

Mini demographics server:

- *Advice awaited from experts (Ian McNicoll, Ewan Davis and Marcus Baw)*
- Needs to be able to connect with the NHS spine service to confirm patient details: http://systems.hscic.gov.uk/ddc/spine
- Should probably contain at the minimum the ability to store all the information that contained within the Spine-PDS service (http://systems.hscic.gov.uk/demographics/pds). API should be FHIR-based.
- See FHIR profiles for use within the NHS on David Stable's Endeavour site (http://fhir.endeavourhealth.org/profiles/)
- ***Suggestion from Ian McNicoll:*** http://www.openempi.org/

MDT Scheduler:

- *Advice awaited from experts (Ian McNicoll, Ewan Davis and Marcus Baw)*
- Useful for patient appointments.



- This may also be useful for scheduling MDT meetings, but I am not sure how that would work exactly:
  - An MDT meeting would typically last 1-2 hours (say 9 - 11 am on Fridays)
  - Do we add new patients to be discussed at an MDT as "Invitees" to an appointment (eg between 9 - 11 am)?
  - How does this link with the MDT app?
    - ie can we get the list of invitees into the MDT app so we know which patients are to be discussed?
    - Can we store a link to the patient's demographics details, so that we can retrieve all the patient's details from within the app?
    - Can we order the patients according to how they will be discussed?
  - Or do we need another type of scheduler for this?
- http://easyappointments.org/ - ***likely best candidate***
- https://sourceforge.net/projects/wass/
- Perhaps bits from openMRS (http://openmrs.org/) or openmaxims (http://www.imsmaxims.com/solutions/opensource/)

openEHR backend:

- Ideally install the etherCIS platform on Azure.
  - http://ethercis.github.io/
  - Persists data on Postgres by default.
  - Can we change this to use VoltDB (as on the platform overview diagram)? Should be theoretically possible.
- Team to liaise with Peter Coates re access to Code4Health Marand system if etherCIS too difficult to set up.
- All data models to be decided early and constructed by OpenCancer team

Form creator:

- Simplified form creation process if possible.
- Needs to persist data to chosen openEHR platform.
- ? Based on open source form creator, but should use the UI element library we created.
  - Is this too big a task?

Forms and Apps:

- MDT Referral Form

- MDT hub app
- Radiology prep form
- Pathology prep form
- MDT clinical data prep form
- MDT recording app
- Prostatectomy planning MDT form

MDT Referral form:

- Basic details required for discussion at MDT.
- Have uploaded a renal cancer MDT referral form (from The Royal Free) to Google drive, which is similar to the one we need.
  - Am in the process of getting the UCLH urology / prostate referral form currently
- Ability to attach images and documents (pdf, ppt, doc)
- Needs some validation
  - Some mandatory fields
  - Needs demographic look up - if patient not already registered, will need to b registered on the demographics server
    - This should connect to the NHS spine service to ensure the details ar correct.
  - Includes asking which date this should be discussed (ie this week, next wee in 2 months)
    - Under the hood, we should have a record recurring day/time of the MDT meeting, which should be changeable by admins.

"MDT hub" web app:

- Should be usable for multiple MDTs (need to have some selection mechanism, to change between MDTS eg uro vs lung)
  - As there are many, many MDTs (eg in urology, there is a main cancer MDT Friday, but also 2 further MDTs on Monday), we need to think about how bes to choose the MDT.
  - A default MDT should be set up for every user, and this can be decided during the on-boarding process.
- In admin mode
  - Allows MDT co-ordinator to look at list of referrals and assign patients to different sections of the MDT (local vs regional, prostate vs bladder vs renal
  - Allows MDT co-ordinator to change the order of patients (ie the order they w be discussed in each section).

- In non-admin mode, this will act as the hub app
  - Users will use this app to go through the ordered list of patients, and then open each patient up
  - For use, before, during and after the MDT.
  - The other forms can be launched from this app.
  - We may need an "MDT meeting mode" to show which patients have been discussed, and which are still to be discussed.
- I have uploaded a doc of the current MDT meeting list to get a feel for it. Note the comments I have also attached to the text.

Pathology prep form:

- For pathologist to make notes when preparing for meeting.
- Simple form to capture details of pathology
  - Coded text
  - Free text
  - Maybe ability to attach images and documents (pdf, ppt, doc)

Radiology prep form:

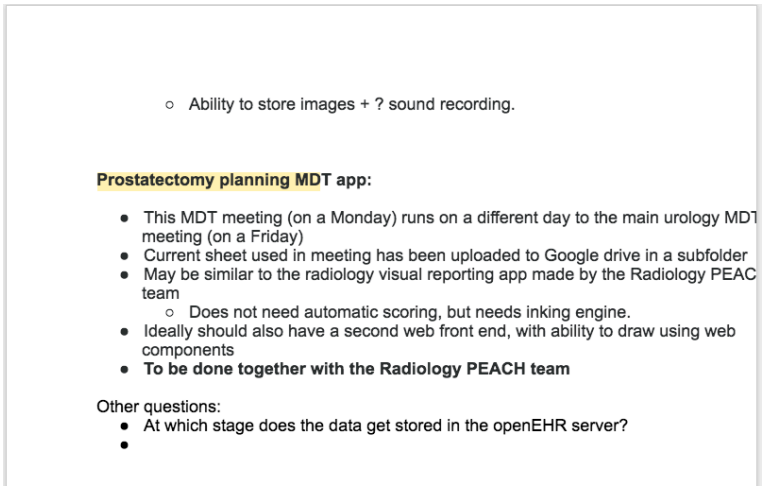
- For radiologist to make notes when preparing for meeting.
- May need to be an app with inking engine (similar to prostatectomy planning MDT
- Mainly free text and coded text
- Ability to attach images and documents

MDT clinical data prep form:

- For surgeon / oncologist to make notes when preparing for meeting.
- Simple form to capture details of pathology
  - Coded text
  - Free text
  - Maybe ability to attach images and documents (pdf, ppt, doc)

MDT recording app:

- For people to make notes **during** the meeting.
- MDT co-ordinator or MDT lead:
  - To record outcomes from meeting
  - To correct errors in the data that has already been put in
    - Should be an auditable change log
- Other people in MDT:
  - To make notes for personal use (not for general view)



- Ability to store images + ? sound recording.

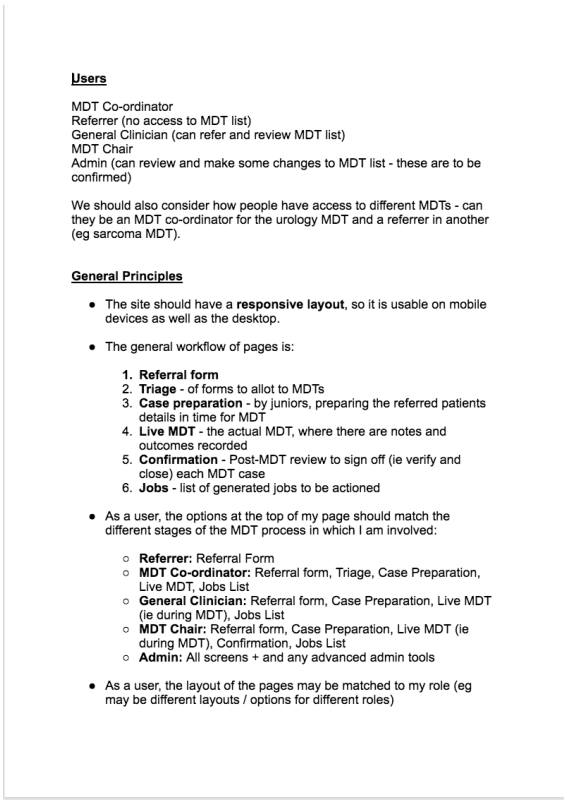
Prostatectomy planning MDT app:

- This MDT meeting (on a Monday) runs on a different day to the main urology MDT meeting (on a Friday)
- Current sheet used in meeting has been uploaded to Google drive in a subfolder
- May be similar to the radiology visual reporting app made by the Radiology PEAC team
  - Does not need automatic scoring, but needs inking engine.
- Ideally should also have a second web front end, with ability to draw using web components
- **To be done together with the Radiology PEACH team**

Other questions:

- At which stage does the data get stored in the openEHR server?
-

figure c.3



Users

MDT Co-ordinator
Referrer (no access to MDT list)
General Clinician (can refer and review MDT list)
MDT Chair
Admin (can review and make some changes to MDT list - these are to be confirmed)

We should also consider how people have access to different MDTs - can they be an MDT co-ordinator for the urology MDT and a referrer in another (eg sarcoma MDT).

General Principles

- The site should have a **responsive layout**, so it is usable on mobile devices as well as the desktop.
- The general workflow of pages is:
  1. **Referral form**
  2. **Triage** - of forms to allot to MDTs
  3. **Case preparation** - by juniors, preparing the referred patients details in time for MDT
  4. **Live MDT** - the actual MDT, where there are notes and outcomes recorded
  5. **Confirmation** - Post-MDT review to sign off (ie verify and close) each MDT case
  6. **Jobs** - list of generated jobs to be actioned
- As a user, the options at the top of my page should match the different stages of the MDT process in which I am involved:
  - **Referrer:** Referral Form
  - **MDT Co-ordinator:** Referral form, Triage, Case Preparation, Live MDT, Jobs List
  - **General Clinician:** Referral form, Case Preparation, Live MDT (ie during MDT), Jobs List
  - **MDT Chair:** Referral form, Case Preparation, Live MDT (ie during MDT), Confirmation, Jobs List
  - **Admin:** All screens + and any advanced admin tools
- As a user, the layout of the pages may be matched to my role (eg may be different layouts / options for different roles)

- I should also be able to search for a patient from the main page and view their whole record
  - ie all the MDT events they have undergone.

- As an advanced feature for admins, we need an audit log of changes:
  - https://www.youtube.com/watch?v=G4DCAJ5MQx4
  - Likely to be very difficult to achieve, but try to formulate a plan.

Referral Form Page

- As a user, I want to add new MDT referral forms.
  - The first question will ask which MDT to refer to.
  - The same adaptive form will be used for each MDT. The fields change as outlined in the MDT data requirements spreadsheet.
  - The form will allow document / image upload. Consider a pdf convertor to change all docs into pdf.
    - This kind of easy to use upload would be perfect:
      - https://help.xero.com/Files
    - openEHR does have file attachment capability

- As a user, I want to review recently submitted forms and make changes
  - Note: This can not be permitted after the case is "prepared". ie the information has been acted on.
  - However, we may need to review this time limit, if it does not work in practice.

Triage Page

- As a user, I should only have access to this page if I am an MDT co-ordinator or Admin

- As an MDT co-ordinator, I will only have access to triage pages for the MDT specialties to which I belong:
  - E.g. Urology, Lower GI, Oesophageal, Sarcoma

- As an MDT co-ordinator I want to be able to do the following:
  - Begin by choosing the MDT specialty I want to see
    - This would bring up a list of all the referrals pending allocation for that specialty

  - Be able to allocate individuals from the pending list to MDTs, either by:

- I should also be able to search for a patient from the main page and view their whole record
  - ie all the MDT events they have undergone.

- As an advanced feature for admins, we need an audit log of changes:
  - https://www.youtube.com/watch?v=G4DCAJ5MQx4
  - Likely to be very difficult to achieve, but try to formulate a plan.

Referral Form Page

- As a user, I want to add new MDT referral forms.
  - The first question will ask which MDT to refer to.
  - The same adaptive form will be used for each MDT. The fields change as outlined in the MDT data requirements spreadsheet.
  - The form will allow document / image upload. Consider a pdf convertor to change all docs into pdf.
    - This kind of easy to use upload would be perfect:
      - https://help.xero.com/Files
    - openEHR does have file attachment capability

- As a user, I want to review recently submitted forms and make changes
  - Note: This can not be permitted after the case is "prepared". ie the information has been acted on.
  - However, we may need to review this time limit, if it does not work in practice.

Triage Page

- As a user, I should only have access to this page if I am an MDT co-ordinator or Admin

- As an MDT co-ordinator, I will only have access to triage pages for the MDT specialties to which I belong:
  - E.g. Urology, Lower GI, Oesophageal, Sarcoma

- As an MDT co-ordinator I want to be able to do the following:
  - Begin by choosing the MDT specialty I want to see
    - This would bring up a list of all the referrals pending allocation for that specialty

  - Be able to allocate individuals from the pending list to MDTs, either by:

        - Opening up the referral form details for a patient, and then allocating that patient to the correct MDT date.
        - Bulk selecting multiple cases from the list and allocating to an MDT date.

    - Be able to query an MDT referral (this would make most sense from within an individual referral, rather than in bulk):
        - Outright rejection, which sends an automated email to the user who referred the case.
            - This referral form is then moved to a “rejected” list, from which it can be retrieved if required, similar to retrieving email from the Trash.
        - Or change the MDT to another specialty.
        - Or easily send a query (? which generates an email ?) to the referrer asking for more details.

    - Have a secondary option of reviewing the list of allocated cases in upcoming MDTs of that specialty, so that I can:
        - Move patients between different dates (? by drag and drop ?)
        - Move patients within different sections within an MDT (eg renal, bladder, prostate)
            - This would probably mean that when patients are first added to the MDT, they would be in an “unallocated” section. By clicking on a referral in the list, it should show up the referral form details on the side, to help you decide which section to move it into.

Case Preparation Page

- As a user I would use this section to put in additional details about the case, which were not in the referral form, but which would be relevant for discussion during the MDT meeting.

- As a General Clinician, I would ideally like a mobile application which allows me to enter this data more seamlessly.
    - This is a minor issue (relevant for pathology and radiology), especially if we have a **responsive** site.
    - But it may be worth having a version of the site which is read only! This will act as the MDT list with small sections for personal notes only, which most people will use.

- Radiologists and pathologists would also have the role of General Clinician and have access to this screen, (and everyone would see

the same layout), but they would probably only fill in their relevant section.
    - Note: which user fills which section can be variable depending on the MDT, so we do not need to be more granular with the roles for now.
    - Can we find a way of having an easy representation of who wrote what and who made which changes?
        - Can you have a button that turns on/off colour coding, similar to google docs.
            - https://www.youtube.com/watch?annotation_id=annotation_747854663&feature=iv&src_vid=jmsHqL9GkSE&v=tZId5UlOGj8
        - Probably very hard, but I thought I would ask.

- As a General Clinician I would like the following in the general section:
    - A list of fields that needs to be completed - the list of fields needs to be discussed with Ben Goretzki.
    - A freetext box for general comments.
    - File / image upload.

- As a General Clinician I would like the following in the radiology section:
    - Freetext box.
    - File / image upload.
        - In the future, I would like a system to draw small diagrams on a mobile device, based on templates, similar to the radiology visual reporting app.

- As a General Clinician I would like the following in the pathology section:
    - Freetext box.
    - File / image upload.
        - In the future, I would like a system to draw small diagrams on a mobile device, based on templates.

Live MDT Page

- This section is for use during the MDT.
    - If multiple people have it open, how do quick can we update people's devices with new data?
    - The main list of fields should be obtained from Ben Goretzki.

- As a user, I should be able to look at the patient's other details (eg referral form, case preparation, other MDT discussions) very easily.

    - Fitting all this into an easily navigable layout will be the main challenge.

- Also as a user, I need the ability to add jobs and assign it to other users:
    - Click a button to add a new job
    - Type out a freetext description of the job
    - Assign job to a user:
        - This would then go on the user's jobs list. At the end of the day, there should also be an automated email to the user if there are outstanding jobs on their list.
        - If the target user is not registered on the same, I would allocate the job to myself, to remind me to contact the user by other means.

Confirmation Page

- List all cases requiring sign off, default arranged by MDT date.
    - Can be re-ordered by other fields too.

- As an MDT chair, I should be able to click on a case, which shows the referral form fields, prep and outcome in an ergonomic fashion.
    - Should have ability to click "Sign off" to complete the case and take off the list.
    - Should have ability to add new jobs (as in the Live MDT section).
    - Should have the ability to tag a case for further review and also add freetext notes.
        - Maybe even upload a file / image.

- As an MDT chair, there should be a separate sections where:
    - I can view the flagged cases.
    - I can see the history of cases I have signed off (starting with the last few cases), and review their details.

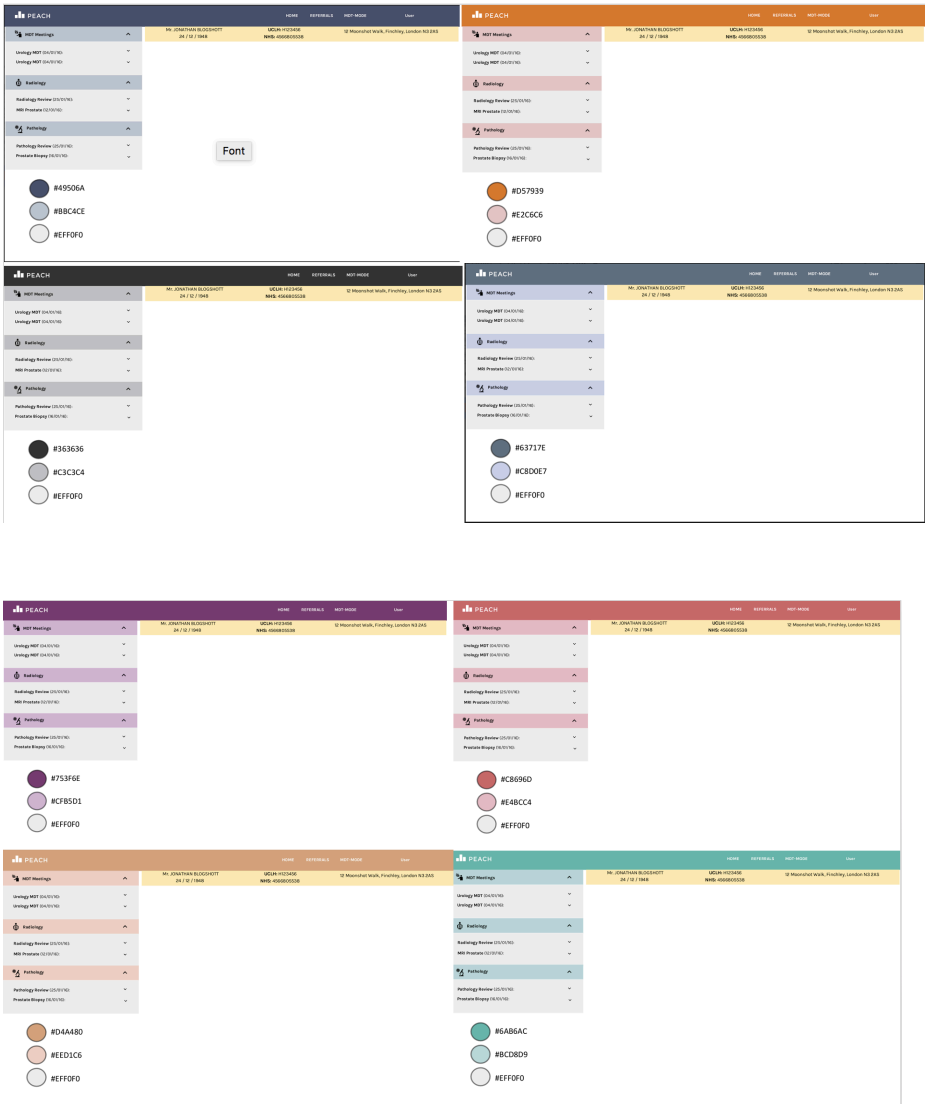
Jobs Page

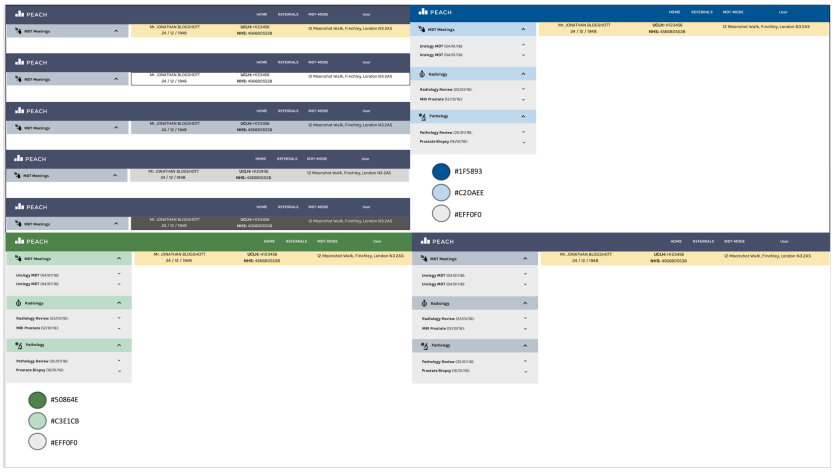
- As a user I should be able to see the jobs assigned to me:
    - Selection box, Date/Time, Patient name, Hospital number, Job description, "Job Completed" button.
    - Take job off list by clicking the "Job Completed" button, or selecting multiple lines (selection box) and clicking a button to complete job

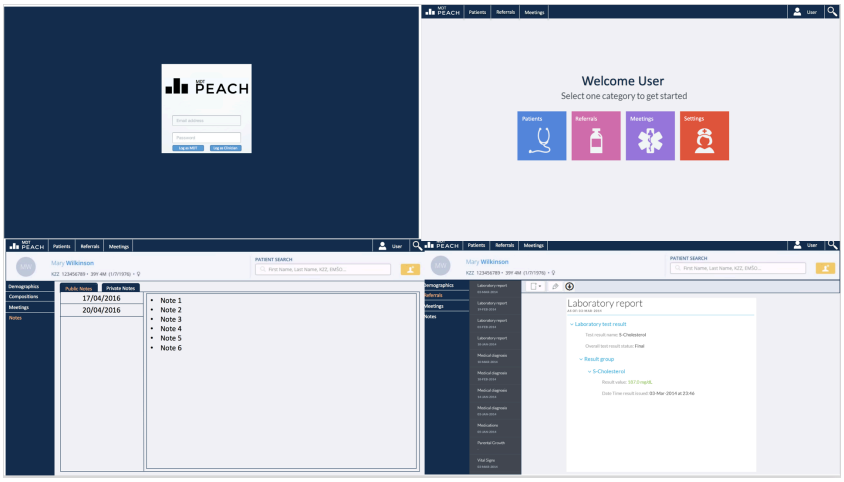
Jobs Page

- As a user I should be able to see the jobs assigned to me:
  - Selection box, Date/Time, Patient name, Hospital number, Job description, "Job Completed" button.
  - Take job off list by clicking the "Job Completed" button, or selecting multiple lines (selection box) and clicking a button to complete job
    - This should trigger a modal to confirm the action.
- As a user I should be able to click on a job to see the MDT record (referral form, prep and live MDT outcome) from the which the job was created.
- As a user, I should be able to see the history of previously completed jobs (starting with the last few cases), and review their details.

figure c.4




#1F5893
#C2DAEE
#EFF0F0
#50864E
#C3E1CB
#EFF0F0


PEACH
Welcome User
Select one category to get started
Patients
Referrals
Meetings
Settings
Mary Wilkinson
17/04/2016
20/04/2016
Note 1
Note 2
Note 3
Note 4
Note 5
Note 6
Laboratory report

Section D

Figure d.1

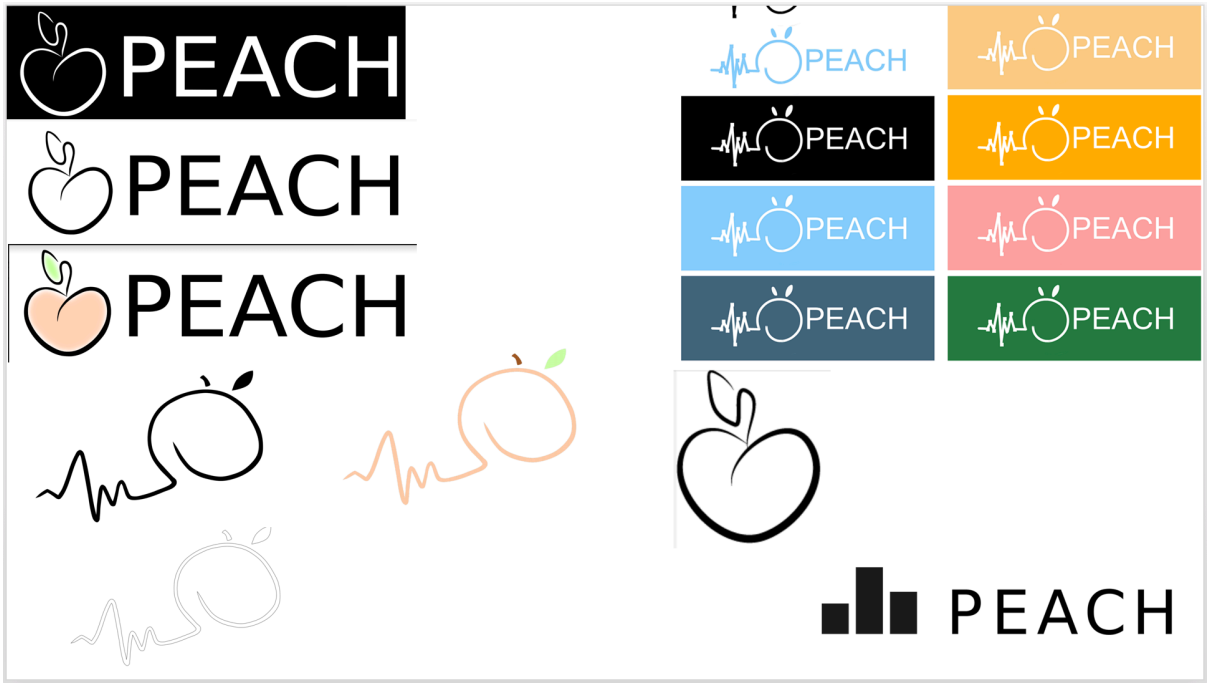
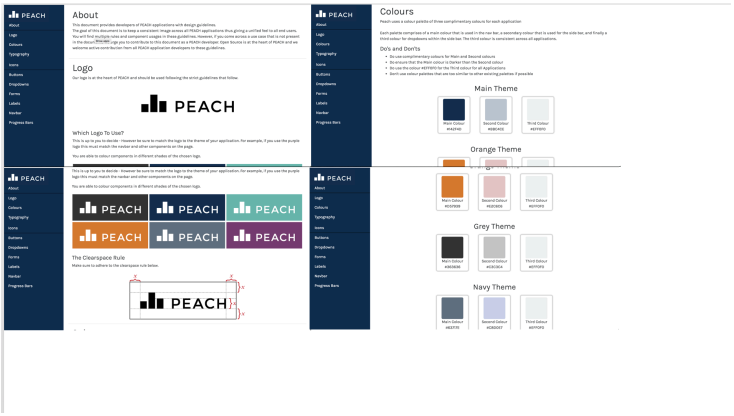
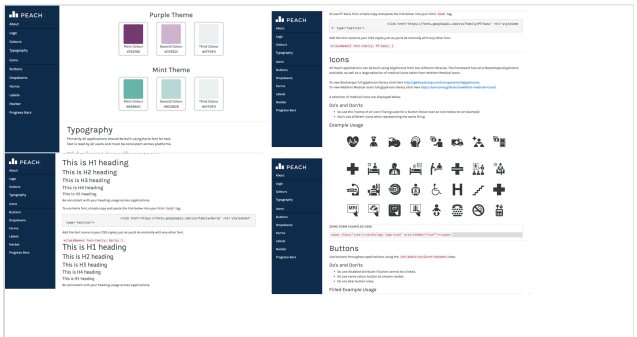
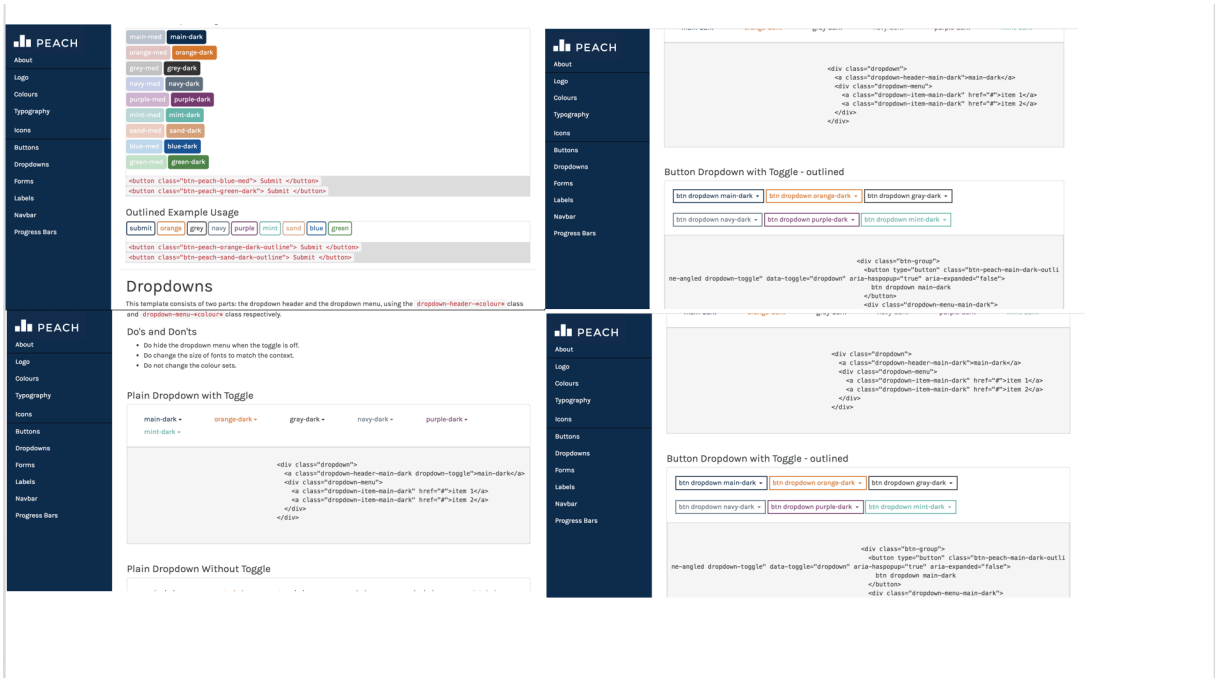


Figure d.2






PEACH
About
Logo
Colours
Typography
Icons
Buttons
Dropdowns
Forms
Labels
Navbar
Progress Bars
main-med
main-dark
orange-med
orange-dark
grey-med
grey-dark
navy-med
navy-dark
purple-med
purple-dark
mint-med
mint-dark
sand-med
sand-dark
blue-med
blue-dark
green-med
green-dark
<button class="btn-peach-blue-med"> Submit </button>
<button class="btn-peach-green-dark"> Submit </button>
Outlined Example Usage
submit
orange
grey
navy
purple
mint
sand
blue
green
<button class="btn-peach-orange-dark-outline"> Submit </button>
<button class="btn-peach-sand-dark-outline"> Submit </button>
Dropdowns
This template consists of two parts: the dropdown header and the dropdown menu, using the dropdown-header--<colour> class and dropdown-menu-<colour> class respectively.
Do's and Don'ts
Do hide the dropdown menu when the toggle is off.
Do change the size of fonts to match the context.
Do not change the colour sets.
Plain Dropdown with Toggle
main-dark
orange-dark
gray-dark
navy-dark
purple-dark
mint-dark

main-dark

item 1
item 2

Plain Dropdown Without Toggle

main-dark

item 1
item 2

Button Dropdown with Toggle - outlined
btn dropdown main-dark
btn dropdown orange-dark
btn dropdown gray-dark
btn dropdown navy-dark
btn dropdown purple-dark
btn dropdown mint-dark

<button type="button" class="btn-peach-main-dark-outline-angled dropdown-toggle" data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">
btn dropdown main-dark
</button>

Button Dropdown with Toggle - outlined
btn dropdown main-dark
btn dropdown orange-dark
btn dropdown gray-dark
btn dropdown navy-dark
btn dropdown purple-dark
btn dropdown mint-dark

<button type="button" class="btn-peach-main-dark-outline-angled dropdown-toggle" data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">
btn dropdown main-dark
</button>

Button Dropdown Without Toggle - filled
btn dropdown main-dark
btn dropdown orange-dark
btn dropdown gray-dark
btn dropdown navy-dark
btn dropdown purple-dark
btn dropdown mint-dark

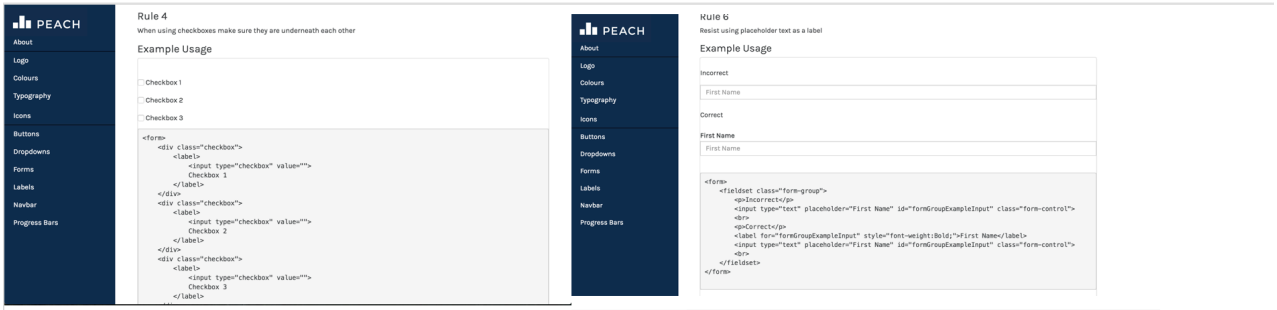
<button type="button" class="btn-peach-main-dark-fill-angled" data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">
btn dropdown main-dark
</button>

Action
Another action
Something else here

Separated link

Forms
Below you will find rules for form designs.
Rule 1
Keep forms in a single column.
Example Usage
First Name
First Name...
Last Name
Last Name...
I am in a vertical form.
Submit
<form>
<fieldset class="form-group">
<label for="formGroupExampleInput">
<input type="text" placeholder="F
<label for="formGroupExampleInput">
<input type="text" placeholder="L
</fieldset>

<label>
<input type="checkbox" value="
Rule 2
Top align labels
Example Usage
First Name
First Name...
<form>
<fieldset class="form-group">
<label for="formGroupExampleI
<input type="text" placeholde
</fieldset>
</form>
Rule 3
Avoid caps on labels
Example Usage
FIRST NAME First Name
First Name...
<form>


PEACH
About
Logo
Colours
Typography
Icons
Buttons
Dropdowns
Forms
Labels
Navbar
Progress Bars
Rule 4
When using checkboxes make sure they are underneath each other
Example Usage
Checkbox 1
Checkbox 2
Checkbox 3
<form>

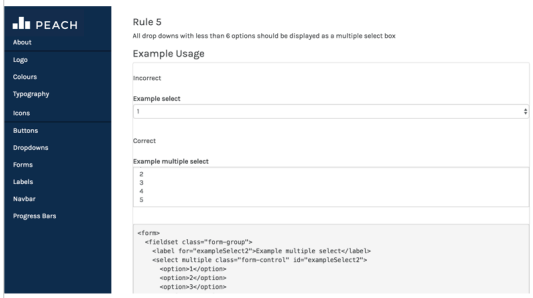
<label>
<input type="checkbox" value="">
Checkbox 1
</label>

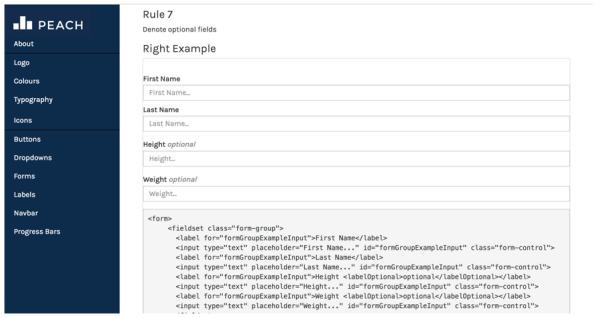
<label>
<input type="checkbox" value="">
Checkbox 2
</label>

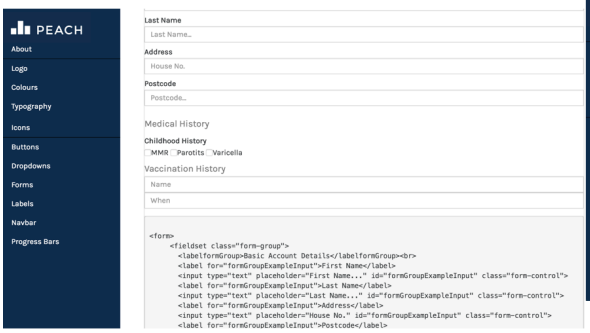
<label>
<input type="checkbox" value="">
Checkbox 3
</label>
Rule 6
Resist using placeholder text as a label
Example Usage
Incorrect
First Name
Correct
First Name
First Name
<form>
<fieldset class="form-group">
Incorrect
<input type="text" placeholder="First Name" id="formGroupExampleInput" class="form-control">

Correct
<label for="formGroupExampleInput" style="font-weight:Bold;">First Name</label>
<input type="text" placeholder="First Name" id="formGroupExampleInput" class="form-control">

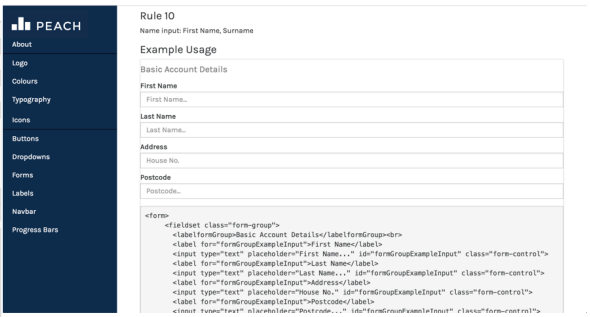
</fieldset>
</form>


PEACH
About
Logo
Colours
Typography
Icons
Buttons
Dropdowns
Forms
Labels
Navbar
Progress Bars
Rule 5
All drop downs with less than 6 options should be displayed as a multiple select box
Example Usage
Incorrect
Example select
1
Correct
Example multiple select
2
3
4
5
<form>
<fieldset class="form-group">
<label for="exampleSelect2">Example multiple select</label>
<select multiple class="form-control" id="exampleSelect2">
<option>1</option>
<option>2</option>
<option>3</option>


PEACH
About
Logo
Colours
Typography
Icons
Buttons
Dropdowns
Forms
Labels
Navbar
Progress Bars
Rule 7
Denote optional fields
Right Example
First Name
First Name...
Last Name
Last Name...
Height optional
Height...
Weight optional
Weight...
<form>
<fieldset class="form-group">
<label for="formGroupExampleInput">First Name</label>
<input type="text" placeholder="First Name..." id="formGroupExampleInput" class="form-control">
<label for="formGroupExampleInput">Last Name</label>
<input type="text" placeholder="Last Name..." id="formGroupExampleInput" class="form-control">
<label for="formGroupExampleInput">Height <labelOptional>optional</labelOptional></label>
<input type="text" placeholder="Height..." id="formGroupExampleInput" class="form-control">
<label for="formGroupExampleInput">Weight <labelOptional>optional</labelOptional></label>
<input type="text" placeholder="Weight..." id="formGroupExampleInput" class="form-control">
Wrong Example
First Name *
First Name...
Last Name *
Last Name...
Height
Height...
Weight
Weight...
Rule 8
Group related information.
Example Usage
Basic Account Details
First Name
First Name...
Last Name
Last Name...


PEACH
About
Logo
Colours
Typography
Icons
Buttons
Dropdowns
Forms
Labels
Navbar
Progress Bars
Last Name
Last Name...
Address
House No.
Postcode
Postcode...
Medical History
Childhood History
MMR Parotitis Varicella
Vaccination History
Name
When
<form>
<fieldset class="form-group">
<labelformGroup>Basic Account Details</labelformGroup>
<label for="formGroupExampleInput">First Name</label>
<input type="text" placeholder="First Name..." id="formGroupExampleInput" class="form-control">
<label for="formGroupExampleInput">Last Name</label>
<input type="text" placeholder="Last Name..." id="formGroupExampleInput" class="form-control">
<label for="formGroupExampleInput">Address</label>
<input type="text" placeholder="House No." id="formGroupExampleInput" class="form-control">
<label for="formGroupExampleInput">Postcode</label>


PEACH
About
Logo
Colours
Typography
Icons
Buttons
Dropdowns
Forms
Labels
Navbar
Progress Bars
Rule 9
Date of Birth as dropdowns.
Example Usage
Basic Account Details
First Name
First Name...
Last Name
Last Name...
Date of Birth
DD
MM
YYYY
Address
House No.
Postcode
Postcode...
<form>
<fieldset class="form-group">
<labelformGroup>Basic Account Details</labelformGroup>


PEACH
About
Logo
Colours
Typography
Icons
Buttons
Dropdowns
Forms
Labels
Navbar
Progress Bars
Rule 10
Name input: First Name, Surname
Example Usage
Basic Account Details
First Name
First Name...
Last Name
Last Name...
Address
House No.
Postcode
Postcode...
<form>
<fieldset class="form-group">
<labelformGroup>Basic Account Details</labelformGroup>
<label for="formGroupExampleInput">First Name</label>
<input type="text" placeholder="First Name..." id="formGroupExampleInput" class="form-control">
<label for="formGroupExampleInput">Last Name</label>
<input type="text" placeholder="Last Name..." id="formGroupExampleInput" class="form-control">
<label for="formGroupExampleInput">Address</label>
<input type="text" placeholder="House No." id="formGroupExampleInput" class="form-control">
<label for="formGroupExampleInput">Postcode</label>

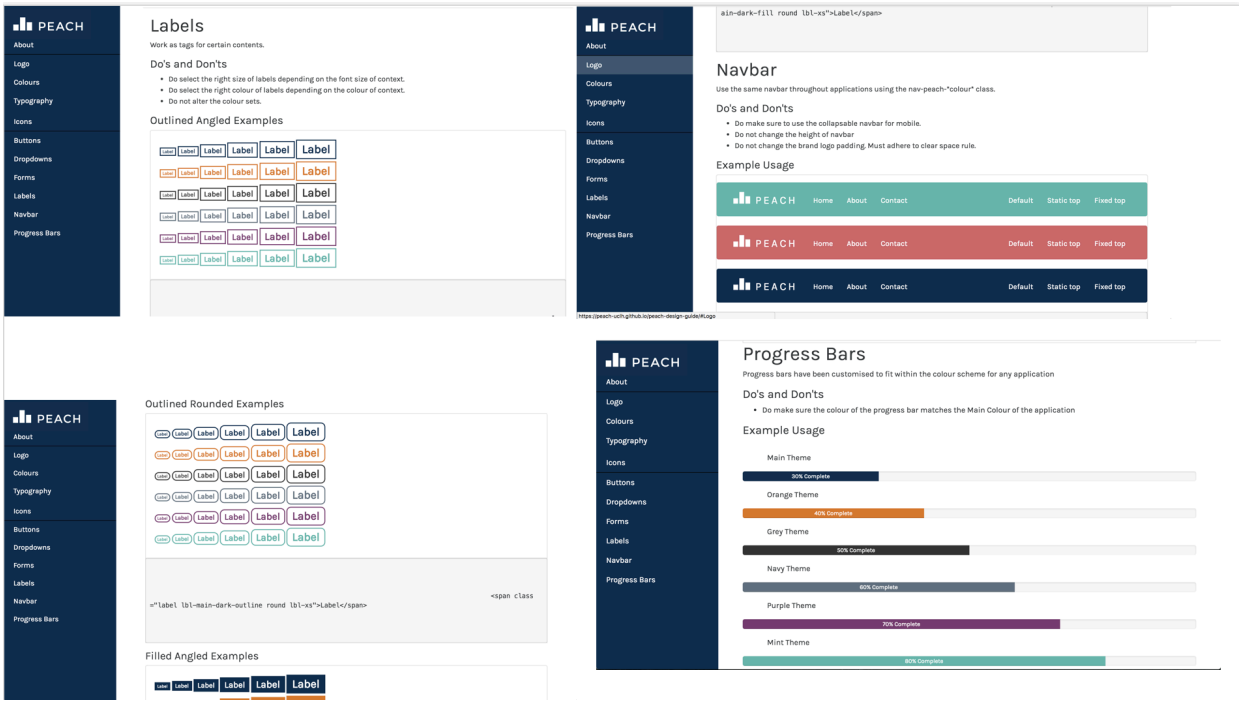
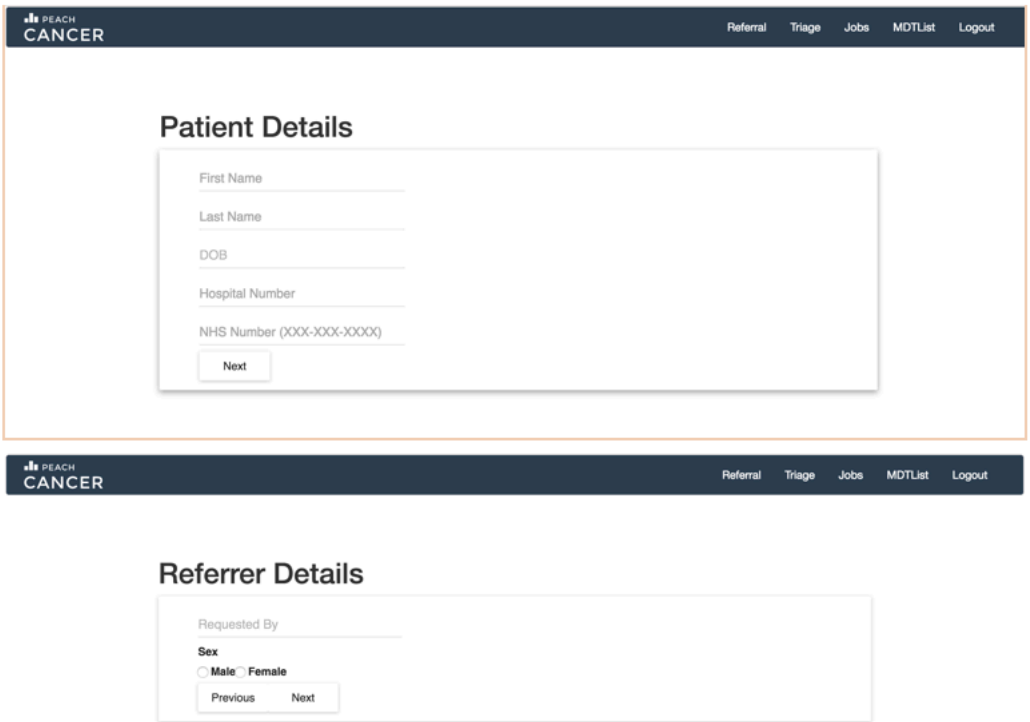

PEACH
About
Logo
Colours
Typography
Icons
Buttons
Dropdowns
Forms
Labels
Navbar
Progress Bars
Labels
Work as tags for certain contents.
Do's and Don'ts
Do select the right size of labels depending on the font size of context.
Do select the right colour of labels depending on the colour of context.
Do not alter the colour sets.
Outlined Angled Examples
Label
Navbar
Use the same navbar throughout applications using the nav-peach-*colour* class.
Do's and Don'ts
Do make sure to use the collapsable navbar for mobile.
Do not change the height of navbar
Do not change the brand logo padding. Must adhere to clear space rule.
Example Usage
Home
About
Contact
Default
Static top
Fixed top
Outlined Rounded Examples
Filled Angled Examples
Progress Bars
Progress bars have been customised to fit within the colour scheme for any application
Do's and Don'ts
Do make sure the colour of the progress bar matches the Main Colour of the application
Example Usage
Main Theme
30% Complete
Orange Theme
40% Complete
Grey Theme
50% Complete
Navy Theme
60% Complete
Purple Theme
70% Complete
Mint Theme
80% Complete

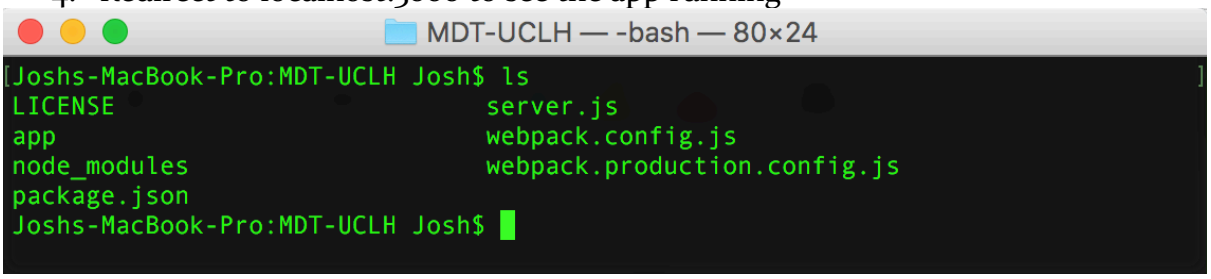
figure d.3


PEACH
CANCER
Referral
Triage
Jobs
MDTList
Logout
Patient Details
First Name
Last Name
DOB
Hospital Number
NHS Number (XXX-XXX-XXXX)
Next
Referrer Details
Requested By
Sex
Male
Female
Previous
Next

Section E - System Manual

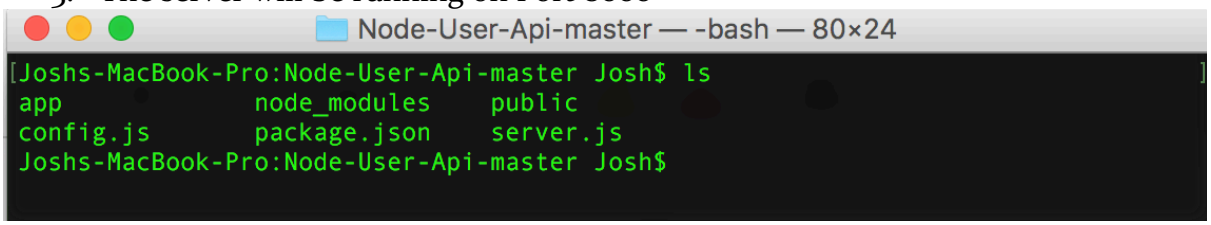
To run the MDT application code:

1. Cd into MDT-UCLH directory
2. Type in npm start and press enter
3. Launch a web browser
4. Redirect to localhost:3000 to see the app running



To run the MDT mock authentication API code:

1. Cd into MDT-UCLH-API directory
2. Type in Node Server.js
3. The server will be running on Port 8000



Section F – Code Listing for front-end

app/index.js

```
14  // Needed for onTouchTap
15  // http://stackoverflow.com/a/34015469/988941
16  injectTapEventPlugin();
17
18
19  const store = configureStore();
20  console.log('store', store)
21
22  const history = syncHistoryWithStore(browserHistory, store);
23
24  render(
25      <MuiThemeProvider>
26      <AppContainer>
27          <Root store={store} history={history} />
28      </AppContainer>
29      </MuiThemeProvider>,
30      document.getElementById('root')
31  );
32
33  if (module.hot) {
34      module.hot.accept('./containers/Root', () => {
35          const NewRoot = require('./containers/Root').default;
36          render(
37              <AppContainer>
38                  <NewRoot store={store} history={history} />
39              </AppContainer>,
40              document.getElementById('root')
41          );
42      });
43  }
44
```

app/routes.js

```
 8 ▼ export default (
 9 ▼   <Route path="/" component={App}>
10      <IndexRoute component={FilterableTable} />
11      <Route path="/about" component={About} />
12      <Route path="/Referral" component={Referral} />
13    </Route>
14  );
15
```

utils/index.js

```
export function createConstants(...constants) {
    return constants.reduce((acc, constant) => {
        acc[constant] = constant;
        return acc;
    }, {});
}
```

store/configureStore.dev.js

```
//own middleware
const logger = store => next => action => {
  console.info('dispatching', action)
  let result = next(action)
  console.log('next state', store.getState())
  return result
}



const enhancer = compose(
  applyMiddleware(thunk, logger),
  DevTools.instrument()
  )

export default function configureStore(initialState) {
    const store = createStore(
        rootReducer,
        initialState,
        enhancer
    );
    return store;
}
```

store/configureStore.prod.js

```
 export default function configureStore(initialState) {
     return createStore(
         rootReducer,
         initialState
     );
 }

```

reducers/auth.js

```
const initialState = {
  uid: '',
  role: '',
  isAuthenticated: false,
  isAuthenticating: false,
  statusText: ''
}

export default function auth(state=initialState, action){
  switch(action.type){
    case authConstants.LOGIN_REQUEST:
      return {...state,
        'isAuthenticating': true
      }
    case authConstants.LOGIN_SUCCESS:
      return {...state,
        'isAuthenticating': false,
        'isAuthenticated': true,
        'uid': action.payload.uid,
        'role': action.payload.role,
        'statusText': ''
      }
    case authConstants.LOGIN_FAILURE:
      return {...state,
        isAuthenticating: false,
        isAuthenticated: false,
        uid: '',
        role: '',
        statusText: 'Auth error: ' + action.payload.statusText + ". CODE = " + action.payload.status
      }
    case authConstants.LOGOUT:
      return {...state,
        isAuthenticated:false,
        isAuthenticating:false,
        uid:'',
        role:'',
        statusText: ''
      }
    default :
      return state;
  }
}
```

reducers/index.js

```
const filter = (state = '', action) => {
    switch (action.type) {
        case types.FILTER:
            return action.filter;
        default:
            return state;
    }
};


const rootReducer = combineReducers({
    filter,
    routing,
    auth,
    form: formReducer
});

export default rootReducer;
```

containers/DevTools.js

```
6   export default createDevTools(
7     <DockMonitor toggleVisibilityKey="ctrl-h"
8                  changePositionKey="ctrl-w">
9       <LogMonitor />
10    </DockMonitor>
11  );
12
```

containers/NavbarContainer.js

```
11  class NavbarContainer extends Component {
12
13      render() {
14        const { user } = this.props;
15        console.log(this.props.onLogin)
16          return (
17        <NavbarType>
18          <NavbarHeader />
19            <NavbarCollapseWrapper>
20              <NavbarListNavigation
21                        role={user.role}
22                        onLogin={this.props.onLogin}
23                        onLogout={this.props.onLogout}
24                      />
25            </NavbarCollapseWrapper>
26        </NavbarType>
27        );
28      }
29  }
30
31
32  const mapStateToProps = (state) => {
33    return {
34      user: state.auth
35    }
36  }
37
38  const mapDispatchToProps = (dispatch) =>{
39    return {
40      onLogin: (email, password) => dispatch(loginUser(email, password)),
41      onLogout: () => dispatch(logout())
42    }
43  }
44
45  export default connect(
46    mapStateToProps,
47    mapDispatchToProps
48    )(NavbarContainer)
```

containers/referral.js

```
class mdtReferral extends Component {
  constructor(props) {
    super(props)

    // Pro tip: The best place to bind your member functions is in the component constructor
    this.nextPage = this.nextPage.bind(this)
    this.previousPage = this.previousPage.bind(this)
    this.state = {
      page: 1
    }
  }

  nextPage() {
    this.setState({ page: this.state.page + 1 })
  }

  previousPage() {
    this.setState({ page: this.state.page - 1 })
  }

  render() {
    const { onSubmit } = this.props
    const { page } = this.state
    return (<div>
        {page === 1 && <WizardFormFirstPage onSubmit={this.nextPage}/>}
        {page === 2 && <WizardFormSecondPage previousPage={this.previousPage} onSubmit={this.nextPage}/>}
        {page === 3 && <WizardFormThirdPage previousPage={this.previousPage} onSubmit={onSubmit}/>}
      </div>
    )
  }
}

// WizardForm.propTypes = {
//   onSubmit: PropTypes.func.isRequired
// }

export default mdtReferral
```

containers/root.dev.js

```
  export default class Root extends Component {
      render() {
          const { store, history } = this.props;
          return (
              <Provider store={store}>
                  <div>
                      <Router history={history} routes={routes} />
                      <DevTools />
                  </div>
              </Provider>
          );
      }
  }

  Root.propTypes = {
      store: PropTypes.object.isRequired,
      history: PropTypes.object.isRequired
  };

```

containers/root.jsc

```
if (process.env.NODE_ENV === 'production') {
    module.exports = require('./Root.prod');
} else {
    module.exports = require('./Root.dev');
}
```

containers/root.prod.js

```
export default class Root extends Component {
    render() {
        const { store, history } = this.props;
        return (
            <Provider store={store}>
                <Router history={history} routes={routes} />
            </Provider>
        );
    }
}

Root.propTypes = {
    store: PropTypes.object.isRequired,
    history: PropTypes.object.isRequired
};
```

constants/auth.js

```
export default createConstants(
  'LOGIN_REQUEST',
  'LOGIN_FAILURE',
  'LOGIN_SUCCESS',
  'LOGOUT'
  )
```

components/app.js

```
class App extends Component {
    render() {
        const { children } = this.props;
        return(
        <div>
            <Navbar />
            { children }
        </div>
        );
    }
}

App.propTypes = {
    children: PropTypes.object
};

export default App;

```

components/mdt-page-1.js

```
const validate = values => {
  const errors = {}
  if (!values.firstName) {
    errors.firstName = 'Required'
  } else if (values.firstName.length > 10) {
    errors.firstName = 'Must be 10 characters or less'
  }
  if (!values.lastName) {
    errors.lastName = 'Required'
  } else if (values.lastName.length > 25) {
    errors.lastName = 'Must be 25 characters or less'
  }
  return errors
}
var formStyles = {
  paddingRight: '15%',
  paddingLeft: '15%',
  paddingTop: '3%',
  paddingBottom: '3%'
}
var divStyles = {
  paddingLeft: '50px',
  paddingTop: '12px',
  paddingBottom: '12px'
}

class mdtPageOne extends Component {
  render() {
    const {
      fields: { firstName, lastName, dob, hospitalNumber, nhsNumber },
      handleSubmit
    } = this.props
    return (<form style={formStyles} onSubmit={handleSubmit}>
      <h1>Patient Details</h1>
        <Paper zDepth={2} style={divStyles}>
        <div>
          <div>
            <TextField type="text" placeholder="First Name" {...firstName}/>
          </div>
          {firstName.touched && firstName.error && <div>{firstName.error}</div>}
        </div>
        <div>
          <div>
            <TextField type="text" placeholder="Last Name" {...lastName}/>
          </div>
          {lastName.touched && lastName.error && <div>{lastName.error}</div>}
        </div>
         <div>
          <div>
            <DatePicker hintText='DOB'/>
          </div>
        </div>
        <div>
          <div>
            <TextField type="text" placeholder="Hospital Number" {...hospitalNumber}/>
          </div>
        </div>
        <div>
          <div>
            <TextField type="text" placeholder="NHS Number (XXX-XXX-XXXX)" {...nhsNumber}/>
          </div>
        </div>
        <div>
```

```
            <RaisedButton type="submit">
              Next <i/>
            </RaisedButton>
          </div>
          </Paper>
        </form>
      )
    }
  }

  mdtPageOne.propTypes = {
    fields: PropTypes.object.isRequired,
    handleSubmit: PropTypes.func.isRequired
  }

  export default reduxForm({
    form: 'wizard',                 // <------ same form name
    fields,                         // <------ only fields on this page
    destroyOnUnmount: false     // <------ preserve form data
                                    // <------ only validates the fields on this page
  })(mdtPageOne)
```

components/mdt-page-2.js

```
const validate = values => {
  const errors = {}
  if (!values.email) {
    errors.email = 'Required'
  }
  if (!values.sex) {
    errors.sex = 'Required'
  }
  return errors
}
var formStyles = {
  paddingRight: '15%',
  paddingLeft: '15%',
  paddingTop: '3%',
  paddingBottom: '3%'
}

var paperStyles = {
  paddingLeft: '50px',
  paddingTop: '12px',
  paddingBottom: '12px'
}

class mdtPageTwo extends Component {
  render() {
    const {
      fields: { requestedBy, email, sex },
      handleSubmit,
      previousPage
    } = this.props
    return (<form style={formStyles} onSubmit={handleSubmit}>
      <h1>Referrer Details</h1>
        <Paper style={paperStyles}>
        <div>
          <div>
            <TextField hintText={'Requested By'} {...requestedBy}/>
          </div>
        </div>
        <div>
          <label>Sex</label>
          <div>
            <label>
              <input type="radio" {...sex} value="male" checked={sex.value === 'male'}/> Male
            </label>
            <label>
              <input type="radio" {...sex} value="female" checked={sex.value === 'female'}/> Female
            </label>
          </div>
          {sex.touched && sex.error && <div>{sex.error}</div>}
        </div>
        <div>
          <RaisedButton type="button" onClick={previousPage}>
            <i/> Previous
          </RaisedButton>
          <RaisedButton type="submit">
            Next <i/>
          </RaisedButton>
        </div>
        </Paper>
      </form>
    )
  }
```

```
        </form>
      )
    }
  }

  mdtPageTwo.propTypes = {
    fields: PropTypes.object.isRequired,
    handleSubmit: PropTypes.func.isRequired,
    previousPage: PropTypes.func.isRequired
  }

  export default reduxForm({
    form: 'mdtReferral',                // <------ same form name
    fields,                         // <------ only fields on this page
    destroyOnUnmount: false,        // <------ preserve form data
    validate                        // <------ only validates the fields on this page
  })(mdtPageTwo)
```

components/common/login.js

```
var styles = {
  loginButton: {
    marginTop: '8px',
    marginLeft: '8px'
  }
}

export default class Login extends Component {

  render() {
    const { errorMessage } = this.props

    return (
      <div className='login-input'>
        <input type='text' ref='username' className="form-control" placeholder='Username' style={styles.loginButton}/>
        <input type='password' ref='password' className="form-control" placeholder='Password' style={styles.loginButton}/>
        <button onClick={(event) => this.handleClick(event)} className="btn btn-primary" style={styles.loginButton}>
          Login
        </button>

        {errorMessage &&
          <p>{errorMessage}</p>
        }
      </div>
    )
  }

  handleClick(event) {
    const username = this.refs.username
    const password = this.refs.password
    const creds = { username: username.value.trim(), password: password.value.trim() }
    this.props.login(creds.username, creds.password)
  }
}

Login.propTypes = {
  login: PropTypes.func.isRequired,
  errorMessage: PropTypes.string
}
```

components/common/logout.js

```
var logoutBtn = {
  color: 'white'
}
export default class Logout extends Component {
  render() {
    return (
        <li><a href='#' style={logoutBtn} onClick={this.props.logout}>Logout</a></li>
      )
  }
}
```

components/common/navbarcollapsewrapper.js

```
  export default class NavbarCollapseWrapper extends Component {
    render (){
      const { children } = this.props
      return(
      <div id="navbar" className="navbar-collapse collapse">
        {children}
      </div>
      )
    }
  }
```

components/common/navbarheader.js

```
var logoStyles = {
  width: '30%',
  marginTop: '-15px'
}
export default class NavbarHeader extends Component {
  render(){
    return(
          <div className="navbar-header">
            <button type="button" className="navbar-toggle collapsed" data-toggle="collapse" data-target="#navbar" aria-expanded="false" aria-co
              <span className="sr-only">Toggle navigation</span>
              <span className="icon-bar"></span>
              <span className="icon-bar"></span>
              <span className="icon-bar"></span>
            </button>
            <Link className="navbar-brand" to="/"><img src='http://uploads.webflow.com/57a668ea3e4192b946e35418/57afa3a1e18710ca164835fe_peach-c
                                                  style={logoStyles}/></Link>
          </div>
    )
  }
}
```

components/common/navbarlistnavigation.js

```
const navbars = {
      mdtCoordinator: ['Referral', 'Triage', 'Live MDT', 'MDT List'],
      referrer: ['Referral', 'Triage', 'Jobs'],
      clinician: ['Referral', 'Triage', 'Jobs', 'MDT List'],
      admin: ['Accounts', 'App Settings', 'Customisation']
};
//cant put in NavbarListNavigation.propTypes here ?

  var listStyles = {
    color: 'white'
  };


export default class NavbarListNavigation extends Component {
  constructor(props){
    super(props)

  }


  buildNav (role){
    let nav = [];
    if (role){
    nav = navbars[role].map((label)=>{
      label = label.replace(/\s/g, '');
      return (
          <li key={label}><Link className='nav-links' to={label} style={listStyles}>{label}</Link></li>
        )
    })
      nav.push(<Logout key={'Logout'} logout={this.props.onLogout}/>)
    }
    else{
      nav.push(<Login key={'Login'} login={this.props.onLogin} />)
    }
    return nav;
  }

  render (){
    const { role } = this.props;
    // const dynamicLogin = (role) ? <Logout /> : <Login />
    return(
    <ul className="nav navbar-nav pull-right">
      {this.buildNav(role)}
    </ul>
    )
  }
}
```

components/common/navbartype.js

```
var navStyle = {
  backgroundColor: '#2d3e4f'
}
export default class NavbarHeader extends Component {
  render(){
    const { children } = this.props
    return(
          <nav className="navbar navbar-default" style={navStyle}>
            <div className="container-fluid">
            { children }
            </div>
          </nav>
      )
  }
}
```

actions/auth.js

```
export function loginUserSuccess(data){
  //set token in localStorage
  localStorage.setItem('uid', data._id);
  localStorage.setItem('role', data.role);
  console.log(data)
  return {
    type: authConstants.LOGIN_SUCCESS,
    payload: {
      uid: data._id,
      email: data.email,
      role: data.role
    }
  }
}

export function loginUserFailure(error){
  //remove token in localStorage
  return {
    type: authConstants.LOGIN_FAILURE,
    payload:{
      status: error.response.status,
      statusText: error.response.statusText
    }
  }
}

export function loginUserRequest(){
  return {
    type: authConstants.LOGIN_REQUEST
  }
}

export function logout(){
  localStorage.removeItem("uid");
  localStorage.removeItem("role");
  return {
    type: authConstants.LOGOUT
  }
}

export function loginUser(email, password) {
  return function(dispatch){
    try{
    dispatch(loginUserRequest());
    // const token = api.getToken(email,password);
    axios.post('http://localhost:8000/api/login', {
```

```
      username: email,
      password: password
    })
    .then(function (response) {
      if(response.data.token){
        var data = decoded(response.data.token)
        dispatch(loginUserSuccess(data));
      } else {
        console.log(response.data.message)
      }

    })
    .catch(function (error) {
      dispatch(loginUserFailure({
          response:{
            status: 403,
            statusText: 'Invalid username and password'
          }
        }))
    });
      } catch (e){
        dispatch(loginUserFailure(e))
      }
  }
}

```

actions/index.js

```
export function filterTable(filter) {
    return {
        type: types.FILTER,
        filter
    };
}

```

Section F – Code Listing for back-end

models/user.js

```
var UserSchema = new Schema({
  email: { type: String, required: true, index: { unique: true }},
  username: { type: String, required: true, index: { unique: true }},
  password: { type: String, required: true, select: false},
  role: { type: String, required: true }
});

UserSchema.pre('save', function(next){
  var user = this;

  if(!user.isModified('password')) return next();

  bcrypt.hash(user.password, null, null, function(err, hash){
    if(err) return next(err);

    user.password = hash;
    next();
  });
});

UserSchema.methods.comparePassword = function(password){
  var user = this;

  return bcrypt.compareSync(password, user.password);
}

module.exports = mongoose.model('User', UserSchema);
```

routes/api.js

```
function createToken(user){
  var token = jsonwebtoken.sign({
    _id: user._id,
    email: user.email,
    username: user.username,
    role: user.role
  }, secretKey, {
    expiresInMinute: 1440
  });

  return token;
};

module.exports = function(app, express){
  var api = express.Router();

  api.post('/signup', function(req, res){
    var user = new User({
      email: req.body.email,
      username: req.body.username,
      password: req.body.password,
      role: req.body.role
    });

  user.save(function(err){
    if(err){
      res.send(err);
      return;
    }
      res.json({message: 'User created'});
    });
  });

  api.get('/users', function(req, res){
    User.find({}, function(err, users){
      if(err){
        res.send(err);
        return;
      }

      res.json(users);

    });
  });


  api.post('/login', function(req, res){
    User.findOne({
```

```
  api.post('/login', function(req, res){
    User.findOne({
      username: req.body.username
    }).select('password').exec(function(err, user){

      if(err) throw err;

      if(!user){
        res.send({message: 'User doesnt exist'});
      } else if (user){
        var validPassword = user.comparePassword(req.body.password);

        if(!validPassword){
          res.send({message: 'Invalid password!'})
        } else {
          User.findOne({
            username: req.body.username
          }).exec(function(err, user){

          var token = createToken(user);

          res.json({
            success: true,
            message: "Successful login!",
            token: token
          });
        })
        }

      }

    });

  });

  api.use(function(req, res, next){
    console.log("Somebody just came to our app");

    var token = req.body.token || req.param('token') || req.headers['x-access-token'];

    console.log(token)
    if(token){
      jsonwebtoken.verify(token, secretKey, function(err, decoded){
        if(err){
          res.status(403).send({success: false, message: 'Authentication failed!'});
        }else{
          req.decoded = decoded;

          next();
        }
      });
    }else{
      res.status(403).send({success: false, message: "No token provided."});
    }
  });

  api.get('/me', function(req, res){
    res.json(req.decoded);
  })
  return api

}
```

server.js

```
10
11  mongoose.connect(config.database, function(err){
12    if(err){
13      console.log(err);
14    }else{
15      console.log('Connected to the database');
16    }
17  });
18
19  app.use(cors());
20
21  app.use(bodyParser.urlencoded({extended: true}));
22  app.use(bodyParser.json());
23  app.use(morgan('dev'));
24
25
26  app.use(express.static(__dirname + '/public'))
27
28  var api = require('./app/routes/api')(app, express);
29
30  app.use('/api', api);
31
32  app.get('*', function(req, res){
33    res.sendFile(__dirname + '/public/views/index.html');
34  });
35
36  app.listen(config.port, function(err){
37    if(err){
38      console.log(err);
39    }else{
40      console.log("Listening on port 8000");
41    }
42  })
```

Glossary

Action – *payload of information that send data from your application to your store.*

Container – *a component that is connected to a redux store.*

Dispatch – *sends actions to redux reducers to implement changes in store state.*

HTTP – *Hypertext Transfer Protocol is the set of rules for transferring files on the World Wide Web.*

Node – *Event-driven I/O server-side JavaScript environment based on V8.*

NPM – *Node Package Manager installs, publishes and manages Node programs*

Promise – *object used for asynchronous computations. It represents a value which may be available now, or in the future, or never.*

Reducer – *specify how the application's state changes in response to something happening.*

Store – *the object that brings all reducers together to hold application state.*

Bibliography

[1] Pelican Cancer Foundation. “What is an MDT”. http://www.pelicancancer.org/what-is-an-mdt, [Aug. 14, 2016].

[2] Professor Sir Mike Richards. “The Characteristics of an Effective Multidisciplinary Team (MDT)”. http://www.nhsiq.nhs.uk/media/2444560/ncatmdtcharacteristics.pdf, Feb. 2010. [Aug. 14, 2016]

[4] Constantine, L. L. (1995). Constantine on Peopleware. Englewood Cliffs, NJ: Yourdon Press

[5] Scott W. Ambler (Oct. 26, 2000). “The Object Primer 2nd Edition, Building Object Applications That Work and Process Patterns”. [Online]. Available: http://www2.fiit.stuba.sk/~bielik/courses/psi-slov/material/ui-design.pdf, [Aug. 14, 2016]

[6] Cancer Research UK. “Worldwide Cancer Statistics”. http://www.cancerresearchuk.org/health-professional/cancer-statistics/worldwide-cancer#heading-Zero, [Aug. 14, 2016]

[7] Kendall Scott. The Unified Process Explained, Addison Wesley, 2001, [Aug. 15, 2016]

[8] J. Arlow & Ila Neustadt. UML 2 and the Unified Process. Addison Wesley, 2005. [Aug. 15, 2016]

[9] E. Dornenburg & J. Nilsson. Test Driven Development (TDD) – an overview. (2005). [Aug. 15, 2016]

[10] NHS England. “Information Standards”. [Online]. Available: https://www.england.nhs.uk/ourwork/tsd/data-info/info-stand/, [Aug. 16, 2016]

[11] Dr. Laura DeNardis. “E-health Standards and Interoperability”. [Online]. Available: https://www.itu.int/dms_pub/itu-t/oth/23/01/T23010000170001PDFE.pdf, [Aug. 15, 2016]

[12] IHTSDO. “SNOMED CT & Other Terminologies , Classifications & Code Systems”. [Online]. Available: http://www.ihtsdo.org/snomed-ct/mapping-to-other-terminologies, [Aug. 18, 2016]

[13] OpenEHR. “What is OpenEHR”. [Online]. Available: http://www.openehr.org/what_is_openehr, [Aug. 18, 2016]

[14] Ethercis. “EtherCIS – an open source openEHR server” [Online.] Available: http://ethercis.github.io/, [Aug. 18, 2016]
[15] Silverlink Software. “SilverLink Software – Power through Interperability” [Online]. Available: http://silverlinksoftware.com/about-us/, [Aug. 18, 2016]
[16] NCIN. “Cancer Outcomes and Services Dataset” [Online]. Available: www.ncin.org.uk/view?rid=2030, [Aug.18, 2016]
[17] M. S. Mikowski & J. C. Powell. *Single Page Web Applications.* Shelter Island: Manning Publications Co. 2014, pp. 8.
[18] Michael Tinning. “A Functional Front-End With React” [Online]. Available: http://blog.scottlogic.com/2016/04/04/a-functional-front-end-with-react.html, [Aug. 19, 2016]
[19] Joannes Vietze. PDCA. [Online]. https://en.wikipedia.org/wiki/PDCA#/media/File:PDCA_Process.png, [Aug. 19, 2016]
[20] A. H. Farmer. “Understanding Webpack HMR”. [Online]. Available: http://andrewhfarmer.com/understanding-hmr/, [Aug. 19, 2016]
[21] Techopedia. “What does Thunk mean?” [Online]. Avaiable: https://www.techopedia.com/definition/2818/thunk
[22] D. Clegg & R. Barker. *Case Method Fast-Track: A RAD Approach.* [Aug. 18, 2016]
[23] Agile Modelling. “Personas” [Online]. Available: http://www.agilemodeling.com/artifacts/personas.htm, [Aug. 19, 2016]
[24] Techopedia. “Keep It Simple Stupid” [Online]. Available: https://www.techopedia.com/definition/20262/keep-it-simple-stupid-principle-kiss-principle, [Aug. 19, 2016]
[25] “Actions” [Online]. Available: http://redux.js.org/docs/basics/Actions.html, [Aug. 20, 2016]
[26] “Reducers” [Online]. Available: http://redux.js.org/docs/basics/Reducers.html, [Aug. 20, 2016]
[27] “Single Page Application”. [Online]. Available: https://en.wikipedia.org/wiki/Single-page_application#Browser_history, [Aug. 22, 2016]
[28] “Getting started with Redux-form” [Online]. Available: http://redux-form.com/6.0.1/docs/GettingStarted.md/, [Aug. 22, 2016]

[29] “Getting Started with Redux” [Online]. Available: https://egghead.io/lessons/javascript-redux-reducer-composition-with-combinereducers, [Aug. 22, 2016]

[30] M. Tuteja & G. Dubey (2012, July). “A Research Study on importance of Testing and Quality Assurance in Software Development Life Cycle (SDLC) Models”. 2(3). [Online]. Available: http://www.academia.edu/2304463/A_Research_Study_on_importance_of_Testing_and_Quality_Assurance_in_Software_Development_Life_Cycle_SDLC_Models, [Aug. 23, 2016]

[30] F. Shull, V. Basili, B. Boehm, A. Winsor Brown, P. Costa, M. Lindvall, D. Port, I. Rus, R. Testoriero & M. Zelkowski. “What We Have Learned About Fighting Defects”. [Online]. Available: http://www.cs.umd.edu/~mvz/pub/eworkshop02.pdf, [Aug. 25, 2016]

[31] K. Beck & Three Rivers Institute. “Test-Driven Development By Example”. [Online]. Available: http://www.eecs.yorku.ca/course_archive/2003-04/W/3311/sectionM/case_studies/money/KentBeck_TDD_byexample.pdf, [Aug. 25, 2016]