Providing mobile access to clinical data in challenging environments

Alex Scott (b2024874)

BSc Computer Science G401, May 2016

Supervisor: Dr. Phil Lord

Word Count: 14,698

Abstract

This dissertation explores and implements the functionality required to serve evidence based clinical guidance content to users in the presence of intermittent or non-functioning networks, in order to ensure that all users have access to this important information. The content is Produced by Clarity Informatics as part of their Prodigy service and serves GP's and primary healthcare professionals globally.

Background information on the importance of this content is discussed and a detailed design and implementation plan is walked-through before the system is tested and analysed with the results discussed at the end of the document.

Declaration

“I declare that this document represents my own work except where otherwise stated”

Alex Scott

Acknowledgments

I would like to thank my supervisor Phil Lord for his advice throughout this project, as well as Clarity Informatics for facilitating me whilst I developed the application.

1. Introduction

1.1 Chapter introduction

This chapter begins by describing the structure of the dissertation, it then sets the context of the project by describing the problem from which it is motivated, and shows how the aims and objectives were derived from this. Following this, a plan for the implementation of the project is shown in the form of a Gantt chart and the timeline is explained.

1.2 Dissertation structure

Chapter One identifies the problem this document is addressing and what aims and objectives have been derived from it.

Chapter Two goes into depth on where the data provided by Clarity originates from, why it is so important to present this content to all users, it also takes a look at competitors in the market and sources of similar information.

Chapter Three goes into depth on the design and physical implementation of the system. Discussing the reasoning behind each piece of functionality, how it was designed and shows code used to implement it as well as examples of each in use.

Chapter Four describes the testing methodologies carried out on the system, showing how the whole application was tested from the ground up. It discusses the results of the tests, which are listed in Appendix C.

Chapter Five assesses the projects successes and failures, it discusses how well the project has come to satisfy the aims and objectives outlined in section 1.4, as well as detailing how the project has gone from a personal perspective. It outlines possibilities for future work on the system and how it could be improved.

Chapter Six holds the details of all references used during the completion of this project, which are referred to in chapters one to five.

Chapter Seven contains the appendices of the documentation. The appendices are made up of 3 sections, A which holds general information (e.g. Glossary of terms), B which holds all figures referred to from the main document body, and C which contains the outputs from all tests carried out under Chapter four.

1.3 Definition of problem

1.3.1 Context

A healthcare informatics company based in the north-east of England named Clarity Informatics currently produce and maintain a large amount of high quality clinical data used by a large number of primary healthcare professionals globally to assist their day to day work. This data comes in the form of ‘Clinical Knowledge Summaries’ (hereby known as CKS) which are summaries of almost 350 different topics (e.g. asthma, meningitis, HIV) consisting of 1000+ clinical presentations or patient scenarios, the topics contents are presented in a common format making them an easily readable resource and are continually reviewed and updated [1].

Primary healthcare professionals (e.g GP’s) use the CKS to obtain specific information on different topics, sometimes in time constrained environments in order to make informed decisions on how best to treat a patient. They can view background information on the topic, different patient scenarios alongside management and prescribing information through their web browser.

1.3.2 Use Case

Consider Dr. Jones, a GP in rural Australia who spends half his time in the office and half his time conducting home visits for patients. He is an active user of CKS and somewhat relies on the information he obtains from the website to diagnose his patients. Dr. Jones receives a call from a patient a long distance away who would like him to visit, so he enquires as to the symptoms of the patient to ensure that it is worth his time to go. Dr. Jones agrees to visit and travels to the patient, he further listens to the patients symptoms - however he is unsure as to what exactly is wrong with the patient and so attempts to access the CKS over the browser of his laptop to confirm a diagnosis. Dr. Jones cannot access the CKS as he cannot gain access to an active internet connection - he fails to make a diagnosis and advises the patient that he will need to return to his office before issuing a prescription, prolonging a potentially critical diagnosis on the basis that he cannot access the information that he needs.

1.3.3 The Problem

Users of the CKS must have access to both a web browser and an active internet connection in order to view the clinical data. As presented in 1.3.2, you can see that there are a number of potential use cases in which users will not be able to satisfy these requirements, another example being healthcare professionals working in ‘no’ zones in hospitals which require that devices be operated on flight safe mode, users in these zones must . Some use cases involve users operating in a time constrained environment and must rely on a weak internet connection in order to fetch data, which may not be up to speed. It is important that primary healthcare professionals have fast and reliable access to the information that they require, and should not need to rely on a consistent internet connection to perform this task. There is a very large amount of data contained in the CKS, including multimedia, and so fetching this data may be too slow in a time constrained environment.

1.3.4 The Solution

Clarity have commissioned me to develop an end user system as a solution to this problem and to ensure that its users have access to this data when they need it, particularly in complex environments. I intend to develop a cross-platform mobile application capable of presenting all CKS data to the user in a responsive and accessible manner. The applications design will comply with NC State University’s Universal Principles of Design [2] in an effort to create a highly usable application that not only serves the requested information to the user but intelligently caches the data offline to account for the accessibility problems introduced with each use case.

1.4 Project Aims

To gauge the success of the project, an achievable aim has been defined, and from it a number of measurable lower level objectives. This section details this overall aim and lists each objective paired with a justification for it.

1.4.1 Aim

Get Clarity’s clinical data into the hands of healthcare professionals in challenging environments, via a cross platform mobile application.

This context aware mobile phone application will be fit for use on iOS, Android and Windows devices, and it will serve as a platform from which primary healthcare professionals can not only view all the topics and information contained in the CKS, but from which the contents can

be downloaded and stored for offline use. The app will serve relevant topics tailored for each specific users needs, as well as accounting for unforeseen events, whilst keeping the user informed and up to date in a user friendly and non-obtrusive manner.

1.4.2 Objectives

Objective 1: Rapid prototyping of the app, taking data from Clarity's API, hosting it on the web and serving it from there

An important part of the project will be its initial production into a state in which it can serve content from Clarity's Web API onto the application. Rapidly prototyping the app allows the project to quickly get into its research and design phase.

Objective 2: Research and implement the required functionality to serve data specifically accounting for intermittent or non functioning network connections

Accounting for users with lack of internet connection or in networks where the app might have a block on it is key, removing the barriers of access to the CKS for those users who cannot always rely on their connection.

Objective 3: Personalise the offline caching of data to tailor accessibility to individual users needs

Allowing users to interact and tailor their storage to their own needs is essential to the UX. Each user has different needs and requirements for the system.

Objective 4: Account for changing epidemics/diseases and the effect this will have on the users requirements for data access

Each year there are a number of disease outbreaks which spread, causing a hike in doctor referrals and panic. It is key that not only our users are aware of the outbreaks (particularly in their area) but that they have planned and are prepared for them.

Objective 5: Extensive user testing to obtain confirmation that each potential use case is satisfied with appropriate mechanisms

Assessing each use case individually and detailing the functionality required to satisfy it. It's key that these extreme use cases are accounted for, and the features present that are needed by the end user.

Objective 6: Research usability, accessibility and UX and construct a UI centred around these principles

The application will be used by a wide variety of users, some with technical expertise and some without, and so it is key that the UX and accessibility of the application allows for access by everyone. Removing the barriers of entry to this app by constructing a UI which does not discriminate against users.

1.5 Implementation plan

Figure 1 shows the Gantt chart used to structure this project. It details all major sections of work on the system, and provides a well thought through plan for delivering this product on time.

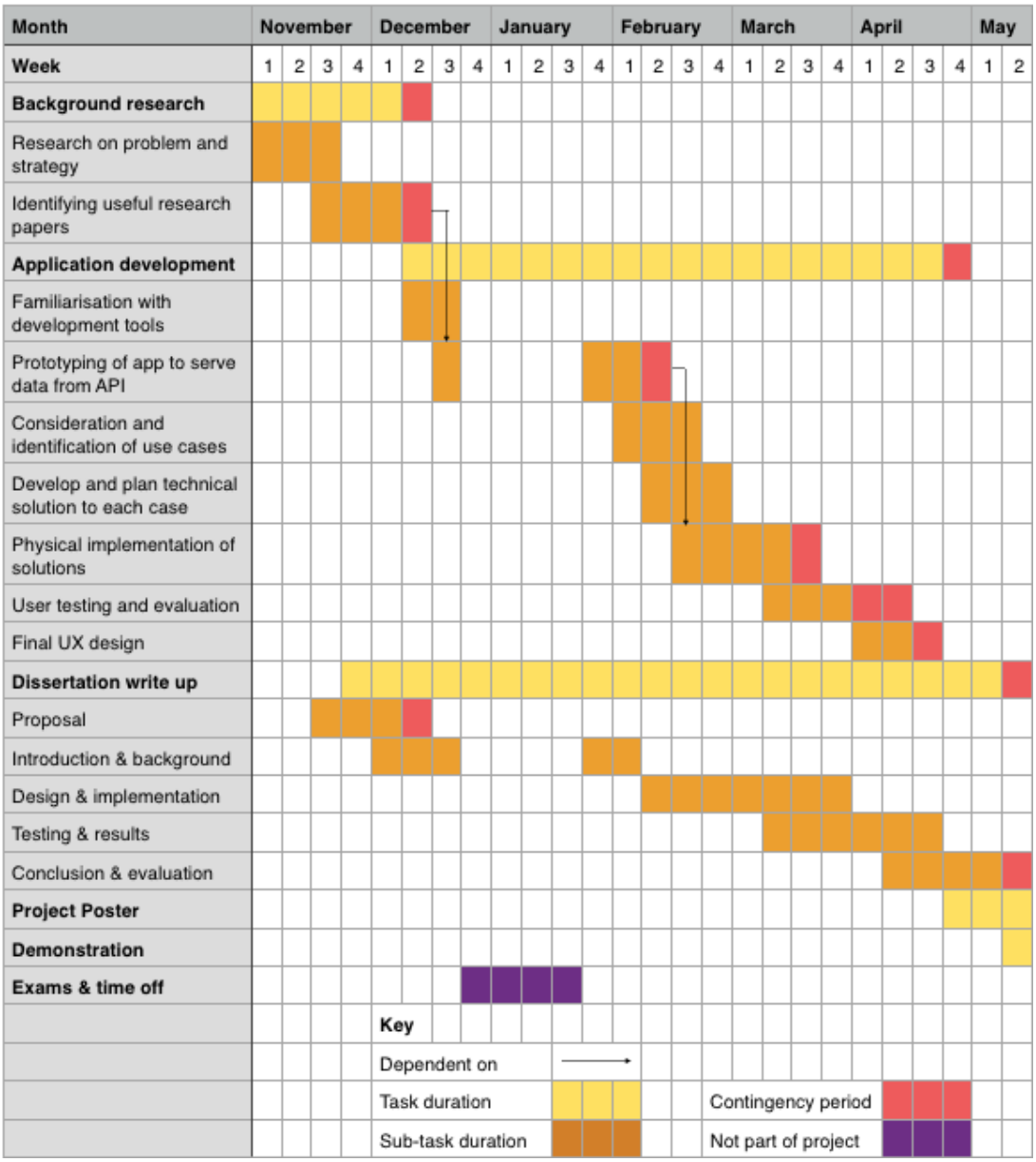


Figure 1: Work plan Gantt chart

2. Background research

2.1 Chapter introduction

Healthcare has long been important area of research, for centuries people have experimented and analysed different illnesses and conditions and documented their results. Fast forward to modern times in which we now possess a massive amount of collated clinical data, data with the ability to help treat people with a multitude of conditions and educate those who wish to learn or study. For centuries it was generally only the privileged who had access to modern literature on healthcare, and up until the boom of the internet in the 1990s [9] those who wished to study this literature required access to a school, university or extensive library. The platforms from which one can access this information have developed a long way since then and it is now feasible for anyone to grab their mobile phone or another internet enabled device and look it up online.

This section describes the type of clinical content presented within this app, comparing it to similar types of information, it goes on to compare this system to those of its competitors, explaining why this cross platform mobile application is the most useful way to provide access to this data.

2.2 Clinical Knowledge Summaries

2.2.1 What are they

The clinical content discussed in this dissertation are Clinical Knowledge Summaries, currently authored by a group of GP's (now ex or part time GP's) at Clarity Informatics as part of their PRODIGY product. These **Clinical Knowledge Summaries (CKS) are evidence based clinical guidelines** for different healthcare topics. Due to the data available, evidence based systems are a gift of the internet age as Eysenbach G and Jadad AR (2001) explain, and are key to the future of healthcare guidance. Each guideline is essentially a report on the healthcare topic based on existing studies (evidence) in the area. It is formatted in a consistent, readable manner and kept regularly updated using processes and guidelines accredited by the NHS (National Health Service). The CKS topics are particularly useful for GP's to quickly get information on a condition, diagnosing sick patients and checking up on the recommended prescription info. It is designed for use primarily by GPs to assist with their diagnosis and prescription of patients however it is also used by GP registrars, nurses and pharmacists in primary care in their day to day work, and the content is available to anybody through the companies website [1].

Each CKS topic contains a number of clearly defined sections including 'Background information', 'Diagnosis', 'Management', 'Prescribing Info' and 'Evidence'. Having the topic so clearly structured is key as to why this is a great resource for professionals to reference in a day to day environment, allowing them to select a topic and quickly extract the information they are looking for.

2.2.2 Where did the content come from?

The large amount of clinical content produced by Clarity did not just spring out of the abyss, it has been researched, developed and improved upon by clinical authors since the idea of PRODIGY was first dreamt up by I.N Purves as a method of providing decision support systems in GP prescribing (I.N Purves, 1997). In its first stage PRODIGY was to investigate the use of computerised guidelines in general practice to assist but not override the GP's clinical ability to prescribe, such a system had been considered highly desirable before Purves began his work.

Purves received funding from the NHS Executive Primary care branch and began to develop the concept of this system, initially giving the GPs decision support after a diagnosis had been made, with clinical advice and therapeutic recommendations. The system would give GPs choices of drug treatments as well as non-drug treatments and information leaflets. Once a choice of drug had been selected, the system would provide information on the appropriate dosage, accounting for allergies and anomalies and finally printing off the prescription. The beauty of Purves tool is that it would give the GP's crucial information without the imposition of its use, where the GP still retains the power to edit any drug recommendations they receive.

To give PRODIGY credibility as a concept, it would need to be tested, and so the development of 250 initial prescribing recommendations alongside drug choices for each were developed by a small group. This information was sourced from the liked of the British National Formulary, the Drug and therapeutics Bulletin and existing guidelines, and was then validated by an expert group nominated by the Royal College of GPs to ensure professional credibility. The second requirement to prove its credibility was to develop software which could be integrated into existing UK computer systems. Five software companies agreed to collaborate to develop packages which could be deployed into systems used commonly by GPs.

Both the guidelines and and software itself received constant feedback from the GPs in order to progressively develop. The GPs who took part in Purves experimental trial not only believe that PRODIGY is a concept worth developing but also that it will be a vital part of the jigsaw in the future of general practice. Stating that the natural GP community has a high level of requirement for this system, which is only backed up by the stat that 87% of GP's asked said that they would like a set of high quality prescribing guidelines (I.N Purves, 1997, Table 39), and 78% of GP's thought that clinical computer systems would enhance their ability to deliver best practice to their patients (I.N Purves, 1997, Table 39).

Following on from Purves research, the guidelines for each topic advanced significantly beyond diagnosis and prescription information with the foundation of the healthcare informatics company Clarity Informatics in Newcastle.

2.2.3 How is it accessed?

During the first phase of PRODIGY and in the years after its initial creation, the guidelines were accessed using the bespoke software created to integrate with existing GPs machines. GP's found themselves accessing this through a desktop application. Following this, and after the foundation of the company Clarity Informatics, the content produced for PRODIGY was compiled into a book entitled 'Prodigy Knowledge: Practical, Reliable, Evidence-based Guidance [19]. The development of a book meant that GPs didn't need to worry about integrating PRODIGY into their system, whilst also allowing them to view the content on the go!

More recently the content has been made available on the internet and can be accessed by GPs using the NHS website [17] or through Clarity's own site. Having access to this content through a simple web browser on the internet makes diagnosis and prescription information accessible to the large proportion of primary healthcare professionals, however just as we have seen before with the transition from desktop application to book, we are now witnessing a requirement for this sort of data to be made available on the most commonly used of devices now - mobile devices.

2.3 Similar Systems Research

2.3.1 Other accesses to similar information

Primary healthcare professionals have a large array of sources from which to obtain information and learn about different conditions. In the case of our primary user base, GPs, who wish to make correct diagnosis and provide their patients with suitable prescriptions, they can consult for example the World Health Organisations (WHO) International classification of Diseases (ICD-10) [13]. ICD-10 is a resource that can be used to classify diseases and other health problems recorded on many types of health and vital records, and although it is namely a resource for diagnosis related classifications, they do provide assistance to primary healthcare professionals to conduct the diagnosis in the form of related information. On a slightly different note, FHIR (Fast Healthcare Interoperability Resources) is a product produced by Health Level 7 International [14] which aims to achieve interoperability within the primary healthcare field with the goal of tracking epidemics and prescription drug fraud amongst others. This resource has the potential to help GP's in their day to day work, although as Bender and K. Sartipi (2013) discuss, there are still ambiguities on how the HL7 FHIR standard can utilise its strengths and become a valuable resource.

There are a number of existing applications which offer access to similar useful information which can be utilised by primary healthcare professionals - an application named Consult is a mobile application developed by Univadis offers users information on symptoms, diagnosis and treatment for over 2000 diseases and disorders [12]. This application is robust and contains a lot of useful functionality for its users with drug information, the ability to search for equivalents to name branded drugs alongside their lab tests which automatically updates to keep the users data current. The information they provide under 'diseases' is comprehensive but not up to the standard of that produced in PRODIGY, the app provides overviews of the diseases but is not evidence based and does not appear to have the same level of research or structure applied, and the authors have gone for quantity of information over quality.

Medscape provide another similar application, with evidence based treatment information for diseases and conditions [16]. Their clinical information is of good quality however it again is more of an education resource for students and may not fare well in a time constrained situation in which healthcare professionals need direct access to information.

The CKS topics, themselves produced for PRODIGY are a free resource although are sold to the NHS for use on their website [17], and are available through there for free. However this site is obviously not streamlined for mobile, offline or persisted use.

2.3.2 Why aren't they viable

Whilst a number of these similar systems contain a vast amount of information like Medscape, or a diverse set of functionality like in the Consult application, the layout and quality of the content is not up to the standard of PRODIGY. We can see in Medscape that they have produced a valuable resource for those looking to learn in depth on healthcare topics and in the position of a medical student this would possibly be a more suitable app, however for GP's or primary healthcare professionals to quickly search for the information they need to make informed diagnosis or prescriptions it is not ideal. The CKS topics are structured well and backed up with evidence making them a key resource for referencing quickly.

If we look at the NICE website [17] which houses the CKS topics, we can see that they are a vastly popular resource with 200000 independent visits in August 2015 [Appendix Figure 1] making it the most popular part of the NICE website by a factor of 20. This is great, and obviously speaks bounds for the quality of the content, however this resource is limited in its functionality. For users to truly make the most of this content, specifically accounting for intermittent networks and user experience, it must be served in a better way. A portable application capable of saving, bookmarking and caching content makes this content more functional, and removes a number of its barriers to access.

2.3 Why develop this app?

2.3.1 Reasoning

Currently the massive amount of useful clinical information developed for Prodigy is only formally available via a web browser, as A. Lipsman and A. Lella show in their research for comScore [18] the desktop browser is fast becoming an outdated digital platform with the number of mobile phone users surpassing the number of desktop users globally in 2014 and continuing to expand the gap. It is therefore timely for this resource to be handed over and presented from a mobile application, this way not only can Clarity reach a wider target audience but they can also comply with the modern times. Should they leave it later, say another 5 years time, we will see a large wasted audience and possibly the arrival and establishment of further competitors in the market.

Another valuable reason to create this application is to provide access to this resource for those users who require offline support and cannot simply navigate through their browsers to gain access each time they wish to reference the information, for example due to reliance on intermittent networks in rural or un-developed regions. There are a number of use cases in which functional networks cannot be assumed like in the presence of a 'no-zone' - a highly sensitive zone in hospitals or other care units which prohibit the use of electronic technologies, in cases like this where the users cannot access a live website it is essential that we provide an alternative access to this information.

Aside from the main benefits highlighted here, there are a number of other positives to creating an app and gaining an active user base. With it Clarity can further develop brand recognition, they have the ability to collect usage data which can be used to further enhance the app and tailor it to the user bases needs and it is also possible to interact with the user base through notifications or prompts tailored to the individual something which can not be done through the current platform.

2.3.2 How does the clinical data influence the finished product

It is of critical importance that the design of the final application is centred around an uncluttered, fast, lightweight access to the core clinical information. The information itself influences the product in its entirety, from the opening of the app to its closing the main focus should be on access to the core information. The CKS topics themselves (of which there are ~330) are roughly 900 kb each. The topics and their siblings receive regular updates from the in-house authors, thus it is necessary for the app to keep up to date with the current versions and present only the latest material.

The size of this collection of information is of influence to the final design as it causes the prospective user base some issues. When providing the access to those users with intermittent networks, there are a number of potential solutions - either all the data can be pre-downloaded, and the user can receive regular updates on the data or we can implement a bespoke caching and storage system coordinated with an API to allow the users to prepare for the times in which they will have no internet. This is obviously a large amount of data and should it be pre-downloaded the app size will swell up dramatically, potentially causing the loss of some users.

2.4 Summary

The high quality evidence based guidelines and knowledge summaries provided in Prodigy are a fantastic resource for clinicians, its well designed structure contributes to its ease of use, helping users quickly retrieve the information they need. Its popularity is proven by its impressive usage statistics. There are other sources to similar information but none compare, either lacking in the quality or structure of their information or in the user experience.

The content in Prodigy was first developed in the form of a desktop application due to massive demand from GPs and primary healthcare professionals, it then progressed into a book and finally took its current form as presented in a website. However due to the emergence of mobile technology and its massive growth in popularity it is timely that this information be served from mobile devices.

With this change into a mobile app it is now possible for us to overcome the barriers to access of the information and provide tailored offline support for those users who cannot rely on their intermittent network connection to allow them to view the info at their discretion. It is key that we support these users and investigate different methods of serving this data to help them make informed healthcare decisions, particularly in rural or underprivileged areas.

3. System design and implementation

3.1 Chapter Introduction

This chapter details the research, design and development of the application, beginning with a walkthrough of the development model and each phase of development. It then continues with an overview of the systems overall architecture, detailing the development platforms, linked databases, API's and other aspects affecting the overall design.

We discuss where the requests of functionality came from, how they can be used to help satisfy the aims and objectives of the project, before giving detail on each notable feature of the final design. A feature based approach has been employed to show the steps taken in design, in which the feature is discussed, the background research shown including a technology comparison, how the feature was designed and finally the details of its implementation. A feature based approach is more logical than a modular structure in which all the research is done, then the design, then the implementation, as the agile methodologies of development meant the implementation was performed this way.

The chapter finishes with a summary of the outcomes from the design, providing an analysis on how the implementation went as a whole.

3.2 Development model

This section details the methodologies chosen for development and the projects overall lifecycle including the phases which are undertaken throughout the apps development.

3.2.1 Methodology

The development of this application employs the agile methodologies, which is a practice using iterative, incremental development. This methodology promotes adaptive planning and continuous improvements on the system accounting for feedback from user involvement. With agile it is common for projects to iterate through their requirements list, fulfilling each one before moving on to the next.

There are a number of benefits to using agile, especially for this project. For one it will allow the project to integrate testing throughout its lifecycle, enabling regular inspection of the working product to get visibility on bugs and problems with the system. It will also enable us to end up with the right product at the end of development, allowing development requirements to emerge and evolve as the implementation is underway through testing or user interaction. J Ferreira et al. [3] describe how agile iterative development can work in sync with UI design to mitigate bugs and produce a much more tailored UI for the user, this is important as the UX is important for this project.

3.2.2 Development lifecycle

Throughout this projects lifecycle it will pass through a number of 'phases' or stages, shown in Figure 2 below. As it can be seen below, the development process begins with the planning of system architecture, diagnosing the requirements in connecting the application to a hosted API, database and researching the possible platforms which can support this in a cross platform application. Following this, the rapid prototyping of an application capable of supporting data retrieval via an API will commence. Based on the research conducted by Gordon and Bieman [4] it would appear that this method of rapid prototyping is a particularly effective method of initial development provided that the purpose and scope of the prototype are carefully designed, in this

case we want to get the application in a suitable position from which to conduct the next phase - ‘Consideration of use cases and their functional requirements’.

This is a key phase - the design phase essentially - in which each potential user case will be considered (for example a doctor going to visit a patient in an area with no internet, who has complained of a number of symptoms) and the functional requirements of this use case will be identified. This phase employs the agile methodology discussed previously, in which a feature is picked, designed, implemented, tested and if it does not meet the criteria to satisfy the use case it will be re-designed until it does.

After all use cases are considered satisfied the final touches will be made to the user interface - it is probable that with the incremental additions of features that the UI will be in an unfit state for deployment thus making this phase entirely necessary. Finally end to end testing will be conducted, testing the full workflow of the application in a realistic end user scenario to ensure it works as desired.

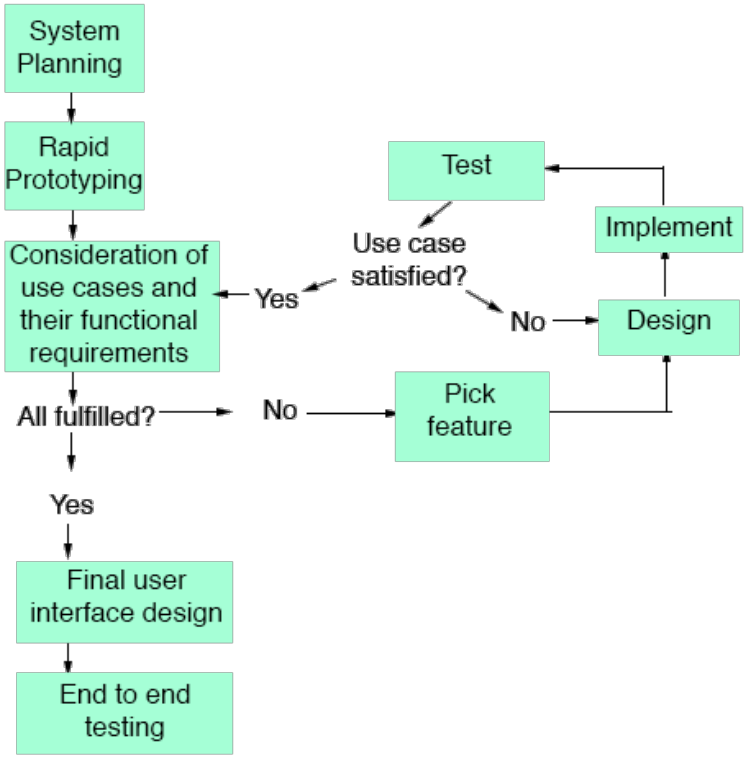


Figure 2. Development phases

3.3 System overview

3.3.1 Achieving cross compatibility

Achieving cross platform compatibility to satisfy the projected extensive user base can be accomplished in a number of ways, with native development and hybrid development being the main routes. Building natively means using the native language for the device, Java for Android and Objective-C for iOS, development which will require the code to be largely re-written for each platform, which is a tricky time consuming process. When apps are written natively their code is compiled into machine code, giving the best possible performance from the mobile phone.

A hybrid build is essentially a web application operating in the phones native browsers, i.e. UIWebView (iOS) and WebView (Android). The apps are built with HTML, CSS and Javascript and converted to native apps using one of a number of platforms like Cordova [20], Ionic [21], Phonegap [22], Xamarin platform [23] or Appcelerator Titanium [24], each with their own quirks, pros and cons. Hybrid development is faster, more simple and faster in comparison to native, and can generally support a lot of useful plugins that are cross compatible. The negative aspect to hybrid apps is that they rely on the phones browser, which means they cant compete with the speed of their native counterparts. Despite this fact, Armgren M. [25] takes a look at the performance of Xamarin Platform and finds that it is the computational tasks which are slower in comparison, and that most user interface and network components work at native speed.

Due to this project not using copious amounts of computation power, and requiring fast development it is therefore logical that a hybrid platform is used. Whilst there are a number of eligible choices for the platform, Ionic suits this project best due to its ability to allow for rapid prototyping, it builds apps with AngularJs a very powerful javascript library, is open source and has a large community and set of documentation behind it. Ionic builds on top of Cordova which takes care of packaging the HTML5 app as a native app that can run in Android, iOS, and other platforms.

As this is a web application underneath, Visual Studio can be used as a structured code editor software, providing useful tools such as ReSharper to help speed up development.

3.3.2 Architecture

There are a number of communication lines between the application and external sources, for example APIs from which to retrieve information and databases in which to store it, these amongst others are outlined in Figure 3 below.

With Ionic building with HTML, CSS and AngularJs we have a web application which connects using HTTP to external APIs and uses AngularJs to access internal databases and handle messages received from external sources like GCM (Google cloud messaging) or APNS (Apple Push Notification Service).

The CKS topics are hosted in a database with Azure web services [25] by Clarity, in order to access the data a new API will need to be created and attached to the Azure project, it will be created in .NET with C# and access the SQL database from there. The API in its first iteration is very basic - merely returning the names and UID's of each topic, or the full HTML content of each.

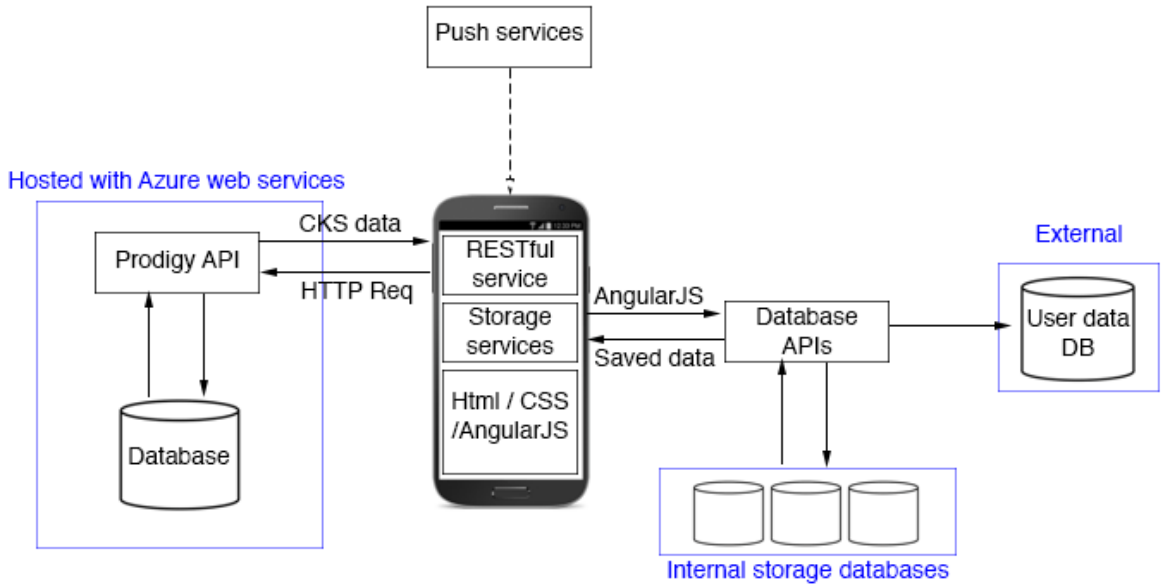


Figure 3

Utilising the existing infrastructure at Clarity is useful and allows the project to achieve rapid prototyping. Azure allows for quick changes, this API and the data served can be updated as required by the features of the app.

Internal storage will be used for the additional features which require data persistence, for example storing of UX details or CKS data.

3.4 Research, Design and Implementation of features

3.4.1 Design Introduction

This section details the additional features appearing in the system, showing the reasoning behind each feature, which technology was used to implement it and why, its design and finally implementation. The features are presented in the order they were considered for the application. It goes into detail on the features implementation and shows the reasoning behind the methodology. While each feature is thoroughly tested as part of the agile development process - the formal testing occurs in section 4. Think of this feature based approach as taking a vertical slice through all layers of the system for the design and implementation of each feature, it shows the effects on the UI, underlaying logic and data access as depicted in Figure 4.

In order to get the application in a position from which is was possible to design and test features, rapid prototyping occurred setting up a basic application with access to the data provided in Clarity's Prodigy API.

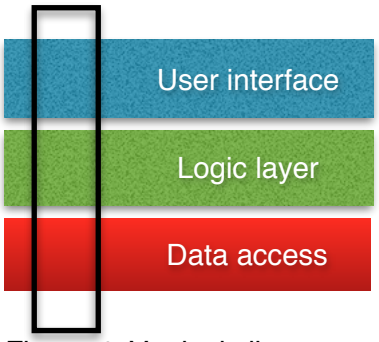


Figure 4. Vertical slice

3.4.2 Caching of downloaded data

Description

Presently each time the user wishes to view a topic they send a get request to Clarity's API to pull the whole topic for display. This is a drain on the users resources and requires an active internet connection, however it is possible to save these topics into the users application memory temporarily, meaning they can avoid sending repeated get requests.

The purpose of this feature is to lower the need for an active internet connection by investigating persistence strategies and implementing functionality with the capability of caching topics viewed by the user without explicit instruction, with the intentions of reducing number of requests to the server, and the size of the responses. This must be done without inflating the applications physical size too large for the average user.

Technology comparison

As data persistence is a common thing to do in mobile applications, there are a number of different technologies available for use, each with their own unique attributes and use cases. Even

though this application isn't technically built natively there are still viable options to choose from [8], a number of them are outlined in Figure 5 below. All storage options listed in this table are embedded in the device, and either run through the phones browser or access the phones storage directly.

	Storage capacity	Storage Format	Speed	Platforms Supported	Ease of use	Persistent / Temporary
Internal (Local) Storage [30]	5mb (browser limit)	key-value (strings)	Fast	Android, iOS, Windows, BB	Easy	Temporary
LocalForage (with SQLite) [27]	Unlimited	key-value (stores objects)	Medium	Android, iOS	Medium	Persistent
IndexedDB [29]	5mb (browser limit)	key-value (stores objects)	Fast	Blackberry, Windows	Easy	Temporary
SQLite Database [26]	Unlimited	structured tables	Fast	Android, iOS via plugin	Medium	Persistent
Web SQL	5mb	structured tables	Fast	Android	Medium	Persistent
Cordova File Caching [28]	Unlimited	Files	Medium	Android, iOS	Medium	Persistent
Figure 5						

After review of the table, there are a number of plausible options for this caching system, firstly the local storage option which comes built in with Ionic with a 5mb limit (meaning we could store up to 5 topics), is easy to use, has a nice API and stores strings as data which is all that is required, however it is temporary and thus will not allow users to cache data over multiple uses of the app.

SQLite stands out as the best technology overall, with an unlimited storage size and persisting of data in the phones memory. It allows users to create and execute any SQL queries meaning it is possible to deploy a truly personalised structured database. SQLite is one of the most widely used database plugins for Ionic/Cordova applications and thus has a large amount of user build documentation and support. Wankhade, N.V. and Deshpande, S.P. [7] analyse the requirement for embedded databases in mobile devices and similarly conclude that SQLite is the optimum technology for applications in which data synchronisation and complex engines are not a concern, and notes that it restores automatically after system collapse or loss of power.

Design

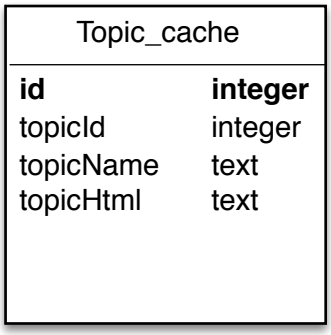


Figure 6

The schema design for the 'Topic_cache' table is shown in Figure 6, containing all the necessary columns to display a standard topic. The content for each is contained in the topicId, topicName and topicHtml fields, with the id field acting as the primary key for each row.

Implementation

The installation of the SQLite Cordova plugin [26] is required for the SQLite to be accessible in this application, this can be done via the CLI and with a few file includes in code.

```
cordova plugin add cordova-sqlite-storage
```

The SQLite database is accessed via an angularJs service (acting as a database API) which contains functionality to execute sql and retrieve/update the table as necessary, as shown in Figure 7 below, referencing code in the appendix.

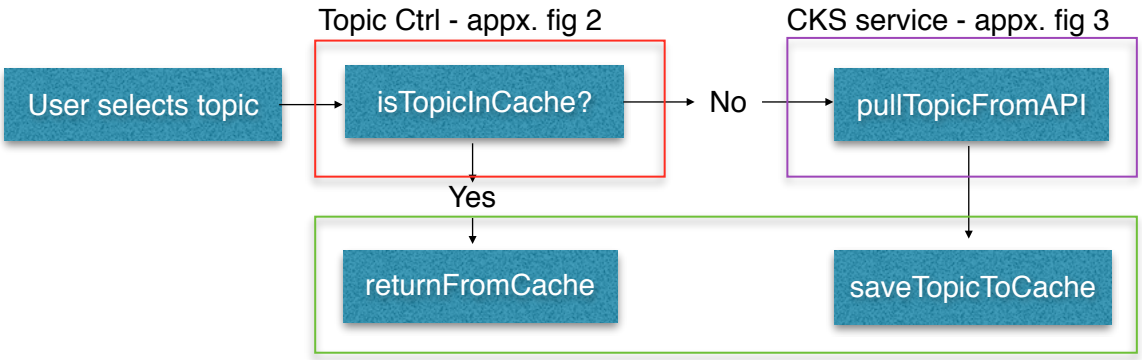


Figure 7 Cache service - appx. figs 4 & 5

This implementation now reduces the payload for opening duplicate topics, and reduces the need for an active internet connection as the users are not required to call the CKS API each time they want to access the information. There is however a problem now - with extended use in which the user will open a large number of topics, the app size will gradually increase beyond an acceptable limit. If the user sees a large cache they may attempt to clear it and thus make the caching service obsolete.

3.4.3 Optimising Cache Functionality

Description

The cache functionality in its current state presents a number of problems. First off, as the data in this cache is saved to the users phone with no upper bound on the size, it is possible that we will see a ballooning in the size of cached data beyond an acceptable limit. This is not ideal as it means users could run out of phone memory and either remove the app or clear its cache. Second, both the response from the CKS API and the row size in the topic cache are very large due to only full topics being available to retrieve, contributing to slow data retrieval. It would be possible to both reduce the requests made by the user and optimise the performance of the cache by:

1. Editing the CKS API in Azure to split up topics into pre-defined sections

2. Limiting the number of rows to be stored in the cache

Technology Comparison

The cache is already designed in SQLite in which it is not possible to set an explicit cap on the number of rows in the database. It is however possible to use SQL tricks to create a virtual cap for the rows, the two possible methods are as follows:

1. Each time a row is to be added to the table, run an SQL query to count the number of current rows in the table, if there is room then add the row, if not then overwrite the oldest row in the table
2. Pre-populate the cache with a number of empty records and overwrite the oldest one on each insert based on the timeStamp

Design

With the ability to restructure the data produced in the CKS API it is possible to split the topics up into different sections - Figure 8 shows how this split groups nodes within the current hierarchy into sections, based on advice from the Prodigy manager in Clarity. In order to accommodate a virtual cap on the number of rows in the cache, the schema will need to be updated to include a time stamp from when the record was inserted, this schema design can be seen in Figure 9.

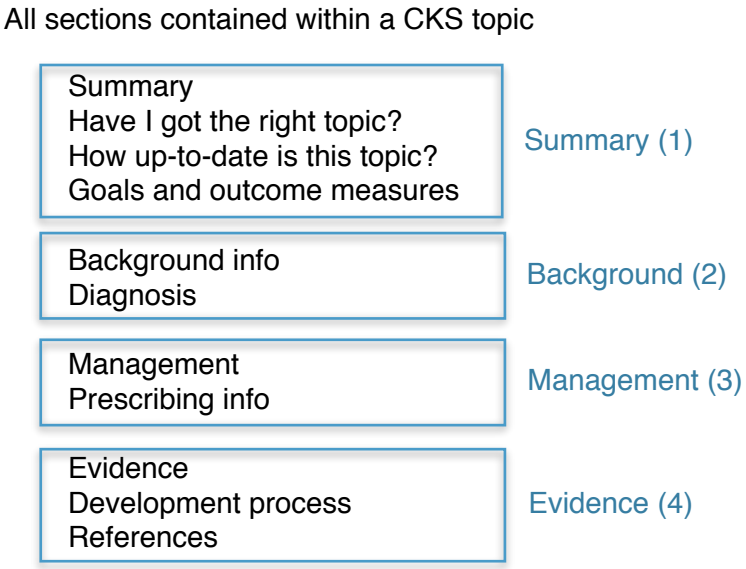


Figure 8

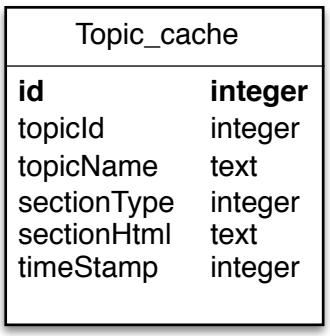


Figure 9

Implementation

The API used to retrieve CKS topics is updated (appx. Figure 6) and now returns sections of topics as opposed to the full block of html as was previously returned. The sections come in the form of an enum the same on the client and the API server side. This reduces the response massively and allows sections to be explicitly stored in the cache.

The topic cache has seen major transformations, with a virtual row limit of 7 implemented. When the database is opened for the first time, a number of blank rows are inserted and given different timestamps (appx. Figure 7) to pre-populate the table. As new rows are required to be

inserted into the cache, the oldest data in the table is overwritten based on its timestamp (appx. Figure 8), allowing us to limit the amount of data stored in the cache.

3.4.4 Saving specific topics

Description

With caching set up to provide users with a faster and more elegant UX, and the topic data being saved behind the scenes, it is logical that users should be given the option to explicitly store content on their devices with the intentions to view it at a later date. Existing Prodigy users at Clarity have requested the ability to save full topics alongside a note - for example 'Check this for peter'. Explicit persistence gives users confidence that their data has saved effectively and is available offline. This feature is suitable for the scenario in which a user may be off to visit a patient in a remote location in which internet is not accessible, but they know roughly what is wrong with the patients they are visiting.

Technology Comparison

Figure 5 shows the investigation carried out into potential storage options for this system. Based on the results from the table, SQLite is the most suitable option due to its structured storage format and >5mb size limit.

Design

Topic_store	
id	**integer**
topicId	integer
topicName	text
sectionType	integer
sectionHtml	text
userComment	text

Figure 10

Figure 10 shows the design for Topic_Store table in the SQLite database, similar to the cache table (fig 9) however with a comment column added on in which the users string is stored. Ionic offers functionality like popups and confirm boxes which we can use to allow users to add a custom message and receive confirmation of their save. The conceptual design of the feature is shown in Figure 11.

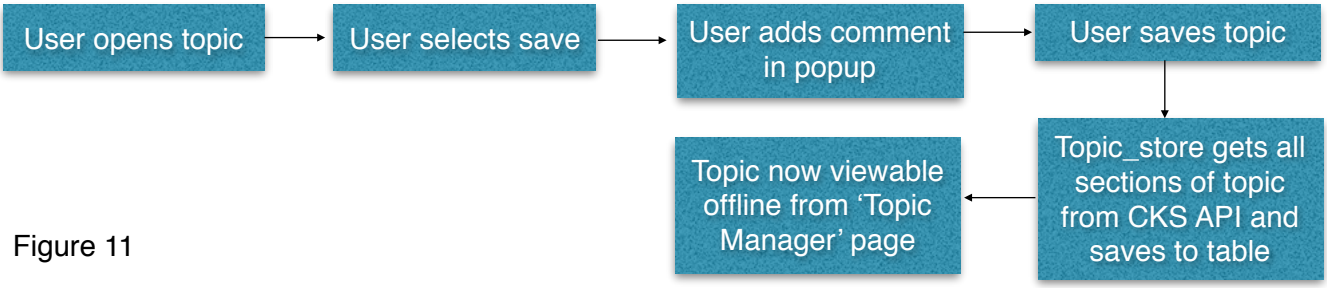


Figure 11

Implementation

The implementation of this feature required the editing of the main side menu in the app to include the new ‘Save topic' button. When a topic is being viewed by the user the side menu now has its buttons replaced with view-specific options. This is achieved by broadcasting the applications state over the root scope (angularJs $rootScope) of the application (appx fig 9).

The implementation of the topic_store API is shown in appx. fig 10, with the full implementation of save feature shown in appx. fig 11.

The application always looks in the store before downloading a topic so that it can save on API calls (appx. fig 12), this is handled in the topic controller, however the store is always checked for the data before requests are made elsewhere.

The view of the ‘Topic Manager’ containing all the users stored topics is shown in Figure 11.5 below.

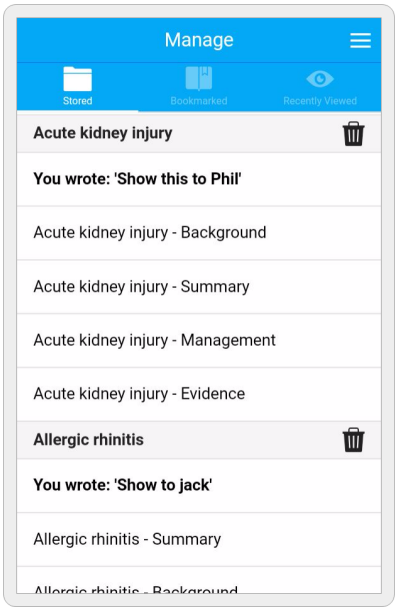


Figure 11.5

3.4.5 Bookmarking

Description

As opposed to saving and persisting full topics - users wish to insert a bookmark to make a note to read something later or reference it when they are with a patient. This is an important piece of functionality and will let users navigate to their intended resource faster. A problem is that the large CKS topics are currently split up into 4 sections, which are much too large to bookmark a specific part of, so structural changes (adding sub-sections) to the topic data and an update to the API are required.

Technology Comparison

With reference to figure 5 it is of note that due to the low amount of memory taken up by a reference to a bookmark, SQLite may not be the most optimal way to store them. LocalForage on top of IndexedDB may provide the structure store required. However setting up and installing a new database engine merely to store bookmarks is not optimal, and so using the existing functionality provided by SQL to generate a small structured table is the most functional option.

Design

In order to accommodate a more accurate bookmark, the retrieved HTML from the CKS API will need to be re-designed on the server side. Instead of returning one full block of html per topic section, the section can be split up and sub-divided into more useful parts, which can contain navigation properties (an integer value). This does not require any changes to the external API or internal database services as the string is JSON.

Bookmark_service	
id	**integer**
topicId	integer
topicName	text
sectionType	integer
subSection	integer

Figure 12

Figure 12 shows the schema for the bookmark service which will store details of each bookmark created by the user. Much like the implementation of the save functionality from 3.4.4, the bookmarks will reside in the 'Topic Manager' page.

Implementation

Much like 3.4.4, bookmarks are initiated from the side menu (appx. figure 11) using a rootScope broadcast (appx. figure 9). Figures 13 and 14 below show a bookmark being created in the UI. When a bookmark is opened, a state parameter is added to the page which gives the positional information of the element to the angular controller. The controller in angularJs loads the specified section, and uses the '$ionicScrollDelegate' in order to scroll the view to the correct position (Fig 15).

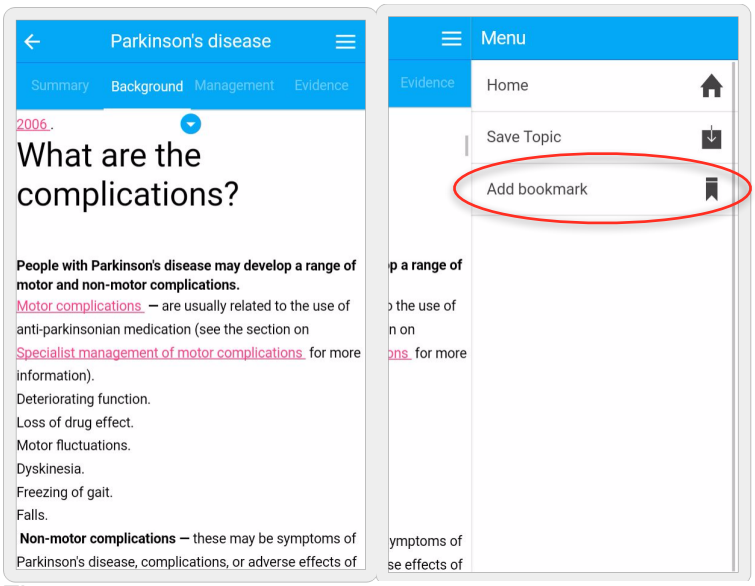


Figure 13

‘Topic Controller’

```
$scope.$on("bookmark", function () {
    var scrollPos = $('#section-content').offset().top - 100;
    bookmarkService.isTopicBookmarked($scope.model.topicId, $scope.model.sectionType, scrollPos).then(function (success) {
        if (success === true) {
            $ionicPopup.alert({ title: "Success", template: 'Bookmark added successfully' });
        } else if (success === false) {

            bookmarkService.bookmarkTopic($scope.model.topicId, $scope.model.topicName, $scope.model.sectionType, scrollPos)
                .then(function (success) {
                    $ionicPopup.alert({
                        title: "Success", template: 'Bookmark added successfully'
                    });
                }, function (error) {
                    $ionicPopup.alert({
                        title: "Bookmark error", template: 'Something went wrong, please try again'

                    });
                });
        }
    }, function (error) { });
});
```

‘Bookmark Service’

```
service.bookmarkTopic = function (topicId, topicName, sectionType, scrollPos) {
    return $q(function (resolve, reject) {
        service.db.executeSql("INSERT INTO bookmark_store (topicId, topicName, sectionType, scrollPos) " +
            "VALUES (?, ?, ?, ?);", [topicId, topicName, sectionType, scrollPos], function (res) {
                resolve(true);
            }, function (error) {
                console.log('INSERT error: ' + error.message);
                reject(false);
            });
    });
};
```

Figure 14

```
<a ng-show="model.listType == 'Bookmarked'"
    ui-sref="root.topicSection({topicId: topic.topicId, topicName: topic.topicName, sectionType: topic.sectionType, scrollPos: topic.scrollPos})">
    {{getSectionName(topic)}}
</a>
```

```
$scope.scrollToScrollPos = function () {
    $timeout(function () {
        $ionicScrollDelegate.scrollTo(0, -$stateParams.scrollPos, true);
        $stateParams.scrollPos = 0;
    }, 750);
}
```

Figure 15

3.4.6 Push Notifications

Description

Communicating with users is useful as it means they can receive useful bytes of knowledge as seen fit, updating them with release info and prompting them to take action in their app. Push notifications can be used to engage users, and contact them without them being required to divulge their email address, they are less messy and quicker to deal with and manage. Push

notifications see a very high response rate and are considered to be the best form of user engagement.

In this first iteration of push functionality, it is required that the app be configured to receive push notifications with the intentions of informing the users that a new release of topics they have stored or bookmarked have been updated and that they should open the app to update to the most recent version.

Technology Comparison

As push functionality is widely used, there are a number of possible technologies for use within a Cordova/Ionic application. Plugins like 'Phonegap plugin push' [31] or 'PushSwoosh' [32] offer multi platform push functionality with similar implementation. The Ionic documents recommend using the official plugin 'Phonegap plugin push' and detail that combining the use of the plugin with native Ionic code will allow users details to be persisted on the applications dashboard online, meaning mass push notifications can be sent out with the click of a button through a sleek UI.

Using Ionics recommended plugin allows push messages to be sent from a server, CLI or from the Ionic dashboard, which then routes through the Push API, and either GCM or APNS depending on the phones OS, as seen in Figure 16 below.

Figure 16

Design

The desired functionality here can be achieved by implementing the basic notification system with the official push plugin, the design of which is already highlighted in Figure 16. The appearance of the notification is not highly customisable, using the applications icon as a picture and personalised title and message body for each notification. For this iteration the users details will be saved to the Ionic dashboard and the notifications can be designed and sent from there.

Implementation

In order to receive notifications, the app must be hooked up to the Ionic dashboard with the necessary plugins installed, which can be achieved from the code below.

```
ionic add ionic-platform-web-client
ionic plugin add phonegap-plugin-push --variable SENDER_ID="GCM_PROJECT_NUMBER"
ionic io init
```

This requires registration with GCM in order for Android phones to have their notifications sent. In addition to the plugin being added and the relevant registration on Ionics Dashboard, code must be added to the system in order to save the users 'push token' which is a unique identifier for

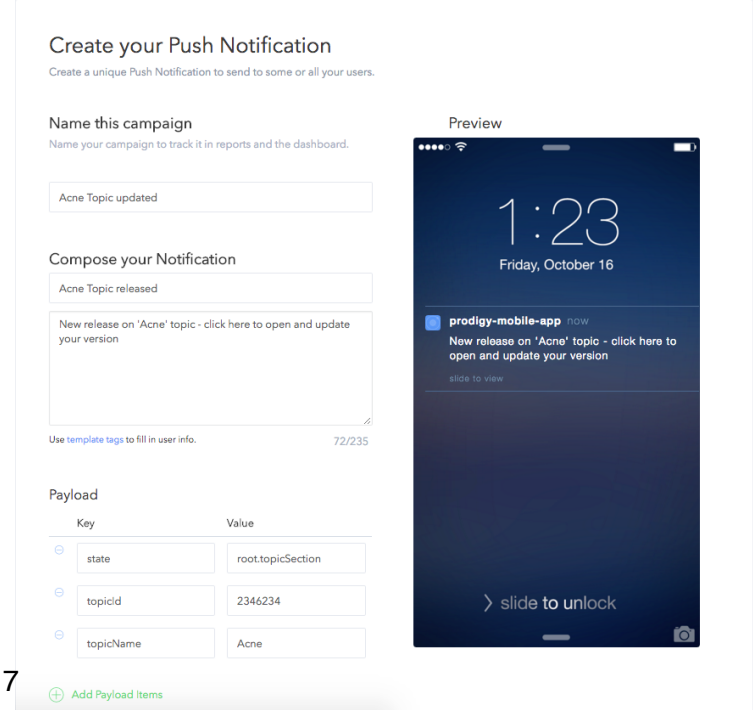


Figure 17

their device from which to receive notifications (see appx. figure 13). After the registration of the users, their details are stored in the Ionic dashboard, from there the notifications can be created and sent out, with custom parameters (Figure 17).

This notification is received by a pre-defined notification handler which allows the app to handle the push and display further information to the user once they have selected it (appx. figure 14), letting them navigate to the topic or update their saved version.

After UI changes were made to include the apps icon and lay out the notification it is implemented and received as in Figure 18. Push is now working as a proof of concept to present useful information to all users.

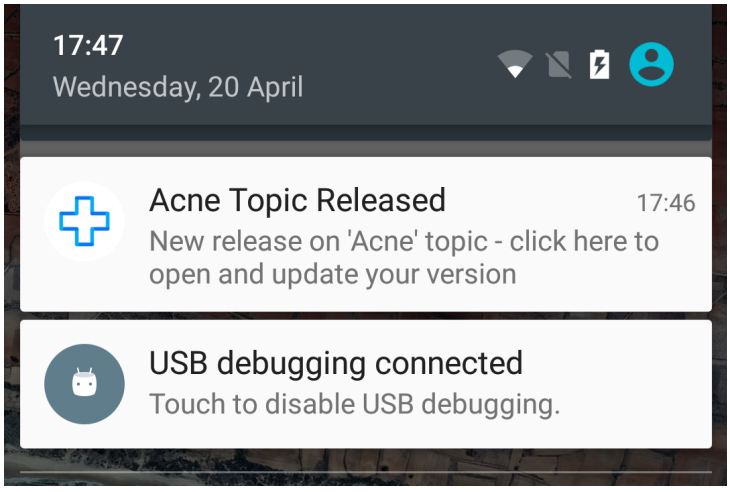


Figure 18

Issues in Implementation

During implementation Ionic deprecated its plugin and user registration which was used in combination with v1, causing notification system to switch to Ionic v2. Documentation for the implementation of notifications was not expansive and missed a number of key concepts, including the issues when sending sample pushes via the CLI, in which the APIs private key required it to be base64's in order to preserve its privacy.

3.4.7 Location based push functionality

Description

With the implementation of basic push functionality to communicate with the users complete there are a number of opportunities for further features. One such feature arises as a solution to objective 4 of the project, in which functionality to account for epidemics and diseases is desired. Should there be an outbreak of say 'Super gonorrhoea', it is desirable that users within a certain distance from the outbreak should be prompted to view or download the topic. The goal is to minimise the number of times users are caught out in a position where they need to reference a topic and cant due to their network capability.

Chosen technologies

We can make use of the existing infrastructure of the notification system, in which users app's are registered with a push token upon opening and are able to receive notifications. In addition to this, a database in which to store the users tokens and their location is required, alongside a method of querying the database to find users in a certain radius from the outbreak and consequently sending them notifications via POST requests.

External Database - Leverage existing infrastructure of Clarity's SQL DB hosted with Azure
Location Service - Cordova official geoLocation plugin [33]
Database Querying - SQL
POST requests - Javascript and Postman RESTful client

Design

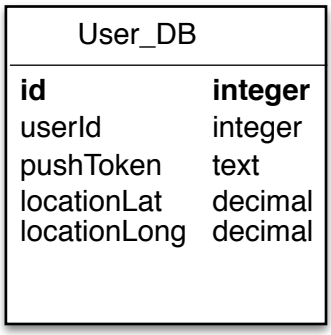


Figure 19

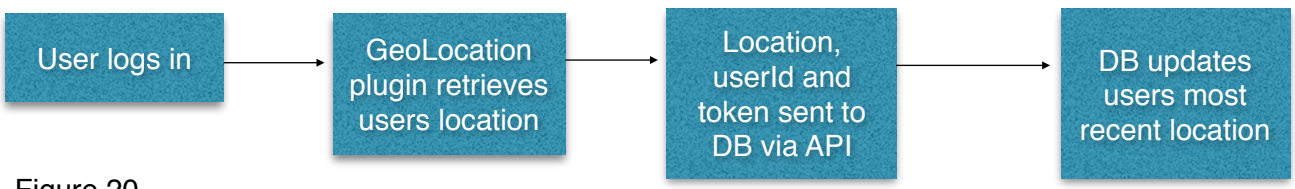


Figure 20

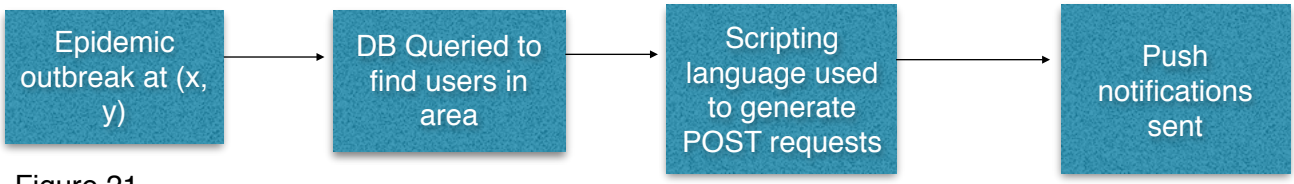


Figure 21

Figure 19 shows the schema for the proposed table in which to store users data and recent location, and Figure 20 shows the workflow for persisting the users information into this table.

Figure 21 shows how the database will be queried in order to find all users residing in the radius of an outbreak. This is a high level view of how the process will work, and requires a large chunk of implementation.

Initially the information for epidemics will be handled manually, for lack of a reliable source of information and to help mitigate the development overhead. The notification itself (when opened) will prompt the user to open and save the topic in question, in oder for them to best prepare.

Implementation

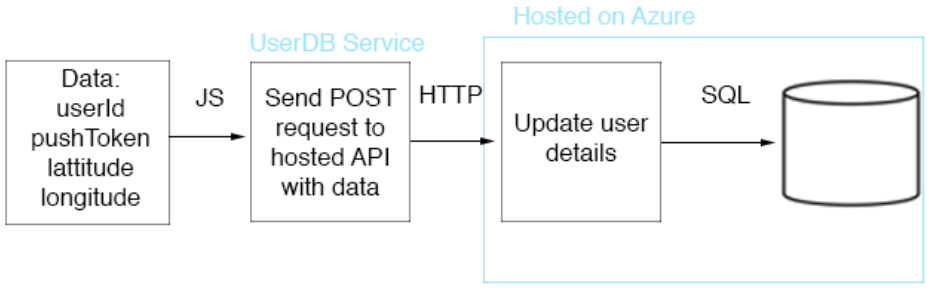


Figure 22

In order to retrieve the users location and save it efficiently, each user will need to be assigned a unique identifier, using Ionic Authentication and the injection of randomness we can achieve this and persist it to their device, as shown in appx figure 15. Once the user is authenticated, their push token is set and Cordovas geolocation plugin is used to retrieve the latitude and longitude of the device (appx figure 16). As Upkar Vashney explores in his paper on location management in mobile applications, functionality like location based services require a lot of overhead as they execute more transactions [6], in an attempt to keep the overhead of this app down and maintain functionality, updating location on login is sufficient.

Once the details are compiled, the API created for the database is called and sends the data to the hosted database to store it, as shown in figure 22.

When an epidemic occurs, coordinates are given to mark its location, and the database is queried with a suitable radius (appx figure 17). This list of users is generated by a GET request to the API and all of the users tokens can be then put into a POST request like in figure 23, and is then sent out, initiating the push notification request to all users close to the outbreak.

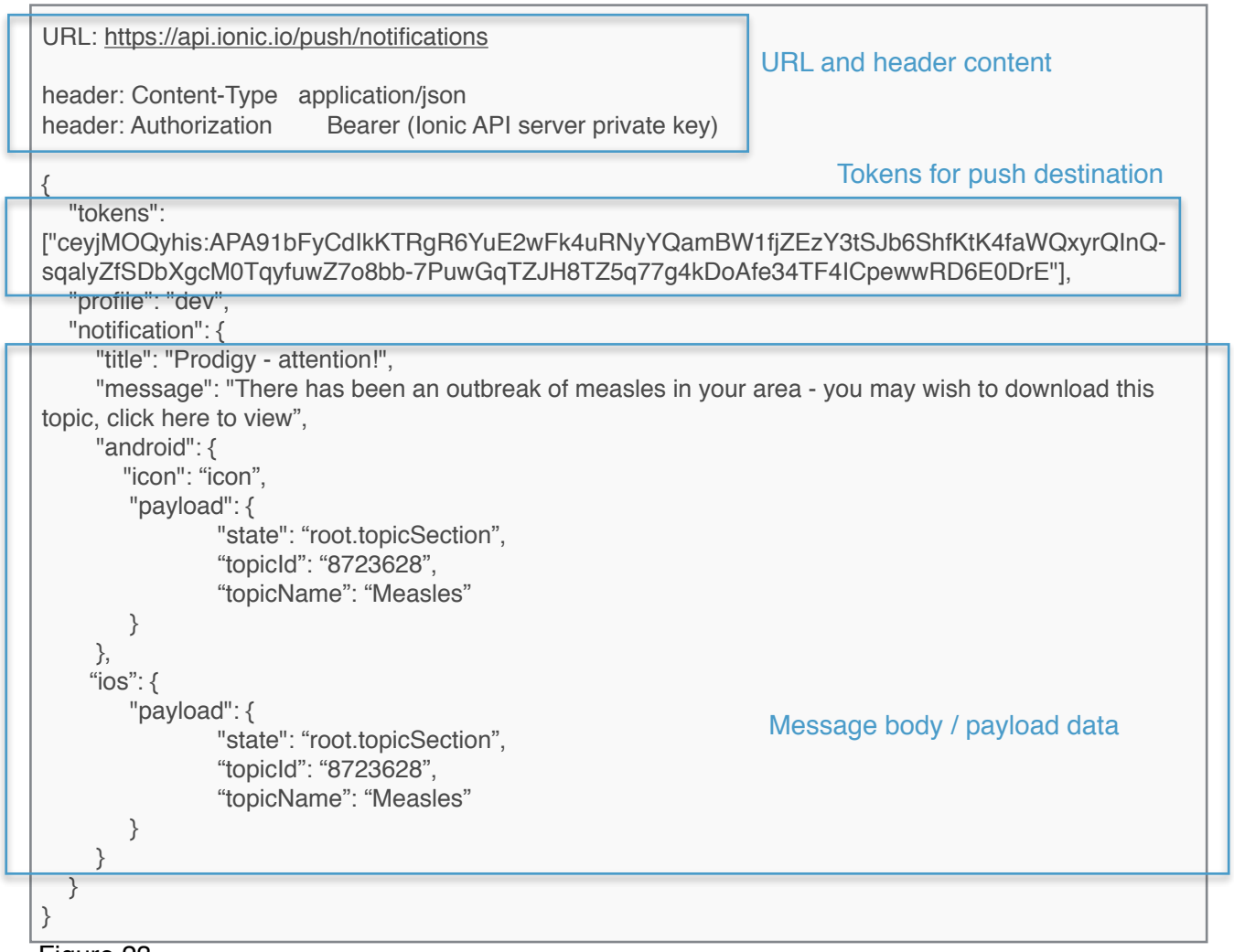


Figure 23

3.4.8 Evidence reading list

Description

As the clinical guidance content presented in this application is evidence based, it is common for the users to read the supplementary material which is linked to from within the 'Evidence' section of the topics. These generally link to the sources of the information, studies, tests etc. The users have the option to confirm the source from which the information has been provided which is fantastic.

However, the problem is that they may not always have an active internet connection and so may wish to come back to the evidence and read it later when they do. Even if they do possess an active connection they may not have the time to read it and may want to save it to a reading list. To accommodate for this, we can implement a page from which they can view all the resources that they would like to read, and automatically add links should the user not have a connection.

Technology comparison

There are a number of technologies to consider, we need to contain the state of the users connection and store the contents of the reading list somewhere. For the storage, with reference to figure 5, SQLite database will be used as it fits all the requirements of this, and the existing infrastructure of the application can be utilised.

Cordova provides an official plugin with methods used to obtain the users internet connection, named 'cordova plugin network information' [34], this is the status quo for obtaining network information in cordova and ionic applications.

Design

The basic flow of this feature is shown in Figure 24.

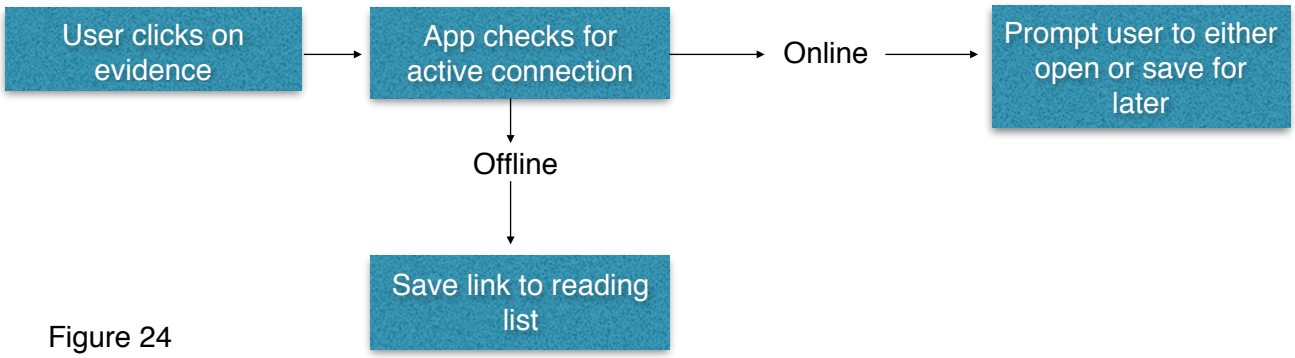


Figure 24

We can catch the attempts at opening the links by making changes to the data coming from the API. To do this, all the hyperlinks in the CKS Evidence sections can be replaced with on-click functionality which calls an angularJs method. From this method we can check for the active internet connection, and then save the data via an angularJs service to its own SQLite table with the schema in figure 25. A new page will be designed from which the user can view their reading list and navigate to each should they possess a connection.

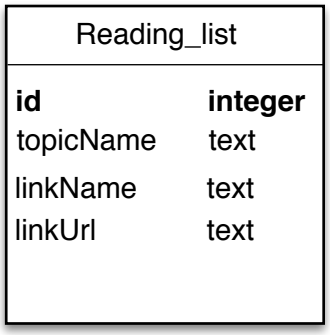


Figure 25

Implementation

The code in appendix figure 18 shows how the controller handles the opening of a piece of evidence, complying with the design in Figure 24. The user is given an option to save the document to their reading list if they have a connection or not, allowing them to compile a list of documents for later. The plugins to access network information [34] and to give responsive messages back to the user [35] have been added to implement this feature using the commands

below. The users receive popups in the form of $ionicPopup and feedback from the toast plugin, maintaining user interaction throughout the procedure.

```
$ cordova plugin add cordova-plugin-network-information
$ cordova plugin add cordova-plugin-x-toast
$ cordova prepare
```

In order to maintain visibility on the users connection status, a variable was added to angular's rootScope (so it is accessible anywhere in the application) named 'userConnected'. It is updated as shown in Figure 26.

```
document.addEventListener("offline", onOffline, false);
document.addEventListener("online", onOnline, false);

$rootScope.firstTimeOpen = true;
$rootScope.userConnected = true;

var state = navigator.connection.type;
if (state == Connection.NONE) {
    $rootScope.userConnected = false;
}

function onOnline() {
    $rootScope.userConnected = true;
    if ($rootScope.firstTimeOpen) {
        window.plugins.toast.showWithOptions({
            message: "Connection re-established",
            duration: "long",
            position: "top",
            styling: {
                opacity: 0.6,
                backgroundColor: '#333333',
                textColor: '#FFFFff'
            }
        });
    }
```

```
function onOffline() {
    $rootScope.userConnected = false;
    window.plugins.toast.showWithOptions({
        message: "Internet connection lost",
        duration: "long",
        position: "top",
        styling: {
            opacity: 0.6,
            backgroundColor: '#333333',
            textColor: '#FFFFff'
        }
    });
};
```

Figure26

In order to display the reading list for the user a new page was added, upon loading it accesses the SQLite database that was set up and retrieves all rows, as shown in appx figure 19. After compiling the list in the angular controller, the HTML snippet in Figure 27 is used to display them, with the results shown in Figure 28.

```
<div class="list">
    <div ng-repeat="(key, value) in model.readingList | groupBy:'topicName'">
        <div class="item item-divider">
            <b>{{key}}</b>
        </div>
        <a ng-repeat="link in value" class="item item-icon-right" href="#" ng-click="openLink(link.linkUrl)">
            {{link.linkName}} <i class="icon ion-trash-a" ng-click="removeLink(link.id)"></i>
        </a>
    </div>
</div>
```

Figure27

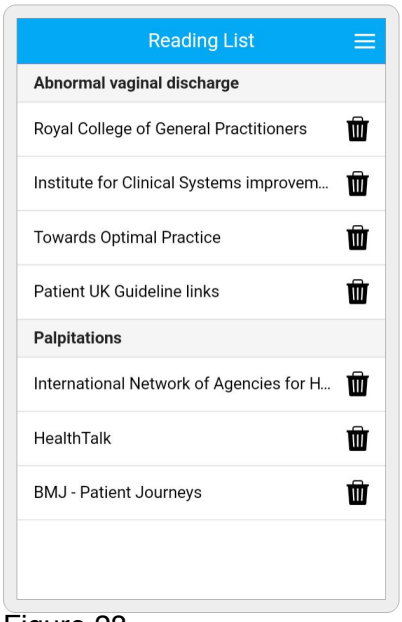


Figure 28

3.4.9 User interface design

Description

A good user interface allows the user to maximise their experience within an application and lets them make use of all the intended functionality. The low level implementation of such interfaces develops with time, however heuristics like the ones outlined by Jakob Nielsen [36] in his 10 Usability Heuristics for Interface Design remain constant and are good platforms from which to base the UI. Guidelines like maintaining 'Visibility of system status' propose that users be kept informed with appropriate feedback mechanisms, while 'Recognition rather than recall' is a good rule of thumb, suggesting users should not have to remember information from one dialogue to the other and the instructions for the applications use should be easily visible and recognisable.

More rigorous, in depth and lower level guidelines are produced by governing body W3C who produce the Web Content Accessibility Guidelines [37] which are essentially recommendations for making Web content more accessible, particularly for those with disabilities.

Technology comparison

There are a number of technologies available to enhance the user interface and keep it consistent and easily usable. Ionic provides a lot of styling functionality right out of the box with its large library and documentation on CSS components. Ionic applications are born with a standard layout which is kept consistent throughout. There are a number of colour scheme generators which help cater for those with colour blindness, like Color Safe [38] to help you define your colour palette.

There are a number of existing css schemas like those provided by Bootstrap, and Google with Google Design Lite. These fit in nicely to the structure of the angularJs/html/css application, however offer much the same functionality in terms of usability and conformation to guidelines as Ionics CSS components.

Design

Storyboard and font-boards provide a good starting point for structuring the flow and design of an application. A mock-up of the page flow and design is shown in Figure 29, with a font-board

shown in Figure 30, and a sample colour scheme shown in Figure 31. You can see in Figure 29 that button placement is kept strict and well placed, and side menu actions are relevant to pages being viewed to ensure that users are aware of actions that they can take on each page. The topic page employs tabs to allow the user to switch between sections of each, with the menus being kept short and concise. The size of buttons and navigation items is kept consistent and is not too large or small for each page.

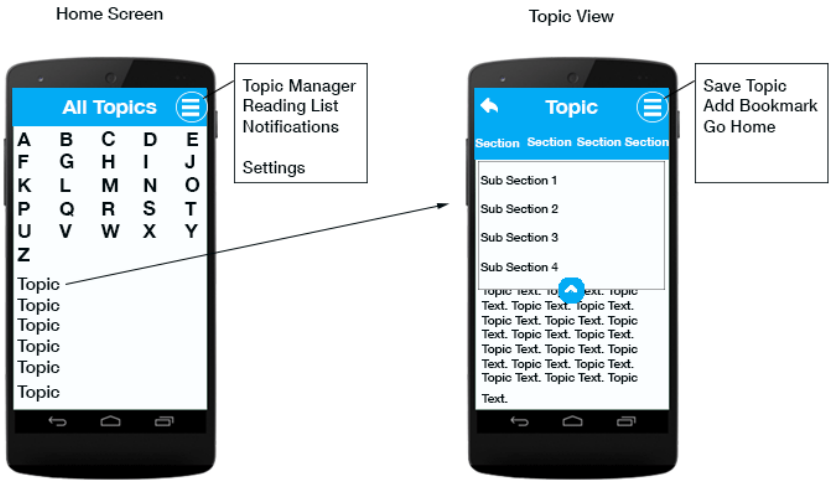


Figure 29

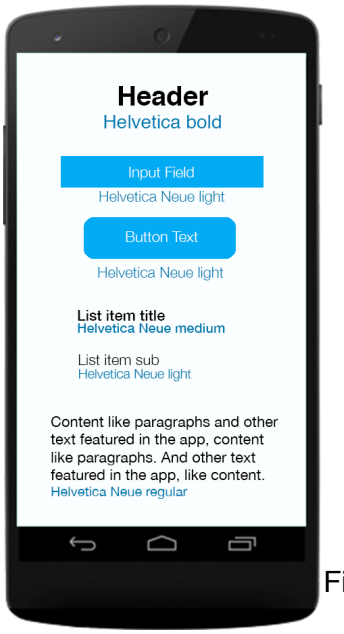


Figure 30

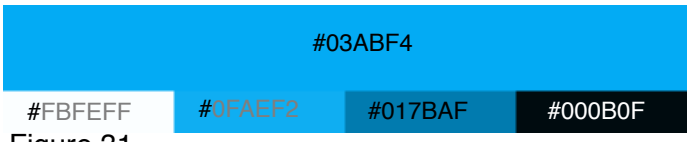


Figure 31

Implementation

In order to provide users with feedback on errors and system status, Ionics in built prompt functionality $ionicPrompt was used alongside the 'toast' plugin to show popups (appx Figure 20). To maintain similar styling throughout the application, Ionics CSS library was the most valuable source of styling, including its Ion-icon library. Developing a useful sheet of classes as shown in appx Figure 21 helped style similar aspects of the application consistently.

The UI for the page in which the user views topics is key as this is where they will spend a large proportion of their time. It is concise, clean and simple but contains all relevant functionality to

the topic on one page, including 2 layers of sub-navigation (appx Figure 22). Users can recognise familiar buttons and will not be distracted with a busy colour scheme. The heading of each page describes its functionality, with the navigation and functionality residing around it, as shown in Figure 32.

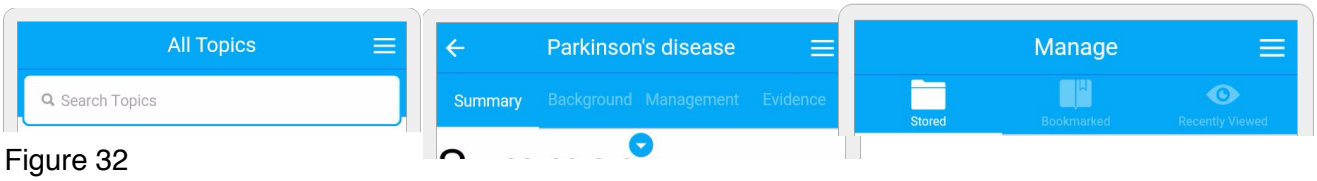


Figure 32

Each page has its own distinguishable functionality, and a sole purpose, keeping pages separate and unique allows users to easily analyse the feature contained on each one, and keep track of where they are and what they are doing. The icons chosen throughout the application are common and easily identifiable to the user (appx Figure 23).

3.5 Summary

This section has walked through the development process of the application beginning with a discussion on the overall platforms used and the architecture of the system. It follows with a feature based design approach, detailing the reasoning behind specific functionality, comparing different technologies that could be used to implement it, designing a feature to solve the problem and then finally implementing it based on the design. Overall the features satisfied a varied range of problems to do with data retrieval, storage and the updating of the stored information. Users are now presented with an application capable of serving their data in the presence of intermittent networks. There are a number of pieces of functionality which could be provided in the future to further prepare for this issue.

Throughout the design C#, HTML, CSS, AngularJS and SQL have been used to design the system with the blocks of information being passed in as XML or JSON objects. Multiple SQL databases have been created both server side and locally in which to store structured data, they account for a large proportion of the storage in this system although operate with a low overhead.

A number of plugins were used throughout the development to provide features, for example the ‘cordova-network-information’ and ‘push-plugin’ and ‘geolocation-plugin’. These are useful and easily added to Ionic applications. Ionic and Cordova have been invaluable throughout the process and are great tools for cross platform application development.

Using an agile approach to development has worked well, it allows for constant analysis and observation of the system and lets the design flow feature by feature as it should. When coming across problems - for example deprecated plugins and pieces of code (Ionics PUSH system) it has allowed for ample time to fix the issue and did not disrupt the rest of the implementation.

Overall the design of the system went well with few upsets although the time taken to implement features was underestimated and development took up a much larger portion of time than first anticipated. Testing still needs to be carried out on the existing functionality, including unit tests and further end-to-end feature testing to ensure that each feature ties in and co-operates with the larger system.

4. Testing and results

4.1 Chapter Introduction

This chapter details the testing of the system, it provides a discussion on the schemes involved with testing and outlines the 3 aspects of testing carried out - Unit testing, Functional testing and User testing, before going into detail on the analysis of results.

4.2 Schemes and approach

As the tests were all performed with an inside knowledge of the product, all cases can be described as white-box testing, although for the most part only the input and output will be assessed. The tests are performed bottom up, initially beginning with individual unit tests of all aspects of the app before grouping the functionality together to test full features. Unit testing must be performed before any further testing in order to obtain confidence in the systems ability to carry out individual low level tasks.

The UI of the product was tested using white box testing, assessing the usability and conformity of each interface. Each button and hyperlink requires testing to ensure that navigation and usability works as expected. There has been a large number of features implemented in this system, each of the database functions, API calls and UI elements will be tested individually. The tests themselves will be carried out in a number of ways, either using the debug tools provided by Ionics debug method (works like the inspection of a web page) or by explicitly created functions.

Each test will comprise of 4 parts, the unit/feature name or location, the test description, expected results and actual results. In the tables containing test results, the ‘actual results’ will be highlighted in red to signify failure of the test.

Following the test of each unit and each feature, end user testing is carried out to assess the system as a whole. There are a number of methods available for this, for example simulating the use of the application and applying end to end testing strategies, shadowing users to observe their behaviour or allowing users to make use of the application before asking them questions.

Testing using these three approaches allows us to ensure that the application works as expected, and provides useful data for analysing the success of the project in terms of the satisfaction of its initial objectives.

4.2 Testing

4.2.1 Unit Testing

Databases / storage

There are a number of internal storage databases present in the system, each is listed here:

- Caching Service (SQLite)
- Storage Service (SQLite)
- Reading List Service (SQLite)
- Bookmarking Service (SQLite)

Each of the databases are managed through these services, each of the services have a number of methods from which to manage the DB. These methods will be tested individually and with both correct and incorrect data, in order to fully ascertain their status.

APIs

API's are used throughout the system via HTTP requests:

- User Location API (external, stores push tokens and location)
- Prodigy API (external, fetches CKS records from the DB)
- PUSH API (used to initiate push requests)
- User Service (accesses User Location API)
- CKS Service (accesses Prodigy API)

Each method in the API's will be executed with correct and incorrect parameters, and the return type of the request will be analysed.

UI

The UI of the application contains many buttons, hyperlinks, and input options. Each HTML page will be analysed and every functional item will be assessed in each, by selecting them and observing the event in order to ascertain if it is correct. It is important that each of the pages works correctly, the templates are listed below:

- Home.html
- Menu.html
- Topic-view.html
- Topic-search.html
- Topic-manager.html
- Reading-list.html
- Database-test.html

Angular Functions

A lot of the applications functionality is executed from within the angular controllers and plugins. In each of the following files the functions present have been run with both correct and incorrect parameters, and the outcome noted.

- HomeController.js
- MenuController.js
- TopicController.js
- TopicManagerController.js
- TopicSearchController.js
- DBTestController.js
- ReadingListController.js
- App.js

Results

The outputs of the unit testing are presented in ‘Test Output 1’, ‘Test Output 2’, ‘Test Output 3’ and ‘Test Output 4’ respectively in the appendices section C. The results show that the above functionality is entirely bug free, with any bugs arising from unit testing being dealt with and re-tested. A large portion of the previously known bugs have been dealt with throughout the agile design process.

4.2.2 Functional Testing

Description

Functional testing assesses the working order of features across the application. These features are made up of multiple units joined together - for example the push functionality relies on units involved with saving user details and geolocation, retrieving user data via an API, and sending pushes via the Ionic API. It is therefore essential that each individual unit is tested before playing a part of a larger system - if they aren't working correctly then they will cause errors in this functional testing section.

In this section the systems expected functionality is identified (largely those features listed in design), the test input data is created based on the functions specifications alongside its expected output. The test is then executed and the resulting outputs are compared to the expected results to assess whether the feature operates as expected. The following pieces of functionality will be tested:

- Apps initialisation
- Caching of downloaded data
- Optimised caching functionality
- Storing specific topics
- Adding bookmarks to topics
- Push notifications
- Location based push notifications
- Evidence reading list

Constraints

There are a number of potential difficulties with testing these features, which will require the tests to be further planned before they are executed. For example the location based push functionality - upon a users login their coordinates are pulled from a geolocation plugin and then stored to a hosted database, however it may be required to simulate the location in a different place in order to get a varied amount of data. This data must be varied as an API function is then called which queries the database in an attempt to find users close to a location, and to achieve a fair test it must be necessary to consider users who are within the distance and those further away.

Results

The results of these tests are shown in Appendix C - Table 5 'Functional Testing'. The results show that all functionality works as expected. This is largely due to it being thoroughly tested throughout the development process, with ample time being left aside from which the features could be fixed. The agile development methodology has a lot to do with the successful passing of these features.

4.2.3 End to end testing

Test Description

End to end testing moves away from the previous style of block testing and is more about the actual flow through a system in an end user scenario. It tests whether the user can operate the application as expected, with a focus on the workflow of the user. As opposed to the previous tests

which ensure that the features and units within them operate effectively, End to End tests determine whether the system as a whole meets requirements or not.

The testing itself can be carried out in a multitude of different ways, users can be given the application to used and then questioned on it after, users can be silently observed with their actions and thought process recorded, or real end user scenarios can be simulated and the expected inputs of the user can be executed virtually.

In this case it is better that we simulate expected use cases of the system in order to account for all eventualities. It would be possible to allow end users to test the app and record their results however as the main focus on the functionality is with offline access it would be hard to find real users in the locality who could take part and provide meaningful data to the test.

In these tests a number of user scenarios will be simulated, i.e. losing connection and the effect it has on their information retrieval. The test will generally end with the attempted retrieval of a topic and marked against the expected result.

Results

The test results are shown in Appendix C - Table 6 ‘End-to-end tests’. These tests run through a number of different sample scenarios a user may execute when using the app in different conditions. The tests all passed and worked as expected, with only minor UI errors being brought to the testers attention. This is largely due to the previous Unit and Functional testing which was carried out to ensure that all aspects of the application were working in their own capacity.

4.3 Summary

A bottom-up testing approach was executed and detailed in this section, beginning with unit testing of all items in the application. These units were then compiled together and each feature or piece of functionality present in the application was tested.

Following these tests End-to-End testing was conducted to ensure that the whole system was compliant and friendly with each other and worked as expected. This full system testing allows us to be confident in the applications ability to work faultlessly for end users, with all the general and specific use cases being identified and simulated.

The testing overall went very well, with very few errors to report, this is due to the rigorous agile development process in which each piece of functionality was fully completed before moving onto the next, and was re-designed to comply with other features of the application as they were introduced. Having this process meant that there was sufficient time and resources allocated to solve problems with integration.

5. Evaluation & Conclusions

5.1 Chapter introduction

This chapter assesses the success of the project as a whole, not only reviewing whether the final system satisfies the aim and objectives outlines in Chapter 1 but also how the project has gone, what was learned throughout the project and how the methodology and planning held together. The results on the conclusion will be justified based on the design process from Chapter 3 and the testing results from Chapter 4. An analysis of the pros and cons of the full system will be conducted and possibilities for future work on the system will be discussed before the final conclusions on the project at the end of this chapter.

5.2 Satisfaction of aims and objectives

The aims and objectives were outlined in Chapter 1 of this document, with the overall aim of the project to get the Clinical Data produced by Clarity into the hands of healthcare professionals in challenging environments, via a cross platform mobile application. A number of objectives were laid out and it is with their fulfilment that the completion of the aim can be determined.

Objective 1: Rapid prototyping of the app, taking data from Clarity's API, hosting it on the web and serving it from there

This objective was more of a developmental requirement. It involved obtaining the CKS data from Clarity, hosting it on a server and then pulling it in as per requirement. The initial iteration of design satisfied this objective, with the creation and deployment of an Ionic/Cordova mobile application capable of sending http requests to external sources.

Objective 2: Research and implement the required functionality to serve data specifically accounting for intermittent or non functioning network connections

This objective has been completed to a certain extent. A good step to satisfying this objective was the re-structuring of content pulled from Clarity's API, instead of pulling a whole topic at once (like was initially intended, with 500kb-1mb topics), it was restructured and split up into sections. The response size of the http request cut by 70%, allowing users even with a slow connection to access the content they needed faster.

Research was undertaken to deduce methods of implementing functionality which would allow users with lacking internet connection still access the data. Due to a larger proportion of users only accessing the clinical data for a small number of topics, it was decided that a method of internal storage would be implemented to allow users to pre-download the content they wished to reference while they were without connection. The issue here is that although users are able to download and bookmark topics, they required to have some idea of which pieces of content they would be wishing to view. It is hard for users to know when they will be without an active connection.

Objective 3: Personalise the offline caching of data to tailor accessibility to individual users needs

The caching mechanism installed in the application alongside the 'Topic Store' satisfy this objective in its entirety. Users are given a cache storing up to 7 of their previously viewed topics. These will get overwritten (oldest date of access first) as new topics are opened. Based on the analytics conducted on the clinical content produced by Clarity this is a large enough cache size for users. To compensate for users who wish to store a larger quantity than this, the 'Topic Store' was implemented allowing users to persist a number of chosen topics, allowing them to choose which topics were stored on their device.

Objective 4: Account for changing epidemics/diseases and the effect this will have on the users requirements for data access

This objective has been satisfied. It was noted initially that users will come from all around the world, and so should there be an outbreak of a disease or illness in a particular location, it is important that users in the disease radius be notified so they are aware and can access information on the disease.

Location based push notifications have been implemented to account for this, with users location details being stored in a DB on Azure each time they log in. Once an outbreak occurs, the DB is queried and the resulting users are sent a push notification to their device detailing the problem and instructing them to review Clarity's content on the topic. This allows users to remain informed, and should they wish to persist the topic for later use they have the option from the notification.

Objective 5: Extensive user testing to obtain confirmation that each potential use case is satisfied with appropriate mechanisms

Testing has been carried out on a number of use cases (Appendix C, table 5) which simulate users in different environments, where the data access may be limited or intermittent. The tests pass and show that for the intended user base of the system the functionality meets the requirements. A total of over 170 tests have been performed to ensure that the functionality implemented works as expected. This objective is satisfied on the basis of the conclusive simulated user testing.

Objective 6: Research usability, accessibility and UX and construct a UI centred around these principles

This objective has been satisfied to some extent. The UI and UX design in this system has been designed thoroughly and implemented effectively following guidelines provided by the likes of Nielsen [36] and the W3C. The design of the system is discussed in 3.4.9. However accessibility for those users with colour blindness or special conditions (larger buttons, voice recognition, audio effects) has not been implemented due to time constraints.

5.3 System Evaluation

5.3.1 Technical point of view

This system is sturdy, contains sufficient functionality and operates quickly with http requests minimised. Not only does the system operate as intended (shown in Appendix C) but underneath the hood it is built modularly and cleanly. The hierarchy of the application is well structured and can easily be added to, with all the code being kept modular and succinct. Ionic and Cordova offer great project tools straight out of the box for applications like this. The application performs its functions quickly and provides good performance for the user. The design features compliment each other and the overall system is easy to navigate and operate.

After analysis of the original objectives, we see that there are a number of caveats to the system in that an alternative method to the offline storage/caching would have been providing a method for users to automatically store all the topics and have them updated as they receive connection, this is elaborated in 5.5.1.

5.3.2 Usability

The finished application is usable and has been thoroughly tested by members of Clarity throughout its agile development process. The systems functionality is explicitly signposted and users are provided with visual feedback and instruction whilst operating.

Although the application is in a working state, due to the constraints involved in obtaining and re-structuring Clarity's data (5.4.4) only a subset of topics are available in the current format.

Aside from the UX, whether the application is in a presentable state to be deployed to the app store is debatable. Whilst the core functionality works dependably and the UI looks and feels good to most users, in order for the app to be deployed it will need to contain further accessibility options for those with disabilities and those who require special assistance.

The data contained in the 'Topic Store' can stagnate and become out of date. Should there be a newer version released, some mechanism for automatically updating the stored version should be implemented. While this doesn't directly impact the usability it is important to ensure users data is up to date.

5.4 Project evaluation

5.4.1 How did the design methodology work out

The time plan defined in section 1.5 outlines the timeframes for the development of the system. For the most part, the time plan was stuck to however the research, development and implementation of the different features required to satisfy the objectives ran on longer than was shown in the Gantt chart. It is hard to diagnose the time required to develop certain features, and while using an Agile development process in which each feature was tweaked and re-designed, certain pieces of functionality and development ate up a larger proportion of time than was intended.

The rapid prototyping of the application allowed us to immediately get set up in a position from which it was possible to assess the requirements and possible functionality of the system.

The agile development methodology as a whole worked fantastically, and provided a great format from which to base the rest of the development. Each requirement was considered, and a suitable feature developed to accommodate it. Throughout the process the requirements to satisfy the projects aims and objectives were all addressed, and the process allowed enough time for the research, design and development of the features. The testing highlights how well implemented each piece of functionality is, with only a small number of tests initially failing out of 170.

A problem arose with the deprecation of a number of Ionics version 1 code, due to Ionics gradual shift out of Beta. This caused a number of problems with the design in that large chunks of development time were used in updating and re-designing push notifications and sqlite to comply with the updated version.

5.4.2 What has been learnt

The author has learnt a large amount from the end to end production of an AngularJS/HTML5/C# application. Learning to communicate with an application from API's and external notification systems has been beneficial, as well as the large amount of programming involved. Planning the development of a mobile system and executing the plan has provided the author with a great deal of understanding of the time and infrastructure involved. Accounting for an undefined number of use cases has taught the author how to consider the wider audience whilst developing public software, and the functionality requirements that might come from each case.

5.4.3 What was good about the project

The end system was designed using modern technologies and and employed an interesting design methodology. Having the option to choose the best suitable technologies for satisfying the original objectives was useful and throughout the project the author was not limited by old development tools meaning each feature was optimally designed.

All in all, the objectives were achieved to a satisfactory level, and it was encouraging throughout the process to see the gradual development of the application.

5.4.4 Mistakes made in design and problems they caused

With regards to the development of the application, the time taken to get functionality working was underestimated. There were periods in design where days of development were spent re-factoring application hierarchy or the navigation of the app with the intentions of mitigating future errors. This was time that would have been better spent researching and designing functionality instead of fixing elements of the app with no overwhelming problems.

With regards to the satisfaction of the aim and objectives, although almost complete, they could have been approached in a different way, with more focus on a general solution to solve the problem, discussed in 5.5.1.

With regards to the content provided via Clarity's Prodigy API, the content itself required a lot of re-structuring to accommodate certain designs like the breakdown into section and sub-section, as well as the addition of the 'onClick' functions added to each piece of evidence. It was time consuming to make changes to this content and so only an isolated number of topics could be edited, there was no pipeline to transform the old content into the new structured version. This meant that the version of the application could not be given to end users as end-user testing and so virtual testing of end users was carried out instead.

5.5 Conclusions

5.5.1 Possibilities for future work on this system

This section outlines further pieces of functionality or approaches which could be considered to further satisfy the projects original aims and objectives.

With regards to the advancement of current functionality in the system there are a number of things which could be improved. Installing an analytics tool to track and records which topics are opened by users could be implemented, allowing us to keep notes on trending topics and those that are most popular. The data collected from this analytics could allow us to pre-download and store the most viewed pieces of information to provide users with offline access, this could be broken down by region so the topics were tailored for each area.

On the storage facility, when the user stores a specific topic it could be positive for the system to automatically download an array of similar topics. This will help satisfy the use case of a doctor visiting an outpatient in an area of no internet, to help them better prepare for the visit. This will require some large scale tailoring of the data provided by Clarity on their end.

The cache functionality could have been optimised further, providing users with a variable 'max-cache-size', allowing them to store as many topics as they wished to their internal memory. The reason the cap was added was to avoid inflation of the average users app size, but leaving this choice up to the user could be beneficial.

With regards to a different approach to the system design, an option to allow users to pre-download all topics available in Prodigy could be investigated. It would be possible to allow users to choose between two versions of the application, one which employed the features listed in this documentation, and one which pre-populated the application with all available content. This content could then be accessed while offline however would still need regular updating, via an application update. This is interesting as it allows the user a somewhat more reliable method of obtaining the data, regardless of whether it is the current version, as it is more useful to have access to somewhat updated content over none at all.

6. References

[1] Clarity Informatics. 2015. Prodigy - CKS. [ONLINE] Available at: http://prodigy.clarity.co.uk/. [Accessed 04 December 15].

[2] The Center for Universal Design - Universal Design Principles. (2015). The Center for Universal Design - Universal Design Principles. [ONLINE] Available at: https://www.ncsu.edu/ncsu/design/cud/about_ud/udprinciplestext.htm

[3] Ferreira, J., Noble, J. and Biddle, R., (2007), August. Agile development iterations and UI design. In Agile Conference (AGILE), 2007 (pp. 50-58). IEEE.

[4] Gordon, V.S. and Bieman, J.M., (1995). Rapid prototyping: lessons learned. IEEE software, (1), pp.85-95.

[5] Armgren M., (2015). Mobile Cross-platform development versus native development A look at Xamarin Platform

[6] Upkar Varshney. 2003. Location management for mobile commerce applications in wireless Internet environment. *ACM Trans. Internet Technol.* 3, 3 (August 2003), 236-255

[7] Wankhade, N.V. and Deshpande, S.P., 2015. A Review on Databases for Mobile Devices. International Journal of Electronics, Communication and Soft Computing Science & Engineering (IJECSCSE), p.276

[8] Gravelle, R. (2015). Apache Cordova Persistence Options. Available: www.htmlgoodies.com/html5/javascript/apache-cordova-persistence-options.html#fbid=AbFeNpQtwRc. Last accessed 25th Jan 2016.

[9] Number of Internet Users (2015) - Internet Live Stats. Number of Internet Users (2015) - Internet Live Stats. Available at: http://www.internetlivestats.com/internet-users/#trend. Last accessed 26th Jan 2016.

[10] Purves, I.N., 1997. Implementing clinical guidance in General Practice using computerised information systems. PRODIGY phase one thesis. Newcastle upon Tyne: Newcastle University

[11] Eysenbach G, Jadad AR. Evidence-based Patient Choice and Consumer health informatics in the Internet age. J Med Internet Res 2001;3(2):e19. URL: http://www.jmir.org/2001/2/e19

[12] Univadis. 2016. Consult - by Univadis. [ONLINE] Available at: http://consult.univadis.com. Accessed 14 March 2016.

[13] World Health Organization, 2004. International statistical classification of diseases and health related problems (The) ICD-10 (Doctoral dissertation, World Health Organization).

[14] Health Level Seven International - Homepage. 2016. Health Level Seven International - Homepage. [ONLINE] Available at: http://www.hl7.org/. Accessed 17 March 2016.

[15] D. Bender and K. Sartipi, "HL7 FHIR: An Agile and RESTful approach to healthcare information exchange," Computer-Based Medical Systems (CBMS), 2013 IEEE 26th International Symposium on, Porto, 2013, pp. 326-331.

[16] Medscape App for Android. 2016. Medscape. [ONLINE] Available at: http://www.medscape.com/public/android. Accessed 17 March 2016.

[17] NICE Clinical Knowledge Summaries (CKS). 2016. NICE Clinical Knowledge Summaries (CKS). [ONLINE] Available at: http://cks.nice.org.uk. Accessed 17 March 2016.

[18] Lella, A. and Lipsman, A., 2014. The US mobile app report. 21st August, available at: http://www.comscore.com/Insights/Presentations-and-Whitepapers/2014/The-US-Mobile-App-Report Accessed 17th March, 2016).

[19] Sowerby Centre for Health Informatics at Newcastle Ltd. (SCHIN), 2005. Prodigy Knowledge: Practical, Reliable, Evidence-based Guidance. 2nd Edition. Stationery Office.

[20] Apache Cordova . 2016. Apache Cordova . ONLINE Available at: https://cordova.apache.org/. [Accessed 17 March 2016].

[21] Ionic: Advanced HTML5 Hybrid Mobile App Framework. 2016. Ionic: Advanced HTML5 Hybrid Mobile App Framework. [ONLINE] Available at: http://ionicframework.com/. [Accessed 17 March 2016].

[22] PhoneGap. 2016. PhoneGap. [ONLINE] Available at: http://phonegap.com/. [Accessed 17 March 2016].

[23] Mobile Application Development to Build Apps in C# - Xamarin. 2016. Mobile Application Development to Build Apps in C# - Xamarin. [ONLINE] Available at: https://www.xamarin.com/platform. [Accessed 17 March 2016].

[24] Mobile App Development & MBaaS Products | Appcelerator. 2016. Mobile App Development & MBaaS Products | Appcelerator. [ONLINE] Available at: http://www.appcelerator.com/mobile-app-development-products/. [Accessed 17 March 2016].

[25] Microsoft Azure: Cloud Computing Platform and Services. 2016. Microsoft Azure: Cloud Computing Platform and Services. [ONLINE] Available at: https://azure.microsoft.com/en-gb/. [Accessed 22 March 2016].

[26] Cordova-sqlite-storage: A Cordova/PhoneGap plugin to open and use sqlite databases on Android & iOS with HTML5/Web SQL API. 2016. [ONLINE] Available at: https://github.com/litehelpers/Cordova-sqlite-storage. [Accessed 22 March 2016].

[27] LocalForage: Offline storage, improved. Wraps IndexedDB, WebSQL, or localStorage using a simple but powerful API. 2016. [ONLINE] Available at: https://github.com/mozilla/localForage. [Accessed 22 March 2016].

[28] Cordova-file-cache: An awesome File Cache for Cordova Apps. 2016. Available at: https://github.com/markmarijnissen/cordova-file-cache. [Accessed 22 March 2016].

[29] Cordova-plugin-indexeddb-async: An asynchronous IndexedDB plug-in for Cordova apps. 2016. Available at: https://github.com/ABB-Austin/cordova-plugin-indexeddb-async. [Accessed 22 March 2016].

[30] LocalStorage - Ionic API Documentation - Ionic Framework. 2016. Available at: http://ionicframework.com/docs/v2/api/platform/storage/LocalStorage/. [Accessed 22 March 2016].

[31] Phonegap-plugin-push: Register and receive push notifications. 2016. Available at: https://github.com/phonegap/phonegap-plugin-push. [Accessed 23 March 2016].

[32] Pushwoosh. 2016. Available at: http://docs.pushwoosh.com/docs/cordova-phonegap. [Accessed 23 March 2016].

[33] Cordova - Geolocation plugin. 2016. [ONLINE] Available at: http://ngcordova.com/docs/plugins/geolocation/. [Accessed 01 April 2016].

[34] Cordova plugin network information: Mirror of Apache Cordova Plugin network-information. 2016. Available at: https://github.com/apache/cordova-plugin-network-information. [Accessed 03 April 2016].

[35] Toast-PhoneGap-Plugin: A Toast popup plugin for your fancy Cordova app. 2016. Available at: https://github.com/EddyVerbruggen/Toast-PhoneGap-Plugin. [Accessed 05 April 2016].

[36] 10 Heuristics for User Interface Design: Article by Jakob Nielsen. 2015. 10 Heuristics for User Interface Design: Article by Jakob Nielsen. [ONLINE] Available at: http://www.nngroup.com/articles/ten-usability-heuristics/. [Accessed 02 December 2015].

[37] Web Content Accessibility Guidelines 2.0, B. Caldwell, M. Cooper, L. G. Reid, G. Vanderheiden (eds.), W3C Recommendation 11 December 2008. This version is http://www.w3.org/TR/2008/REC-WCAG20-20081211/ . The latest version of WCAG 2.0 is available at http://www.w3.org/TR/WCAG20/

[38] Color Safe - accessible web color combinations. 2015. Color Safe - accessible web color combinations. [ONLINE] Available at: http://colorsafe.co/. [Accessed 02 December 2015].

7. Appendices

7.1 Appendix A

Glossary

Term	Definition
Clarity	Clarity Informatics [1], a Healthcare Informatics company based in Jesmond, Newcastle. They provide quality improvement services to primary healthcare professionals. Clarity authors the content produced for Prodigy, and they have commissioned me to address the issue in this project.
Prodigy	Prodigy is a product produced by Clarity, and provides evidence based clinical guidelines in the form of CKS topics, used by primary healthcare professionals (namely GPs) who require access to information on a number of healthcare topics in order to help them in decision making.
CKS	A Clinical Knowledge Summary is a report on a healthcare topic, for example 'Acne'. A CKS is produced for the Prodigy product and contains evidence based clinical guidance to be used by primary healthcare professionals. Each CKS is created over a common node hierarchy and contains important information on the subject.
Cordova	Apache Cordova packages an HTML5 app as a native app that can run in Android, iOS, and other platforms.
Ionic	Ionic builds on top of Cordova, it provides a set of front-end components that lets you write an HTML5 app that looks like a native app.
Azure	It is cloud computing platform and infrastructure created by Microsoft for building, deploying, and managing applications and services through a global network of Microsoft-managed data centres.
NICE	National Institute for Health and Care Excellence

7.2 Appendix B

Google Analytics

Live - http://cks.nice.org.uk Go to this report
CKS

OLD - CKS Service Report

Aug 1, 2015 - Aug 31, 2015

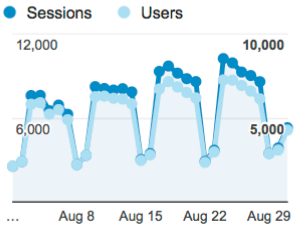
All Sessions
100.00%

Visits
205,223
% of Total: **100.00% (205,223)**

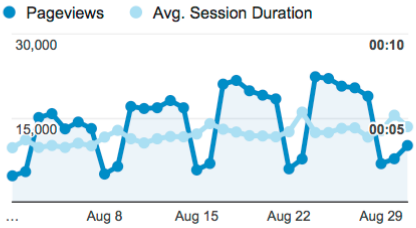
Pageviews
433,793
% of Total: **100.00% (433,793)**

Total Unique Searches
40,628
% of Total: **100.00% (40,628)**

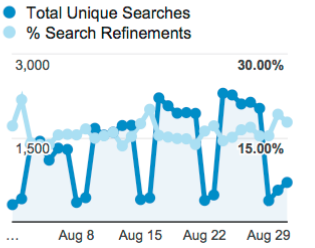
Visits and Unique Visitors



Pageviews and Avg. Visit Duration



Total Unique Searches and % Sear...



Unique Visitors and Visits by Month ...

Month of Year	Users	Sessions
201508	109,235	205,223

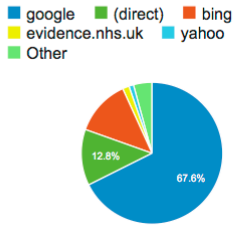
Pageviews and Avg. Visit Duration by Month of Year

Month of Year	Pageviews	Avg. Session Duration
201508	433,793	00:03:58

Total Unique Searches and % Sear...

Month of Year	Total Unique Searches	% Search Refinements
201508	40,628	15.41%

Visit Referrals



Visits by Page

Page	Sessions
/	55,371
/search	3,437
/urinary-tract-infection-lower-women	3,220
/anaemia-b12-and-folate-deficiency	2,749
/anticoagulation-oral	2,226
/menopause	2,158
/hypertension-not-diabetic	2,050
/diabetes-type-2	2,004
/hypothyroidism	1,795
/hyponatraemia	1,681

Visits with Search by Search Term

Search Term	Sessions with Search
uti	61
gout	48
acne	44
hypertension	42
shingles	33
asthma	32
copd	32
migraine	31
thrush	25
hrt	22

Appendix Figure 1

```
    cachingService.isTopicInCache($scope.model.topicName, $scope.model.sectionType)
.then(function (response) {
    if (response === true) {

        cachingService.getContentFromCache(topicId, $scope.model.sectionType).then(function (section) {
            $scope.model.section = section;
            $scope.model.section.sectionNodes = angular.fromJson(section.sectionHtml);

            $("#topicSpinner").hide();
            $("#section-content").show();
            $scope.scrollToScrollPos();
        }, function (error) { });

    } else if (response === false) {
        cksService.getSection(topicId, $scope.model.sectionType).then(function (section) {

            $scope.model.section = section;
            var nodes = angular.toJson(section.sectionNodes);


            $("#topicSpinner").hide();
            $("#section-content").show();
            $scope.scrollToScrollPos();

            cachingService.saveContentToCache(topicId, $scope.model.topicName, $scope.model.sectionType, nodes);

        }, function (error) { });
    }
});
```

Appendix Figure 2

```
[HttpGet]
[Route("api/content/get-topic/{dosserId}")]
0 references
public IHttpActionResult Get(string dossierId)
{
    var context = new ProdigyDataContext();
    var content =  context.Content.FirstOrDefault(x => x.SectionId == dossierId);
    if (content == null)
    {
        return NotFound();
    }
    return Ok(new ContentModel { dossierId = content.DossierId, sectionId = content.SectionId,
        sectionName = content.SectionName, sectionHtml = content.SectionHtml, sectionType = (int) content.SectionType });
}
```

Appendix Figure 3
In: CKS API Azure

```
service.getContentFromCache = function (topicId, sectionType) {
    return $q(function (resolve, reject) {
        service.db.executeSql("SELECT topicId, topicName, sectionType, sectionHtml FROM content_cache WHERE topicId=? AND sectionType=?;",
            [topicId, sectionType], function (res) {
            if (res.rows.length > 0) {
                resolve(res.rows.item(0));
            } else {
                resolve(false);
            }
        }, function (error) {
            reject(console.log('SELECT error: ' + error.message));
        });
    });
};
```

Appendix Figure 4
In: Cache Service

```
service.saveContentToCache = function (topicId, topicName, sectionType, sectionHtml) {
    return $q(function (resolve, reject) {
        service.db.executeSql("UPDATE content_cache SET [topicId] = ?, [topicName] = ?, [sectionType] = ?, [sectionHtml] = ?, [currentTimeMillis] = ? " +
            "WHERE currentTimeMillis = (SELECT min(currentTimeMillis) FROM content_cache);"
            , [topicId, topicName, sectionType, sectionHtml, service.getTime()], function (res) {
                resolve(true);
            }, function (error) {
                console.log('INSERT error: ' + error.message);
                reject(false);
            });
    });
};
```

Appendix Figure 5
In: Cache Service

```
[HttpGet]
[Route("api/content/get-section-2/{topicId}/{sectionType}")]
0 references
public IHttpActionResult GetSection2(string topicId, int sectionType)
{
    var context = new ProdigyDataContext();
    var content = context.Content.FirstOrDefault(x => x.DossierId == topicId && (int)x.SectionType == sectionType);

    if (content == null)
    {
        return NotFound();
    }


    var nodes = Regex.Unescape(content.SectionHtml);

    var returnType = new ContentModel2 {
        dossierId = content.DossierId,
        sectionId = content.SectionId,
        sectionName = content.SectionName,
        sectionNodes = JsonConvert.DeserializeObject<List<Node>>(nodes),
        sectionType = (int)content.SectionType };

    return Ok(returnType);
}
```

Appendix Figure 6
In: CKS API Azure

```
var i = 0;
var populateDb = function () {
    $timeout(function () {
        i++;
        service.db.executeSql("INSERT INTO content_cache (topicId, topicName, sectionType, sectionHtml, currentTimeMillis) " +
               "VALUES (?, ?, ?, ?, ?);", ['0-0-0', '0-0-0', '0-0-0', '0-0-0', i], function (res) {
                   resolve(true);
               }, function (error) {
                   console.log('INSERT error: ' + error.message);
                   reject(false);
               });
        if (i < 7) {
            populateDb();
        }
    }, 50);
};

service.init = function () {
    if (!service.db) {
        service.db = window.sqlitePlugin.openDatabase({ name: "my.db" });
        service.db.executeSql('CREATE TABLE IF NOT EXISTS content_cache (id integer primary key, topicId text,' +
            'topicName text, sectionType text, sectionHtml text, currentTimeMillis integer);');
        service.getAllRows().then(function (rows) {
            if (rows.length < 7) {
                i = rows.length;
                populateDb();
            }
        }, function (error) { });
    }
};
```

Appendix Figure 7
In: Cache Service

```
service.saveContentToCache = function (topicId, topicName, sectionType, sectionHtml) {
    return $q(function (resolve, reject) {
        service.db.executeSql("UPDATE content_cache SET [topicId] = ?, [topicName] = ?, [sectionType] = ?, [sectionHtml] = ?, [currentTimeMillis] = ? " +
            "WHERE currentTimeMillis = (SELECT min(currentTimeMillis) FROM content_cache);"
            , [topicId, topicName, sectionType, sectionHtml, service.getTime()], function (res) {
                resolve(true);
            }, function (error) {
                console.log('INSERT error: ' + error.message);
                reject(false);
            });
    });
};
```

Appendix Figure 8
In: Cache Service

```
<ion-item menu-close ng-click="broadcastSaveTopic()" class="item-icon-right">
    Save Topic
    <i class="icon ion-ios-download"></i>
</ion-item>
```

```
$scope.broadcastSaveTopic = function () {
    $rootScope.$broadcast("saveTopic");
};
```

Appendix Figure 9
In: Menu.html/Menu Controller

```
angular.module('app.services').factory('storingService', ['$http', '$q', function ($http, $q) {

    var service = {};

    service.init = function () {
        if (!service.db) {
            service.db = window.sqlitePlugin.openDatabase({ name: "my.storing.db" });
            service.db.executeSql('CREATE TABLE IF NOT EXISTS content_store (id integer primary key,' +
                ' topicId text, topicName text, sectionHtml text, sectionType text, userComment text);');
        }
    };

    service.isTopicInStore = function (topicName)...;

    service.saveContentToStore = function (topicId, topicName, sectionHtml, sectionType, userComment)...;

    service.getContentFromStore = function (topicId, sectionType)...;

    service.removeContentFromStore = function (topicName)...;

    service.getAllRowsStore = function ()...;

    return service;
}
]);
```

Appendix Figure 10
In: Storing Service

```
$scope.$on("saveTopic", function () {
    storingService.isTopicInStore($scope.model.topicName).then(function (success) {
        if (success === true) {
            $ionicPopup.alert({ title: "Saved", template: 'This topic is already saved in your topic manager' });
        } else if (success === false) {

            $ionicPopup.show({
                template: '<input type="text" ng-model="model.saveComment">',
                title: 'Add a note',
                subTitle: 'Something to remind you later',
                scope: $scope,
                buttons: [
                  { text: 'Cancel' },
                  {
                      text: '<b>Save</b>',
                      type: 'button-positive',
                      onTap: function (e) {
                          if (!$scope.model.saveComment) {
                              e.preventDefault();
                          } else {
                              for (var key in SECTION_TYPE) {
                                  if (key !== 'Topic') {
                                      cksService.getSection(topicId, SECTION_TYPE[key]).then(function (section) {
                                          var nodes = angular.toJson(section.sectionNodes);
                                          storingService.saveContentToStore(topicId, $scope.model.topicName, nodes, section.sectionType, $scope.model.saveComment)
                                              .then(function (success) { }, function (error) { });
                                      }, function (error) { });
                                  }
                              }
                              $ionicPopup.alert({
                                  title: "Saved success",
                                  template: 'You can now access this topic offline from the topic manager'
                              });
                              return 0;
                          }
                      }
                  }
                ]
            });
        }
    }, function (error) { });
});
```

Appendix Figure 11
In: Topic Controller

```
$scope.loadSection = function (sectionType) {
    $scope.model.sectionType = sectionType;
    $("#topicSpinner").show();
    $("#section-content").hide();
    storingService.getContentFromStore(topicId, $scope.model.sectionType).then(function (val) {
        if (val !== false) {
            $scope.model.section = val;
            $scope.model.section.sectionNodes = angular.fromJson(val.sectionHtml);

            $("#topicSpinner").hide();
            $("#section-content").show();
            $scope.scrollToScrollPos();
        } else {
            cachingService.isTopicInCache($scope.model.topicName, $scope.model.sectionType)
        .then(function (response) {
            if (response === true) {
```

Appendix Figure 12
In: Topic Controller

```
var uid = uuid2.newuuid();

var details = {
    'email': uid + '@example.com',
    'password': 'secretpassword'
};

Ionic.Auth.signup(details).then(function () {
    var options = { 'remember': true };
    Ionic.Auth.login('basic', options, details).then(function () {
        user = Ionic.User.current();
        user.set('uid', uid);
        user.save();

        var push = new Ionic.Push({
            "debug": true,
            "onNotification": function (notification)...,
            "onRegister": function (data)...,
            "pluginConfig": {
                "android": ...,
                "ios": ...
            }
        });


        push.register(function (token) {
            console.log("Device token:", token.token);
            push.saveToken(token);
```

Appendix Figure 13
In: App.js

```
var push = new Ionic.Push({
    "debug": true,
    "onNotification": function (notification) {
        console.log("on notification: " + notification.title);
        if (notification.payload.state) {
            var redirect = $ionicPopup.confirm({
                title: notification.title,
                template: notification.text + "<br/><br/>" + "Go to '" + notification.payload.topicName + "'?"
            });

            redirect.then(function (res) {
                if (res) {
                    var params = {
                        topicId: notification.payload.topicId,
                        topicName: notification.payload.topicName,
                        sectionType: 1
                    };
                    $state.go(notification.payload.state, params);
                }
            });
        }
        return true;
    },
```

Appendix Figure 14
In: App.js

-

```
var uid = uuid2.newuuid();

var details = {
    'email': uid + '@example.com',
    'password': 'secretpassword'
};
```

```
'use strict'
angular.module('angularUUID2', []).factory('uuid2', [
    function () {
        function s4() {
            return Math.floor((1 + Math.random()) * 0x10000)
                .toString(16)
                .substring(1);
        }

        return {

            newuuid: function () {
                // http://www.ietf.org/rfc/rfc4122.txt
                var s = [];
                var hexDigits = "0123456789abcdef";
                for (var i = 0; i < 36; i++) {
                    s[i] = hexDigits.substr(Math.floor(Math.random() * 0x10), 1);
                }
                s[14] = "4"; // bits 12-15 of the time_hi_and_version field to 0010
                s[19] = hexDigits.substr((s[19] & 0x3) | 0x8, 1); // bits 6-7 of the
                s[8] = s[13] = s[18] = s[23] = "-";
                return s.join("");
            },
            newguid: function () {
                return s4() + s4() + '-' + s4() + '-' + s4() + '-' +
                    s4() + '-' + s4() + s4() + s4();
            }
        }

    }]);
```

Appendix Figure 15
In: UUID Plugin

```
push.register(function (token) {
    console.log("Device token:", token.token);
    push.saveToken(token);

    var onSuccess = function (position) {
        lat = position.coords.latitude;
        long = position.coords.longitude;
        console.log('position success');
        userService.saveUser(uid, token.token, lat, long).then(function (success) {
            console.log("user location saved");
        }, function (error) { });
    };

    var onError = function (error) {
        console.log('error in position');
    };

    navigator.geolocation.getCurrentPosition(onSuccess, onError);

});
```

Appendix Figure 16
In: App.js

```
[HttpGet]
[Route("api/users/get-users-within-distance/{latitude}/{longitude}/{maxDistance}")]
0 references
public IHttpActionResult GetCoordinatesInRange(double latitude, double longitude, double maxDistance)
{
    var context = new ProdigyDataContext();

    var users = context.Users.Where(x => x.UserUid != null).ToList();
    var tokensOfUsersInRange = new List<string>();
    var epidemicCoordinates = new GeoCoordinate(latitude, longitude);

    for (var i = 0; i < users.Count; i++)
    {
        var userCoords = new GeoCoordinate(System.Convert.ToDouble(users[i].Latitude), System.Convert.ToDouble(users[i].Longitude));
        var distanceBetween = userCoords.GetDistanceTo(epidemicCoordinates);
        if(maxDistance > distanceBetween)
        {
            tokensOfUsersInRange.Add(users[i].PushToken);
        }
    }

    return Ok(tokensOfUsersInRange);
}
```

Appendix Figure 17
In: Azure API

```
$scope.openLink = function (linkName, linkAddress) {
    if ($rootScope.userConnected) {
        var readingListSave = $ionicPopup.confirm(...);
        readingListSave.then(function (res)...);
    } else {
        var readingListSave = $ionicPopup.confirm({
            title: 'No active internet connection',
            template: 'You cannot access this resource at the moment - would you like to save it to your "Reading List" for later?'
        });

        readingListSave.then(function (res) {
            if (res) {
                readingListService.saveLinkToReadingList($scope.model.topicName, linkName, linkAddress).then(function () {
                    window.plugins.toast.showWithOptions({
                        message: "Link saved to reading list",
                        duration: "long",
                        position: "top",
                        styling: {
                            opacity: 0.6,
                            backgroundColor: '#333333',
                            textColor: '#FFFFff'
                        }
                    });
                }, function (error) { });
            }
        });
    }
};
```

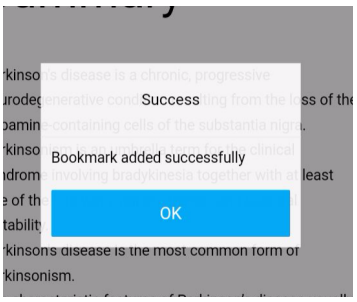
Appendix Figure 18
In: Topic Controller

```
angular.module('app.controllers').controller('ReadingListCtrl',
    ['$scope', 'readingListService', '$rootScope',
    function ($scope, readingListService, $rootScope) {

        $scope.model = {};
        $scope.model.readingList = [];

        $scope.init = function () {
            $scope.refreshReadingList();
        };

        $scope.refreshReadingList = function () {
            $scope.model.readingList = [];

            readingListService.getAllLinks().then(function (rows) {
                for (var i = 0; i < rows.length; i++) {
                    $scope.model.readingList[i] = rows.item(i);
                }
            }, function (error) { });
        };
```

```
service.getAllLinks = function () {
    return $q(function (resolve, reject) {
        service.db.executeSql("SELECT * FROM reading_list;", [], function (res) {
            resolve(res.rows);
        }, function (error) {
            reject(console.log('SELECT error: ' + error.message));
        });
    });
};
```

Appendix Figure 19
In: ReadingList Controller/Service

```
bookmarkService.bookmarkTopic($scope.model.topicId, $scope.model.topicName, $scope.model.sectionType, scrollPos)
 .then(function (success) {
     $ionicPopup.alert({
         title: "Success", template: 'Bookmark added successfully'
     });
 }, function (error) {
     $ionicPopup.alert({
         title: "Bookmark error", template: 'Something went wrong, please try again'

     });
 });
```

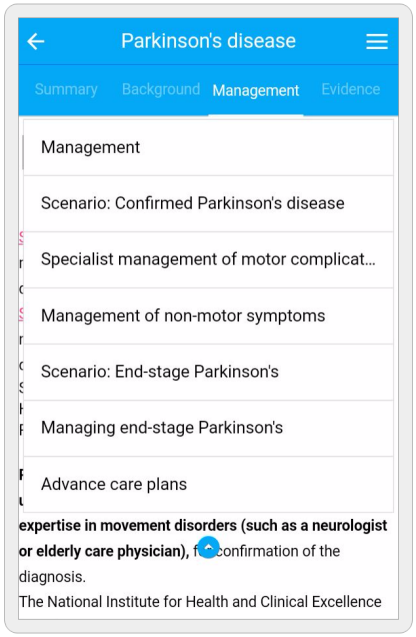


Appendix Figure 20
In: Topic Controller

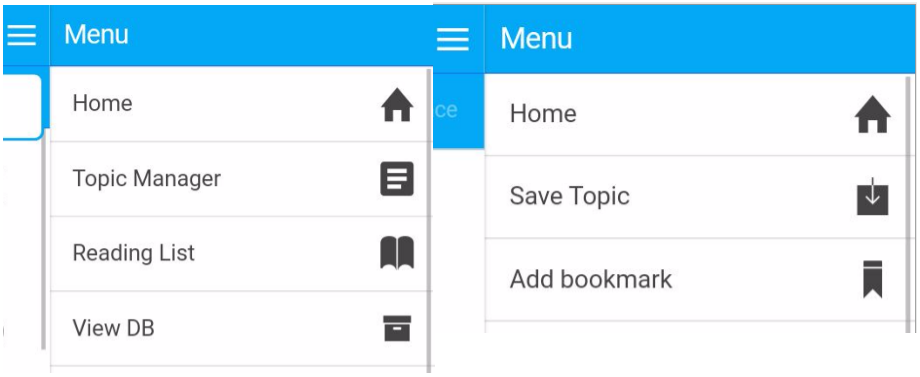
-

```css
.condition-icons {
    width: 100%;
    margin: 0 auto;
}
.condition-icons > button {
    width: 1.9em;
    margin: 0 0.1em;
    display: inline-table;
    background-color: #FFFFFF;
}
.condition-icons > button.active{
    background-color: #03A9F4;
    color: white
}
.condition-icons > button:hover{
    background-color: #03A9F4;
    color: white
}
.condition-icons > button.disabled > span {
    color: grey;
}
.letter-link {
    font-size: 2.4em;
    line-height: 1.5em;
}
.condition-list{
    font-size: 1.6em;
    line-height: 1.8em;
}
.condition-list > li {
    width:100%;
}
.condition-list > li > a {
    width:100%;
    color: #444;
    text-decoration: none;
}
.condition-list > li:hover {
    background-color: #03A9F4;
    color:white;
}
.condition-list > li:hover > a {
    color:white;
}
div.spinner {
    margin: 0 auto;
    position: absolute;
    top: 100px; left: 0; bottom: 0; right: 0;
    height: 85px;
    text-align: center;
}
.spinner svg {
  width: 56px;
  height: 56px;
}
.page-has-header {
    padding-top: 60px;
}
.section-tab {
    left: 0px;
}
.tabs-striped.tabs-background-topic .tabs {
  background-color: #03A9F4;
  background-image: none;

}
.glowing-border {
    border: 2px solid #03A9F4;
    border-radius: 7px;
}
.glowing-border:focus {
    outline: none;
    border-color: #03A9F4;
    box-shadow: 0 0 10px #03A9F4;
}
.subSection-list{
    margin-top: 90px;
    width:100%;
}
```

Appendix Figure 21
In: Style.css



Appendix Figure 22
In: Topic View



Appendix Figure 23
In: Menu.html

7.3 Appendix C

Testing results displayed on following page.

Test Output 1 - Databases/storage				
Test #	**Name of unit**	**Description**	**Expected Result**	**Actual Result**
1	Caching Service	init() test : Load DB for first time and allow for auto-population of 7 rows	7 insert SQL statements executed populating cache	7 insert SQL statements executed populating cache
2	Caching Service	init() test : Load DB first second time	No SQL inserts executed	No SQL inserts executed
3	Caching Service	call isTopicInCache method to check for topic not in cache	false returned by function	false returned by function
4	Caching Service	call isTopicInCache method to check for topic in cache	true returned by function	true returned by function
5	Caching Service	saveContent to cache with correct params	true returned signifying successful insert	true returned signifying successful insert
6	Caching Service	saveContent to cache with incorrect params	false returned and error output to console	false returned and error output to console
7	Caching Service	get content from cache that is present	record returned as object	record returned as object
8	Caching Service	get content from cache that is not in cache	false returned	false returned
9	Caching Service	get content from cache with incorrect params	error logged and false returned	error logged and false returned
10	Caching Service	get all rows function called	all cache rows returned	all cache rows returned
11	Caching Service	get all rows cache function called	all cache rows containing CKS data returned	all cache rows containing CKS data returned
12	Storage Service	init() test: Load DB for first time	Database opened and new table created	Database opened and new table created
13	Storage Service	init() test: Load DB for second time	Database opened, no new table	Database opened, no new table
14	Storage Service	call isTopicInStore with correct parameters	True returned from function	True returned from function
15	Storage Service	call isTopicInStore looking for topic not in store	False returned from function	False returned from function
16	Storage Service	call saveContentToStore with correct parameters	Row inserted and true returned	Row inserted and true returned

Test Output 1 - Databases/storage				
Test #	**Name of unit**	**Description**	**Expected Result**	**Actual Result**
17	Storage Service	call saveContentToStore with incorrect parameters	'INSERT error' thrown	'INSERT error' thrown
18	Storage Service	call getContentFromStore to retrieve row	Row returned as object	Row returned as object
19	Storage Service	call getContentFromStore to retrieve row not in Store	false returned from function	false returned from function
20	Storage Service	call removeContentFromStore to remove topic	All sections of topic removed from store and true returned	All sections of topic removed from store and true returned
21	Storage Service	call removeContentFromStore with null param	'SELECT error' thrown	'SELECT error' thrown
22	Storage Service	call getAllRowsStore to return contents of DB	All rows returned from DB	All rows returned from DB
23	Reading List Service	init() test: Load DB for first time	Database opened and new table created	Database opened and new table created
24	Reading List Service	init() test: Load DB for second time	Database opened, no new table	Database opened, no new table
25	Reading List Service	call saveLinkToReadingList with correct parameters	Row inserted into table and true returned from function	Row inserted into table and true returned from function
26	Reading List Service	call saveLinkToReadingList with incorrect parameters	INSERT error thrown and false returned	INSERT error thrown and false returned
27	Reading List Service	call removeLinkFromList on row in DB	row removed from database and true returned	row removed from database and true returned
28	Reading List Service	call removeLinkFromList on row not in DB	true returned	true returned
29	Reading List Service	call clearList	remove all rows from database	all rows where id != 999 removed
29a	Reading List Service	call clearList	all rows removed from database	all rows removed from database
30	Reading List Service	call getAllLinks from table	return all rows from table	return all rows from table

Test Output 1 - Databases/storage				
Test #	**Name of unit**	**Description**	**Expected Result**	**Actual Result**
31	Bookmarking Service	init() test: Load DB for first time	Database opened and new table created	Database opened and new table created
32	Bookmarking Service	init() test: Load DB for second time	Database opened, no new table	Database opened, no new table
33	Bookmarking Service	call isTopicBookmarked on bookmark in table	true returned from function	true returned from function
34	Bookmarking Service	call isTopicBookmarked on bookmark not present in table	false returned from function	false returned from function
35	Bookmarking Service	call isTopicBookmarked with incorrect parameters	SELECT error returned	SELECT error returned
36	Bookmarking Service	bookmark Topic with correct parameters	Row inserted into database and true returned from function	Row inserted into database and true returned from function
37	Bookmarking Service	bookmark Topic with incorrect parameters	INSERT error logged and false returned	INSERT error logged and false returned
38	Bookmarking Service	call removeBookmark on bookmark in table	Row removed from table and true returned	Row removed from table and true returned
39	Bookmarking Service	call removeBookmark on bookmark not present in table	true returned from function without any rows affected	true returned from function without any rows affected
40	Bookmarking Service	call removeBookmark on bookmark with incorrect parameters	SELECT error logged and false returned	SELECT error logged and false returned
41	Bookmarking Service	call getAllBookmarks	All rows in table returned from function	All rows in table returned from function

Test Output 2 - API's				
Test #	**Name of unit**	**Description**	**Expected Result**	**Actual Result**
42	User Location API	api/users/save-user called with correct parameters	new row added to User database and response code 200 returned	new row added to User database and response code 200 returned
43	User Location API	api/users/save-user called with incorrect parameters	400 returned due to bad request	200 returned
43A	User Location API	api/users/save-user called with incorrect parameters	400 returned due to bad request	400 returned due to bad request
44	User Location API	api/users/update-user called with correct parameters	User location updated in DB and 200 returned	User location updated in DB and 200 returned
45	User Location API	api/users/update-user called with user Id not in database	400 returned due to bad request	200 (OK) returned
45a	User Location API	api/users/update-user called with user Id not in database	400 returned due to bad request	400 returned due to bad request
46	User Location API	api/users/update-user called with invalid parameters	400 returned due to bad request	400 returned due to bad request
47	User Location API	api/users/get-users-with-distance called with valid parameters	200 returned	200 returned
48	Prodigy API	call getAllTopics to retrieve list of topic name and ID	full list of topics returned and 200 status code	full list of topics returned and 200 status code
49	Prodigy API	call get-topic to retrieve full topic with correct ID	200 returned with topic object	200 returned with topic object
50	Prodigy API	call get-topic to retrieve full topic with incorrect ID	404 not found returned	404 not found returned
51	Prodigy API	call get-section to retrieve section of topic from DB	200 returned with object	200 returned with object
52	Prodigy API	call get-section to retrieve section of topic not in DB	404 not found returned	404 not found returned
53	Prodigy API	call get-section with invalid parameters	500 internal server error thrown	500 internal server error thrown
54	PUSH API	call the ionic push notification API with correct parameters	201 created returned	201 created returned
55	PUSH API	call the ionic push notification API with invalid parameters	422 returned from server	422 returned from server
56	User Service	saveUser called called with correct parameters	200 returned from function	200 returned from function

Test Output 2 - API's				
Test #	**Name of unit**	**Description**	**Expected Result**	**Actual Result**
57	User Service	saveUser called with null parameters	400 returned due to bad request	400 returned due to bad request
58	User Service	updateUser called with correct parameters	200 returned from function	200 returned from function
59	User Service	updateUser called with null parameters	400 returned from function	400 returned from function
60	User Service	updateUser called with ‘userId’ as null	400 returned from function	400 returned from function
61	CKS Service	getConditions called to retrieve all name and ids of topics	full list of topics returned as JSON array	full list of topics returned as JSON array
62	CKS Service	getTopic called to retrieve topic with valid ID	topic object returned from function	topic object returned from function
63	CKS Service	getTopic called to retrieve topic with invalid ID	no object returned from function	no object returned from function
64	CKS Service	getSection called to retrieve section of topic with valid parameters	Section object returned - topicId, topicName, sectionType, sectionHTML (JSON array of sub-sections)	Section object returned - topicId, topicName, sectionType, sectionHTML (JSON array of sub-sections)
65	CKS Service	getSection called to retrieve section of topic with null parameters	Empty object returned from function	Empty object returned from function

Test Output 3 - User Interface				
Test #	**Name of unit**	**Description**	**Expected Result**	**Actual Result**
66	Home.html	Search box selected	Input box active and native keyboard appears	Input box active and native keyboard appears
67	Home.html	Search box text input and submitted	Navigation to topics search page	Navigation to topics search page
68	Home.html	Letter selected from alphabet list	Letter background ‘activated’ and list of visible topics updated	Letter background ‘activated’ and list of visible topics updated
69	Home.html	Greyed out letter selected from alphabet list	No action	No action
70	Home.html	Page scrolled down	Page scrolls with header remaining fixed at the top	Page scrolls with header remaining fixed at the top
71	Home.html	Topic selected from list	Navigation to Topic View page	Navigation to Topic View page
72	Home.html	Menu icon selected	Side menu pops out from right hand side	Side menu pops out from right hand side
73	Menu.html	Home button selected	User navigated to home screen	User navigated to home screen
74	Menu.html	Topic Manager button selected	User navigated to Topic Manager screen	User navigated to Topic Manager screen
75	Menu.html	Reading list button selected	User navigated to Reading List screen	User navigated to Reading List screen
76	Menu.html	View DB button selected	User navigated to View DB screen	User navigated to view DB screen
77	Menu.html > in topic view	Home button selected	User navigated to home screen	User navigated to home screen
78	Menu.html > in topic view	Save Topic button selected	User presented with confirm dialogue containing input box	User presented with confirm dialogue containing input box
79	Menu.html > in topic view	Add bookmark button selected	User presented with alert dialogue with output of bookmark function	User presented with confirm dialogue containing input box
80	Topic-view.html	Summary tab selected	Summary section loaded and Summary tab ‘Active’ class applied	Summary section loaded and Summary tab ‘Active’ class applied
81	Topic-view.html	Background tab selected	Background section loaded and Background tab ‘Active’ class applied	Background section loaded and Background tab ‘Active’ class applied
82	Topic-view.html	Management tab selected	Management section loaded and Management tab ‘Active’ class applied	Management section loaded and Management tab ‘Active’ class applied

Test Output 3 - User Interface				
Test #	**Name of unit**	**Description**	**Expected Result**	**Actual Result**
83	Topic-view.html	Evidence tab selected	Evidence section loaded and Evidence tab ‘Active’ class applied	Evidence section loaded and Evidence tab ‘Active’ class applied
84	Topic-view.html	Sub-section dropdown selected	Drop down appears with sub-sections of section	Drop down appears with sub-sections of section
85	Topic-view.html	Sub-section dropdown minimised	Sub-section drop down minimises	Sub-section drop down minimises
86	Topic-view.html	Back button selected	User navigated out of topic view to previous page	User navigated out of topic view to previous page
87	Topic-view.html	Menu button selected	Side menu pops out from right hand side	Side menu pops out from right hand side
88	Topic-view.html	Page scrolled	Topic content scrolls with header remaining fixed	Topic content scrolls with header remaining fixed
89	Topic-view.html	Item from sub-section navigation menu selected	Page scrolls to sub-section	Page scrolls to sub-section
90	Topic-search.html	Back button selected	User navigated back to home page	User navigated back to home page
91	Topic-search.html	Menu button selected	Side menu pops out from right hand side	Side menu pops out from right hand side
92	Topic-search.html	Search input selected	Input box active and native keyboard appears	Input box active and native keyboard appears
93	Topic-search.html	Search input submitted	List of topics refreshed with new string	List of topics refreshed with new string
94	Topic-search.html	Topic from list selected	App navigates to Topic View page	App navigates to Topic View page
95	Topic-manager.html	Bookmarked tab selected	“Bookmarked” tab active and list of bookmarks displayed	“Bookmarked” tab active and list of bookmarks displayed
96	Topic-manager.html	Stored tab selected	“Stored” tab active and list of stored topics displayed	“Stored” tab active and list of stored topics displayed
97	Topic-manager.html	Recently viewed tab selected	“Recently Viewed” tab active and list of recently viewed topics displayed	“Recently Viewed” tab active and list of recently viewed topics displayed
98	Topic-manager.html	Menu Icon selected	Side menu pops out from right hand side	Side menu pops out from right hand side
99	Topic-manager.html	Link selected in recently viewed tab	App navigates to Topic View - specific section	App navigates to Topic View - specific section

Test Output 3 - User Interface				
Test #	**Name of unit**	**Description**	**Expected Result**	**Actual Result**
100	Topic-manager.html	Bookmark selected in bookmarked tab	App navigates to Topic View - specific section & scrolls to sub-section	App navigates to Topic View - specific section & scrolls to sub-section
101	Topic-manager.html	Bookmark delete icon selected	User given confirm dialogue & bookmark deletes	User given confirm dialogue & bookmark deletes
102	Topic-manager.html	Link selected in stored tab	App navigates to Topic View - specific section	App navigates to Topic View - specific section
103	Topic-manager.html	Topic delete selected in stored tab	User given confirm dialogue & stored topic deletes	User given confirm dialogue & stored topic deletes
104	Reading-list.html	Menu icon selected	Side menu pops out from right hand side	Side menu pops out from right hand side
105	Reading-list.html	Evidence link selected - with active connection	Link opens in native browser	Link opens in native browser
106	Reading-list.html	Evidence link selected - without active connection	User alerted that they require an active connection	User alerted that they require an active connection
107	Reading-list.html	Evidence link deleted	Evidence link removed from list	Evidence link opened in native browser and link deleted from list
108	Reading-list.html	Evidence link deleted	Evidence link removed from list and not opened	Evidence link removed from list and not opened
109	Database-test.html	Menu icon selected	Side menu pops out from right hand side	Side menu pops out from right hand side
110	Database-test.html	“Refresh cache” button selected	List of cached topics refreshed	List of cached topics refreshed
111	Database-test.html	“Add something to cache” button selected	No visible action	No visible action

Test Output 4 - Angular Functions				
Test #	**Name of unit**	**Description**	**Expected Result**	**Actual Result**
112	HomeController.js	Let the controller initialise and test its set up functionality	List of topics pulled from Prodigy API and added to array from which to display	List of topics pulled from Prodigy API and added to array from which to display
113	HomeController.js	Execute the search functionality	Set application state to 'root.search' with search Text as parameter	Set application state to 'root.search' with search Text as parameter
114	HomeController.js	Run select letter function	Set a letter as active in the homepage	Set a letter as active in the homepage
115	MenuController.js	Run broadcast to save topic function	Broadcast sent over rootScope with the message 'saveTopic'	Broadcast sent over rootScope with the message 'saveTopic'
116	MenuController.js	Run broadcast to bookmark section function	Broadcast sent over rootScope with the message 'bookmark'	Broadcast sent over rootScope with the message 'bookmark'
117	TopicController.js	run init() function	Load section specified in state params	Load section specified in state params
118	TopicController.js	Listen for 'saveTopic' broadcast	Saves current topic to the Topic Store alongside message	Saves current topic to the Topic Store alongside message
119	TopicController.js	Listen for 'bookmark' broadcast	Bookmarks the current viewpoint of the user	Bookmarks the current viewpoint of the user
120	TopicController.js	Run loadSection for a topic/section that is stored in store	Section is retrieved from store and API call is not made	Section is retrieved from store and API call is not made
121	TopicController.js	Run loadSection for a topic/section that is stored in cache	Section is retrieved from cache and API call is not made	Section is retrieved from cache and API call is not made
122	TopicController.js	Run loadSection for a topic/section that is not stored in cache or store	Section is retrieved via API call to Prodigy	Section is retrieved via API call to Prodigy
123	TopicController.js	Run 'Scroll to sub section' function to navigate to 3rd sub-section	Container element scrolls to position of sub-section	Container element scrolls to position of sub-section
124	TopicController.js	Run 'openLink' function with name: Google and Link: http://www.google.com and no internet connection	User given the option to store link in Reading List' for later	User given the option to store link in Reading List' for later
125	TopicManagerController.js	Load List 'Stored'	Store Service accessed and list of stored topics returned	Store Service accessed and list of stored topics returned

Test Output 4 - Angular Functions				
Test #	**Name of unit**	**Description**	**Expected Result**	**Actual Result**
126	TopicManagerController.js	Load List ‘Bookmarked’	Bookmark Service accessed and list of bookmarks returned	Bookmark Service accessed and list of bookmarks returned
127	TopicManagerController.js	Load List ‘Recently Viewed’	Cache Service accessed and list of recently viewed topics returned	Cache Service accessed and list of recently viewed topics returned
128	TopicManagerController.js	Run ‘setListType’ with parameter ‘Bookmarked’	List type changed to ‘Bookmarked’ and list of bookmarks retrieved	List type changed to ‘Bookmarked’ and list of bookmarks retrieved
129	TopicManagerController.js	Run ‘setListType’ with parameter ‘null’	No visible outputs	No visible outputs
130	TopicManagerController.js	Run ‘getSectionName’ with valid parameters - an entry with Section Type 1	‘- Summary’ added to end of entry name - and returned from function	‘- Summary’ added to end of entry name - and returned from function
131	TopicManagerController.js	Run ‘getSectionName’ with invalid parameters - an entry with Section Type 5	Nothing added to end of entry name - entry name returned from function	Nothing added to end of entry name - entry name returned from function
132	TopicManagerController.js	Run ‘removeTopicFromList’ with valid parameters - entry in List	Nothing returned - item removed from list	Nothing returned - item removed from list
133	TopicManagerController.js	Run ‘removeTopicFromList’ with invalid parameters - null	Error logged in console from invalid arguments	Error logged in console from invalid arguments
134	TopicSearchController.js	init() functionality executed	Controller retrieves list of topics from Prodigy API and calls search function with search text as parameter	Controller retrieves list of topics from Prodigy API and calls search function with search text as parameter
135	TopicSearchController.js	search functionality executed with appropriate search text	List of topics containing sub-string of the search text returned	List of topics containing sub-string of the search text returned
136	TopicSearchController.js	search functionality executed with null search text	No topics returned	No topics returned
137	DBTestController.js	call refresh function	List of rows retrieved from topic cache and returned	List of rows retrieved from topic cache and returned
138	DBTestController.js	call addToDb function	Call cachingService.saveContentToCache with valid but dummy data	Call cachingService.saveContentToCache with valid but dummy data

Test Output 4 - Angular Functions				
Test #	**Name of unit**	**Description**	**Expected Result**	**Actual Result**
139	ReadingListController.js	run init() function	Refresh Reading List executed	Refresh Reading List executed
140	ReadingListController.js	run refreshReadingList() function	Reading list array retrieved from the database	Reading list array retrieved from the database
141	ReadingListController.js	run ‘removeLink' function on link in array	Link removed from array	Link removed from array
142	ReadingListController.js	run ‘removeLink' function on link not in array	No visible functionality	No visible functionality
143	ReadingListController.js	run openLink function with http://www.google.com as parameter and active internet connection	Link opened in native browser	Link opened in native browser
144	ReadingListController.js	run openLink function with http://www.google.com as parameter and no active internet connection	Popup sent to inform user of lack of internet connection	Popup sent to inform user of lack of internet connection
145	App.js	Call ‘onOnline’ function	Set user connection status to true and display popup	Set user connection status to true and display popup
146	App.js	Call ‘onOffline’ function	Set user connection status to false and display popup	Set user connection status to false and display popup

Test Output 5 - Functional Testing				
Test #	**Name of feature**	**Instruction**	**Expected Result**	**Actual Result**
147	App initialisation	Load application and ensure that all initialisation functionality is executed - all databases loaded	Console window displaying the opening and initialisation of all databases and global variables	Console window displaying the opening and initialisation of all databases and global variables
148	Caching of downloaded data	Open a topic and navigate through all 4 sections. Then navigate to the 'Recently Viewed' tab in Topic Manager	All four sections cached and visible on the recently viewed screen	All four sections cached and visible on the recently viewed screen
149	Caching of downloaded data	Open a topic and navigate to the 'Management' section. Go back out of the topic view and then navigate back to the same section	The topic section is loaded without making any API calls - it is retrieved from the cache	The topic section is loaded without making any API calls - it is retrieved from the cache
150	Caching of downloaded data - optimised	Open 2 topics and navigate through all four sections. Then navigate to 'Recently Viewed'	All sections cached and visible on the recently viewed screen apart from the Summary section of the first topic (which was overwritten)	All sections cached and visible on the recently viewed screen apart from the Summary section of the first topic (which was overwritten)
151	Caching of downloaded data - optimised	Load app and open a topic. Then navigate to 'View DB' page	Cache is displayed on screen with 6 dummy records and the 'Summary' section of the topic just opened	Cache is displayed on screen with 6 dummy records and the 'Summary' section of the topic just opened
152	Saving specific topics	Storage functionality will be tested, user opens topic and selects 'Save Topic' from the side menu. Enter a note into the input box and save, then navigate to the 'Topic Manager' page	Saved topic and all 4 sub-sections viewable in 'Stored' tab of 'Topic Manager' page, alongside message input in test	Saved topic and all 4 sub-sections viewable in 'Stored' tab of 'Topic Manager' page, alongside message input in test
153	Saving specific topics	Save topic using method from 152. Select a section from the saved topic and open it	No API call is issued - topic section is retrieved from the 'Store' and displayed in Topic View page	No API call is issued - topic section is retrieved from the 'Store' and displayed in Topic View page
154	Bookmarking a section	Navigate to a topic section and scroll down. Open the side menu and select 'Add Bookmark'. Navigate to 'Topic Manager' page	Bookmark viewable in 'Bookmarked' tab in Topic Manager	Bookmark viewable in 'Bookmarked' tab in Topic Manager

Test Output 5 - Functional Testing				
Test #	**Name of feature**	**Instruction**	**Expected Result**	**Actual Result**
155	Bookmarking a section	Repeat test 154. Select a bookmark to open it	Section is loaded and opens Topic View for that section. The screen is then scrolled to the position from which the bookmark was made	Section is loaded and opens Topic View for that section. The screen is then scrolled to the position from which the bookmark was made
156	Push Notifications	Load application for first time, then log in to Ionic dashboard and send out a notification to all users on the system	Default push received on device	Default push received on device
157	Push Notifications	Load application, log in to Ionic dashboard and send out a notification to all user with a payload of ‘state’=‘root.topicSection’, ‘topicId’=‘5555’ and ‘topicName’=‘Acne’. Open the notification when it appears	Prompt on screen asking user if they wish to navigate to the topic discussed in the notification. If confirmed then navigated to topic view for that topic	Prompt on screen asking user if they wish to navigate to the topic discussed in the notification. If confirmed then navigated to topic view for that topic
158	Location based push	Initialise application for the first time. Testing: Geolocation/ updating db via API	User details (unique id, push token and geolocation) stored to hosted User database on Azure	User details (unique id, push token and geolocation) stored to hosted User database on Azure
159	Location based push	Initialise application after registration. Testing: Geolocation/ updating db via API	User details (unique id, geolocation) updated in hosted User database on Azure	User details (unique id, geolocation) updated in hosted User database on Azure
160	Location based push	Populate DB with multiple different users and coordinates, include the user from test 158. Send a GET to the API to find users in a 10km radius of (54.9855737, -1.6027238)	User push token returned in JSON array as response from query	User push token returned in JSON array as response from query
161	Location based push	Use output from test 160 and code from Figure 23 to send push notification via POSTMAN	Push notification received on device	Push notification received on device

Test Output 5 - Functional Testing				
Test #	**Name of feature**	**Instruction**	**Expected Result**	**Actual Result**
162	Evidence reading list	Navigate to ‘Evidence’ section of a Topic. Turn off internet connection on device. Select a number of links and allow them to store to reading list. Navigate to reading list page and select one of the links	Toastr notification appears informing the user that they require a connection to access the resource	Toastr notification appears informing the user that they require a connection to access the resource
163	Evidence reading list	Navigate to ‘Evidence’ section of a Topic. Select a number of links and save them to reading list. Navigate to reading list page and select one of the links	Link opens in phones native browser	Link opens in phones native browser

Test Output 6 - End-to-end tests			
Test #	**Test Details**	**Expected Result**	**Actual Result**
164	- User downloads application - User loads application - User searches for ‘Parkinson’ - User opens Parkinson topic and navigates to Management section - User navigates to ‘Managing end-stage Parkinson’s via dropdown’ - User selects ‘Add Bookmark’ from the side menu - User closes app - User opens app and goes into bookmarked section - User selects bookmark	Parkinson's disease loaded from the cache (no API req) and view scrolled to bookmarked position	Parkinson's disease loaded from the cache (no API req) and view scrolled to bookmarked position
165	- User loads application - User opens ‘Acne’ topic - User navigates to ‘Evidence’ section - User selects ‘Save Topic’ from the side menu - User enters ‘Keep this for visit to Jonny’ in the input box and selects ‘Save’ - User closes application and kills its process - User loses internet connection - User navigates to ‘Topic Manager’ and selects Acne - Management from Store list	Application pulls ‘Acne - Management’ from Topic Store and loads in topic view to display contents	Application pulls ‘Acne - Management’ from Topic Store and loads in topic view to display contents
166	- User loads application - User opens ‘Parkinson’s’ topic and navigates through the ‘Summary’, ‘Background’, ‘Management’ and ‘Evidence’ sections - User loses internet connection - User attempts to navigate from ‘Evidence’ back to ‘Summary’	Application loads ‘Summary’ section from the cache and displays it on screen	Application loads ‘Summary’ section from the cache and displays it on screen
167	- User loads application - User opens ‘Parkinson's' topic - User selects ‘Save Topic’ from the side menu and completes the process - User navigates through 4 other topics, 7 sections in total - User closes app and kills process - User loses internet connection - User opens app and opens ‘Parkinson’s - Evidence’ from the store - User attempts to open one of the evidence links that appear - User closes application and kills process - User regains internet connection - User opens application and navigates to ‘Reading List’ - User selects reading list item	Link from reading list opened in native browser of mobile	Link from reading list opened in native browser of mobile

Test Output 6 - End-to-end tests			
Test #	**Test Details**	**Expected Result**	**Actual Result**
168	- User loads application - User searches for ‘Parkinson’s’ topic and opens it, navigating to Management and Evidence sections - User goes back and searches for ‘Acne’ topic and loads ‘Background’ and ‘Management’ sections - User navigates to ‘Recently Viewed’ tab in ‘Topic Manager’ and selects and ‘Evidence’ section of ‘Parkinson’s’	Topic loaded from cache and does not require an API call	Topic loaded from cache and does not require an API call
169	- User downloads application - User loads application - User uses application to gather information on ‘Acne’ - User closes application - Tester navigates to Ionic dashboard and generates a push notification with a message directing users to a new topic which has been released, and sends it	User receives notification on their device detailing the topic release. Once notification is opened the user is directed to topic in application.	User receives notification on their device detailing the topic release. Once notification is opened the user is directed to topic in application.
170	- User is based in Newcastle, UK - User downloads application - User loads application - User uses application to gather information on ‘Acne’ - User closes application - Outbreak of ‘Lyme disease’ in Sunderland, UK - Push Notifications API queried to find application users in a 20 mile radius - Push Notification sent with details on ‘Lyme disease’ outbreak to all users retrieved from previous step	User receives notification on their device instructing them to open the topic. Once notification is opened the user is directed to topic in application.	User receives notification on their device instructing them to open the topic. Once notification is opened the user is directed to topic in application.
171	- User is based in Newcastle, UK - User downloads application - User loads application - User uses application to gather information on ‘Acne’ - User closes application - User moves to London, UK - Outbreak of ‘Lyme disease’ in Sunderland, UK - Push Notifications API queried to find application users in a 20 mile radius - Push Notification sent with details on ‘Lyme disease’ outbreak to all users retrieved from previous step	No notification is sent to users device.	No notification is sent to users device.

Test Output 6 - End-to-end tests			
Test #	**Test Details**	**Expected Result**	**Actual Result**
172	- User loads application - User uses application to store ‘Parkinson’s’ and ‘Acne’ topics - User browses multiple other topics - User closes application - User loses internet connection - User loads application - User navigates to ‘Acne’ topic	‘Summary’ section of topic loaded from the ‘Store’, without an API call being made	‘Summary’ section of topic loaded from the ‘Store’, without an API call being made
173	- User downloads application - User loads application - User searches for ‘Parkinson’ - User opens Parkinson topic and navigates to Management section - User navigates to ‘Managing end-stage Parkinson’s via dropdown’ - User selects ‘Add Bookmark’ from the side menu - User browses 8 other topics in the application - User closes app - User opens app and goes into bookmarked section - User selects bookmark	API call executed to retrieve content from Prodigy, topic-view containing section is displayed and screen is scrolled to position of bookmark. Topic loaded is now stored in cache	API call executed to retrieve content from Prodigy, topic-view containing section is displayed and screen is scrolled to position of bookmark. Topic loaded is now stored in cache