GC02 Team 7 Project Report
MINAP Mobile Application

Amer Kamil Zainal Abidin, Ke Wei, Marco Corrales Estrada

January 27, 2014

Contents

1 Background

1.1 MINAP/NICOR

The Myocardial Ischaemia National Audit Project (MINAP) investigates the management of heart attacks. Participating hospitals and ambulance services are provided with a record of their quality of management. These records are then compared to national and international clinical standards. The National Institute of Cardiovascular Outcomes Research (NICOR) is a partnership of clinicians, IT experts, analysts, academics and managers. Its mission is to "provide information to improve heart disease patients' quality of care and outcomes". Using different audits, MINAP being one of them, NICOR aims to provide the NHS, government and regulatory bodies real world data to be used in patient care [1][2].

1.2 Web Application

Participating hospitals are requested to enter all patients with suspected heart attack into MINAP's central database system. Data are collected by nurses and other clinical audit staff and entered in a dedicated web application. This web application collects all patient data, from demographics to diagnosis and release information. As it stands, the web application contains a 130 field dataset that is updated every two years (latest revision, June 2013); it is currently only available to the web browser Internet Explorer versions 9 or below. The data is then validated against a set of rules which include, but are not limited to, value range checks, date range checks, data type checks and encryption rules. Once completed, records are stored on MINAP's Domino server[3].

1.3 Client

The project's primary point of contact is Lucia Gavalova, Project Manager for the MINAP project, who we will refer to as the 'client' throughout this document (unless stated otherwise). Sue Manuel, MINAP's web app developer has also been a key stakeholder throughout the course of this project.

[1] MINAP Reports

[2] NICOR Website

[3] IBM Domino is used for enterprise collaboration, and is used primarily for its Lotus Notes capabilities in this instance

2 Requirements

The following requirements have been formulated from a series of client meetings and e-mail communication, available in brief in appendix B.

2.1 Overview

Patient data record creation is currently restricted to when the nurses and other clinical audit staff are sitting in front of a computer. This would often be quite some time after a patient has been identified as being eligible for being entered into the MINAP dataset. A reduced-function mobile solution of the current web app was proposed by our client as a means of starting a patient record in MINAP's data set, allowing for prospective data collection. Completion of the record is expected to be done later on using the web app (using a full sized keyboard and screen).

2.2 Use Cases

The system's use case we are concerned with for our app can be modelled by the following table. Although the user-facing view of the system is fairly simple, most of the application logic lies in the interdependency and validation checks of the values entered by the user.

A nurse or clinical audit staff wants to create a record
She first needs to login to the server with her credentials
She then selects an option to create a new record
She then is brought to a page to enter in her initial diagnosis
If she doesn't understand a field, she can click on a button for help
She then enters the patient details into a data input field
If the data entered is invalid, a pop-up will appear, prompting her for correction
She can then navigate through each page using 'next' buttons, or through a navigation map
Throughout this process, she will save the data to the server by clicking a save button
Once complete, she logs out of the system

2.3 Functional Requirements

2.3.1 User Authentication and Security

As with the web application, the user must login to the central server prior to using the mobile app for data input. For security purposes, the user authentication details should not be held on the device beyond the initial authentication phase (or instead, use tokens), and all sensitive data retrieved from the server and created on the device should be destroyed at the end of a user session.

2.3.2 Data Collection

In order to create a patient record, the app needs to provide the user with forms for data entry. Depending on the data fields, certain format restrictions are imposed, for example: date and time related fields are in the dd/mm/yy hh:mm format; combo boxes; and numeric only data fields. Data fields should be divided into sections (pages) based on the current web app structure for familiarity. Data entered in by the user should be stored locally for the duration of the session, and ideally sent to a server once validated[4].

2.3.3 Data Validation

Validation currently occurs on the web app automatically at the point of exiting a data input object, page, or on saving. All 130 fields within the dataset must pass at least one validation rule. Certain fields will then either trigger additional validation rules involving other fields, default values to other fields, and/or open new sections of data collection to be filled out. Our app should follow this behaviour (with a subset of data) to ensure data integrity.

2.3.4 Data Submission

Once all mandatory data is entered and all validations rules have been passed, the user will be able to store the local record to a SQLite database. Due to time constraints, the team was unable to test calls to a web service of our own design as agreed with the client (20/11/13 Meeting). However, the SQLite database should prove to be flexible if used in future implementations as it is self-contained and follows the dataset's specifications.

[4]See Data Submission

2.3.5 Mobile Usability

Since the app will be running on smart phones (our target platform), factors such as the screen size, touch navigation, and use of an on-screen keyboard have to be taken into account during the user interface design phase. Certain features in the current web app (e.g. navigation and data submission) are not intuitive and need to be rethought for mobile use, but should still retain the familiarity of the existing interface.

2.4 Architectural Requirements

The core purpose of our mobile application is to provide proof of concept for a solution what would offer a starting point for record creation. To that end, we came to a mutual agreement with the client that the original dataset be reduced from 130 fields. In exchange for reducing the volume of data, certain architectural requirements were requested so as to create a framework for future work to build upon (e.g. porting the application to other platforms).

2.4.1 Web Services

Although we have reached an agreement that we will not actually need to touch the live Domino Server, there are certain functions that our application will have to implement to ensure future compatibility with MINAP's technology stack (IBM Lotus Notes on Domino Server 8.5.3). Processes such as how data will be retrieved from the server and authentication processes should be modelled into the system as stubs that can be easily implemented at a later stage in the development of the app. Further research into the topic of communication with the server from other services was also deemed useful, since the client was unable to give a clear answer as to how this could be done.

3 Target Mobile Platform

3.1 Client Preference

Our client was unable to give us a clear answer as to which mobile platform she wanted us to develop for during the early stages of development. Due to this, the team decided to choose the Android mobile platform, primarily due to reasons of security and programming a native app. However, to help better inform future decisions, we released a survey to MINAP users with the help of our client.

3.2 User Survey

We have designed a user device survey which the client has agreed to running over the period of a few months, with a suggested completion date within 3 weeks of rolling out. The primary purpose of pushing out this survey was to better inform decisions of what secondary platforms should be supported to increase mobile app uptake by hospital staff. This was not meant to affect which platform we would develop for, but was instead meant to be used as research to direct future work. More details regarding the contents of the device survey can be found in appendix B.

3.2.1 Survey Results

Perhaps unsurprisingly, the majority (60.9%) of the 82 user responses replied "Apple" when asked the brand of their smartphone/tablet, with only 33.3% responding to "Android" when asked what their mobile operating system was.

Despite the bias towards iOS, 72.4% of respondents answered "No" to using their smartphone or tablet during work hours. User comments also indicated that they were unwilling to use their own devices for work purposes, with comments indicating that they would be more likely to use work-related mobile apps with a device provided by their institution.

Security was also a key issue that was discussed in user comments - something that we were aware of in the initial stages of our requirements gathering. This highlights the issue that we are unable to deploy our app in the near future, as a proper security audit would have to be performed before any actual patient data is used with the app.

3.2.2 Discussion

Unfortunately, 82 responses are not enough to have a thorough statistical analysis of user preferences, but the survey has given us a good sense of user concerns with regards to patient data entry on mobile devices.

4 User Interface

4.1 Preliminary UI Design

As part of ensuring our understanding of the requirements were in line with the client's expectations, we sketched out a preliminary user interface (UI) and presented our ideas to our client. The order in which each page was presented is indicated by red arrows. A summary of the key points relating to UI during our conversation are included below.

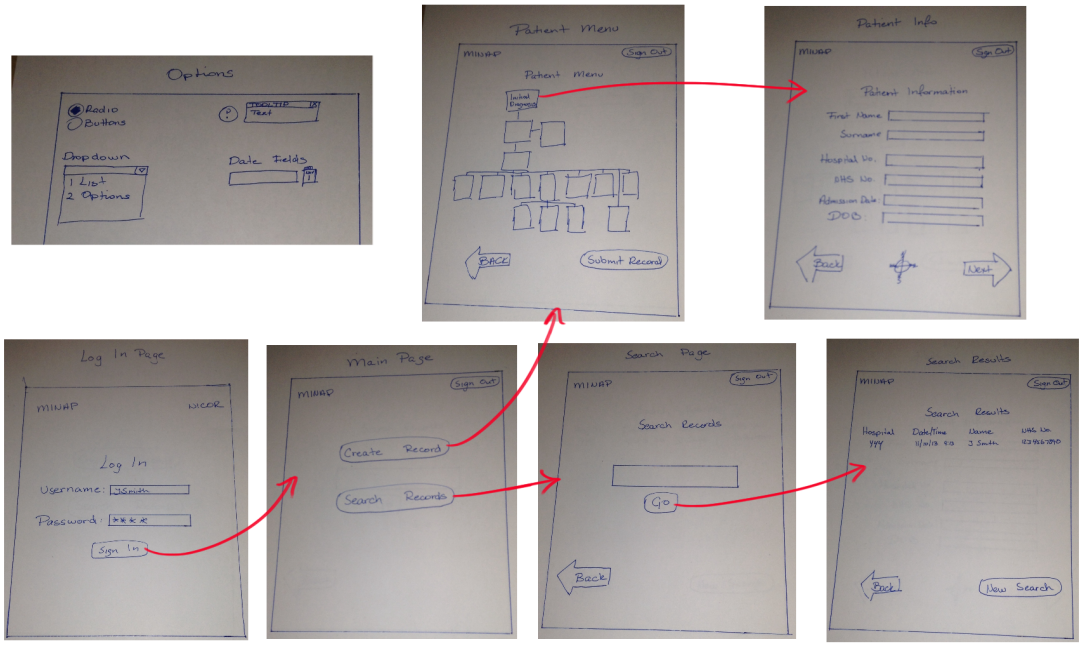


Figure 1: Preliminary UI sketches

4.1.1 Initial User Interface Feedback

Client response:

- happy with proposal of similar design to web app
- request for an application tutorial for "non-techie" users
- navigation menu: a simplified version of the map could be used for mobile

- an 'auto-save' feature would be useful ("like Microsoft Word") in case of battery loss, device breakage, or more pressing matters to attend to (i.e. emergency)

4.1.2 Commentary and Response

We have accepted the feedback from the client as being in line with our initial requirements. However, the request for an auto-save feature in the app would violate the privacy requirements of having volatile local data. A way to mitigate this would be to save the record to a 'draft' database when auto-saving (i.e. "submitting" a record without performing validation-on-send), but is a low-priority feature at this point.

4.1.3 UI Tweaking

After taking these comments on board, we have more formally modelled the user's flow of use through the app, mapping each UI element to Java's graphical implementation classes. It is worth noting that the UI is still in its initial stages, pending further adaptation for mobile use.

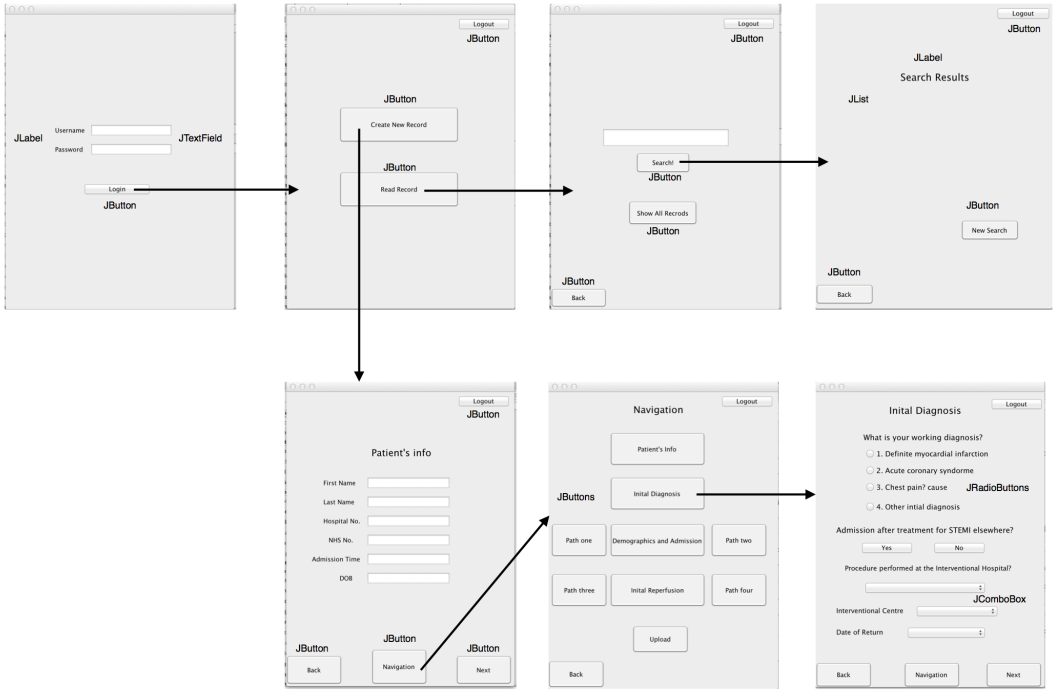


Figure 2: Java GUI

4.2 Android UI Design

Shortly after our first milestone, the team managed to nail down Android as our target platform. With a specific mobile platform in mind, we could now use design elements native to the mobile operating system, that existing users should already be familiar with. This design process involved heavy research into the Android design guidelines, and several prototypes, before coming to the final design that was implemented in our app. The entire design process has been documented extensively, and can be found on the team's development blog[5].

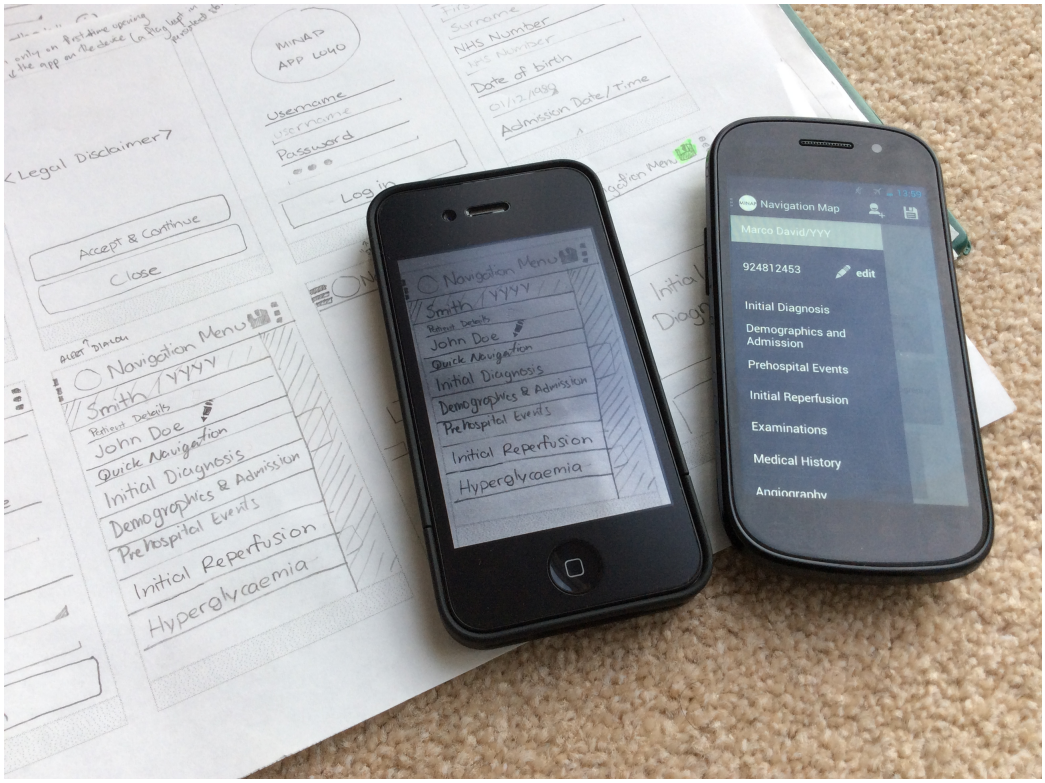


Figure 3: Snapshot of the Android UI design output

As we had agreed to provide our client with a proof-of-concept app for demonstration purposes, the UI took up a significant portion of development time, and it had the most direct client input throughout the project duration.

[5] http://akz08.github.io/minap-mobile/

5 Software Design

5.1 Proposed System Design

As part of our initial requirements gathering process, we designed a proposed system in which our mobile app could possibly reside in (see whiteboard sketches in appendix C). This infrastructure should contain all the necessary components that would allow the app to function as part of the MINAP data entry ecosystem, if fully implemented. It is however important to note that only the items marked red in fig. 4 are actually covered by our agreed specifications with the client and have been implemented.

5.1.1 Overview

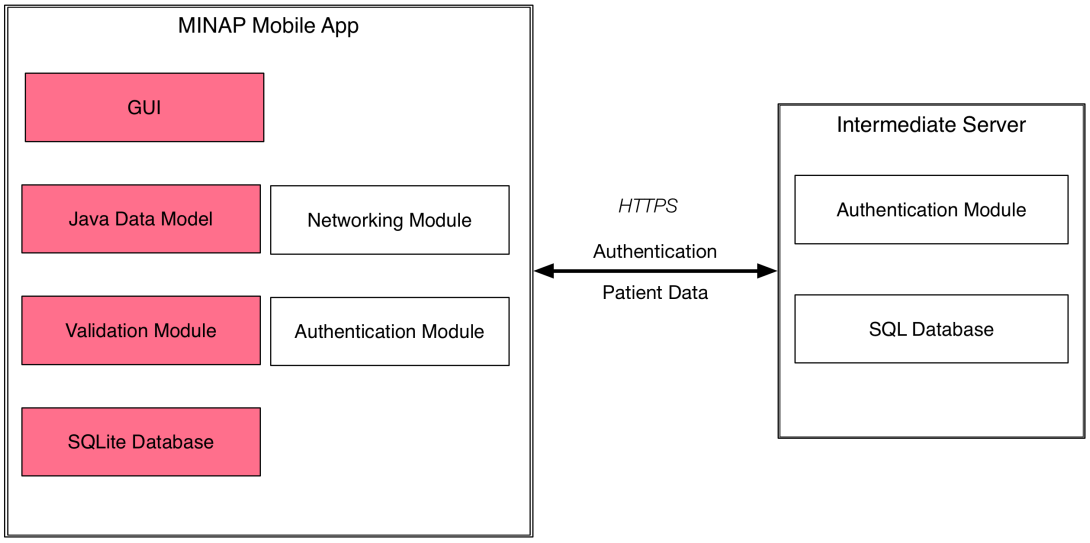


Figure 4: Sketch of proposed system design

We have modelled our system as consisting of a mobile app and intermediate server. This system could theoretically be developed on its own, without having to deal with any current MINAP servers, during the development process. This helps defer the need to communicate with any Domino server, until a reasonably complete has been built.

5.1.2 Intermediate Server

The use of an intermediate server removes the necessity to deal with the processing overhead of using protocols such as SOAP and WSDL with a mobile app. Instead,

communication with the actual MINAP Domino server can be done via the more powerful intermediate server, allowing the app to use a more lightweight RESTful service for communication. As a minimum, this server should implement a secure module for authentication, and use a standard SQL database for data storage. Processing of data stored in the SQL database to an IBM Domino-compatible data format could be done in batches to reduce processing load.

5.1.3 Network Communication

To retain the same level of security as the MINAP web app, authentication and patient data should be transmitted over HTTPS.

5.1.4 Mobile App

To enable our app to function with our intermediate server, it would require the existence of a networking and authentication module. If designed correctly, these two modules could be developed on top of the core mobile app (indicated in red in fig. 4). The core app should primarily consist of a solid data model foundation (as discussed in section 5.3), with a GUI layer to interact with it.

5.2 Designing for Android

The Android base application mostly follows a typical Android project structure, making use of elements such as Intents and Fragments - but with a few key differences:

- Throughout the process of creating the app, the book *Android Programming: The Big Nerd Ranch Guide*[1] was used extensively as learning material and reference. The book advocated the use of Fragments everywhere to allow for flexibility in the overall screen design. For our particular specifications, this flexibility would make development for tablets much easier in the near future, and thus it was adopted.

- A typical Android project is divided into 3 main parts: the model, Activities, and screen layout XML files. For our mobile app, there was a considerable amount of screens to model, with associated logic for hiding and showing input boxes. This would cause the underlying Activities (in our case, Fragments) to easily grow in size, making readability and maintainability difficult as the project develops. To avoid this, research was done to find a design pattern that would best fit our needs to maintain a decent level of organisation within our programming workspace. A variation of MVC was finally

chosen at the most appropriate candidate[2], separating the code into the model, Fragments, and Views.

- XML layout files were extensively used to design all the base data input views, as well as the data input views that are hidden by MINAP's conditional logic (based on the web app). Despite the large number of layout files, a naming convention has been followed to allow these layouts to be distinguished from each other.

Following the key resources mentioned and understanding the above design decisions, a developer with prior experience with Android should be able to understand and continue the development of the base app with considerable ease.

5.3 Data Model & Validation

In order to capture the functional requirements previously explained, three major steps were to be completed for the inner workings of the mobile application.

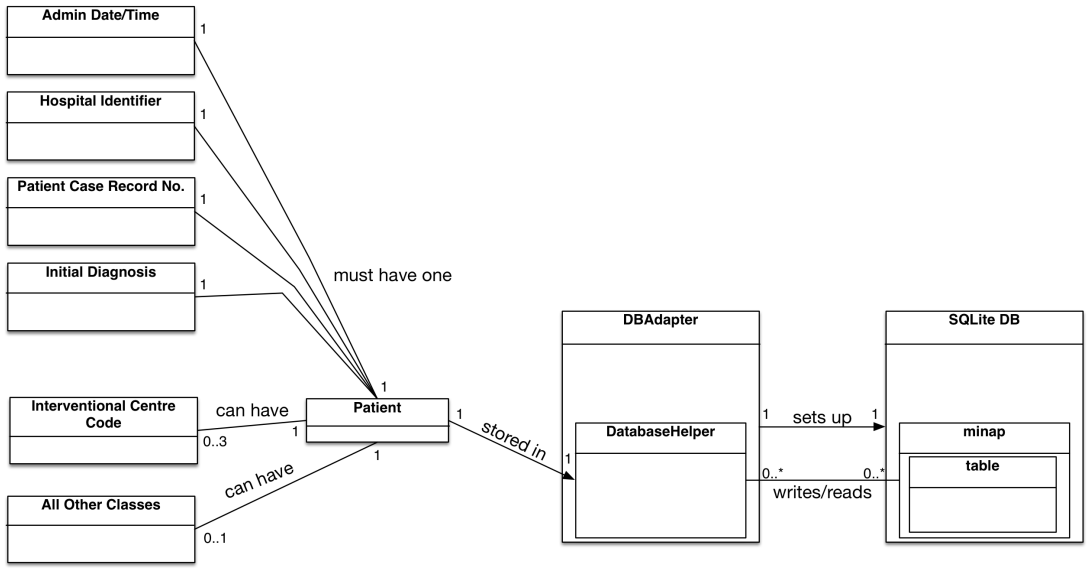


Figure 5: Database workflow class diagram

5.3.1 Designing the patient singleton

The Patient class would, in essence, capture the entirety of the dataset for this project. For security reasons, the team agreed with the client (5/12/13 Meeting)

that the application should not hold more than one record per session. As such, the patient class was deemed fit to be designed as a Singleton. A call to a Patient object will return either an existent instance of the class or create one if such instance does not exist. Furthermore, the clone method of the Patient class has been overridden to throw a CloneNotSupported exception in order prevent stray clones from being created. All fields from the dataset are then declared as part of the patient class.

5.3.2 Designing and implementing field classes and the Value superclass

As previously mentioned, all fields must pass at least one validation rule before entering long term storage, either to the local database or MINAP's server. Although some fields shared similar validation rules, such as date fields needing a certain display format, the usage of a super class was minimized due to the volatility of the dataset itself. Due to this, our superclass, Value, only encapsulates variables that the team decided may not change as frequently as others; namely the field number within the dataset and the description of said field. These two values are used as tooltips on both the existing web application as well as our GUI. All field classes have been designed to be independent of each other.

Each class contains its own data validation rules. The MINAP dataset and sanity checks files provided by the client proved essential in deciding what data types to use for each field. Length check validations were handled by declaring a private short variable that would hold said length value. Date and time check validations were slightly trickier as they required the use of Date methods as well as importing the SimpleDateFormat library to better implement comparisons. Alphanumeric checks would need to be compared against regular expressions.

The getter methods would take an appropriate typed value as a parameter and return boolean true if said parameter passed the field's validation rules. An if statement was the most appropriate structure to check a parameter's value. Strings would need to clear regular expressions; Dates, Integers and Doubles have ranges they must fall within; to capture the drop down list or radio button behavior seen in the web application, classes with switch statements were implemented that both checked the selection for validity and set the fields "Long Code" variable to the appropriate value.

Setter methods, for reasons that will become clearer in the next session, would return the data necessary as a String in the needed display format.

5.3.3 Designing and implementing the DB

The SQLite database used in this project uses no third party libraries. It was clear that the less dependencies present in our code, the easier it may be to fully implement the dataset and network capabilities further down the road. As this approach was chosen, the implementation of the database was more time consuming than expected. The final result, however, was well worth it.

The database itself consists of only one table, one record long. The DBAdapter class starts by declaring the field names to be used, each logical "page" declaration finished by declaring an array of strings that can hold all column names to a particular page. The concatenateArrays method can then declare a larger array that will hold all columns that were previously declared, this method can and indeed is, then be used when a SELECT * statement needs to be used on the database. The database is then created by assigning it a name and a tag that will be used when updating the schema. After assigning the table a name, a String is declared that holds the SQL statement to create the database.

All of the database logic is held from the declaration of the DatabaseHelper class down. Methods to open and close the database are followed by methods that insert information to a set page, updating information in said pages. Special attention must be given to the insertPatient method as it represents the Patient Info page on the UI and is the only way to actually insert one more record onto the database. The DBAdapter class ends with the implementation of a method that will, due to the restriction of one record per session, destroy all data; and a method that will select all data from the record.

The columns on our patient table have all been set to be Text. This decision facilitates writing from the field classes getter methods without having to worry about server format compatibility. This decision, as well as the decision to only hold one record, can be easily expanded and corrected if necessary.

6 Testing & Analytics

6.1 Automated Testing

As preparation for integrating the data model into the base Android app, MonkeyTalk[6] was included in the app to allow for automated testing of features. This allowed us to create simple scripts that would simulate touch events on our app, removing the need to perform the same tasks repeatedly to check for any bugs that may occur during the integration process. A side effect of doing this, meant that our Android project had to be converted to AspectJ - something that must be noted when trying to compile the app from the source code.

Unfortunately, due to time and resource constraints, the data model was not fully connected to the base Android app, and so MonkeyTalk was only used for very basic tests.

6.2 User Testing

As part of keeping our client informed of our progress, TestFlight[7] was used to deploy new Android app builds, starting from the first meeting of the year (13/1/14). This gave out client an easy UI to download new builds straight to her Android smartphone. To help ease the process, a pre-existing account was installed on our client's smartphone (with prior consent). New builds were uploaded every 3-4 days, and an e-mail was sent out to the client to notify her to download the new app. A final feedback document was produced by our client based off these app builds (see section 8).

Our app was also casually tested by our peers. The initial lengthy explanation that was necessary to get them to understand how our app should be used indicates the possibility that it may be reasonable to have some expectation of prior experience with the MINAP dataset as a minimum requirement for users of our app.

6.3 Analytics

Although our mobile app is unlikely to be used by actual end-users in the near future, TestFlight enabled us to place checkpoints in various places in our app, flagging up if the tester triggers them. This allowed us to see how thoroughly the app was tested, and what features were least used (or not even discovered) through TestFlight's web app dashboard.

[6] `https://www.cloudmonkeymobile.com/monkeytalk`

[7] `https://www.testflightapp.com/`

Crashlytics was also added to our app for future crash reporting, but was unable to be fully utilised with the current userbase of one (our client).

7 Application Limitations

While this application does fulfil our client's requests, it is far from a finished product that could be implemented in the workplace. In order to provide a complete solution, our application would need to implement all missing fields from the dataset. These fields would then need to be reflected on the UI. However, the most obvious limitation of our solution is, without a doubt, the lack of network connectivity.

Our main priority during the second half of this project was to produce something tangible. To have something that our client could hold in her hands and, not to put it lightly, play with. To that end, and with her consent, networking research was put on the back burner in favour of delivering a functioning application.

Still, the need for a further study onto the networking technologies used at MINAP was necessary. If not for our project, for future hands to grapple with.

7.1 Client-Server Communication

As mentioned in the 20/11/13 Meeting, the team confirmed that making our app communicate with a Domino Server is not a key priority, and that spinning off another web server (e.g. Apache, + MSFT SQL Server) with a similar capabilities will satisfy the requirements. Therefore finding compatibility solutions between Java/Android and the Domino Server was reassigned as a research topic.

The aim of the research was to understand the domino web server communication protocol, to some degree, as it could help explore the possibility of making our app cross-platform and future-proof. With little understanding on the topic, I started with searching through the user guide for Domino 8.5 (IBM.COM) and found out that Domino supports Web services as defined in the W3C documents Simple Object Access Protocol (SOAP) 1.1 and Web Services Description Language (WSDL) 1.1.

SOAP (Simple Object Access Protocol) is a protocol specification for exchanging structured information in the implementation of Web Services within computer networks. It relies on the XML Information Set for its message format, and usually makes use of other Application Layer protocols, most notably Hypertext Transfer Protocol (HTTP) or Simple Mail Transfer Protocol (SMTP), for message negotiation and transmission.

The official SOAP Version 1.1 specification document has been added to resources used (W3.com). It provides not only background informations of SOAP, but also extensive knowledge how a SOAP message is constructed, how to use SOAP in HTTP, and examples of SOAP messages.

As an example of how SOAP procedures can be used, a SOAP message could be sent to a web service-enabled site; such as a real-estate price database, with the

parameters needed for a search. The site would then return an XML-formatted document with the resulting data, e.g., prices, location, features. With the data being returned in a standardized machine-parsable format, it can then be integrated directly into a third-party web site or application.

A SOAP message is simply an XML file which contains the following elements:

- An Envelope element that identifies the XML document as a SOAP message
- A Header element that contains header information
- A Body element that contains call and response information
- A Fault element containing errors and status information

A basic structure of a SOAP message is offered (w3schools.com):

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Header>
...
</soap:Header>

<soap:Body>
...
  <soap:Fault>
  ...
  </soap:Fault>
</soap:Body>

</soap:Envelope>
```

Figure 6: Basic structure of a SOAP message [3]

Examples of SOAP Messages (w3schools.com): In this example, a "GetStockPrice" was sent to the test server (`http://www.example.org/stock`). As a response, the "Price" value was returned.

In terms of the real world implementation of the communication between Domino and Android device, many sources were found online, however few of them were actually helpful to this research. In the end, an interesting web page called How to Call Web Service in Android Using SOAP was found (c-sharpcorner.com). From what I have gathered so far, it seems Android does not provide any sort of SOAP library; according to the web page, our best bet is kSOAP 2. It is a SOAP

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

  <soap:Body xmlns:m="http://www.example.org/stock">
    <m:GetStockPrice>
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>

</soap:Envelope>
```

Figure 7: Example HTTP Request

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

  <soap:Body xmlns:m="http://www.example.org/stock">
    <m:GetStockPriceResponse>
      <m:Price>34.5</m:Price>
    </m:GetStockPriceResponse>
  </soap:Body>

</soap:Envelope>
```

Figure 8: Example HTTP Response

web service client library for constrained Java environments such as Applets or J2ME applications (CLDC / CDC / MIDP). In the webpage that was mentioned before, a connection between an Android device and a non-Domino web service was established through SOAP. Since the Domino web server supports SOAP, with modifications it should be possible to establish a connection between an Android device and the Domino web server. In future research, this should be attempted.

8 Final Client Feedback

As part of our beta testing with out client using TestFlight, we requested our client to provide us with written feedback on our final app (and earlier builds). This document serves as both a "sign off" sheet and a last-minute feature wish-list (our client sent the document on 23/01/14). This document is included in an unedited state (with the exception of formatting in LaTeX) in the section below.

8.1 Unedited Client Feedback Document

Comments on MINAP App

- The app allowed me to save the record even though initial diagnosis and hospital number sections have not been completed. I think in terms of design the compulsory 4 fields requirement must be built in to prevent occurrence of essential missing data.
- Initial reperfusion treatment check if agreed to complete limited number of fields and not the entire section, although that makes sense.
- Date and time formats I have no problem with using the widget as presented as long as on transmission on to the servers, the information appears in the correct format.
- All buttons are easy to use. Although the side panel is not visible it is very easily picked up, yet not in an annoying sort of way.
- Save button on every page is a great idea
- There is no close button for when I have completed one record and a nurse is moving onto another patient
- In principle, it would be helpful to address in your report whether data would be accessible for editing even if for a limited period of time, e.g. 24 hours. It is possible that a user has made a mistake and wishes to correct it while he/she remembered it using the mobile device
- I wonder whether hospital fields needs to be on the first, patient page. After all the hospital is selected by default based on the user ID. If there is a way of replacing it with the initial diagnosis with drop down menu options, then all the core fields are on one page. They could be marked in a different colour so that the user is aware that without this information there is no point to go any further as a record will not be saved. This is not negotiable part of

saving a MINAP record. Then the other parts of the initial diagnosis e.g. whether patient returned after treatment for STEMI elsewhere or if ACS is selected, whether it is a high risk or not could appear on the same page as initial diagnosis does at present.

- It took me a little while to figure out how to go back to the patient details i.e. via edit button but got there in the end.

Overall very good design indeed, easy to navigate, with logical order and intuitive for someone as impatient as me. Really well done. The font works really well. Having tried the app I am even more convinced that prospective data collection is not such a mad idea as it seemed at the beginning.

8.2 Discussion

Most of the problems with the mobile app are due to the fact that the data model has yet to be hooked up to the core Android app. However, the comment regarding the creation of a new patient record from an existing session was flagged as a key item to be dealt with, and a button for patient record creation was added to the navigation map home screen. The issue regarding the placement of Initial Diagnosis together with Patient Details, and the difficulty of finding out how to edit the patient details are also important from a usability perspective, but we were unable to rectify this issue with the given time.

9 Future Plans

9.1 Use of XML/Portable Data Files

The application rules are currently defined by a series of data fields stored across two excel spreadsheets. From what we understand (20/11/13 Meeting), these rules are implemented in client-side Javascript for the current web app. Ideally, these rules would need to be stored in a standard machine-readable format which can be easily translated to validation code in any programming language. This would also serve as a more structured form of the current documentation available for the MINAP dataset validation rules.

9.2 UI Tweaks and The Tablet

As mentioned in the discussion of our final client feedback document (section 8), the current UI requires some adjustments to the layout and underlying structure to correct some usability issues that are integral to how the MINAP dataset works as a system. These should be addressed before attempting any other redesign when the opportunity arises.

Additionally, a tablet UI should be explored in the near future if mobile data entry is being seriously considered. As most of the business logic has already been implemented, creating a tablet layout would consist mainly of creating new XML layouts for Android[8].

[8] `http://akz08.github.io/minap-mobile/blog/2014/01/13/trying-to-be-responsive/`

10 Project Management

10.1 Work Packages

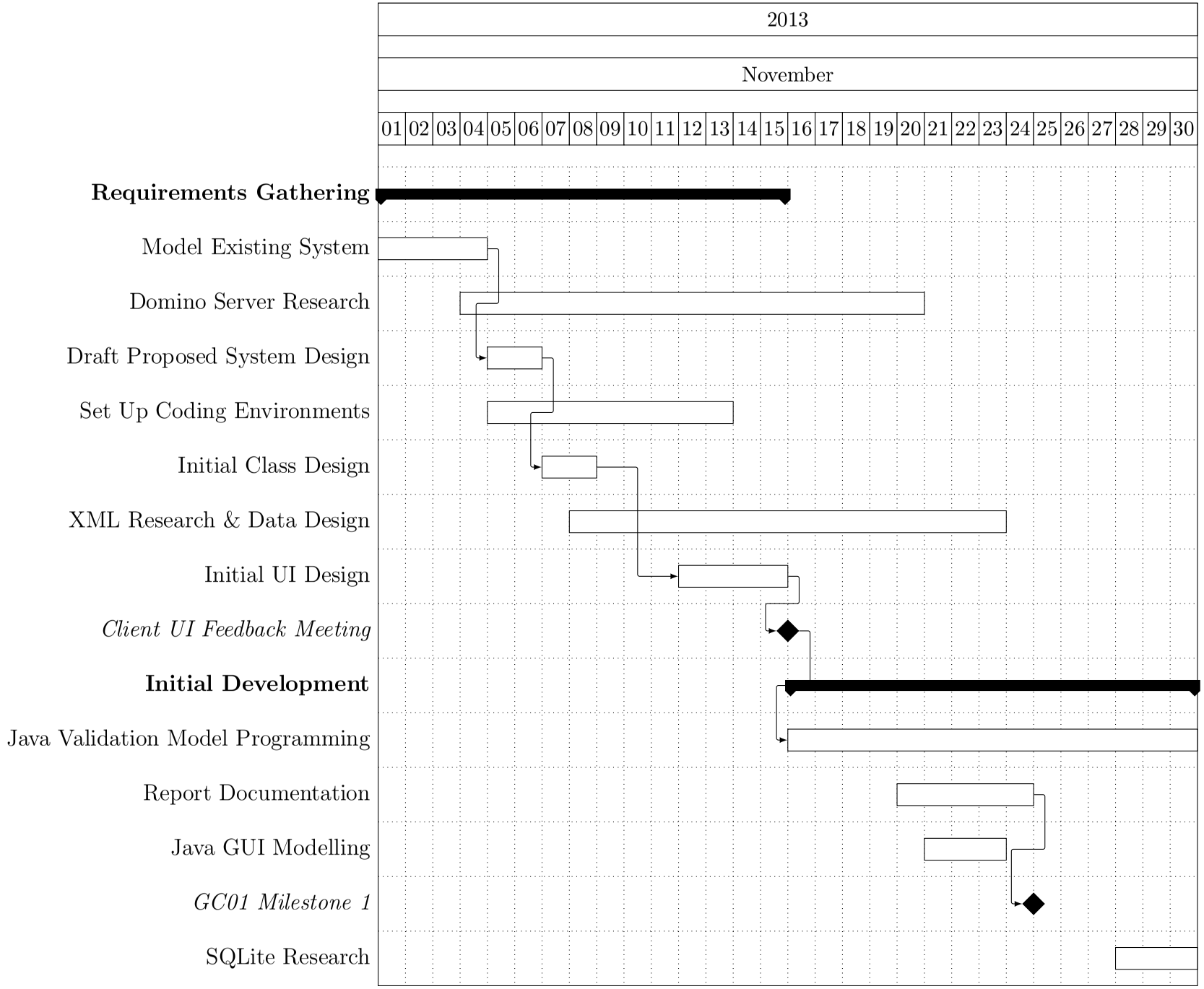
Throughout the duration of the project, work packages were assigned to each team member informally, and documented on Trello. A list of key work packages for our second stage of development are listed below, with the time and resources that were necessary to complete them:

- MINAP Data Modelling (2 people, 21 days)
- Android UI Skeleton (1 person, 35 days)
- SQLite Design (1 person, 33 days)

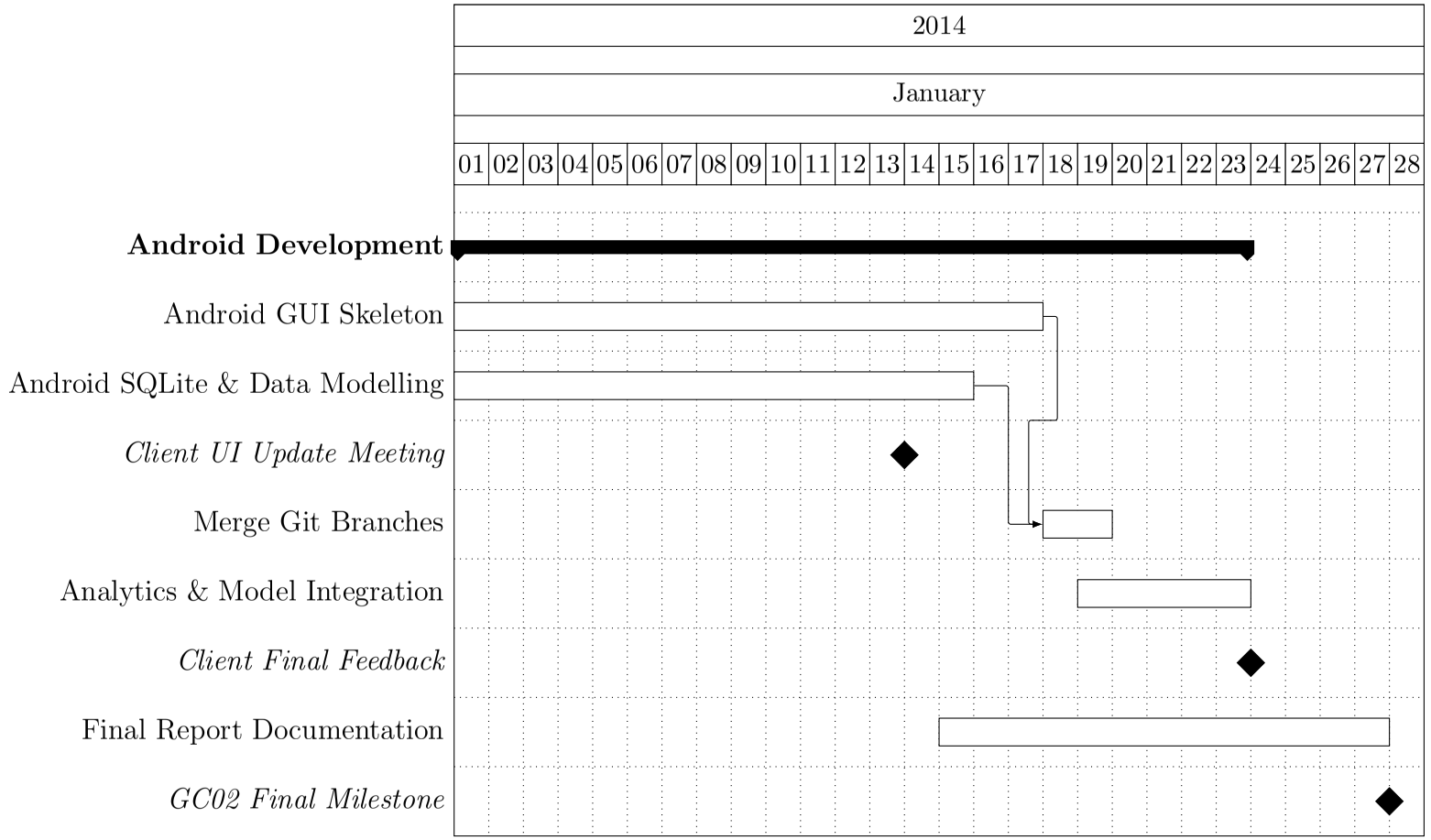
These core work packages were essential to the development of the MINAP mobile app, with delays in their completion causing a knock-on effect on all other work. It must be noted however that the delay in MINAP data modelling is primarily due to our lack of experience with the MINAP dataset, and the creation of the Android UI Skeleton and SQLite design due to our inexperience with handling those systems.

One core work package that was unable to be completed was the integration between the Android core application and the SQLite model. We estimate that the completion of the work package would be approximately 1 week for someone with previous knowledge of the Android project code.

10.2 Gantt Chart




2013
December
01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Initial Development
SQLite Research
Java Validation Model Programming
Android UI Research & Development
Client Android UI Feedback Meeting
Android Learning/Research
Android Development
Android GUI Skeleton
Android SQLite & Data Modelling
GC01 Milestone 2


2014
January
01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
Android Development
Android GUI Skeleton
Android SQLite & Data Modelling
Client UI Update Meeting
Merge Git Branches
Analytics & Model Integration
Client Final Feedback
Final Report Documentation
GC02 Final Milestone

11 Teamwork

11.1 Team Roles

Team roles were initially assigned based on the recommended list provided in the GC01 document requirements. These team roles were useful as a set of guidelines leading up to the first project milestone, but we quickly found that the assignment of multiple formal roles to team members was unrealistic for such a fast-paced project with team members at various skill levels. This meant that roles previously assigned to team members were not always reflective of their current work package, as the rest of the team pitched in to complete critical unfinished packages. As such, the currently assigned roles are reflective of the work that was mostly completed by the team member, rather than a formally assigned role.

Name	Primary Roles	Secondary Roles
Marco **David** Corrales	Secondary Team Lead Secondary Developer Data/Database Designer	Background Researcher
Amer **Kamil** Zainal	Primary Team Lead Primary Developer Systems Designer & Analyst Documentation Lead	Technology Researcher Software Tester Client Main Contact
Ke Wei	Networking Researcher	

Table 1: Team Member Roles

11.2 Teamwork Commentary

11.2.1 Team Workflow

Trello

Trello[9] has been used extensively by the team for information organisation. Agendas for the next meeting are typically decided at the end of each meeting, with unfinished items of the day deferred; ensuring that each agenda is a summary of what was achieved at each meeting. To supplement this, any details (e.g. comments, photos, resources) related to the meeting are included in a separate section with a corresponding date.

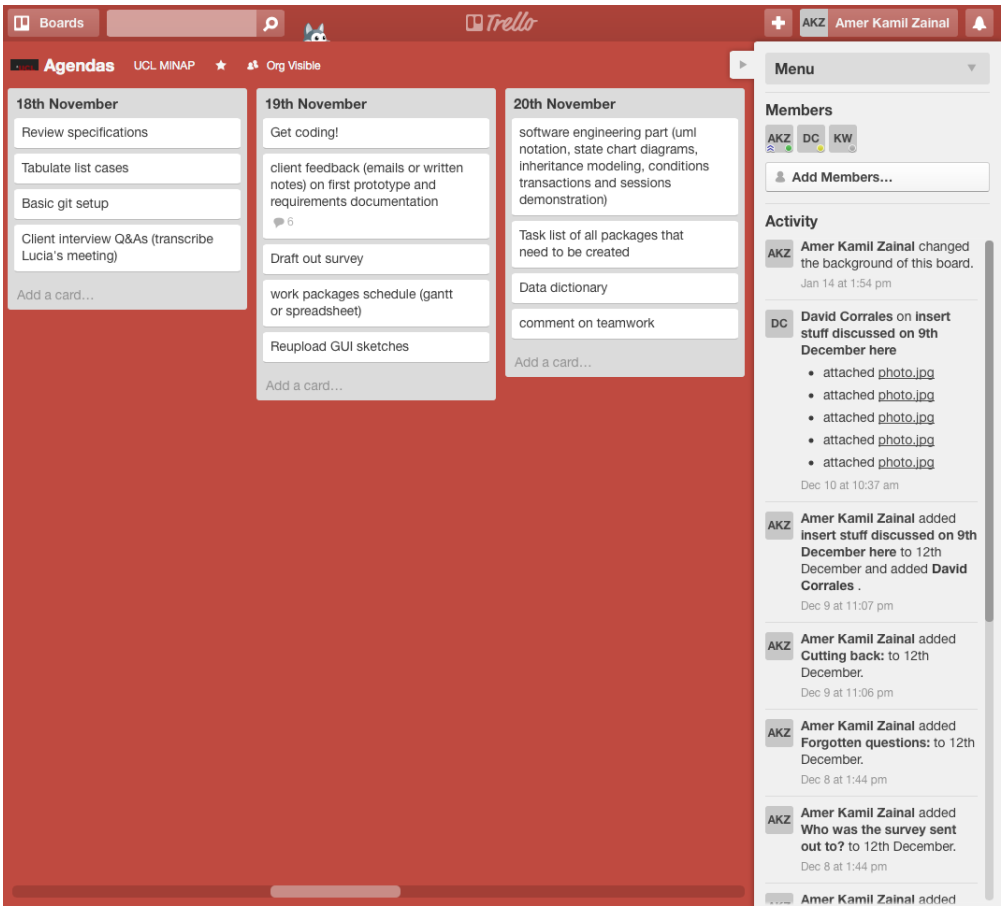


Figure 9: Snapshot of a board in the team's Trello

[9]Information disclosed on Trello are private to the team

GitHub

To further consolidate our memory of team progress, David updated a blog (http://akz08.github.io/minap-mobile/) on GitHub Pages using Octopress, summarising our meeting outcomes up to our first academic milestone. Due to the public nature of blogs, we initially ensured to omit any client identifying information to avoid any potential privacy problems, however we later decided that this was unnecessarily restrictive. This openness of communication was also necessary for us to write about our design process for the HCI requirement of our blog. All of the blog posts created are tagged with the author's name for attribution.

For source code collaboration, we used a private GitHub repository which was worked on by all members of the team. GitHub's repository wiki functionality was used for collaboration of more long-form documentation (e.g. writing this report) and structured ordering of resources. Due to the varying experience levels with git, the team mostly made use of simple git branches for each person.

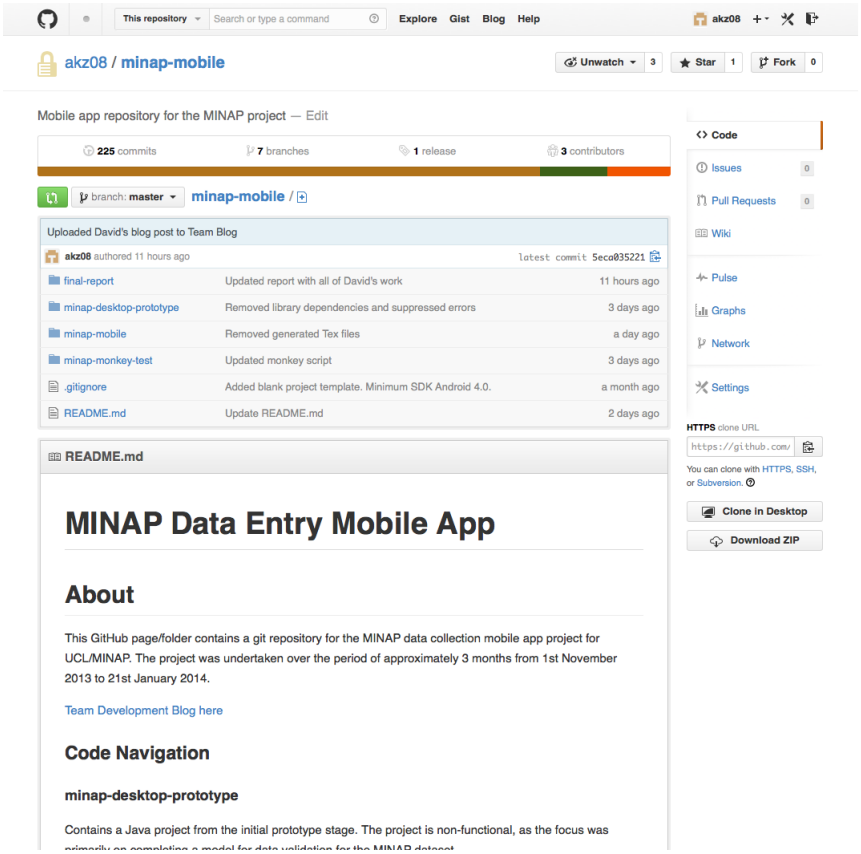


Figure 10: Snapshot of the team's GitHub repository page

Flowdock

For a unified cross-platform communication service, and to avoid having to check both Trello and GitHub frequently, we also used FlowDock extensively to easily track changes and keep each other updated. The service also allowed team members to control the level of notifications, to avoid a constant barrage of information (as well as work-life separation).

The service proved to be especially useful in tracking the progress of work packages, allowing other members to intervene if the pace of work was noticeably slower than expected.

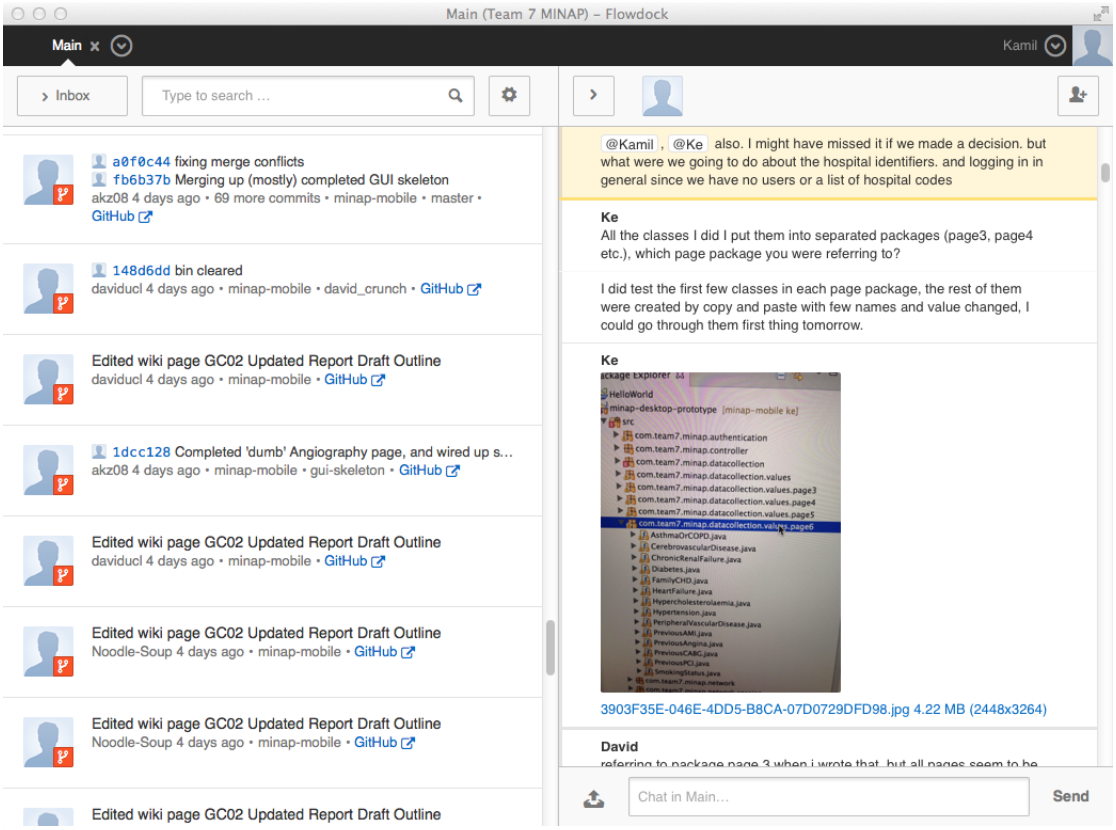


Figure 11: Snapshot of the team's Flowdock activity

11.2.2 Task Assignment

Team members typically were assigned work packages by the team leader, based on their current capabilities. These tended to be general work outlines, and were negotiable in their scope. The decisions on work packages at any given time were

strongly influenced by ongoing technology research, the current state of the overall system, and client feedback.

11.2.3 Meetings

The team typically met every workday to complete work together up to the first academic milestone. Most meetings since then have been conducted informally online using Flowdock, with occasional in-person catch-up meetings. This structure was more suited to the team once working on the final product, as work by each member was mostly independent of each other. Snapshots of early team meetings can be found in appendix C.

References

[1] Bill Phillips and Brian Hardy, *Android Programming: The Big Nerd Ranch Guide*. Big Nerd Ranch Guides, 28 Mar 2013.

[2] Android Architecture: Part 10, The Activity Revisited, http://www.therealjoshua.com/2012/07/android-architecture-part-10-the-activity-revisited/

[3] w3schools.com, Example of SOAP message: http://www.w3schools.com/webservices/ws_soap_syntax.asp

[4] ibm.com, Domino 8.5 user guide: http://publib.boulder.ibm.com/infocenter/domhelp/v8r0/index.jsp?topic=%2Fcom.ibm.designer.domino.main.doc%2FH_EXAMPLES_WEB_SERVICES.html

[5] w3.org, SOAP document from W3: http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

[6] kSOAP: http://ksoap2.sourceforge.net/

[7] C# Corner, How to Call Web Service in Android Using SOAP: http://www.c-sharpcorner.com/UploadFile/88b6e5/how-to-call-web-service-in-android-using-soap/

A Data Dictionary

A.1 Patient Class

AdminStatus	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.DemographicsAdmission	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- adminStatus : Byte	Holds the short code for patient's admission status
- statusLongCode : String	Holds the long code associated with adminStatus
+ <<constructor>> AdminStatus()	Sets the appropriate superclass values
+ setAdminStatus(admStatus : Byte) : Boolean	Returns true if parameter passes validation rules
+ getAdminStatus() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with adminStatus variable

AdmissionAfterNSTEMI	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.InitialDiagnosis	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- admissionAfterSTEMI : Boolean = false	Holds a boolean value, reveals items on UI
+ <<constructor>> AdmissionAfterNSTEMI()	Sets the appropriate superclass values
+ setAdmissionAfterSTEMI(admASTEMI : Boolean) : Boolean	Sets, then returns admissionAfterSTEMI
+ getAdmissionAfterSTEMI() : String	Returns a String value of admissionAfterSTEMI

AdmissionDate	<<**extends Value**>>
˜com.ucl.appteam7.minapmobile.model.values.PatientInfo	
<<import:>> com.ucl.appteam7.minapmobile.model.Value;	
<<import:>> java.text.ParseException;	
<<import:>> java.text.SimpleDateFormat;	
<<import:>> java.util.Date;	
- adminDate : String	Holds Date String, formatted as needed
- adminTime : String	Holds Time String, formatted as needed
- adminDateTime : String	Combines both previous variables
- now : Date = new Date()	Holds the current date, used for validation
sd : SimpleDateFormat = new SimpleDateFormat("dd/mm/yyyy")	Holds the String format desired for adminDate
st : SimpleDateFormat = new SimpleDateFormat("HH:mm")	Holds the String format desired for adminTime
sdt : SimpleDateFormat = new SimpleDateFormat("dd/mm/yyy HH:mm")	Holds the String format for adminDateTime
+ <<constructor>> AdmissionDate()	Sets the appropriate superclass values
+ setDateTime() : Boolean	Checks date and time validation, returns true if passed
+ setADate(aDate : Date) : Boolean	Checks the passed Date against date validation rules, returns true if passed
+ setATime(aTime : Date) : Boolean	Checks the passed Date against time validation rules, returns true if passed
+ getDateTime() : String	Returns adminDateTime variable
+ getADate() : String	Returns adminDate variable
+ getATime() : String	Returns adminTime variable

AdmissionMethod	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.DemographicsAdmission	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- admissionMethod : Byte	Holds the short code for patient's admission method
- admissionLongCode : String	Holds the long code associated with admissionMethod
+ <<constructor>> AdmissionMethod()	Sets the appropriate superclass values
+ setAdmissionMethod(admMethod : Byte) : Boolean	Returns true if parameter passes validation rules
+ getAdmissionMethod() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with admissionMethod variable

AdmissionWard	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.DemographicsAdmission	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- admissionWard : Byte	Holds the short code for patient's admission ward
- wardLongCode : String	Holds the long code associated with admissionWard
+ <<constructor>> AdmissionWard()	Sets the appropriate superclass values
+ setAdmissionWard(admWard : Byte) : Boolean	Returns true if parameter passes validation rules
+ getAdmissionWard() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with admissionWard variable

AdmittingConsultant	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.DemographicsAdmission	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- admittingConsultant : Byte	Holds the short code for patient's admitting consultant
- consulLongCode : String	Holds the long code associated with admittingConsultant
+ <<constructor>> AdmittingConsultant()	Sets the appropriate superclass values
+ setAdmittingConsultant(admConsul : Byte) : Boolean	Returns true if parameter passes validation rules
+ getAdmittingConsultant() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with admittingConsultant variable

AngioDate	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.Angiography	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
<<import>> java.text.ParseException	
<<import>> java.text.SimpleDateFormat	
<<import>> java.util.Date	
sd : SimpleDateFormat = new SimpleDateFormat("dd/mm/yyyy")	Holds the String format desired for angioDate
st : SimpleDateFormat = new SimpleDateFormat("HH:mm")	Holds the String format desired for angioTime
sdt : SimpleDateFormat = new SimpleDateFormat("dd/mm/yyyy HH:mm")	Holds the String format desired for angioDateTime
- angioDate : String	Holds the patient's angiography date
- angioTime : String	Holds the patient's angiography time
- angioDateTime : String	Holds the patient's angiography date and time
- *DATE_FORMAT* : String = "01/01/2000"	Lower bound for date range check
- now : Date = new Date()	Holds current Date, used for range check
+ <<constructor>> AngioDate()	Set the appropriate superclass values
+ setDateTime() : Boolean	Returns true if both dates passed validation
+ setangioralDate(rDate : Date) : Boolean	Returns true if Date parameter passes validation
+ setangioralTime(rTime : Date) : Boolean	Returns true if Time parameter passes validation
+ getDateTime() : String	Returns angioDateTime variable
+ getAngioDate() : String	Returns angioDate variable
+ getAngioTime() : String	Returns angioTime variable

AngioDelay	**<<extends Value>>**
~com.ucl.appteam7.minapmobile.model.values.Angiography	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- angioDelay : Byte	Holds the short code for patient's angiography delay
- angioDelayLongCode : String	Holds the long code associated with angioDelay
+ <<constructor>> AngioDelay()	Sets the appropriate superclass values
+ setAngioDelay(aDelay : Byte) : Boolean	Returns true if parameter passes validation rules
+ getAngioDelay() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with angioDelay variable

AsthmaOrCOPD	**<<extends Value>>**
~com.ucl.appteam7.minapmobile.model.values.MedicalHistory	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- asthmaCOPD : Byte	Holds the short code for patient's cerebrovascular disease status
- asthmaCOPDLongCode : String	Holds the long code associated with asthmaCOPD
+ <<constructor>> AsthmaOrCOPD()	Sets the appropriate superclass values
+ setAsthmaCOPD(cDisease : Byte) : Boolean	Returns true if parameter passes validation rules
+ getAsthmaCOPD() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with asthmaCOPD variable

BMI	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.Examinations	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- bmi : Double	Holds the patient's BMI
+ <<constructor>> BMI()	Set the appropriate superclass values
+ setBMI(height : Double, weight : Double) : Boolean	Calculates BMI, returns true if parameter passes validation
+ getBMI() : String	Returns bmi as a formatted String

CerebrovascularDisease	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.MedicalHistory	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- cerebrovascularDisease : Byte	Holds the short code for patient's cerebrovascular disease status
- cerebrovascularDiseaseLongCode : String	Holds the long code associated with cerebrovascularDisease
+ <<constructor>> CerebrovascularDisease()	Sets the appropriate superclass values
+ setCerebrovascular(cDisease : Byte) : Boolean	Returns true if parameter passes validation rules
+ getCerebrovascular() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with cerebrovascularDisease variable

ChronicRenalFailure	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.MedicalHistory	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- chronicRenalFailure : Byte	Holds the short code for patient's chronic renal failure status
- chronicRenalFailureLongCode : String	Holds the long code associated with chronicRenalFailure
+ <<constructor>> ChronicRenalFailure()	Sets the appropriate superclass values
+ setRenalFailure(cRenalFailure : Byte) : Boolean	Returns true if parameter passes validation rules
+ getRenalFailure() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with chronicRenalFailure variable

CoronaryAngiography	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.Angiography	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- coronaryAngiography : Byte	Holds the short code for patient's coronary angiography
- coronaryAngiographyLongCode : String	Holds the long code associated with coronaryAngiography
+ <<constructor>> CoronaryAngiography()	Sets the appropriate superclass values
+ setCoronaryAngiography(cAngiography : Byte) : Boolean	Returns true if parameter passes validation rules
+ getCoronaryAngiography() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with coronaryAngiography variable

CoronaryIntervention	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.Angiography	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- coronaryIntervention : Byte	Holds the short code for patient's coronary intervention
- coronaryInterventionLongCode : String	Holds the long code associated with coronaryIntervention
+ <<constructor>> CoronaryIntervention()	Sets the appropriate superclass values
+ setCoronaryIntervention(cIntervention : Byte) : Boolean	Returns true if parameter passes validation rules
+ getCoronaryIntervention() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with coronaryIntervention variable

DaycaseTransferDate	<<**extends Value**>>
˜com.ucl.appteam7.minapmobile.model.values.Angiography	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
<<import>> java.text.ParseException	
<<import>> java.text.SimpleDateFormat	
<<import>> java.util.Date	
sd : SimpleDateFormat = new SimpleDateFormat(”dd/mm/yyyy”)	Holds the String format desired for daycaseDate
- daycaseDate : String	Holds the patient’s daycase transfer date
- *DATE_FORMAT* : String = ”01/01/2000”	Lower bound for date range check
- now : String = sd.format(new Date())	Holds current Date as a String, used for range check
+ <<constructor>> DaycaseTransferDate()	Set the appropriate superclass values
+ setDaycaseDate(iDate: Date) : Boolean	Returns true if parameter passes validation
+ getDaycaseDate() : String	Returns daycaseDate variable

DBAdapter	**Holds all SQLite logic**
˜com.ucl.appteam7.minapmobile.model	
<<import>> java.util.Collections	
<<import>> java.util.Date	
<<import>> java.util.List	
<<import>> java.util.ArrayList	
<<import>> android.content.ContentValues	
<<import>> android.content.Context	
<<import>> android.database.Cursor	
<<import>> android.database.SQLException	
<<import>> android.database.sqlite.SQLiteOpenHelper	
<<import>> android.database.sqlite.SQLiteDatabase	
<<import>> android.util.Log	
+ $\underline{HOSPITAL\_IDENTIFIER}$: String = "HospitalID"	Declares HospitalID as a column for a database's table
+ $\underline{RECORD\_NO}$: String = "RecordNum"	Declares RecordNum
+ $\underline{NHS\_NUMBER}$: String = "NHSNumber"	Declares NHSNumber
+ $\underline{PATIENT\_SURNAME}$: String = "PatientSurname"	Declares PatientSurname
+ $\underline{PATIENT\_FORENAME}$: String = "PatientForename"	Declares PatientForename
+ $\underline{PATIENT\_DOB}$: String = "PatientDOB"	Declares PatientDOB
+ $\underline{ADMISSION\_DATE}$: String = "AdminDate"	Declares AdminDate
+ $\underline{PATIENT\_INFO}$: String[] = $\{HOSPITAL\_IDENTIFIER, RECORD\_NO, NHS\_NUMBER, PATIENT\_SURNAME, PATIENT\_FORENAME, PATIENT\_DOB, ADMISSION\_DATE\}$	Compiles previous columns for the Patient Info page
+ $\underline{INITIAL\_DIAGNOSIS}$: String = "InitialDiagosis"	Declares InitialDiagnosis
+ $\underline{ADMISSION\_AFTER\_NSTEMI}$: String = "AdminStemi"	Declares AdminStemi
+ $\underline{HIGH\_RISK\_NSTEMI}$: String = "HiRisknSTEMI"	Declares HiRisknSTEMI
+ $\underline{INTERVENTIONAL\_PROCEDURE}$: String = "InterventionalProcedure"	Declares InterventionalProcedure
+ $\underline{RETURN\_TO\_REFERRING\_HOSPITAL}$: String = "ReferHospitalReturn"	Declares ReferHospiralReturn
+ $\underline{INTERVENTIONAL\_CENTRE\_CODE\_ID}$: String = "InterventionCentreCode"	Declares InterventionCentreCode

+ $\underline{DIAGNOSIS}$: String[] = {$INITIAL\_DIAGNOSIS$, $ADMISSION\_AFTER\_NSTEMI$, $HIGH\_RISK\_NSTEMI$, $INTERVENTIONAL\_PROCEDURE$, $RETURN\_TO\_REFERRING\_HOSPITAL$, $INTERVENTIONAL\_CENTRE\_CODE\_ID$}	Compiles previous columns for Initial Diagnosis page
+ $\underline{PATIENT\_GENDER}$: String = "Gender"	Declares Gender
+ $\underline{PATIENT\_ETHNICITY}$: String = "Ethnicity"	Declares Ethnicity
+ $\underline{ADMISSION\_METHOD}$: String = "AdminMethod"	Declares AdminMethod
+ $\underline{ADMISSION\_WARD}$: String = "AdminWard"	Declares AdminWard
+ $\underline{GP\_PCT\_CODE}$: String = "GPCode"	Declares GPCode
+ $\underline{PATIENT\_POST\_CODE}$: String = "PostCode"	Declares PostCode
+ $\underline{ADMITTING\_CONSULTANT}$: String = "AdminConsul"	Declares AdminConsul
+ $\underline{ADMISSION\_STATUS}$: String = "AdminStatus"	Declares AdminStatus
+ $\underline{PLACE\_FIRST\_12\_LEAD\_ECG\_PERFORMED}$: String = "FirstECG"	Declares FirstECG
+ $\underline{NHS\_VERIFICATION}$: String = "NHSVerif"	Declares NHSVerif
+ $\underline{REFERRAL\_HOSPITAL}$: String = "RefHospital"	Declares RefHospital
+ $\underline{ADMISSION}$: String[] = {$PATIENT\_GENDER$, $PATIENT\_ETHNICITY$, $ADMISSION\_METHOD$, $ADMISSION\_WARD$, GP_PCT_CODE, $PATIENT\_POST\_CODE$, $ADMITTING\_CONSULTANT$, $ADMISSION\_STATUS$, $PLACE\_FIRST\_12\_LEAD\_ECG\_PERFORMED$, $NHS\_VERIFICATION$, $REFERRAL\_HOSPITAL$}	Compiles previous columns for Demographics and Admission page
+ $\underline{INITIAL\_REPERFUSION\_TREATMENT}$: String = "InitialReperfusionTreat"	Declares InitialReperfusionTreat
+ $\underline{REPERFUSION\_NOT\_GIVEN}$: String = "ReperNotGiven"	Declares ReperNotGiven
+ $\underline{ECG\_DETERMINING\_TREATMENT}$: String = "ECGDetermineTreat"	Declares ECGDetermineTreat
+ $\underline{ECG\_QRS\_COMPLEX\_DURATION}$: String = "$ECG\_QRSComplex$"	Declares $ECG\_QRSComplex$
+ $\underline{STEMI\_LOCATION}$: String = "LocationSTEMI"	Declares LocationSTEMI
+ $\underline{INTERVENTIONAL\_CENTRE\_CODE}$: String = "ReperInterventionCentre"	Declares ReperInterventionCentre
+ $\underline{INFARCTION\_SITE}$: String = "InfarctionSite"	Declares InfarctionSite
+ $\underline{REPERFUSION}$: String[] = {$INITIAL\_REPERFUSION\_TREATMENT$, $REPERFUSION\_NOT\_GIVEN$, $ECG\_DETERMINING\_TREATMENT$, $ECG\_QRS\_COMPLEX\_DURATION$, $STEMI\_LOCATION$, $INTERVENTIONAL\_CENTRE\_CODE$, $INFARCTION\_SITE$}	Compiles previous columns for Initial Reperfusion page

+ $\underline{CORONARY\_ANGIO}$: String = "CoronaryAngio"	Declares CoronaryAngio
+ $\underline{REFERRAL\_DATE}$: String = "ReferralDate"	Declares ReferralDate
+ $\underline{ANGIO\_PERFORM\_DELAY}$: String = "AngioDelay"	Declares AngioDelay
+ $\underline{ANGIO\_DATE}$: String = "AngioDate"	Declares AngioDate
+ $\underline{INTERVENTIONAL\_CENTRE\_CODE\_AN}$: String = "AngioInterventionCentre"	Declares AngioInterventionCentre
+ $\underline{LOCAL\_INTERVENTION}$: String = "LocalIntervention"	Declares LocalIntervention
+ $\underline{CORONARY\_INTERVENTION}$: String = "CoronaryIntervention"	Declares CoronaryIntervention
+ $\underline{PATIENT\_RETURN}$: String = "ReturnExpected"	Declares ReturnExpected
+ $\underline{DAYCASE\_TRANSFER}$: String = "DaycaseTransfer"	Declares DaycaseTransfer
+ $\underline{REFERRING\_RETURN}$: String = "ReferringHospitalReturn"	Declares ReferringHospitalReturn
+ $\underline{ANGIOGRAPHY}$: String[] = $\{CORONARY\_ANGIO, REFERRAL\_DATE, ANGIO\_PERFORM\_DELAY, ANGIO\_DATE, INTERVENTIONAL\_CENTRE\_CODE\_AN, LOCAL\_INTERVENTION, CORONARY\_INTERVENTION, PATIENT\_RETURN, DAYCASE\_TRANSFER, REFERRING\_RETURN\}$	Compiles previous columns for Angiography page
+ $\underline{SYSTOLIC}$: String = "SystolicBP"	Declares SystolicBP
+ $\underline{HEART\_RATE}$: String = "HeartRate"	Declares HeartRate
+ $\underline{KILLIP\_CLASS}$: String = "KillipClass"	Declares KillipClass
+ $\underline{BMI}$: String = "BMI"	Declares BMI
+ $\underline{HEIGHT}$: String = "Height"	Declares Height
+ $\underline{WEIGHT}$: String = "Weight"	Declares Weight
+ $\underline{EXAMINATIONS}$: String[] = $\{SYSTOLIC, HEART\_RATE, KILLIP\_CLASS, BMI, HEIGHT, WEIGHT\}$	Compiles previous columns for Examinations page
+ $\underline{PREVIOUS\_AMI}$: String = "PrevAMI"	Declares PrevAMI
+ $\underline{HYPERTENSION}$: String = "Hypertension"	Declares Hypertension
+ $\underline{CEREBROVASCULAR}$: String = "CerebroDisease"	Declares CerebroDisease
+ $\underline{PREVIOUS\_PCI}$: String = "PrevPCI"	Declares PrevPCI
+ $\underline{SMOKING}$: String = "Smoking"	Declares Smoking
+ $\underline{DIABETES}$: String = "Diabetes"	Declares Diabetes
+ $\underline{PREVIOUS\_ANGINA}$: String = "PrevAngina"	Declares PrevAngina
+ $\underline{HYPERCHOLESTEROL}$: String = "Hypercholesterol"	Declares Hypercholesterol
+ $\underline{ASTHMA\_COPD}$: String = "AsthmaCOPD"	Declares AsthmaCOPD
+ $\underline{PREVIOUS\_CABG}$: String = "PrevCABG"	Declares PrevCABG
+ $\underline{HEART\_FAILURE}$: String = "HeartFailure"	Declares HeartFailure
+ $\underline{PV\_DISEASE}$: String = "PeriphVascularDisease"	Declares PeriphVascularDisease

+ $\underline{RENAL\_FAILURE}$: String = "RenalFailure"	Declares RenalFailure
+ $\underline{FAMILY\_CHD}$: String = "FamilyCHD"	Declares FamilyCHD
+ $\underline{MEDICAL\_HISTORY}$: String[] = {$PREVIOUS\_AMI$, $HYPERTENSION$, $CEREBROVASCULAR$, $PREVIOUS\_PCI$, $SMOKING$, $DIABETES$, $PREVIOUS\_ANGINA$, $HYPERCHOLESTEROL$, $ASTHMA\_COPD$, $PREVIOUS\_CABG$, $HEART\_FAILURE$, $PV\_DISEASE$, $RENAL\_FAILURE$, $FAMILY\_CHD$}	Compiles previous columns for Medical History page
+ $\underline{TAG}$: String = "DBAdapter"	Used when updating the Database's schema
- $\underline{DATABASE\_NAME}$: String = "minap"	Names the Database
- $\underline{TABLE\_NAME}$: String = "patient"	Names the patient table
- $\underline{DATABASE\_VERSION}$: Integer = 11	Must be increase when the schema is changed
- $\underline{DATABASE\_CREATE}$: String	Holds the SQL Statement that creates the Database and table
- Context : Context	Provides context for the session when needed
- DBHelper : DatabaseHelper	Used for helping SQLite operations
- db : SQLiteDatabase	The Database itself
- <<class>> DatabaseHelper()	¡¡extends SQLiteOpenHelper¿¿ Inner class, creates and/or updates the database
+ <<override>> onCreate(db : SQLiteDatabase) : void	Creates the database, throws exception if unable to
+ <<override>> onUpgrade(db : SQLiteDatabase, oldVersion : Integer, newVersion : Integer) : void	Clears, then upgrades the database
- concatenateArrays(page0 : String[], page1 : String[], page2 : String[], page3 : String[], page4 : String[], page5 : String[], page6 : String[]) : String[]	Concatenates all page column arrays
+ <<constructor>> DBAdapter(ctx : Context)	Uses DBHelper to create and operate the database
+ open() : DBAdapter	Returns the writable database
+ close() : void	Closes the database

+ insertPatient(id : String, forename : String, surname : String, dob : String, nhs : String, hId : String, admDate : String) : Boolean	Returns true if parameters have been inserted onto database's patient info section
+ updatePatientInfo(oldid : String, newid : String, forename : String, surname : String, dob : String, nhs : String, hId : String, admDate : String) : Boolean	Returns true if parameters were used to update database's patient info section
+ getPatientInfo(id : String) : Cursor	Returns a cursor object containing the database's patient info section where id = recordNumber
+ updateInitialDiagnosis(id : String, diagnosis : String, admission : String, nstemi : String, procedure : String, refer : String, code : String) : Boolean	Returns true if parameters were used to update the database's initial diagnosis section
+ getDiagnosis(id : String) : Cursor	Returns a cursor object containing the database's initial diagnosis section where id = recordNumber
+ updateDemographics(id : String, gender : String, ethnicity : String, method : String, ward : String, gpcode : String, post : String, consultant : String, status : String, ecg, verif : String, refer : String) : Boolean	Returns true if parameters were used to update the database's Demographics and Admission section
+ getDemographics(id : String) : Cursor	Returns a cursor object containing the database's demographics and admission section where id = recordNumber
+ updateInitialReperfusion(id : String, treatment : String, reperfusion : String, ecgtreat : String, ecgqrs : String, location : String, iccode : String, infarction : String) : Boolean	Returns true if parameters were used to update the database's Initial Reperfusion section
+ getReperfusion(id : String) : Cursor	Returns a cursor object containing the database's initial reperfusion section where id = recordNumber
+ updateAngiography(id : String, angio : String, drefer : String, adelay : String, adate : String, acode : String, inter : String, coronary : String, patret : String, daycase : String, refret : String) : Boolean	Returns true if parameters were used to update the database's Angiography section

+ getAngiography(id : String) : Cursor	Returns a cursor object containing the database's angiography section where id = recordNumber
+ updateExaminations(id : String, systolic : String, heart : String, killip : String, bmi : String, height : String, weight : String) : Boolean	Returns true if parameters were used to update the database's Examinations section
+ getExaminations(id : String) : Cursor	Returns a cursor object containing the database's examinations section where id = recordNumber
+ updateMedicalHistory(id : String, ami : String, tension : String, cerebro : String, pci : String, smoke : String, diabetes : String, angina : String, choles : String, asthma : String, cabg : String, heart : String, vascular : String, renal : String, chd : String) : Boolean	Returns true if parameters were used to update the database's Medical History section
+ getMedicalHistory(id : String) : Cursor	Returns a cursor object containing the database's medical history section where id = recordNumber
+ deletePatient(id : String) : Boolean	Returns true if record where id = recordNum has been deleted
+ allPatients() : Cursor	Returns a cursor object containing all information on the database

Diabetes	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.MedicalHistory	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- diabetes : Byte	Holds the short code for patient's diabetes status
- diabetesLongCode : String	Holds the long code associated with diabetes
+ <<constructor>> Diabetes()	Sets the appropriate superclass values
+ setDiabetes(diabete : Byte) : Boolean	Returns true if parameter passes validation rules
+ getDiabetes() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with diabetes variable

DOB	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.PatientInfo	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
<<import>> java.text.ParseException	
<<import>> java.text.SimpleDateFormat	
<<import>> java.util.Date	
sd : SimpleDateFormat = new SimpleDateFormat("dd/mm/yyyy")	Holds the String format desired for dob
- dob : String	Holds the patient's date of birth
- *DATE_FORMAT* : String = "01/01/1900"	Lower bound for date range check
- twentyYrsAgo : String = "01/01/1994"	Upper bound for date range check
+ <<constructor>> DOB()	Set the appropriate superclass values
+ setDOB(bDate : Date) : Boolean	Checks the passed Date against validation rules, returns true if passed
+ getDOB() : String	Returns dob variable

EcgDetermineTreatment	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.InitialReperfusion	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- ecgDetermineTreatment : Byte	Holds the short code for patient's ECG determining treatment
- ecgDetermineTreatmentLongCode : String	Holds the long code associated with ecgDetermine-Treatment
+ <<constructor>> EcgDetermineTreatment()	Sets the appropriate superclass values
+ setECGTreatment(eDetermineTreatment : Byte) : Boolean	Returns true if parameter passes validation rules
+ getECGTreatment() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with ecgDetermine-Treatment variable

EcgQRSComplex	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.InitialReperfusion	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- ecgQRSComplex : Byte	Holds the short code for patient's ECG / QRS Complex
- ecgQRSComplexLongCode : String	Holds the long code associated with ecgQRSComplex
+ <<constructor>> EcgQRSComplex()	Sets the appropriate superclass values
+ setECGQRSComplex(eQRSComplex : Byte) : Boolean	Returns true if parameter passes validation rules
+ getECGQRSComplex() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with ecgQRSComplex variable

FamilyCHD	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.MedicalHistory	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- familyCHD : Byte	Holds the short code for patient's family CHD status
- familyCHDLongCode : String	Holds the long code associated with familyCHD
+ <<constructor>> FamilyCHD()	Sets the appropriate superclass values
+ setFamilyCHD(hFailure : Byte) : Boolean	Returns true if parameter passes validation rules
+ getFamilyCHD() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with familyCHD variable

FirstECG	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.DemographicsAdmission	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- firstECG : Byte	Holds the short code for patient's first ECG instance
- ecgLongCode : String	Holds the long code associated with firstECG
+ <<constructor>> FirstECG()	Sets the appropriate superclass values
+ setFirstECG(fECG : Byte) : Boolean	Returns true if parameter passes validation rules
+ getFirstECG() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with firstECG variable

Gender	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.DemographicsAdmission	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- patientGender : Byte	Holds the short code for patient's gender
- genderLongCode : String	Holds the long code associated with patientGender
+ <<constructor>> Gender()	Sets the appropriate superclass values
+ setPatientGender(gen : Byte) : Boolean	Returns true if parameter passes validation rules
+ getPatientGender() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with patientGender variable

GPCode	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.DemographicsAdmission	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- gpCode : String	Holds the patient's GP code
+ *VAL_LENGTH* : Short = 6	Holds the maximum length for gpCode
+ <<constructor>> GPCode()	Sets the appropriate superclass values
+ setGPCode(pctCode : String) : Boolean	Returns true if parameter passes validation rules
+ getGPCode() : String	Returns gpCode

HeartFailure	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.MedicalHistory	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- heartFailure : Byte	Holds the short code for patient's heart failure status
- heartFailureLongCode : String	Holds the long code associated with heartFailure
+ <<constructor>> HeartFailure()	Sets the appropriate superclass values
+ setHeartFailure(hFailure : Byte) : Boolean	Returns true if parameter passes validation rules
+ getHeartFailure() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with heartFailure variable

HeartRate	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.Examinations	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- heartRate : Double	Holds the patient's heart rate
+ <<constructor>> HeartRate()	Set the appropriate superclass values
+ setHeartRate(hRate : Double) : Boolean	Returns true if parameter passes validation
+ getHeartRate() : String	Returns heartRate

Height	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.Examinations	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- height : Double	Holds the patient's height
+ <<constructor>> Height()	Set the appropriate superclass values
+ setHeight(h : Double) : Boolean	Returns true if parameter passes validation
+ getHeight() : String	Returns height

HiRisknSTEMI	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.InitialDiagnosis	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- hiRisknSTEMI : Byte	Holds the short code choice for High Risk nSTEMI
- nSTEMILongCode : String	Holds the long code associated with hiRisknSTEMI
+ <<constructor>> HiRisknSTEMI()	Sets the appropriate superclass values
+ setHiRisknSTEMI(hiNSTEMI : Byte) : Boolean	Returns true if parameter passes validation rules
+ getHiRisknSTEMI() : String	Returns the selected short code and corresponding long code as a String

HospitalIdentifier	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.PatientInfo	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- hospitalID : String	Holds a three letter string for hospitalID
+ *VAL_LENGTH* : Short = 3	Holds the valid length for hospitalID
+ <<constructor>> HospitalIdentifier()	Sets the appropriate superclass values
+ setHospitalIdentifier(hospId : String) : Boolean	Checks the paramenter against validation rules, returns true if passed
+ getHospitalIdentifier() : String	Returns hospitalID

Hypercholesterolaemia	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.MedicalHistory	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- hyperCholesterolaemia : Byte	Holds the short code for patient's hypercholesterolaemia status
- hyperCholesterolaemiaLongCode : String	Holds the long code associated with hyperCholesterolaemia
+ <<constructor>> Hypercholesterolaemia()	Sets the appropriate superclass values
+ setHypercholesterol(hCholesterolaemia : Byte) : Boolean	Returns true if parameter passes validation rules
+ getHypercholesterol() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with hyperCholesterolaemia variable

Hypertension	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.MedicalHistory	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- hyperTension : Byte	Holds the short code for patient's hypertension status
- hyperTensionLongCode : String	Holds the long code associated with hyperTension
+ <<constructor>> Hypertension()	Sets the appropriate superclass values
+ setHypertension(hTension : Byte) : Boolean	Returns true if parameter passes validation rules
+ getHypertension() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with hyperTension variable

InitialDiagnosis	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.InitialDiagnosis	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- initialDiagnosis : Byte	Holds the short code choice for Initial Diagnosis
- diagnosisLongCode : String	Holds the long code associated with initialDiagnosis
+ <<constructor>> InitialDiagnosis()	Sets the appropriate superclass values
+ setInitialDiagnosis(iniDiagnosis : Byte) : Boolean	Returns true if parameter passes validation rules
+ getInitialDiagnosis() : String	Returns the selected short code and corresponding long code as a String
+ getInitialLongCode() : String	Returns diagnosisLongCode

InitialReperfusion	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.InitialReperfusion	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- initialReperfusion : Byte	Holds the short code for patient's initial reperfusion
- initialReperfusionLongCode : String	Holds the long code associated with initialReperfusion
+ <<constructor>> InitialReperfusion()	Sets the appropriate superclass values
+ setInitialReperfusion(iReperfusion : Byte) : Boolean	Returns true if parameter passes validation rules
+ getInitialReperfusion() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with initialReperfusion variable

InfarctionSite	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.InitialReperfusion	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- infarctionSite : Byte	Holds the short code for patient's infarction site
- infarctionSiteLongCode : String	Holds the long code associated with infarctionSite
+ <<constructor>> InfarctionSite()	Sets the appropriate superclass values
+ setInfarctionSite(iSite : Byte) : Boolean	Returns true if parameter passes validation rules
+ getInfarctionSite() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with infarctionSite variable

InterventionalCentreCode	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.InitialDiagnosis	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- interventionalID : String	Holds the patient's Interventional Centre Code
+ *VAL_LENGTH* : Short = 3	Holds the maximum length for interventionalID
+ <<constructor>> InterventionalCentreCode()	Sets the appropriate superclass values
+ setInterventionalCentre(iCode : String) : Boolean	Returns true if parameter passes validation rules
+ getInterventionalCentre() : String	Returns interventionalID
Note: This class is used three times within the Patient singleton	

InterventionalProcedure	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.InitialDiagnosis	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- interventionalProcedure : Byte	Holds the short code for Interventional Procedure
- procedureLongCode : String	Holds the long code associated with interventionalProcedure
+ <<constructor>> InterventionalProcedure()	Sets the appropriate superclass values
+ setInterventionalProcedure(interHosProcedure : Byte) : Boolean	Returns true if parameter passes validation rules
+ getInterventionalProcedure() : String	Returns the selected short code and corresponding long code as a String

KillipClass	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.Examinations	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- killipClass : Byte	Holds the short code for patient's coronary angiography
- killipClassLongCode : String	Holds the long code associated with killipClass
+ <<constructor>> KillipClass()	Sets the appropriate superclass values
+ setKillipClass(kClass : Byte) : Boolean	Returns true if parameter passes validation rules
+ getKillipClass() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with killipClass variable

LocationAtSTEMI	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.InitialReperfusion	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- locationAtSTEMI : Byte	Holds the short code for patient's location at STEMI
- locationAtSTEMILongCode : String	Holds the long code associated with locationAtSTEMI
+ <<constructor>> LocationAtSTEMI()	Sets the appropriate superclass values
+ setSTEMILocation(lAtSTEMI : Byte) : Boolean	Returns true if parameter passes validation rules
+ getSTEMILocation() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with locationAtSTEMI variable

LocalInterventionDate	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.Angiography	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
<<import>> java.text.ParseException	
<<import>> java.text.SimpleDateFormat	
<<import>> java.util.Date	
sd : SimpleDateFormat = new SimpleDateFormat(”dd/mm/yyyy”)	Holds the String format desired for interventionDate
- interventionDate : String	Holds the patient’s local intervention date
- *DATE_FORMAT* : String = ”01/01/2000”	Lower bound for date range check
- now : String = sd.format(new Date())	Holds current Date as a String, used for range check
+ <<constructor>> LocalInterventionDate()	Set the appropriate superclass values
+ setInterventionDate(iDate: Date) : Boolean	Returns true if parameter passes validation
+ getInterventionDate() : String	Returns interventionDate variable

NHSNumber	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.PatientInfo	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- nhsNumber : String	Holds the patient’s NHS number
+ *VAL_LENGTH* : Short = 10	Holds the maximum length for nhsNumber
+ <<constructor>> NHSNumber()	Sets the appropriate superclass values
+ <<setNHSNum>>(nhsNum : String) : Boolean	Checks the parameter against validation rules, returns true if passed
+ <<getNHSNum>>() : String	Returns nhsNumber

NHSVerification	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.DemographicsAdmission	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- verification : Byte	Holds the short code for patient's NHS verification
- verificationLongCode : String	Holds the long code associated with verification
+ <<constructor>> NHSVerification()	Sets the appropriate superclass values
+ setVerification(verif : Byte) : Boolean	Returns true if parameter passes validation rules
+ getVerification() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with verification variable

Patient	<<**singleton**>>
˜com.ucl.appteam7.minapmobile.model	
<<import>> com.ucl.appteam7.minapmobile.model.values.PatientInfo.*	
<<import>> com.ucl.appteam7.minapmobile.model.values.InitialDiagnosis.*	
<<import>> com.ucl.appteam7.minapmobile.model.values.DemographicsAdmission.*	
<<import>> com.ucl.appteam7.minapmobile.model.values.InitialReperfusion.*	
<<import>> com.ucl.appteam7.minapmobile.model.values.Angiography.*	
<<import>> com.ucl.appteam7.minapmobile.model.values.Examinations.*	
<<import>> com.ucl.appteam7.minapmobile.model.values.MedicalHistory.*	
- instance : Patient	Controls the singleton behavior of this class
+ HospitalIdentifier : HospitalIdentifier	Declares a HospitalIdentifier object to the singleton
+ RecordNumber : PatientCaseRecordNumber	Declares a PatientCaseRecordNumber object to the singleton
+ NHSNumber : NHSNumber	Declares a NHSNumber object to the singleton
+ Surname : PatientSurname	Declares a PatientSurname object to the singleton
+ Forename : PatientForename	Declares a PatientForename object to the singleton
+ DOB : DOB	Declares a DOB object to the singleton
+ AdmissionDate : AdmissionDate	Declares a AdmissionDate object to the singleton
+ InitialDiagnosis : InitialDiagnosis	Declares a InitialDiagnosis object to the singleton
+ AdmissionAfterNSTEMI : AdmissionAfterNSTEMI	Declares a AdmissionAfterNSTEMI object to the singleton
+ HighRisknSTEMI : HiRisknSTEMI	Declares a HiRisknSTEMI object to the singleton

+ InterventionalProcedure : InterventionalProcedure	Declares a InterventionalProcedure object to the singleton
+ ReferHospitalReturn : ReferHospitalReturn	Declares a ReferHospitalReturn object to the singleton
+ InterventionalCentreCode : InterventionalCentreCode	Declares a InterventionalCentreCode object to the singleton
+ Gender : Gender	Declares a Gender object to the singleton
+ Ethnicity : PatientEthnicity	Declares a PatientEthnicity object to the singleton
+ AdmissionMethod : AdmissionMethod	Declares a AdmissionMethod object to the singleton
+ AdmissionWard : AdmissionWard	Declares a AdmissionWard object to the singleton
+ GPCode : GPCode	Declares a GPCode object to the singleton
+ PostCode : PatientPostcode	Declares a PatientPostcode object to the singleton
+ AdmitConsul : AdmittingConsultant	Declares a AdmittingConsultant object to the singleton
+ AdminStatus : AdminStatus	Declares a AdminStatus object to the singleton
+ FirstECG : FirstECG	Declares a FirstECG object to the singleton
+ NHSVerification : NHSVerification	Declares a NHSVerification object to the singleton
+ ReferralHospital : ReferralHospital	Declares a ReferralHospital object to the singleton
+ InitialReperfusion : InitialReperfusion	Declares a InitialReperfusion object to the singleton
+ ReperfusionNotGiven : ReperfusionNotGiven	Declares a ReperfusionNotGiven object to the singleton
+ ECGTreatment : EcgDetermineTreatment	Declares a EcgDetermineTreatment object to the singleton

+ *ECG_QRSComplex* : EcgQRSComplex	Declares a EcgQRSComplex object to the singleton
+ LocationSTEMI : LocationAtSTEMI	Declares a LocationAtSTEMI object to the singleton
+ ReperfusionCentreCode : InterventionalCentreCode	Declares a second InterventionalCentreCode object to the singleton
+ InfarctionSite : InfarctionSite	Declares a InfarctionSite object to the singleton
+ CoronaryAngiography : CoronaryAngiography	Declares a CoronaryAngiography object to the singleton
+ ReferralDate : ReferralDate	Declares a ReferralDate object to the singleton
+ AngioDelay : AngioDelay	Declares a AngioDelay object to the singleton
+ AngioDate : AngioDate	Declares a AngioDate object to the singleton
+ AngioCentreCode : InterventionalCentreCode	Declares a third InterventionalCentreCode object to the singleton
+ InterventionDate : LocalInterventionDate	Declares a LocalInterventionDate object to the singleton
+ CoronaryIntervention : CoronaryIntervention	Declares a CoronaryIntervention object to the singleton
+ ReturnExpected : ReturnExpected	Declares a ReturnExpected object to the singleton
+ DaycaseTransfer : DaycaseTransferDate	Declares a DaycaseTransferDate object to the singleton
+ ReferReturnDate : ReferHospitalReturnDate	Declares a ReferHospitalReturnDate object to the singleton
+ Systolic : SystolicBP	Declares a SystolicBP object to the singleton
+ HeartRate : HeartRate	Declares a HeartRate object to the singleton

+ Killip : KillipClass	Declares a KillipClass object to the singleton
+ Height : Height	Declares a Height object to the singleton
+ Weight : Weight	Declares a Weight object to the singleton
+ BMI : BMI	Declares a BMI object to the singleton
+ PreviousAMI : PreviousAMI	Declares a PreviousAMI object to the singleton
+ Hypertension : Hypertension	Declares a Hypertension object to the singleton
+ Cerebrovascular : CerebrovascularDisease	Declares a CerebrovascularDisease object to the singleton
+ PreviousPCI : PreviousPCI	Declares a PreviousPCI object to the singleton
+ Smoking : SmokingStatus	Declares a SmokingStatus object to the singleton
+ Diabetes : Diabetes	Declares a Diabetes object to the singleton
+ PreviousAngina : PreviousAngina	Declares a PreviousAngina object to the singleton
+ HyperCholesterol : Hypercholesterolaemia	Declares a Hypercholesterolaemia object to the singleton
+ AsthmaCOPD : AsthmaOrCOPD	Declares a AsthmaOrCOPD object to the singleton
+ PreviousCABG : PreviousCABG	Declares a PreviousCABG object to the singleton
+ HeartFailure : HeartFailure	Declares a HeartFailure object to the singleton
+ PeripheralVascular : PeripheralVascularDisease	Declares a PreipheralVascularDisease object to the singleton
+ RenalFailure : ChronicRenalFailure	Declares a ChronicRenalFailure object to the singleton

+ FamilyCHD : FamilyCHD	Declares a FamilyCHD object to the singleton
+ get() : Patient	Returns the instance variable, or creates one if it does not exist
- <<constructor>> Patient()	Private constructor, restricts creation of Patient objects to this class only
# <<override>> clone() : Object	Forbids a clone() method call. Completes the singleton declaration

PatientCaseRecordNumber	<<**extends Value**>>
~com.ucl.appteam7.minapmobile.model.values.PatientInfo	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- caseNumber : String	Hold the patient's Case Number
- oldCase : String = caseNumber	Holds a copy of a patient's previous record number, used when updating Database
- *VAL_LENGTH* : Short = 10	Holds the maximum length for caseNumber
+ <<constructor>> PatientCaseRecordNumber()	Sets the appropriate superclass values
+ <<setCaseNumber>>(cNumber : String) : Boolean	Returns true if parameter passes validation rules
+ <<getCaseNumber>>()	Returns caseNumber
+ <<getOldCase>>()	Returns oldCase

PatientEthnicity	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.DemographicsAdmission	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- ethnicity : Byte	Holds the short code for patient's Ethnicity
- ethnicLongCode : String	Holds the long code associated with ethnicity
+ <<constructor>> PatientEthnicity()	Sets the appropriate superclass values
+ setEthnicity(ethnic : Byte) : Boolean	Returns true if parameter passes validation rules
+ getEthnicity() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with ethnicity variable

PatientForename	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.PatientInfo	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- forename : String	Holds the patient's forename
+ *VAL_LENGTH* : Short = 35	Holds the maximum length for forename
+ <<constructor>> PatientForename()	Sets the appropriate superclass values
+ setForename(fName : String) : Boolean	Returns true if parameter passes validation rules
+ getForename() : String	Returns forename variable

PatientPostcode	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.DemographicsAdmission	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- postCode : String	Holds the patient's Postcode
+ *VAL_LENGTH* : Short = 8	Holds the maximum length for postCode
+ <<constructor>> PatientPostcode()	Sets the appropriate superclass values
+ setPostcode(rId : String) : Boolean	Returns true if parameter passes validation rules
+ getPostcode() : String	Returns postCode

PatientSurname	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.PatientInfo	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- surname : String	Holds the patient's surname
+ *VAL_LENGTH* : Short = 35	Holds the maximum length for surname
+ <<constructor>> PatientSurname()	Sets the appropriate superclass values
+ setSurname(sName : String) : Boolean	Returns true if parameter passes validation rules
+ getSurname() : String	Returns surname variable

PeripheralVascularDisease	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.MedicalHistory	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- peripheralVascularDisease : Byte	Holds the short code for patient's peripheral vascular disease
- peripheralVascularDiseaseLongCode : String	Holds the long code associated with peripheralVascularDisease
+ <<constructor>> PeripheralVascularDisease()	Sets the appropriate superclass values
+ setPeripheralVascularDisease(pVascularDisease : Byte) : Boolean	Returns true if parameter passes validation rules
+ getPeripheralVascularDisease() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with peripheralVascularDisease variable

PreviousAMI	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.MedicalHistory	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- previousAMI : Byte	Holds the short code for patient's previous AMI status
- previousAMILongCode : String	Holds the long code associated with previousAMI
+ <<constructor>> PreviousAMI()	Sets the appropriate superclass values
+ setPreviousAMI(pAMI : Byte) : Boolean	Returns true if parameter passes validation rules
+ getPreviousAMI() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with previousAMI variable

PreviousAngina	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.MedicalHistory	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- previousAngina : Byte	Holds the short code for patient's previous angina status
- previousAnginaLongCode : String	Holds the long code associated with previousAngina
+ <<constructor>> PreviousAngina()	Sets the appropriate superclass values
+ setPreviousAngina(pAngina : Byte) : Boolean	Returns true if parameter passes validation rules
+ getPreviousAngina() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with previousAngina variable

PreviousCABG	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.MedicalHistory	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- previousCABG : Byte	Holds the short code for patient's previous CABG status
- previousCABGLongCode : String	Holds the long code associated with previousCABG
+ <<constructor>> PreviousCABG()	Sets the appropriate superclass values
+ setPreviousCABG(pCABG : Byte) : Boolean	Returns true if parameter passes validation rules
+ getPreviousCABG() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with previousCABG variable

PreviousPCI	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.MedicalHistory	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- previousPCI : Byte	Holds the short code for patient's previous PCI status
- previousPCILongCode : String	Holds the long code associated with previousPCI
+ <<constructor>> PreviousPCI()	Sets the appropriate superclass values
+ setPreviousPCI(pPCI : Byte) : Boolean	Returns true if parameter passes validation rules
+ getPreviousPCI() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with previousPCI variable

ReferHospitalReturn	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.InitialDiagnosis	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
<<import>> java.text.ParseException	
<<import>> java.text.SimpleDateFormat	
<<import>> java.util.Date	
- rDate : String	Holds the Date for return from Referring Hospital
- now : Date = new Date()	Holds the current date used for validation
sd SimpleDateFormat = new SimpleDateFormat("dd/mm/yyyy")	Holds the string format desired for rDate
+ <<constructor>> ReferHospitalReturn()	Sets the appropriate superclass values
+ setRDate(bDate : Date) : Boolean	Returns true if paramenter passes validation rules
+ getRDate() : String	Returns rDate

ReferHospitalReturnDate	<<**extends Value**>>
˜com.ucl.appteam7.minapmobile.model.values.Angiography	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
<<import>> java.text.ParseException	
<<import>> java.text.SimpleDateFormat	
<<import>> java.util.Date	
sd : SimpleDateFormat = new SimpleDateFormat(”dd/mm/yyyy”)	Holds the String format desired for returnDate
- returnDate : String	Holds the patient’s date of return to referral hospital
- *DATE_FORMAT* : String = ”01/01/2000”	Lower bound for date range check
- now : String = sd.format(new Date())	Holds current Date as a String, used for range check
+ <<constructor>> ReferHospitalReturnDate()	Set the appropriate superclass values
+ setReturnDate(rhrDate: Date) : Boolean	Returns true if parameter passes validation
+ getReturnDate() : String	Returns returnDate variable

ReferralDate	<<**extends Value**>>
˜com.ucl.appteam7.minapmobile.model.values.Angiography	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
<<import>> java.text.ParseException	
<<import>> java.text.SimpleDateFormat	
<<import>> java.util.Date	
sd : SimpleDateFormat = new SimpleDateFormat(”dd/mm/yyyy”)	Holds the String format desired for referDate
st : SimpleDateFormat = new SimpleDateFormat(”HH:mm”)	Holds the String format desired for referTime
sdt : SimpleDateFormat = new SimpleDateFormat(”dd/mm/yyyy HH:mm”)	Holds the String format desired for referDateTime
- referDate : String	Holds the patient’s referral date
- *DATE_FORMAT* : String = ”01/01/2000”	Lower bound for date range check
- now : Date = new Date()	Holds current Date, used for range check
+ <<constructor>> ReferralDate()	Set the appropriate superclass values
+ setDateTime() : Boolean	Returns true if both dates passed validation
+ setReferralDate(rDate : Date) : Boolean	Returns true if Date parameter passes validation
+ setReferralTime(rTime : Date) : Boolean	Returns true if Time parameter passes validation
+ getDateTime() : String	Returns referDateTime variable
+ getReferralDate() : String	Returns referDate variable
+ getReferralTime() : String	Returns referTime variable

ReferralHospital	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.DemographicsAdmission	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- referralId : String	Holds the patient's ReferralHospital
+ *VAL_LENGTH* : Short = 3	Holds the maximum length for referralId
+ <<constructor>> ReferralHospital()	Sets the appropriate superclass values
+ setReferralIdentifier(rId : String) : Boolean	Returns true if parameter passes validation rules
+ getReferralIdentifier() : String	Returns referralID

ReperfusionNotGiven	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.InitialReperfusion	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- reperfusionNotGiven : Byte	Holds the short code for reason reperfusion was not given
- rNotGivenLongCode : String	Holds the long code associated with reperfusionNotGiven
+ <<constructor>> ReperfusionNotGiven()	Sets the appropriate superclass values
+ setReperfusionNotGiven(rNotGiven : Byte) : Boolean	Returns true if parameter passes validation rules
+ getReperfusionNotGiven() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with ReperfusionNotGiven variable

ReturnExpected	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.Angiography	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- returnExpected : Boolean = false	Holds a boolean value, reveals items on UI
+ <<constructor>> ReturnExpected()	Sets the appropriate superclass values
+ setReturnExpected(admASTEMI : Boolean) : Boolean	Sets, then returns returnExpected
+ getReturnExpected() : String	Returns a String value of returnExpected

SmokingStatus	**<<extends Value>>**
˜com.ucl.appteam7.minapmobile.model.values.MedicalHistory	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- smokingStatus : Byte	Holds the short code for patient's smoking status
- smokingStatusLongCode : String	Holds the long code associated with smokingStatus
+ <<constructor>> SmokingStatus()	Sets the appropriate superclass values
+ setSmokingStatus(sStatus : Byte) : Boolean	Returns true if parameter passes validation rules
+ getSmokingStatus() : String	Returns the selected short code and corresponding long code as a String
+ getLongCode() : String	Returns the long code associated with smokingStatus variable

SystolicBP	**<<extends Value>>**
~com.ucl.appteam7.minapmobile.model.values.Examinations	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- systolicBP : Double	Holds the patient's systolic blood pressure
+ <<constructor>> SystolicBP()	Set the appropriate superclass values
+ setSystolicBP(sBP : Double) : Boolean	Returns true if parameter passes validation
+ getSystolicBP() : String	Returns systolicBP

Value	**<<superclass>>**
~com.ucl.appteam7.minapmobile.model	
- id : String	Will contain the dataset field value for each field class
- notes : String	Will contain the field description for each class
- notesTitle : String	Will hold the title for tooltips used in the UI
- isValid : Boolean	To be used in future networking endeavors
- isComplete : Boolean	To be used in future networking endeavors
+ <<constructor>> Value (id : String, notes : String)	Called and populated from field class declarations
+ <<constructor>> Value (id : String, notes : String, notesTitle : String)	Called and populated from field class declarations
+ isValidated() : Boolean	Getter method for isValid variable
+ isCompleted() : Boolean	Getter method for isComplete variable
+ getId() : String	Getter method for id variable
+ getNotes() : String	Getter method for notes variable
+ getNotesTitle() : String	Getter method for notesTitle variable

Weight	<<**extends Value**>>
˜com.ucl.appteam7.minapmobile.model.values.Examinations	
<<import>> com.ucl.appteam7.minapmobile.model.Value	
- weight : Double	Holds the patient's weight
+ <<constructor>> Weight()	Set the appropriate superclass values
+ setWeight(w : Double) : Boolean	Returns true if parameter passes validation
+ getWeight() : String	Returns weight

B Client Communication

Our first contact with the client was on Monday 28th October, with a short meeting outlining the aims of the MINAP project the following day. We have had 3 meetings with the client and other members of MINAP/NICOR so far. All meetings have been recorded (audio) with the consent of those present. A brief summary of the people present and key talking points are outlined below.

B.1 Project Introduction & Requirements (1/11/13)

B.1.1 Meeting Agenda

To get a clear understanding of the project requirements, and to hammer down a realistic project specification that can be delivered within 3 months.

B.1.2 People Present

App Team 7, Lucia Gavalova (Project Manager), Fabian D'Souza (Server Admin), Owen Nicholas, "Tech-guy"

B.1.3 Discussion

1. Feasibility of creating a mobile app from the current web app
   (a) Tech-guy suggests that it is likely that XPages (open-source javascript framework used for business solutions) is the only way of creating an app
   (b) Tech-guy has no idea if the Domino server housing the MINAP data can be accessed via mobile, though suggests XPages as a starting point
2. Fabian called in to hopefully get better ideas on how to implement a mobile solution working with the current technology stack
   (a) Recommends looking at using IBM Lotus Notes (client software to Domino server) as an intermediary for the app
   (b) Suggests that Java could theoretically be used by including a notes.jar file which allows the development of a Lotus Notes plug-in
   (c) Promises to send relevant links that may help with finding a solution
3. Owen and tech-guy suggests that the validation of 130 values of the MINAP dataset will definitely take more than 3 months.

B.1.4 Outcomes

We were thoroughly confused after the meeting with regards to the Domino Server. Fabian did send the e-mail but it only served to confuse even further. We however decided to spend some time researching on the possible ways to communicate with this legacy technology. Lucia provided us with documents containing the full set of data values and validations rules for us to understand. We also provided details to gain access to MINAPs development server to have access to the current web application - this was only processed 2 weeks later.

B.2 UI Feedback (15/11/13)

B.2.1 Meeting Agenda

To obtain client feedback on our preliminary UI, clarify confusions about the MINAP dataset, validation, and use. The target platform needed to be locked down.

B.2.2 People Present

App Team 7, Lucia Gavalova

B.2.3 Discussion

1. Marco David Corrales presented the UI sketches to Lucia
    (a) Lucia is happy with the proposal of a design similar to the current web app
    (b) Mentions that the search functionality can just mirror the simple search in the current web app
    (c) Requests that a tutorial for non-techie users
    (d) David notes down minor comments on UI sketches
2. Team requests a simplified version of the navigation map (and by extension, the 130 value dataset)
    (a) Lucia informs us of her initial motivation for a MINAP mobile app
        i. Mobile app serves as a starting point for medical staff to create a record for patients eligible to be monitored through MINAP when visiting non-cardiac wards
        ii. This helps alleviate the current problem of having positively skewed outcomes as the current patients being recorded are in specialist wards, and more likely to receive treatment when needed

(b) The mobile app could therefore exist with just a subset of the functionality of the existing web app

3. Lucia requests an auto-save feature like Microsoft Word to allow sessions to be resumed later within the same day

4. Amer Kamil Zainal requests if a survey could be designed to get a solid feel of the target audiences primary mobile operating system

   (a) Lucia agrees - Kamil promises to design survey

B.2.4 Outcomes

We managed to come to a mutual agreement that the mobile app could serve its main purpose by implementing only a subset of the total MINAP dataset. Lucia promised to send a document detailing the exact values that would need to be included to fulfil this requirement - the document was received some time after the next meeting. A flow diagram of the survey was completed and handed over to Lucia at the next meeting.

B.3 Developer Session (20/11/13)

B.3.1 Meeting Agenda

To understand how the current web app works and how it interacts with the Domino Server; to hand over the device survey flow chart for critique and deployment on SurveyMonkey.

B.3.2 People Present

App Team 7, Lucia Gavalova, Sue Manuel (Developer), Fabian D'Souza

B.3.3 Discussion

1. Sue informs that the web app is made in Javascript, running on top of a Lotus Notes form (in turn, running on the Domino Server)

2. The Javascript methods use the Lotus Notes interface to retrieve data from the database

3. Sue mistakenly tells that validation of values is all server-side - a major setback if correct

4. Fabian called in to verify server-side validation claims - validation is client-side

5. Team request to have a copy of client-side validation code to cross-check with the data validation documents given

6. Discussion with Fabian regarding connecting to Domino Server

   (a) Team confirms research on how to communicate with Domino Server is correct

      i. The Domino Server allows the creation of a web service (consumer and provider)

      ii. This would make the server be compatible with any application that can consume a web service

   (b) Fabian however insists that making our app communicate with a Domino Server is not a key priority, and that spinning off another web server (e.g. Apache + Microsoft SQL Server) with a similar web service capabilities will satisfy the requirements

7. Device survey flow diagram presented and accepted as is by Lucia

B.3.4 Outcomes

The team is now confident in delivering an application with the scaled-down specifications (fewer than 130 validations and Domino Server compatibility not being a priority). A link to the SurveyMonkey device survey was sent to the team, pending approval (by the team). However, due to the reduction of requirements occurring so close to the first report milestone (24/11/13), the team was unable to fully capture this more manageable system in Java.

B.4 UI & Features Demonstration (05/12/13)

B.4.1 Meeting Purpose

To present the User Interface to the client as well as to show the features that will be in the Android app.

B.4.2 People Present

App Team 7, Lucia Gavalova

B.4.3 Discussion

1. Kamil presented the UI and app features to Lucia
   (a) Confirmed there will be a tutorial for the app after the user logged in
      i. Lucia suggested that the tutorial should be before the logging in
   (b) Confirmed that the disclaimer will be only shown once when the app is launched the first time
   (c) Lucia liked the way Kamil presented the UI of the app
   (d) Lucia suggested the hospital name should be auto-filled when creating patient records, the team promised to look into it
   (e) Lucia promised to get different people to test the app, and gather feedback
   (f) Lucia asked some questions about the saving features of the app
   (g) A question from Kamil about an indicator on the current web app, which turned out to be a completion indicator, which is not working properly on the web service
      i. Confirmed the colours should be used as indicators as to whether the section has been filled
   (h) Lucia suggested the compulsory fields should be highlighted
   (i) Lucia is happy with the UI
2. David mentioned to Lucia that the app does not have the read record function at the moment for connection and security reasons
   (a) Lucia concurred that this is satisfactory enough at this stage
   (b) The time-out/log-out functions were presented
   (c) Lucia asked about the notification sound
   (d) Lucia mentioned her security concerns, an application with only record creation function is simple and secure
3. Lucia promised to give us an update on the survey

B.4.4 Outcomes

The main features and UI for the app were shown, an update for the survey will be sent, team is continuing with the current app development schedule.

B.5 Progress Updates & Post-Holiday Arrangement (12/12/2013)

B.5.1 Meeting Purpose

Questions about data sets, ask for survey update, introducing the team blog, arrange next meeting

B.5.2 People Present

App Team 7, Lucia Gavalova

B.5.3 Discussion

1. Questions from David about complication data set
   (a) Lucia said she will take a look later
2. Asked for further survey updates
   (a) Lucia mentioned that the security concern stood out in the survey, tablets rather than mobile phones are used more, iOS devices are used more than other devices etc.
   (b) Lucia was also very encouraged that survey indicated that potential users are considering purchasing mobile devices for medical record use if the app works well
   (c) Lucia mentioned the difficulties when trying to convince potential users to complete the survey
3. Arrangements for the next meeting
4. Introducing our blog to Lucia
   (a) Lucia is generally happy with the contents of the blog, however she mentioned few amendments may be needed
   (b) Lucia said she will offer feedback to us once she has read it
5. Lucia suggested if we could turn off the auto remember/fill in function on the keyboard that Android provides on the app in order to protect patients confidentiality
   (a) Kamil explained to her that the function is offered by the native Android keyboard app not our app so we are unable to disable it
6. Ke explained a use case to Lucia with Kamil's story board

(a) Lucia suggested that the app could have a section which stores all the records that were created by the app, which allows the nurse go back to the app and continue to work on the records

7. David gave updates to Lucia, the team will go on holiday and work separately, gave a promise as to what each member of the team will deliver by the next meeting
   (a) David will be working on the local database and the validation rules
   (b) Ke will be working on how to communicate with the domino server
   (c) Kamil will be working on the UI, the navigation map may not be done by then
8. Reminded Lucia of the deadline of the project, informed her that what will be presented to her in the next meeting will be very close to what we will be handing in by the project deadline
9. Reminded Lucia that the development of the app helped the organisation in terms of mobilising the current web solution
   (a) Lucia agreed to view the app as a proof of concept

B.5.4 Outcomes

The next meeting is set for 10th Jan, 2014; the team will be working separately during holiday, client has agreed to view the app as a proof of concept.

B.6 Post-Holiday Meeting (13/01/2014)

B.6.1 Meeting Purpose

Give updates on the app, show the UI skeleton on Android phone , and gather feedback

B.6.2 People Present

App Team 7, Lucia Gavalova, Sue Manuel

B.6.3 Discussion

1. Remind Lucia the agreement between the team and client in terms of the app, the app is a proof of concept on the Android phones
2. Gave updates to Lucia

(a) Ke was researching for the communication of the Domino server, SOAP and WSDL as communication protocols were found, he asked for access for a test Domino database in case a real test is needed

(b) Lucia promised to hold another meeting with tech team to sort that out

(c) David is working on the database, it was not fully implemented, but nearly done

3. Kamil presented the UI skeleton on Android phone and installed it on Lucias phone in order for her to try it out and offer feedback as quickly as possible

   (a) Lucia is pleased with the skeleton of the app

   (b) Lucia raised a question about just keeping the side bar as a main navigation section instead of a separate navigation page

   (c) Kamil explained the thinking behind it to help users that are accustomed to the web page solution to navigate their way in the app

4. Sue came into the meeting

   (a) Sue was presented the app skeleton as well

   (b) She asked for handover documents, team promised they will produce them

B.6.4 Outcomes

Client is happy with the skeleton of the app as well as the outcome of the app, handover documents will be provided for the future development of the app, a meeting will be set up with the tech group for Kes research on the domino communication.

B.7 Domino Communication (17/01/2014)

B.7.1 Meeting Purpose

To show the research findings to the tech group and find out more information about Domino communication protocol

B.7.2 People Present

Ke, the tech group(Fabian, Andy, Nadeem, David)

B.7.3 Discussion

1. Ke presented the research findings to the group
2. the group thought the research is going in the right direction, and more time is needed before testing
3. one of the tech guys asked if third party plugins need to be installed on the server, because for security reasons they cannot do that
   (a) Ke assured the tech team that no plugins or any types of software will be installed on the server
4. The tech team showed their interest to make use of the web service functions that are provided by Domino
   (a) Fabian promised to send more information regarding the Domino web server to Ke
5. Tech team told Ke once he thinks a real test is needed he should come back and they will arrange it

B.7.4 Outcomes

The tech team agreed that the research is going the right direction and looking forward to see further results from it, the research continues, depending on the progress of the research, further meeting will be arranged.

B.8 Device Survey

Draft proposal: To determine how many target medical personnel own smartphones (or considering purchase in the near future), and would use it in a work setting. + find out specific OS and screen sizes via requesting visit of a site on their mobile (loaded with Google Analytics).

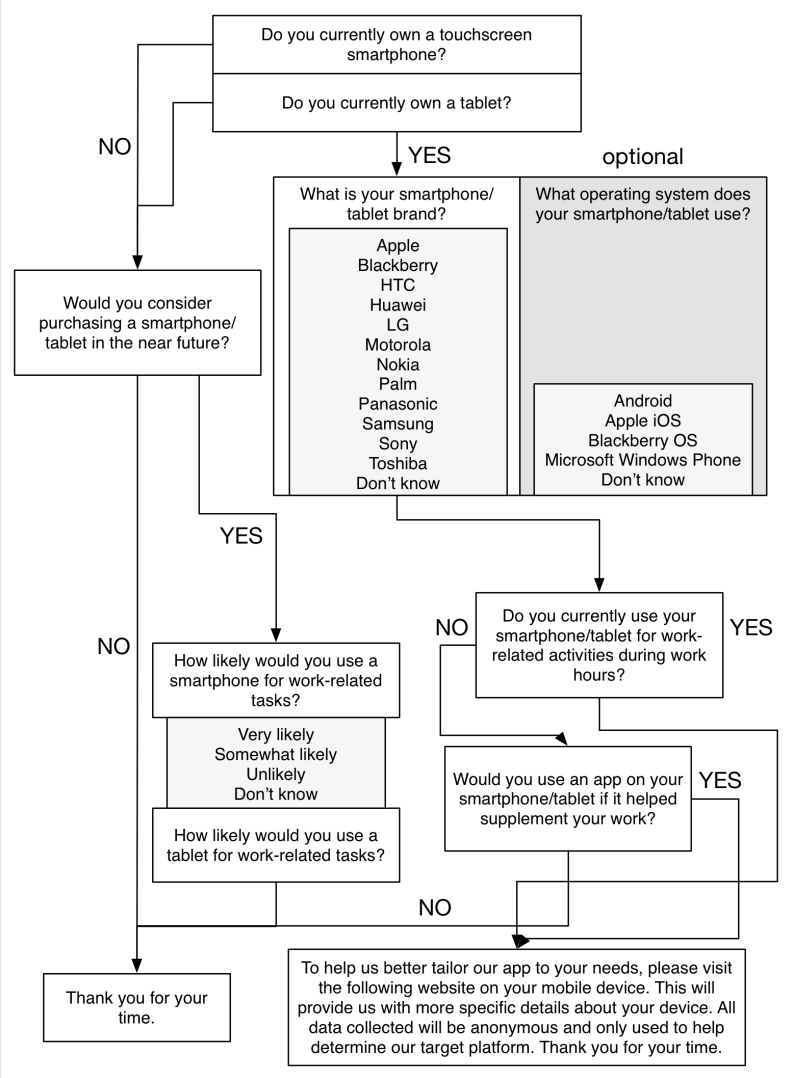


Figure 12: Initial survey flow

It was decided through client feedback that the survey flow logic could be replaced with a simple one-page linear survey due to the straightforward nature of the questions. The contents of the final survey are below.

Final SurveyMonkey Survey

1. Do you currently own a touchscreen smartphone or a tablet?
   (a) Smartphone only
   (b) Tablet only
   (c) Smartphone and tablet
   (d) Neither
2. Would you consider purchasing a smartphone/tablet in the near future?
   (a) Yes
   (b) No
3. What is your smartphone/tablet brand?
   (a) Apple
   (b) Blackberry
   (c) HTC
   (d) Huawei
   (e) LG
   (f) Motorola
   (g) Nokia
   (h) Palm
   (i) Panasonic
   (j) Samsung
   (k) Sony
   (l) Toshiba
   (m) Other (please specify)
4. What operating system does your smartphone/tablet use (if known)?
   (a) Android
   (b) Apple iOS
   (c) Blackberry OS
   (d) Microsoft Windows Phone

(e) Other (please specify)

5. Do you currently use your smartphone/tablet for work-related activities during work hours?

   (a) Yes

   (b) No

6. Would you use an app on your smartphone/tablet if it helped supplement your work?

   (a) Yes

   (b) No

7. How likely would you be to use a smartphone for work-related tasks?

   (a) Very likely

   (b) Somewhat likely

   (c) Unlikely

   (d) Don't know

8. How likely would you be to use a tablet for work-related tasks?

   (a) Very likely

   (b) Somewhat likely

   (c) Unlikely

   (d) Don't know

9. To help us better tailor our app to your needs, please visit the following website on your mobile device `http://goo.gl/KEevis`. This will provide us with more specific details about your device. All data collected will be anonymous and only used to help determine our target platform. Please feel free to add any comments.

C Internal Meetings and Documentation

Included below are some pictures of our internal meetings (documented on Trello). These are only a portion of the meeting 'minutes', and are meant to supplement the client meeting minutes and project schedule.

C.1 01/11/13

First meeting, the team started to think about the way to communicate with the Domino server, based on the information gathered from the first client meeting, Lotus Notes and XPages could be helpful, team still thinking about how to let Java and Domino communicate with each other. Using XML to store the validation rules, so it could be applied to not only Android, but also other OS.

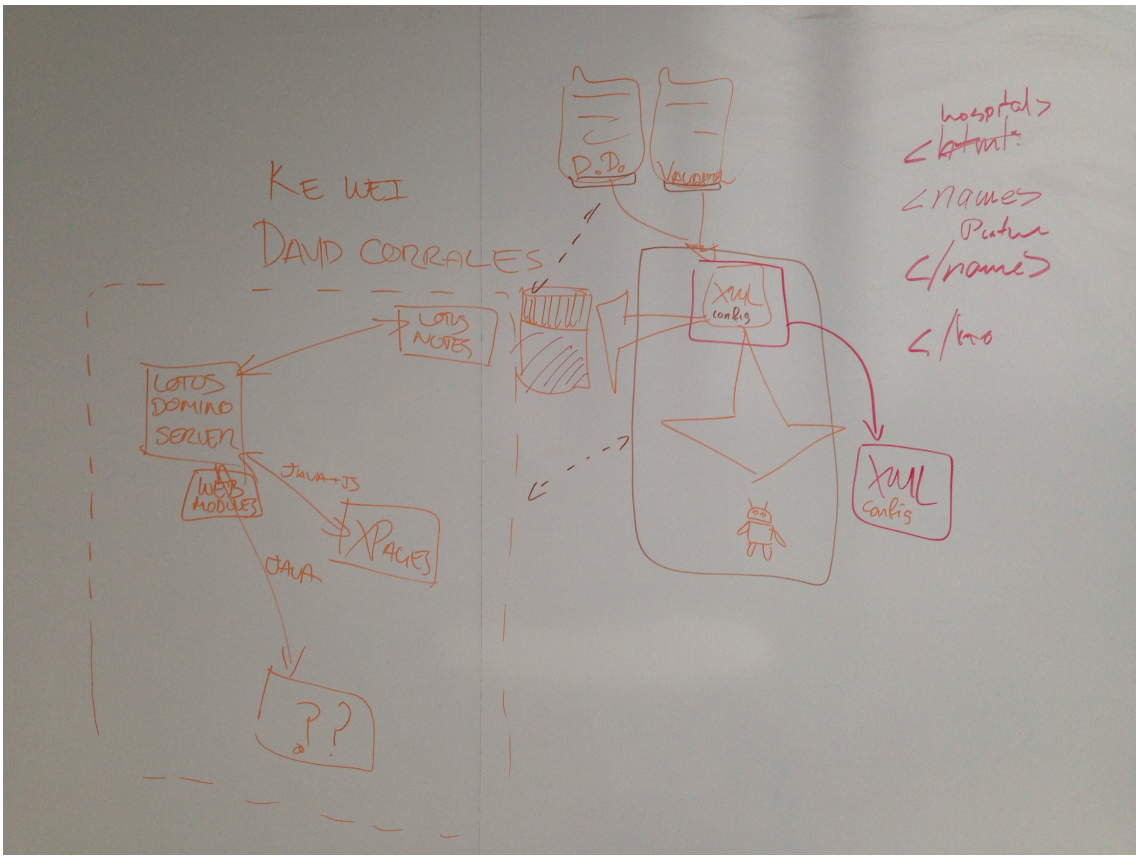


Figure 13: First meeting

C.2 04/11/13

Completed a basic page flow layout showing the links between each page. The user should login first, then move to the main page where he or she can choose to create or read a record, then based on the choice, record creation page or search page will be shown. Also completed the basic structure of functional parts.

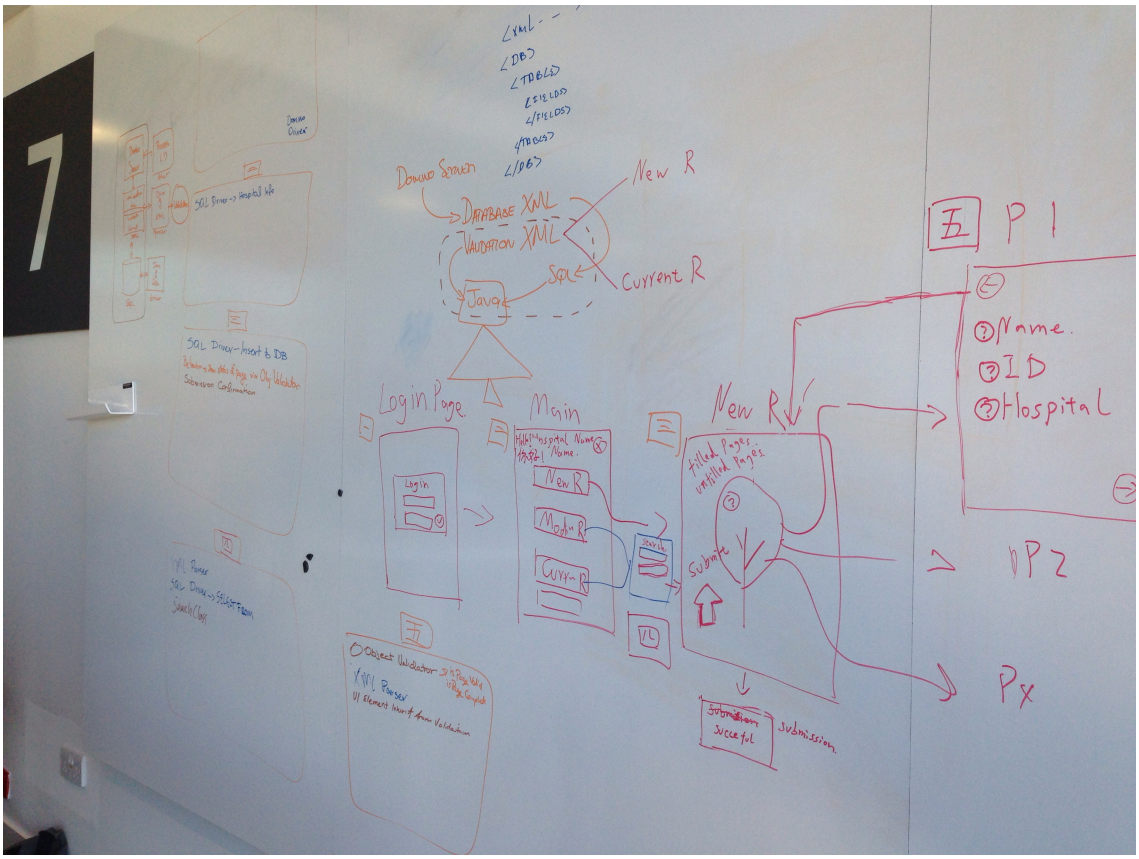


Figure 14: Basic page flow

C.3 05/11/13

Converting Mondays sketches into classes and deciding how these mock Classes would interact with each other based on the current software documents provided by the client, the new record will be checked for completion and validation before submission to Domino through Domingo as it was seen as a plausible option to network the clients Domino server to the app teams Java product.

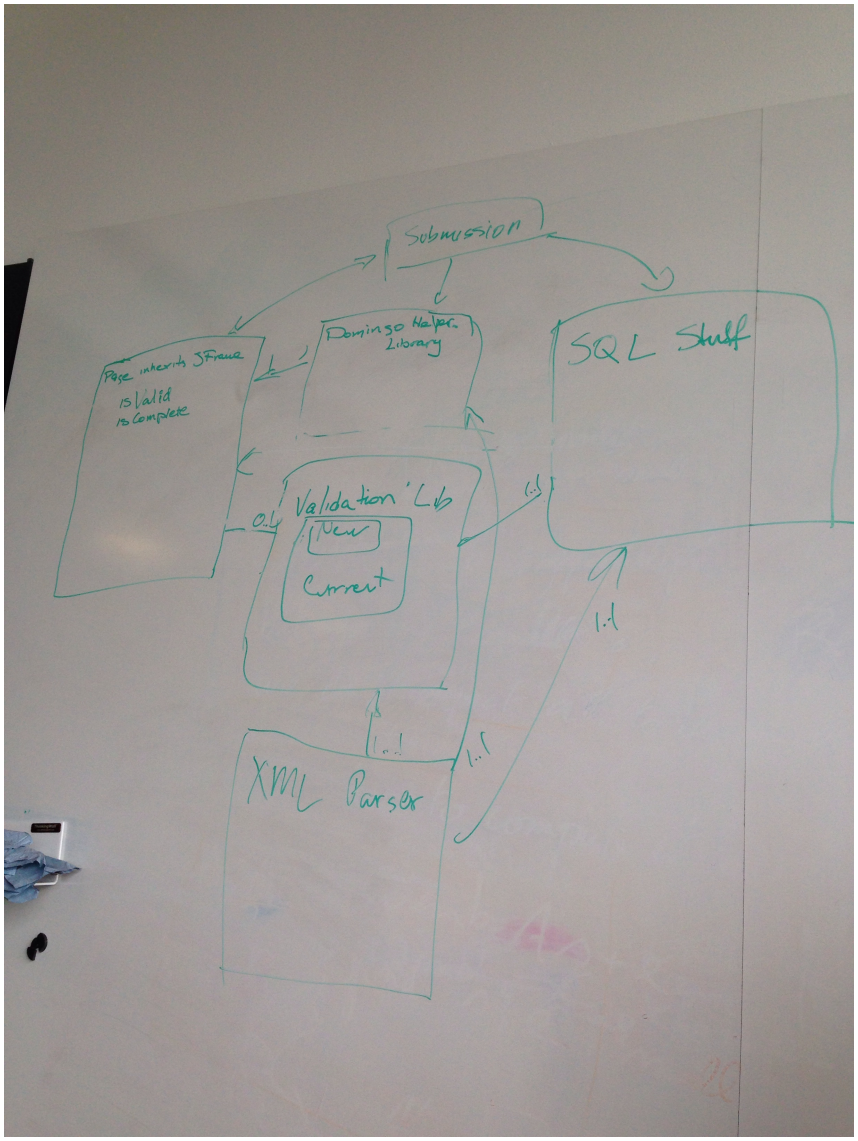


Figure 15: Mocking classes

C.4 06/11/13

Further work on the class construction, the team also started thinking about the structure for the XML Validation and parser.

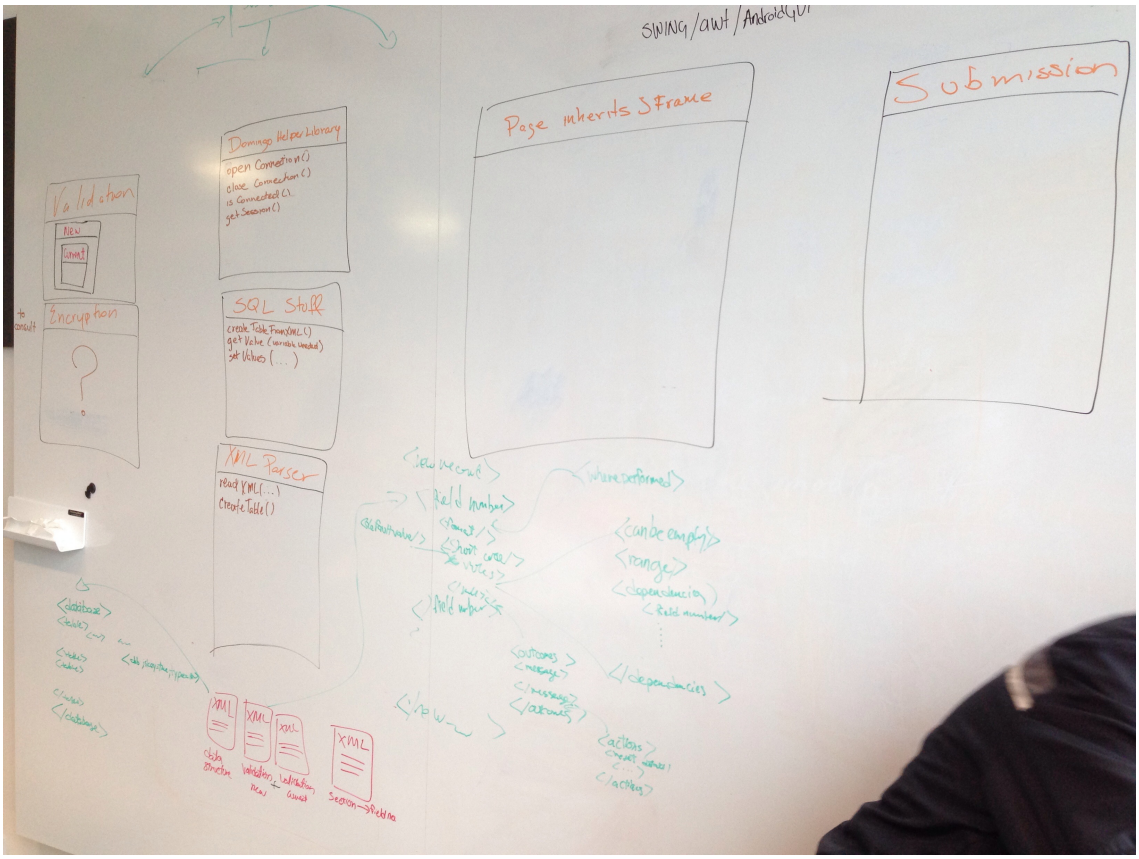


Figure 16: Key system architecture design

C.5 07/11/13

Logical steps for the product were discussed; from communication with the Domino server, to the display of a GUI on the chosen platform and back to the Domino server as a write operation.

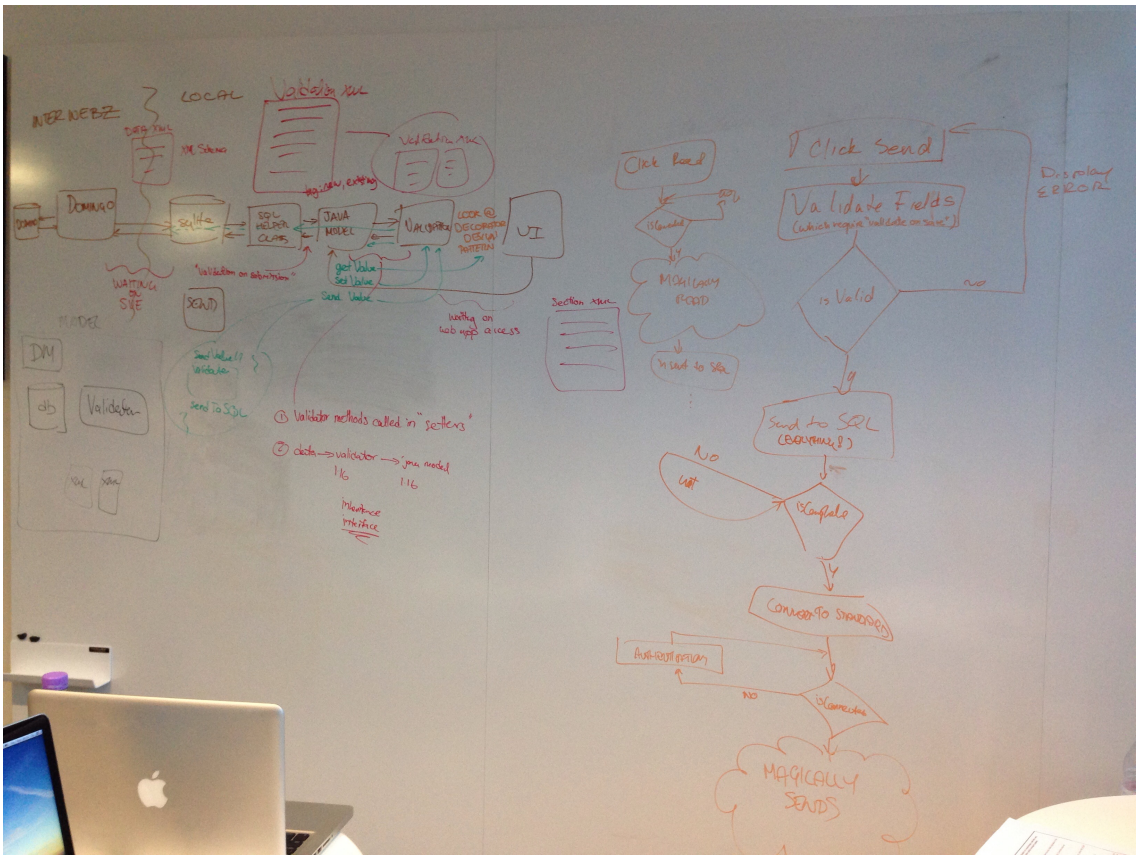


Figure 17: More planning

C.6 08/11/13

Continue with the class creation, the actual XML validation and connection to Domino were temporarily deferred, as they were time consuming.

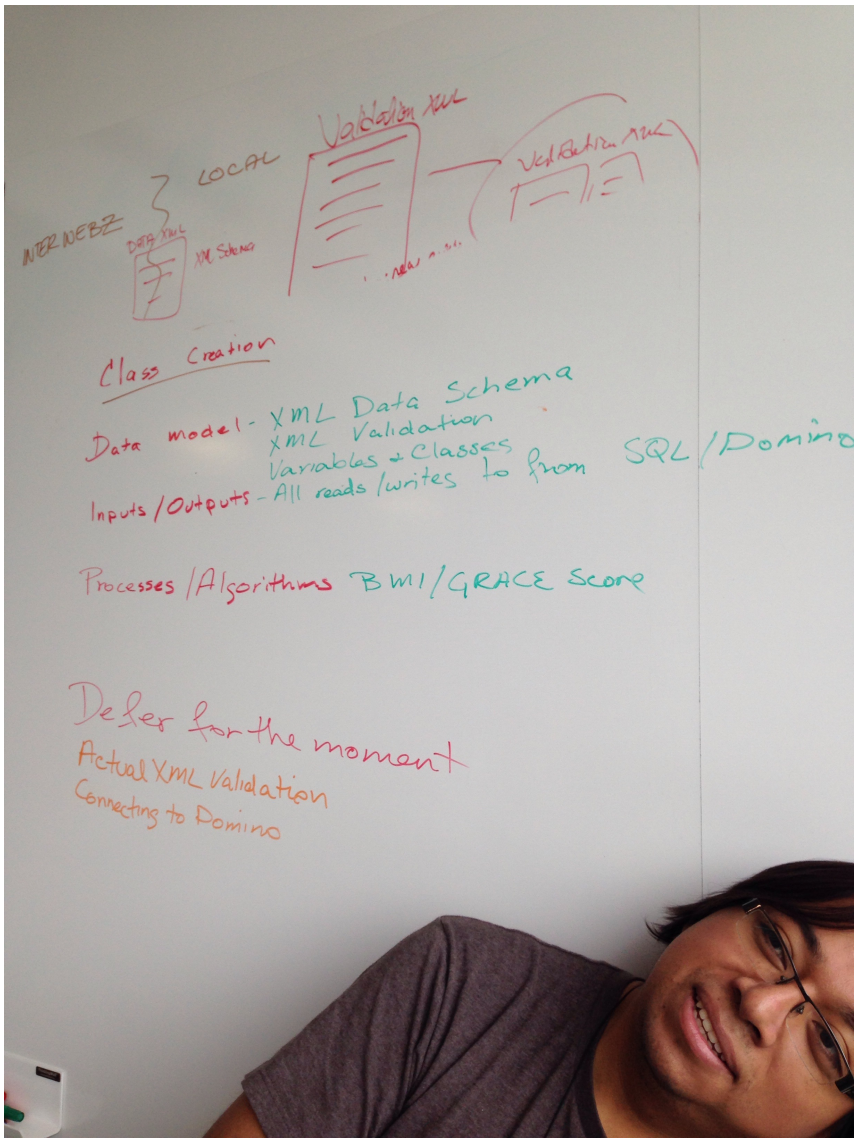


Figure 18: Dealing with XML

C.7 21/11/13

Team reviewed the page flow and communication flow of the app, the structure did not change much from last meeting.

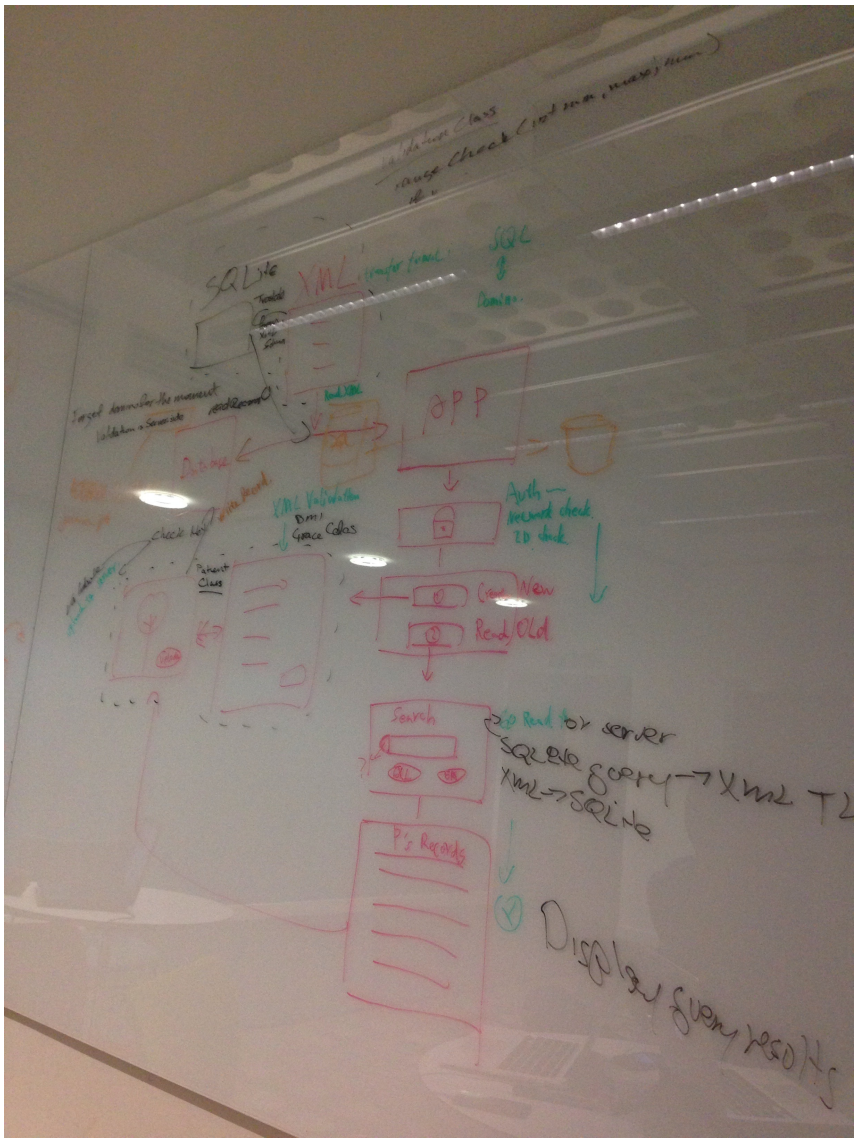


Figure 19: Page flow

C.8 26/11/13

After having received detailed sanity checks from client, team started to sort out the logic relations between data fields and pages, all the fields then were created in our Java prototype under Patient class.

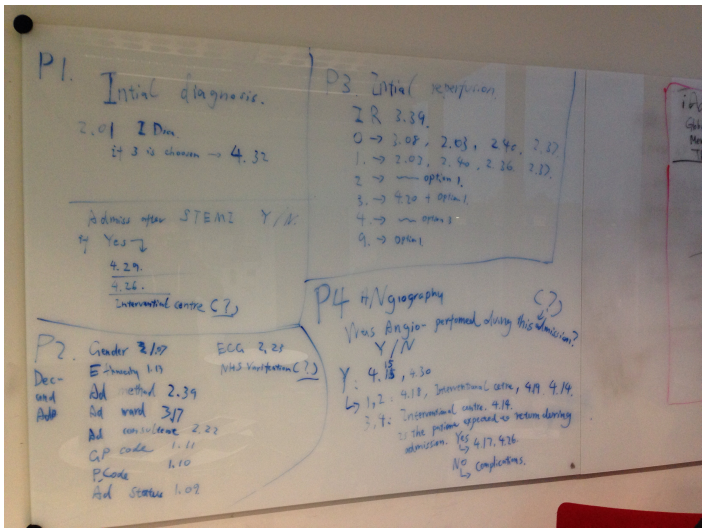


Figure 20: Validation pt.1

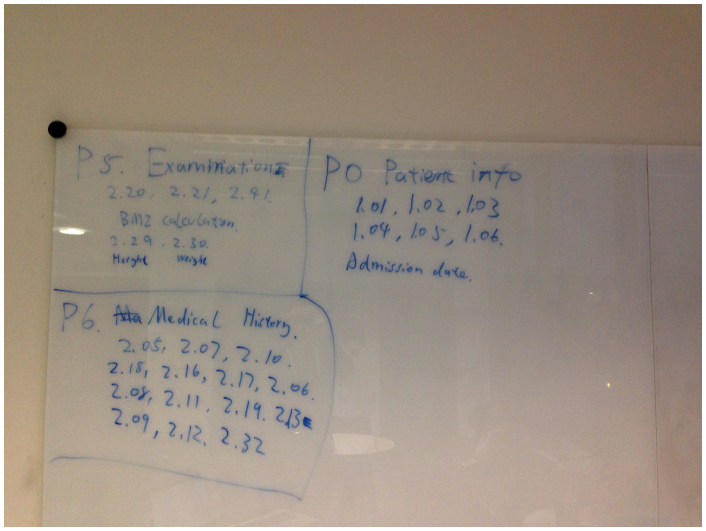


Figure 21: Validation pt.2

C.9 03/12/13

Drafting the poster for the app.

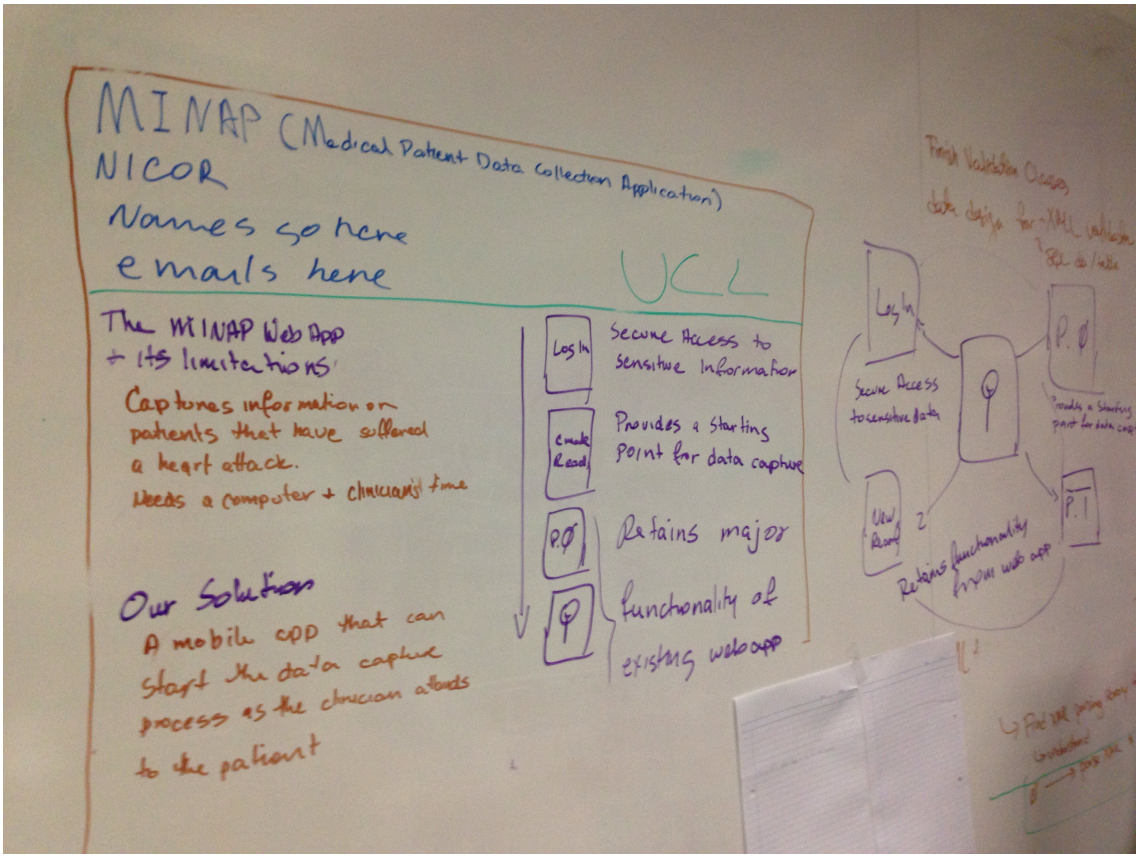


Figure 22: Poster planning