UNIVERSITY OF ST ANDREWS

SCHOOL OF COMPUTER SCIENCE

PRE -SURGICAL CHECKLIST MANAGEMENT SYSTEM

An android software application for managing pre-surgical checklists to improve patient safety

Adebayo I. Osipitan - 130024147

4/26/2015

Dissertation for the acquisition of an MSc degree in Software Engineering, supervised by, Dr. Thomas Kelsey

Declaration

Adebayo Ibiyemi Osipitan

26/04/2015

Acknowledgement

First and foremost, I would like to thank GOD, for his wisdom, grace, mercy and favour.

I wish to thank my supervisor, Dr Tom Kelsey for his advice, support and direction during the course of this project.

I also wish to thank Dr Peter Clark and Dr Colin Allison; former vice principal (governance) and the Pro-provost of Science respectively, who more than anything else lent a listen ear to me, and facilitated this project attempt. An expression of words cannot explain how grateful I am.

I wish to also acknowledge open access programming blocks a Finally, I must thank my parents, Mr Wale and Mrs Omolola Osipitan, who have been understanding and nurturing during the course this project

Abstract

Mobile technology has become a near essential utility in the lives of people today. Many sectors now utilize mobile technology and innovative application software to deliver better goods, services and solutions. This includes the health care sector, as this domain sector is gradually recognising that administration of health care can be made more efficient with mobile technology.

With health care being a complex domain, it is evident that the use of IT systems can help achieve better quality of care especially in the patient safety subdomain. Patient safety is a very delicate domain of health care, where harm may come upon a patient if clinical practices are not managed properly and speedily. Several studies have documented both the economic cost and patient harm resulting from human error. This is particularly true in high dependency units' e.g. surgical units, emergency units.

To this end, this project's client; a paediatric hospital, specialising in cancer patients, proposed a pre-surgical, checklist management system via a mobile software application as a replacement for the problematic paper based system currently in use.

This report details the development cycle of this project, from conception to submission, highlighting the aims, objectives, and milestone, as well as the software development approach taken to create this application. The type of IT system is one that facilitates information transaction i.e. integrates and transforms information from a source.

During the development life cycle, system assumptions and requirements were gathered from the client and analysed so as to identify the driving requirements to be used to form the core architecture of the system.

The architectural design developed on the basis of the requirement is *an object oriented-database* architecture and this report discusses the various architectural perceptions of the system.

The system implemented is such that it ensures timely discharge of key responsibilities and accountability on task completion, by recording the name of the checker and the date and time of the check.

The system developed was evaluated by representatives of the client, who expressed satisfaction with the features and functionality of the mobile application.

Table of Contents

Declaration 1

Acknowledgement 2

Abstract 3

Terminologies and Abbreviations 7

1 Introduction 8

1.1 Overview 8

1.2 Motivation 8

1.3 Project Aim 9

1.4 Objectives 9

1.5 Document Outline 9

2 Literature Review 11

2.1 Patient Safety 11

2.2 Improving Patient Safety Using IT systems 12

3 Project Methodology 14

3.1 Project Management 14

3.2 System Development Life Cycle 15

3.3 Project Schedule 17

4 System Requirement Specification 18

4.1 User Group 18

4.2 Assumptions 18

4.3 Functional Requirements 19

4.4 Non-functional System Requirements 21

4.5 Priority Requirements 21

5 System Design 23

5.1 System Components 23

5.2 System Architecture 23

5.3 System View 25

5.3.1 Logic View 25

5.3.2 Process View 27

5.3.3 Development View 30

5.3.4 Physical View 31

5.3.5 Use Case Scenario View....32
5.4 Major System Design decision....35
6 System Implementation....37
6.1 System Component Implementation....37
6.1.1 The Database....37
6.1.2 Database Management....42
6.1.3 The Mobile Application....47
6.2 System Integration Implementation....48
6.2.1 Task Completion....48
6.2.2 Accountability....52
6.2.3 Human Error Reduction....55
6.2.4 Usability....55
6.3 Interface Layout Implementation....56
7 System Testing and Evaluation....60
7.1 Unit Testing....60
7.2 System Integration Testing....60
7.3 System Evaluation....61
7.3.1 Usability....61
7.3.2 Utility....62
7.3.3 Participants....63
7.3.4 Evaluation Materials....63
7.3.5 Evaluation Procedure....65
7.3.6 Evaluation Results....65
8 Conclusion....68
8.1 Critical Appraisal....68
8.1.1 Key Achievements....68
8.2 Future Works....69
8.3 Summary....69
9 Reference....70
Appendix A – System Usability Scale (SUS) Questionnaire....72
Appendix B – Custom Utility Questionnaire....73
Appendix C – Custom free form Questionnaire....74

Appendix D – User Manual 75

Terminologies and Abbreviations

- **C.R.U.D.:** Abbreviation for the four types of database management operations; Create, Retrieve, Update, Delete.
- **Class:** Schema or blueprint from which individual objects can be created (Oracle 2015).
- **Object:** This is an instance of a class that inherits the state and behaviour of such a class.
- **I.D.E:** Integrated Development Environment.
- **X.M.L:** Extensible Mark-up Language
- **SQL:** Structured Query Language.
- **PhP:** Hypertext Pre-Processor
- **UML:** Unified Modelling Language
- **I.T:** Information Technology
- **VB:** Visual Basic.
- **S.U.S.:** System Usability Scale
- **I.O.M.:** Institute of Medicine
- **I.o.T.:** Internet of things

1 Introduction

1.1 Overview

Mobile technology and software application development together, are an ever expanding technological sector. New mobile applications continue to be launched worldwide to address all manners of general and special needs. From mobile applications that help organise personal, daily activities and tasks, to applications that help learning experiences etc. with the list so endless it is more than evident the impact of mobile technology on lives today. The ease of access to knowledge that mobile/application technology brings has also encouraged exponential and fast growth in various economic sects e.g. banking, agriculture, oil and gas, merchandising (via online transactions), tourism to mention a few.

In the health sector, the impact of technology in revolutionizing the way healthcare is delivered, is well known and now, mobile applications are making in-roads into the sector and both the professionals and patients, are experiencing the benefits. According to ***Evidence-Based Medicine Online First***, it states [MP1] by the end of 2015 over 500 million smart-phone users worldwide will be on one form of medical app or the other (Buijink et al 2012).

In this age, patient safety is of dire importance at any high dependency care units, and could lapse for any number of reasons, including adverse events, staff negligence or miscommunication amongst members of staff or some other form of human error. These could lead to patient harm or in severe scenarios, death of a patient. This project leverages a solution to help reduce human errors that have an influence on affecting patient safety when managing pre-surgical patients, in the form of a mobile software application, using the features of today's mobile devices to capture healthcare information and improve accountability and increase staff communication.

1.2 Motivation

With the activities of professional practitioners (doctors, nurses, physicians etc.) being complex and overwhelming is some situations. Discontinuities in patient care could easily occur, either in form of loss of information or interruptions of care for whatever reason (Cook et al 2000). This can result in patient harm or death in extreme cases. Medical care today, has also strived to individualise care to a much higher degree than times past, but this adds more complexity to the domain for practitioners increasing the chances of human error occurring. As such, this necessitates the development of technological systems that will help reduce operational risks and errors, assisting in decision making, error corrections and provision of performance feedback (Bates et al 2003).

In the same vein, the pre-surgical management of patients is highly associated with patient safety, ensuring certain steps are taken before proceeding with patient operation in the theatre. This has prompted the clientele, a paediatric hospital, specializing in clinical surgery, to propose a mobile software system to replace the paper based system currently in use in their facilities, in other to improve patient safety by reducing human error and increasing accountability.

1.3 Project Aim

The aim of this project is to design, develop and implement a concept for a mobile patient safety software application, in the form of a pre-surgical patient checklist system. This software application is developed for a paediatric hospital, in order to facilitate better healthcare delivery, by reducing the human error and poor accountability that is common place with paper based systems currently in use.

1.4 Objectives

In order to achieve the aim stated above, the following project objectives were inferred;

- To develop a facility that allows the client to schedule patient treatment
- To develop auto-generation of required pre-surgical checklists questions, for the procedure to be performed per patient.
- To develop input facilities for answering the checklist questions.
- To develop a database that records inputs for the checklist question.
- To develop an accountability system in the application that automatically records the identity of the checker and the time stamp on updating the checklist.

1.5 Document Outline

Following this section, the report is structured as follows;

- **Section 2:** This section discusses research on the problem domain, like patient safety, as well as relevant software technologies developed for improving patient safety
- **Section 3:** discusses the project methodology which includes the project management, its development life cycle and schedule for achieving tasks and milestones.
- **Section 4:** discusses the system requirement specifications for the application, which includes identifying user groups, system assumptions, functional and non-functional requirements of the system.
- **Section 5:** discusses the system design, design perspectives and design designs undertaken during system development.

- **Section 6:** discusses the implementation process of developing the application system according to the design specifications of the previous section.
- **Section 7:** discusses the development testing processes as well as the system evaluation process conducted on the system.
- **Section 8:** concludes this report by critically appraising key achievements as well as limitations.

2 Literature Review

This section of the report discusses the context survey surrounding the problem domain of this project and related work relevant to the domain solution.

2.1 Patient Safety

Patient safety is a domain of health care services, where errors in services could lead to patient harm. The Institute of Medicine IOM defines it as;

"*The prevention of harm caused by errors of commission and omission.*" (Henneman 2010)

Patient harm is defined as the injury, disability, suffering or death of a patient (NHS 2014). In years past, focus has been on the cause and the effect of adverse events on patient, but in this technological age, and global awareness, more scrutiny has fallen on healthcare practices and their effects on the safety of patients. According to L.L. Leape et al (2002), epidemiological findings from decades of study, show that lapses and errors in medical practices amongst professionals caused up to 100,000 deaths and one million injuries in the United States annually. In the UK, surveys conducted by the National Patient Safety Agency in August of 2006 and again in February of 2007, on high care dependency unit, recorded patient safety incidents of over 12000 from various health organisations, these incidents were associated with several care practices such as drug administrations or drug prescriptions etc. (Thomas and Panchagnula 2008), some of these incidents were reported to cause serious harm to patients. These incidents are as a result of discontinuities in the quality and safety of patient care.

According I.O.M. definition these discontinuities were found to be associated with miscommunication between staff members, resulting in errors of care commission. For example, medication administered either at the wrong time or in the wrong dosage, this type of error is more common place.

Sudden interruptions of patient care could also be an associated discontinuity, resulting in errors of care omission where necessary treatment is not carried out entirely (Henneman). In most cases discontinuities in patient care are identified early enough to avoid harmful consequences, but with the medical domain being such a complex and momentous environment, patient safety can be at a tremendous risk if these discontinuities remain unidentified. As such, continuous research is being conducted to determine effective practices to improve patient safety, including the use of IT technologies as a means of managing care administration.

2.2 Improving Patient Safety Using IT systems

As stated above, health care is a complex domain, where the quality of care is of uttermost importance, and to achieve a higher quality of care, the use and improvement of IT systems will play an integral part. IT systems have already been proven to increase the efficiency in producing goods and administering services in other major domain sectors such as banking, and education, to satisfy the specific needs of individual customers.

Medical care today, has also strived to individualise care to a much higher degree than times past, but this adds more complexity to the domain, increasing the stakes. This facilitates the need of computerised decision support. The growing sophistication of software and IT systems can help reduce operational risks and errors, assisting in decision making, error corrections and provision of performance feedback (Bates et al 2003).

Focusing on the scope of this project, IT systems that facilitate information transactions i.e. software systems that integrate and transform information from a source e.g. electronic patient records, can help change the existing clinical and administrative process in health care, and can also help address problems such as;

- Access to information by health care workers in developing countries, enabling effective care in such regions.

The advantage of adopting Information Technology in health care and safety is that as this type technology evolves, so does health care. With the evolution of mobile technology, there are now more IT system options to provide solutions to the domain problems of patient safety.

2.3 Related Work

Attempts have been made to develop systems that are centred on patient safety,. An example of such as system was a "Medication Error Information Report" web based system developed for University of Mississippi Hospitals and Clinics (UMHC). This system collected occurrence reports which were used to collect information on all medication errors or mistakes regardless of whether or not they resulted in patient harm. The goals of this Web-based medication error reporting system were to increase the number of medication errors reported, to increase the number of intercepted errors, and to decrease the number of adverse events. According to results on evaluating the usefulness of this system for three years (2003 -2005), the system was successful as more error incidents were reported by staff members relating patient medication (Brown et al 2005).

3 Project Methodology

This section of the report discusses the methodology adopted for the development of the system application during the course of the project.

3.1 Project Management

The approach adopted to manage the progress of this project was, that of an iterative management approach. With this approach, the project was broken down into several stages, and these stages were progressively completed. These stages are highlighted below;

- **Project Initiation:** At this stage the initial concept for developing a mobile application system as a patient checklist for a hospital was conceived, along with the aims and objectives of the project.
- **Project Planning:** This stage of the project entails, gathering system requirements from the client, outlining milestones, as well as tasks to be accomplished and generating a project plan. Also, in the stage of the project, research was conducted into development of similar system artefacts (mobile application) as that of the project system in terms of techniques for data management for mobile devices.
- **Project Execution and Monitoring:** This next stage of the project entailed the iterative design and implementation of the application system. The Design sub stage involved;
  - ➢ *System component Identification:* Selection of the most appropriate of possible system components that could be used in developing the system based on the collected system requirements in the previous stage.
  - ➢ *System Architecture design*: designing a suitable architecture for this system, by identifying the constraints of the project system, prioritization of the system requirements in order to properly identify the architectural drivers of the system, then identifying the most suitable of software architectural styles and patterns that will easily capture the earlier identified system drivers and allow the system components to easily interact with each other. On identification of these software architectural entities, A design package and language (Unified Modelling Language in this case) was used to visually represent the various perspectives or views of the system by the creating models of the system operation and interaction.
  - ➢ *System Implementation:* The design above serves as input for developing a conceptual system for the project (i.e. a prototype), this is then evaluated by both the client and the colleagues, to obtain feedback on functionality applicability and

interface design. The implementation is also done iteratively based on such feedback which helps refine the system requirements for further improvement of the system. This is done as a continuous cycle till the project period comes to a close, along with appropriate supervision that ensures the project is still within scope.

- **Project Closing:** After iteratively testing and evaluating the system on each developed version, a final version of the application system and its entities is built for deployment. Documentation on the development process for this project will also be completed at this stage as a deliverable.

The cycle of these stages is demonstrated in the figure below;

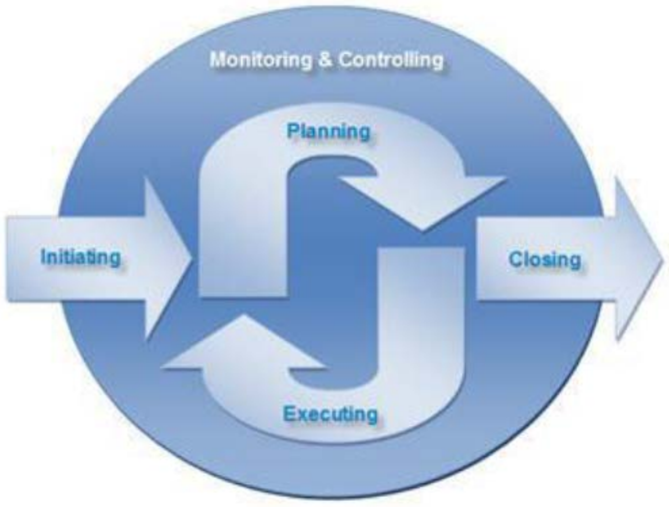


Figure 3-1: IT Management Approach (Courtesy: McGill University 2013)

3.2 System Development Life Cycle

The development life cycle chosen for this project was that of an iterative waterfall model, this is an agile life cycle to adopt for the system methodology because of its simplicity and ease of use, it is also easy to manage and allows for stages to be completed one at a time. Least of all, as this project serves a proof of concept for an application system and will still most-likely require further improvement for more scalable deployment capability, the waterfall development life cycle is best suited. The stages in this cycle include;

- **Requirement Specification and Analysis:** To address the problem domain of this project, extensive research was conducted on android mobile application development; relevant mobile applications that help improve patient safety and also information management on mobile devices. Then system requirements for this project were gathered from the client who is an expert in the medical domain.

- **System Design, and Component Acquisition:** After requirements for this systems are gathered, then decisions on component identification, analysis and acquisition were made based on the gathered requirement. This lead to the formulation of an architectural design for the system.
- **Design implementation:** After a design has been formulated implementation of prototypes commences. The system components are developed and integrated together to create the overall application system.
- **System Evaluation and Testing:** This stage of the cycle, tests the system components after implementation and integration. Once a prototype has been created, further testing/evaluation is carried out to test the interaction between components of the system. Feedback is then generated as a result of such evaluation process, and further refinement can be carried out on such basis.
- **System Maintenance and Future Works:** The feedback generated on testing/evaluating of each system implemented iteration, helps in maintaining the system and also planning future works (in terms of requirement refinement, design, version implementation) for the system.

This software development lifecycle can be illustrated in the diagram below'

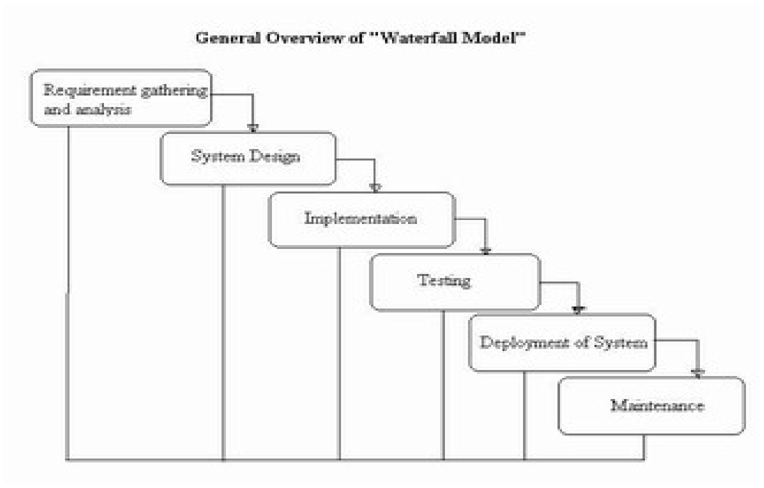

General Overview of "Waterfall Model"
Requirement gathering and analysis
System Design
Implementation
Testing
Deployment of System
Maintenance

Figure 3-2: Iterative waterfall lifecycle model (Courtesy: One Stop Testing 2015).

3.3 Project Schedule

The table below shows the various tasks/Milestones for this project:

S/N	Task/Milestone Title	Date Completed
1	Project Start	27/01/215
2	Research and Project Planning	10/02/2015
3	System Requirement Gathering	03/04/2015
4	Software Design	03/04/2015
5	System Implementation	26/04/2015
6	Project Client Demonstration and evaluation	24/04/2015
7	System Documentation (Dissertation)	26/04/2015

Table 3-1: Project milestone schedule

4 System Requirement Specification

Before designing any software system, the requirements of such as system must be gathered from the client. Additionally the user groups of the software system must be identified to better capture the system's requirement.

4.1 User Group

In the case of this project, there is only a single user group for this system application, which is the client. The client is a paediatric hospital that specializes in cancer treatment, and is based in Edinburgh, and the client representative is Dr. Hamish Wallace. They simply use this system as replacement of the obsolete paper based method of managing patient data in form of a checklist, they can register patients, set reminders to administer certain treatments, and answer routine questions pertaining pre-surgery checks. The result of this requirements gathering are the system's software architectural drivers that will help identify suitable system components and define the software architecture of the system.

4.2 Assumptions

Before gathering requirements, first the assumptions on the system being developed must be identified, they are as follows:

- The system is to be used to manage patients that have been already booked for surgery.
- The hospital/client will have a separate system for booking patients into wards pre and post-surgery.
- The details of all authorized personnel for this application system will be provided to serve as system input for data access.
- The hospital/client will have a separate system for discharging patients.
- All treatment facilities, their stock, and use are recorded in another system.
- The system is to be used locally i.e. within the hospital grounds
- The android mobile devices in used in the hospital have the latest operating system that can support the application system.

4.3 Functional Requirements

On identifying requirements of this software system, these requirements are also ranked on their importance on the system's functionality implementation; this is based on a three level ranking system – High, Medium and Low. This helped prioritise the system requirements and helped identify requirements that were considered the architectural drivers for the system development.

Requirement Specification Number	Requirement	Ranked Importance on the System
FR001	The system must only allow authorised hospital staff personnel to access patient data via logging in, on application launch	High
FR002	The system must have a storage facility to record information gathered during system operation	High
FR003	The system must record which hospital staff personnel logged into the system	High
FR004	The system must record what changes hospital staff personnel made to patient data.	Medium
FR005	The system must allow hospital staff register a new patient on the system	Medium
FR006	The system must all hospital staff to view and manage already register patient's data on the system	High
FR007	The system must allow the hospital staff choose between registering a new patient or reviewing and managing already registered patient data	Low
FR008	The system must allow the hospital staff to select a specific patient out of a cache of registered patients to manage	High
FR009	The system must allow hospital staff to search for patients as means of selecting desired patient	Medium
FR010	The system must allow hospital staff to add a specific treatment for a certain patient	High
FR011	The system must allow hospital staff to set a	High

	reminder for a specified treatment for a specific patient	
FR012	The system must alert hospital staff on set date and time to administer a logged patient treatment	High
FR013	The system must allow hospital staff to answer routine questions for pre-surgery checks for specific patient	High
FR014	The system must allow hospital staff to record if patient needs to be booked in a ward	Medium
FR015	The system must allow the hospital staff to check which of the other staff members altered patient data on the system	Medium

Table 4-1: Functional Requirements of the application system

4.4 Non-functional System Requirements

These are the requirements of the system that concern the constraints of the system operation. The specifications of these requirements are ranked as above.

Requirement Specification Number	Requirements	Ranked Importance on the System
NFR001	The system must work seamlessly on android mobile devices	High
NFR002	The system User Interface (UI) must be user friendly	Medium
NFR003	The system UI must have a layout and display that are easy to read and understand to carry out tasks.	High
NFR004	The system must only be accessible to authorised hospital staff members	High
NFR005	The system must all authorised hospital staff to manage patient data.	High

Table 4-2: Non-functional Requirements

4.5 Priority Requirements

From the requirements specified above, there was a need to prioritize some of these requirements against others, so as to determine which of these requirements will serve as the drivers for developing the software architecture of the application system. The driving requirements are listed below;

Requirement Specification Number	Requirements	Ranked Importance on the System
NFR001	The system must work seamlessly on android mobile devices	High
FR001	The system must only allow authorised hospital staff personnel to access patient data via logging in, on application launch	High

FR002	The system must have a storage facility to record information gathered during system operation	High
FR005	The system must allow hospital staff register a new patient on the system	Medium
FR006	The system must all hospital staff to view and manage already register patient's data on the system	High
NFR003	The system UI must have a layout and display that are easy to read and understand to carry out tasks.	High
FR008	The system must allow the hospital staff to select a specific patient out of a cache of registered patients to manage	High
FR010	The system must allow hospital staff to add a specific treatment for a certain patient	High
FR011	The system must allow hospital staff to set a reminder for a specified treatment for a specific patient	High
FR012	The system must alert hospital staff on set date and time to administer a logged patient treatment	High
FR013	The system must allow hospital staff to answer routine questions for pre-surgery checks for s specific patient	High

Table 4-3: Priority requirements that serve as architectural drivers for the system

Other requirements were also further implemented for the application system, once these driving requirements were used to derive a suitable architecture and were implemented into the system.

5 System Design

In developing any software for any system, there must be a defined architecture in which such software attains its structure. This chapter of the report will focus on the design and design decisions that were adopted in developing the architecture of this system. This chapter will discuss how the requirements, as well as the attributes of the system were used as architectural drivers for the design, as well present Unified Modelling Language (UML) models of the system function and operations.

5.1 System Components

The identification and selection of system components for developing this mobile application is a crucial aspect of the system design. First the constraints of the design must be identified, and these are as follows;

- ➢ The application must be downloadable onto android devices.
- ➢ The application must run seamlessly on most android mobile device
- ➢ Patient information must be easy to store and retrieve.

With these constraints in mind the components of the system were easily identified. There two main components of the software system;

- **The Database:** This is a structured repository that stores all information entered used by the application system.
- **The mobile Application:** This component houses both the user interface for content creation, modification and management and the local functions for database access and management.

5.2 System Architecture

Taking into consideration the priority requirements for the system, design constraints and the system components, the next step is developing an acceptable architecture that integrates the components together, while meeting the priority requirements and without violating the highlighted constraints (see sub-chapter 5.1). The system architecture allows authorised system users to input patient data and stores such data in an internal database, housed in the mobile device itself; it also allows such users to retrieve this data as at when required. This data management service is handled by the application itself; having local functions that perform C.R.U.D. (Create, Retrieve, Update, and Delete) operations on the database. The display of the data is handled by the interface which is also local to the android device. In other words, for this concept design, the architectural style decision for this system is that of an *Object Oriented* design with a database and the architectural pattern is that of M.V.C. (Model – View- Controller).a

- **The model**: This pattern component, houses classes of the application that control the behaviour and manage data of the system application domain by the use of local functions; in this case, these include; C.R.U.D operations for managing the database as well Object classes of the system application, by responding to function requests.
- **The Controller**: This pattern component, houses the UI classes that handle the interface events, like button click, or list item clicks etc. these handled events then call functions from classes housed in the model components perform various tasks such as changing the state of the model.
- **The View**: This just simply manages the display of information retrieved, or how information can be inserted into the system.

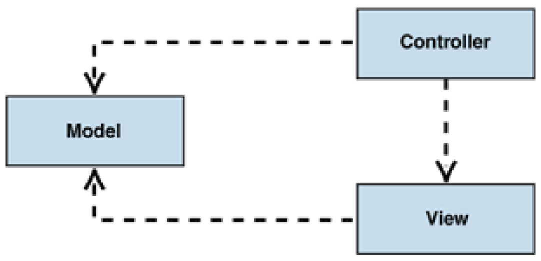


Figure 5-0: MVC Class Structure (courtesy: MSDN 2015)

Usability design decisions include, allowing for ease of use when the user in navigating through the application, and accomplishing task. This is accomplished by using simple layouts with visible hints, and making task completion achievable with minimal number of steps.

5.3 System View

This entails modelling the system application, respective to various view points, these model structures give better depiction on how the system components, style and pattern come together. The view pattern chosen for system application view of the architecture is the “4+1” view, expressed using U.M.L models, this has five view perspectives. These views include;

- Logical view
- Process View
- Development View
- Physical View
- And, Use case View.

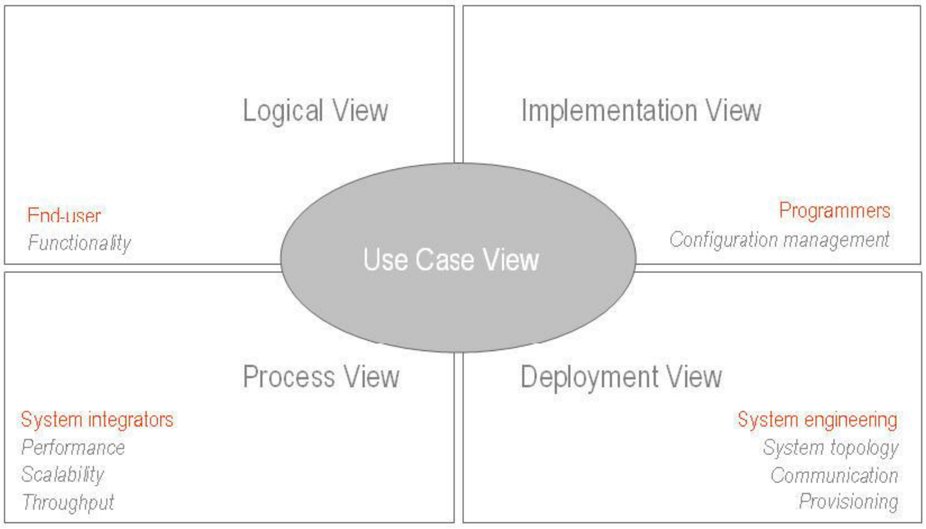


Figure 5-1: 4+1 view Architecture model (J Parnitzke 2009).

The modelling was done using Unified Modelling Language (UML), these models and the view they represent and are discussed below.

5.3.1 Logic View

For an object oriented design such as the architecture used for this project, the logic view is perfect to give perspective on the object oriented decomposition of the application system (Kruchten 1995). This architectural perspective is modelled using UML *Class Diagram* model and notations and can be seen in the figure below. The first class diagram below shows the classes in the model, and their logical relationship and the properties of the system application objects like; Patients, Answers/Comments, Login logs, Treatment logs, Database Helper etc., how these classes and the corresponding objects are called within the model.

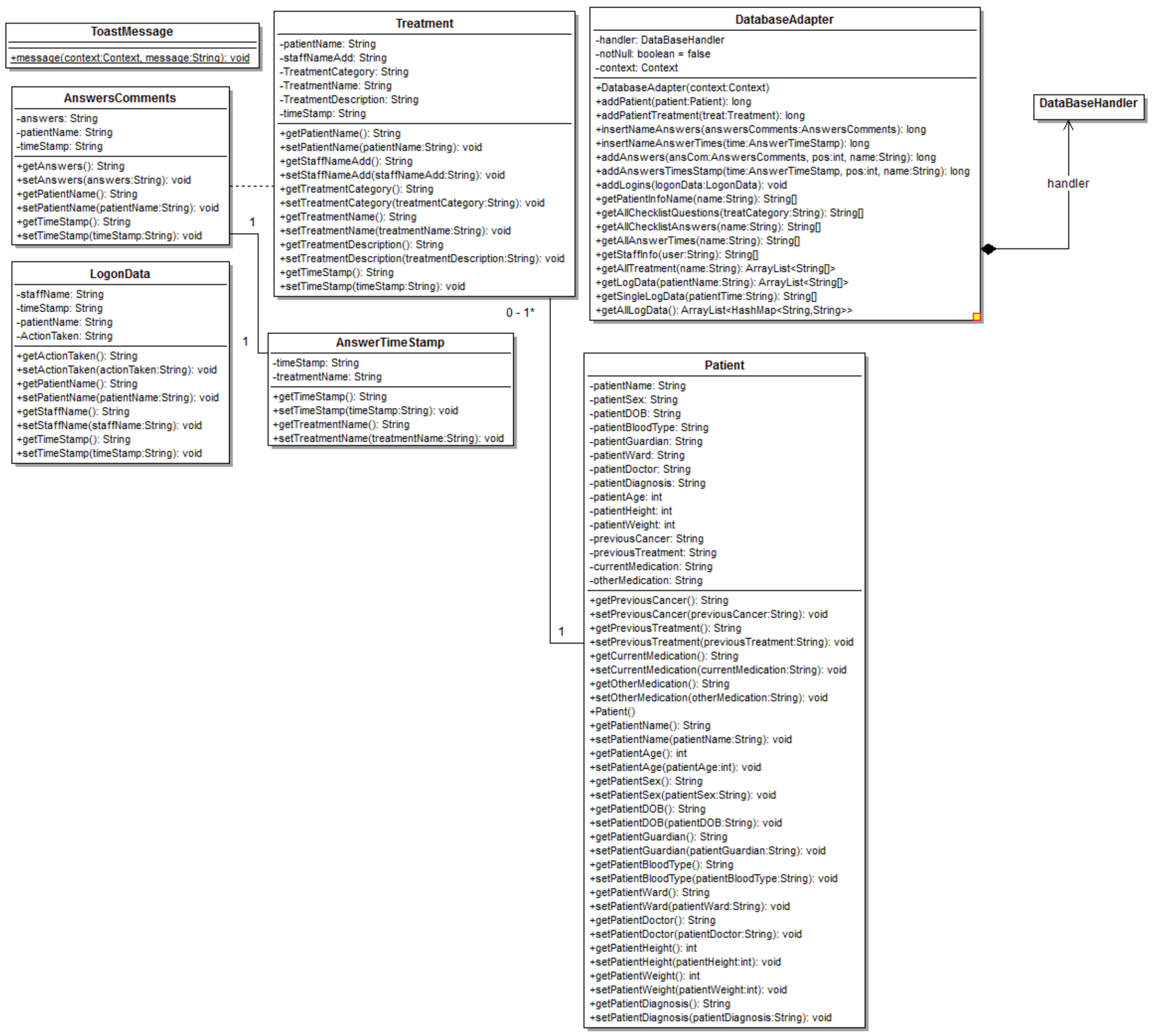

ToastMessage
+message(context:Context, message:String): void
AnswersComments
-answers: String
-patientName: String
-timeStamp: String
+getAnswers(): String
+setAnswers(answers:String): void
+getPatientName(): String
+setPatientName(patientName:String): void
+getTimeStamp(): String
+setTimeStamp(timeStamp:String): void
LogonData
-staffName: String
-timeStamp: String
-patientName: String
-ActionTaken: String
+getActionTaken(): String
+setActionTaken(actionTaken:String): void
+getPatientName(): String
+setPatientName(patientName:String): void
+getStaffName(): String
+setStaffName(staffName:String): void
+getTimeStamp(): String
+setTimeStamp(timeStamp:String): void
Treatment
-patientName: String
-staffNameAdd: String
-TreatmentCategory: String
-TreatmentName: String
-TreatmentDescription: String
-timeStamp: String
+getPatientName(): String
+setPatientName(patientName:String): void
+getStaffNameAdd(): String
+setStaffNameAdd(staffNameAdd:String): void
+getTreatmentCategory(): String
+setTreatmentCategory(treatmentCategory:String): void
+getTreatmentName(): String
+setTreatmentName(treatmentName:String): void
+getTreatmentDescription(): String
+setTreatmentDescription(treatmentDescription:String): void
+getTimeStamp(): String
+setTimeStamp(timeStamp:String): void
1
1
0 - 1*
AnswerTimeStamp
-timeStamp: String
-treatmentName: String
+getTimeStamp(): String
+setTimeStamp(timeStamp:String): void
+getTreatmentName(): String
+setTreatmentName(treatmentName:String): void
DatabaseAdapter
-handler: DataBaseHandler
-notNull: boolean = false
-context: Context
+DatabaseAdapter(context:Context)
+addPatient(patient:Patient): long
+addPatientTreatment(treat:Treatment): long
+insertNameAnswers(answersComments:AnswersComments): long
+insertNameAnswerTimes(time:AnswerTimeStamp): long
+addAnswers(ansCom:AnswersComments, pos:int, name:String): long
+addAnswersTimesStamp(time:AnswerTimeStamp, pos:int, name:String): long
+addLogins(logonData:LogonData): void
+getPatientInfoName(name:String): String[]
+getAllChecklistQuestions(treatCategory:String): String[]
+getAllChecklistAnswers(name:String): String[]
+getAllAnswerTimes(name:String): String[]
+getStaffInfo(user:String): String[]
+getAllTreatment(name:String): ArrayList<String[]>
+getLogData(patientName:String): ArrayList<String[]>
+getSingleLogData(patientTime:String): String[]
+getAllLogData(): ArrayList<HashMap<String,String>>
DataBaseHandler
handler
1
Patient
-patientName: String
-patientSex: String
-patientDOB: String
-patientBloodType: String
-patientGuardian: String
-patientWard: String
-patientDoctor: String
-patientDiagnosis: String
-patientAge: int
-patientHeight: int
-patientWeight: int
-previousCancer: String
-previousTreatment: String
-currentMedication: String
-otherMedication: String
+getPreviousCancer(): String
+setPreviousCancer(previousCancer:String): void
+getPreviousTreatment(): String
+setPreviousTreatment(previousTreatment:String): void
+getCurrentMedication(): String
+setCurrentMedication(currentMedication:String): void
+getOtherMedication(): String
+setOtherMedication(otherMedication:String): void
+Patient()
+getPatientName(): String
+setPatientName(patientName:String): void
+getPatientAge(): int
+setPatientAge(patientAge:int): void
+getPatientSex(): String
+setPatientSex(patientSex:String): void
+getPatientDOB(): String
+setPatientDOB(patientDOB:String): void
+getPatientGuardian(): String
+setPatientGuardian(patientGuardian:String): void
+getPatientBloodType(): String
+setPatientBloodType(patientBloodType:String): void
+getPatientWard(): String
+setPatientWard(patientWard:String): void
+getPatientDoctor(): String
+setPatientDoctor(patientDoctor:String): void
+getPatientHeight(): int
+setPatientHeight(patientHeight:int): void
+getPatientWeight(): int
+setPatientWeight(patientWeight:int): void
+getPatientDiagnosis(): String
+setPatientDiagnosis(patientDiagnosis:String): void

Figure 5-2: Class diagram for the model

The second class diagram shows the overall class diagram for the system application, depicting the logical interactions between the classes, especially in the model and controller.

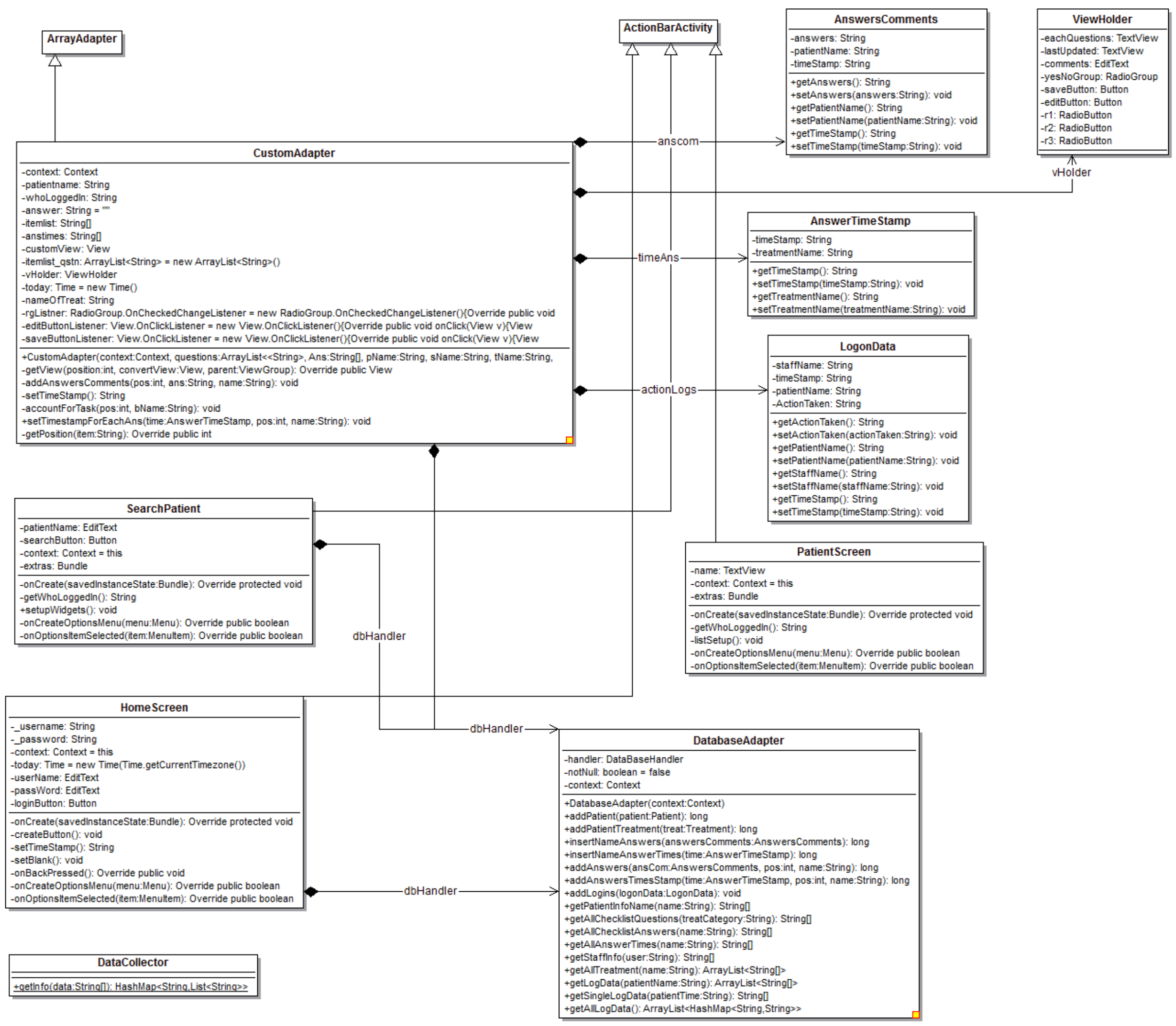

ArrayAdapter
ActionBarActivity
AnswersComments
-answers: String
-patientName: String
-timeStamp: String
+getAnswers(): String
+setAnswers(answers:String): void
+getPatientName(): String
+setPatientName(patientName:String): void
+getTimeStamp(): String
+setTimeStamp(timeStamp:String): void
ViewHolder
-eachQuestions: TextView
-lastUpdated: TextView
-comments: EditText
-yesNoGroup: RadioGroup
-saveButton: Button
-editButton: Button
-r1: RadioButton
-r2: RadioButton
-r3: RadioButton
anscom
vHolder
CustomAdapter
-context: Context
-patientname: String
-whoLoggedIn: String
-answer: String = ""
-itemlist: String[]
-anstimes: String[]
-customView: View
-itemlist_qstn: ArrayList<String> = new ArrayList<String>()
-vHolder: ViewHolder
-today: Time = new Time()
-nameOfTreat: String
-rgListner: RadioGroup.OnCheckedChangeListener = new RadioGroup.OnCheckedChangeListener(){Override public void
-editButtonListener: View.OnClickListener = new View.OnClickListener(){Override public void onClick(View v){View
-saveButtonListener: View.OnClickListener = new View.OnClickListener(){Override public void onClick(View v){View
+CustomAdapter(context:Context, questions:ArrayList<<String>, Ans:String[], pName:String, sName:String, tName:String,
-getView(position:int, convertView:View, parent:ViewGroup): Override public View
-addAnswersComments(pos:int, ans:String, name:String): void
-setTimeStamp(): String
-accountForTask(pos:int, bName:String): void
+setTimestampForEachAns(time:AnswerTimeStamp, pos:int, name:String): void
-getPosition(item:String): Override public int
timeAns
AnswerTimeStamp
-timeStamp: String
-treatmentName: String
+getTimeStamp(): String
+setTimeStamp(timeStamp:String): void
+getTreatmentName(): String
+setTreatmentName(treatmentName:String): void
actionLogs
LogonData
-staffName: String
-timeStamp: String
-patientName: String
-ActionTaken: String
+getActionTaken(): String
+setActionTaken(actionTaken:String): void
+getPatientName(): String
+setPatientName(patientName:String): void
+getStaffName(): String
+setStaffName(staffName:String): void
+getTimeStamp(): String
+setTimeStamp(timeStamp:String): void
SearchPatient
-patientName: EditText
-searchButton: Button
-context: Context = this
-extras: Bundle
-onCreate(savedInstanceState:Bundle): Override protected void
-getWhoLoggedIn(): String
+setupWidgets(): void
-onCreateOptionsMenu(menu:Menu): Override public boolean
-onOptionsItemSelected(item:MenuItem): Override public boolean
dbHandler
PatientScreen
-name: TextView
-context: Context = this
-extras: Bundle
-onCreate(savedInstanceState:Bundle): Override protected void
-getWhoLoggedIn(): String
-listSetup(): void
-onCreateOptionsMenu(menu:Menu): Override public boolean
-onOptionsItemSelected(item:MenuItem): Override public boolean
HomeScreen
-_username: String
-_password: String
-context: Context = this
-today: Time = new Time(Time.getCurrentTimezone())
-userName: EditText
-passWord: EditText
-loginButton: Button
-onCreate(savedInstanceState:Bundle): Override protected void
-createButton(): void
-setTimeStamp(): String
-setBlank(): void
-onBackPressed(): Override public void
-onCreateOptionsMenu(menu:Menu): Override public boolean
-onOptionsItemSelected(item:MenuItem): Override public boolean
dbHandler
dbHandler
DatabaseAdapter
-handler: DataBaseHandler
-notNull: boolean = false
-context: Context
+DatabaseAdapter(context:Context)
+addPatient(patient:Patient): long
+addPatientTreatment(treat:Treatment): long
+insertNameAnswers(answersComments:AnswersComments): long
+insertNameAnswerTimes(time:AnswerTimeStamp): long
+addAnswers(ansCom:AnswersComments, pos:int, name:String): long
+addAnswersTimesStamp(time:AnswerTimeStamp, pos:int, name:String): long
+addLogins(logonData:LogonData): void
+getPatientInfoName(name:String): String[]
+getAllChecklistQuestions(treatCategory:String): String[]
+getAllChecklistAnswers(name:String): String[]
+getAllAnswerTimes(name:String): String[]
+getStaffInfo(user:String): String[]
+getAllTreatment(name:String): ArrayList<String[]>
+getLogData(patientName:String): ArrayList<String[]>
+getSingleLogData(patientTime:String): String[]
+getAllLogData(): ArrayList<HashMap<String,String>>
DataCollector
+getInfo(data:String[]): HashMap<String,List<String>>

Figure 5-3: Complete Class diagram

5.3.2 Process View

This type of perspective of the application system simply depicts how events are sequentially executed between the system entities. There are two main entities represented in the figure below, they are; the user, and the mobile device. Using a *Sequence Diagram* to model this perspective, the system's communication sequence between these system entities can be best depicted. The mobile device houses two sub entities of the system, which are; the interface and the database. A hospital staff member, by use of an android smart device (tablet/phone), launches the application; on launch, such a staff member is required to log in, in order to access and manage patient data and treatment. Hospital staff members' credentials are stored in an internal database, created on initial application launch, which is then checked against login details entered. This design ensures that only authorised members of the hospital staff are able to access confidential data of patients within the hospital.

On successful log on, the staff member can create new patient data or manage old patient data, all updates on patient data are accountable by logged on staff member, as the application will take note of which staff member changed the data, as well as a time stamp on such change, this is designed to enable accountability between staff members. All patient data as well as hospital staff details are stored in an internal database created on initial application launch on any android smart device. Certain tasks can be completed, such as setting and confirming administration of a treatment or therapy that is set in a timed schedule, this schedule acts as a reminder that is setup to help staff members to remember to administer such treatments, even in a busy environment. Also the system application is designed to help hospital staff account for certain routine checks on surgical patients before surgery. These design decisions and more, better enable patient safety and try eliminate human error, as well as increase staff member accountability.

An *Activity Diagram* can also be seen below that demonstrated the flow of information within the states of the system application.

Figure 5-4: Activity Diagram of the system

5.3.3 Development View

This type of perspective decomposes the components of the system application and serves as a basis for requirement allocation for ease of development (Kruchten 1995). To model this perspective, a component diagram was used to demonstrate how the model, view and controller are connected. An requirement responsibility allocation table is also can also been seen below, demonstrating which of the architecture components (i.e. MVC) are responsible for accomplishing driving system requirements.

Insert component diagram here

S/N	Requirement	Model	Control	View
NFR001	The system must work seamlessly on android mobile devices	**X**	**X**	**X**
FR001	The system must only allow authorised hospital staff personnel to access patient data via logging in, on application launch	**X**		
FR002	The system must have a storage facility to record information gathered during system operation	**X**		
FR005	The system must allow hospital staff register a new patient on the system	**X**	**X**	**X**
FR006	The system must all hospital staff to view and manage already register patient's data on the system	**X**	**X**	**X**
NFR003	The system UI must have a layout and display that are easy to read and understand to carry out tasks.			**X**
FR008	The system must allow the hospital staff to select a specific patient out of a cache of registered patients to manage	**X**		
FR010	The system must allow hospital staff to add a specific treatment for a certain patient		**X**	**X**
FR011	The system must allow hospital staff to set a reminder for a specified treatment for a		**X**	**X**

	specific patient			
FR012	The system must alert hospital staff on set date and time to administer a logged patient treatment		**X**	
FR013	The system must allow hospital staff to answer routine questions for pre-surgery checks for s specific patient	**X**	**X**	**X**

5.3.4 Physical View

This perspective shows the packaging of the software application as well as the hardware needed to operate it.

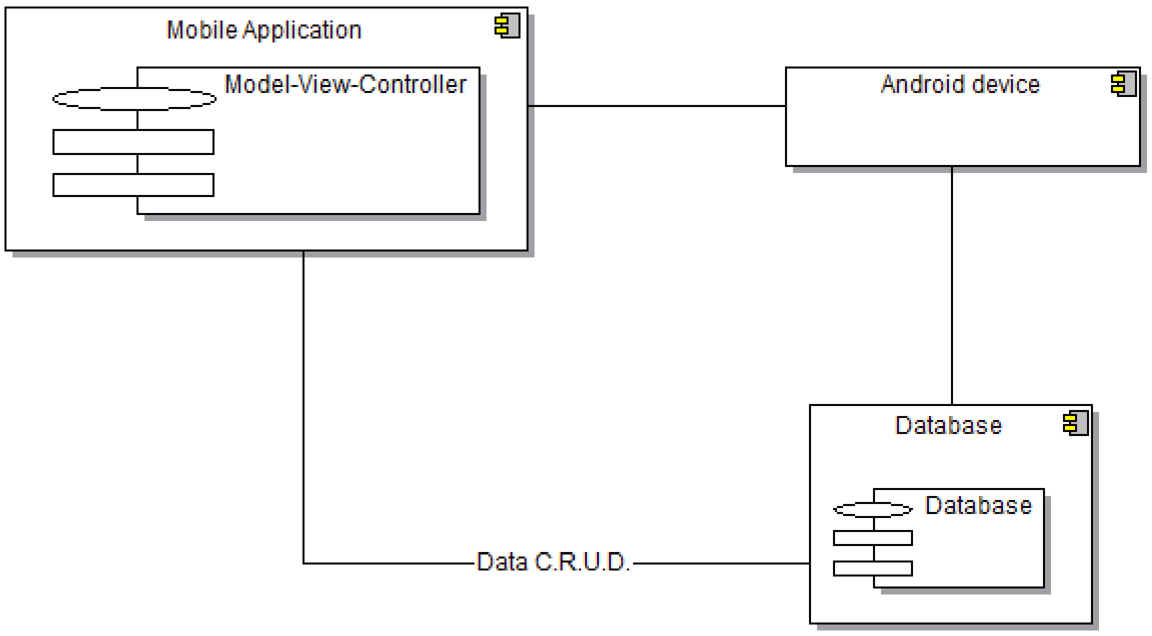


Figure 5-5: Physical View of the pre-surgical patient checklist application system

5.3.5 Use Case Scenario View

This perspective of the system application depicts scenarios by which end users (staff members), can accomplish certain tasks on the application system. A *Use Case* diagram is used to model how users perform functions. Authorised hospital personnel can manage data in terms of creating, modifying and viewing patient information. They can also set reminders for other staff members to administer patient treatment, scans or therapy. The system, on creation or modification of any patient data creates a time stamp as well as details of the hospital personnel who modified or created such patient data.

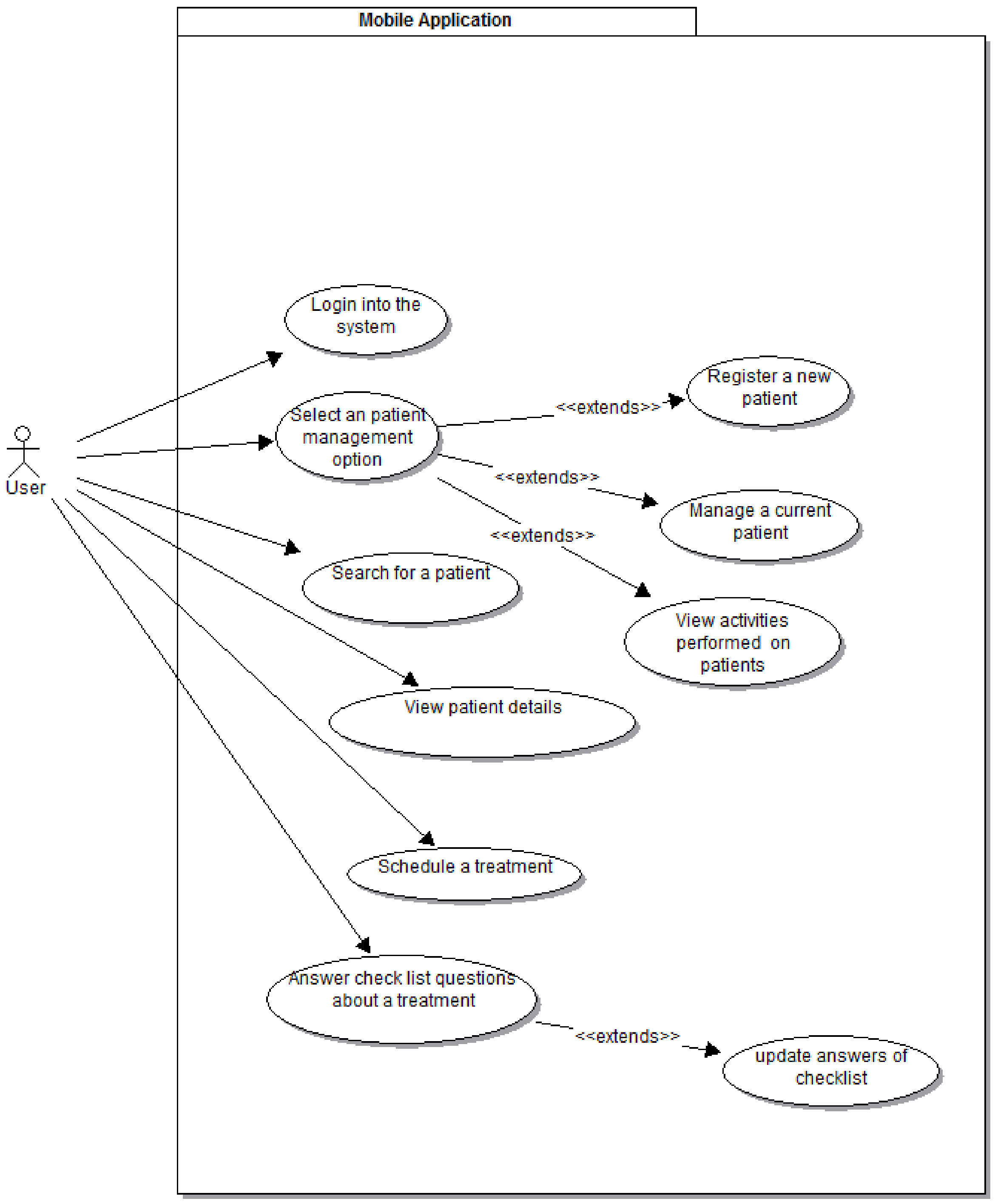


Figure 5-6: Use case scenario for the user of the mobile application

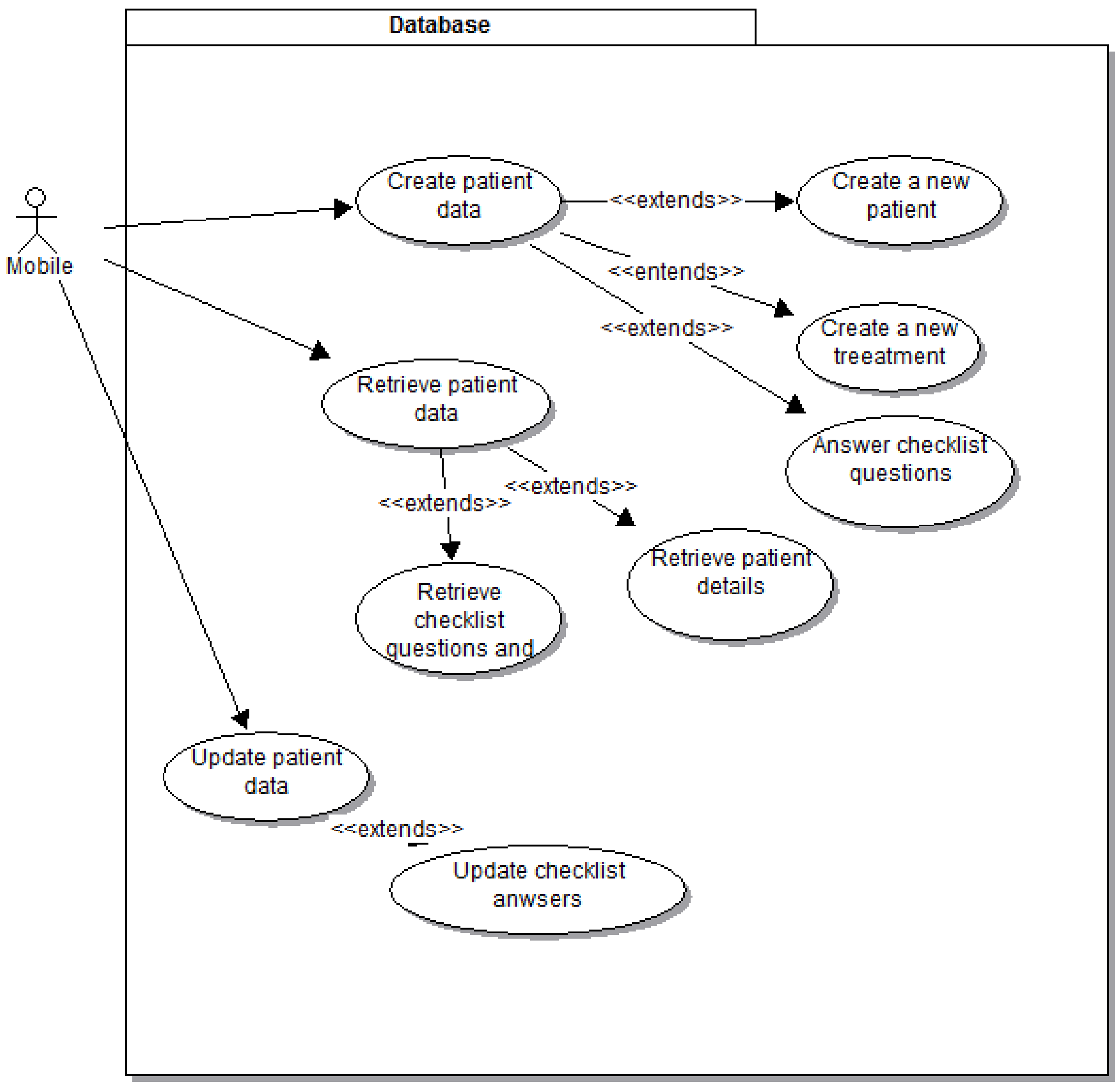


Figure 5-7: use case of mobile application accessing the database

5.4 Major System Design decision

To implement these architectural designs of this system application, design decisions on development tools used to create system components were made according to the table below.

System Components	Language	Alternative	Component Description
Database	Android integrated SQLite (Server-less SQL) – Written in Java	MySQL (External)	The database serves as a housing container for patient data managed by the application system. It was created using the integrated SQLite database management system on android devices which can operate without using a server. This kind of private database system was chosen because of its ease of use and access for development purposes.
Mobile Application:	Android Java and XML – Android studio IDE	PhP, Javascript HTML5/CSS, C# (Xamarin) , C++ (Xamarin), VB.NET	To develop the mobile system, decisions on the development language to use was made. Using the native development languages for application development on android devices was chosen. This is because, project system is simply to serve as a concept for the idea of a mobile patient checklist system, and as such, the scale of use for the system was quite small, and certain functionalities such as an alarm sub-system for generating

			alarms to remind hospital staff to administer certain treatments to certain patients are much easier to implement in the native development domain, as android provides an internal *AlarmMangager* class that handles reminders as well a *Notification* class that in combination can alert the staff of any pending patient treatment.

Table 5-1: Summary of key Design Decision

6 System Implementation

After the design stage of this software project, the implementation of such designs and design decisions as highlighted in chapter 5 are made. This section discusses such implementations for developing the said system and its software architecture, highlighting the step taken to achieve component development and integration.

6.1 System Component Implementation

6.1.1 The Database

In line with **FR002** of the functional system requirements in section 4.3, amongst the probable storage options the application system could use in recording information inputted into the application system, the implementation of a storage facility was done, using the android integrated SQLite database. This software library is a variation of the SQL database engine which is; self-contained, server less, and requires zero configurations (SQLite 2015). The database created using this internal library consists of several tables, which serve as containers for information recorded about entities in the application system. These tables are highlighted with brief description below;

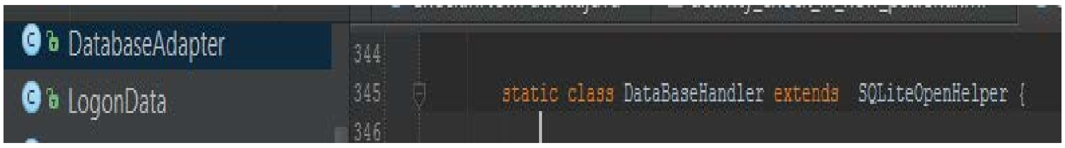
- **HOSPITAL_STAFF:** This table stores logging in details of authorised staff members; it contains the personnel name, their role in the hospital, username and password. It enables such personnel to authenticate themselves using predefined usernames and password.
- **PATIENT_INFO:** This table stores the basic information about patients that are registered in the application system. It allows the hospital staff members (authorised) to add new patients booked in for surgery; it contains columns for patient names, date of birth, age, blood type, and other general information needed, as well as the type of surgery the patient is to have.
- **HQ_QUESTIONS:** This table stores the routine pre-surgery check list questions per procedure type. These questions are to be answered by the authorised hospital staff before a procedure is given the go ahead. It simply contains columns for the procedure category and each question to be asked. Currently there are eleven questions stored in this table per procedure category.
- **ANSWERS:** This table stores the answers authorised hospital staff gives for each of the pre-surgery questions asked for each patient. It contains columns for patient names (the patient name in this context is a combination of the patient name and the procedure to

be undergone), and the answer to each question asked. The table is setup such that each set of answers can be assigned to each patient in combination with the procedure being undergone.

- **PATIENT_TREATMENT:** This table stores the set treatments for each patient assigned by an authorised member of staff. It contains columns for recording the patient name, the staff member who added the treatment, the name of the treatment, the type of treatment, and at what date and time the treatment is required.
- **LOGINS:** This table records each patient management action taken by a staff member on the application system. It contains columns for recording the name of the staff member currently logged on to the system, the name of the patient such staff member is using the application system to manage, the date and time stamp of when any patient management was done, and lastly, what patient management action was performed using the application system.
- **TIMESTAMP:** This database table simply records the date and time stamp of each answer in the **ANSWER** table per patient/treatment combination.

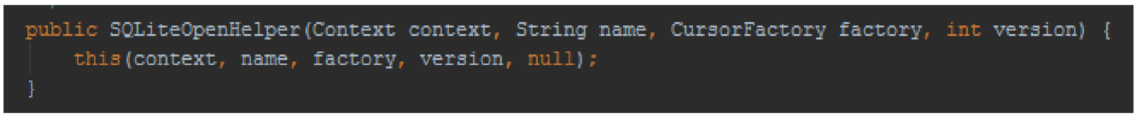
In most cases, especially when developing websites or desktop applications, creating a SQLite database, involved using a SQLite development environment separate from that of the native development environment for the application syste. Android mobile devices on the other hand, can create SQLite databases within the native environment, using native android *Class* libraries in combination with SQL queries in native android java language. The class libraries used for this application system are; the *SQLiteOpenHelper* and *SQLiteDatabase* android java classes libraries. The *SQLiteOpenHelper* class is a helper class for management of database creation and version management (Android 2015) while the *SQLiteDatabase* class enables methods for managing a SQLite database.

First, by inheriting properties from the *SQLiteOpenHelper* class, which takes *constructor* arguments; Inheriting class context, the name of the database created, factory cursor (instantiated as *null*), and the version number of the database, an SQLite database will be created inside any android device on application installation by the database handler class implemented (*DataBaseHandeler*). This handler class (*DataBaseHandeler*) is a static inner class of the *DatabaseAdapter* class, which houses declaration and initialisation of local functions for the database operation management.



```java
static class DataBaseHandler extends SQLiteOpenHelper {
```

Figure 6-1: Code Snippet showing DataBaseHandler Class extending from SQLiteOpenHelper class

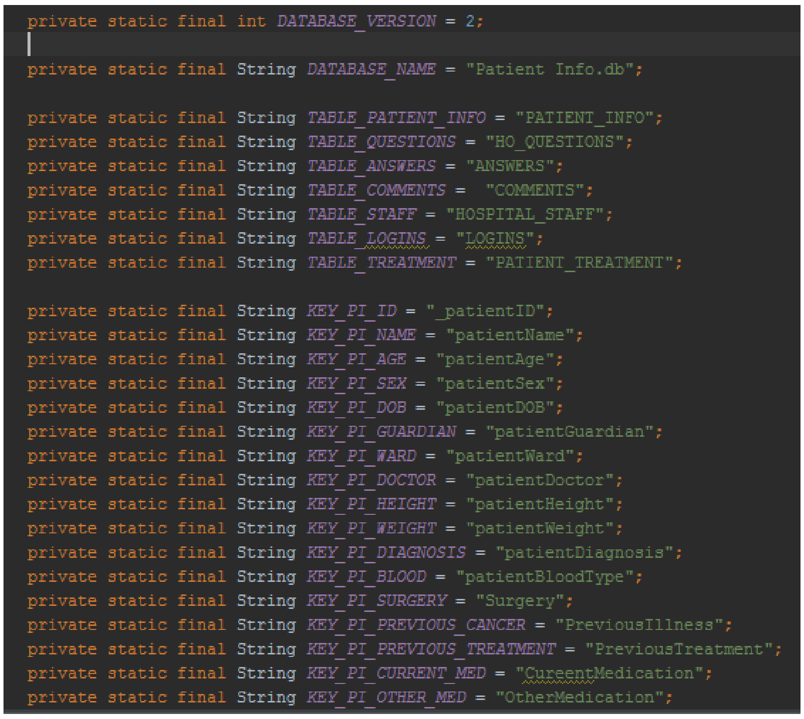


```java
public SQLiteOpenHelper(Context context, String name, CursorFactory factory, int version) {
    this(context, name, factory, version, null);
}
```

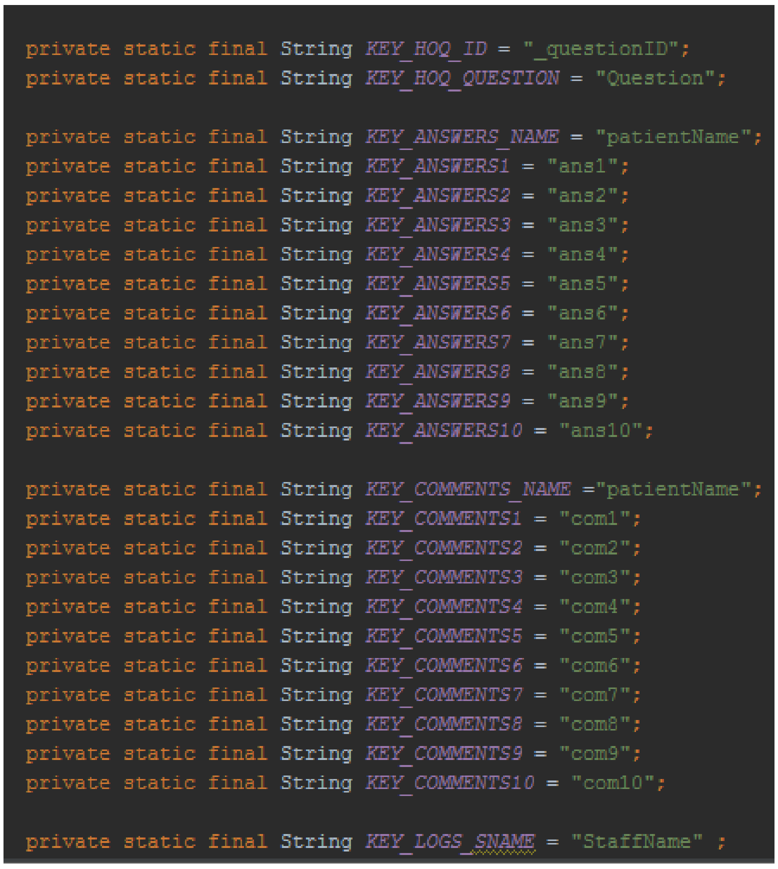
Figure 6-2: Code snippet showing super class constructor method.

By inheriting from the SQLiteOPenHelper class, method overrides to create and upgrade the database were implemented, but only after the declaration and initialisation of variables for the database. These variables include;

- The database version number, which changes per iteration of the application, which is of an integer data type.
- The name of the database file created on application installation.
- The table names to be created on application installation.
- The columns for each table in the database on application installation.



```java
private static final int DATABASE_VERSION = 2;

private static final String DATABASE_NAME = "Patient Info.db";

private static final String TABLE_PATIENT_INFO = "PATIENT_INFO";
private static final String TABLE_QUESTIONS = "HO_QUESTIONS";
private static final String TABLE_ANSWERS = "ANSWERS";
private static final String TABLE_COMMENTS =  "COMMENTS";
private static final String TABLE_STAFF = "HOSPITAL_STAFF";
private static final String TABLE_LOGINS = "LOGINS";
private static final String TABLE_TREATMENT = "PATIENT_TREATMENT";

private static final String KEY_PI_ID = "_patientID";
private static final String KEY_PI_NAME = "patientName";
private static final String KEY_PI_AGE = "patientAge";
private static final String KEY_PI_SEX = "patientSex";
private static final String KEY_PI_DOB = "patientDOB";
private static final String KEY_PI_GUARDIAN = "patientGuardian";
private static final String KEY_PI_WARD = "patientWard";
private static final String KEY_PI_DOCTOR = "patientDoctor";
private static final String KEY_PI_HEIGHT = "patientHeight";
private static final String KEY_PI_WEIGHT = "patientWeight";
private static final String KEY_PI_DIAGNOSIS = "patientDiagnosis";
private static final String KEY_PI_BLOOD = "patientBloodType";
private static final String KEY_PI_SURGERY = "Surgery";
private static final String KEY_PI_PREVIOUS_CANCER = "PreviousIllness";
private static final String KEY_PI_PREVIOUS_TREATMENT = "PreviousTreatment";
private static final String KEY_PI_CURRENT_MED = "CureentMedication";
private static final String KEY_PI_OTHER_MED = "OtherMedication";
```

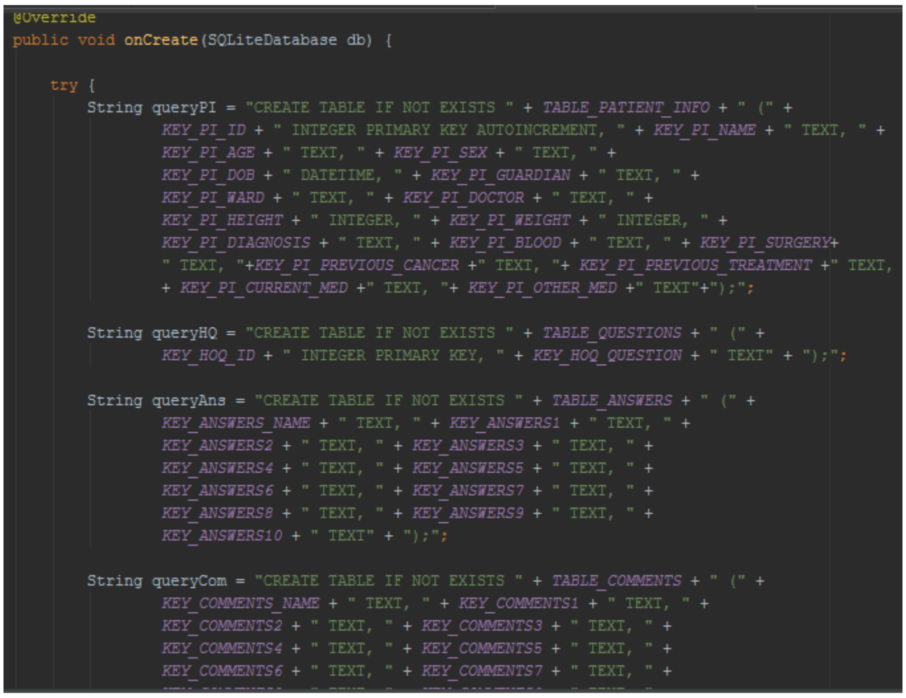


```
private static final String KEY_HOQ_ID = "_questionID";
private static final String KEY_HOQ_QUESTION = "Question";

private static final String KEY_ANSWERS_NAME = "patientName";
private static final String KEY_ANSWERS1 = "ans1";
private static final String KEY_ANSWERS2 = "ans2";
private static final String KEY_ANSWERS3 = "ans3";
private static final String KEY_ANSWERS4 = "ans4";
private static final String KEY_ANSWERS5 = "ans5";
private static final String KEY_ANSWERS6 = "ans6";
private static final String KEY_ANSWERS7 = "ans7";
private static final String KEY_ANSWERS8 = "ans8";
private static final String KEY_ANSWERS9 = "ans9";
private static final String KEY_ANSWERS10 = "ans10";

private static final String KEY_COMMENTS_NAME ="patientName";
private static final String KEY_COMMENTS1 = "com1";
private static final String KEY_COMMENTS2 = "com2";
private static final String KEY_COMMENTS3 = "com3";
private static final String KEY_COMMENTS4 = "com4";
private static final String KEY_COMMENTS5 = "com5";
private static final String KEY_COMMENTS6 = "com6";
private static final String KEY_COMMENTS7 = "com7";
private static final String KEY_COMMENTS8 = "com8";
private static final String KEY_COMMENTS9 = "com9";
private static final String KEY_COMMENTS10 = "com10";

private static final String KEY_LOGS_SNAME = "StaffName" ;
```

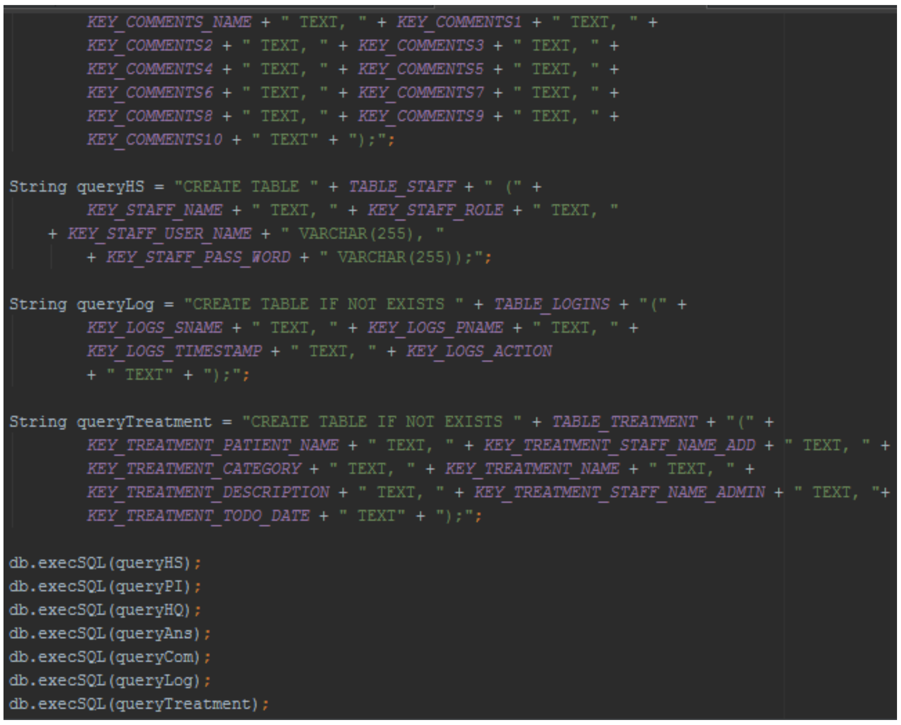
Figure 6-3: Code snippet of the table and column declaration for the database

After declaring and initialising all the variables needed for the database, as well as the constructor for the DatabaseHelper class, implementation of the override methods inherited from the SQLiteOpenHelper class were implemented, these methods are;

- **onCreate() method:** this method is called only once, when the application is first installed, this method implemented, is responsible for creating all the tables in the database, creating the columns and defining column properties for each table. The method takes a SQLiteDatabase object as its parameter. Tables, columns and settings for column properties are created using *String* queries (each table has its individual string query for defining its properties), executed by calling several instances of the execSQL() method from the SQLiteDatabase class, which takes these *String* queries as its only parameter. This method is a void return method.



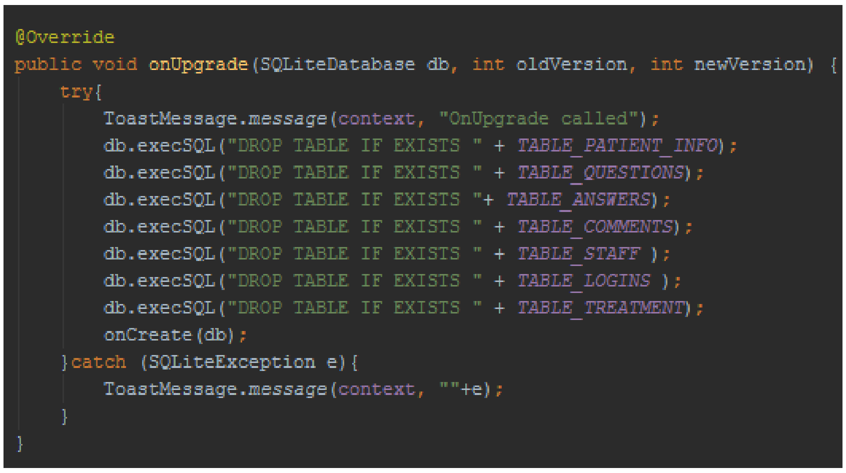
```
@Override
public void onCreate(SQLiteDatabase db) {

    try {
        String queryPI = "CREATE TABLE IF NOT EXISTS " + TABLE_PATIENT_INFO + " (" +
                KEY_PI_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " + KEY_PI_NAME + " TEXT, " +
                KEY_PI_AGE + " TEXT, " + KEY_PI_SEX + " TEXT, " +
                KEY_PI_DOB + " DATETIME, " + KEY_PI_GUARDIAN + " TEXT, " +
                KEY_PI_WARD + " TEXT, " + KEY_PI_DOCTOR + " TEXT, " +
                KEY_PI_HEIGHT + " INTEGER, " + KEY_PI_WEIGHT + " INTEGER, " +
                KEY_PI_DIAGNOSIS + " TEXT, " + KEY_PI_BLOOD + " TEXT, " + KEY_PI_SURGERY+
                " TEXT, "+KEY_PI_PREVIOUS_CANCER +" TEXT, "+ KEY_PI_PREVIOUS_TREATMENT +" TEXT,
                + KEY_PI_CURRENT_MED +" TEXT, "+ KEY_PI_OTHER_MED +" TEXT"+");";

        String queryHQ = "CREATE TABLE IF NOT EXISTS " + TABLE_QUESTIONS + " (" +
                KEY_HOQ_ID + " INTEGER PRIMARY KEY, " + KEY_HOQ_QUESTION + " TEXT" + ");";

        String queryAns = "CREATE TABLE IF NOT EXISTS " + TABLE_ANSWERS + " (" +
                KEY_ANSWERS_NAME + " TEXT, " + KEY_ANSWERS1 + " TEXT, " +
                KEY_ANSWERS2 + " TEXT, " + KEY_ANSWERS3 + " TEXT, " +
                KEY_ANSWERS4 + " TEXT, " + KEY_ANSWERS5 + " TEXT, " +
                KEY_ANSWERS6 + " TEXT, " + KEY_ANSWERS7 + " TEXT, " +
                KEY_ANSWERS8 + " TEXT, " + KEY_ANSWERS9 + " TEXT, " +
                KEY_ANSWERS10 + " TEXT" + ");";

        String queryCom = "CREATE TABLE IF NOT EXISTS " + TABLE_COMMENTS + " (" +
                KEY_COMMENTS_NAME + " TEXT, " + KEY_COMMENTS1 + " TEXT, " +
                KEY_COMMENTS2 + " TEXT, " + KEY_COMMENTS3 + " TEXT, " +
                KEY_COMMENTS4 + " TEXT, " + KEY_COMMENTS5 + " TEXT, " +
                KEY_COMMENTS6 + " TEXT, " + KEY_COMMENTS7 + " TEXT, " +
```



```
        KEY_COMMENTS_NAME + " TEXT, " + KEY_COMMENTS1 + " TEXT, " +
        KEY_COMMENTS2 + " TEXT, " + KEY_COMMENTS3 + " TEXT, " +
        KEY_COMMENTS4 + " TEXT, " + KEY_COMMENTS5 + " TEXT, " +
        KEY_COMMENTS6 + " TEXT, " + KEY_COMMENTS7 + " TEXT, " +
        KEY_COMMENTS8 + " TEXT, " + KEY_COMMENTS9 + " TEXT, " +
        KEY_COMMENTS10 + " TEXT" + ");";

String queryHS = "CREATE TABLE " + TABLE_STAFF + " (" +
        KEY_STAFF_NAME + " TEXT, " + KEY_STAFF_ROLE + " TEXT, "
    + KEY_STAFF_USER_NAME + " VARCHAR(255), "
        + KEY_STAFF_PASS_WORD + " VARCHAR(255));";

String queryLog = "CREATE TABLE IF NOT EXISTS " + TABLE_LOGINS + "(" +
        KEY_LOGS_SNAME + " TEXT, " + KEY_LOGS_PNAME + " TEXT, " +
        KEY_LOGS_TIMESTAMP + " TEXT, " + KEY_LOGS_ACTION
        + " TEXT" + ");";

String queryTreatment = "CREATE TABLE IF NOT EXISTS " + TABLE_TREATMENT + "(" +
        KEY_TREATMENT_PATIENT_NAME + " TEXT, " + KEY_TREATMENT_STAFF_NAME_ADD + " TEXT, " +
        KEY_TREATMENT_CATEGORY + " TEXT, " + KEY_TREATMENT_NAME + " TEXT, " +
        KEY_TREATMENT_DESCRIPTION + " TEXT, " + KEY_TREATMENT_STAFF_NAME_ADMIN + " TEXT, "+
        KEY_TREATMENT_TODO_DATE + " TEXT" + ");";

db.execSQL(queryHS);
db.execSQL(queryPI);
db.execSQL(queryHQ);
db.execSQL(queryAns);
db.execSQL(queryCom);
db.execSQL(queryLog);
db.execSQL(queryTreatment);
```

Figure 6-4: Code snippet of overridden *onCreate* method

- **onUpgrade() method:** This method is called only on revision of any of the tables and their properties in the database and on changing the version number of the database. It is implemented to delete the tables in the database, then, recreate them again with the revised properties (provided that such tables existed initially before this method was called). It also takes a SQLiteDatabase object as its only parameter and also returns as a void. The deletion of pre-existing tables is also performed using *String* queries executed by instances of the execSQL() function from the SQLiteDatabase class, which takes a *String* as its only parameter, then the *onCreate()* method is called to recreate the deleted tables with their revised properties and/or any new tables added. It is imperative that for this method to be called; the version number of the database, be changed to a higher integer than previously set.



```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    try{
        ToastMessage.message(context, "OnUpgrade called");
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_PATIENT_INFO);
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_QUESTIONS);
        db.execSQL("DROP TABLE IF EXISTS "+ TABLE_ANSWERS);
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_COMMENTS);
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_STAFF );
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_LOGINS );
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_TREATMENT);
        onCreate(db);
    }catch (SQLiteException e){
        ToastMessage.message(context, ""+e);
    }
}
```

Figure 6-5: Code snippet of the *onUpgrade()* method

6.1.2 Database Management

To fully grasp how the *Database* operates as a system component about data management must be discussed. The database management are the operations performed on the database, in order to manipulate the data stored within. These operations are called C.R.U.D. operations, this stands for; data *Creation, Retrieval, Update* and *Deletion*. The local functions that perform these operations are implemented in the *DatabaseAdapter* class. For more or less every table in the database there are local functions implemented to perform C.R.U.D. operations on the data set contained within each table of the database.

Data Creation: For dynamic data sets, which are inputs captured during the operation of the application system, *Object classes* were used to temporarily hold each instance of these data sets before being transferred to the appropriate table in the database. The dynamic datasets handled in this manner include;

- **Patient:** This *Object class* is responsible for retrieving instances of general patient information entered into the system. Its properties and attributes define the column properties of the *PATIENT_INFO* table. These properties are set into the object class by calling *setter* methods for each of the properties, and retrieved from the class by calling *getter* methods for each property.
- **AnswerComments:** This object class is responsible for storing instances of answers and comments received when pre-surgery questions are answered concerning a patient. Its primary properties and attributes define the column properties of the *ANSWERS* table. The properties are set into the object class by calling *setter* methods for each of the properties, and retrieved from the class by calling *getter* methods for each property.
- **Treatment:** This object class is responsible for storing instances of patient treatment prescribed to a patient. Its properties define the column properties of the *PATIENT_TREATMENT* table. These properties are set into the object class by calling *setter* methods for each of the properties, and retrieved from the class by calling *getter* methods for each property.
- **LogonData:** This object class is responsible for storing instances of retrieved system operation details once a task is completed in the application system. Its properties define the column properties of the *LOGINS* table. These properties are set into the object class by calling *setter* methods for each of the properties, and retrieved from the class by calling *getter* methods for each property.
- **AnswerTimeStamp:** This object class is responsible for recording a date and time stamp for each question answered per patient name and treatment combination.

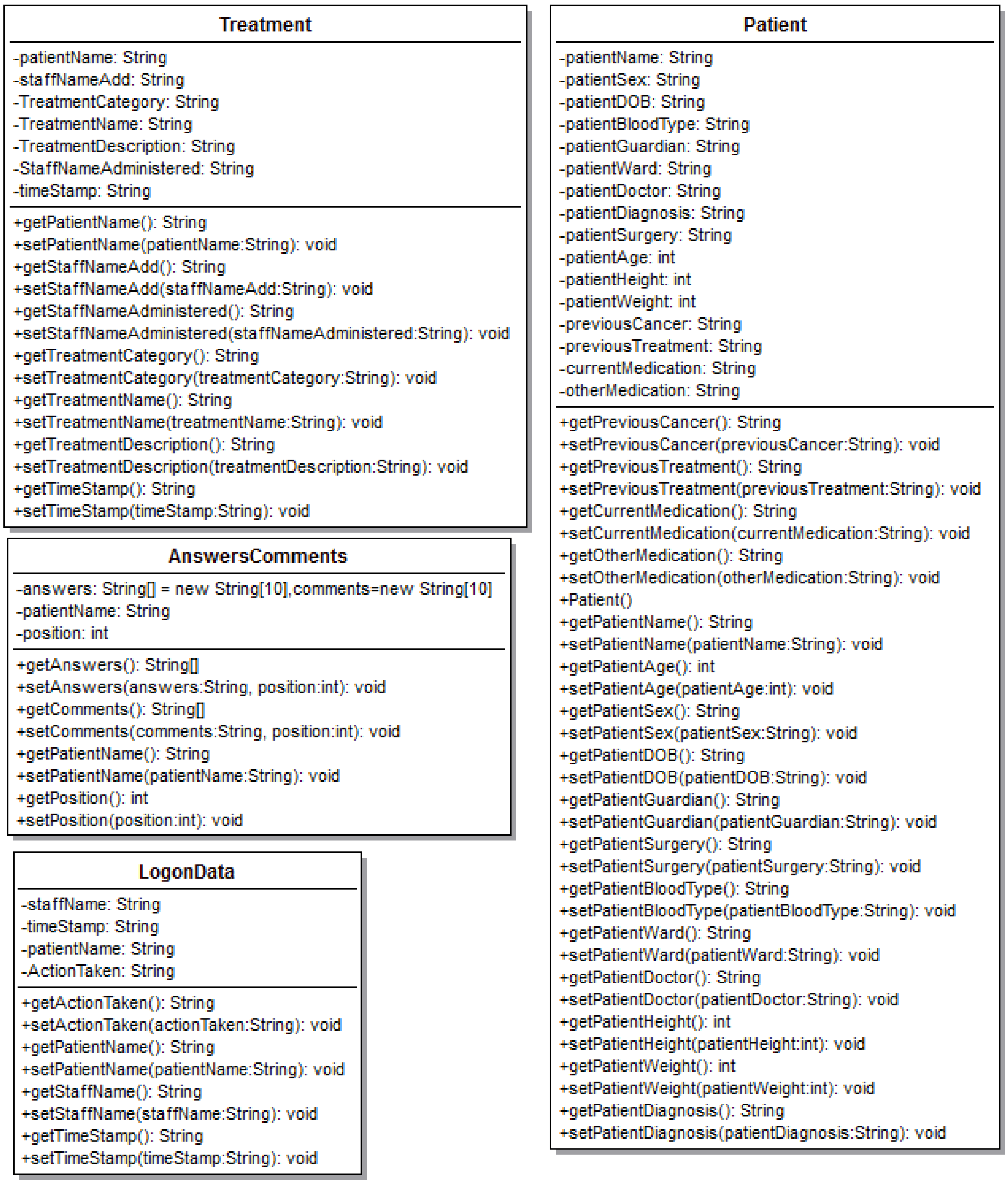
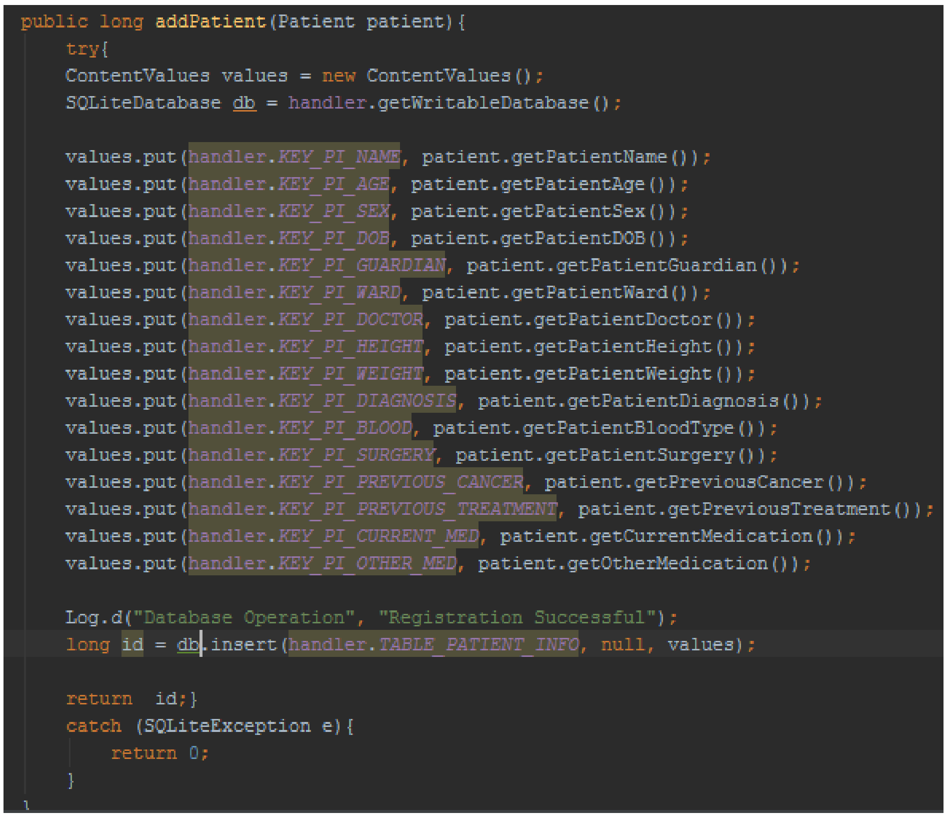

Treatment
-patientName: String
-staffNameAdd: String
-TreatmentCategory: String
-TreatmentName: String
-TreatmentDescription: String
-StaffNameAdministered: String
-timeStamp: String
+getPatientName(): String
+setPatientName(patientName:String): void
+getStaffNameAdd(): String
+setStaffNameAdd(staffNameAdd:String): void
+getStaffNameAdministered(): String
+setStaffNameAdministered(staffNameAdministered:String): void
+getTreatmentCategory(): String
+setTreatmentCategory(treatmentCategory:String): void
+getTreatmentName(): String
+setTreatmentName(treatmentName:String): void
+getTreatmentDescription(): String
+setTreatmentDescription(treatmentDescription:String): void
+getTimeStamp(): String
+setTimeStamp(timeStamp:String): void
AnswersComments
-answers: String[] = new String[10],comments=new String[10]
-patientName: String
-position: int
+getAnswers(): String[]
+setAnswers(answers:String, position:int): void
+getComments(): String[]
+setComments(comments:String, position:int): void
+getPatientName(): String
+setPatientName(patientName:String): void
+getPosition(): int
+setPosition(position:int): void
LogonData
-staffName: String
-timeStamp: String
-patientName: String
-ActionTaken: String
+getActionTaken(): String
+setActionTaken(actionTaken:String): void
+getPatientName(): String
+setPatientName(patientName:String): void
+getStaffName(): String
+setStaffName(staffName:String): void
+getTimeStamp(): String
+setTimeStamp(timeStamp:String): void
Patient
-patientName: String
-patientSex: String
-patientDOB: String
-patientBloodType: String
-patientGuardian: String
-patientWard: String
-patientDoctor: String
-patientDiagnosis: String
-patientSurgery: String
-patientAge: int
-patientHeight: int
-patientWeight: int
-previousCancer: String
-previousTreatment: String
-currentMedication: String
-otherMedication: String
+getPreviousCancer(): String
+setPreviousCancer(previousCancer:String): void
+getPreviousTreatment(): String
+setPreviousTreatment(previousTreatment:String): void
+getCurrentMedication(): String
+setCurrentMedication(currentMedication:String): void
+getOtherMedication(): String
+setOtherMedication(otherMedication:String): void
+Patient()
+getPatientName(): String
+setPatientName(patientName:String): void
+getPatientAge(): int
+setPatientAge(patientAge:int): void
+getPatientSex(): String
+setPatientSex(patientSex:String): void
+getPatientDOB(): String
+setPatientDOB(patientDOB:String): void
+getPatientGuardian(): String
+setPatientGuardian(patientGuardian:String): void
+getPatientSurgery(): String
+setPatientSurgery(patientSurgery:String): void
+getPatientBloodType(): String
+setPatientBloodType(patientBloodType:String): void
+getPatientWard(): String
+setPatientWard(patientWard:String): void
+getPatientDoctor(): String
+setPatientDoctor(patientDoctor:String): void
+getPatientHeight(): int
+setPatientHeight(patientHeight:int): void
+getPatientWeight(): int
+setPatientWeight(patientWeight:int): void
+getPatientDiagnosis(): String
+setPatientDiagnosis(patientDiagnosis:String): void

Figure 6-6: Class diagram of Object classes and their properties

From these object classes, data from each instance is retrieved and set into the appropriate table in the database. This operation is performed by putting each property of an object in a *String key* and *value* data set of a *ContentValue* object, then inserting the *ContentValue* object into its appropriate table using the *insert()* method of a *writeable* SQLiteDatabase object. The *insert()* method takes three parameters when called, the more significant parameters being; the table name where data is to be inserted to, and the *ContentValue* object. This method returns a *long* value, equivalent to the *row id* in which the data set was inserted in, of the specified table in the database



```
public long addPatient(Patient patient){
    try{
    ContentValues values = new ContentValues();
    SQLiteDatabase db = handler.getWritableDatabase();

    values.put(handler.KEY_PI_NAME, patient.getPatientName());
    values.put(handler.KEY_PI_AGE, patient.getPatientAge());
    values.put(handler.KEY_PI_SEX, patient.getPatientSex());
    values.put(handler.KEY_PI_DOB, patient.getPatientDOB());
    values.put(handler.KEY_PI_GUARDIAN, patient.getPatientGuardian());
    values.put(handler.KEY_PI_WARD, patient.getPatientWard());
    values.put(handler.KEY_PI_DOCTOR, patient.getPatientDoctor());
    values.put(handler.KEY_PI_HEIGHT, patient.getPatientHeight());
    values.put(handler.KEY_PI_WEIGHT, patient.getPatientWeight());
    values.put(handler.KEY_PI_DIAGNOSIS, patient.getPatientDiagnosis());
    values.put(handler.KEY_PI_BLOOD, patient.getPatientBloodType());
    values.put(handler.KEY_PI_SURGERY, patient.getPatientSurgery());
    values.put(handler.KEY_PI_PREVIOUS_CANCER, patient.getPreviousCancer());
    values.put(handler.KEY_PI_PREVIOUS_TREATMENT, patient.getPreviousTreatment());
    values.put(handler.KEY_PI_CURRENT_MED, patient.getCurrentMedication());
    values.put(handler.KEY_PI_OTHER_MED, patient.getOtherMedication());

    Log.d("Database Operation", "Registration Successful");
    long id = db.insert(handler.TABLE_PATIENT_INFO, null, values);

    return  id;}
    catch (SQLiteException e){
        return 0;
    }
}
```

Figure 6-7: Code snippet for adding properties of an instance of the Patient object class to the PATIENT_INFO table, this follows for all other objects mention above.

Data Retrieval: To retrieve data from a table in the database, simply involves performing queries on the database as one would in any SQL engine operations. By writing the appropriate query to search through a specific table in the database, either for all the data in such a table or a specific data set, the SQLiteDatabase method *query()* was used, which takes an android format SELECT query as its parameter. The parameter format for the query is; Table name, String array of the column names that data needs to be retrieved from, the common place data between all rows in the table (if all the data in such table is to be retrieved, this parameter is set to *null*), and four other parameters which were set to *null*. This *query()* method returns a cursor object that can be looped through using any loop type (*while* loops or *do-while* loops ideally) and the retrieved data is then inserted into either an Array or an ArrayList object.

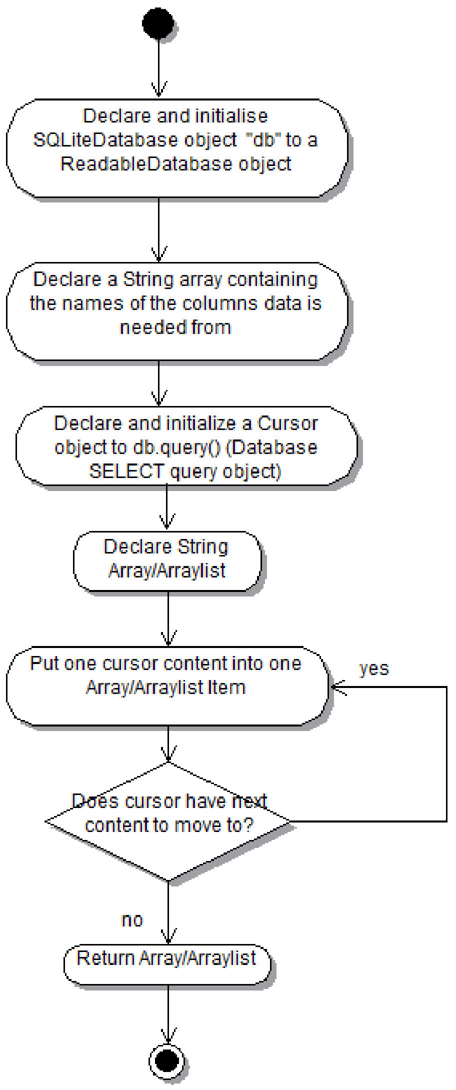

Declare and initialise SQLiteDatabase object "db" to a ReadableDatabase object
Declare a String array containing the names of the columns data is needed from
Declare and initialize a Cursor object to db.query() (Database SELECT query object)
Declare String Array/Arraylist
Put one cursor content into one Array/Arraylist Item
yes
Does cursor have next content to move to?
no
Return Array/Arraylist

Figure 6-8: Activity diagram of data retrieval

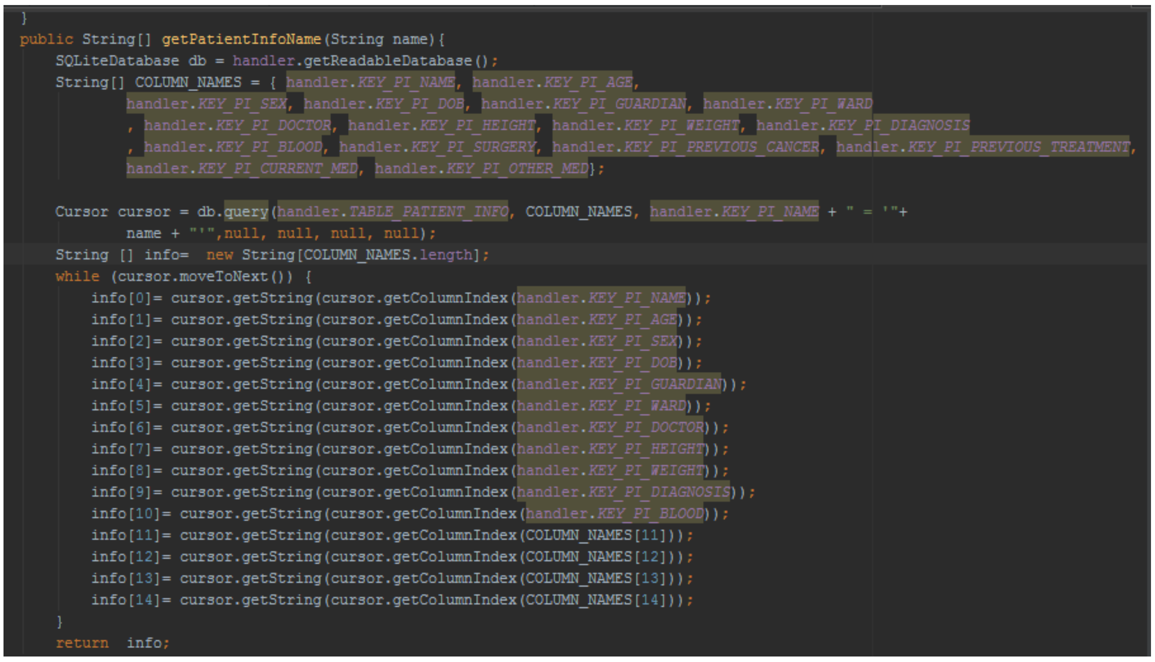
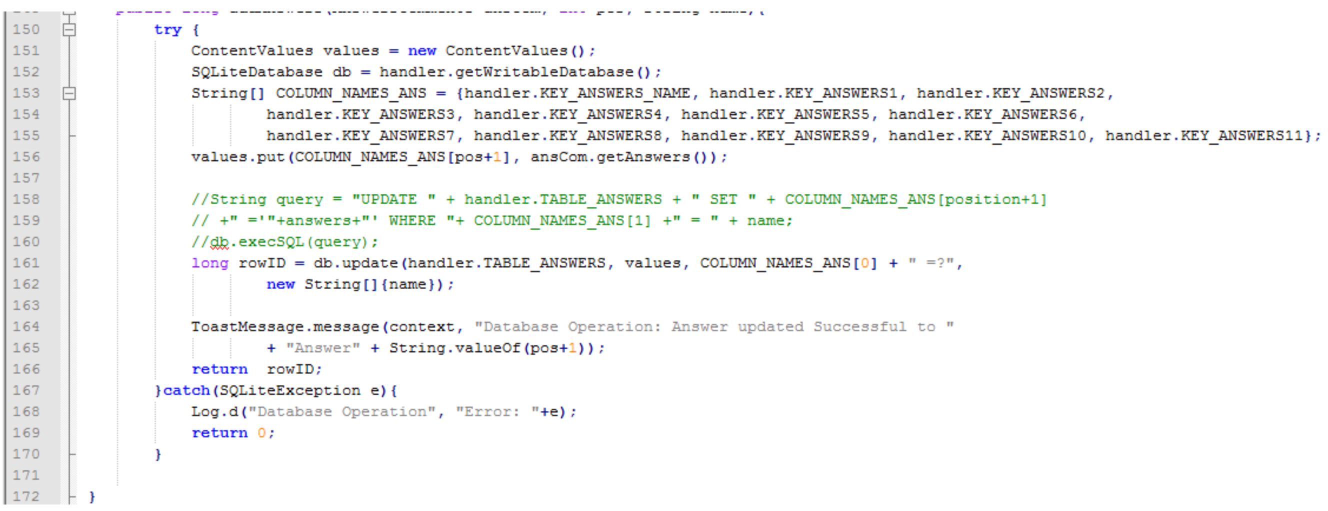

```
}
public String[] getPatientInfoName(String name){
    SQLiteDatabase db = handler.getReadableDatabase();
    String[] COLUMN_NAMES = { handler.KEY_PI_NAME, handler.KEY_PI_AGE,
            handler.KEY_PI_SEX, handler.KEY_PI_DOB, handler.KEY_PI_GUARDIAN, handler.KEY_PI_WARD
            , handler.KEY_PI_DOCTOR, handler.KEY_PI_HEIGHT, handler.KEY_PI_WEIGHT, handler.KEY_PI_DIAGNOSIS
            , handler.KEY_PI_BLOOD, handler.KEY_PI_SURGERY, handler.KEY_PI_PREVIOUS_CANCER, handler.KEY_PI_PREVIOUS_TREATMENT,
            handler.KEY_PI_CURRENT_MED, handler.KEY_PI_OTHER_MED};

    Cursor cursor = db.query(handler.TABLE_PATIENT_INFO, COLUMN_NAMES, handler.KEY_PI_NAME + " = '"+
            name + "'",null, null, null, null);
    String [] info=  new String[COLUMN_NAMES.length];
    while (cursor.moveToNext()) {
        info[0]= cursor.getString(cursor.getColumnIndex(handler.KEY_PI_NAME));
        info[1]= cursor.getString(cursor.getColumnIndex(handler.KEY_PI_AGE));
        info[2]= cursor.getString(cursor.getColumnIndex(handler.KEY_PI_SEX));
        info[3]= cursor.getString(cursor.getColumnIndex(handler.KEY_PI_DOB));
        info[4]= cursor.getString(cursor.getColumnIndex(handler.KEY_PI_GUARDIAN));
        info[5]= cursor.getString(cursor.getColumnIndex(handler.KEY_PI_WARD));
        info[6]= cursor.getString(cursor.getColumnIndex(handler.KEY_PI_DOCTOR));
        info[7]= cursor.getString(cursor.getColumnIndex(handler.KEY_PI_HEIGHT));
        info[8]= cursor.getString(cursor.getColumnIndex(handler.KEY_PI_WEIGHT));
        info[9]= cursor.getString(cursor.getColumnIndex(handler.KEY_PI_DIAGNOSIS));
        info[10]= cursor.getString(cursor.getColumnIndex(handler.KEY_PI_BLOOD));
        info[11]= cursor.getString(cursor.getColumnIndex(COLUMN_NAMES[11]));
        info[12]= cursor.getString(cursor.getColumnIndex(COLUMN_NAMES[12]));
        info[13]= cursor.getString(cursor.getColumnIndex(COLUMN_NAMES[13]));
        info[14]= cursor.getString(cursor.getColumnIndex(COLUMN_NAMES[14]));
    }
    return  info;
```

Figure 6-9: Code Snippet of data retrieval from the PATIENT_INFO table in the database as described above.

Data Update: This operation is implemented in the same manner as the *Creation* operation, using a *ContentValue* object to hold *String* data key sets for updating particular data in a table. Then by calling the *update()* method from a *writeable* SQLiteDatabase object, instead of *insert()*, the database is updated with the *ContentValue* object. This *update()* method takes in four parameters; the table name, the *ContentValue* object, the "Where Clause" (which column), and the "where clause argument" in form of a *String array*(what value inside that column). This method returns a *long* value.



```
        try {
            ContentValues values = new ContentValues();
            SQLiteDatabase db = handler.getWritableDatabase();
            String[] COLUMN_NAMES_ANS = {handler.KEY_ANSWERS_NAME, handler.KEY_ANSWERS1, handler.KEY_ANSWERS2,
                    handler.KEY_ANSWERS3, handler.KEY_ANSWERS4, handler.KEY_ANSWERS5, handler.KEY_ANSWERS6,
                    handler.KEY_ANSWERS7, handler.KEY_ANSWERS8, handler.KEY_ANSWERS9, handler.KEY_ANSWERS10, handler.KEY_ANSWERS11};
            values.put(COLUMN_NAMES_ANS[pos+1], ansCom.getAnswers());

            //String query = "UPDATE " + handler.TABLE_ANSWERS + " SET " + COLUMN_NAMES_ANS[position+1]
            // +" ='"+answers+"' WHERE "+ COLUMN_NAMES_ANS[1] +" = " + name;
            //db.execSQL(query);
            long rowID = db.update(handler.TABLE_ANSWERS, values, COLUMN_NAMES_ANS[0] + " =?",
                    new String[]{name});

            ToastMessage.message(context, "Database Operation: Answer updated Successful to "
                    + "Answer" + String.valueOf(pos+1));
            return  rowID;
        }catch(SQLiteException e){
            Log.d("Database Operation", "Error: "+e);
            return 0;
        }

    }
```

Figure 6-10: Code snippet showing implemented update method for updating answers in the database

These describe the operations that make up the database management functionality of the application system.

6.1.3 The Mobile Application

In line with the aim, objectives, and requirements of this project, this application system was developed as a mobile application, enabling on-the-go management of surgical patients, while reducing human error and increasing accountability in the hospital, by making use of features within android smart devices (tablets). These features include an integrated database system, alarm and notifications systems that facilitate a schedule management system to remind staff members of actions to be performed on surgical patients. Using the database tables and their contents, layouts, input and display widgets, the mobile application is able to allow staff members of the staff better manage surgical patients, than the old paper based system currently in use.

6.2 System Integration Implementation

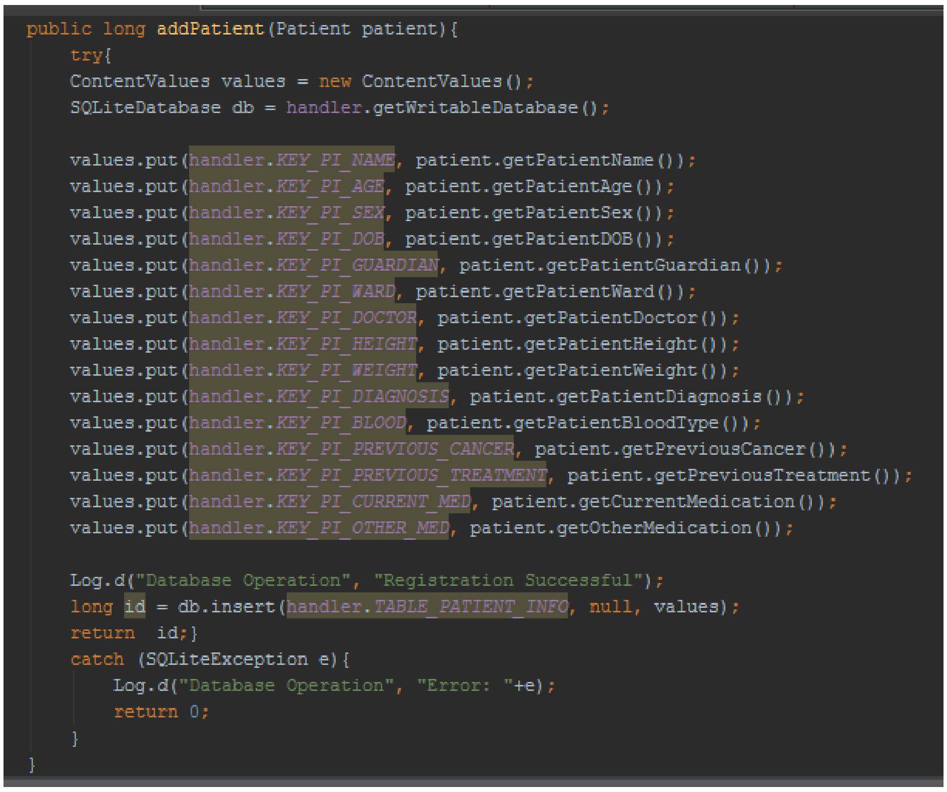
There are three key features implemented in line with the aims, objectives and requirements of the system on component integration implementation of the system, they include;

- Task Completion
- Accountability
- Human Error Reduction

6.2.1 Task Completion

The application system allows hospital staff to perform several tasks, including;

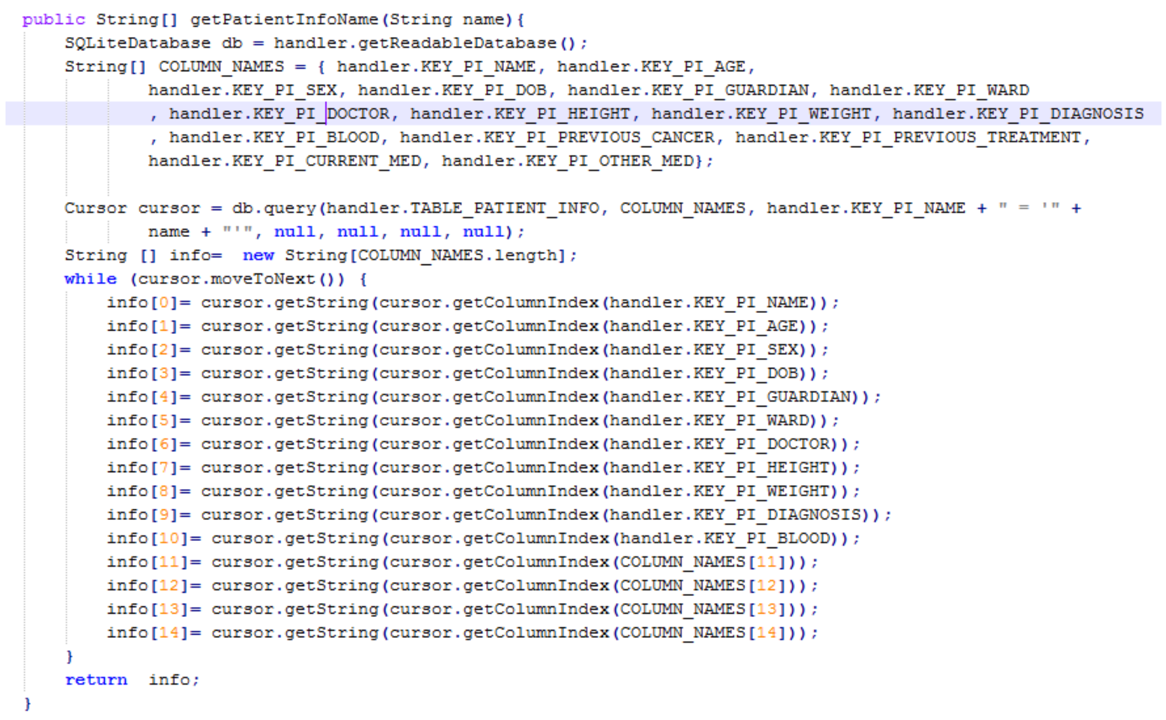
- Registering booked surgical patients (i.e. awaiting surgery) into the system, by entering general information about the patients like; patient name, age, date of birth, gender, blood type, diagnosis, height, weight, surgery booked for, the patient guardian, their assigned doctor and ward. Other information like the patient's medical history can also be inputted into the system via the *View* pattern architectural components, this includes; previous cancer diagnosis (if any), previous treatment and any previous medication the patient was on before being booked into surgery. These information, are collected from their various modes of entry, by a button click event handled by the *Controller* architecture, then they are all saved in a single Patient object instance and then inserted into the PATIENT_INFO table, inside the database via the "addPatient()" function in the *Model* architecture. The *controller* architecture component of the system application is responsible for handling the events for this task completion.



```
public long addPatient(Patient patient){
    try{
    ContentValues values = new ContentValues();
    SQLiteDatabase db = handler.getWritableDatabase();

    values.put(handler.KEY_PI_NAME, patient.getPatientName());
    values.put(handler.KEY_PI_AGE, patient.getPatientAge());
    values.put(handler.KEY_PI_SEX, patient.getPatientSex());
    values.put(handler.KEY_PI_DOB, patient.getPatientDOB());
    values.put(handler.KEY_PI_GUARDIAN, patient.getPatientGuardian());
    values.put(handler.KEY_PI_WARD, patient.getPatientWard());
    values.put(handler.KEY_PI_DOCTOR, patient.getPatientDoctor());
    values.put(handler.KEY_PI_HEIGHT, patient.getPatientHeight());
    values.put(handler.KEY_PI_WEIGHT, patient.getPatientWeight());
    values.put(handler.KEY_PI_DIAGNOSIS, patient.getPatientDiagnosis());
    values.put(handler.KEY_PI_BLOOD, patient.getPatientBloodType());
    values.put(handler.KEY_PI_PREVIOUS_CANCER, patient.getPreviousCancer());
    values.put(handler.KEY_PI_PREVIOUS_TREATMENT, patient.getPreviousTreatment());
    values.put(handler.KEY_PI_CURRENT_MED, patient.getCurrentMedication());
    values.put(handler.KEY_PI_OTHER_MED, patient.getOtherMedication());

    Log.d("Database Operation", "Registration Successful");
    long id = db.insert(handler.TABLE_PATIENT_INFO, null, values);
    return  id;}
    catch (SQLiteException e){
        Log.d("Database Operation", "Error: "+e);
        return 0;
    }
}
```

Figure 6-11: Code Snippet of the model function that adds patient details to the database

- After the user has registered a new patient, a user can search for such patient entity via a full name search facilitated by a search activity in the *View*. This contains an EditText field for entering desired patient name and a "Search" button, which when clicked, triggers a handle event function in the *Controller* architecture, which in turn calls a function "getPatientInfoName()" function in the *Model* architecture. This function queries the database for a patient details via the name entered.



```
public String[] getPatientInfoName(String name){
    SQLiteDatabase db = handler.getReadableDatabase();
    String[] COLUMN_NAMES = { handler.KEY_PI_NAME, handler.KEY_PI_AGE,
            handler.KEY_PI_SEX, handler.KEY_PI_DOB, handler.KEY_PI_GUARDIAN, handler.KEY_PI_WARD
            , handler.KEY_PI_DOCTOR, handler.KEY_PI_HEIGHT, handler.KEY_PI_WEIGHT, handler.KEY_PI_DIAGNOSIS
            , handler.KEY_PI_BLOOD, handler.KEY_PI_PREVIOUS_CANCER, handler.KEY_PI_PREVIOUS_TREATMENT,
            handler.KEY_PI_CURRENT_MED, handler.KEY_PI_OTHER_MED};

    Cursor cursor = db.query(handler.TABLE_PATIENT_INFO, COLUMN_NAMES, handler.KEY_PI_NAME + " = '" +
            name + "'", null, null, null, null);
    String [] info=  new String[COLUMN_NAMES.length];
    while (cursor.moveToNext()) {
        info[0]= cursor.getString(cursor.getColumnIndex(handler.KEY_PI_NAME));
        info[1]= cursor.getString(cursor.getColumnIndex(handler.KEY_PI_AGE));
        info[2]= cursor.getString(cursor.getColumnIndex(handler.KEY_PI_SEX));
        info[3]= cursor.getString(cursor.getColumnIndex(handler.KEY_PI_DOB));
        info[4]= cursor.getString(cursor.getColumnIndex(handler.KEY_PI_GUARDIAN));
        info[5]= cursor.getString(cursor.getColumnIndex(handler.KEY_PI_WARD));
        info[6]= cursor.getString(cursor.getColumnIndex(handler.KEY_PI_DOCTOR));
        info[7]= cursor.getString(cursor.getColumnIndex(handler.KEY_PI_HEIGHT));
        info[8]= cursor.getString(cursor.getColumnIndex(handler.KEY_PI_WEIGHT));
        info[9]= cursor.getString(cursor.getColumnIndex(handler.KEY_PI_DIAGNOSIS));
        info[10]= cursor.getString(cursor.getColumnIndex(handler.KEY_PI_BLOOD));
        info[11]= cursor.getString(cursor.getColumnIndex(COLUMN_NAMES[11]));
        info[12]= cursor.getString(cursor.getColumnIndex(COLUMN_NAMES[12]));
        info[13]= cursor.getString(cursor.getColumnIndex(COLUMN_NAMES[13]));
        info[14]= cursor.getString(cursor.getColumnIndex(COLUMN_NAMES[14]));
    }
    return  info;
}
```

Figure 6-12: Code Snippet of Patient details retrieval function

- Another task that can be completed on the system is, setting reminders to perform a treatment on a patient, if need be, before surgery. Implementation of this facility involved, taking input from user like;
    - ➢ The specific name of the treatment; using an input widget (Edit Text).
    - ➢ The type of treatment (i.e. if it is a scan, a drug or a form of therapy, or surgery to be performed), this is facilitated by three radio buttons, from which a single selection which best represents the treatment category (i.e. Scan, Line Insertion and Surgery).
    - ➢ A full description of what such treatment entails,
    - ➢ And then finally, settings of a date and time of when that treatment has to be administered, using a *DatePickerDialog* and a *TimePickerDialog* respectively, triggered by the "*Set Time of Reminder*" button onClick event handler.

This is then logged into the system by a button onClick event ("+" button). When this event is triggered, user inputs stated above are stored temporarily in an instance of the *Treatment* class, and then inserted into the *PATIENT_TREATMENT* table in the database, when the *addPatientTreatment()* local function is called from the DatabaseAdapter class. When this is done, a schedule list is automatically created at the bottom of the activity screen that contains a custom layout of;

- Text View for displaying for the name of the treatment.
- Text View for displaying for type of treatment.
- And last, a Text View for displaying the set date and time of the treatment.

A reminder service is created using the *NotificationManager* and *AlarmManager* android in-built classes, these are used to set a reminder that alerts staff members at a set time and date, to administer treatment to a patient. This alert can be triggered even if the application is not running or the android device is asleep.

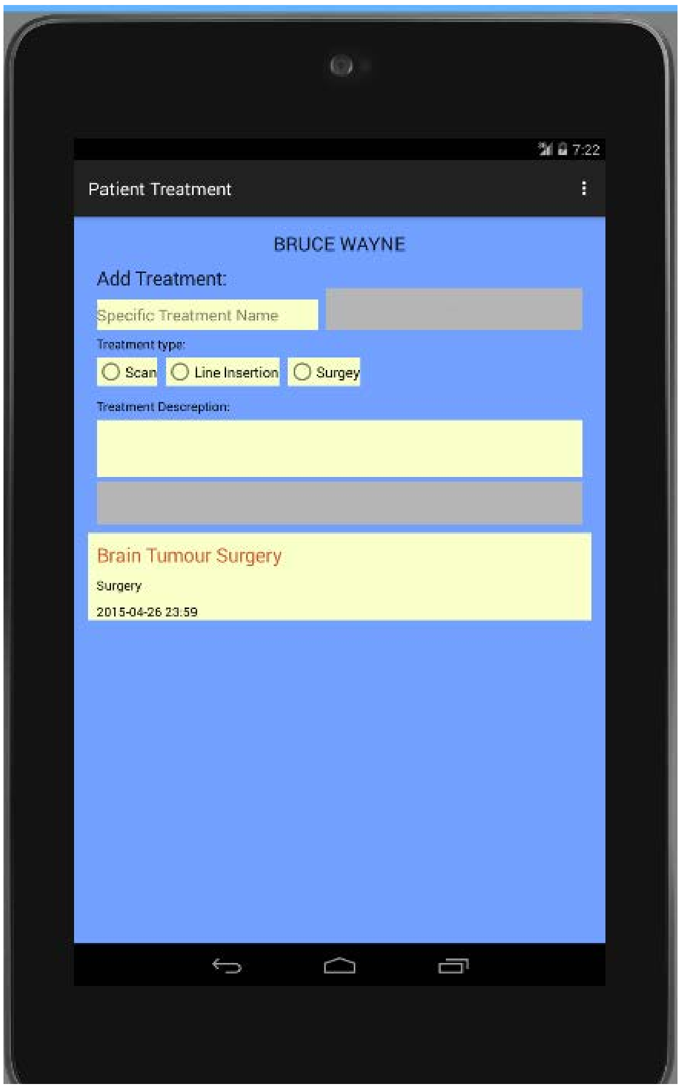


Figure 6-13: Treatment activity

- Users can then select any of the set treatments to view the full details of such a treatment, this then takes the user to the *pre-surgery checklist* activity. This activity facilitates the main task for this system, which is answering pre-surgery questions for such treatment/procedure as selected. This is line with the objectives in section 1.4 and the functional requirements stated in section 4. This is a major task to be completed on the application system, as these pre-surgery questions are routine for every patient on the system per specified treatment. They are a reflection on routine pre-surgery checks on surgical cancer patients and give accounts on when they were accomplished and who they were accomplished by. In reality, these checks could be completed on various occasions, by different staff members. To accommodate such a scenario, recording each answer for each question in a checklist is implemented to be independent of the next question. This allows for these questions to be answered at different times as soon as such activities have been accomplished in real life. These routine questions are easily answered in the application as a "*yes, no or not applicable*" format, with further comments that one might have regarding these questions. As such, each row entity contains generic layouts, namely

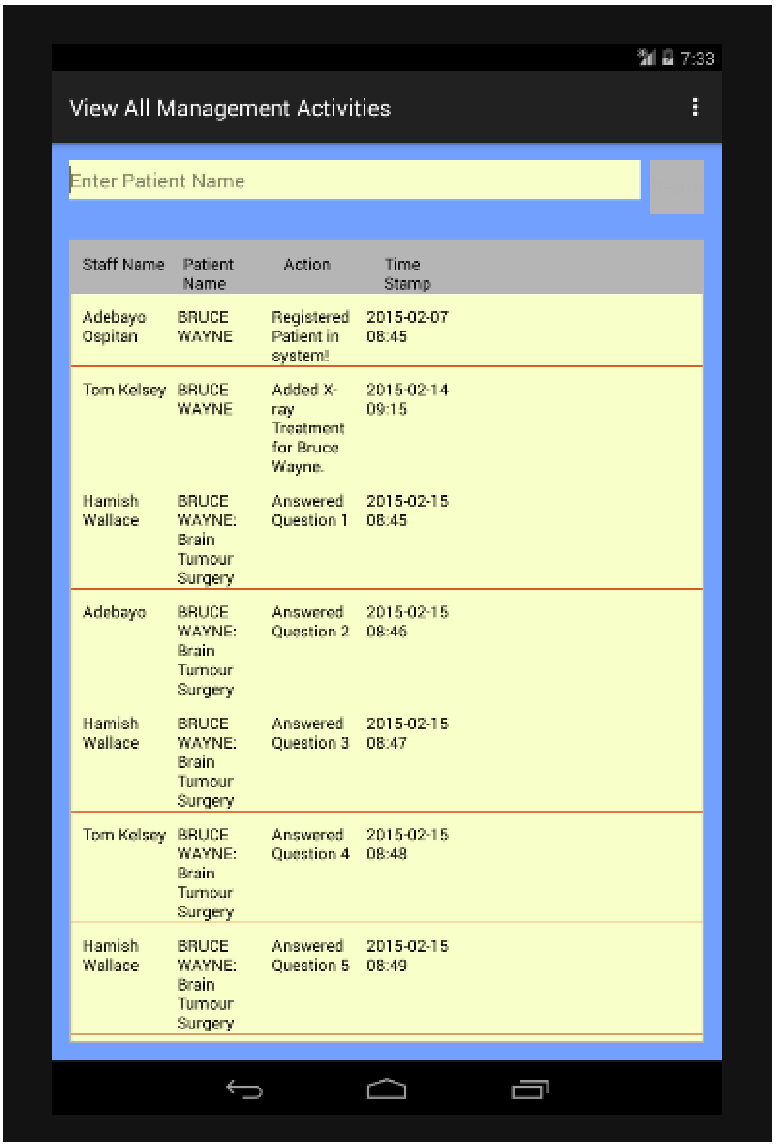
- A Text view to hold a question
- A group of three radio buttons that represent the three options of the answer format highlighted above. Only a single option can be selected at a time.
- An Edit Text to facilitate comment addition on each questions
- Buttons for editing these answers and saving the answer and comments for each question.

First, to edit answers concerning a particular question, the user clicks on the "*Edit*" button, which enables all the input widgets. Once satisfactory input has been entered, the user clicks the "*Save*" button, which triggers an "*onClickListener()*" method for handling the event. This event when handled will get the answer and comment from the input facilities per list row, then store each of them temporarily in an instance of the AnswerComment instance, which is then retrieved by the a local function *addAnswer()* from the DatabaseAdapter class when called. This inserts the collected answer and stores it in the database table ANSWERS accordingly.

This concept of the system application provides for multiple tasks to be completed with relative ease, and in a single device, while undergo, even in work heavy environment such as the client's.

6.2.2 Accountability

One of the key motivations for developing this application system is increasing accountability during day to day activities in the client's organisation, the medical domain such as the client's, this goes hand-in-hand with human error reduction, and together they help improve patient safety. How this application system facilitates accountability is simply by recording certain details about each task event completed by a user of the system on a particular patient, and storing each logged event in an instance of the LogonData class. From an instantiated object of the LogonData class, properties of these class can then be retrieved and inserted into the LOGINS table in the database.


7:33

View All Management Activities

Enter Patient Name

Staff Name	Patient Name	Action	Time Stamp
Adebayo Ospitan	BRUCE WAYNE	Registered Patient in system!	2015-02-07 08:45
Tom Kelsey	BRUCE WAYNE	Added X-ray Treatment for Bruce Wayne.	2015-02-14 09:15
Hamish Wallace	BRUCE WAYNE: Brain Tumour Surgery	Answered Question 1	2015-02-15 08:45
Adebayo	BRUCE WAYNE: Brain Tumour Surgery	Answered Question 2	2015-02-15 08:46
Hamish Wallace	BRUCE WAYNE: Brain Tumour Surgery	Answered Question 3	2015-02-15 08:47
Tom Kelsey	BRUCE WAYNE: Brain Tumour Surgery	Answered Question 4	2015-02-15 08:48
Hamish Wallace	BRUCE WAYNE: Brain Tumour Surgery	Answered Question 5	2015-02-15 08:49

Figure 6-14: Activity showing logged events of tasks completed

These Logged actions are displayed in a simple table grid in the application with a filter mechanism for filtering out tasks performed concerning a particular patient. This filter mechanism can be facilitated to query the database table for tasks performed on particular patient's data, then displayed in the table grid. Implementing this facilitates a simple but adequate way of reporting on all actions performed on patient data in the system application.

Accountability is also implemented in the checklist activity, such that on updating the answer to a checklist question, the system record which staff personnel updated each answer and at what date and time. This is information is then displayed when the list is refreshed or accessed by another user.

6.2.3 Human Error Reduction

In any domain, human error is an inevitable phenomenon, for the domain of the client specializes in, human error could either be as a result of simple complacency, or ignorance, or miscommunication amongst staff members. This could lead to patient safety being compromised, resulting in patient harm. Working in hand with the accountability feature, each unique treatment procedure will lead to the checklist for such a treatment based on its category displaying the most recent answers, a An example of patient safety being compromised is hospital staff members erring while administering treatments to patients or staff members completely forgetting to do so at all. By facilitating a scheduler per patient, incorporated with a timed notification service for treatments (with inputs described above, in section 6.2.1), as well as dialog to display full information about such treatment on selection in the schedule list, this application concept could help reduce human error among staff members, when managing surgical patients. Also by communicating (by displaying in the edit text field) the last updated answer for each checklist question as well as when last updated and by whom, ignorance is eliminated.

6.2.4 Usability

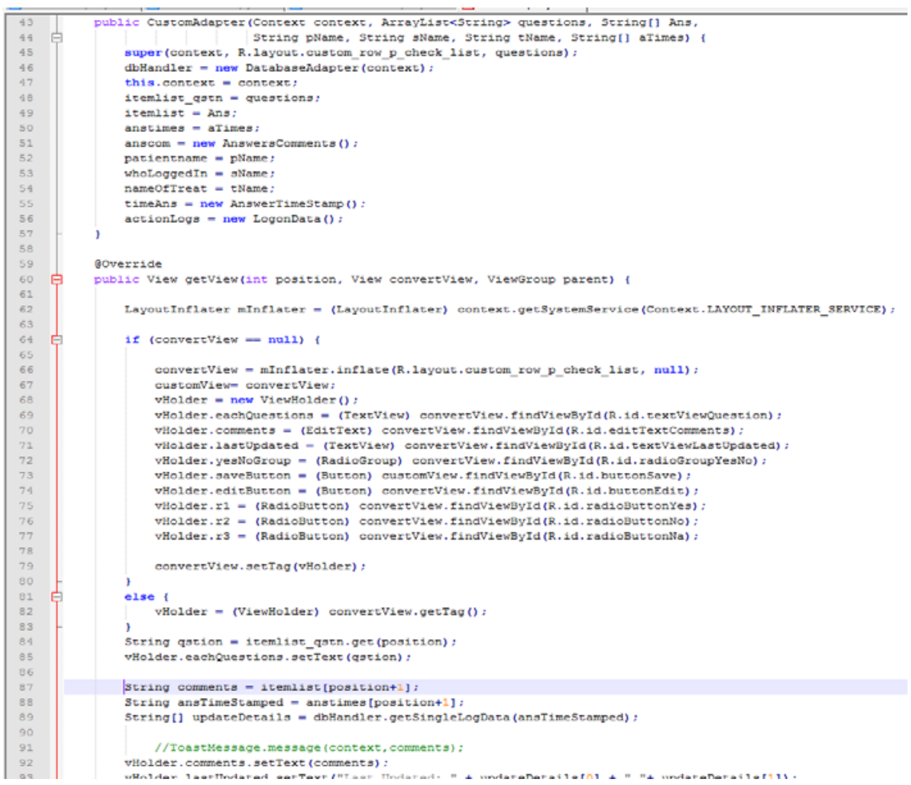
The ease with which users of this application system accomplish tasks is as important as accomplishing the tasks themselves. The layouts for each activity (*View*) for task completion were designed and implemented to be as simple as possible and require little technical knowledge to navigate through the application. Widgets are clearly labelled to indicate user input they require, and even buttons that handle data insertion are disabled till input is detected in some instances.

6.3 Interface Layout Implementation

The layouts for each activity were implemented using XML language generic to android development, and facilitate the View architecture for the application system. Layouts for each activity contain input and output widgets for taking in data into the system and outputting data for the user to see, constituting a well-defined User Interface U.I. for the system. Some layouts were simple to implement using predefined widgets provided by the IDE, other layouts such as layouts that holds the checklist itself or for patient treatment scheduling had widgets that were custom made to perform certain functions using pretexts of android studio predefined widgets. These were *List View* widgets, this type of widget by default displays a list of texts, but if the need arises (such as for the List of checklist questions and treatment scheduling information), the *List View* can be custom adapted to suit a propose. For example the checklist question *List View*; this view was adapted to contain;

- A Text View to display each checklist question
- A Radio Group that contain three Radio Buttons, represent the three choices a user can select as a direct answer to each question.
- An Edit Text to serves as an input for further comments a user might have pertaining to a checklist question.
- A Text View to display the last updated date and time for each question and answer.
- Two button, "*EDIT*" and "*SAVE/UPDATE*"; the *EDIT* button enable the edit text field above so that comment input can be received, while the *SAVE/UPDATE* button saves both the answer from the Radio button selected and the comments received from the edit text field.

To adapt a ListView to contain all these widgets, an adapter class was created, that defines and instantiates all the above mentioned widgets and their event handles when clicked.



```java
    public CustomAdapter(Context context, ArrayList<String> questions, String[] Ans,
                         String pName, String sName, String tName, String[] aTimes) {
        super(context, R.layout.custom_row_p_check_list, questions);
        dbHandler = new DatabaseAdapter(context);
        this.context = context;
        itemlist_qstn = questions;
        itemlist = Ans;
        anstimes = aTimes;
        anscom = new AnswersComments();
        patientname = pName;
        whoLoggedIn = sName;
        nameOfTreat = tName;
        timeAns = new AnswerTimeStamp();
        actionLogs = new LogonData();
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {

        LayoutInflater mInflater = (LayoutInflater) context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);

        if (convertView == null) {

            convertView = mInflater.inflate(R.layout.custom_row_p_check_list, null);
            customView= convertView;
            vHolder = new ViewHolder();
            vHolder.eachQuestions = (TextView) convertView.findViewById(R.id.textViewQuestion);
            vHolder.comments = (EditText) convertView.findViewById(R.id.editTextComments);
            vHolder.lastUpdated = (TextView) convertView.findViewById(R.id.textViewLastUpdated);
            vHolder.yesNoGroup = (RadioGroup) convertView.findViewById(R.id.radioGroupYesNo);
            vHolder.saveButton = (Button) customView.findViewById(R.id.buttonSave);
            vHolder.editButton = (Button) convertView.findViewById(R.id.buttonEdit);
            vHolder.r1 = (RadioButton) convertView.findViewById(R.id.radioButtonYes);
            vHolder.r2 = (RadioButton) convertView.findViewById(R.id.radioButtonNo);
            vHolder.r3 = (RadioButton) convertView.findViewById(R.id.radioButtonNa);

            convertView.setTag(vHolder);
        }
        else {
            vHolder = (ViewHolder) convertView.getTag();
        }
        String qstion = itemlist_qstn.get(position);
        vHolder.eachQuestions.setText(qstion);

        String comments = itemlist[position+1];
        String ansTimeStamped = anstimes[position+1];
        String[] updateDetails = dbHandler.getSingleLogData(ansTimeStamped);

            //ToastMessage.message(context,comments);
        vHolder.comments.setText(comments);
        vHolder.lastUpdated.setText("Last Updated: " + updateDetails[0] + " "+ updateDetails[1]);
```

Figure 6-15: Code snippet for the adapter class created for the custom ListView pre-surgery checklist

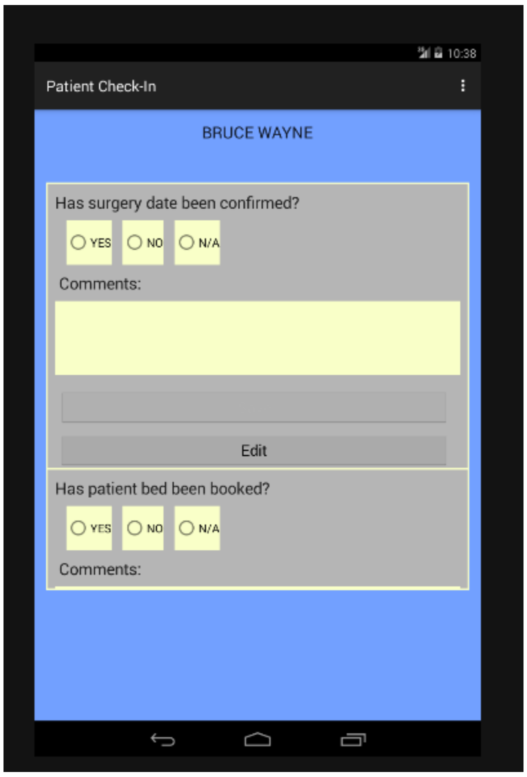


Figure 6-16: Pre-surgery checklist custom ListView

Other custom adapted ListViews were those of the Treatment scheduling list and the Management Log list.

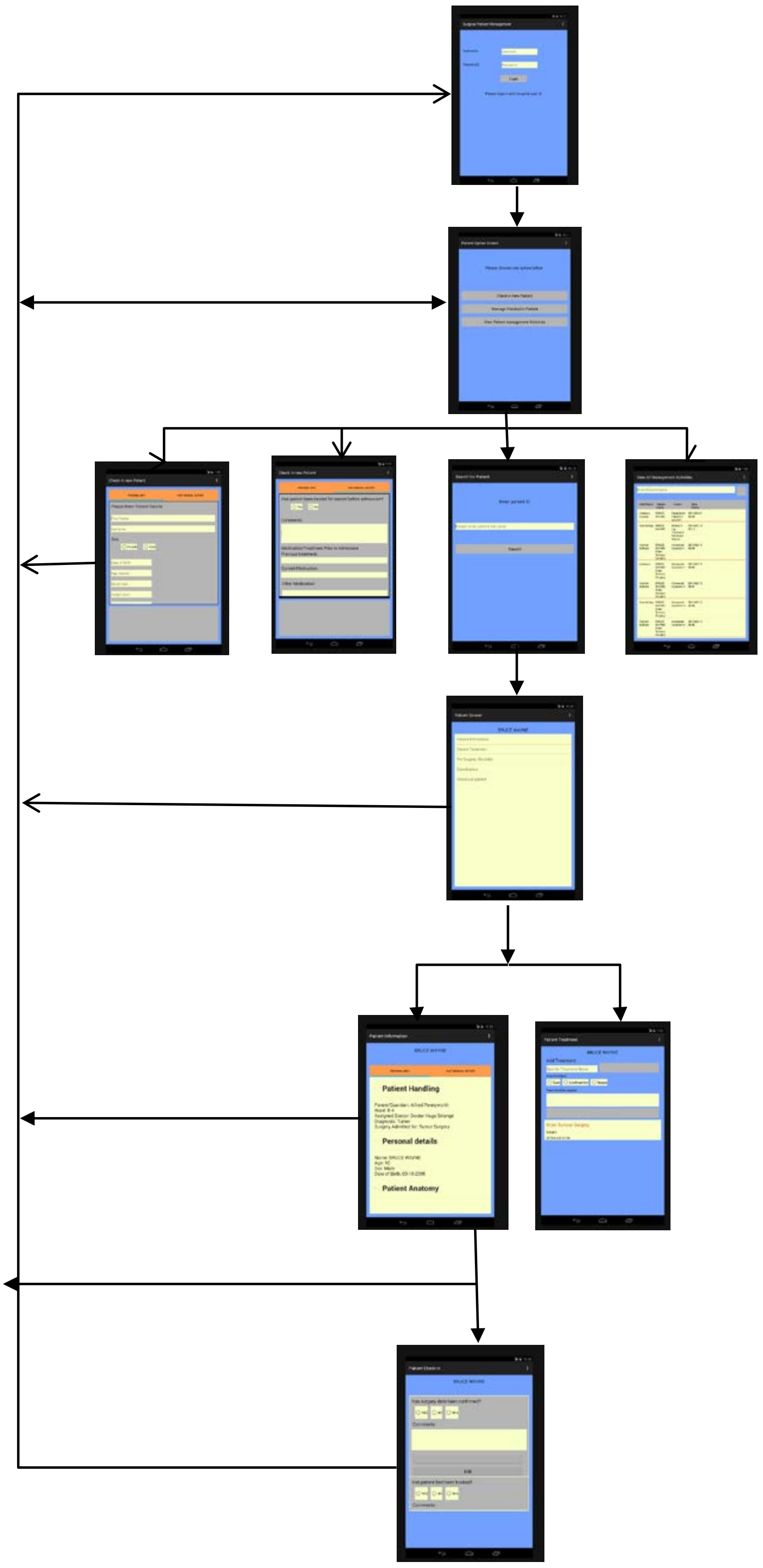


Figure 6-17: Navigation tree diagram of app in operation (unfinished)

To make navigation between activities easier, an option menu was created to help user traverse to the first two activities of the application, i.e. the Login screen and the Patient management options screen. This option menu can be accessed by clicking on the ⋮ button, at the top right corner of the application activity screen.

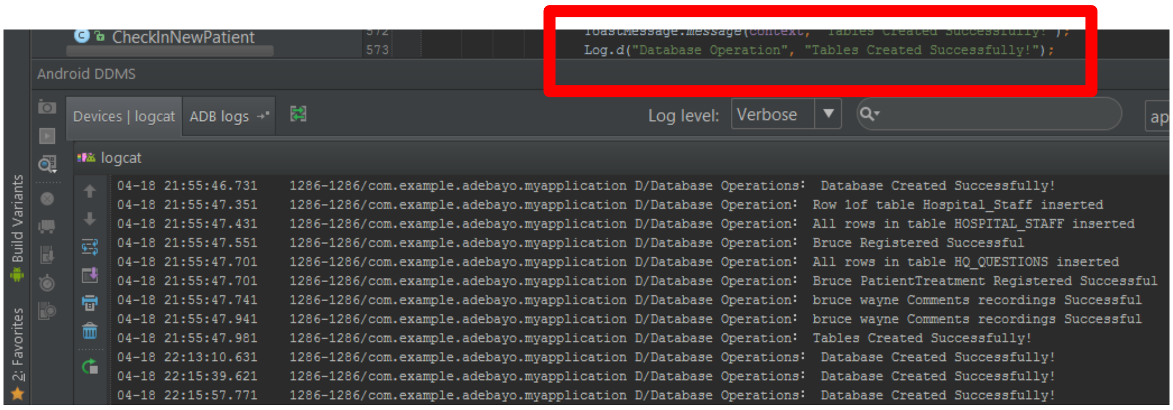
Implementations of the application system discussed above were achieved using Android Studio. It is the official development environment for android application development. It offers expansive features that solely aid in developing suitable android application projects.

7 System Testing and Evaluation

The system and its functionalities underwent numerous testing exercises during the development stages for this project. The tests conducted on the system were both on simulated devices and real android devices, these tests are discussed below.

7.1 Unit Testing

The individual entities like the database and local functions of the mobile application, as well as events to be handled were tested as individual units during the development process of this application system. These tests were performed to check for both syntactic and semantic errors that could arise during development. Testing of units was carried out using the logging facility integral to android studio call "***logcat***". The ***logcat*** allows for creation of logs for an event during app simulation by using the *Log.d()* function call. This function call takes in two parameters; A log tag identifier, and message. This function was used mostly to detect bugs on local function call, print except caught and print results produced by such functions. By using the log tag identifier to filter out desired log outputs, these are then displayed in the ***logcat*** window.



```
Log.d("Database Operation", "Tables Created Successfully!");
```

```
04-18 21:55:46.731   1286-1286/com.example.adebayo.myapplication D/Database Operations:  Database Created Successfully!
04-18 21:55:47.351   1286-1286/com.example.adebayo.myapplication D/Database Operation:  Row 1of table Hospital_Staff inserted
04-18 21:55:47.431   1286-1286/com.example.adebayo.myapplication D/Database Operation:  All rows in table HOSPITAL_STAFF inserted
04-18 21:55:47.551   1286-1286/com.example.adebayo.myapplication D/Database Operation:  Bruce Registered Successful
04-18 21:55:47.701   1286-1286/com.example.adebayo.myapplication D/Database Operation:  All rows in table HQ_QUESTIONS inserted
04-18 21:55:47.701   1286-1286/com.example.adebayo.myapplication D/Database Operation:  Bruce PatientTreatment Registered Successful
04-18 21:55:47.741   1286-1286/com.example.adebayo.myapplication D/Database Operation:  bruce wayne Comments recordings Successful
04-18 21:55:47.941   1286-1286/com.example.adebayo.myapplication D/Database Operation:  bruce wayne Comments recordings Successful
04-18 21:55:47.981   1286-1286/com.example.adebayo.myapplication D/Database Operation:  Tables Created Successfully!
04-18 22:13:10.631   1286-1286/com.example.adebayo.myapplication D/Database Operations:  Database Created Successfully!
04-18 22:15:39.621   1286-1286/com.example.adebayo.myapplication D/Database Operations:  Database Created Successfully!
04-18 22:15:57.771   1286-1286/com.example.adebayo.myapplication D/Database Operations:  Database Created Successfully!
```

Figure 7-1: Log function call and *logcat* window, displaying log user set log output during simulation

Each function was test individually in an attempt to eliminate bugs that might arise in the system, and then the next stage of testing discussed below followed.

7.2 System Integration Testing

On testing individual units of the application system like database creation, button event handling etc. as they were developed, tests on system unit integrations were also conducted to ensure components and functions worked to achieve an expected result. These tests include view (activity) transitions, data retrieval from the database, insertion of data collected from activity widgets, data rendering in activity interface. These tests were conducted on both the android device simulator provided by the android studio IDE and a real android device (7.0' Nexus 7 android tablet).

7.3 System Evaluation

After system integration implementation and testing were completed, the system was evaluated for its usability. The evaluation process and results are discussed below. The application was evaluated on its usefulness as a solution to the problem domain. The usefulness of any product is defined as;

"*The measure of whether the actual uses of a product can achieve the goals the designers intend them to achieve*" – MSDN 2015

The evaluation of the concept system's usefulness was broken down into two;

- Usability
- Utility

7.3.1 Usability

The usability of the system in this context is the measure of how easy it is to use this system to complete prescribed tasks (MSDN 2015) as highlighted in the requirements in section 4.3. The usability of the application system was evaluated using the System Usability Scale (SUS). This is a generic ten statement Likert-scale for subjective assessment, proposed by Brooke (1986) to assess usability. According to ***Usability.gov*** (2015), the SUS assessment has become an industry standard, with citations in over 1300 articles. ***Usability.gov*** also notes that the following are benefits of using the SUS to assess the usability of a system:

- It is a very easy scale to administer to participants.
- It can be used on small sample sizes with reliable results.
- It is valid – it can effectively differentiate between usable and unusable systems.

Therefore, the choice of selecting the SUS as a form of assessment to evaluate the usability of this application system was clear. The SUS questionnaire is shown in Appendix …

7.3.2 Utility

The utility of any system is the ability of that system to perform tasks (MSDN 2015). In the case of this software application system, the utility is the ability of the system to fulfil the functional requirements in section 4.3, and serving as a possible solution to the problem domain associated with patient safety. The utility of the system (as perceived by the client) was measured using a custom made 10-statement questionnaire inspired by the SUS Likert-scale and statement format, each question having its own significance highlighted in the table below. The questionnaire can be seen in appendix …

S/N	Statement	Significance
1	I feel the system improved the pre-surgical management of surgical patients	Assesses the impact the system has on the client's surgical patient management
2	I found it difficult to manage surgical patients using this system.	Assesses how easy it is to use the system for surgical patient management.
3	I feel navigating through activities was consistent	Assess the level of consistency across the system activities
4	I found system navigation difficult.	
5	I feel the application interface was simple and made task completion easy	Assess how the interface impacts task completion
6	I feel task completion on the system was tedious	Assess how easy it was to complete tasks on the system.
7	I feel the system improved accountability amongst staff members	Assess the impact of the application system on accountability as perceived by the client
8	I would not use this system as a replacement for the current paper based system	Assess the application system as a replacement for the clients paper based system currently in use
9	I feel the system has some use in reducing human error and improving patient safety	Assess the clients perception on the system's ability to reduce human error improve patient safety
10	I would not consider this system to be useful in reducing error and improving patient safety	Assess the client's perception of the overall usefulness of the system regarding patient safety

Table 7-1: Utility questions (Statements) and their evaluation significance.

7.3.3 Participants

The only participants for the evaluation exercise were the representatives of the clientele, Dr. Hamish Wallace and Dr. Emma Johnson, who are both practicing professionals in the medical field. Meetings were arranged via the project supervisor, who was in continuous close contact with both the application system developer and the clientele representatives.

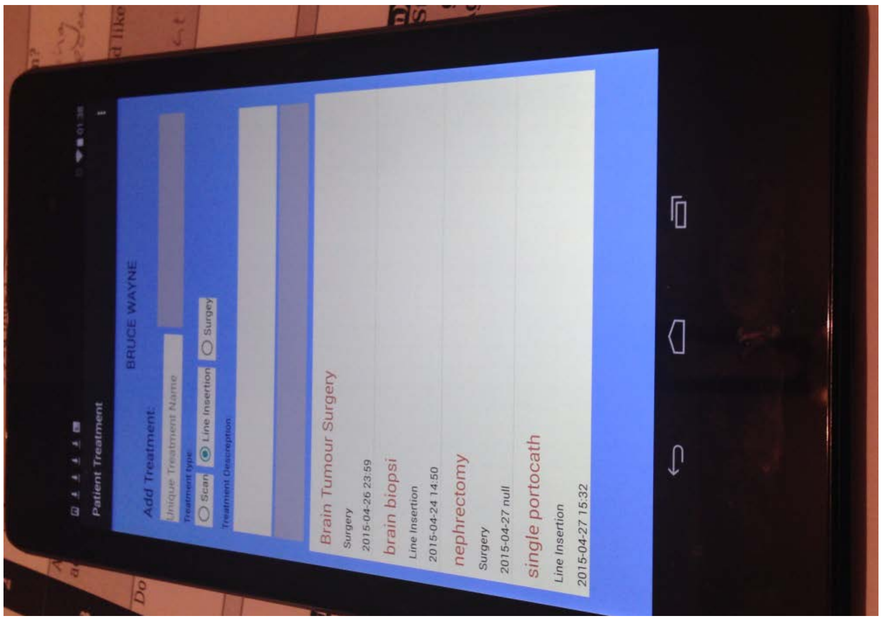
7.3.4 Evaluation Materials

The materials used for the evaluation exercise are highlighted below:

- **Android Device:** An Asus Nexus 7 (2012) tablet was the android device on which the application system was demonstrated and evaluated on. The specification for this device can be seen below.

Asus Nexus 7 (2012)	
Operating System	Android v4.4 (Kit Kat)
Chip set	Quad-core 1.2 GHz Cortex-A9
Screen Size	7.0 inches (~59.6% screen-to-body ratio)
Screen Input	Multi touch, up to 10 fingers
Body dimension	198.5 x 120 x 10.5 mm (7.81 x 4.72 x 0.41 in)
Weight	340 g (11.99 oz)

Table 7-2: Specifications of the Asus Nexus 7 (2012) tablet used for the evaluation exercise (GSM Arena 2015)



Brain Tumour Surgery
Surgery
2015-04-26 23:59
brain biopsi
Line Insertion
2015-04-24 14:50
nephrectomy
Surgery
2015-04-27 null
single portocath
Line Insertion
2015-04-27 15:32

Figure 7-2: Asus Nexus 7 (2012)

- **Questionnaires:** There were three questions used for the evaluation exercise, namely;
  - **System Usability Questionnaire** for measuring system usability.
  - **A custom free form questionnaire** for acquiring feedback for the clientele on the features of the system, difficulties in system operation, suggestions of new features or improvements to current features and any other comments about the system.
  - **A custom Likert-scale questionnaire** to measure utility in terms improving patient safety.

7.3.5 Evaluation Procedure

Evaluation exercises were conducted in two meeting. The first meeting was to acquire initial feedback from the clientele about the first iteration of the application system so detect flaws, fix them and also to ensure requirements of the application system were met.

The second meeting was to acquire a second feedback on second iteration of the system, where client system requirements that were not met in the first iteration were implemented.

After evaluation of the second iteration, the participants were then asked to fill in the questionnaires mention in section 7.3.4.

7.3.6 Evaluation Results

Usability: Although there were not enough participants from the clientele to full utilize the SUS assessment system, the results from both participants gave a rough idea on the usability of the system. The Likert-scale spans between five responses with two extremes; *Strongly Disagree* to *Strongly Agree* and a median of *neither agree or disagree.* This scale can be mapped to numbers between 0-4 depending of positivity or negativity of each statement i.e.

- Positive worded statements (1, 3, 5, 7 and 9) had a direct mapping of the Likert scale to the numbers 0-4 (e.g. *Strongly Disagree* = 0, … *Strongly Agree* = 4).
- While negative worded statements (2, 4, 6, 8, and 10) had an indirect mapping of the Likert scale to the numbers 0-4 (e.g. *Strongly Disagree* = 4,… *Strongly Agree* = 0).
- The maximum score attainable is therefore 40.

For the participants (clientele) the scores for the filled SUS questionnaire can be seen in the table below;

Statement S/N	Dr. Hamish Wallace	Dr. Emma Johnson
1	2	4
2	3	3
3	3	3
4	3	3
5	2	3
6	3	3
7	3	3
8	3	3
9	2	2

10	3	2
Total	**27**	**29**

Table 7-3 SUS score from both client participants.

By multiplying the total for each participant by 2.5 to scale up the original score (0-40) to that of a 0-100 SUS score range;

- The SUS score by Dr. Wallace for the application system is **67.5**.
- The SUS score by Dr. Johnson for the application system is **72.5**.

On average the SUS score is for the system is **70.0**. As pertaining to the clientele, this ranks the application system above average for usability.

Utility: Since the custom made utility questionnaire was inspired by the SUS scale and format, it follows suit that the mapping and scoring follows the same as that described above. The scores for this questionnaire can be seen in the table below.

Statement S/N	Dr. Hamish Wallace	Dr. Emma Johnson
1	2	3
2	2	3
3	3	3
4	3	3
5	3	3
6	3	3
7	2	3
8	3	3
9	3	3
10	3	3
Total	**27**	**30**

Table 7-4: Scores for custom made Likert questionnaire to measure the system's utility

Here also, by multiplying the total for each participant by 2.5 to scale up the original score (0-40) to that of a 0-100 score range;

- The Utility score by Dr. Wallace for the application system is **67.5**.
- The Utility score by Dr. Johnson for the application system is **75**.

On average the Utility score is for the system is **71.25**. As pertaining to the clientele, this ranks the application system above average for utility also.

Free Form Questions: For this part of the questionnaire, the clientele were asked what features were particularly likeable about the application system in the individual opinions. Dr. Wallace liked the comment input field for each checklist question while Dr. Johnson liked that the application system was quick and simple.

When asked for suggestions of features they would like to see added to the application system, they both suggested a summary page, where the checklist and their answers can be viewed at a glance. Both clientele representatives also suggested for the internet of things (I.o.T.) to be possibly implemented in addition to this system, allowing staff members to simultaneously manage patient information from more than one device. Finally, when enquired about overall comments on the application system, both gave a positive response, acknowledging the potential of this system to be useful in improving patient safety by reducing human error during pre-surgical patient management.

8 Conclusion

To conclude this report, this section discusses the key achievements attained on development of the application system developed at the end of this project course by self-critic appraisal. Future works are also highlighted in this section. These are suggestions on additional features that could be implemented to advance this application system, and finally a summary of this report to conclude this section.

8.1 Critical Appraisal

From the results obtained in section 7.3.6, there was an overall positive response from the clientele in terms of usefulness (usability and utility) of the application system. They also offered suggestions for features they would like improved in the system as well as new features they would like to see added.

8.1.1 Key Achievements

Taking a retrospective look at the course of this project subjective to the project objectives identified in section 1.4, the following key achievements can be identified;

- The treatment scheduling feature that provides the user the ability to schedule treatments for a patient.
- The checklist feature, that auto-generates checklist questions from the database based on the type of treatment to be performed on a patient.
- The accountability feature that records all saved input actions of the user.
- The checklist accountability feature that updates who last updated the answer to each question from the checklist per treatment to be performed on each patient.
- The results of the evaluation process showed that the clientele deemed the system useful in both managing pre-surgical patients and improving patient safety in that aspect.
- The architecture adopted for the application system allows for updates to be easily implemented without affecting the core structure of the mobile application.

8.2 Future Works

From the evaluation exercises conducted, the following have been identified as prospective future works on this application system.

- **The internet of things:** The incorporation of a client and server architecture for the system, to allow simultaneous management of patient data on more than one devices. Depending on the extensibility of the server, this could allow prescriptions for medication to passed on to the pharmacists, and also increase the efficiency of the domain environment for the client.
- **A summary page/Activity:** The implementation of a summary page for each patient treatment checklist could allow the users to quickly see what pre-surgery procedures are yet to be completed and which are, at a single glance. This will increase the usability of the system.

8.3 Summary

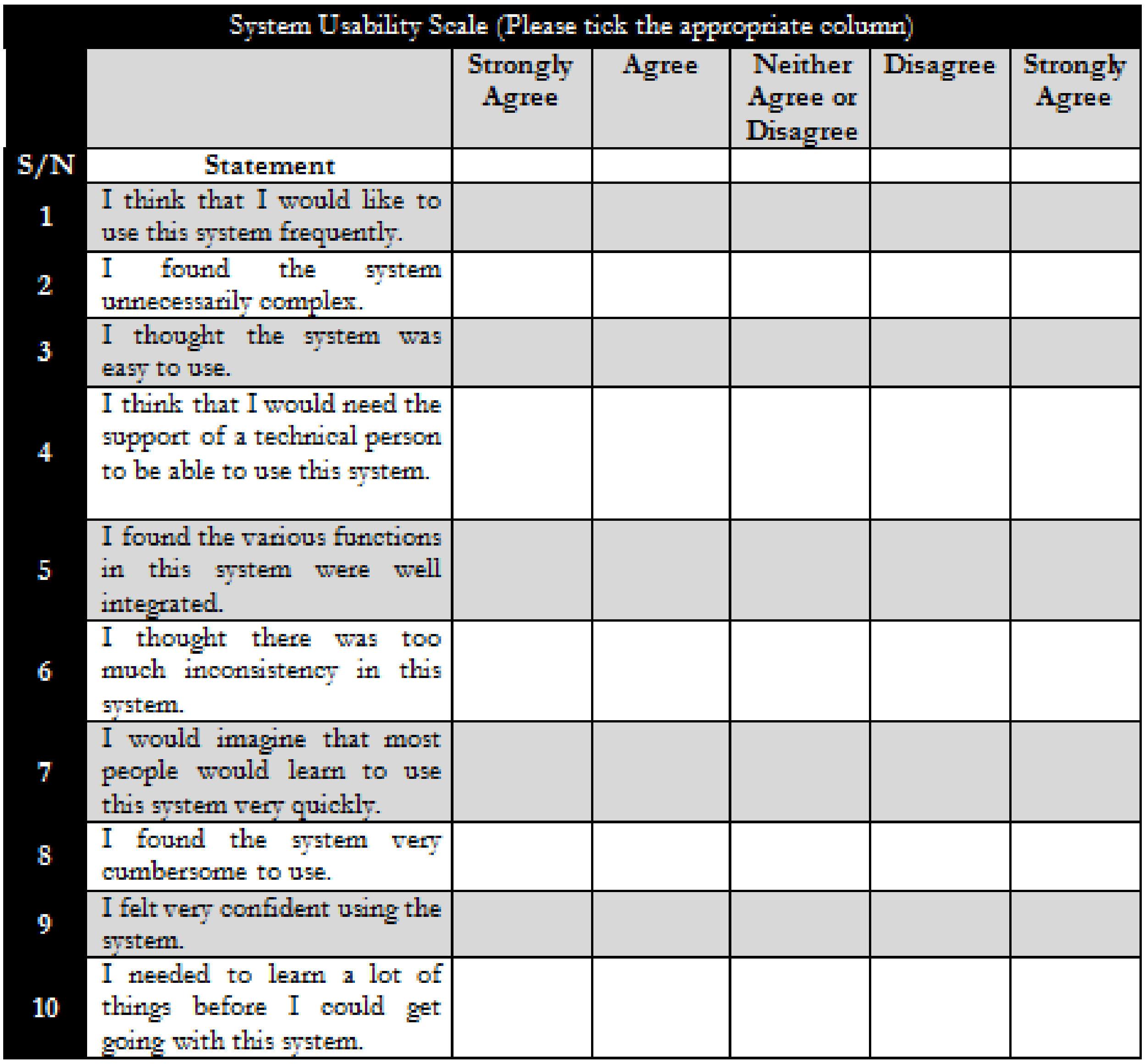
In summary, the aim of this project was to develop a mobile patient safety oriented software application system that can manage treatments and checklists for pre-surgical patients. This document presented a summary of the project development process. This document begins with introducing patient safety as a medical domain and motivation behind this project . Then, a context survey discussing patient safety, the importance and use of IT systems in proving patient safety, especially in regards to software systems that manage patient data transactions, and also. Examples include…….

After the context survey, the methodology and project management approach that were adopted are then discussed, showing the various stages and milestones throughout the project course. Designs for the software architecture of the application system were then discussed, showing the five design perspectives used to create the system's architecture. Based on the architectural design, the system's implementation featured the creation of a database, and a mobile application specific to the client requirements. The system was then tested and evaluated by representatives of the clientele, who are the clinical professionals with extensive understanding of the problem domain. Their evaluation concluded the system as being useful, and also gave some suggestions for the system's improvement.

9 Reference

- Android Developers. (2015). *SQLiteDatabase | Android Developers*. [ONLINE] Available at:http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html.
- Android Developer. (2015). *SQLiteOpenHelper | Android Developers*. [ONLINE] Available at:http://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html.
- A.N. Thomas, U. Panchagnula. (2008). Medication-related patient safety incidents in critical care:a review of reports to the UK National Patient SafetyAgency. *Anesthesia: Journal of the Association of anesthetists of Great Britain and Ireland*, 63, p.726.
- A.W. Buijink, B.J. Visser, L. Marshall. (2012). Medical apps for smartphones: lack of evidence undermines quality and safety. *Evidence-Based Medicine Online First*, p.1.
- B.W. Bates M.D., A.A. Gawande M.D. (2003). Patient safety: Improving Safety with Information Technology. *The new england journal of medicine*, 348, p.2526.
- C. A. Brown, et al.. (2005). "Improving Patient Safety through Information Technology." *Perspectives in Health Information Management* 2004. 2:5
- C. Huckvale et al. (2010). Information technology for patient safety. *Qual Saf Health Care*, 19, p.i25.
- L.L. Leape, D.M. Berwick, D.W. Bates (2002). What Practices Will Improve Safety? Evidence-Based Medicine Meets Patient safety. *JAMA*, 288, 501.
- MSDN (2015). *Model-View-Controller*. Available at: https://msdn.microsoft.com/en-us/library/ff649643.aspx.
- NHS. (2015). *About patient safety* - Patient safety in the NHS- *NHS Choices*. [ONLINE]: http://www.nhs.uk/nhsengland/thenhs/patient-safety/pages/about-patient-safety.aspx.
- Onestoptesting. (2015). *Waterfall Model*. Available at: http://www.onestoptesting.com/sdlc-models/waterfall-model/waterfall-model.asp.
- R.I. Cook, M. Render, D.D. Woods. (2000). Gaps in the continuity of care and progress on patient safety. *BMJ - Education and debate*, 320, p.791.
- SQLite (2015). *SQLite Home Page*. Available at: http://www.sqlite.org/.https://msdn.microsoft.com/en-us/library/ms997577.aspx
- Usability.gov. (2015). *System Usability Scale (SUS) | Usability.gov*. Available at: http://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html.http://www.gsmarena.com/asus_google_nexus_7-4850.php

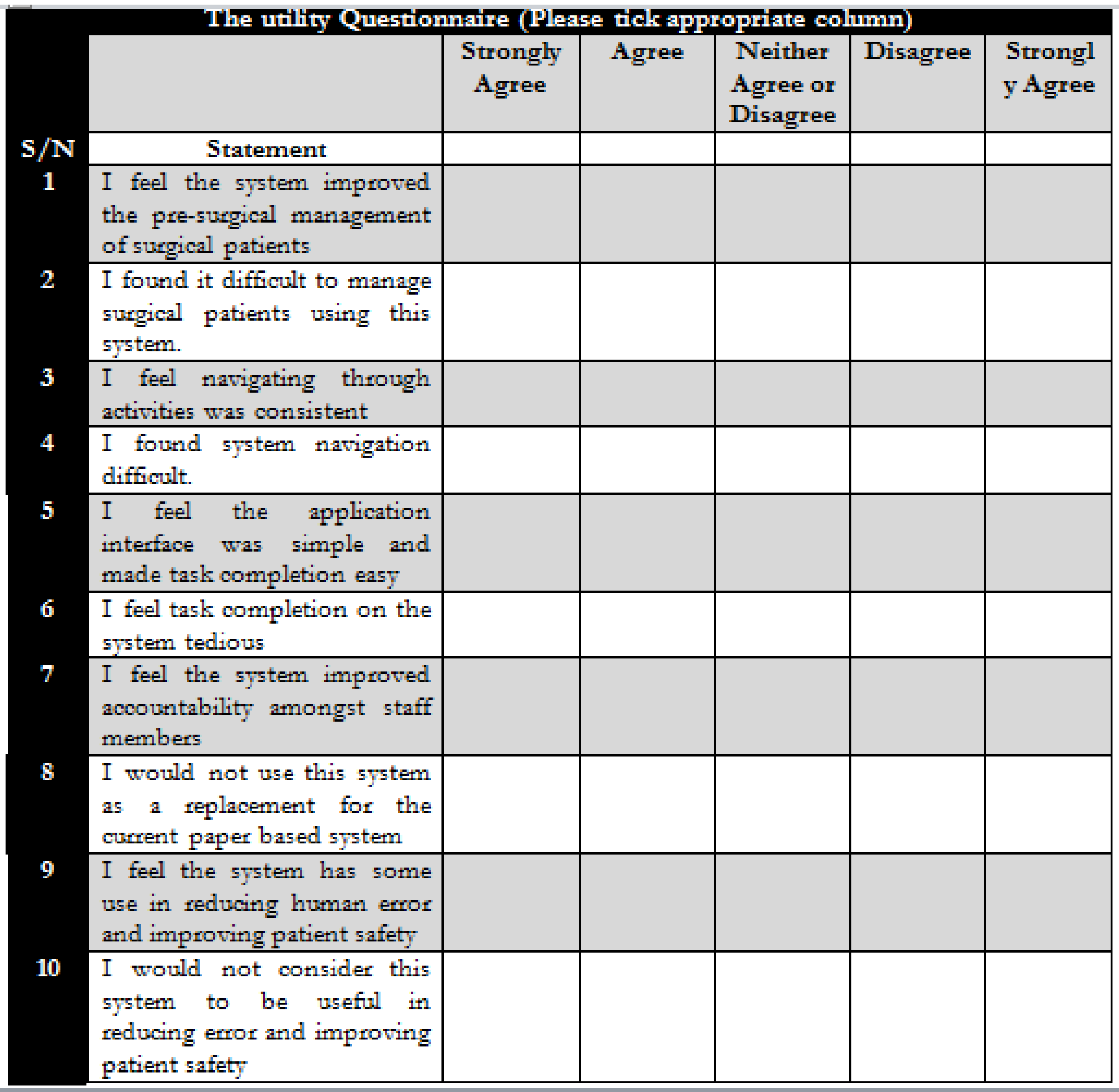
Appendix A – System Usability Scale (SUS) Questionnaire



System Usability Scale (Please tick the appropriate column)

		Strongly Agree	Agree	Neither Agree or Disagree	Disagree	Strongly Agree
S/N	Statement					
1	I think that I would like to use this system frequently.					
2	I found the system unnecessarily complex.					
3	I thought the system was easy to use.					
4	I think that I would need the support of a technical person to be able to use this system.					
5	I found the various functions in this system were well integrated.					
6	I thought there was too much inconsistency in this system.					
7	I would imagine that most people would learn to use this system very quickly.					
8	I found the system very cumbersome to use.					
9	I felt very confident using the system.					
10	I needed to learn a lot of things before I could get going with this system.					

Appendix B – Custom Utility Questionnaire



The utility Questionnaire (Please tick appropriate column)

		Strongly Agree	Agree	Neither Agree or Disagree	Disagree	Strongl y Agree
S/N	Statement					
1	I feel the system improved the pre-surgical management of surgical patients					
2	I found it difficult to manage surgical patients using this system.					
3	I feel navigating through activities was consistent					
4	I found system navigation difficult.					
5	I feel the application interface was simple and made task completion easy					
6	I feel task completion on the system tedious					
7	I feel the system improved accountability amongst staff members					
8	I would not use this system as a replacement for the current paper based system					
9	I feel the system has some use in reducing human error and improving patient safety					
10	I would not consider this system to be useful in reducing error and improving patient safety					

Appendix C – Custom free form Questionnaire

	Free Form Questions
1	Are there any features/aspects of the system you particularly liked?
2	Was there anything you found difficult to do the system?
3	Are there any features not already present that you would like to see added?
4	Do you have any comments about the system in general?

Appendix D – User Manual

- Log in with staff credentials

- Once logged in, the next screen is the "Patient Option Screen", where the user can choose to either register a new patient, Search for an already registered patient or view account log of management activities in the app.

- If "Check in New Patient" button is clicked, the user is then taken to activity where they can enter the details of patient as hinted in the Edit Text field, there are 2 tabs, one for personal details the other contain questions about the patient's past medical history, these details will all be saved by clicking on the button at the bottom.

- Once the details have been saved, the user can go back to the "Patient Option Screen" by clicking the menu at the top-right corner of the screen, which gives the user the option of logging out also.

- Once back at the "Patient Option Screen", the user can then choose to manage a registered patients by clicking on the "Manage Checked-in Patient" button.

- This button opens a "Search Patient" activity that allows the user to search for a patient via the full name using an Edit Text field (currently there is dummy patient called Bruce Wayne in the database). It is to be noted that the patient full name should be entered in 'Firstname' and 'Surname' separated by a space (" ") format, text case is irrelevant.

- Once a patient is found, a "Patient Screen" activity is opened where the user can choose to either view the information details about the patient by selecting "Patient Information" from the list, or select "Patient Treatment" from the same list.

- If the user selects "Patient Treatment", the patient treatment activity is opened up. Here, the user can set schedules of procedure for a patient. This is done by first entering the specific treatment name, and then followed by selecting out from the three types of treatments, then inserting the full description of the treatment in the description edit text field. The user then set a reminder for the treatment by clicking on the "Set time for reminder" button which opens dialogs to set the date and time. Finally after all the inputs are satisfied, the user then clicks on the "+" button add the treatment.

- To get to the checklist of each treatment, the user selects the treatment from the list of treatments scheduled for a patient, this moves the application to the "Patient Pre-surgery

Checklist" activity, where a checklist is auto-generated from the database based on the type of treatment.

- To answer each question of the checklist, the user can select from the three possible answer selection, then press "Edit" button that enables the comment edit text field and the "Save/Update" button, after writing a comment, the user can then click the "Save/Update" to save the answer and comments (the "Last updated Text view is only updated when the check list activity is recalled") for that particular checklist question.
- The user can go back and do the above activities again or log out by selecting log out from the option menu at the upper right hand corner of the screen.