



ASHESI UNIVERSITY COLLEGE

NATIONAL HEALTH INSURANCE MODULE FOR YARESA MOBILE HEALTH APPLICATION

UNDERGRADUATE APPLIED PROJECT

B.Sc. Computer Science

George Esiful Assan

2016

ASHESI UNIVERSITY COLLEGE

National Health Insurance Module for Yaresa Mobile Health Application

UNDERGRADUATE APPLIED PROJECT

Applied project submitted to the Department of Computer Science, Ashesi
University College in partial fulfilment of the requirements for the award of
Bachelor of Science degree in Computer Science

George Esiful Assan

April 2016

DECLARATION

I hereby declare that this [capstone type] is the result of my own original work and that no part of it has been presented for another degree in this university or elsewhere.

Candidate's Signature:

.....

Candidate's Name:

.....

Date:

I hereby declare that preparation and presentation of this [capstone type] were supervised in accordance with the guidelines on supervision of [capstone type] laid down by Ashesi University College.

Supervisor's Signature:

.....

Supervisor's Name:

.....

Date:

Acknowledgement

The author wishes to thank several people. I would first and foremost like to thank my family for the love, kindness and support they have shown during the past four years at Ashesi University College. I would also like to extend my deepest gratitude to my supervisor Mr Aelaf Dafla for all the guidance, support and time he dedicated to ensuring the success of my project.

Finally, I would like to thank my colleagues especially Christian Biassey-Bogart, David Tandoh and Makafui Amezah for helping in the testing and review of the application.

Abstract

As a middle income country, Ghana has made formidable strides in improving health care delivery in the country. Notable among them is the introduction of the Community Health and Planning Services system. This system was adopted to increase access to and use of health care services by setting up health centers known as CHPS zones in the remote communities where it is particularly difficult for Ghanaians to receive proper and timely medical attention. These health centers are manned by few trained healthcare officers whose tasks are not limited to just providing health care to patients but involve book keeping, report drafting and a number of other more administrative tasks which takes a lot of time away from providing actual health care services to patients. One such task identified by health officers is the preparation of NHIS insurance claims.

An ongoing Mobile Health project for CHPS zones has been developing an application to assist health officers in performing some of these tasks. However this does not include preparation of NHIS insurance claims which provides the basis for the project that this report discusses. The aim of the project is to provide functionality to the existing Mobile Health application that facilitates the preparation of NHIS insurance claims.

Table of Contents

Chapter 1: Introduction	1
1.1 Introduction	1
1.2 Background	2
1.3 Objective	3
Chapter 2: Requirements Specification	4
2.1 Overview of Chapter	4
2.2 Product Scope.....	4
2.3 Feasibility	4
2.4 Product Perspective	5
2.5 Product Functions.....	6
2.6 User classes and Characteristics.....	7
2.7 Operating Environment	7
2.8 Assumptions and Dependencies.....	8
2.9 Requirement Gathering Procedures	8
2.10 Requirement Analysis	8
2.11.1 Scenario A	10
2.11.2 Scenario B	11
2.12 Deductions from Scenarios	12
2.13 Functional and Non-Functional Requirement	13
2.13.1 Functional Requirements.....	13
2.13.2 Non- Functional Requirements	14
Chapter 3: Architecture.....	15
3.1 Overview of Chapter	15
3.2 Architectural Goals and Constraints	15
3.3 System Overview	16
3.4 Architectural design	17
3.4.1 View	18
3.4.2 Controller	18
3.4.3 Model	18
3.5 Database Architecture	18
3.5.1 Database design decisions.....	19
3.5.1.1 Information required to be stored by virtue of requirement analysis.....	19

3.5.1.2 Multiple records for procedures, investigation and medicines	20
3.5.1.3 Maintenance of existing table design.....	20
Chapter 4: Implementation	21
4.1 Overview of Chapter	21
4.2 Implementation Environment.....	21
4.2.1 Development environment setup.....	21
3.2.1 Version management.....	22
4.3 Implementation approach.....	22
4.3.1 Database implementation	22
4.3.2 Implementation of model and controller classes.....	25
4.3.3 Core functionalities	26
4.3.3.1 Adding claims	26
4.3.3.2 Viewing of recorded claims.	30
4.3.3.3 Updating of Recorded Claims	31
4.3.3.3 Viewing of claim report	31
Chapter 5: Testing and Results	33
5.1 Unit Testing	33
5.1.1 Table creation and alteration	33
5.1.2 Data entry and Retrieval	33
5.2 Integration tests.....	34
5.3 User Testing.....	34
5.4 Results.....	34
Chapter 6: Conclusion.....	37
6.2 Project Review	37
6.2 Challenges	37
6.3 Further work.....	38
6.3.1 Configuring of Eclipse project in Android Studio.....	38
6.3.2 Limitations with working on existing project.....	38
References.....	39
Appendix.....	40

List of Abbreviations

CHO – Community Health Worker

CHPS – Community-based Health and Planning Services

mHealth - Mobile Health

NHIS – National Health Insurance Scheme

NHIA – National Health Insurance Authority

OPD - Out Patient Department

GDRG – German Diagnosis Related Code

XML - Extensible Markup Language

IDE – Integrated Development Environment

SQL - Structured Query Language

Chapter 1: Introduction

1.1 Introduction

The citizens of any country are undeniably an important resource for its development. The correlation is a simple one – better quality of life of people means higher productivity. The larger the number of unhealthy people within a population means a reduction in its total workforce and consequently a decrease in the total output from the country. In view of this, the quality in Health care delivery is a key indicator in measuring the developmental challenges of a country. It is therefore a mission for most countries to put up the structures that allow access to quality healthcare to all.

As a middle income country, health care in Ghana is very variable through the country. With its population of about 22.4 million, about 60% of Ghanaian's participate in subsistence farming and an astounding 80% work in the informal sector as farmers, fishermen, or roadside vendors. These class of people are mostly impoverished individuals living in remote villages (Russell, 2008). With a large percentage of its population living in rural areas, urban areas still experience better healthcare services, having the most hospitals, clinics, pharmacies and health officers in the country (Central Intelligence Agency, 2015). The rural areas often have no modern health care. The challenge here is that patients in these areas have to travel great distances to have access to healthcare services. Also there is the increased chances that they may not reach on time to access medical care when needed. The unattractiveness of visiting these health centers due to distance coupled with existing cultural practices and beliefs make most people opt for traditional medicine. To increase the access of health care in these rural areas, Ghana has adopted the Community-based Health and Planning Services (CHPS) system. This system aims to bring trained health workers

directly into the communities while mobilizing volunteerism, resources and cultural institutions for supporting community-based primary health care (Russell, 2008).

Starting out as an experiment in the Northern region of Navrongo where it was a success, it has been extended to other regions and districts nationwide (Russell, 2008). The system despite being revolutionary in the sense that it rallies community support and acceptance to ensure sustainability, still has issues with its management and health care delivery to the people (Russell, 2008).

1.2 Background

In an attempt to improve the health care delivery from the CHPS system, an ongoing mHealth project for CHPS identified an inefficiency in the reporting and management of data compiled from people within the community. The current system requires a lot of time and effort. For example, the compiling of monthly reports requires an extensive amount of effort reviewing documents and files. Transfer of data between the various levels of the system requires the actual carrying of reports, documents and files from one office to the other. These tasks take away from time that can be used to attend to health related issues in the community.

In light of this, the CS department of Ashesi University under the mHealth for CHPS project has been working with the health directorate of Akuapim South in developing mobile and web applications to use on tablets and laptops as a tool for improving efficiency of health officers. Currently an Android application named Yaresa has been deployed for use by health officers to manage their data. Feedback from health workers has been promising. They confirm significant reductions in the compilation of summary reports on OPD case attendance, immunization and family planning.

1.3 Objective

Regardless of the work done on the app so far, further work is required. Among these is the implementation of a module for compiling NHIS insurance claims which is the objective of my project. This necessary feature is based on feedback from health officers who have expressed how preparation of insurance claims for the month is extremely time consuming. They are tasked with calculating the costs of each claim made with patient visits to the health center, computing the total cost of claims for the month and the submission of the claims with summary reports to district management for NHIA processing. Aside being time consuming, its labor intensiveness gives room for a large number of errors which leads to the rejection of some of these submitted claims. In view of this, the NHIA module once implemented will seek to reduce the time spent on compiling insurance claims as well as the number of errors by automatically performing the computations involved with preparing the claims for submission. Eventually, this will help reduce the amount of paper work for health officers in the health centers thereby making available more time to dedicate to meeting the health care needs of community members.

Chapter 2: Requirements Specification

2.1 Overview of Chapter

This chapter gives an overview of the design decisions undertaken during the development of the NHIS module for the existing Yaresa application. It will provide a background for the requirements of the module, discuss in detail its main functions, the various components of the module and their interactions with one another and the conditions under which the system would function.

2.2 Product Scope

Upon completion, the NHIS module for the Yaresa application will provide a platform that will enable CHOs gather the necessary information from patient visitations for making and submitting of insurance claims to NHIA for processing via their mobile devices. The NHIS insurance claim forms provide a template that define the inputs for recording a single claim for an OPD case. These claims will be compiled on a monthly basis. An important feature of this module will be the automatic computation of costs per claim and the overall costs of claims per month. This seeks to replace the intensive process involved in performing these calculations manually on paper.

2.3 Feasibility

The existing application for which this module is being added to, has been developed primarily using the Android platform. Android is open source therefore the tools necessary for development are free and readily available for reference, thereby reducing any costs involved in making new purchases.

At present, mobile devices with the application have been provided to the 24 CHPS zones under this MHealth project. The addition of this module will not require any system specifications not

already supported by the mobile devices currently in use. As such, no new set of mobile devices will be required for the implementation of the module thus reducing implementation costs as well.

The similarity in the processes involved in compiling of NHIS claims between the various CHPS zones reduces costs involved in designing multiple systems to meet the needs of each CHPS zone.

These positives provide a basis to assert that the development and deployment of this module for the existing Yaresa application is feasible.

2.4 Product Perspective

The system is to be integrated into the existing Yaresa application which currently has modules that enable CHOs to receive and send resource material on health care practices to and from the District Health services, record digital health information on OPD cases, immunization and family planning and management of community health related tasks.

As already stated, the NHIS module will facilitate the recording, compiling and electronic submission of NHIS insurance claims using the application. The module will not be stand-alone, independent of the existing modules of the application. Rather it will be an extension of the Record module of the application. The Record module enables CHOs to record data on OPD case diagnoses of clients that visit the health center. The data recorded includes the basic information of patients such as their full name, birthdate and NHIS identification card number. A single NHIS insurance claim requires this basic information of the client and OPD case records. In view of this the NHIS module will be developed on top of the OPD case recording function of the Record module. However, it would be modified to include the other necessary information required for adding an insurance claim.

As a data-centric product, the application will need somewhere to store the data. For this, provision will be made for storing this data in the application's database. The above mentioned relationship between the NHIS module and the Record module implies that the existing database will be modified to allow the NHIS module to access data currently stored by the Record module.

The electronic submission of the NHIS insurance claims to NHIA for processing will not be done by the application itself. Rather the application will generate an NHIA standard xml document of each claim being submitted. These will then be uploaded by CHOs onto the NHIA website.

2.5 Product Functions

Compiling and Submission of NHIS claims by CHOs involve the following key tasks:

- Recording claim details – This involves the filling of the NHIS claim forms specific to a patient's case.
- Calculating of cost per claim – The cost of each NHIA acknowledged procedure, service or medicine provided to patient is accounted for and the total sum computed as the cost of the claim.
- Monthly compilation – A summary of the month's claims is made. The total costs of the claims is computed.
- Submitting of insurance claims – All claims for the month, as well as a summary of these claims are handed over to the District Authority for NHIA to be sent for processing.

These tasks serve as the basic functions that the application would perform.

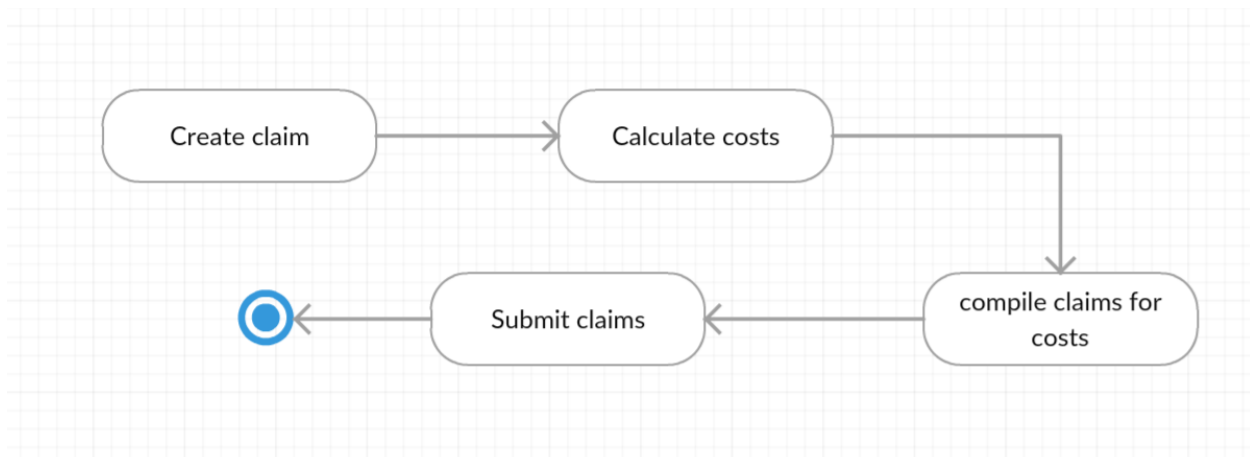


Figure 2.1: Activity diagram for NHIS claim processing tasks

2.6 User classes and Characteristics

The primary users of the module will be the CHOs of the CHPS zones currently using the application on the provided mobile devices. As certified Health professionals, their level of training evident in their ability to perform the tasks stated in the previous subsection, qualifies them to use the NHIS module appropriately and efficiently. This coupled with their conversance with the application reduces the learning curve for using the module. Hence no intensive training would be required.

Personal characteristics of the user such as age, sex, ethnic background and race have no relevance to the operation of the module.

2.7 Operating Environment

The module will be an addition to the existing application which runs on the Android Operating system. The application is meant to provide support to the health processes carried out in the Health centers however its operation is not limited to the center alone. As such, the use of the module has no dependence on the location of CHPS zone. Nevertheless, the application does recognize the limited internet connectivity experienced in remote areas and caters for this with the inclusion of

a local database. This reduces the dependence on internet to perform the application's functions. In view of this, the NHIS module would be developed in conformance to this system.

2.8 Assumptions and Dependencies

- The system assumes that the CHOs are using android tablets pre-installed with a version of the application for which the module is being modified.
- The system also assumes that CHOs are conversant with the NHIA health processes conducted in the health centers.

2.9 Requirement Gathering Procedures

The processes involved in gathering information relevant to the development of the module are as follows:

- Interview with CHO – This was aimed at understanding processes involved in preparing insurance claims for submission.
- Studying of Insurance Claim forms – This provided insight on the inputs and outputs expected for system features.
- Review of documentation of on NHIS Claim XML format - This also provided insight on the inputs and outputs of system features as well as information on the accepted XML format of claims submitted to NHIA for processing.

2.10 Requirement Analysis

From the information gathered during requirement elicitation, the tasks involved in the processing of the NHIS claims begins with filling the NHIS claim form. On this form, the first section requests for the client's relevant health insurance information usually obtained from the client's insurance card details. In the next section, details on the type of service, outcome of the case, and dates of

service provision are demanded. With respect to these dates, space is made for four entries indicating that a number of four visits can be made by a client in a single claim. The subsequent dates to the first entry are to be considered as follow-ups to an initial diagnosis made on the first visit. The following section allows CHOs to provide information on the diagnosis made for the client, the procedures performed on the client, the drugs prescribed and investigations carried out on client cases. For each of these in this section of the form, multiple entries can also be made. As such, a client can be diagnosed for more than one of the limited OPD cases that the center is allowed to handle. The reason for the limited number of OPD cases is that unlike larger urban hospitals and clinics, CHPS zones provide primary healthcare only. As such they are restricted to handling only commonly experienced diseases in the rural communities they operate in.

The point of the claim is to serve as a bill for health services provided. As such, the charges of these services are tracked. Each service item, that is a diagnosis, procedure, investigation or drug prescribed has a specific charge determined by way of their GDRG code for which each of these is assigned. Upon completion of the form, a summary of the claim's charge is provided which is the summation of each of the entered service items.

The forms are filled and updated upon client visitations. The filled forms within a given month are compiled into a single bundle of paper claims. This introduces the need for such a function in NHIS module. One that does such bundling to a set of claims. A summary of all claim details within a bundle is also prepared on a separate sheet. On this sheet, the month within which this claim has been bundled under is specified. The rows of information on the sheet represent each claim in the bundle. Each row specifies the calculated total charge of the claim and the insurance number of the client. This indicates the familiar way in which such claims within a bundle should be viewed by a CHO on the application.

As previously stated, the submission of the bundled claims will not be done by the application itself. The claims will be submitted online by uploading it on the NHIA website. For electronic submissions, NHIA accepts claims in a standard xml format. In view of this, claims created on the application would have to be converted to this xml format prior to submission.

Once claims are submitted they cannot be updated. As such, submitted claims must be differentiated from those that are still pending. For this reason, each claim must have a status attribute that states if it is either closed or open. An open claim would mean that the claim has not been submitted yet and can be updated while a closed claim would mean otherwise.

2.11 Scenarios

The use of the NHIS module can be better understood and described properly if the appropriate system scenarios are envisaged. The minimum set of scenarios have been defined with the intention of providing the basis for which the module may be tested upon implementation. They describe the circumstances within which CHOs may use the application.

The first scenario describes a possible event for which a CHO may use the application to record an insurance claim. It outlines the sequence of actions that a CHO will need to take, as well as the inputs that the system will require to successfully save a single claim into the database.

The second scenario assumes that it is the end of a service month and a number of claims have already been recorded. For that, there is a need to prepare and compile these claims to be submitted to NHIA. The scenario outlines how a CHO will ideally use the application to perform this task.

2.11.1 Scenario A. Akosua Boahemaa is a member of a community on the National Health Insurance Scheme and an existing client of the community health center. At the beginning of the month, she visits the health center because she has been complaining of a stomach ache that has

been persistent for a week. She is received by a health officer who uses the mobile application to access her health profile at the center. Upon examining Akosua, the health officer proceeds to record the diagnosis into the application. To do this, the health officer fills a form that opens up upon tapping the record button. The inputs required for filling this form are similar to that on the paper. The health officer inputs the diagnosis, procedure performed and the drugs prescribed. Upon saving this, a new insurance claim with these details is recorded

2.11.2 Scenario B. During the course of the month, other members in the community also on the NHIS scheme visit the Health center with different health problems. The health officers that received these clients go through a similar process for adding each handled case as a claim into the application's database just as was done with Akosua Boahemaa's case in the previous scenario. It is the end of the month, time for all claims to be compiled and a report generated. This includes calculating the total charge for all services provided for a patient be it diagnosis given, procedures performed, investigations conducted or drugs prescribed. After this the total service charge for all claims within the month is computed as part of the summary. For the convenience of the health officer, she does not have to make any of these calculations when generating the report. The application handles this computation and generates a summary report of the claims for the month. The health officer would simply view the generated report or summary of the claims for the month and upon a click submit this data for NHIA processing.

2.12 Deductions from Scenarios

Outlined in this section are use case and activity diagrams that model the actions that take place within this system.

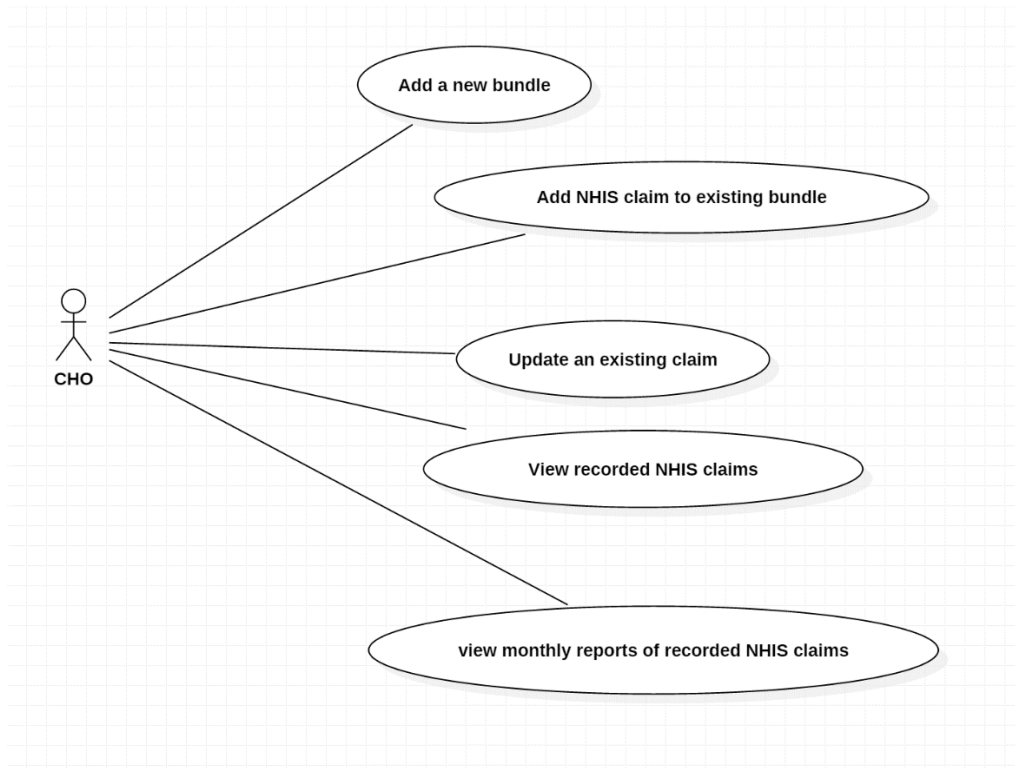


Figure 1.2: Use case diagram for CHO in NHIS module

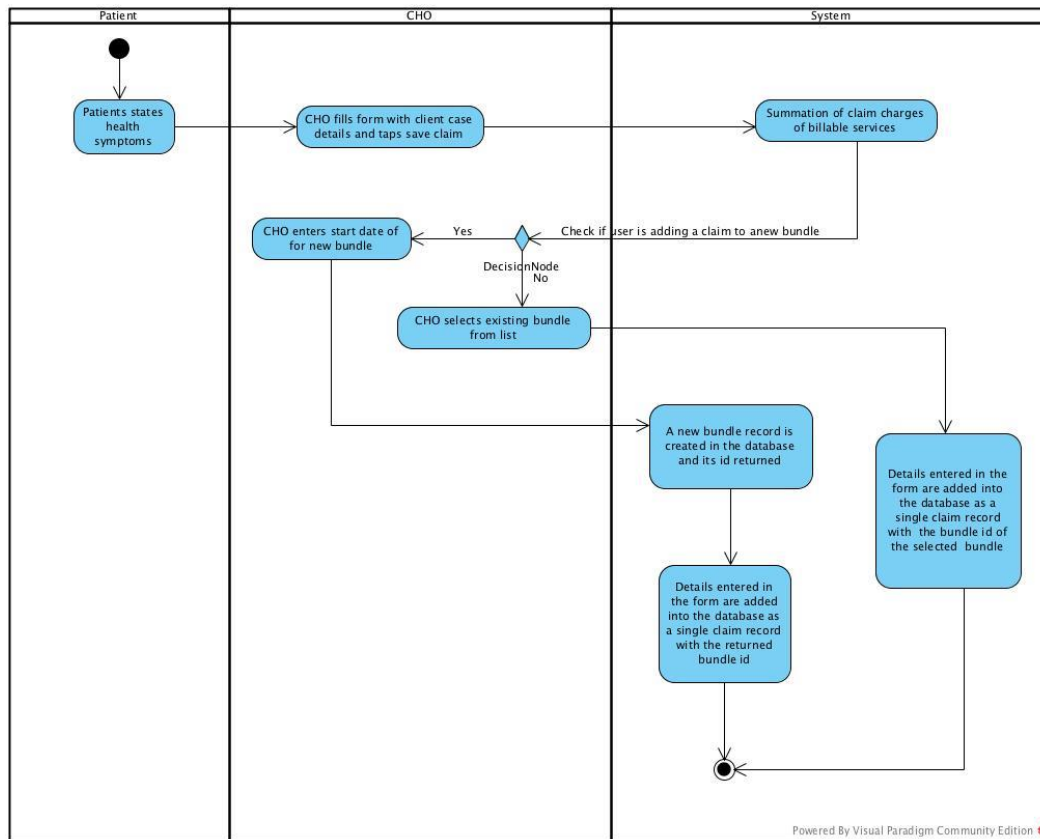


Figure 2.2: Activity diagram for adding a claim in the system

2.13 Functional and Non-Functional Requirement

2.13.1 Functional Requirements

- a. The user should be able input the necessary information for a claim and add it as a single insurance claim record.
- b. The user should be able to view recorded claims.
- c. The user should be able to view automatically generated monthly reports of claims.
- d. The user should be able to create a bundle.
- e. The user should be able to update a claim

- f. The user should be able submit claims.

2.13.2 Non-Functional Requirements

- a. The system should be accessible offline
- b. The system should be able to identify the insurance service code of any service included in an insurance claim.
- c. The system should be able to calculate costs of services provided and medicines prescribed based on the charges specific to their insurance codes
- d. As a result of (b) & (c), the system must store information on the insurance codes and their corresponding charges.
- e. The system must be able to compute total costs of claims for generating monthly reports.
- f. For submission to NHIS for processing, the generated insurance reports must be converted into a standard xml data format supported by NHIA.
- g. Module must conform to NHIA policies.
- h. The module must be scalable and extensible. That is, due to the fact that the system provides a service highly dependent on the insurance system of the NHIA, it must be able easily evolve to meet new user and system requirements defined by NHIA.

Chapter 3: Architecture

3.1 Overview of Chapter

This chapter discusses the structures of the system, which comprise of the product's components, the externally visible properties of those components, and the relationships between them. It also describes the architectural constraints of the system, and the significant impact that the functional requirements has on the architectural and database design

3.2 Architectural Goals and Constraints

This section describes the software requirements and objectives that have some significant impact on the architecture of the module.

- Technical platform - The application is being developed in Android hence must satisfy the android platform constraints. Failure to do so will mean the system does not meet stakeholders' needs. In view of this, the architecture designed for the application must satisfy these platform constraints as well as make good use of system resources
- Conformity with whole application - The application code must be uniform, as such the architecture designed must conform to the existing architecture of the application. Modifications made to the design for implementing this module cannot be far from the parent structure adopted by previous contributors to the project.
- Scalability/Extensibility – The architecture design must facilitate the easy addition of subsequent modules to the project. There must be reduced costs to making changes.
- Persistence – The application would persist data. The nature of data to be stored would define the database design as well as the architecture of the application code that would interface with it

- Testability – The architecture designed for the module must facilitate easy testing and debugging of application code.

3.3 System Overview

The figure below shows the general deployment diagram of the whole application for which the NHIA module is to conform to.

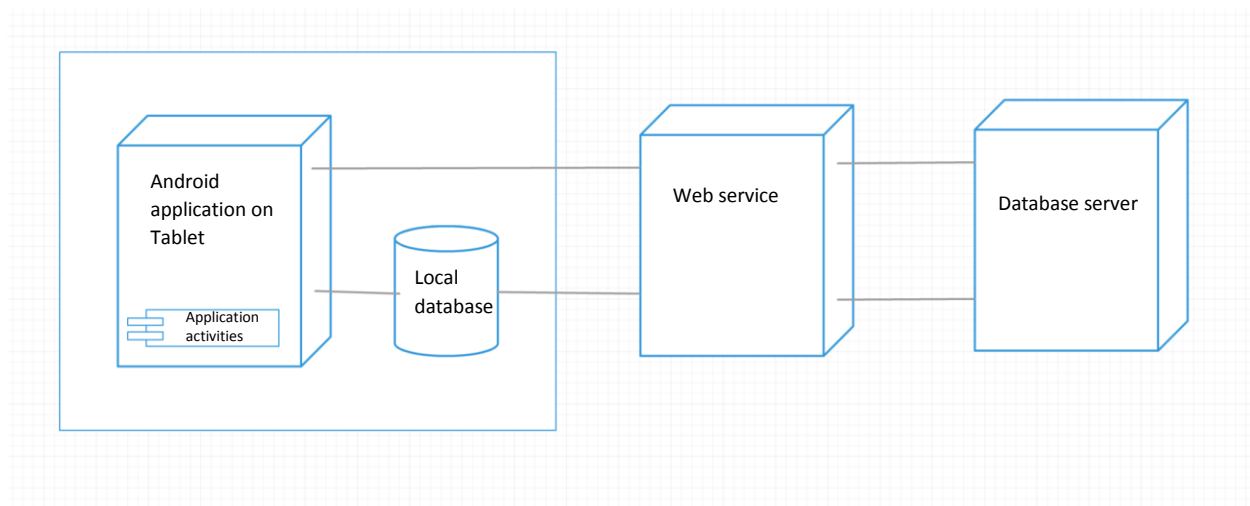


Figure 3.1: Deployment diagram for overall system

The local storage has been deployed to provide availability of the application's data. As already mentioned, internet connectivity in this CHPS zones is poor, as such there is a need to keep a local copy of the data in order to facilitate the operational duties of CHOs.

The web service connects to the database and acts as a bridge between the android application and the database, managing database operations like CRUD.

Both local and external databases are both necessary for storing data used by the NHIA module.

3.4 Architectural design

The overall application has been developed using Android. Android is designed to make heavy use of the model-view-controller (MVC) pattern. This divides the application code into three main entities to achieve a modular design. The application code of the existing Yaresa app currently conforms to the MVC pattern.

As a result of the structural advantages of the design pattern and the importance of maintaining uniformity within the application code, the MVC pattern will also be adopted for the NHIA module.

The aim of this pattern is to separate the components of user interface (View); business logic and data (Model) and the response to user inputs (Controller). This way there is a clean separation of concerns that makes testing during implementation easier and more efficient. Also, a key goal of the project is to promote scalability and extensibility of the application to make it easier for further work to be done on the module after this project.

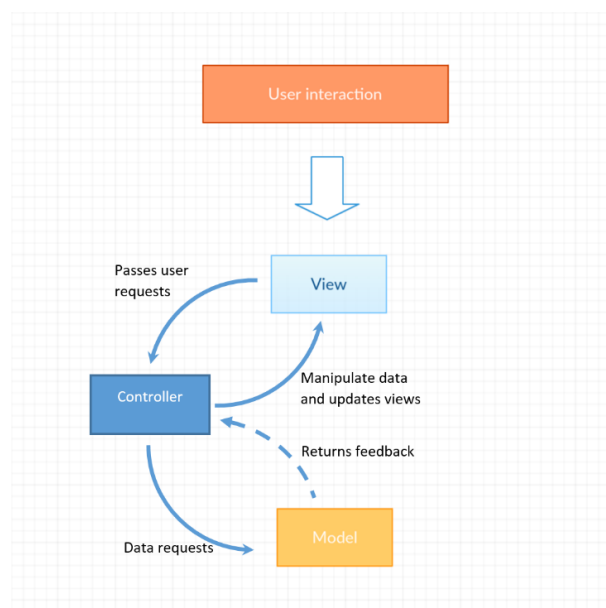


Figure 3.2: MVC Architecture for system

3.4.1 View. This entity includes the visual components that the user would interface with. It will not have any logic or knowledge of the data displayed. The View will know nothing about the business logic or how to retrieve the data from storage. Its primary functions will be to display the visual components and present the data to user. For the android project this includes xml files and activity classes that define and control the NHIS claim form view and other views displayed by application.

3.4.2 Controller. This entity provides the logic for controlling when, what and how to display information to the user. It is responsible for capturing and performing the associated actions to the events made by user during interaction with the user interface. It also acts as a bridge between the View and the Model fetching any data needed from the data layer, making any necessary or requested transformations and display the resulting data in the appropriate visual components for the users viewing.

3.4.3 Model. This entity manages the business layer logic and data access from the online web service and local database. The models represent the classes of the android project that work with data from the database. When data is returned from the database via functions the data is set into the class variables which are used as objects by the controller.

3.5 Database Architecture

This section discusses the decisions that were made when designing the database for the NHIS module including problems, alternative solutions, and design assumptions that had to be made. The result of these decisions is seen in the database modeled in the figure below.

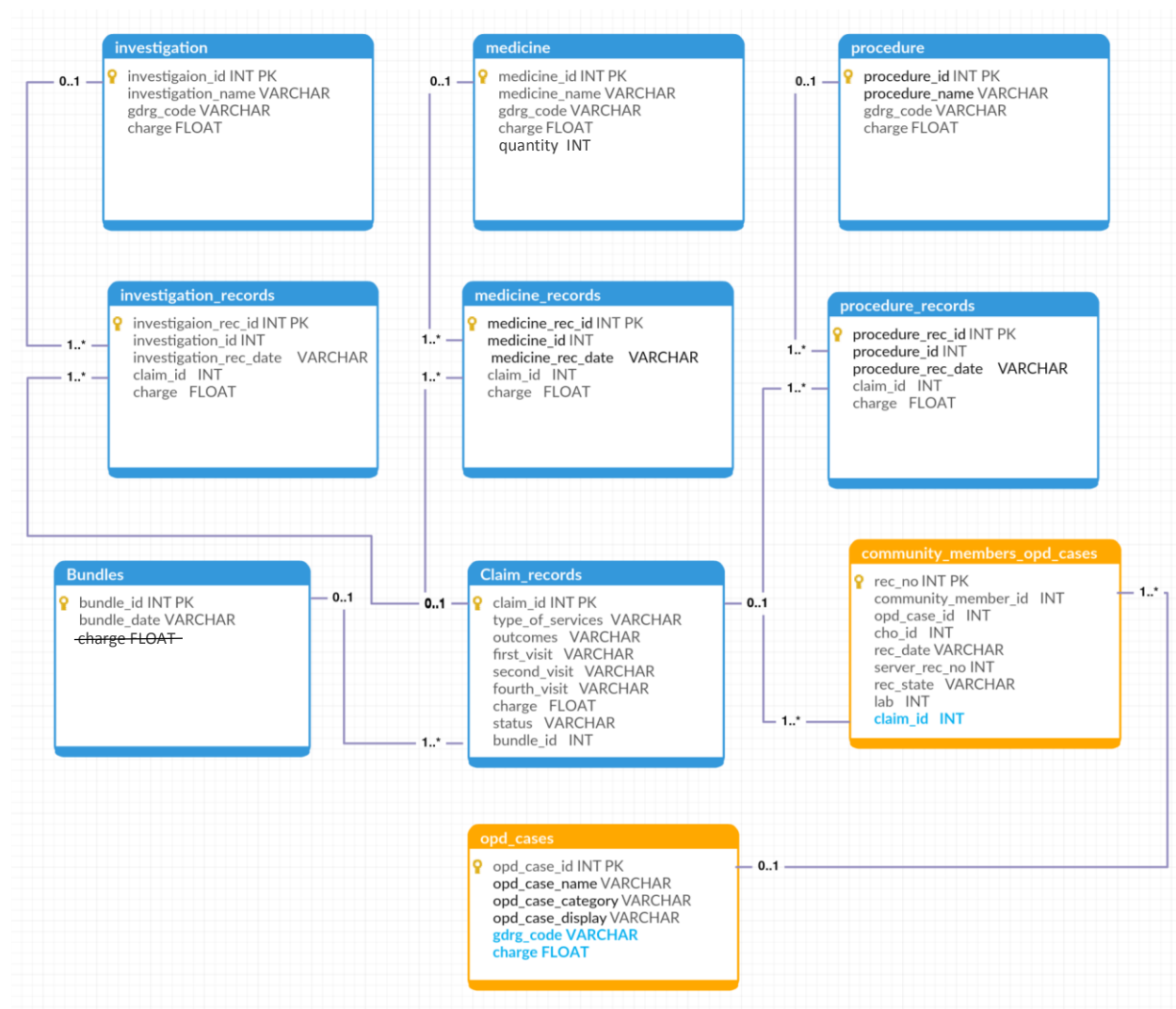


Figure 3.3: Database design

3.5.1 Database design decisions

3.5.1.1 Information required to be stored by virtue of requirement analysis. The most probable table to create is a table to record claims hence the claim_records table in the database model. From the requirement analysis, it was determined that a claim record requires information on the type of service, outcomes, date of visitations (which according to the paper claim form allows four visitations), claim charge, status of the claim, medicines, procedures, investigation and diagnosis provided. In view of this, the claim_records table has columns to hold these values.

3.5.1.2 Multiple records for procedures, investigation and medicines. It can be observed that the claim_records table does not store data on medicines, procedures and investigation specified in form. This is because multiple data can be stored for each of them hence their information cannot be stored in a single row. The primary key constraint in the claim_records table also prevents the insertion of rows of data for each instance of the attributes. As such they each must be viewed as an entity on its own with a reference to the associated claim records. This explains the introduction of the procedure_records, investigation_records and medicine_records. No mention is made of the diagnosis_records table because it is explained in the next design decision

3.5.1.2 Maintenance of existing table design. The module is to be integrated into the application and so changes made must not drastically affect existing system functionalities. As such, rather than create new tables, tables with some relevant information for a necessary record of the claim component are altered instead. This also aims to avoid duplicates in data. For example, rather than create a diagnosis_records table the community_opd_case table which stores some of the necessary information for a diagnosis record was altered by adding new column which stores integer values that reference the claim id of the associated claim. In the figure above, altered tables have been given the orange color with the added columns in blue.

Chapter 4: Implementation

4.1 Overview of Chapter

The chapter describes the decisions made and processes involved in integrating the designed NHIA module into the existing Yaresa application. It discusses the development approach for ensuring the proper operation of the module which include, database setup, development environment configuration, running and making of necessary design changes based on implementation requirements.

4.2 Implementation Environment

This subsection describes the components required to support and facilitate the implementation process

4.2.1 Development environment setup. An android development environment will be required to build and integrate the module into the existing android application. To do this I will need to:

- Install Android SDK and related tools – This includes android sdk platform tools for android API level 22
- Installing of Android Studio - Prior work that was done on the application code was implemented in the Eclipse IDE with Android Development Tools (ADT) which was the official integrated developing environment for Android applications. However since then, Android has moved towards the Android Studio platform. With structural differences between the two IDEs and the gaining popularity of Android studio it has become prudent to move to Android studio for which there is better support.

- Configuration of eclipse project for Android Studio - Android Studio uses the quick growing Gradle build system which Eclipse does not. As such the Eclipse project was exported to Android studio and configured to use Gradle build.
- Use of Mobile device and creation of Android Virtual Devices (AVD) – This is for running and debugging of application

4.2.2 Version management. For the duration of the project the changes made to the application code would be tracked. This is important as it provides back-up points to revert to in the case a current version of the application code is not working. The past and present contributors to the application code have used GIT as the version control system. As such for the duration of the project GIT would be used.

4.3 Implementation approach

4.3.1 Database implementation. The DataClass in the application code implemented before my contribution to the project, serves as a database handler class which extends the default Android SQLiteOpenHelper class. The DataClass provides an abstract layer between the controller classes and the SQLite database. For creating the necessary tables in the existing database, a controller class for each entity to be added to the database was created except the OPDcaseRecord and OPDcases classes which already existed hence were simply edited to include necessary changes. These classes namely Bundles, ClaimRecords, InvestigationRecords, Investigations, MedicineRecords, Medicines, Outcomes, ProcedureRecords, Procedures, TypeOfAttendances, Outcome and TypeOfServices each have getInsertSQLString() methods that return an SQL create statement containing instructions to create new tables with the appropriate columns. The OPDcaseRecord and OPDcases classes have these methods already implemented therefore they

were altered to include new columns necessary for the NHIA module's operation. Figure 4.2 below shows edit that was made to the `getCreateSQLString()` method of the `OPDCases` class.

```
public static String getCreateSQLString(){
    return "create table " + TABLE_NAME_CLAIM_RECORDS + " ("
        + CLAIM_ID + " integer primary key, "
        + TYPE_OF_SERVICE + " text, "
        + OUTCOMES + " text, "
        + FIRST_VISIT + " text default '1900-01-01', "
        + SECOND_VISIT + " text default '1900-01-01', "
        + THIRD_VISIT + " text default '1900-01-01', "
        + FOURTH_VISIT + " text default '1900-01-01', "
        + CLAIM_CHARGE + " real default 0, "
        + STATUS + " text default 'open', "
        + BUNDLE_ID + " integer default 0"
        + ")";
}
```

Figure 4.1: Code snippet of `getCreateSQLString` method in `ClaimRecords`

```
public static String getCreateSQLString(){
    return "create table " + TABLE_NAME_OPD_CASES + " ("
        + OPD_CASE_ID + " integer primary key, "
        + OPD_CASE_NAME + " text, "
        + OPD_CASE_CATEGORY + " integer, "
        + OPD_CASE_DISPLAY_ORDER + " integer default 0, "
        + OPD_CASE_GDRG_CODE + " text, "
        + OPD_CASE_CHARGE + " real default 0"
        + ")";
}
```

Figure 4.2: Code snippet of `getCreateSQLString()` method in `OPDCaseRecord`

As a database handler class, the `DataClass` includes the `onCreate()` and `onUpgrade()` methods which are responsible for creating and populating tables on initial database creation and database upgrade respectively. As such, to include the new tables to the existing database, the return value `getCreateSQLString()` methods of the controller classes are passed as arguments to the `execSQL()` method of the `SQLiteDatabase` object 'db'.

```

//added bundles, claim records, Investigations, procedures, medicines, medicines record,
//investigation records,procedures records tables
db.execSQL(Bundles.getCreateSQLString());
db.execSQL(ClaimRecords.getCreateSQLString());
db.execSQL(Medicines.getCreateSQLString());
db.execSQL(Procedures.getCreateSQLString());
db.execSQL(Investigations.getCreateSQLString());
db.execSQL(MedicineRecords.getCreateSQLString());
db.execSQL(ProcedureRecords.getCreateSQLString());
db.execSQL(InvestigationRecords.getCreateSQLString());
db.execSQL(TypeOfServices.getCreateSQLString());
db.execSQL(TypeOfAttendances.getCreateSQLString());
db.execSQL(Outcomes.getCreateSQLString());

```

Figure 4.3: getCreateSQLString methods in onCreate() method of DataClass

The DataClass has a declared constant for the database version of the application. This integer value is incremented based on database schema changes. The onUpgrade() method is called when the DataClass is invoked with a greater database version number from the one previously used. The purpose of this is to prevent the data stored in the database of the older version from being lost. As this will be the case for the CHOs who will have to update the application to make use of the NHIS claim functionalities, there will be a need to include alter statements for changing the schema of the database. To facilitate upgrading the database rather than creating a new one, two methods namely upgradeVersion15() and upgradeVersion16() were implemented and called in the onUpgrade() method of the DataClass.

The upgradeVersion15() method similarly has getCreateSQLString() methods for the controller classes earlier mentioned. However the getCreateSQLString() methods called here are only for creating tables that don't already exist in the database in order to avoid duplicates. For the ones that do exist, that is the community_member_opd_cases table and opd_case table, SQL alter statements that add the new columns are passed to the execSQL() method of the SQLiteDatabase object. The SQL alter statements contain instructions that add the 'claim_id' column to the

community_members_opd_cases table and the 'gdrg_code' and 'charge' columns to the opd_case table.

```
private void upgradeToVersion15(SQLiteDatabase db) {

    db.execSQL(Bundles.getCreateSQLString());
    db.execSQL(ClaimRecords.getCreateSQLString());
    db.execSQL(Medicines.getCreateSQLString());
    db.execSQL(Procedures.getCreateSQLString());
    db.execSQL(Investigations.getCreateSQLString());
    db.execSQL(MedicineRecords.getCreateSQLString());
    db.execSQL(ProcedureRecords.getCreateSQLString());
    db.execSQL(InvestigationRecords.getCreateSQLString());
    db.execSQL(TypeOfServices.getCreateSQLString());
    db.execSQL(TypeOfAttendances.getCreateSQLString());
    db.execSQL(Outcomes.getCreateSQLString());

    db.execSQL("alter table " + OPDCaseRecords.TABLE_NAME_COMMUNITY_MEMBER_OPD_CASES +
        " add column " + ClaimRecords.CLAIM_ID + " integer default 0");
    db.execSQL("alter table " + OPDCases.TABLE_NAME_OPD_CASES +
        " add column " + OPDCases.OPD_CASE_GDRG_CODE + " text");
    db.execSQL("alter table " + OPDCases.TABLE_NAME_OPD_CASES +
        " add column " + OPDCases.OPD_CASE_CHARGE + " real default 0");
}
```

Figure 4.4: upgradeVersion15() method in DataClass

4.3.2 Implementation of model and controller classes. In following the existing mvc architecture of the application code, a third group of classes (models) was created to hold database entry data. These classes are namely Medicine, Investigation, Procedure, MedicineRecord, ProcedureRecord, InvestigationRecord, ClaimRecord, TypeOfAttendance, Outcome and TypeOfService. Each have constructors and getters for instance variables that bear one-to-one correspondence with attributes of the respective tables they hold row data for. The controller classes are as many as the model classes in order to maintain the one-to-one relationship that the preceding contributors to the Yaresa application maintained in the application code. As opposed to having one controller class having multiple corresponding models.

```

public class Outcome {

    private int id;
    private String outcome;

    public Outcome(int id, String outcome) {
        this.id = id;
        this.outcome = outcome;
    }

    public int getId() { return id; }

    public String getOutcome() { return outcome; }

    @Override
    public String toString() {
        return outcome;
    }

}

```

Figure 4.5: Outcome class

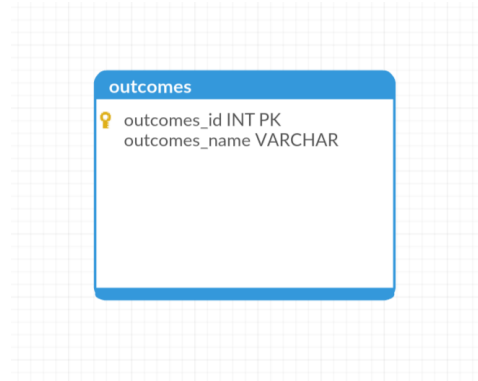


Figure 4.6: Outcome table

Figure 4.5 shows the outcome class which has instance variables `id` and `outcome` both of which respectively correspond to the `outcomes_id` and `outcomes_name` columns of the outcome table shown in figure 4.6

4.3.3 Core functionalities

4.3.3.1 Adding of claims. With the most of the activities of the module dependent on the use of insurance claim data, the creation of claims becomes the most important feature of the module. The implementation of the 'add a claim' feature follows the activity diagram figure shown in chapter 2. In view of this, the first step involves the user inputting the necessary claim information in a form. To do this, the user must navigate to the `NhisClaimFormActivity` by clicking the record button of the OPD fragment of the `CommunityMemberRecordActivity` which previously recorded community member OPD cases only. The `NhisClaimFormActivity` upon being launched sets the content view with its corresponding xml layout that is the `activity_nhis_claim_form_activity.xml` which is shown in the figures below.

Filled list view

Figure 4.7: NHISclaimFormActivity Figure 4.8: NHISclaimFormActivity with filled list view

As can be seen in Figure 4.8 a list view exists between the Diagnosis and Procedure fields. This list view is updated anytime the add button next to the diagnosis spinner component is clicked. This is the case for the procedure, investigation and medicine fields. For each of these fields, as the user adds items to the list views of each field, the total charge of items in the list is computed.

```
private void addDiagnosisRecord() {
    OPDCase p = new OPDCase();
    p = (OPDCase) spinnerOPDCases.getSelectedItem();
    totalDiagnosisCharge += p.getCharge();
    OPDCaseRecord prec = new OPDCaseRecord(communityMemberId, p.getID(),
        p.getOPDCaseName(), getLastVisitDate(), choId, p.getCharge());
    diagnosisStringList.add(prec);
    diagnosisAdapter.notifyDataSetChanged();
    ViewGroup.LayoutParams params = diagnosisListView.getLayoutParams();
    params.height = getItemHeightOfListView(diagnosisListView);
    diagnosisListView.setLayoutParams(params);
    diagnosisListView.requestLayout();
}
```

Figure 4.9: addDiagnosisRecord() in NHISclaimFormActivity

Once all the fields on the form have been completely filled, the add claim button at the bottom is clicked and the total charge of the claim is computed by summing the total charges of diagnosis, investigations, medicines and procedures added to the claim.

```
public float computeTotalClaimCharge() {
    return totalDiagnosisCharge + totalInvestigationCharge + totalMedicinesCharge + totalProcedureCharge;
}
```

Figure 4.10: computeTotalClaimCharge() method that computes total charge of claim.

According to the activity diagram the second step involves determining if this claim is being added to a new bundle or an existing bundle. As a result of this, clicking the ‘add claim’ button creates an AlertDialog.Builder object displaying the two options shown in Figure 4.11.

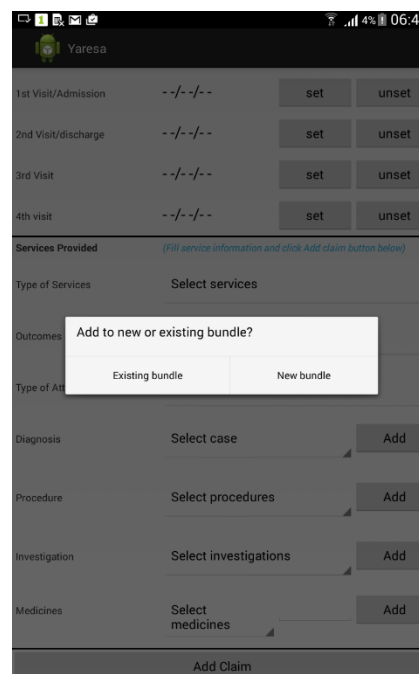


Figure 4.11: AlertDialog with new or existing bundle.

Selecting the New bundle option creates a Bundles class object and calls the addOrUpdate() method on it. This adds a new record to the bundle table in the database, returns the id of this

record to the NhisClaimFormActivity class, and stores the value in the justAddedBundleId global variable to be used by the addClaimRecord(). This is necessary because a claim object requires the bundle id for which it is associated with. The ‘Existing bundle’ option on the other hand enables the user to select from a list of bundles retrieved from the database. In this case, the id of the selected bundle is what is used by the addClaimRecord() function. The addClaimRecord() mentioned is the function responsible for retrieving the inputs from the input fields on the form and passing them as arguments to the addOrUpdate() method of a ClaimRecords object. The addOrUpdate() method adds the data passed to it into their respective columns in the claim_records table in the database.

```
public void addClaimRecord(){
    ClaimRecords claim = new ClaimRecords((this.getApplicationContext()));
    claim.addOrUpdate(spinnerTypeOfServices.getSelectedItem().toString(), spinnerOutcomes.getSelectedItem().toString(),
        visitOne.getText()+"",visitTwo.getText()+"",visitThree.getText()+"",visitThree.getText()+"",computeTotalClaimCharge(),
        justAddedBundleId);
    justAddedClaimId =claim.getClaimRecordId();
    addProcedures();
    addInvestigations();
    addMedicines();
    addDiagnosis();
}
```

Figure 4.11: addClaimRecord() method of NhisClaimFormActivity.

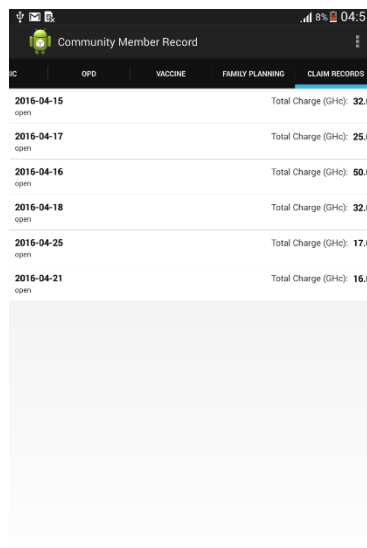
```
public boolean addOrUpdate(String typeOfService,String outcomes,String firstVisit, String secondVisit,String thirdVisit,
    String fourthVisit, float charge, int bundleId){
    try{
        SQLiteDatabase db=getWritableDatabase();
        ContentValues cv=new ContentValues();

        cv.put(TYPE_OF_SERVICE, typeOfService);
        cv.put(OUTCOMES, outcomes);
        cv.put(FIRST_VISIT, firstVisit);
        cv.put(SECOND_VISIT, secondVisit);
        cv.put(THIRD_VISIT, thirdVisit);
        cv.put(FOURTH_VISIT, fourthVisit);
        cv.put(CLAIM_CHARGE, charge);
        cv.put(BUNDLE_ID, bundleId);
        idAfterInsertion = (int) db.insertWithOnConflict(TABLE_NAME_CLAIM_RECORDS, null,cv,SQLiteDatabase.CONFLICT_REPLACE);
        if(idAfterInsertion <=0){
            return false;
        }
        close();
        return true;
    }catch(Exception ex){
        close();
        return false;
    }
}
```

Figure 4.12: addOrUpdate() method of the ClaimRecords class.

The `addProcedures()`, `addInvestigations()`, `addMedicines()` and `addDiagnosis()` in the `addClaimRecord()` shown in Figure 4.11 are each responsible for respectively adding the values in the listviews into the `procedure_records`, `investigation_records`, `medicine_records` and `opd_records` tables.

4.3.3.2 Viewing of recorded claims. Claims added in the application can be viewed by user. To view the claims of each client or Community Member as it is named in the application, the user first selects a community member which opens the `CommunityRecordActivity`. Here the user navigates to the tab labelled `Claim Records` which displays the `ClaimFragment`. The `ClaimFragment` has a list view to display the list of recorded claims for the selected client. The list view uses a custom adapter which defines the layout of data in each list item. This layout has three text views that display the first visiting date of the claim, its total charge and status i.e. either closed or open. Figure 4.13 shows how the claims are shown in the list view.



Date	Status	Total Charge (GHS)
2016-04-15	open	32.0
2016-04-17	open	25.0
2016-04-16	open	50.0
2016-04-18	open	32.0
2016-04-25	open	17.0
2016-04-21	open	16.0

Figure 4.13: Populated listview in ClaimFragment .

4.3.3.3 Updating of Recorded Claims. As discussed in the requirements analysis, only open claims can be updated. To do this, the user taps on an open claim list item in the list view which starts the NhisClaimsFormActivity. An integer value of 1 stored in the isEditable variable along with claim's id value is passed as an argument to the activity. In the onCreate() method of the NhisClaimsFormActivity the value of the isEditable variable is checked to determine if it is equal to 1. Once it is confirmed to be true, the loadContents() method is called which retrieves the necessary data for the selected claim and maps them to their corresponding fields in the NhisClaimsFormActivity. The user may edit any of these input fields and tap the save claim button to add the updated claim.

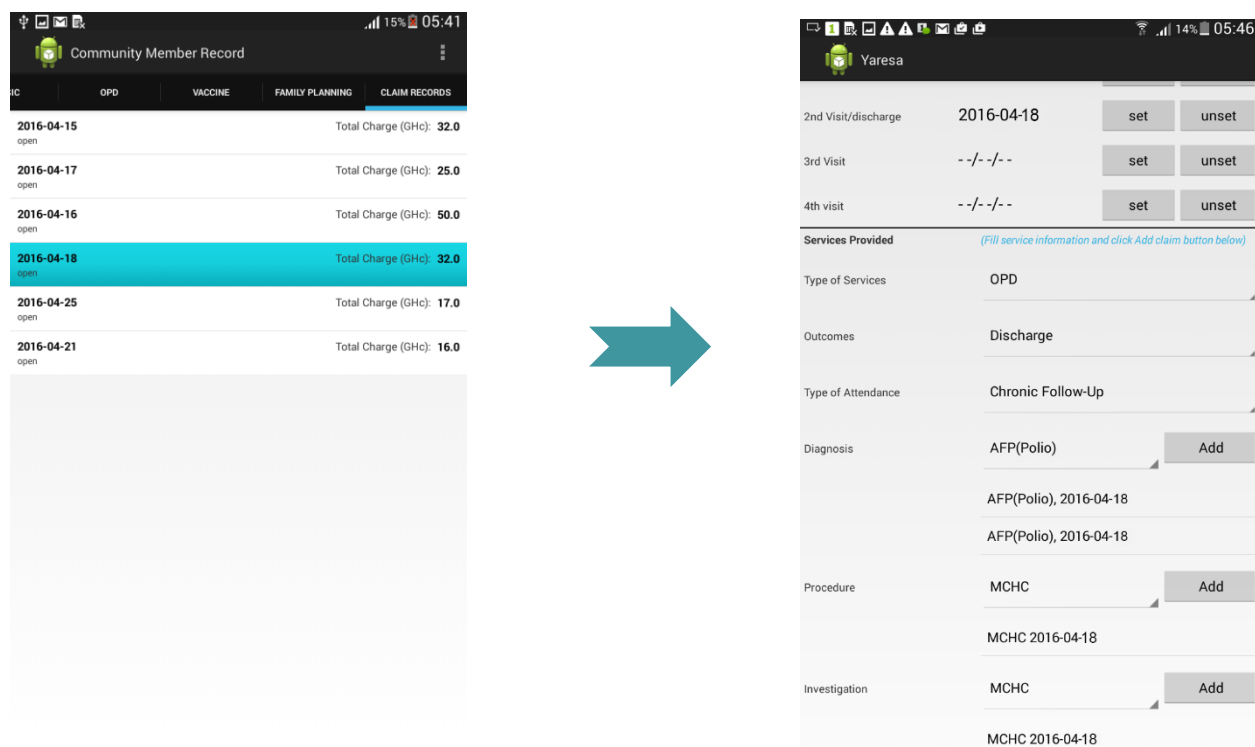


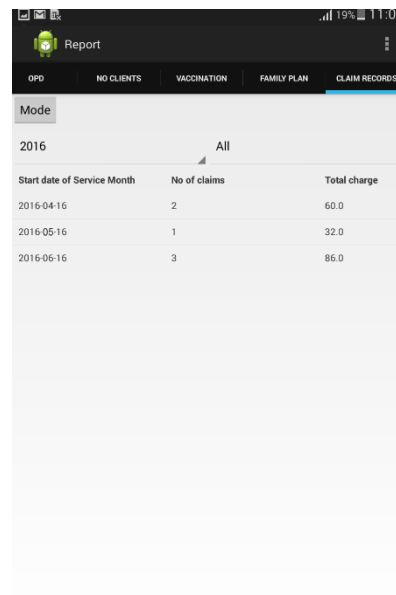
Figure 4.14: Updating of claims.

4.3.3.3 Viewing of claim report. All reports generated for the various modules in the application are displayed in the ReportActivity. As such, to view the claims report, a ClaimsReport fragment

was implemented in the ReportActivity class. The fragment contains a listview which also uses a custom adapter to define the layout for each list item. The data displayed in each list item is a summary of claims of service month's claims (bundle). This includes the start date, number of claims and the sum of claim charges within that service month. For this report, an SQL view was created which performs a left join of the bundles and claim_records tables. The columns of the created SQL view are namely bid, bdate, num_of_claims, and total_claim. The number of claims and total sum of claim charges for each bundle were computed using the count() and sum() functions in SQLite. The Figure 4.15 shows the query for creating this view.

```
CREATE VIEW view_bundle_details AS SELECT bundles.bundle_id AS bid ,bundle_date AS bdate ,
COUNT ( * ) AS num_of_claims, SUM ( claim_records.charge ) AS total_charge_claims FROM bundles
LEFT JOIN claim_records ON bundles.bundle_id = claim_records.bundle_id GROUP BY
bundles.bundle_id
```

Figure 4.15: Code snippet of SQL query the view_bundle_details SQL view.



Start date of Service Month	No of claims	Total charge
2016-04-16	2	60.0
2016-05-16	1	32.0
2016-06-16	3	86.0

Figure 4.16: Claim records tab in ReportActivity.

Chapter 5: Testing and Results

During the software verification phase, various types of testing were conducted:

- Unit Testing: to verify that each component works very well separately
- Integration Testing: to verify that components works well when integrated
- User testing

5.1 Unit Testing

The unit tests were conducted during the course of developing the application. It focusses on testing database queries and individual functions of the classes created for the project. These tests are critical for ensuring their correct operation once combined with other parts of the application. The tests performed were manual meaning they required human intervention without the use of automation tools. For the purpose of testing queries made to the database, the SQLite browser which is an open source tool for database creation and design was used to view changes made to the database.

5.1.1 Table creation and alteration. Each `getCreateString()` method of the various controller classes responsible for creating their respective tables in the database were tested for errors. Undesired results were corrected and re-tested. The SQL alter statements executed in the `upgradeVersion()` methods of the `DataClass` were also tested to determine if they alter existing tables as expected. For these tests the `onUpgrade()` method must be called, as such the application was repeatedly launched on a mobile device with a pre-installed lower version of the application.

5.1.2 Data entry and Retrieval. These involve testing methods of the controller classes responsible for populating and retrieving data from the created tables. For the data entry tests the dummy data used were viewed in the SQLite browser to ensure that they were stored as expected

in the appropriate tables and columns. The methods to retrieve this dummy data were also tested to determine if the right data was stored in the objects of their respective classes.

5.2 Integration tests

For the integration tests, the individual components tested in the unit testing phase were integrated to determine how the system processes meet the functional requirements of the module. Test cases were simulated via appropriate parameter and data inputs to determine whether all integrated components interface correctly with each other.

5.3 User Testing

User tests were conducted to determine the general usability of the system. Some students of Ashesi University were used as representative users of the application for the tests. During these tests, qualitative data was collected observing users perform typical tasks on the application. The aim collecting this data was to determine changes required to improve the user friendliness and performance of the module.

5.4 Results

The primary focus of these tests was to determine which parts of the application code had errors that needed correction. Unsuccessful tests during unit testing included syntax errors in queries made to the database. Integration tests identified components that were not working properly. This was particularly relevant during the adding of claims because it involves multiple components to make a successful database entry. Errors from both integration and unit tests were corrected and repeatedly retested. Ultimately the various the integration and unit tests performed were passed.

Feedback from users during the user testing phase acknowledged the use of user interface elements that allow the user to select from a list of options rather than typing in a text field. It however identified an inconvenience in setting the visiting dates on the NHIS claim form. Users struggled

understanding that to set a visiting date the date picker widget at the top of the form had to be set to the appropriate date before pressing the set button. As a result of this, users made the common mistake of pressing the set button and realising after that the appropriate date on the date picker was not set.

1. Select date on date picker

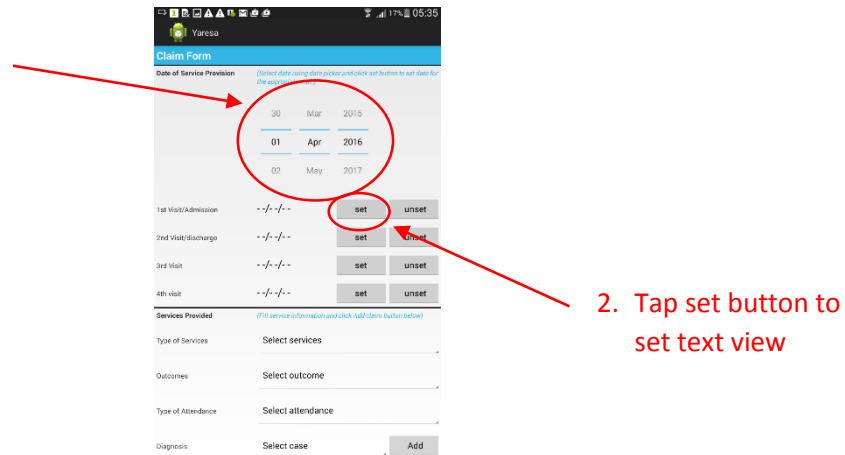


Figure 5.1: NHIS add claim form with fixed datepicker.

To improve the setting of the visiting dates, the earlier design as shown in the Figure 5.1 was replaced with a date picker dialog which pops up when a set button is pressed. This way the date picker is presented to the user in each instant the set button is pressed. The Figure 5.2 shows the implementation of this design.

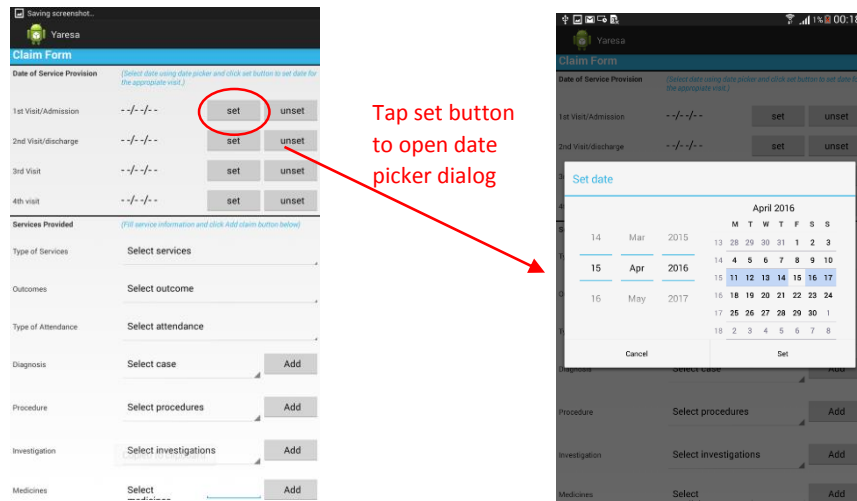


Figure 5.2: User interface design with popup datepicker.

A second set of user tests were conducted to assess the effectiveness of this design. The result from the tests proved this design to be more efficient than the first.

The design was further improved with the use of a listview to display the dates. With this design, the user has a single button that updates the listview with visiting dates. This was implemented because it was observed in most user tests that with all the input fields for the visiting dates displayed on the form, users had the impression that they had to fill each one for a claim to be successfully created.

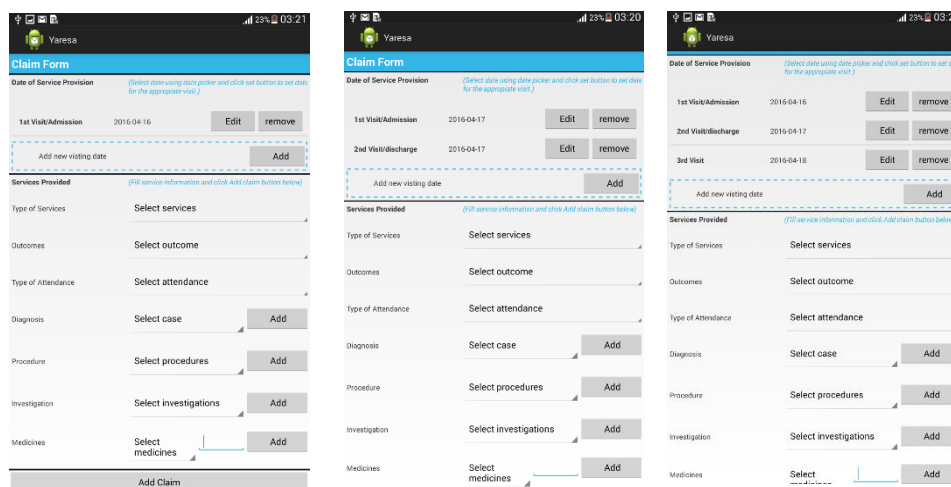


Figure 5.3: User interface design with list view that updates when add date button is clicked.

Chapter 6: Conclusion

6.2 Project Review

The main objective of the project is to develop a system that reduces the time spent by health officers on compiling National Health Insurance (NHIS) claims. To achieve this, an NHIS module was developed primarily to further work undergone by previous contributors of the Yaresa mobile health application. The system built provides a strong foundation for further work. Its core functionality facilitates the recording and compiling of NHIS insurance claims using the application. The recording of claims functionality enables the user to add and manage claims on the application. The compiling of claims functionality automatically computes claim charges for which reports can be generated. These met requirements collectively reduce the time spent by health officers in preparing of insurance claims ultimately leaving them with more time to dedicate to meeting health care needs of community members.

6.2 Challenges

During the course of development these are some of the challenges that were faced.

6.2.1 Configuring of Eclipse project in Android Studio. The initial import of the Eclipse project to Android studio was not successful. As such a considerable amount was spent configuring the project to be buildable in Android Studio. The

6.2.2 Limitations with working on existing project. Working on an existing project was challenging and limiting because decisions taken during design and implementation had to conform to standards set by previous contributors of the project. The database design was an aspect of the project that proved particularly difficult in this respect as the design developed had to ensure that the existing tables were maintained.

6.3 Further work

The recording, compiling and generation of reports for NHIS claims as stated in the project review section of this chapter have been successfully implemented. However as identified in the requirements specification, for the claims to be submitted electronically the application must facilitate the saving of the claim data in the NHIA standard XML format. This requirement has not been completely implemented and as such provides a basis for further work. The xml file generation currently writes to file saved in an xml format, however the use of xml libraries would be more efficient.


References

- Ankomah, D. (2014). (Unpublished undergraduate dissertation). Ashesi University College, Berekuso.
- Central Intelligence Agency. (2015, October 20). *The World Factbook*. Retrieved from Central Intelligence Agency: <https://www.cia.gov/library/publications/the-world-factbook/geos/gh.html>
- Grabowski, G. (2015). *Standardized e-claims interface for health providers' Hospital Information Systems (HIS), XML Methodology (ver. 8.0)*. Gdansk: Nearshoring Solutions.
- Russell, S. (2008). Community-based Health and Planning Services: Decentralizing Ghana's Health System. *Georgetown University Journal of Health Sciences*, Vol. 5, No.1.
- Sommerville, I. (2011). *Software Engineering*. Boston, United States of America: Addison-Wesley.

APPENDIX

Appendix A

Sample NHIS claim form

NATIONAL HEALTH INSURANCE SCHEME							
 Claim Form HI Code: _____							
Client Information							
Claim number:	334567	Claim Date	2010-10-05				
Scheme Name	National health insurance schemes	NHIS no.	4325678				
Hospital Record Number:	120901	Sex	Male				
First Name:	Adwoa	Surname	Annan				
Date of Birth:	1972-02-15	Age	30				
Service Provided							
Type of services:	OutPatient - All inclusive	Dates of Services	1st visit/Admission 2010-10-05 2nd visit/Discharge 3rd visit 4th visit				
Outcome:	Discharged	Duration of Visit:					
Type of Attendance							
Chronic Follow-up		Special Codes:					
		Special Description:					
Procedure(s)							
	Description	Date	Code				
Procedure 1	Surgery for lung and bronchus lesions T_12 VR 8	2010-09-29	A SUR02A				
Diagnosis(es)							
	Description	Date	Code				
Diagnosis 1	Stenosis	---	---				
Investigation(s)							
	Description	Date	Code				
Investigation 1	WCHC	2010-09-29	---				
Medicine(s)							
	Description	Price	Qty	Total Cost	Date	Code	
1	6-Fluorouracil Injection, 80 mg/ml	8.00	3	24	2010-09-29	46468	
2	Allopurinol Tablet 300 mg	0.20	3	0.6	2010-09-29	ALLOPUA2	
Client Claim Summary							
	Type of Service	ICD-9 Code	Total Amount (USD)				
A	In-Patient	---	-----				
B	Out-Patient	---	534.9				
C	Investigations	---	0				
D	Pharmacy	---	24.9				
TOTAL			535.5				
Signature: _____ Name: _____							
Scheme Use Only							
Date Received:	_____	Action 1:	_____	Date:	_____	Signed:	_____
Signed:	_____	Action 2:	_____	Date:	_____	Signed:	_____