An Adaptable iOS Mobile Application

for Mobile Data Collection

by

Zahra Al-Kaf

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved April 2016 by the
Graduate Supervisory Committee:

Tim Lindquist, Chair
Srividya Bansal
Ajay Bansal

ARIZONA STATE UNIVERSITY

May 2016

ABSTRACT

Mobile data collection (MDC) applications have been growing in the last decade especially in the field of education and research. Although many MDC applications are available, almost all of them are tailor-made for a very specific task in a very specific field (i.e. health, traffic, weather forecasts …etc.). Since the main users of these apps are researchers, physicians or generally data collectors, it can be extremely challenging for them to make adjustments or modifications to these applications given that they have limited or no technical background in coding. Another common issue with MDC applications is that its functionalities are limited only to data collection and storing. Other functionalities such as data visualizations, data sharing, data synchronization and/or data updating are rarely found in MDC apps.

This thesis tries to solve the problems mentioned above by adding the following two enhancements: (a) the ability for data collectors to customize their own applications based on the project they're working on, (b) and introducing new tools that would help manage the collected data. This will be achieved by creating a Java standalone application where data collectors can use to design their own mobile apps in a user-friendly Graphical User Interface (GUI). Once the app has been completely designed using the Java tool, a new iOS mobile application would be automatically generated based on the user's input. By using this tool, researchers now are able to create mobile applications that are completely tailored to their needs, in addition to enjoying new features such as visualize and analyze data, synchronize data to the remote database, share data with other data collectors and update existing data.

Keywords: mobile data collection, data visualization, data synchronization, data share

DEDICATION

To my mother, father and beloved husband.

ACKNOWLEDGMENTS

This thesis wouldn't have been possible without the input and supervision of Dr. Tim Lindquist. I would also like to show my gratitude to Dr. Heather Bateman, Dr. Srividya Bansal and Dr. Ajay Bansal for their support.

TABLE OF CONTENTS

Page

LIST OF TABLES……vi

LIST OF FIGURES……..vii

CHAPTER

1 INTRODUCTION…….……..1

Overview……..……1

Thesis/Problem Statement……...2

What is Mobile Data Collection? ……...2

Related Work…...……....4

2 MODEL…….……11

Technology Model…….……11

System Design…….……..11

System Purpose, Scope and Overview……...…11

Java Tool Architecture…….....….12

iOS Mobile App Architecture…….……...12

Java Tool Design (Form Design Phase)…….……...…13

iOS Mobile App Design (Mobile Data Collection Phase)…….……15

Database Design…….....16

Activity Diagram……...17

CHAPTER Page

3 IMPLEMENTATION……..……...…18

Case Study…….……18

System Functions…….……..20

Java Application Components……...20

Mobile Application Components…….....21

Performance……..…….……...……..23

Java tool……..…….……..23

iOS Mobile Application. …….…….....23

Constraints…….…….…...23

Testing…….…..…24

Java App…….…….……...……24

Mobile App……..……..…….24

4 RESULTS……..……...……..25

Prototype Assessment & Evaluation……..……...…..……25

Challenges…….…….....26

Strengths…….……...…….…..27

Weaknesses.…….…… ……...……..…28

5 CONCLUSION AND FUTURE WORK..…….……...……..…...29

WORKS CITED……..…….…….…...……31

APPENDIX

A TABLES AND FIGURES...……..…...…….…...33

LIST OF TABLES

Table Page

1. Promised Functionalities and Features……..….34
2. Technology Model…….……....……35
3. Java Application User Interface Components for Creating a New Application and their Purposes…….……36
4. Java Application User Interface Components for Updating an Existing Application and their Purposes…….…….37
5. Common Viruses Database Example……..…….……..38
6. Performance Metrics…….….……39
7. System Assessment and Evaluation……..……40

LIST OF FIGURES

Figure Page

1. Java Application Architecture……..……..41
2. iOS Application Architecture ……..……42
3. Phases of Using the System & System Input/output…….……43
4. Java Application Local Database Example…….……..…..44
5. The Process of Selecting a Database Name…….……..45
6. Java Application Activity Diagram…….……..46
7. iOS Application Activity Diagram…….……..…..47
8. Clinics and Common Viruses Case Study…….……48
9. Java Application Screenshots: Create a New Application…….……49
10. Java Application Screenshots: Design Form and Summary…….....……50
11. Java Application Screenshots: Verify Application Using a Key……..…….51
12. iOS Application Screenshots: New Entry Tab View…….52
13. iOS Application Screenshots: Pie Charts……...……53
14. iOS Application Screenshots: Update Tab View……...…54
15. iOS Application Screenshots: Upload Data (Data Synchronization)……..…….55
16. iOS Application Screenshots: Share Data Tab View…….56
17. Java Application Class Diagram……....57
18. iOS Application Class Diagram…….……58

Introduction

Overview

Regardless of the field of study, quantitative or qualitative data, precise and accurate data collection is fundamental in maintaining the research's integrity. Inaccuracy in collecting data can lead to flawed results, and the inability to answer the research questions. Recently, mobile devices were introduced to the data collection process attempting to increase accuracy and reduce error rate. Not only that, but Mobile Data Collection (MDC) has also increased the productivity since it allowed field workers to collect data on the fly and have access to data at all times. It also eliminated redundancy. With the use of mobiles to collect data, data has to be entered only once.

On the other hand, designing mobile applications for data collection will most likely require hiring a dedicated developer, which will cost a lot of time and a lot of money. Even after the application is designed, making any changes to that app is another challenge. If researchers had the ability to design and generate their own mobile applications without the need for any technical or programming skills, this will increase the usage of mobile data collection applications and will make it feasible to everyone. Data collection is a great example for creating an adaptable/customizable mobile app because it has the right amount of standardization, yet it can be customized to the user's needs. The form can be customized according to the research conducted, and the app in general can have all the standard features needed in a mobile data collection application. Entering and storing data, visualizing and analyzing existing data, updating data, and sharing data with other users are some of the standard functionalities needed in any MDC

application. In the next sections, the aim of this thesis will be explained as well as the problem statement.

Problem Statement

The motivation behind this thesis was to create adaptable mobile applications that are not only used just for data collection, but for data analysis and data communication as well. This research aims to achieve that by creating a Java tool in which researchers and data collectors can use to customize their mobile apps. This tool is the only interface users need to interact with in order to design their mobile applications. Once the app has been designed, automatically, an iOS application will be generated. The mobile application will not only serve the purpose of mobile data collection, it will also have functionalities to visualize data, synchronize local data to the centralized database, and share data among different users by allowing the mobile devices to communicate with each other's. Table 1 lists all the features and functionalities this prototype promises to deliver.

What is Mobile Data Collection?

Data collection is the process of gathering information using one or several standardized methods. The collected data enables researchers to answer a particular question, validate a hypothesis, or just simply analyze the results. Data collection isn't restricted to a specific field or area. Data can be collected in almost any field whether it's for the purpose of research or feedback.

Mobile Data Collection (MDC) is using a mobile device such as a smartphone or a tablet to collect and gather any type of information (Wogan, 2014). Data collecting and analysis is an old concept that started years and years ago. Originally, people used to

collect data using simple tools such as pen and paper, which was tedious and inefficient. Then localized databases came along. If data need to be collected on-site, then local databases are not the best option, since data will be collected from one place, then re-entered into the local system again.

Here's when the introduction of mobile devices to this process has revolutionized the way data is being collected. With the use of mobile devices, data can be collected and recorded directly on-site eliminating the need of data re-entry. Not only that, but also it made it much more efficient and accurate. Introducing lists and mandatory fields can reduce the error rate drastically. Also, specifying the format in which an entry should look like makes the data more standard and easier to analyze. Several more features can be introduced to the mobile app to control and unify the data.

Related Work

Unsurprisingly, lots of tools and systems were created that serve a similar goal of this research. One tool that stands out in this area is Open Data Kit (ODK) (Anokwa, 2010). ODK is an open tool that helps organizations manage mobile data collection solutions. It's used to build forms, collect data on a mobile device, and aggregate the collected data on a server and extract it in useful formats. ODK was developed by researchers at the University of Washington's Department of Computer Science and Engineering and active members of Change, a multi-disciplinary group at UW exploring how technology can improve the lives of under-served populations around the world. Before using ODK, both ODK-collect[1] and ODK-aggregate[2] need to be installed. After that, users need to design the form (more specifically, XLSForms[3]), set up a server, and connect a mobile device to that server.

One disadvantage of using ODK Aggregate is that data has to be passed through Google App Engine which is not recommended for sensitive information (cartONG, 2012). Additionally, deploying ODK Aggregate 1.0 requires a dedicated IT staff to install and manage properly.

Another famous tool for generation mobile data collection services is called OpenXData. OpenXdata is an open-source software package that was created by 10 researchers from Addis Ababa University, Cell-Life, Handheld Solutions & Research

[1] ODK Collect takes the forms and transforms them into a list of input prompts (Sundt, 2015).

[2] ODK Aggregate is a tool that provides a server and data repositories in order to export, visualize and publish the collected data (Sundt, 2015).

[3] XLSForm is a standard user to help users create their own surveys and forms in Excel (Sylvia, 2014).

Labs, Interactive Research and Development, Sustainable Sciences Institute, University of Bergen, Makerere University, and the University of Southern Maine (openXdata, 2010). OpenXData is a data collection web app where users can create their own forms, collect data and review results. Data can be recorded and reviewed using a mobile device. Then, an XML file will be generated with the collected data. OpenXData is written in Java, and can be installed on any Java-enabled phone (e.g. Nokia 1680).

One shortcoming of OpenXData is that it has no data analysis or visualization feature, which requires the use of another external data analysis tool. Also, since it's Java enabled software, it is only supported on Android devices and not iOS devices.

Last Mile Mobile Solutions (LMMS) is another popular tool for mobile data collection. Jay Narhan is the team leader at LMMS along with Benny Law, Richard Lankas, Keith Chibafa and Paul Mwirichia (Narhan, 2013). LMMS enables the use of mobile applications to effectively manage responses to disasters. This system collects data about affected population and then automates how aid agencies will provide and deliver services. Using LMMS, data can be captured both online and offline. Here, LMMS definitely serves the use of data collection. However, it's served in only one particular field, which is delivering help and services to those in need. It also requires the use of a special hardware kit (handheld device with a barcode reader and camera).

In this area, several papers have been published. One interesting dissertation written by Suganya Baskaran at Iowa State University in 2012 titled "A rapid deployment model for VGI projects in mobile field data collection" (Baskaran, 2012). This paper was about spatial data collection[4]. It focused on collecting and evaluating Voluntary

[4] *Spatial Data* is data that describes the location and/or shape of an object on the earth

Geographic Information (VGI). VGI is the geographic data voluntarily provided by individuals. Examples of VGI can be children's walkability to schools, or users' willingness to share their locations on Twitter to be used when creating spatial queries. Another example of VGI is during an emergency situation such as flooding, attributes like amount of flooding, number of houses damaged and number of people affected are collected and evaluated (Blanford, 2014). This research paper created a model for a mobile VGI system. They created an iOS mobile application, which they were using to collect VGI and spatial data and feed it to a Geographic Information System (GIS). This paper covered a number of essential points; creating GIS services, consuming the mapping services in a client mobile app, and presenting a dynamic decision tree to collect non-spatial attributes. However, there were still some points this paper didn't focus on. An important addition to this system can be tracking users, involving route planning, and enabling the system to have offline support in case there was no network coverage.

Another effort is reported in a paper written by Deo Shao at Malmö University in 2012 titled "A Proposal of a Mobile Health Data Collection and Reporting System for the Developing World" (Shao, 2012). This study proposed a prototype using ODK to test its feasibility in improving health data collection. The prototype offered ways of collecting health data through mobile phones and visualizes the collected data in a web application. It attempted to bridge the information gap between primary health care facilities and secondary facilities to allow timing and consistency in reporting routine health data. It majorly focused on collecting statistical data coming from both primary and secondary health care facilities. The general aim of this paper was to improve the overall health service by improving the decision making process. Improving the data collection quality

can help doing that. The author of the paper was motivated to create such system because he noticed that the main communication system between health care facilities was unsustainable. Also, he noticed that the facilities were reporting health data manually, which can increase the error rate and inconsistency. That's why he proposed a system where mobile devices and open source system (ODK in this case) can be used to improve the data communication and reporting process.

One shortcoming of this paper was that the proposed prototype didn't support any kind of data visualization. This means, researchers and decision makers will either have to navigate through the data manually or use an external tool for visualizing it. Another gap is the lack of any data analysis feature. If such feature were introduced, that would automate the decision making process and save health practitioners a lot of time.

Another interesting paper was written by Poonam Songar in 2012 at Kent State University was titled "Learning Assessment Data Collection From Educational Game Applications" (Songar, 2012). The main goal of this paper was to propose an open source mechanism for collecting assessment data from an educational game. Then, the collected data was stored in a spate media for analysis. The outcome of this thesis wasn't a single tool, but a whole packaged solution. Here is the list of the package components:

- An educational game app with clear set of goals. The source of the game is also needs to be specified (i.e. the apple store, website, etc.)
- Assessment Data: The instructor will need to set how the students/players will be assessed (e.g. set of challenge questions, game score, etc.)
- JAVA API to facilitate the communication between the app and the remote database.

- Remote database to store the data.
- Database interface to help analyze and read the data.

One problem with the proposed prototype is that it doesn't allow for any adaptability or customization in designing the game. Once the game has been designed, there isn't an easy way where the teacher can make modifications to the game. Also, this prototype doesn't have an offline support, and it only supports Android platform. Another gap is that instructors don't have the ability to go back and edit the assessment data. Introducing such a feature can change a lot. Giving the instructor the ability to make modifications is very important and can happen quite frequently.

On a different note, mobile devices are popularly used nowadays in enterprises and organizations and not only for personal purposes. Although the introduction of mobile devices to companies can have a positive impact, it's also important to consider the risks. These devices can include huge amounts of personal and work-related data, which can compromise internal investigations, effect intellectual property, and cause sabotages and embezzlements within the organization. A paper was written to try to overcome this problem. It was written by Justin Grover in 2013 at Rochester Institute of Technology under the title "Android Forensics: Automated Data Collection and Reporting from a Mobile Device" (Grover, 2013). As part of this paper, a prototype to monitor Android mobile devices was developed. The system is to collect as many data as possible about incidents, security auditors, security monitors, and forensic investigators. Most importantly, collecting data that can't be found in any of the other company's security tools. One problem with this prototype is that there's no monitoring tool. Implementing a remote screen where personnel with the right authority can watch curtain

mobile devices for monitoring purposes is important. Also, configuring proximity GPS alerts where the app will be triggered as soon as the phone reports a location near to a pre-defined set of coordinates would be helpful.

Lastly, another paper was written in the field of MDC titled "Remote Mobile Data Collection" written by Keenan Adamson in 2010 at the University of Western Cape (Adamson, 2010). EMIT, an open source web-app created by Cell-Life was an important part of this paper. EMIT is used to collect data remotely and in the field by the use of mobile devices. The author's main goal of this paper was to extend the features of EMIT by adding the following functionalities; data visualization, photos and videos integration, and network detection.

One disadvantage of this prototype is that it only works when connected to the Internet. Also, it doesn't have any support for data sharing or communication between different users.

Taken together, there are several tools and papers written in the field of mobile data collection. Mobile data collection revolutionized the old process of data collection and made it much more efficient and accurate, and by using mobile devices to collect data, error rate can be reduced drastically. Also, since data collection is used in mostly every field (i.e. education, business, politics, …etc.), having a tool to collect, store, visualize, and share data all in one place is crucial. The papers and tools listed above all had the same main goal. However, there are some areas where they all lacked. First and the most important area is data visualization. Most of those apps were used for primarily collecting data only. To visualize and analyze the data, a separate external tool might be needed. Second is the offline support. The main goal of MDC is the ability to collect data

on-site, which doesn't necessarily mean there will be Internet connectivity. Third, data sharing and communication is an area most existing tools lack of. This feature can be very helpful since it allows researchers and data collectors to share data among each other's instantly. The last and the most important shortcoming of most of these mobile apps is that they're created for a single use only. There's no room for customizing and modifying an existing app.

Model

Technology Model

A technology model is the list of al the services and resources needed to fulfill the application's requirements (Microsoft, 2016). Some of these resources include OS, network protocol, security standards, and many more. Since this thesis proposes two separate applications: a standalone Java application and an iOS mobile application, separate technology models will be discussed for each application. Table 2 explains the technology model for both applications.

System Design

System purpose, scope and overview. This paper presents a prototype that aims to allow users to create their own version of a modifiable iOS mobile app. The app serves the purpose of mobile data collection and mobile data analysis along with other important features.

The Java application is the interface in which users use to design their own app. Once the design phase is complete, a customized iPhone mobile application will be automatically generated. Users are able to create a new mobile app or update an existing app using the Java application. For now, updating an existing app is still under construction and is about 85% ready. After the mobile app is generated, it needs be installed on an iPhone, then it's ready for deployment.

The mobile app has the following features; add new entry, visualize existing data in 3 different formats (pie chart, bar chart and line chart), update old data, and share data with other instances of the app. In addition to that, downloading and uploading data enables the app to work on both online and offline modes.

Java application architecture. Figure 1 explains the architecture of this tool. The Java tool consists of 6 layers. First is the hardware layer. A Mac OS device is needed here. Second comes the interface layer, which includes all of the UI components (i.e. views, buttons and menus). The interface layer connects to two data access layers: local (file system), and remote database (using HTTP request). Then comes the database specification layer where the mobile app's database will be created. The specification will be taken from the previous layers. The final layer is the mobile app generation.

iOS mobile app architecture. Figure 2 explains the architecture of the mobile application. The architecture of the mobile app starts with the hardware device. It has to be an iOS device, preferably, iPhone 5 or 6. The first layer is the interface components. This layer consists of view controllers, more specifically, view controllers for creating dynamic forms that are responsible for entering new data and updating old data. Another view controller is needed for data visualization and charts libraries. Similar to the Java tool, this layer connects to two layers of data access: remote and local. The remote data layer is responsible for data synchronization. The local database on the other hand is used for temporarily storing data and sharing data with other instances of the app using Bonjour[5].

As illustrated in Figure 3, the prototype's main goal is to automatically generate an iOS mobile app. The app will be generated based on the user's input in the Java Application. Since this prototype presents two different applications, the design of each application will be discussed separately starting with the Java Standalone application.

[5] Bonjour discovers automatically other services and devices if connected to the same LAN.

Java application (form design phase). This is the only interface that the user (data collector) will be involved in. Everything else will be automated. At first, when this application is launched, the data collector will have two options; create a new application, or update an existing application.

Create new application. By filling a number of fields and answering couple of questions, this tool will generate a customized mobile app according to the user's design. This part consists of three views only.

View #1. First, the data collector will be asked couple of general questions about the mobile app such as the developer's name, the app's name, a brief description, and to what industry this app belongs to. All of these fields are optionally entered except for the app's name. That's because the app's name will be used locally to store the information about that particular app.

View #2. This view is divided into 26 rows. Each row represents one field. So the app can have from 1 to 26 different fields. Not all fields should necessarily be filled. One field at least has to be filled, and the rest can be empty. Each row contains a text field, and a menu item. The *name* of the field goes in the text field, and the *type* of the field will be selected from the menu. A field can be of type text, number, date, list, toggle button, or text view. It's recommended that the users spend as much time as needed to design the form since everything moving forward will depend on that. This will be the last step the user is involved in. The rest is all taken care of by the application.

View #3. This view will consist of a summary of the form design. The fields' names and types the user had picked along with the application's name will be presented on the screen. Together with that, every time the user generates a new app, a new "key" is

generated and is associated with that app. The key's value will also be shown to the user on the screen once the app has been generated. It's very important that the user keeps this key somewhere safe because they're going to need it if they decided to make any updates to the app. Lastly and most importantly, by now, the final iOS mobile application would have been generated on the user's local device. Since this is the step where the app gets created, it might take a while. The response time here takes between 20-30 seconds depending on how big the form is. Along with the app, a new database will be created with all the columns needed. The mobile app will be generated in the form of an Xcode project. All the user needs to do now is to connect their iPhone (or any iOS device) to the computer, select their phone as the running device, and finally run the project. By doing so, the app will be installed on the device, and it can run independently.

Updated existing app. This view will be used when the designer choses to update an existing mobile application. This part consists of two views.

View #1. Since it's possible that the user has created multiple applications using this tool, a list of all of the apps created previously will be shown in drop down menu. The user here has to select the app needs to be modified. For security reasons, the user here needs to provide the key that has been provided when the app was first created. If they click on verify, and the key was verified, only then they can move forward

View #2. In this view, the previously created form's information will be plugged into the fields. The user here can make as many updates as they desire. They have the option to remove old fields and add new ones. It's important to note that this part of the tool is still under construction, and not 100% complete.

iOS mobile application (data collection phase). After the mobile application is installed on an iPhone, it's recommended that the app be used on the landscape mode, and preferably, on an iPhone 5 or 6. While designing this app, it was reported from one of the supervising professors the importance of making the app functional online and offline as well. So, whenever there's Internet connectivity, the user can download all of the data located in the remote database and save them locally on the mobile device. This way, whenever the app is disconnected from the Internet, the user still has access to all of the data. The user will also have the ability to enter more data locally and visualize it. As soon as the mobile is connected again, the user can upload the local data to the remote database. The mobile app consists of 4 tab views:

Tab view #1 (data entry). Based on the form design phase, the fields will be dynamically created along with their labels in this view. On the navigation bar, two buttons are available: "Enter Data" to enter new data locally, and "Upload Data" to sync the local data with the remote database.

Tab view #2 (data analysis). This is a master-detail view controller[6]. The master side will list the fields' name. Once an element is selected from the master (left) side, a chart describing that element will be presented on the detail (right) side of the view. The user has three charts options: pie, bar and line chart. They can navigate through the charts. This allows the data collector to visualize the collected data as soon as it's been entered to the database. On the navigation bar, a button named "Download Data" is

[6] A master-detail view controller in Xcode allows the user to select an element from the list on the left (master), and inspect that element on the right side of the view (detail)

located. As explained previously, this allows researchers to download all of the remote data locally and access it offline.

Tab view #3 (update data). This is another master-detail view. A list of all of the records will be displayed on the master (left) side. Once an element is selected, information about that particular element will be displayed on the detail (right) side. The user here can make as many updates to the data as needed [7].

Tab view #4 (share data). Since entering data happens locally only, it can be helpful if multiple researchers had the ability to share the data they're collecting at the time. Sharing data requires all users to be connected to the same Local Area Network (LAN).

Database design. As in for the Java tool, the database wasn't designed in the conventional form. Instead, it uses local filing system to hold the form's information. A text file named same as the app's name is first created. Fields' number, names and types are stored into the txt file (See Figure 4).

When the mobile app is being created, in a previously created database, a new *table* will be added. This table will have as many columns as there are in the above text file, and it will hold all of the records to be collected in the future. The name of the table will be selected randomly from another database named "rand" (See Figure 5). "rand" has 100 records and two columns: "randNumber" and "taken". Randomly generated 6-digit numbers were fed to this table with the "taken" value of "0". Whenever a new table needs to be created, a "randNumber" with "taken = 0" will be selected as the table's name.

[7] The update button is still under construction.

“taken’s” value then will be updated to “1”. This “randNumber” later works as the “key” associated with the app.

The iOS Mobile database is quite different. Since the app works on both offline and online modes, there are two databases this app will connect to: remote one and a local one. The remote table is the one created previously using the java tool. This table will be used only when the user clicks on “Download” or “Upload”. The remote database will have access to the recent and most updated data. The local database on the other hand is temporarily. It’s used to store the currently collected data. Later on, the data collector needs to sync the temp data with one in the remote database.

Activity Diagram

Figure 7 and Figure 8 describe the flow of the system going from one activity to another. The rounded edge shapes represent an activity, and the diamond shapes represent a decision process.

Implementation

Case Study

Throughout this chapter, an example of the *common viruses* will be used to explain the implementation part of the system. The example seen in Figure 8 presents four clinics. Each clinic needs a mobile app to collect data inside the clinic about the common viruses. The data needs to be stored in a centralized DB, and the collectors need to visualize and analyze the data. Data collectors also need to share the data among each other's if needed. The research department manager had already designed the form of the app as seen in Table 5. The form consists of the following fields: virus name, virus number, patient's ID, patient's first name, patient's last name and recovery status. To start the process of developing the mobile app, first the manager will launch the Java tool and click on "Create New Application". Next, he/she will fill in the app's general information such as the app's name, brief description and industry as seen in Figure 9. Only the app's name is the required field, hence the red star. After clicking on next, he/she can start filling the form in Figure 10 based on the design they had in mind in Table 5. After clicking on next, the iOS mobile app will be generated, and the final view will be shown on the screen (See Figure 13). This view includes a summary of the app: the app's name, fields and types, and the key associated with the app.

Let's say that after 6 months, the manager decided that's important to add a new field named "blood pressure" of type "Number Field". To do that, the manager needs to launch the Java tool again, and click on "Update Old Application". This will take him/her to another view where they need to verify the app (See Figure 11). In the menu bar, all the apps created locally will be listed. After selecting the app to be modified, the manager

now needs to verify that they're the owner of the app be entering the *key* generated when the app was first created. Only when verified, they can move on to the next step, which will take them to the next view. This next view has all of the existing fields and their types. Now the manager can make as many updates as they desire: adding new fields or removing old fields.

Now, let's see what does the generated mobile app (Common_Viruses) look like. Going to the Java Tool directory, the manager needs to go to: "/Application/<Common_Viruses>/xcode/IntelSplitDemo/IntelSplitDemo.xcodeproj". He/she will open and clean the project, connect an iPhone to the laptop, select the iPhone as the running device, and finally run the project. After several minutes, the app will be installed on the iPhone[8]. The main view of the iOS app will look like Figure 12. The app has four different tab views: new entry, data analysis, data update and data share. Comparing Figure 12 with Figure 10, you will notice that the same form design used in the Java tool is reflected now in the mobile application. For the sake of this demo, part of the data in Table 1 was entered.

Now, the entered data can be visualized in the second tab view "Data Analysis". In figure 13, after Ptient_First_Name is selected, a pie chart showing the data based on the patient' first name is presented. The same chart was applied on Virus_Name. Bar chart and line chart can also be selected as the chart type. As seen in Figure 13, you can extract meaning from the entered data by taking a look at the chart. From the first pie chart, you can tell that that Sara has two viruses where Isabella and Mary have only one

[8] Note that the iOS screenshots are taken from an iPhone 6SPlus simulator, not an actual iPhone.

virus. From the second pie chart, you can see that two people are suffering from Chickenpox, one person has Hepatitis A, and another person has Hepatitis B.

Update Data tab view is for updating data. Before updating data, the physician needs to click on "Upload Data" first in the "Data Entry" view. That's because the updates made will happen on the remote database not the local one (See Figure 15). As seen in Figure 14, this is another master-detail view. The master side takes the *first* columns in the database (i.e. in this case, Virus_number), and shows the data accordingly. Clicking on the virus number, data about people with that virus will be displayed. As seen in Figure 14, two people are having the Chickenpox: Isabella and Sara. To get more information about these two people, you can scroll right and left. The "Update Info" button is still under construction.

The final tab view is sharing data. If multiple physicians are collecting viruses' data, and it happens that they are both connected to the same LAN, they can easily share their data. Using NSService and Bonjour protocol, multiple instances of the app can discover each other, and share their local data by clicking on "Share" as seen in Figure 16.

System Functions

In this section, the technical details and functions will be explained of both the Java and the iOS applications.

Java application components. The Java application was created using NetBeans and JavaFX. The main class is named DesktopApp. Table 3 lists all the UI components for creating a new application. Table 4 lists all the UI components for updating an old application. See Figure 16 for the Java application class diagram.

Mobile application components. The iOS Mobile application is created using Xcode and objective-c. In Figure 17, a class diagram shows how the classes are categorized in this application. As seen, it starts with:

Main tab bar controller. The class named "IntelligentSplitViewContoller" is Responsible for rotating the tab bar properly (Combs, 2014).

Data analysis. This part has the following classes:

- MasterViewContoller: Lists column names (i.e. fields) in a table form
- DetailViewContoller: Shows visual graphics based on the selected item from MasterViewContoller and the selected chart type:
  - Line Chart: This library draws a line chart based on the given values (Spencer, 2014).
  - Bar Chart: This library draws a bar chart based on the given values (Dhilipsiva, 2014).
  - Pie Chart: This library draws a pie chart based on the given values (Dlilaramani, 2013).

Data entry. This part has the following classes:

- newEntryViewContoller: This class dynamically creates all the fields and labels and places them on a UIScrollView. Two important functions are located here:
  - onEnter: Enters data locally in two databases: db.sql and db2.sql
  - onUpload: Uploads the data in db2.sql to the remote database.
- DBManager: Responsible for loading data from the sql file, and executing queries (Theodoropoulos, 2014).

Data update. This part has the following classes:

- updateMaster: This class is responsible for the master side of the view. It lists all the unique data of the first column in the database. For example, virus number was the first field in the common viruses example. The data listed here will be all the unique virus numbers.
- updateDetails: Based on the element selected from the master side (for example, virus number 101), then information about that virus will be shown in editable text fields. All labels and text fields are created dynamically and added to a UIScrollView

Data share. When multiple instances/users of the same app would like to discover and share data with each other's, they first need to be joined in the same room. Once that happens, they will be able to share data.

- ShareViewContoller: In this class, there's the option to either create a room or join a room. If the user is the first to start the sharing the process, then the program will go with the option of "Creating a new room". All other users will be joined in that same room. Therefor, they can start sharing their data. The function responsible for this is named "createOrJoin". After the user has created a new room, or just joined an existing room, this class calls ChatRoomViewController automatically.
- ChatRoomViewContoller: Here is where the user can click on "Share"**.** onShare basically broadcasts the shared message to all the other users who're joined in that room. The shared data will be stored in each app's local DB. (Bakhyryev, 2013).

Performance

The performance of the software depends a lot on the design of the system. Both the Java application and the iOS Mobile application are designed precisely and carefully to make the user experience easy and smooth. Table 6 lists all the performance metrics for both the Java and the mobile applications.

Java application. It consists of only 3-4 views. Each one of them is self-explanatory. All the fields and buttons and the menu items are labeled in an understandable way.

iOS mobile application. The design of the tabs made the application clear and easy to use. A tab named "Data Entry" easily indicates that this tab is used for entering new data, same for "Data Analysis", and "Data Update".

Constraints

Although this prototype overcomes a lot of the existing problems available in mobile data collection, it still has a number of constraints:

1. Single-table form design: When designing the form, the user has to keep the form design very simple. There is no room for multiple tables and relationships among them. They have to compress all of their data into one table and one form only. For example, having a table for students and another table for courses is not possible with this project, or at least for now.
2. Number of users: For now, only 100 apps can be created using this tool. Further expansion can take place.

3. Number of fields allowed: For now, the app can have a maximum number of 26 fields only.

Testing

Java tool. This tool was tested on a Mac OSX 10.10.5.

Mobile application. This app was tested on an iPhone 6S Plus simulator in addition to an iPhone 5. The app works best on a landscape mode.

Results

Prototype Assessment & Evaluation

In Table 1, a list of promised functionalities and features were provided. Here in this section, the evaluation of this list will be provided (See Table 6). Next to each feature, a percentage value will be given to show how much of that feature was completed. The completed functionalities were all explained in chapter 3. Here, the incomplete parts will be explained.

1- Specifying fields types in the Java tool: Users are able to specify fields' types in the Java application. However, these specifications will not be reflected yet in the mobile application. All fields for now are text fields no matter what type where they.

2- Generate mobile app based on the updated form design: Users are able to update the form design, but the changes will not be reflected on the current database. The changes will be only reflected on the mobile app itself.

3- Update existing data: In the mobile app, although users can view the data under "Update Data" tab, the "Update" button doesn't function yet.

Challenges

1. Lack of input: The input of Dr. Lindquist was greatly valuable. However, input and feedback from other researchers and potential users would've been extremely helpful. Switching from the developer's mode to the user's mode can be challenging sometimes. Dr. Bateman from the Biology department had great comments in general that helped a lot, which was very much appreciated.
2. Lack of time and resources: It was extremely challenging to create this whole prototype in the time constraints. Also, since this whole prototype was created from scratch, being able to finish it with little or no help was another challenge.
3. One major part of the mobile app is the ability to share data among other instances of the app. This part was one of the most challenging features in this app. The whole idea of Bonjour and NSService was completely new to me. However, after lots of readings, tutorials, and many hours of testing and coding, I was able to conquer the concept and achieve the idea in mind.

Strengths

1. The general purpose of this prototype is one of its main strengths. Having a system where users can generate their own mobile applications with no coding involved is powerful.
2. In the field of mobile data collection, not many applications have all of the proposed features combined. Having data collection, data analysis, and data sharing features all in one tool is another advantage.
3. This system is complex yet so simple. From the user's point of view, they only need to be involved in 2-3 simple steps, and the rest is all taken care of. It seems like nothing much is happening, but getting a fully functioning mobile application that was customized based on the user's needs is not an easy thing to achieve.
4. One of the mobile app's strongest features is the ability to share the data among different users. By using Bonjour protocol and NSService, that was achievable.

Weaknesses

1. No deployment: It would've been more helpful if the prototype was actually tested and deployed by different researchers. However, due to the lack of time and resources, that wasn't possible. Getting feedback from actual users on how the system is responding and the feel of it would've enriched the whole experience
2. Security: The code of the mobile app has immediate access to the remote database. If the users know what they're doing, they can manipulate the database and the data in it.

Conclusion and Future Work

Mobile data collection has revolutionized the process of data collection. It's become feasible to collect data on site without the need of an isolated environment. In this research, two main problems with mobile data collection were detected. The first problem is the lack of customization and adaptability. Every application is used to serve one purpose only. Second, most mobile data collection applications are used primarily for data collection. If data analysis and data communication features were needed, an external tool needs to be used.

In this thesis, an attempt to solve the two problems listed above has been made. First, Java application was created to guide users to in designing their mobile applications. Once the mobile app is designed, an iOS mobile application will be generated. Additionally, the mobile application allows users to visualize data in three different charts, synchronize data to the remote database, share data with other data collectors and update existing data. This prototype can be extended and advanced in the future by accomplishing the following:

1. Android Version: This system allows users to generate their own iOS mobile application. It would be also extremely helpful if an Android version was also issued.
2. Update existing app: For now, updating the existing app is only 85% done as explained earlier. Having this feature completed and at 100% can be valuable. Allowing users to update their apps along with the associated database is crucial at this point.

3. Fields Types: As mentioned earlier, in the Java tool, if the user selects any field type other than “Text Field”, that change won’t be reflected in the mobile app. Having this feature can be very important in reducing error rate and increasing efficiency. Having a list in which the user has to select only one element from instead of manually typing it is very helpful.
4. More Complex Database: The ability to design a more sophisticated database with multiple tables and relationships.
5. Incorporating video, audio and photos as data format

Works Cited

Adamson, K. (2010). *Remote mobile data collection* (Unpublished master's thesis). University of the Western Cape.

Anokwa, Y. (2010, November 9). Open Data Kit. Retrieved March 20, 2016, from https://opendatakit.org/blog/page/35/

Apple: Most popular app store categories 2016 | Statistic. (2016, March). Retrieved March 20, 2016, from http://www.statista.com/statistics/270291/popular-categories-in-the-app-store/

Bakhyryev, P. (2013, May 23). Chatty [Computer software]. Retrieved March 21, 2016, from https://github.com/nuthatch/chatty

Baskaran, S. (2012). *A rapid deployment model for VGI projects in mobile field data collection* (Unpublished master's thesis). Iowa State University.

Blanford, J. (2014). Technology Trends: Volunteered Geographic Information. Retrieved March 21, 2016, from https://www.e-education.psu.edu/geog583/node/43

Brain Games & Brain Training - Lumosity. (2005). Retrieved March 20, 2016, from http://www.lumosity.com/

Combs, G. (2014, June 8). Intelligent SplitView Controller [Computer software]. Retrieved March 21, 2016, from https://github.com/grgcombs/IntelligentSplitViewController

Grover, J. (2013). *Android Forensics: Automated Data Collection and Reporting from a Mobile Device* (Unpublished master's thesis). Rochester Institute of Technology.

Microsoft. (2016). Technology Model. Retrieved March 20, 2016, from https://msdn.microsoft.com/en-us/library/aa266924(v=vs.60).aspx

Narhan, J. (2013, July). Last Mile Mobile Solutions. Retrieved March 21, 2016, from http://www.lastmilemobilesolutions.com/

Quizlet. (2016). Retrieved March 20, 2016, from https://quizlet.com/

Shao, D. (2012). *A Proposal of a Mobile Health Data Collection and Reporting System for the Developing World* (Unpublished master's thesis). Malmö University.

Songar, P. (2012). *Learning assessment data collection from educational game applications* (Unpublished master's thesis). Kent State University.

Spencer, S. (2014, May 26). BEM Simple Line Graph View [Computer software]. Retrieved March 21, 2016, from https://github.com/Boris-Em/BEMSimpleLineGraph

Sundt, M. (2015, July 7). Open Data Kit » Collect. Retrieved March 20, 2016, from https://opendatakit.org/use/collect/

Sylvia, M. (2014, January 26). XLSForm.org. Retrieved March 20, 2016, from http://xlsform.org/

Theodoropoulos. (2014, June 25). DBManager [Computer software]. Retrieved March 21, 2016.

Wogan, D. (2014, January 28). What is Mobile Data Collection? Retrieved March 20, 2016, from http://www.quicktapsurvey.com/blog/2014/01/28/what-is-mobile-data-collection/

CartONG. (2012, May 25). *Instructions/Training Material for Android: ODK* (Manual). Retrieved March 23, 2016, from UNHCR website: http://parkdatabase.org/files/documents/odk_stepbysteproadmapv2.pdf

Dhilipsiva. (2014, July 15). DS Bar Chart [Computer software]. Retrieved March 21, 2016, from https://github.com/dhilipsiva/DSBarChart

Dlilaramani. (2013, May 28). DL Pie Chart [Computer software]. Retrieved March 21, 2016, from https://github.com/dilipajm/piechart/tree/master/Pie Chart Library

OpenXdata. (2010, October). OpenXdata. Retrieved March 21, 2016, from http://www.openxdata.org/

APPENDIX A

TABLES AND FIGURES

Table 1

Promised Functionalities and Features

Application	Functionality
Java Application	- Design a new form - Specify fields names - Specify fields types - Generate app based on the new form design - Update an old form - Generate app based on the old form design
iOS Mobile Application	- Enter data offline - Upload local data to remote DB - Download remote data to local DB - Visualize data in a pie chart - Visualize data in a bar chart - Visualize data in a line chart - Update existing data - Share data

Note. This is the list of features to be delivered in this prototype for both the Java tool and the mobile application.

Table 2

Technology Model

Service/Resource	Java Application	iOS Mobile Application
Operating System	Mac OS	iOS
Hardware	Any Mac OS Device	Any iOS device. Most preferably, iPhone 5 or 6
Network Protocol	In order for this application to function properly, an access to 802.11-network protocol needs to be provided.	The application works on two different modes: online and offline. When the app is connected to an 802.11 protocol, all the app's features and functionalities will be available in addition to access to the centralized data. When the app is disconnected from the internet, it will also have access to all of the app's functionalities, only locally this time without access to any remote data.
Database Accessibility	This application will have access to a small database where it will take the new database's name. It More on this will be explained in Section 2.2.3.1. It also has access to a local data file system where information about the designed mobile app and form are stored.	If the app was on the online mode, it will have direct access to the centralized database. The user then can download the data locally, and upload the local data to the remote DB. If not connected to Internet, then all data will be stored temporarily in a local database.
Remote Data	Accessible	Accessible only when it's connected to the internet.
Tools needed to deploy application	JDK	Xcode

Table 3

Java tool UI components for creating a new app.

Class Name	UI Component Type	UI Component Name	Purpose
DesktopApp (Figure 10)	Button	newApplication	Takes the user to the class named newInfo.
	Button	existingApplication	Takes the user to the class named oldInfo.
newInfo (Figure 11)	Text Field	dNameField	Takes the developer's name.
	Text Field	appNameField	Takes the app's name.
	Text View	appDescField	Takes the app's description.
	Combo Box (Menu)	industryField	Lists a number of industries in which the app might belong to.
	Button	start	Takes the user to the class named newApp.
newApp (Figure 12)	Text Field	fields	26 of them to hold the fields names.
	Combo Box	fieldsTypesComboBox	Has the following field types: Text Field, Number Field, Date Picker, List, Toggle Button, Text View.
	Button	next	Takes the user to the class named AppGenerate.
AppGenerate (Figure 13)	Label	key	Holds the key value.
	Label	appName	Holds the app's name.
	Table View	table	Holds the fields' names and types.

Table 4

Java tool UI components for updating an old app.

Class Name	UI Component Type	UI Component Name	Purpose
oldInfo (Figure 14)	Combo Box	oldAppNames	Lists all the local apps.
	Text Field	key	Takes the key's value.
	Button	verify	Responsible for verifying the entered key.
	Button	updateThisOne	Takes the user to the class named oldApp.
oldApp (Figure 15)	Text Field	fields	26 of them to hold the existing fields' names.
	Combo Box	fieldsTypesComboBox	Has the following field types: Text Field, Number Field, Date Picker, List, Toggle Button, Text View.
	Button	next	Takes the user to the class named OldAppGenerate.

Table 5

Common Viruses DB

Virus_ Number	Virus_ Name	Patient_ ID	Patient_ Fname	Patient_ Lname	Recovery_ Status
412	Chickenpox	852	Isabella	Hernandez	70%
413	Hepatitis A	125	Sara	Lee	45%
414	Hepatitis B	327	Mary	Robb	35%
415	Hepatitis C	349	James	Jones	60%
570	Influenza	240	Jacob	Miller	50%
699	Measles	240	Jacob	Miller	20%
720	Shingles	470	Ethan	Garcia	30%
745	West Nile River	317	Tyler	Rodriguez	25%
750	Yellow Fever	459	Aiden	Wilson	10%
412	Chickenpox	125	Sara	Lee	75%

Table 6

Performance Metrics

Metric	Java Tool	iOS Mobile App
Max Page Load Time	5-7 Seconds (App Generation Page)	3 Seconds (Data Sharing Page)
Max Number of Registered Users	100 users	100 users
Max Records in local DB	26 Fields	Up to 270Mb (Approx. 6000 records)
Max Records in remote DB	NA	Unlimited
Max Concurrent Connections	NA	217 connections

Table 6

System Assessment and Evaluation

Application	Functionality	Completion Percentage
Java Application	Design a new form	100%
	Specify fields names	100%
	Specify fields types	50%
	Generate app based on the new form design	100%
	Update an old form	100%
	Generate mobile app based on the updated form design	85%
iOS Mobile Application	Enter data offline	100%
	Upload local data to remote DB	100%
	Download remote data to local DB	100%
	Visualize data in a pie chart	100%
	Visualize data in a bar chart	100%
	Visualize data in a line chart	100%
	Update existing data	75%
	Share data	100%

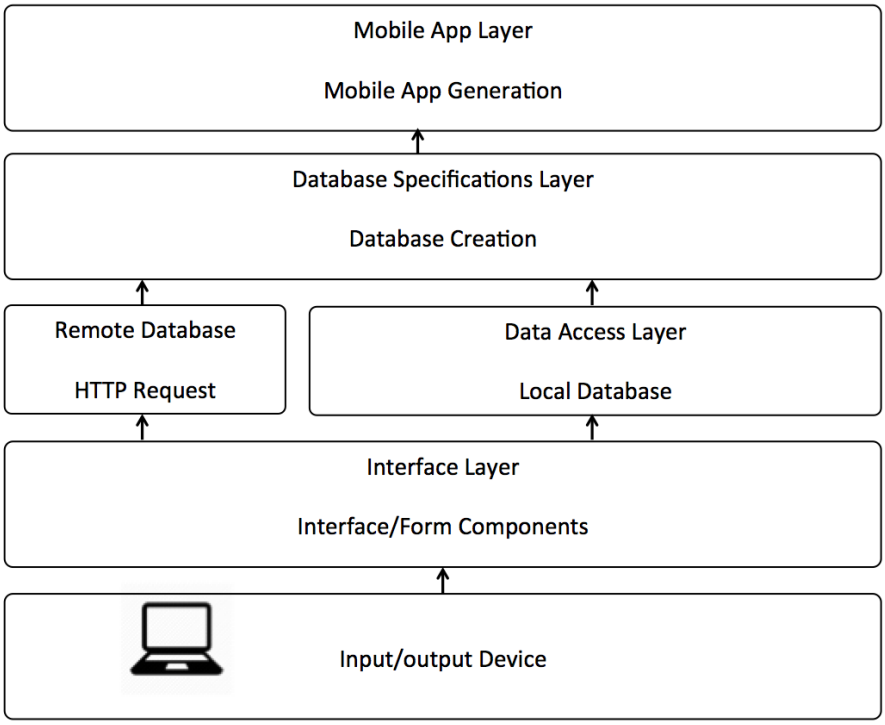


Figure 1. Java Application Architecture

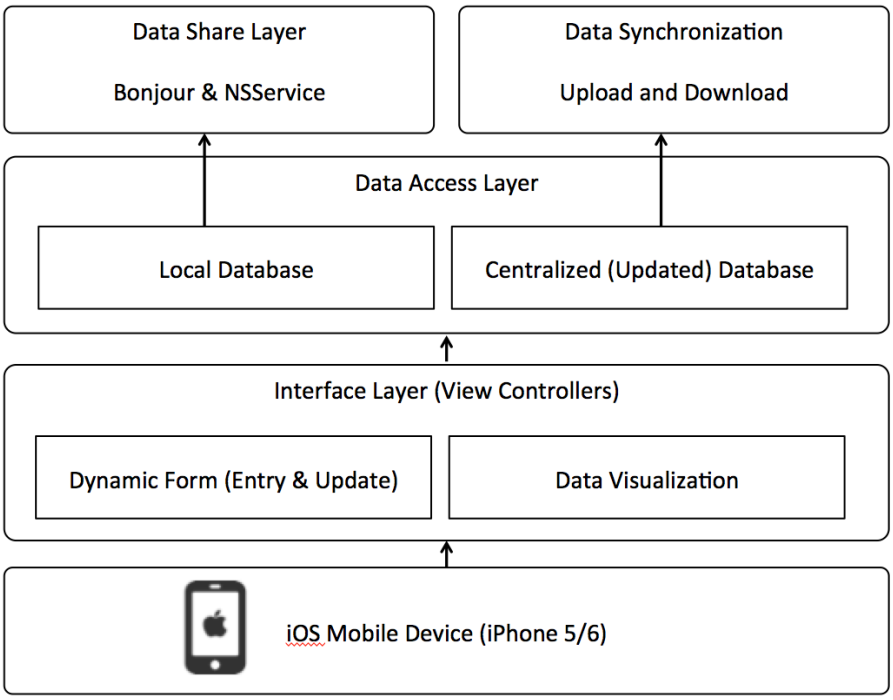


Figure 2. iOS Mobile Application Architecture

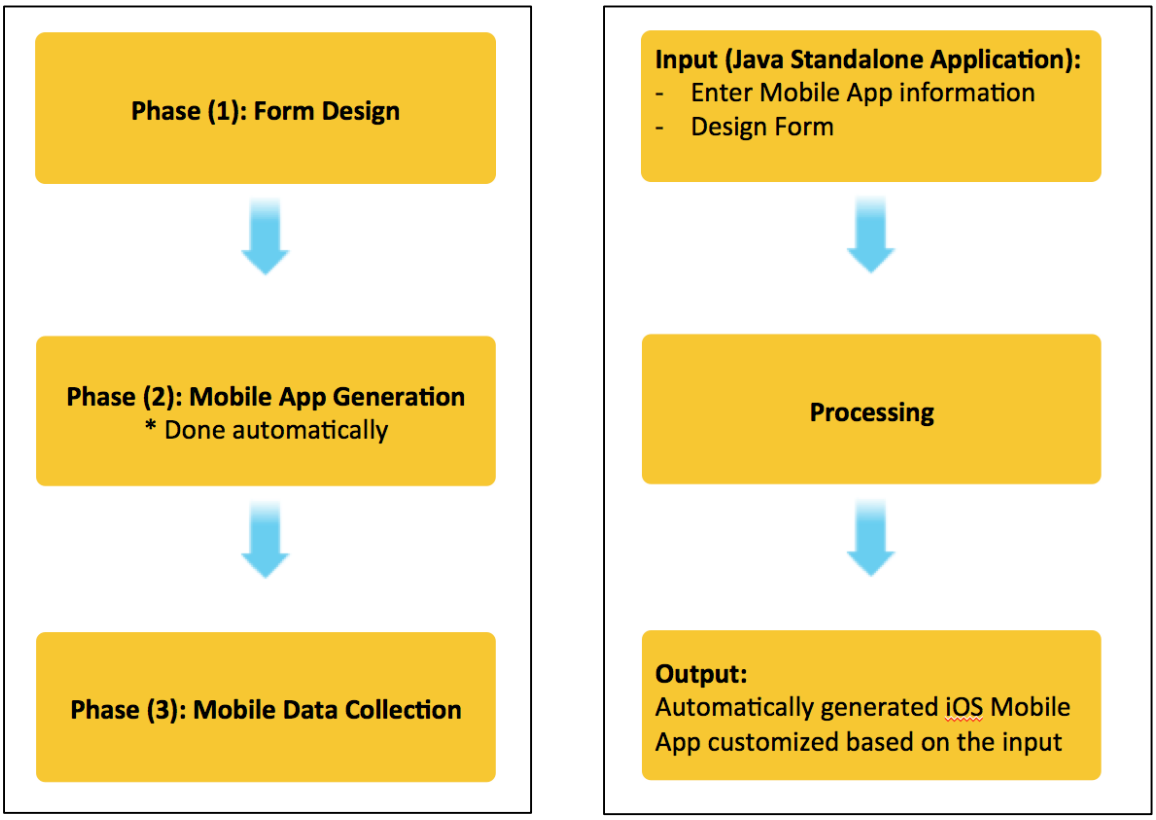


Figure 3. The diagram on the left conveys the three phases this system goes through. After the user has designed their form, the mobile application will get automatically generated, and then the mobile data collection can start. On the right side, this figure represents the input, processing and output of the system.

```
Row Number: 1
Virus_Number: Number Field
Row Number: 2
Virus_Name: Text Field
Row Number: 3
Patient_ID: Number Field
Row Number: 4
Patient_Name: Text Field
```

Figure 4. This picture here is an example of how the local database might look like in the Java application. Its starts with the row number, then the fields name then the type.

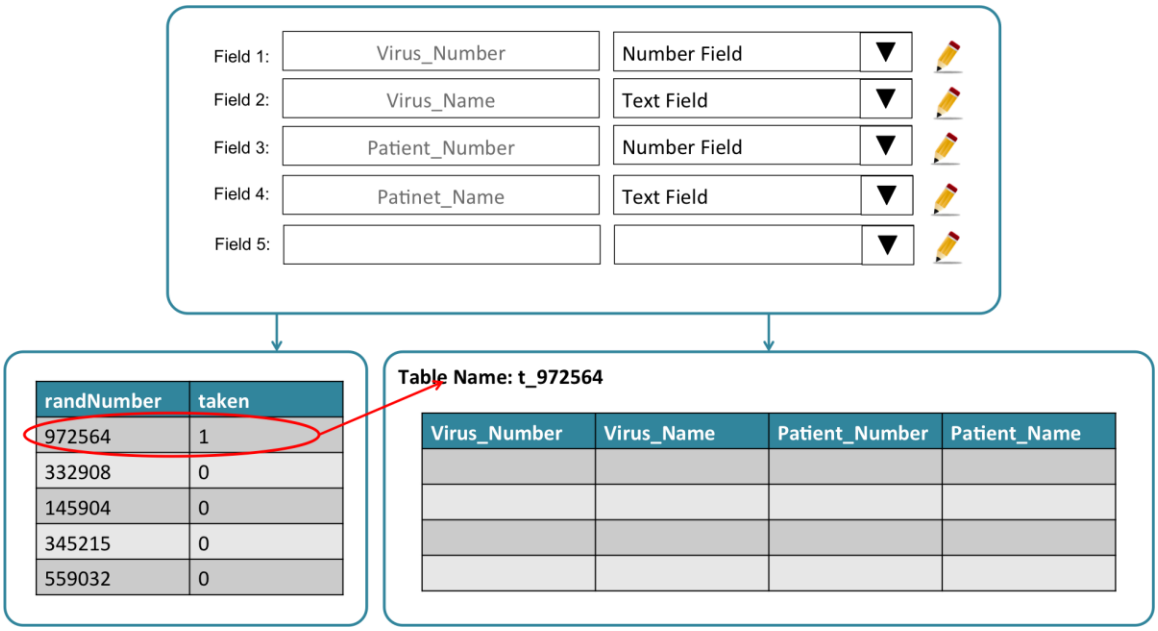


Figure 5. This figure shows the process of choosing the table's name where the data entered will be stored. From the database "rand", a random number with the value "taken" is zero will be selected as the table name. Then "taken" will be updated to one as seen in the picture.

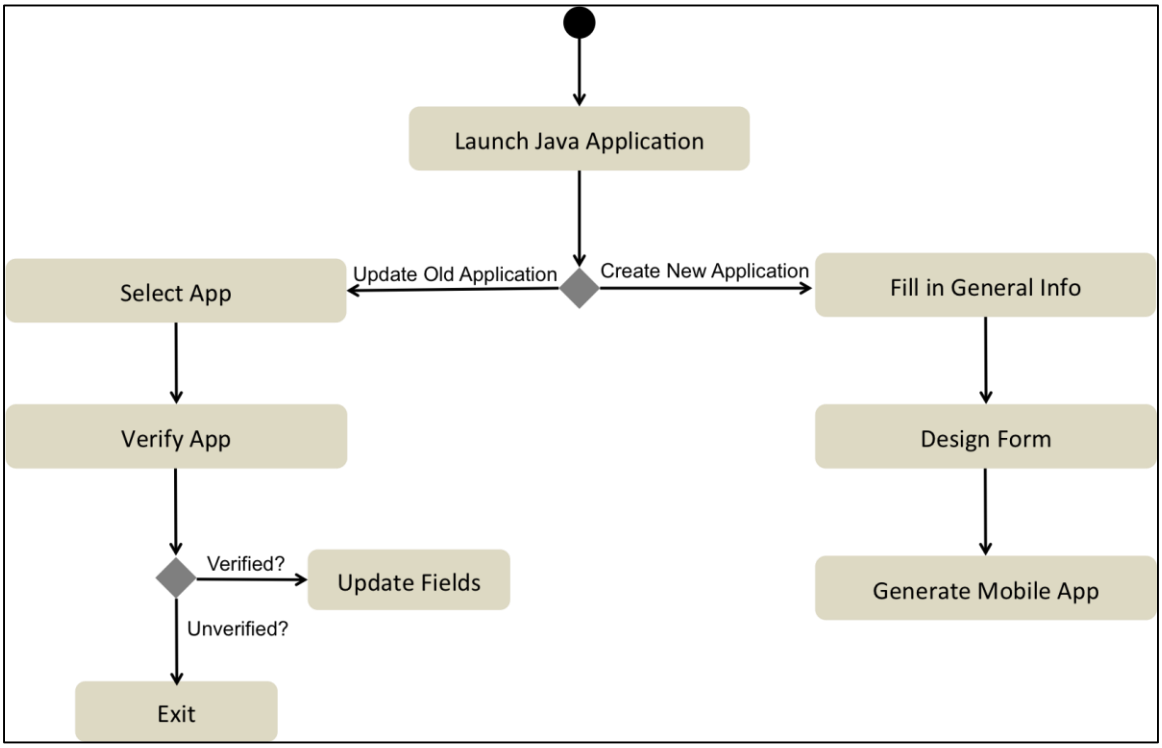


Figure 6. This diagram shows an activity diagram of the Java application. The black circle is the starting point, the round-edged rectangular shape is an activity and the diamond shape is a decision.

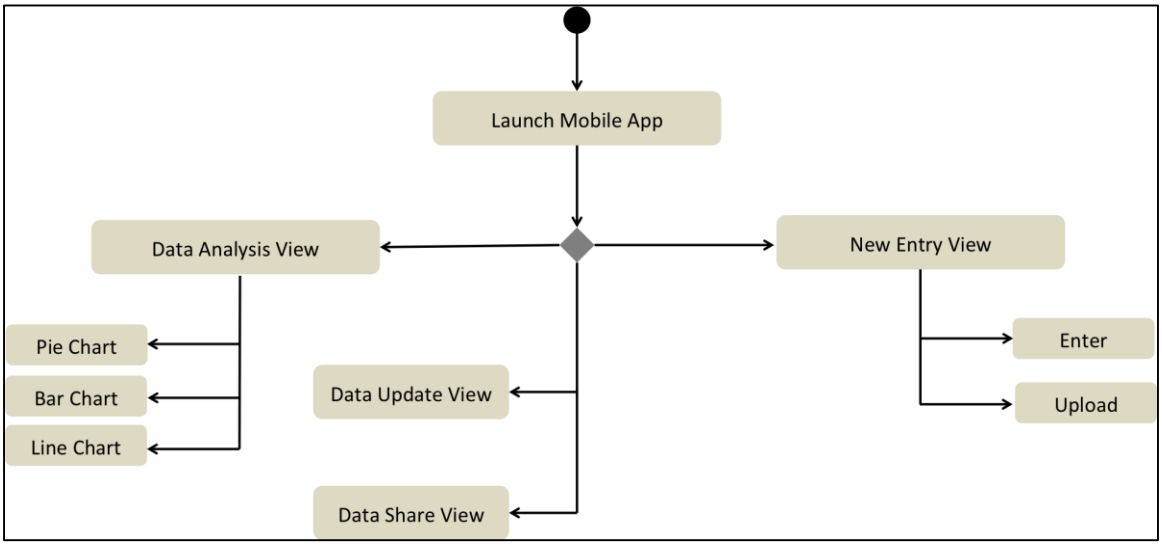


Figure 7. This diagram shows an activity diagram of the iOS mobile application. The black circle is the starting point, the round-edged rectangular shape is an activity and the diamond shape is a decision.

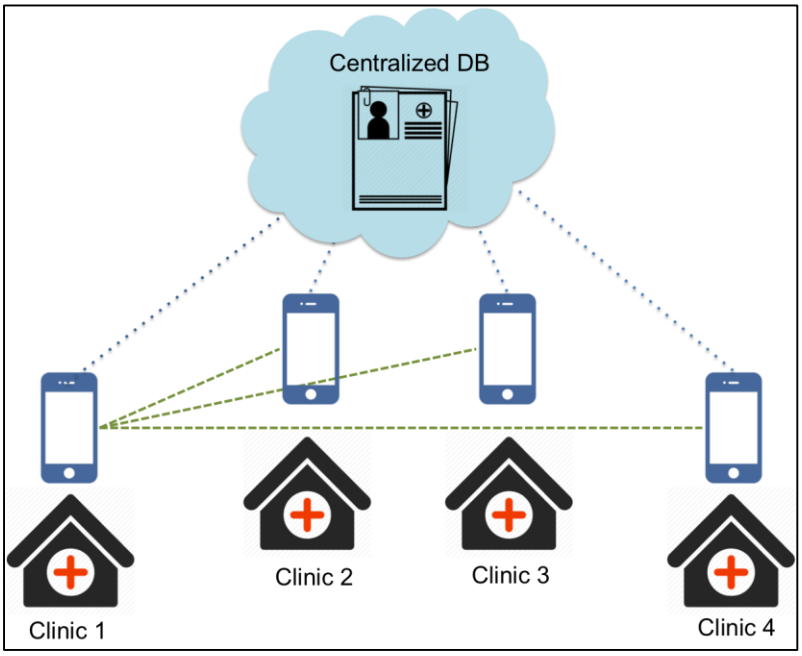


Figure 8. This picture illustrates the example of common viruses explained in chapter 3. As seen, multiple clinics/physicians can have access to the same mobile application. They later can collect data locally, upload it to the centralized DB, and share data among each other's.

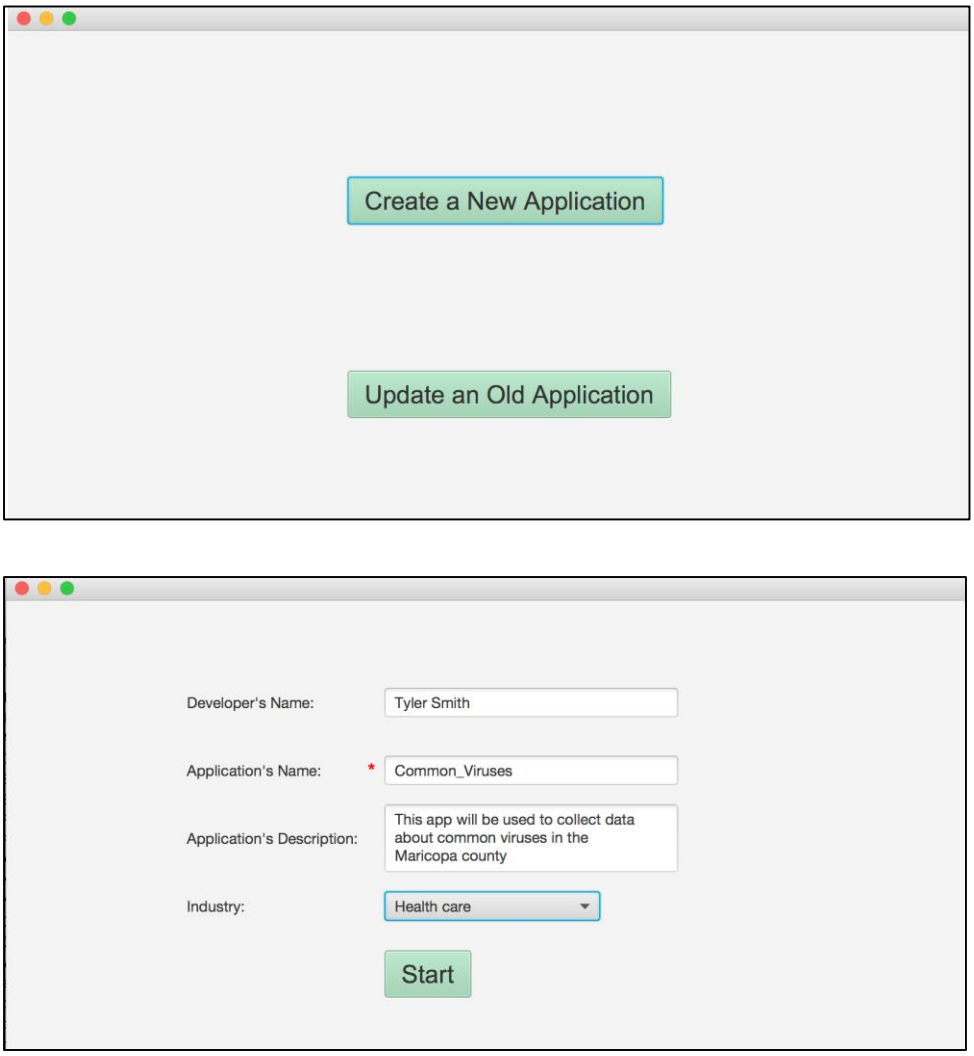


Figure 9. This figure shows the first 2 screens the user will see when this app is launched. After clicking on "Create a New Application", they will be guided to another view where they need to fill the general information of the app as seen in the picture.

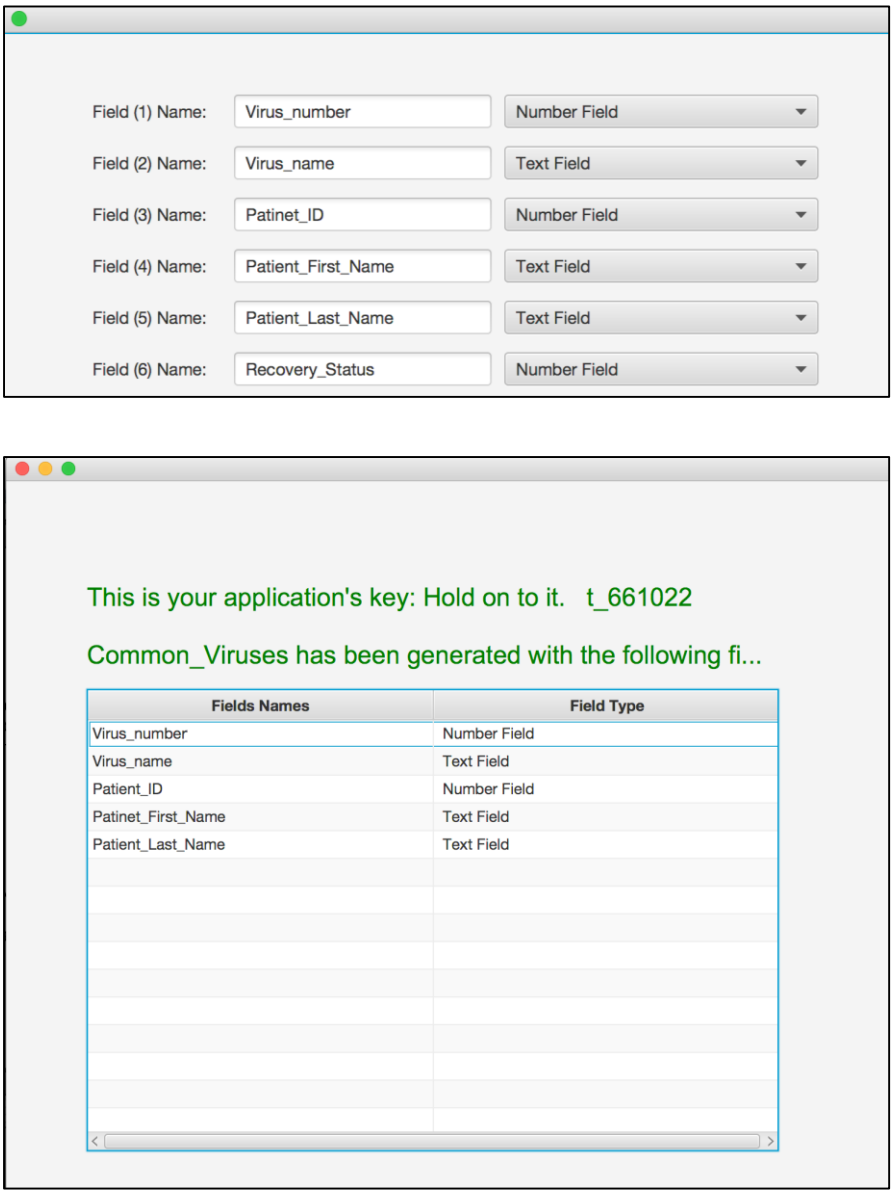


Figure 10. This figure shows the last 2 screens the user will see. The 1st one looks like a form they have to fill. Here, they will specify the fields' names and types, then click on "Next". After that, a screen similar to the 2nd picture will pop. It will hold the app's name, the designed form and app's key.

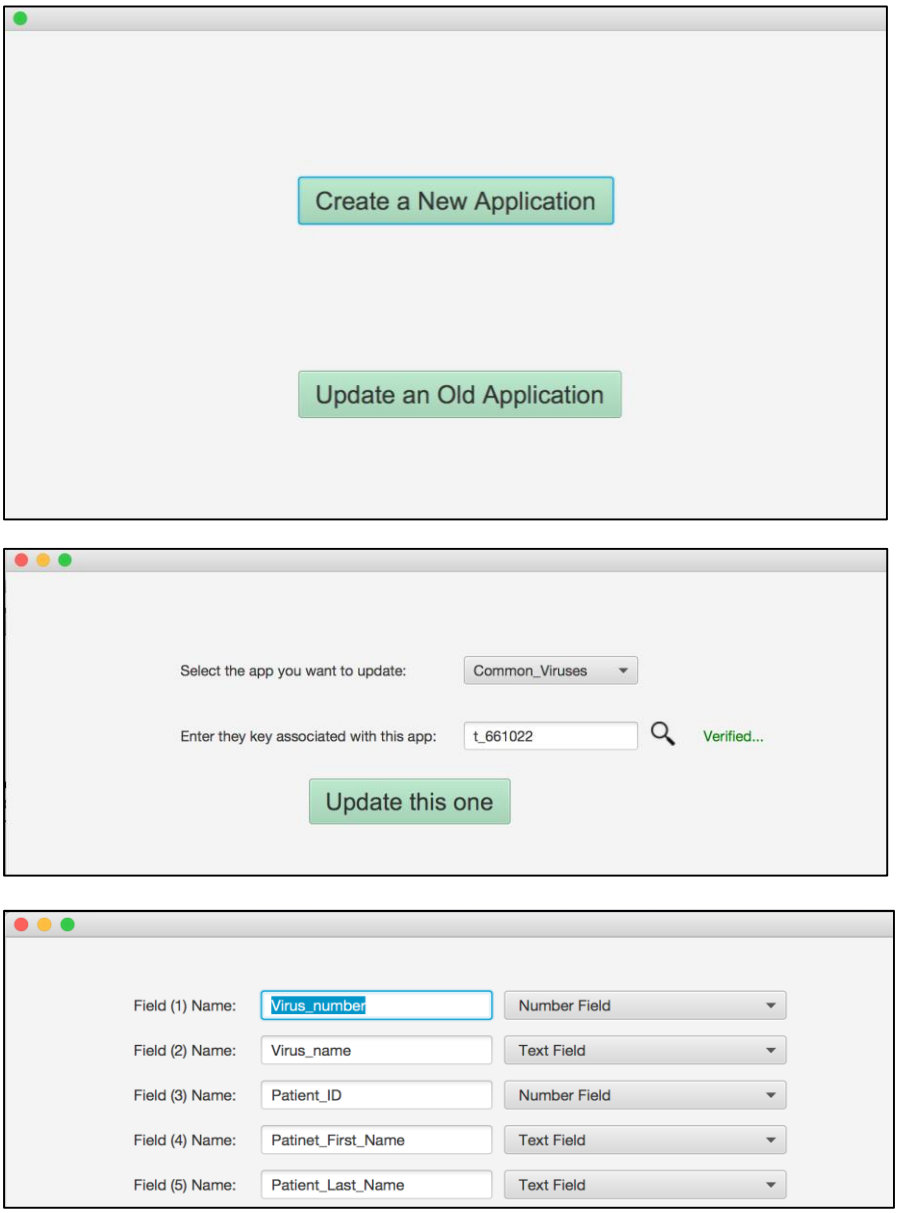


Figure 11. If the user chose to select on "Update Existing App", the 2nd screen will pop. Here, they need to select the app they want to update. For security reasons, they need to enter the app's key. If verified, they can move on to the next step. After clicking on "Update", they will be guided to the 3rd screen where they can make as many modifications as the desire.

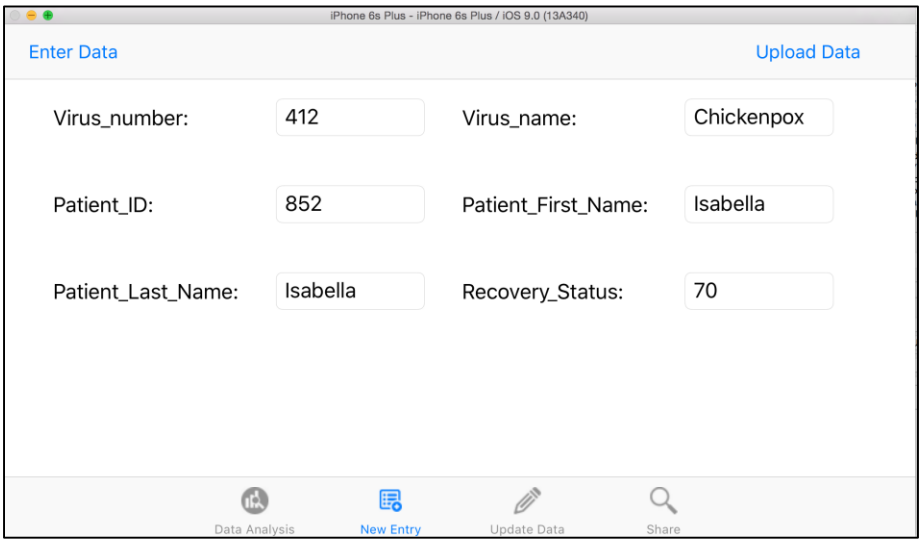
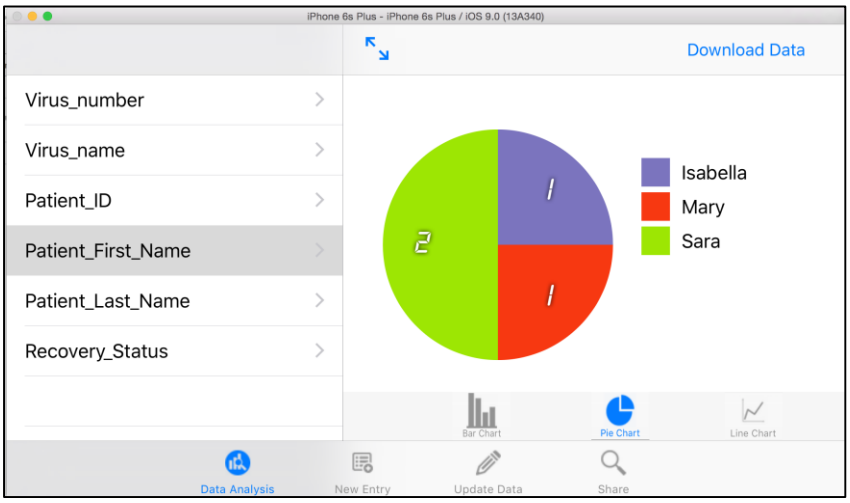


Figure 12. The "New Entry" tab view in the mobile application will hold all the fields' names specified in the Java tool.



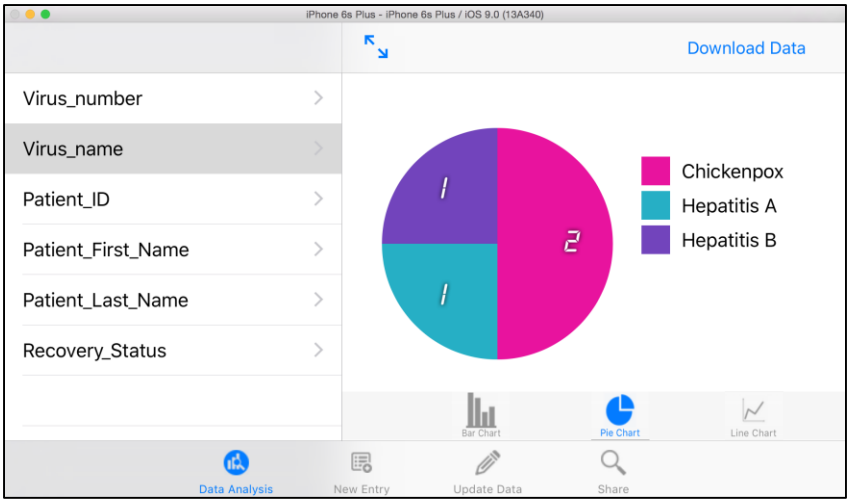


Figure 13. These 2 screenshots show how users can use the app to visualize the data they entered. There are 3 types of charts: pie, bar and line. In the 1st screenshot, data are visualized based on the patient's first name in a pie chart. The 2nd one shows the same data but based on the virus name.

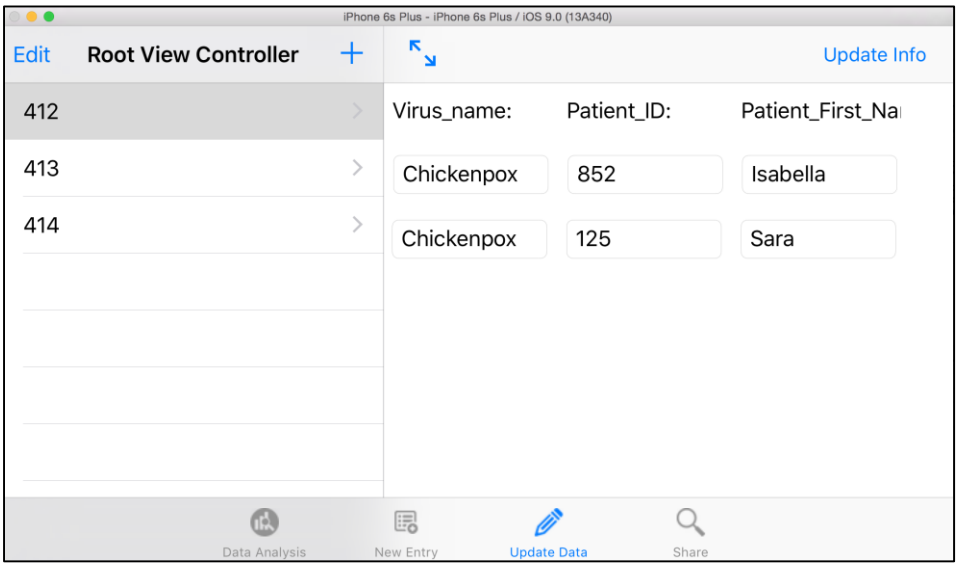


Figure 14. This screenshot shows the "Update Data" tab view. Patient's IDs are on the left side, and the information of each patient will be showed on the right once selected.

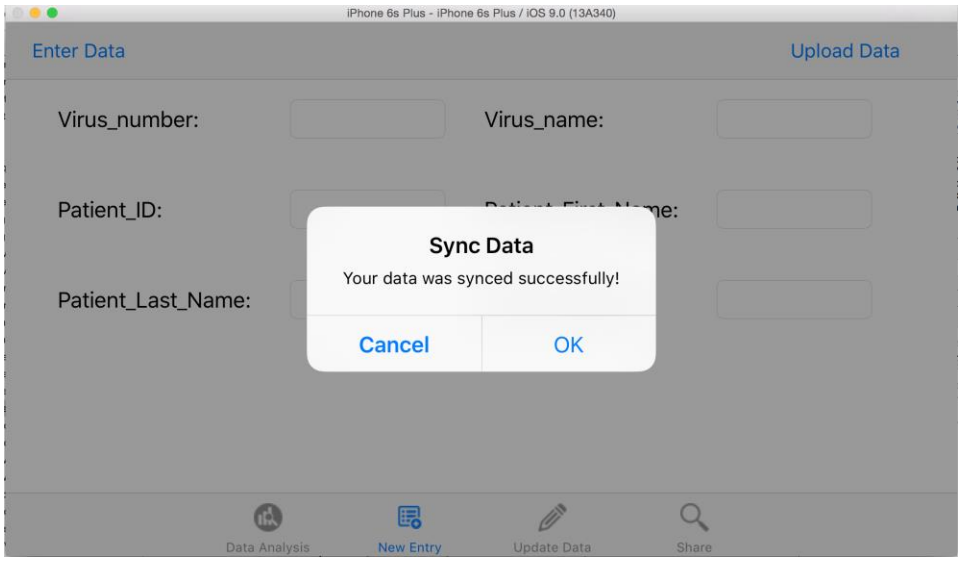


Figure 15. Whenever the user is done collecting data locally, then can go to the "New Entry" tab view and select "Upload Data". This button will send all of the local data to the remote database.

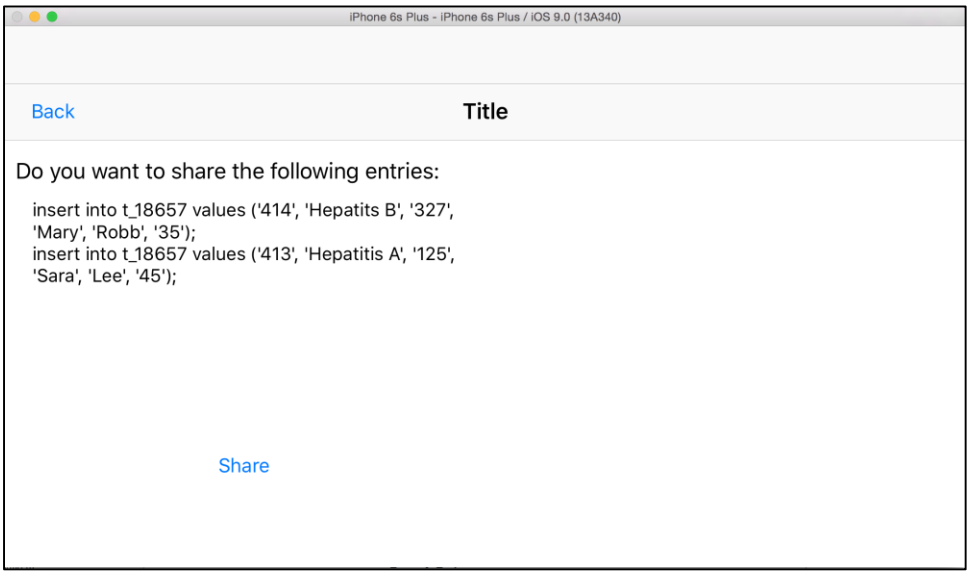


Figure 16. The user can choose to share their local data to other users. To do so, they need to go to the "Share" tab view. By doing so, the app will look for other instances of the app, join you in a room with them. If a room was initiated, then the user can click on "Share" which will send their local data over to the other users.

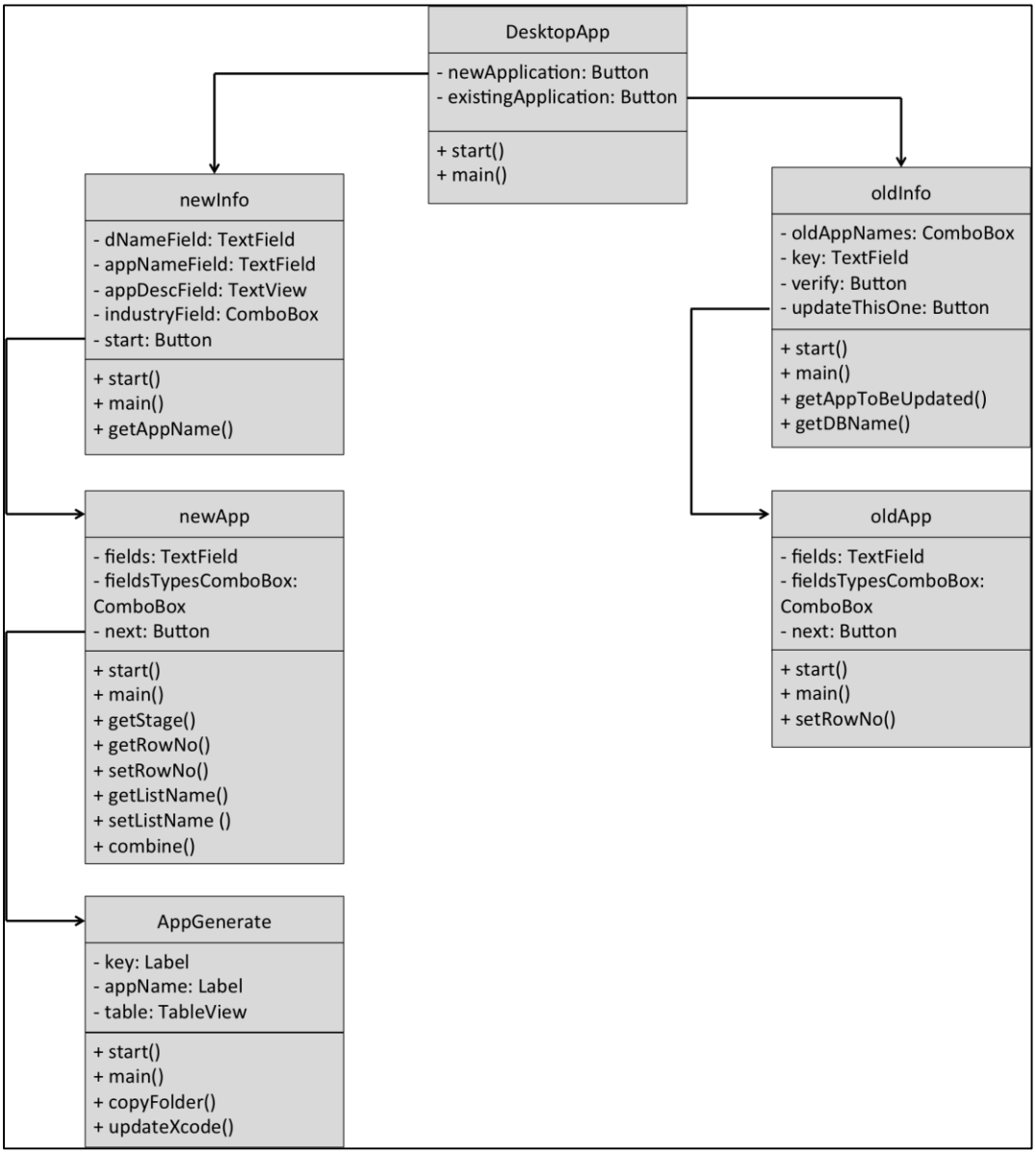


Figure 17. This is the class diagram of the Java tool. It represents what classes are there and how do they communicate with other classes.

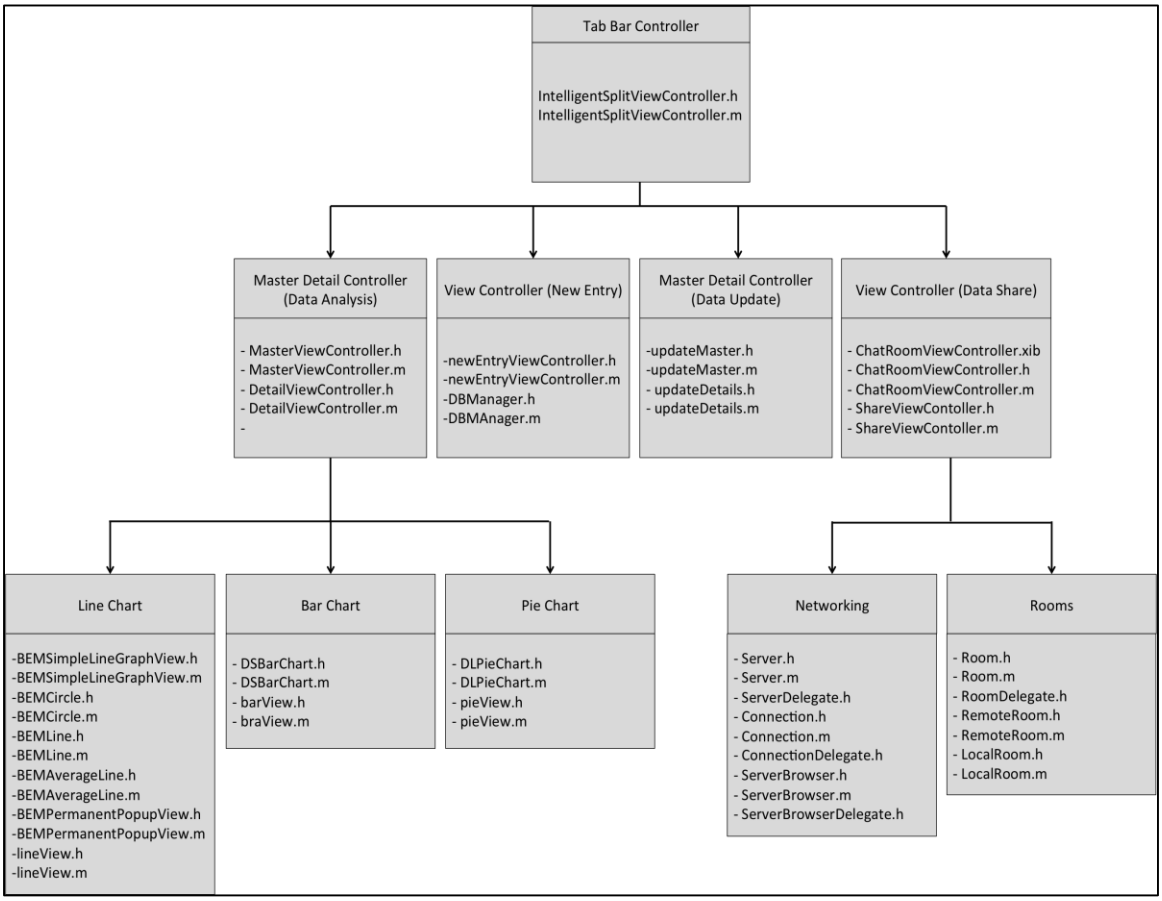


Figure 18. This is the class diagram of the iOS mobile application. It represents what