Smart-phone-based Platform for Patient Medical Compliance

A Major Qualifying Project Report:

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Bachelor of Science

By:

_____

Ryan Crook

_____

David Keeley-DeBonis

Date: 3/06/2014

Approved:

_____

Prof. Krishna Kumar Venkatasubramanian, Advisor

ABSTRACT

Patient medication and prescription adherence has been a widely recognized problem in the healthcare industry since doctors began prescribing medications to patients. In the past half century, several studies have been conducted on the subject, and many significant strides have been made in increasing patient compliance. With the advent of mobile technology comes an opportunity to further develop the methods used to confront the ongoing problem of medical compliance in patients and to revolutionize the way doctors can reach out to and keep track of their patients. Attempts have already been made to utilize mobile technology for this purpose; however, there exists a disconnect where the user is uncomfortable with using a mobile device. The aim of this project is to close that gap by designing a medication compliance application that is intuitive and easy to use, even for those individuals who find the concept of a smart phone outlandish and daunting, and is capable of passively accumulating data on patient medical compliance. As mobile technology becomes more prevalent, the increased availability of information becomes an asset that can be leveraged by medical researchers.

Table of Contents

Table of Figures

Table of Tables

1. Introduction

This section provides an outline of the objectives and overall goal, as well as an overview of the rest of this document.

1.1 Our Goals

The aim of this project is to provide an easy to use medical compliance smartphone application to individuals on a large scale. The deliverable for this project is a smartphone application. Many medical compliance smartphone applications already exist for both Android and iOS; however, due to the complicated nature of prescriptions and medical regimens, most of them require a lot of user input, screen changes, and other undesirable features. This makes them much more difficult to use, especially for elderly patients, who arguably need the reminders the most. Our objective is to create an application, which uses QR codes and other methods of gathering information to minimize user input. We also implemented a four tab system that allows users to easily move around the application, and minimizes the overall number of screen changes.

1.2 Document Overview

The rest of this document contains relevant background information, related work, and our methods used. Section 2 contains background information as it relates to different areas of the project. This section includes definitions of terms used and compliance related research done prior to designing the application. Section 3 outlines past work related to this project, namely, other medical compliance applications. It also outlines improvements on existing technology that are crucial in providing a solution to patient noncompliance. Section 4 contains the methods used, including software used, documentation and design strategies, database information, as well as thorough explanation of the choices made during the development phase. Section 5 outlines the overall architecture of the application and details both front and back end implementation. Section 6 outlines the criteria used to evaluate the success application. Section 7 includes the results of the evaluation with respect to the criteria outlined in section 6. Section 8 provides a conclusion of our work and describes areas of success as well as areas with room for improvement. Finally, section 9 proposes future work to be done in order to both add to and

improve on existing functionality. Potential solutions to the shortcomings mentioned in section 8 are also addressed here.

2. Background Information

2.1 Compliance Background

2.1.1 Medical Compliance Definitions

Medical Compliance has been assigned a myriad of slightly different definitions over the last five decades. For the remainder of this document we use the following definition for medical compliance:

> "[Medical] Compliance is defined as the extent to which a person's behavior… coincides with medical advice" [1]

Furthermore, terms that are related to compliance, such as adherence, acquiescence, or obedience are used interchangeably.

The term mHealth in the context of this document is used to refer to healthcare industry's use of mobile technology. This is a broad term and is not necessarily related to patient compliance.

2.1.2 Issues Facing Medical Compliance

Identifying the issues facing medical compliance is a relatively simple task. The nature of the problem is not complicated, in fact, the bare bones of the matter can be boiled down to one simple sentence: patients don't take the medication they are prescribed. The issue comes in finding a solution to this seemingly trivial problem. Many attempts have been made to create a model for solving the patient adherence problem, but the models developed cannot be widely applied. The root cause of this issue can be attributed to a number of factors, mostly revolving around the way these studies are conducted [1]. The scientific value of medical compliance studies conducted in the past has been extremely inconsistent. This issue stems from contrariety of criteria used to judge compliance and inconsistencies in the abilities of researchers to recognize and maintain control variables in these studies [1].

Difficulties in scientific studies of medical compliance also arise from the ways in which compliance is measured. There are two types of compliance metrics, direct and indirect. Direct measurements usually involve blood, urine and other tests, which measure the level of certain chemicals in the body. Indirect methods include patient interviews, pill counts and physicians impressions. Each of these methods has its advantages and disadvantages; however, none of

them are foolproof ways of determining medical compliance [1]. Direct methods are too expensive to be used widely in medical studies, and different patients react to medications in different ways, so chemical levels can vary widely, even across those patients who are compliant. Indirect methods have been shown to be inconsistent as well. Physician's impressions have been shown to be no more accurate than a roll of the dice in predicting medical compliance. Pill counts and patient interviews are slightly more accurate, but in one study, these two methods only agreed with each other sixty-eight percent of the time [1].

Methods used to measure compliance and non-compliance can be split into two groups. The first group is known as direct measurements. These methods boil down to a drug test which assess the amount of a given medication in a patient's bloodstream. These tests are expensive and often quite invasive, and as such, are not widely used. The second group is known as indirect measurements. This group is much more expansive than the direct variety. Indirect measures include pill counts, patient interviews, the use of electronic pill containers, and even physicians impressions of their patients [3]. None of these metrics have been shown to be foolproof. Indirect methods are so unreliable for a number of reasons. There is a lack of empiricism in these methods. Many of them are subjective and based purely on the opinion of an individual or group of individuals. This leads to inconsistencies in data, and damages the integrity of compliance studies. Mobile technology has not been used to great effect in this area as of yet. Needless to say, a metric by which medical compliance can be reliably determined is a necessity for the advancement of medical compliance research.

The issues facing medical compliance revolve primarily around the lack of reliable data. In an age where smart phones are prevalent and the connectivity attributed to the internet has become part of the norm, reliable or not, data is readily available. It is the goal of this project to address these issues and provide a reliable application that addresses the issue of medical compliance and seeks to improve compliance behavior in patients.

2.1.3 Past Approaches and Evolution of Methods Used

Past approaches to solving the medical compliance dilemma can be summarized by three basic schools of thought. They include educational strategies, behavioral strategies, and a somewhat ambiguous combination of the two [1]. Each has its advantages and disadvantages, but

all have been shown to improve medical compliance in a statistically significant way. Educational strategies have been shown to improve compliance drastically.

Patients who were subject to some type of one-on-one counseling in various past studies were between thirty and seventy percent more likely to be compliant [1]. However, as the length of studies increased, adherence of patients subjected to counseling fell off to levels, which were almost consistent with those who received no counseling [1]. Written information inserted into prescriptions was also shown to increase patient compliance slightly. Patients who received literature detailing the side effects and benefits of taking their medicine properly were slightly more likely to be compliant than those who did not. In one study, written and verbal education were combined to great effect, however the improvements in compliance did not persist past six weeks, and so would not be effective for long term treatments such as chemotherapy or ADHD medication. Certain adherence studies, conducted before 1980, also showed no statistical difference between patients who received counseling and those who did not [1]. Reasons for this disparity in results is up to speculation, but can almost certainly be attributed to one or more of the issues outlined in Section 2.1.2.

Behavioral interventions showed a less drastic increase in patient compliance, however the improvements which occurred proved to be much more consistent. Patient behavior can be adjusted at many points during the patient-doctor interaction timeline, and so behavioral methods for improving compliance are quite varied. Methods include reminders, specialized calendars and medication containers, refill reminders both written and verbal, and having patients record results, i.e. having epilepsy patients report the number of seizures they experience on a weekly basis [1]. Each of these methods has its strengths and weaknesses and many of them need to be tailored to a specific condition or patient; however all of them have been shown to increase compliance when used correctly.

The combination of these two schools of thought has proven to be the most effective way to improve medical compliance [1]. Combining these two methods is supported by many widely recognized and accepted compliance models, and as such has been adopted by many medical facilities [1]. Due to the nature of behavioral compliance improvement methods, these are the central focus of the project. Mobile technology as it has been integrated into the lives of so many people around the world provides a unique opportunity to have a seriously positive effect on the compliance behavior of these individuals.

2.1.4 Effect of Mobile Technology on Medical Compliance

The use of mobile technology (measured by the use of mobile data) doubled every year from 2007 to 2011, and it is estimated that this figure will increase by a factor of eighteen by 2016. Furthermore, it is estimated that by this time, there will be more than ten billion mobile devices in use worldwide [2]. With rapid expansion come many opportunities for a variety of industries. Mobile technology is specifically suited to assist with the medical adherence dilemma. People have their mobile devices with them throughout the day. This offers the potential to have some direct influence on the patient's life while they are away from the doctor's office. This influence can be leveraged in many ways, not the least of which is improving ways in which patients can be reminded to take and refill their medications.

Elderly patients are the largest problem demographic from a medical compliance standpoint. Fewer than fifty percent of elderly patients take their medication as it is prescribed to them [2]. Reasons for this have been subject to speculation in the past, but the simple fact is, elderly people are forgetful. Elderly patients often have more medications they are prescribed to take at various times throughout the day than the average patient. Increased complexity of a medicinal regimen has been shown to reduce compliance in all patients and this effect is compounded heavily when a patient becomes forgetful [1]. The advent of mobile technology provides an opportunity to improve compliance drastically in this circumstance. Providing elderly patients with personal, automatic reminders could have an extremely positive effect on a patients overall compliance.

Expectant mothers are another problem demographic for medical compliance [2]. As medical technology advances, more and more variables that affect prenatal health are being discovered. Along with the discoveries of these variables, medical regimens for expectant mothers have become increasingly complex. Automatic electronic reminders are the perfect solution in this situation, especially considering the fact that the majority of pregnant women are younger and thus, more tech savvy than elderly patients [2].

One of the indirect effects that mobile technology has on medical compliance is cost. Mailed reminders and one on one counseling to improve compliance are expensive and an inefficient ways of fighting medical noncompliance. Mobile technology allows patients to receive free, automatic reminders and offers a way to improve the efficiency and availability of one on one counseling sessions.

2.1.5 Current State of Medical Compliance

Over the last thirty years, many models attempting to predict medical compliance have been developed. In addition to predicting compliance, measures have been taken to persuade patients to take medications correctly and enhance compliance. Despite the efforts of the past three decades, the level of compliance on a widespread scale remains largely unchanged. The compliance problem is an ever present issue that faces medical professionals around the world.

Many medical journals have outlined the inadequacies of medical compliance research. The issues listed include lack of concrete guidelines for measuring compliance, lack of reliable data, and lack of consistency in metrics used to test compliance [3]. The fact is, compliance studies are invasive to patients and expensive to medical professionals. As such these studies are not commonly conducted. There is a need in the healthcare industry for a reliable, cheap and accurate way to measure and assess medical compliance.

The current state of medical compliance is directly related to the current state of the healthcare industry in the US as a whole. The United States healthcare industry has become an industry that accepts waste. It is estimated that upwards of twenty percent of healthcare costs are wasteful [4]. These wasteful costs include unnecessary treatments, administrative errors and even fraud. The majority of the waste associated with unnecessary treatments is due to the lack of medical compliance in patients. When patients do not take their medicine, especially when that medicine is preventative, they often become unhealthier than they would have otherwise. This leads to treatments that would not have been necessary if the patient had been compliant in the first place.

Current models by which medical compliance is assessed are based on theories in behavioral psychology decades old. The original model for medical compliance, and basis for many of the current models, is the Health Belief Model [1]. Developed by Becker and Maiman, this model incorporates general principles of motivation from behavioral psychology and modifies them to pertain to health and preventative medicine. Since the development of the Health Belief Model, other entirely new derivative models have been developed. These models are intended to explain variance in medical compliance and reduce and organize data. The Health Belief Model explains away about ten percent of variance, and while improvements have been made, a concrete model still does not exist [1].

2.1.6 Future of Medical Compliance

Despite the myriad of issues facing medical compliance and the relative stagnation of the issue in recent years, medical compliance has a very bright future. Legislation and medical reform, combined with the advent of mobile technology pave the way for drastic improvements in medical compliance [2]. Issues that need to be overcome include inconsistencies in measurements, lack of empirical metrics by which medical compliance can be assessed, underdeveloped models for explaining and predicting compliance, and a wasteful attitude towards health care as a whole. While these issues seem quite insurmountable, mobile technology, crowdsourcing, and increasing awareness towards waste are all possible solutions well within the reach of the medical industry in the years to come.

This project aims to leverage mobile technology to build a more stable means of collecting live data from patients that can be used to establish empirical metrics for medical compliance. In addition to being a platform useable by researchers, the application delivers a more dependable means of encouraging medical compliance in patients on an individual basis. Other applications exist to log and schedule patient information; however beyond basic scheduling these applications do not provide metrics that encourage patient compliance.

3. Related Work

There are several other medical compliance applications currently available on the market. While none of them have the capacity to collect compliance information and store it for public use, many of them still have relevant features, which should be considered for this project.

All of these medication compliance applications contain a basic set of features. All of them maintain a list of medications, allow the ability to add new medications, remind the user to take medications and get refills, and log medication history at a minimal level. Some of the applications have more notable features; these are discussed in the following paragraphs.

The first application with a unique feature is Dosecast. The most unique feature of the Dosecast medical compliance application is the way in which the action of a user taking a prescription is logged. Dosecast gives three options: take, postpone, and skip, where the others offer only two or one. This application allows the user to postpone when their medication is taken, instead of just skipping it. This allows for improved accuracy of compliance data reporting. Dosecast also allows for the storage of names and phone numbers for medical professionals [22]. Though it appears that not much is done with this professional information, Dosecast has taken one step closer towards synergy between doctor and patient.

Drugs.com is a more information based compliance application. It gives access to a database which lists drugs based on the condition the drugs are used to treat, identifies pills by shape, color, and imprint, and allows users to check their symptoms. The application is not designed to service the compliance needs of a patient on a daily basis but the application seeks to provide information that would reduce potential error where patients do not take the right medication. These types of educational tactics have been shown to increase compliance, and are considered in the final application for this project [23].

The MediSafe application also has a unique feature in that it maintains a database of photos of all of the user's medications. This way, when a notification is received, even the most forgetful of users should remember which medication they should be taking [24].

The above features have been considered in the final application design for this project. We believe that these existing features both improve medical compliance in patients and improve the accuracy of the compliance information that can be reported. While each of these current applications have a subset of the features we necessary for optimal medical compliance, they

were lacking in other regards. Not a single application provides both extensive information about medications to the user and allows extensive user flexibility while maintaining a complete record of all user compliance history.

4. Methods

4.1 Core Application Functionality

The goal is to develop an android smart phone application with the necessary resources to reduce the economic impact of non-adherence with respect to medication. The application must, at a minimum, keep track of all medication a patient is supposed to take at any given time; track medication compliance at all times and maintain an accessible history of all patient actions. The goal is to not only generate a high fidelity application that facilitates these needs, but also to create environment that users of diverse ages find easy and enjoyable to use.

4.2 High Fidelity Application Design

Before any actual implementation, a low fidelity prototype was developed. This prototype was used to capture and finalize the minimal functionality desired by the potential users. The following sections contain images taken directly from the prototype derived from the low fidelity mockups.

4.2.1 Logging in

A user needs able to sign-in or to gain access to the application while preventing others from accessing their personal information. For initial purposes, it is unnecessary for an individual to sign-in via a username and password, as they are not accessing their medical profile from more than one device. The application is one that would potentially be accessed over ten times a day, therefore a user must be able to obtain access to their information quickly while preventing other people from view that information from the users device. A simple access pin is quick for the user to enter and prevents unwanted access to a moderate degree. The very first time that the user accesses the application, they are asked to enter some personal information. This personal information is basic biometric information like name, birthdate, gender and weight.

Although this information does not have any impact on the user's interaction with the application, it is an important supplement to the medical data that is being collected from users. The user is also asked to select a pin. A simple 6-digit pin is used to prevent other people from accessing the medical information from the user's phone but still be simple and convenient enough for the user to use. The android activity used to collect this information is only be seen

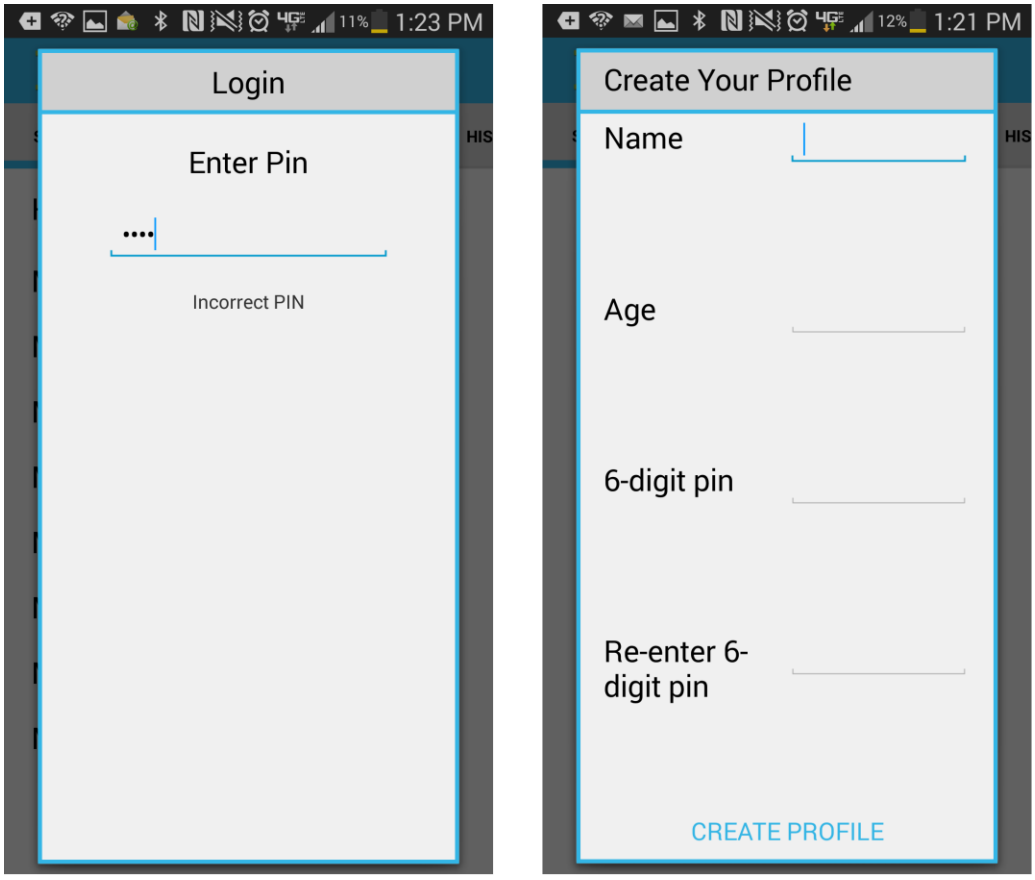

Figure 1: The profile creation and login screen mockups

by the user the first time that they launch the application. Once a user profile has been created, the user is only asked to enter the 6-digit pin when launching the application. If the user leaves the application open on their device for an extended period of time, it times out and they have to enter the pin again to gain access to their profile. Figure 1 shows the profile creation screen and pin access screen.

4.2.2 Medication Schedule

The application itself utilizes three primary tabs, Schedule, Medication, and Buddies. When launching the application, the displayed tab is the Schedule tab. Figure 2 shows this tab. The Schedule supports the functionality for the user to:

1. Quickly view their medical schedule for the day based on time.
2. Comply with a scheduled event. A particular event is of one of four event types: Done, Missed, Complete, or Pending. A user is not able to interact with an event that is considered Done. A user may interact with an event that is either Missed or Complete.

3. Call a Pharmacy or a Doctor
4. Access their Compliance Score
5. View their Compliance History

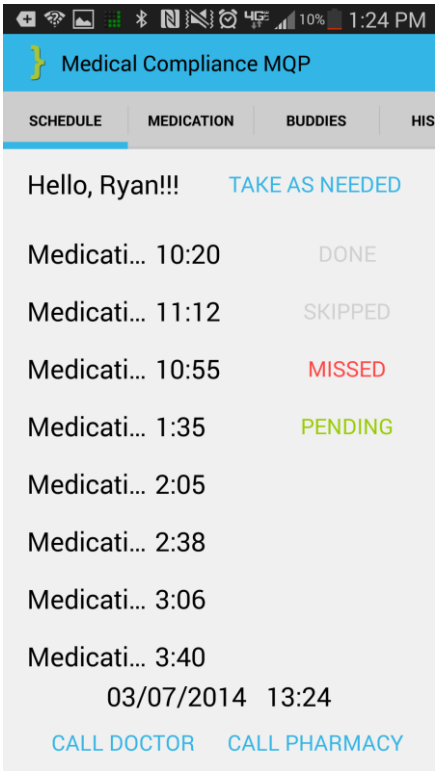

Figure 2: Daily Schedule screen mockup

The schedule screen contains the features that are most used. Other features are important but are not used on a daily basis. Once the user has properly configured the application to their liking and added their medications to the application, frequent daily interaction primarily originates from the Schedule screen.

For numerous reasons, the user may want to contact a pharmacy or their doctor while using the application. The ability to quickly call a doctor or pharmacy is often desirable. When clicking on either of the buttons, a list of all added doctors or pharmacies is displayed. This list also contains the option to add a pharmacy or doctor respectively. The user may add a doctor of pharmacy or doctor by name and phone number or import from their contacts. A third quick call button to 911 may also be considered. Figure 3 demonstrates these features. In future iterations, the list of pharmacies may also contain the two nearest to the user's current location regardless of whether or not the user has added them.

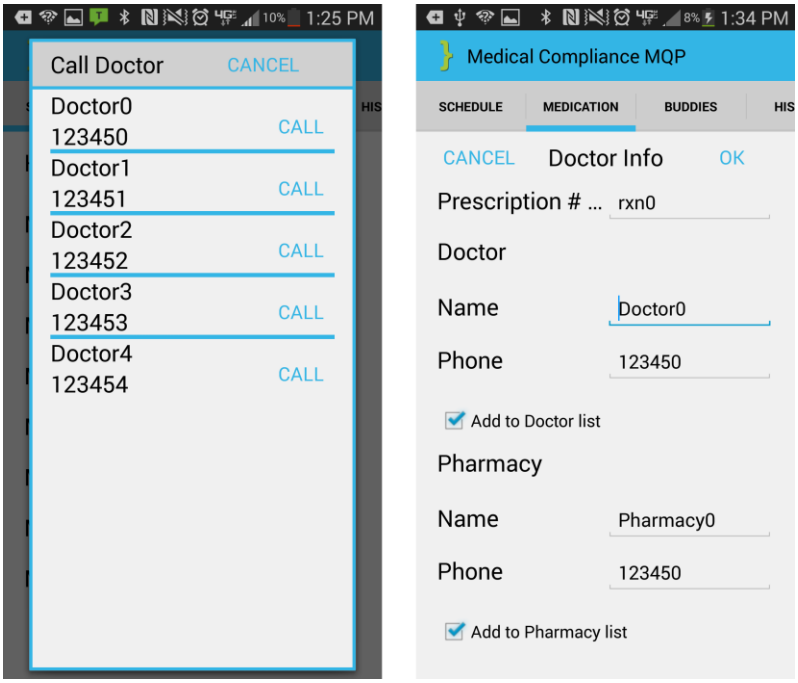

Figure 3: Call and add doctor/pharmacy screen mockups

4.2.3 Medication Compliance

The main feature of the schedule tab, previously shown by Figure 2 is the medication schedule component. This is a schedule of all medications that the user needs to take for the current day. Entries are arranged by time from earliest to latest. Each entry in the list contains the time the user is scheduled to take the medication, the name of the medication and a button reflecting one of four possible states for a compliance event. Each entry may either be Done, Missed, Pending or Future. Done implies that user has already completed the event. It is still listed in the daily schedule but they may not interact with it. Missed means that the compliance window has already passed without the user taking any action. A pending event is one that is within the compliance window and awaiting user interaction. The actions that a user may take for a Missed and Pending are the same; however it is important to separate the two scenarios. The fourth event type is the Future event. An event processing this status is one that has yet to enter the compliance window in which user interaction is permitted. A user may not interact with an event that is still considered a future event.

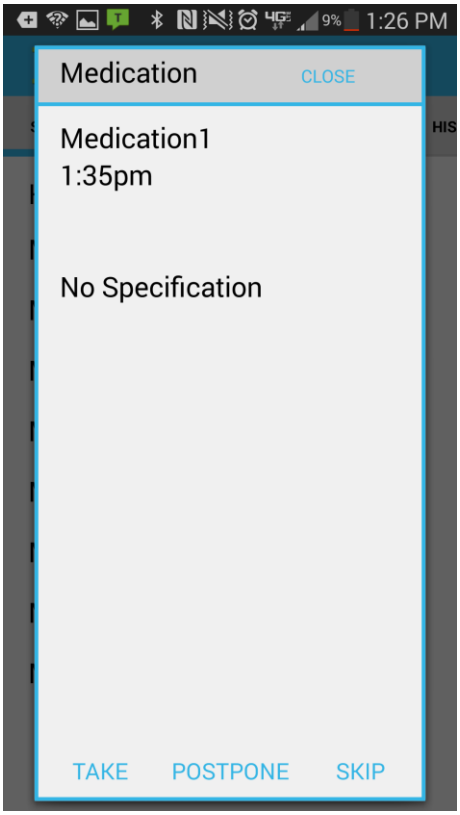

Figure 4: Complete event screen mockup

When a user interacts with an event that is either Missed or Pending, the following information is displayed to the user:

1. Medication Name
2. Medication Picture
3. Dosage
4. Instructions in the form of informational snippets of text that are supplied by the user when adding a medication to their medication list. Examples of possible instructions are "take with food" or "don't take with alcohol".

Figure 4 shows the window shown to a user when interacting with a Pending event. The window for a Missed event is the same; however the Postpone feature is disabled. A user has the option to Take, Postpone or Skip a scheduled medication.

Selecting the Take option signifies that the user has taken the scheduled medication. Every medication has a compliance window or time within which the user is allowed to take the medication before it switches from a Pending event to a Missed event. This window defaults to within thirty minutes of the scheduled time. Take medications outside of the

window can be dangerous. The event status changes do not affect the ability for a user to take a medication but allow for the user to make a more informed decision. When taking a medication, the user is given the option to enter any personal events that they may want logged in their history. After confirming that the medication has been taken, the compliance event switches from Pending or Missed to Done and the user may no longer interact with it. Figure 4 shows the display present to the user when taking a medication.

When postponing a medication, the user is presented with an opportunity to enter notes. The user must also select a preset time or enter a time for which to postpone the event. Postponing moves the entire compliance event and the window within which the user may take action. The user should not be able to push the compliance window too far because doing so may interfere with the next scheduled compliance event for the medication. The actual postpone limit should most likely vary based on the medication.

The Skip option signifies that the user is consciously deciding that they are not taking the medication. If a user decides that they are not taking a medication, it is an important part of their compliance history. The Skip feature allows them to record this action. When a user skips a scheduled medication, the event is marked as Done in the schedule, not missed. The Missed Event type corresponds to the user not taking any action during the compliance window. The act of Skipping is the user consciously deciding that they are not going to take the medication.

4.2.4 Medication System

The second of three tabs is the Medication tab. While the Schedule tab contained a schedule for the current day, the Medication Tab is an alphabetical list of all medications, vitamins or anything that the user has entered into the application. The user uses this screen to manage their current medications and record important information. The current proposed features include the ability to edit an existing item, remove an existing item, add a new item, and record a blood glucose reading or blood pressure reading. Figure 5 demonstrates the Medication Tab.

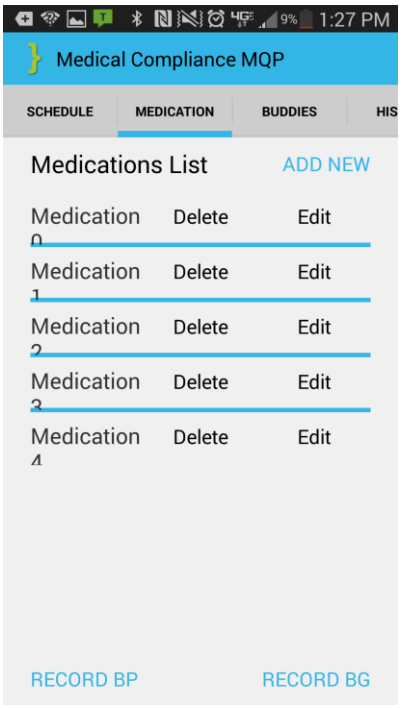

Figure 5: Medications list screen mockup

For individuals with heart disease, diabetes and many other conditions, taking blood pressure and blood glucose reading are a daily occurrence. The ability to record this information allows the user to keep a log of important information relating to their condition in concurrence with their medication logs. Though the initial proposed featured readings in the application are related to blood pressure and blood glucose, others may be added. Android Applications already exist to measure and take reading for things like Blood Pressure. This version of the application does not offer a means to take the reading; it merely allows the user to record the readings that they have taken through some other means. In addition to this, the user may set critical limits for things like BP and BG. A reading that is not within the bounds of their critical limits should, in the future, prompt for a certain medication that the user may have flagged as "As needed" to be taken i.e. insulin.

For a user to be able to adequately keep a history of important information with respect to their blood pressure, the application should allow for the user to record both the Systolic and Diastolic number of the reading as well as their pulse rate. The critical limits for both Systolic at 180+ mm Hg and Diastolic 110+ mm Hg are preset and are not able to be changed by users. These preset critical limits are indications of a hypertensive crisis and are determined by the American Heart Association. Figure 6a reflects this feature.

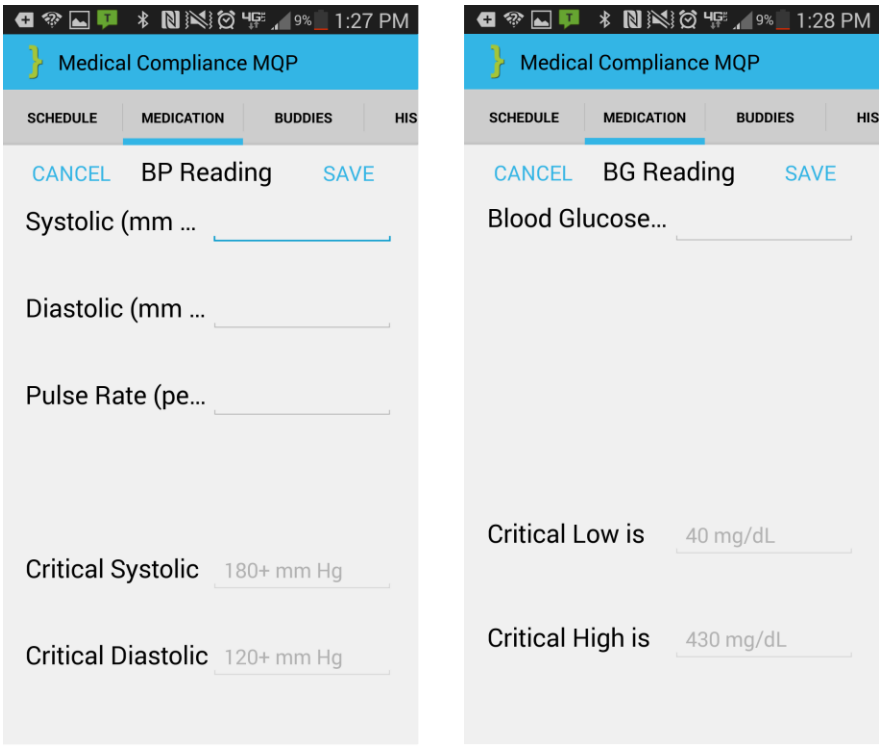

Figure 6: (a) Blood pressure screen mockup (b) Blood glucose screen mockup

A blood glucose reading consists of a single reading in terms of mg/dL. There exists both a critical high reading of 430 mg/dL and a critical low reading of 40 mg/dL. These two numbers are pulled from the critical indication ranges of hyperglycemia and hypoglycemia as defined by the American Diabetes Association. As with the blood pressure critical limits, users are not allowed to change these values. As shown by Figure 6b, the user also has the ability to record when they took the reading. For diabetics, recording information as to when the reading was taken is also important and it is common to record blood glucose levels before and after eating.

4.2.5 Adding a Medication

From the Medication Tab, a user may add a medication. This may be done in one of two ways. They can manually add the information or they may scan a QR code for their medication. As of now, no common standard exists for the plain text format of a prescriptions QR code. As such, it is very likely that adequate information cannot be added by the QR code feature. The user must manually fill in any information not supplied by the QR code.

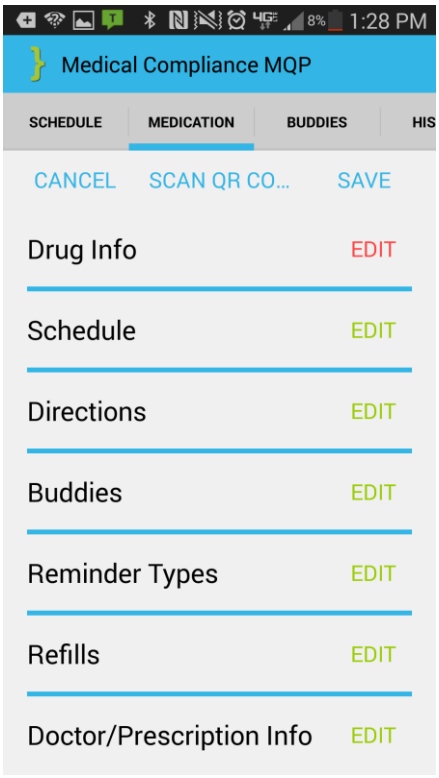

Figure 7: Add medication before drug info is filled in

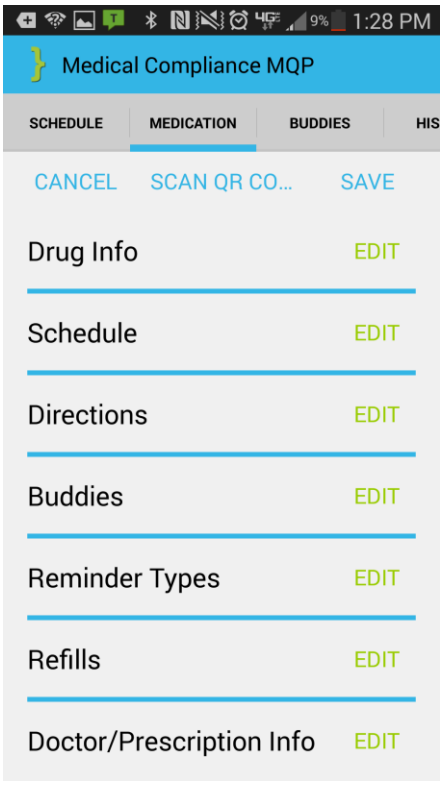

Figure 8: Add medication screen mockup

The Add medication screen as shown Figure 7 has seven sub categories of required information. These seven categories are Drug Info, Schedule, Directions, Buddies, Reminder Types, Refills and Doctor/Prescription Info. A user must populate all seven categories before the medication can be added. Information is not required for every part of each subsection but all core information must be supplied. As in Figure 7, all incomplete sections are red. A section becomes green after filling in the minimum required information, as demonstrated by Figure 8. A user may revisit any section at any time. Once all seven categories have been populated to the minimum requirements, the user may add the medication. Adding a medication updates both the Schedule and Medication tabs as necessary. The process of adding a new medication may be canceled at any time. When editing an already existing medication, the user is presented with the exact same set of screens that they would see if they were adding a new medication. The subsections are populated with the previously supplied information. Saving the medication updates the existing instance. When canceling an edit, the medication reverts to its previous state. Figure 8 shows all seven sections after being adequately populated by the user when they save the drug.

Figure 7: QR code scanner screen mockup

Electing to scan a QR code launches a QR code scanner application called ZXing. This is shown by Figure 9. If the user does not have this application installed, the google play store is launched and the user is prompted to download the barcode scanner. The QR code scanner option does not become available to the user until they have downloaded the application. Once ZXing has been downloaded, the user is not asked to download it again.

Each section in the Add Medication screen corresponds to a different screen and a different set of information. When a user clicks on a section, a new screen is displayed. Figure 10a shows the information required by Drug Info section. This screen allows the user to enter

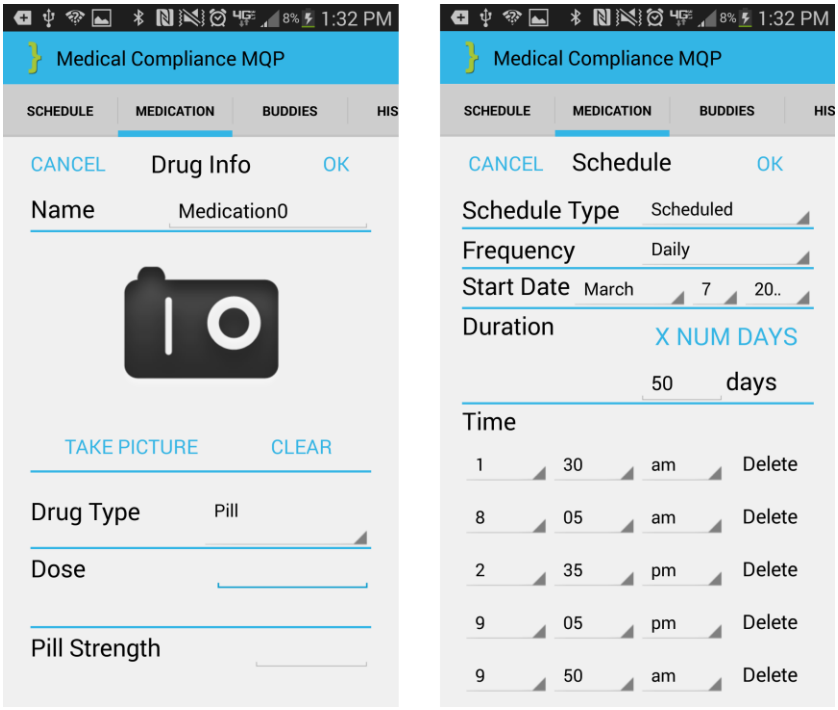

Figure 8(a) drug information screen mockup (b) drug schedule screen mockup

the name of the medication, supply a picture, and specify the drug type, dose and pill strength. The only piece of information that is absolutely required is the name of the medication. The user is encouraged to take a picture of the medication that they will be taking. Electing to take a picture launches the phone's camera application and the picture taken by the user is imported into the medication information.

The Schedule section is where the user configures their desired schedule for the medication. The user may select from one of two schedule types, scheduled or take as needed. The ability to specify a medication as "take as needed" allows and encourages people to add medications that they have and may not be taking on a scheduled basis but still want to be able to keep a record of. If "take as needed" is selected, the user is not required to enter any other information in this section. If it is a "scheduled" medication the user is required to populate the remaining fields. The resulting schedule is determined by four factors; Frequency, Starting date, Duration, and Time. Frequency refers to how often the medication is taken in terms of days. The three values the user may enter are Daily, Weekly, or Every X Days. If choosing daily, this indicates that the user plans on taking the medication once or more each day. If choosing weekly, the user is asked to select which days of the week that they are taking the medication on. This allows for abnormal schedules in which a medication is supposed to be taken two days per week. Some medications are to be taken every x number of days. The final option is used to create schedules for those medications. After selecting a schedule type the user must enter a start date and duration. The start date corresponds to the first day that the user will be taking the medication. The duration must be either continuous or some number in terms of days. In the future it may be advantageous also allow users to determine a duration based on number of taken doses. The last piece of required information is the Time section. The user may enter up to three times a day to take the medication. If a medication is to be taken only once per scheduled day, then only one time is necessary. The functionality of this section is displayed by Figure 10b.

The next section is the Directions page. In addition to specific times, many medications have specific instructions as to how to take them. This page provides an area for the user to enter that information. The most common specifications are whether a medication should be taken or not taken with food and not taken with alcohol. A blank area is also provided for the user to enter other personal instructions that they may have. This section defaults to "No Specification"

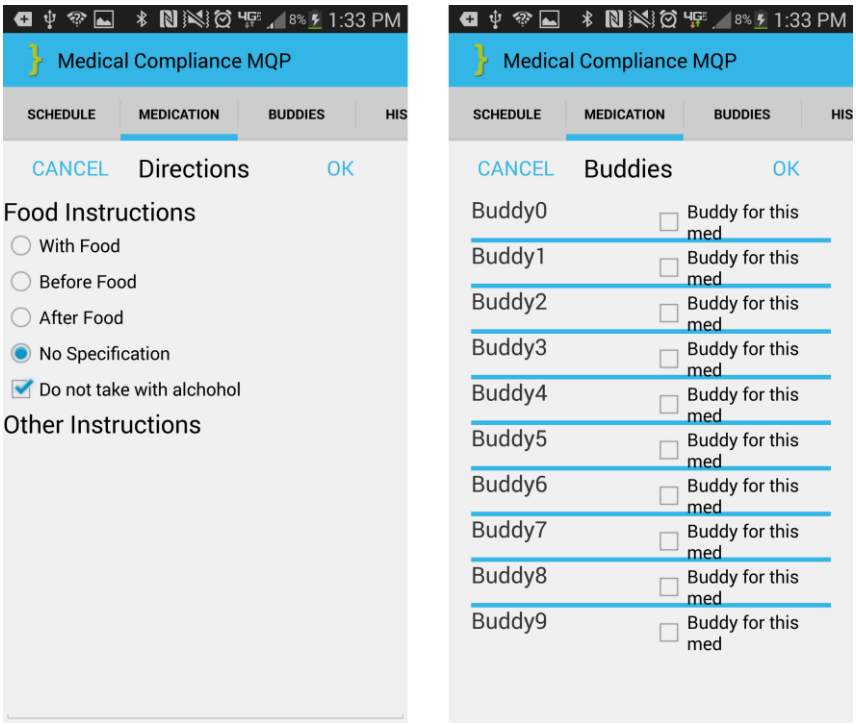

Figure 9: (a) drug directions screen mockup (b) buddies screen mockup

and "Do Not Take With Alcohol". The user is not required to make any changes or supply any additional information on this section. Figure 11a shows this section.

The next section is the Buddies section. This is part of the unique buddies system used to encourage medical compliance and reduce the cost associated with noncompliance caused medical concerns. The buddies system is meant to allow a user to add individuals to each medication who will act as their safeguard. As shown by Figure 11b, buddies may be added from either a preexisting buddy list or from the contact list of the user's phone. Buddies are important because the user can elect to have a SMS message sent to the buddies associated with a medication in the event that a scheduled medication is not taken. This a feature that will not be desired for every medication, however in the case of life threatening diseases, complications associated with noncompliance could be significantly decreased. Once buddies have been added, configuration of these buddy notification options are available through the Reminders Section, shown by Figure 12a.

The Reminders Section is a simple set of toggleable options. As of now, there are three toggle options:Android Alert Notifications, Self-texts, Buddy texts. The first two options are toggled on by default and the final one is turned off. If Android Alert Notifications are turned on, the user receives notifications in their phone's notification tray whenever they are scheduled to

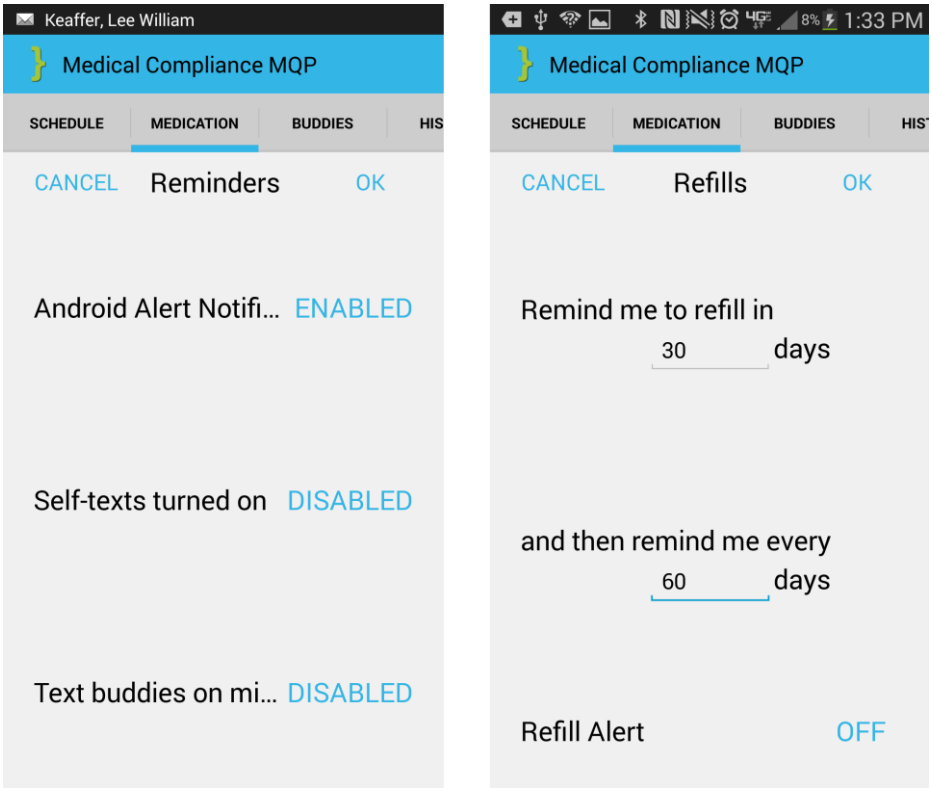

Figure 10: Reminders screen mockup (b) refill screen mockup

take a medication. The Self-text feature sends the user a text if they have missed a compliance event. The third Text Buddies option sends a text to all buddies of the medication if the user has not taken the medication during the compliance window. This feature is not necessary for most medications.

The sixth section is the Refill section. Similarly to medication alerts, a user may schedule refill alerts. The alerts are scheduled by entering the number of days until the next refill and then the number of days between each refill after that. This option is turned off by default because alerts may be unnecessary or undesired. . The user may toggle this feature on or off at any point as shown by Figure 12b.

The final section is the Doctor Info Section. Although the user is not required to populate any of

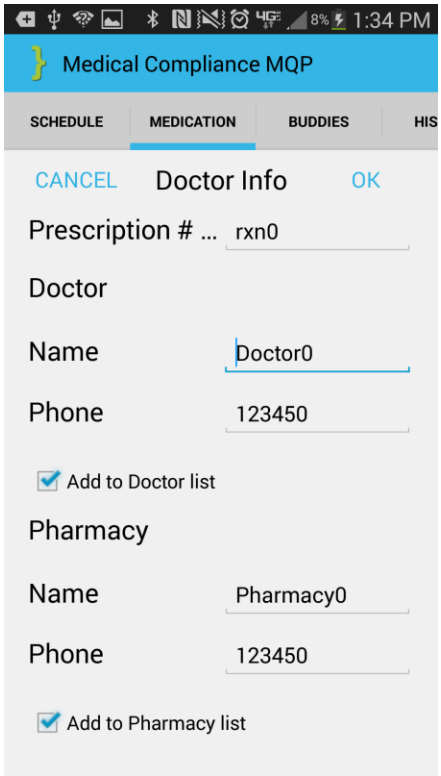

Figure 11: Doctor Information screen mockup

these fields, the user is encouraged to supply the Prescription RxN, Prescribing doctor contact information and the contact information of the pharmacy where the prescription was filled. Should the user choose to supply this information, the doctor and pharmacy contact list is updated as needed and the user now has the ability to quickly call this doctor or pharmacy from the front page. Figure 13 demonstrates this feature. Upon populating this section, the user has successfully supplied the required information for every section and the medication can be added.

4.2.6 Buddy System

The third tab on the main application screen is the Buddies tab. The buddy system is a means of encouraging compliance. As previously discussed, users may add Buddies to each medication who will receive a text in when the user misses a compliance event. The collective buddy feature encourages people to look out for each other and increases compliance rates with

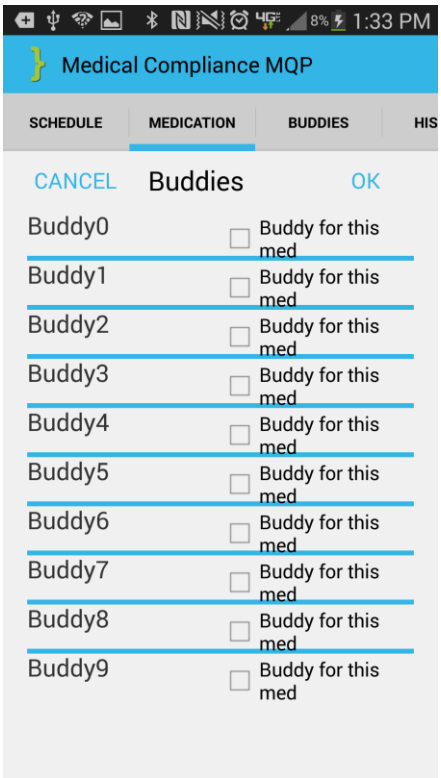

Figure 12: Add buddy screen mockup

crucial medications. This tab serves as a means of organizing and adding and removing buddies across all medications. The user is also able to browse through a list of all buddies that have

been added in alphabetical order. Adding buddies to the list is done by selecting them from the users existing contact list. The Buddies tab is shown by Figure 14.

4.2.7 Compliance History

Figures 15a and 15b illustrate the Compliance History feature. This feature allows the user to view a log of their respective compliance history. The list is originally sorted by time from most recent event to oldest. The user may also sort the list by medication. From this screen, the user also has access to their blood pressure and their blood glucose reading history, displayed in a similar fashion. A user may also email their compliance history to any desired address. They also have the ability to set up an automatic email list. Individuals added to this list will receive scheduled emails containing a copy of their medical compliance history. This

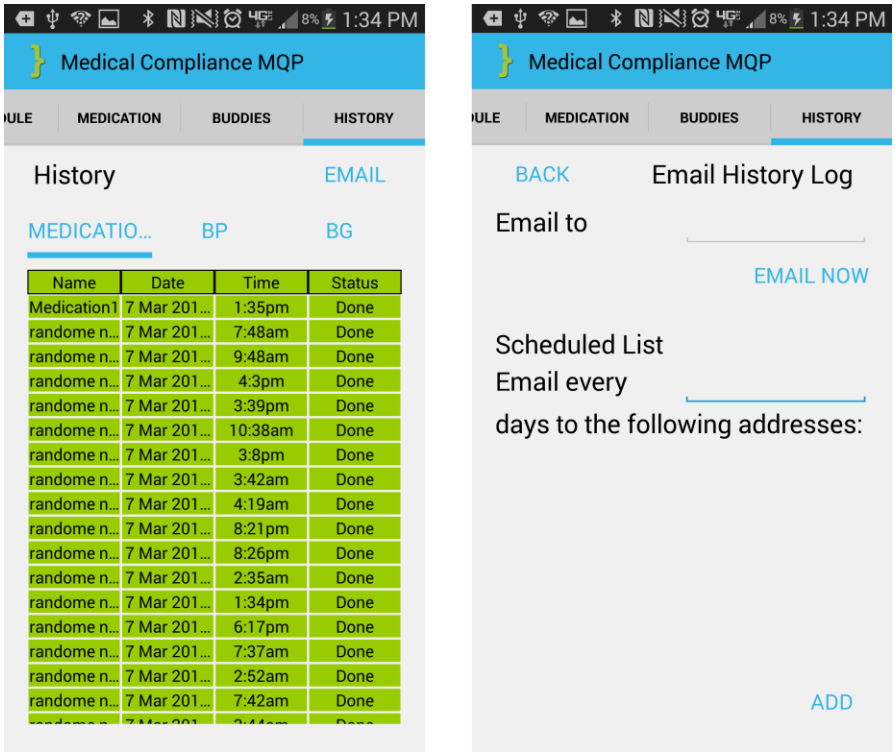

Figure 13: (a) History screen mockup (b) email history screen mockup

feature allows users to periodically back up a copy of their records and or email them to their doctors without having to manually send it. The user may specify the frequency by which their history is sent.

5. Application Architecture

5.1 Application Design Paradigm

The goal in proper object oriented design has always been to maximize the separation between the UI and business layers. One of the most common design patterns that adheres to these OO principles is the Model View Controller or MVC pattern. This focuses on abstracting the user interface responsible for rendering the content, referred to as the view from the model, the component responsible for maintaining the state of the application. The controller component serves as the messenger between the view and model. This component interprets events and interactions made between the user and the UI and updates the model accordingly. The model component then updates the view as necessary. An instance of a controller is rarely maintained throughout the entire lifecycle of the application and only handles a single user interaction. A new controller is often created every time the user repeats the interaction. Figure 16 is a UML class diagram of an example implementation of the Model View Controller paradigm [6]. As shown, an event is fired by the view after some user interaction. Then when the controller receives this event, the controller makes the necessary

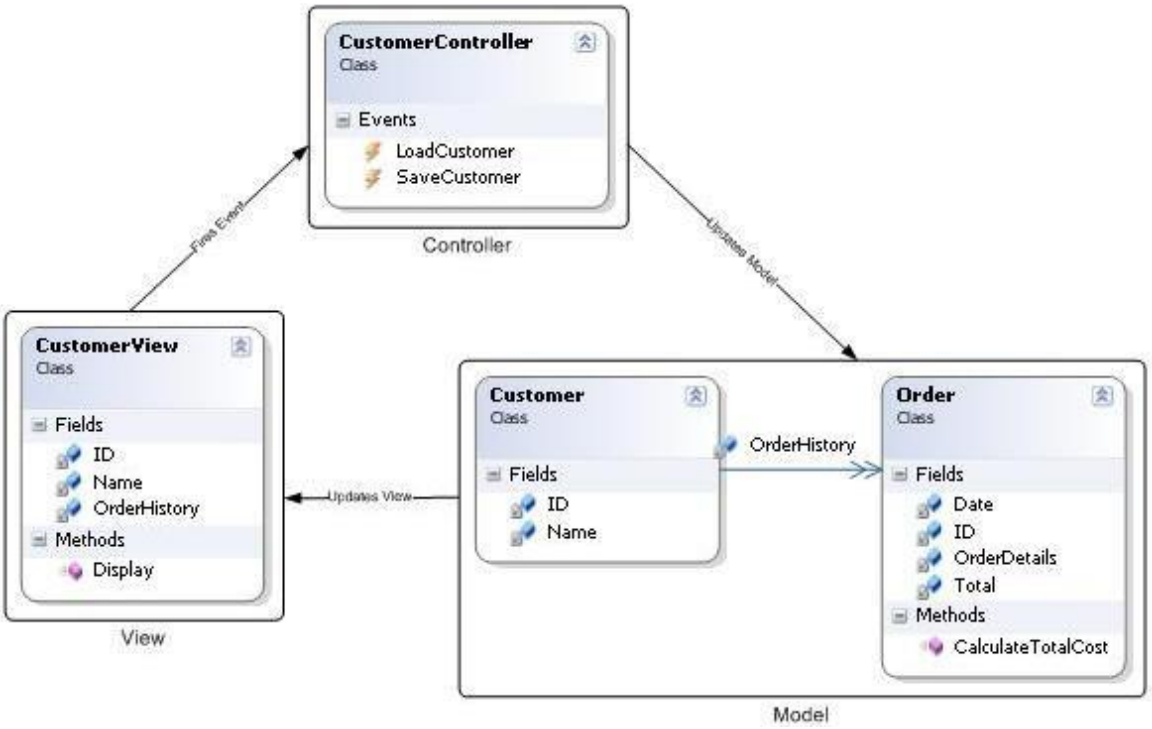

Figure 14: Model View Controller Example [6]

method calls to the model. The model's state is updated and then the model updates the view, thus completing the sequence of actions associated with the single event.

The level of abstraction between the view and model that the MVC pattern achieves is not optimal. In this particular instance, the view is responsible for creating the instance of the controller. In order for the controller carry out its sequence of actions, it must possess a reference to both the view and the model. This means that the view must also possess a reference to the model, in order to pass it to the controller that it creates. After the model has been updated, the model directly updates the view. In order for this to happen, the model must then possess a reverence to the view and this is what makes the model view controller design paradigm less ideal for mobile application development.

The MVC design paradigm is lacking because it is hard to test both the model and view independently without creating complex mock objects. Native android application development is done primarily in java. Unit testing of the application can still be achieved through the use of tools like JUnit. The standard JUnit framework cannot make calls to the Android API [8]. If proper abstraction has been achieved, only the view classes should contain calls to the Android API. In the MVC pattern, the model is encapsulated and abstracted from the view but not decoupled. The process of abstraction refers to the "separation of ideas from the specific instances of those ideas at work" [7]. The combination of this with proper encapsulation, the process of hiding the state of an object and restricting access to it, are two goals in all object

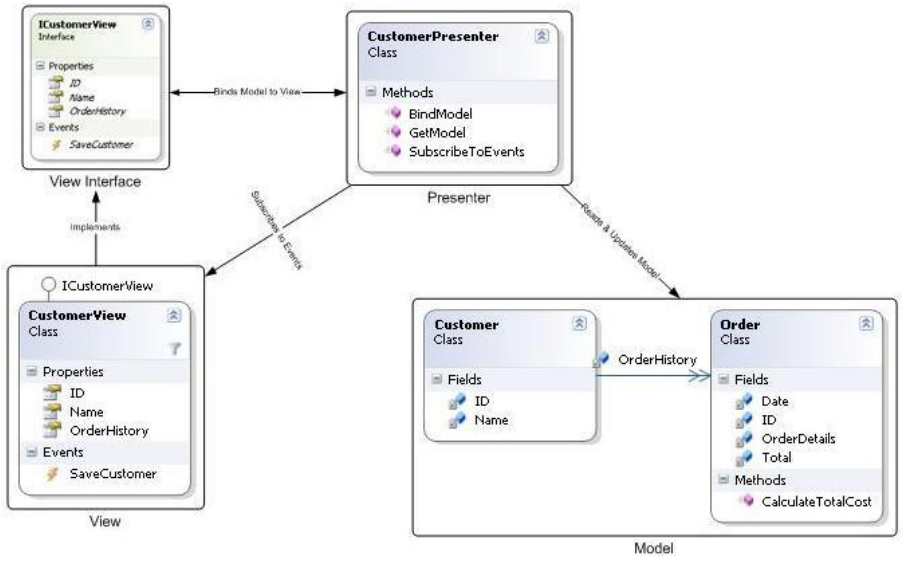

Figure 15: Model View Presenter Example [6]

oriented programming paradigms. Decoupling is a way of ensuring that two different software components are not tightly dependent on one another, in this case, the model and view [9]. If decoupling can be achieved, the model can be fully tested independently of the view. JUnit extensions exist to allow for the testing of Android API calls. These frameworks could then be used to test the view classes separately.

This application was designed using the Model View Presenter design pattern. The MVP design pattern is based on the concepts of the previously discussed MVC pattern; however, this pattern separates responsibilities across four components to allow for a far looser coupling between the model and the view. The four components are the view, the view interface, the presenter and the model. As with the MVC pattern, the view refers to the set of classes that facilitate user interaction and have the responsibility of rendering the user interface. The model is still the encapsulated business logic of the application. The presenter is responsible for interaction between the view and the model. The view interface is used to loosely couple the presenter from its view [6]. Figure 17 shows the MVP implementation of the exact same set of classes shown in the MVC representation shown by Figure 16.

Each view typically has a single corresponding presenter. The view does not maintain an instance of the model. Access to the state of the application is a responsibility left solely to the presenter. The presenter is linked to the view when the view is created. When an event is triggered on the view, the presenter retrieves the model and updates the state of the retrieved instance. The presenter updates the view via calls made to the view interface. The presenter never directly interacts with the view itself. The model and view are entirely decoupled from

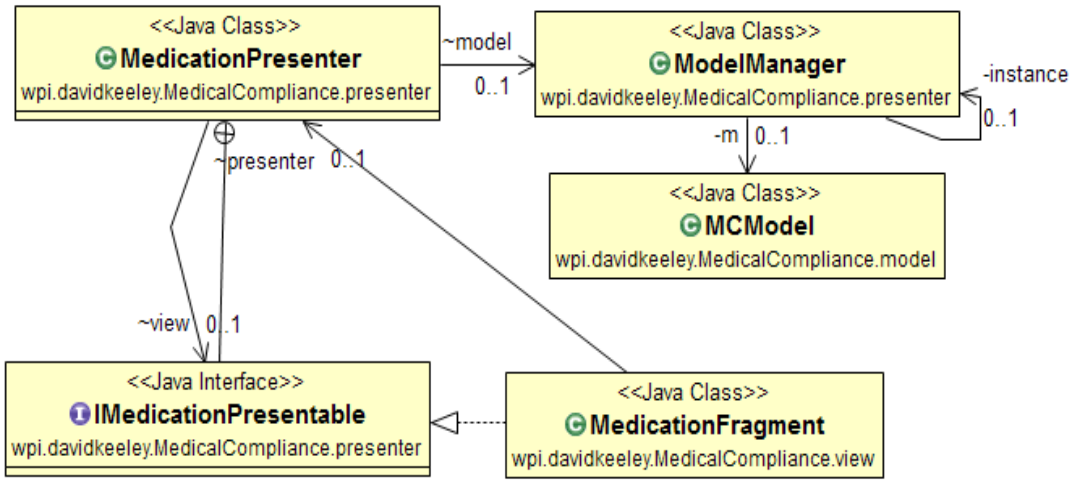

Figure 16: Example of our MVP implementation

each other because the presenter serves as an intermediary allowing the view and model to evolve independently of each other [6]. This is far easier to unit test. The intermediary nature of the presenter also makes it easier to maintain a persistent model between multiple views and from execution to subsequent execution. An additional data access layer can then be added to the presenter and remain entirely abstracted from the model or view.

The previous MVC and MVP pattern examples shown in Figure 16 and Figure 17 are entirely independent of the medical compliance application. The medical compliance application itself has many views. Each view has at least one corresponding presenter that interact with a single model. Figure 18 is a direct UML representation of one of the MVP implementation in the application. This particular view is the called the `MedicationFragment`. Its view interface is `IMedicationPresentable`. The single model used by all views is `MCModel`. The intermediary presenter for this particular view and view interface is called `MedicationPresenter`. This class diagram contains a fifth class called `ModelManager`. `ModelManager` is the aforementioned data access layer that can be added to each presenter. It is important that every presenter used by the application is maintaining the exact same state since this application contains many views. The `ModelManager` loads a singleton instance of the model from memory so that every presenter is updating a single instance of the state.

A singleton in its simplest form is a design pattern that only allows for a single instance of a class to be instantiated [11]. In this particular case, The `ModelManager` loads an instance of the model from memory and then returns a singleton instance of that `MCModel`. The loading and saving of the state of the medical compliance application is handled by the main activity of the android application.

5.2 Activity and Fragment lifecycles

An activity is a one of the core components of an android application. Every application consists of at least one activity. Each activity provides a window in which a user interface can be drawn. Applications with multiple views or screens that the user may interact with typically

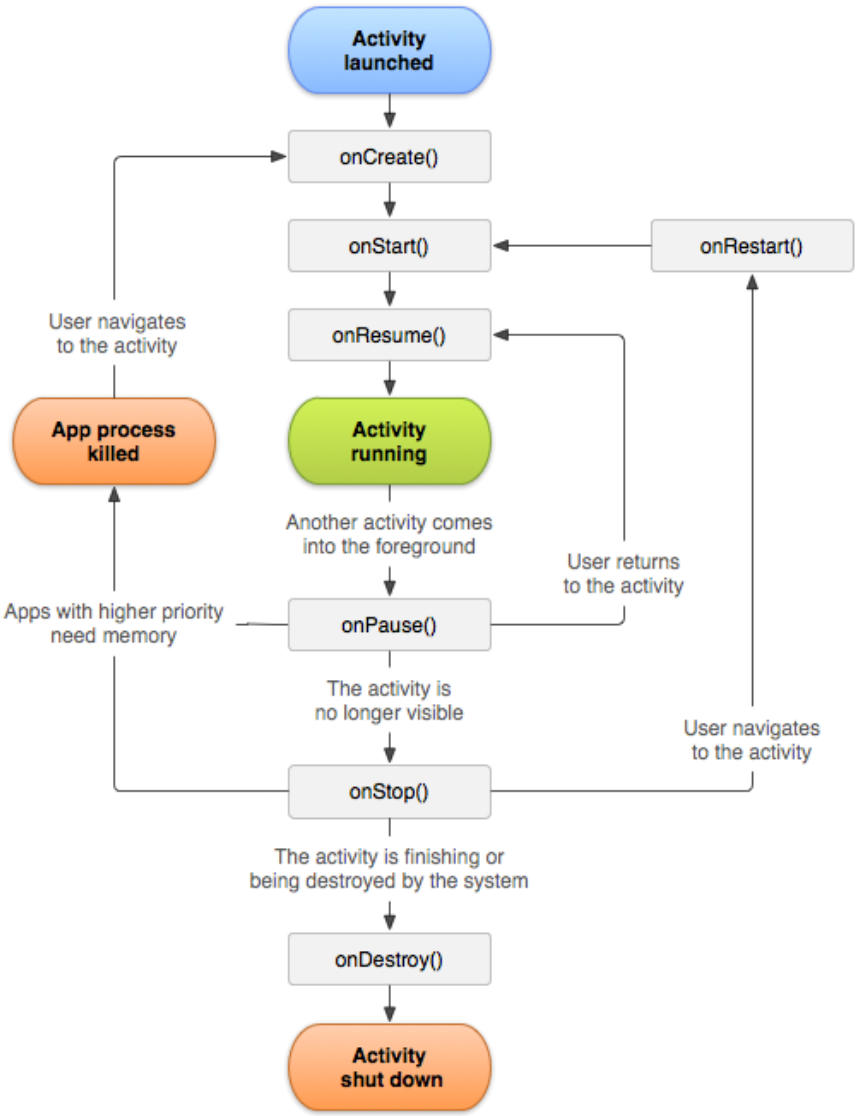

Figure 17: Android activity lifecycle [10]

have one or more activities. The core activity is often designated the main activity. Activities may create and start other activities. Every time an activity is started, the previous activity stops [10]. This means that only one activity may be visible at a time and that the user may only interact with one activity at a time. The system preserves a record of activities and the user may return to a previous activity via the back button. When a user opts to return to a previous activity, the currently existing one is destroyed. When a new activity is created, the state of the application must be passed to it or loaded from memory [10]. Using an activity for each screen in an application that has many screens becomes very resource intensive. The state must be repeatedly saved and loaded or bundled and passed. For this reason, the medical compliance application possesses only a single "main" activity. Using a single activity removes the

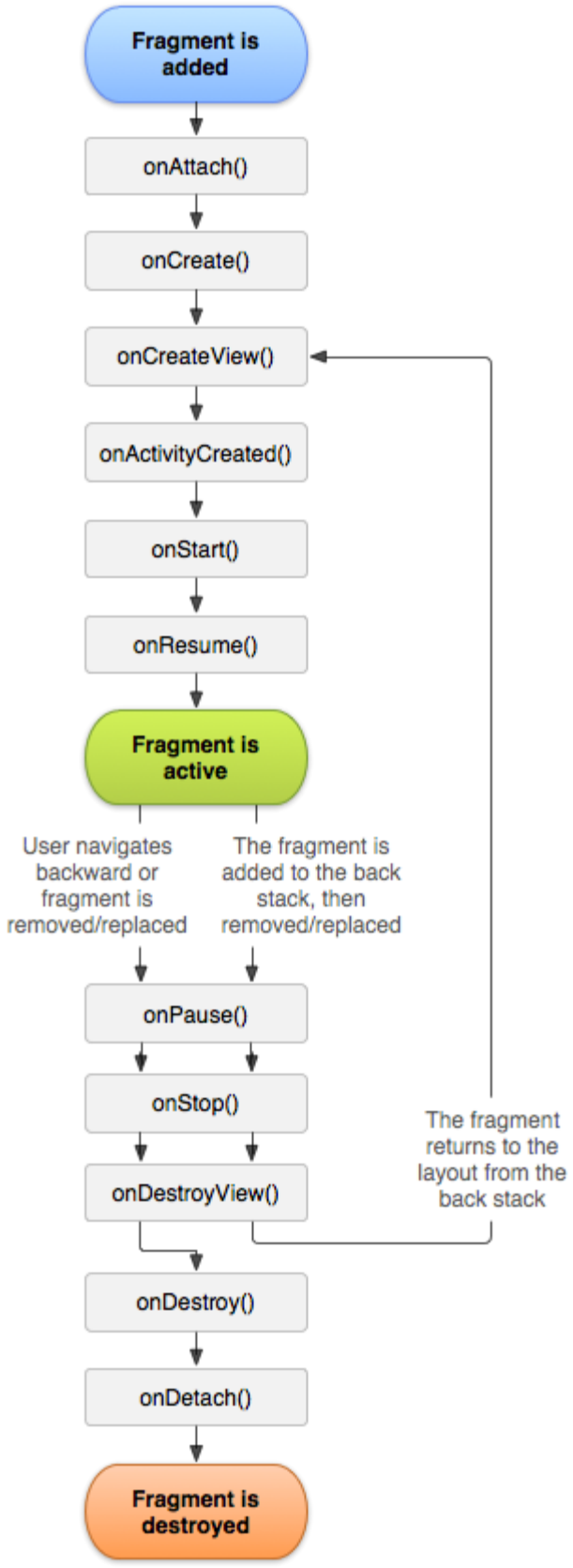

Figure 18: Android Fragment Lifecycle [11]

overhead associated with constantly creating and destroying activities, however the need to display many screens within the application still exists. An activity may exist in one of three states, Resumed, Paused, Stopped [10]. Maintaining the application throughout its entire runtime is knows as managing the activity lifecycle. Figure 19 shows the events that affect the state of the activity and the method calls that are made as the activity changes state. The medical compliance application primarily uses the `onCreate()` on Resume() and `onPause()` methods to maintain its state. `onCreate()` is called when the application is initially launched. The model is loaded during this time. When an activity is paused or stopped, the system may drop the activity from memory by killing its process. The activity must be completely restarted and restored to its previous state. For this reason, every time `onResume()` is called, the model is reloaded from memory. Any changes to the model are saved in memory when `onPause()` is called.

To provide the many screens required by the application, dynamic fragments are used. A fragment is

individual element of the user interface and is always running within the activity it is associated with. More than one fragment may exist within a single activity at the same time. Each View used by the medical compliance application is a separate fragment. Each fragment has and must maintain its own lifecycle. That life cycle is dependent on the state of the lifecycle of that activity that it exists within. When an activity is paused, all fragments with in the activity are paused. Figure 20 shows the events that affect the state of the fragment and the method calls that are made as either the activity or fragment changes state [11]. When a user of the medical compliance application changes screens through some interaction, a new fragment representing the new view is created and substituted for the existing one. When the view is created, a corresponding presenter is also created for that view. The visibility of fragments can be manipulated from the parent activity and may be dynamically added and removed. These actions are achieved through the use of a fragment transaction. A fragment transaction provides the functionality to change the visibility of fragments and manipulate the lifecycles of all activities. When a transaction occurs and an activity is removed, it is placed in what is called a back stack. The back stack preserves a record of all removed fragments in the order in which they were removed. This archive allows users to seamlessly traverse views and return to previous views in the reverse order that those views were visited with the state still preserved [12].

5.3 Additional Application Resources

BroadcastReceiver

The android `BroadcastReceiver` class is used by a number of applications to handle broadcast messages sent by the android operating system. The android operating system handles all alarms using these system broadcasts. When an alarm is set, the alarm manager passes an `Intent`, wrapped inside of a `PendingIntent`, to the android operating system. At the specified time, the android system sends a broadcast message containing this intent which calls the `onReceive` method in the `BroadcastReciever` class for the relevant application. The information stored in the intent when the alarm is set can then be extracted and acted upon [18].

In order to receive these broadcasts, the specific implementation of the `BroadcastReceiver` class needs to be registered in the `AndroidManifest.xml`

file. The `AndroidManifest.xml` file contains all information about the application that is pertinent to the android system. It contains all information such as package names, permissions, minimum and target API information, and a slew of other relevant information, including registering `BroadcastRecievers` and `ContentProviders` [20].

ContentProvider

Android `ContentProviders` provide structured, regulated access to specified data. In the context of this application, the `ContentProvider` is used to supply the Gmail application with access to the Microsoft Excel file containing the user's history without saving the file in a public directory. This works by passing an intent, which is usable only by the Gmail application, containing the authority of the `ContentProvider`. In this case, the excel file is the only content which the Gmail application needs to access, so this authority matches the URI of the file. Gmail application then uses this authority to request temporary access to the file provided by the `ContentProvider`. The specific implementation of the `ContentProvider` also needs to be registered in the `AndroidManifest.xml` file [17].

Alarms

Alarms are set for a number of reasons in our application. Refill notifications are set when medications are created, and compliance event notifications and text alerts are set when the schedule page is loaded. The reason for this difference is that each time the schedule page is loaded, the list of compliance events can be different, and so alarms need to be cancelled or set based on the changes to the schedule. In order to get the information from the application to the background process that handles the alarms, intent extras are used. This allows for flexible transfer of multiple types of information and keeps the background process completely separate from the application. A hashmap of currently scheduled events is maintained using the android `PendingIntent` as the key, and the Intent containing the information and extras as the hashed value. Compliance event notifications and buddy text reminders are identified using the unique ID of the relevant compliance event, and refill notifications are identified using the unique ID of the associated medications. This allows the alarm algorithm to compare the list of scheduled alarms, texts and refill events with the current list of scheduled events and medications so that if

new compliance events or medications are added or removed, alarms can be scheduled and cancelled accordingly.

Additional Libraries Used

The Excel file that is sent when the user chooses to export his/her history is generated using the JExcelApi as published and maintained by Andy Khan. This library allows for easy reading and writing of Microsoft Excel files using objects corresponding to workbooks, sheets, and individual cells. Using this library provides a more flexible way to create an Excel readable file than hardcoding text to output as a .csv file. The JExcelApi supports formulas as of version 2.3. This could be useful in future work for providing some statistical analysis of the compliance history [16].

5.4 Implementation of the Model

Overview

The state of the Medical Compliance application that has been continually referred to is maintained by the set of model classes referred to as the model. Figure 21 shows the primary object that the various presenters interact with is the `MCModel`. The model portion of this application is kept completely separate from the view portion, in keeping with the MVP design paradigm described in previous sections. This decision allows for easy unit testing of model classes using conventional Java testing tools such as JUnit. Also in keeping with the MVP design paradigm, the presenter portion of the application needs only interact with a single model class. In our application, this class is the `MCModel`. The `MCModel` class contains references to each of the four subsections of the model. While the MVP design paradigm does allow for complete separation between view and model to allow each to evolve independently, the models four sections loosely mirror the user-side views of the application. This allows for a much more direct translation of data from front-end to back-end. These four sections are the `DailySchedule`, the `MedicationCabinet`, the `BuddyBook`, and the `History`.

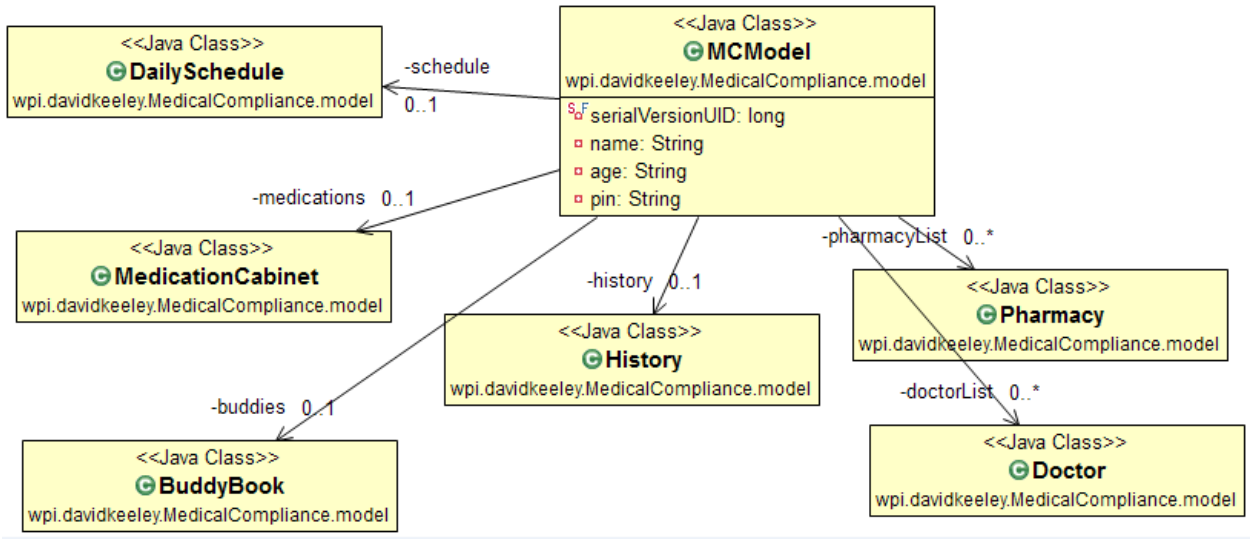

Figure 19: UML Overview of Model

DailySchedule

The daily schedule portion of the model is responsible for, maintaining, and updating all compliance events for a given day. When the user choses to view the daily schedule tab, a list of all of the user's medications is passed to the `DailySchedule` via the `MCModel`. This method then iterates through this list and determines which `ComplianceEvent` objects should be created and added to, or removed from, the day's event list. The algorithm first checks to see if the medication has any events scheduled for the current day. If there are, the algorithm iterates through the list of times and creates a `ComplianceEvent` object containing the proper information. Once a list of new events has been generated, the algorithm checks the current event list for events for medications that no longer exist, and events from previous days. It then checks the new event list for events that have already been scheduled. If an event has not been scheduled, then it is added to the list. Finally the algorithm sorts the events by time, earliest to latest. A pseudocode implementation of the scheduling algorithm is provided in appendix H. A UML diagram of the `DailySchedule` section of the model is shown in Figure 22 below.

Times for compliance events are stored in the application data by using the `SimpeTime` object. This decision was made in order to save memory by not storing an entire java

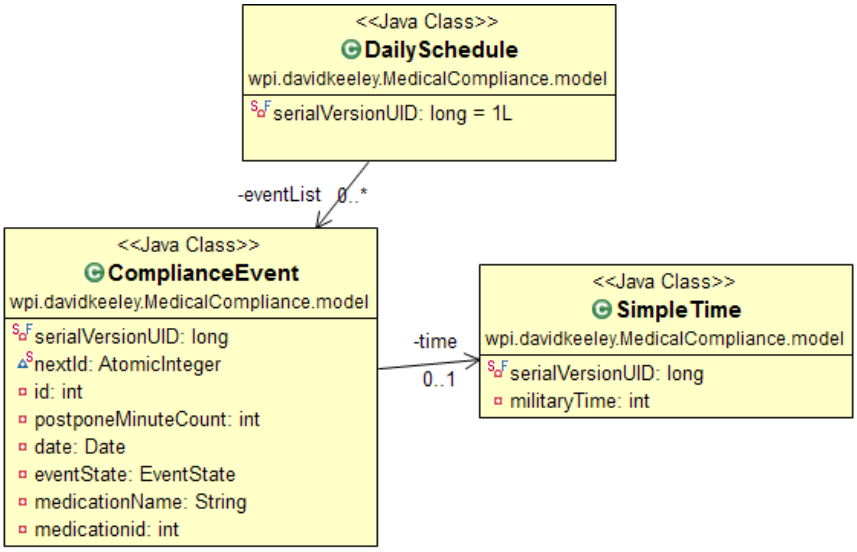

Figure 20: UML diagram of schedule portion of Model

`Calendar` or similar object in memory for each event and to simplify the process by which the time objects can be manipulated.

All calculations based on dates and times are done using the Joda time library. The Joda time library was chosen over the standard Java date and time libraries for two reasons. The first is that it is simply easier to use. Joda provides many intuitive calendar functions that are not available in the standard libraries such as a function to get the number of days between two dates. The second is that it provides a useful abstraction from the system clock of the device that allows the date and time to be spoofed for testing purposes. This functionality is also not present in the Java standard libraries [19].

MedicationCabinet

The medication cabinet section of the model is responsible for storing and maintaining the medications the user wishes to have recorded in his/her compliance history. The medication cabinet portion of the model does not do any significant processing of information such as the event scheduling algorithm in the `DailySchedule`. Instead, this section of the model provides simple functionality for adding/removing medications, getting, updating and removing the `Medication` that is currently being viewed by the user. The list of medications is

maintained using a Java standard hashmap with an integer, which is the medications ID, as the key and the `Medication` as the hashed value. Hashmaps provide instant (O(1)) lookup speeds. This allows the number of medications to be scaled up very high without having a detrimental effect on processing time. Each medication, upon creation, gets a unique serialized ID that is provided by using the Java `AtomicInteger` class. This prevents helps to prevent collisions in the hashmap by ensuring that each medication ID is unique. It also allows for easy comparison between Medication objects. If the ID's are not the same, they are not the same medication. Each `Medication` object contains a number of fields containing information pertinent to dosage, pill strengths, times etc. a UML diagram of the `Medication` section of the model is displayed in Figure 23.

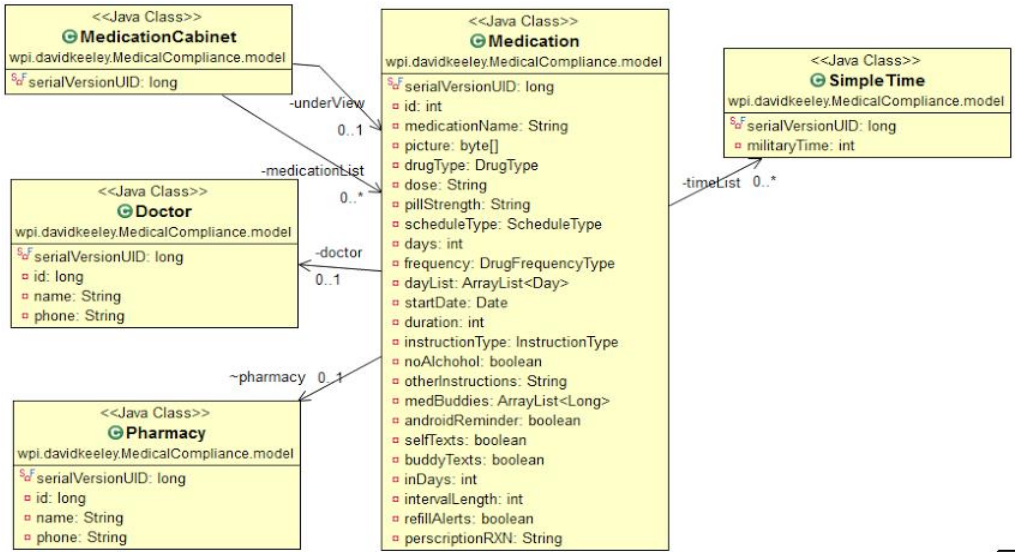

Figure 21: UML diagram of medication portion of Model

BuddyBook

The `BuddyBook` section of the model is implemented in much the same way as the medication cabinet. The `BuddyBook` maintains a list of buddies just as the medication cabinet maintains the list of medications. The `Buddy` object contains contact information that is populated by the contacts on the user's phone. Each buddy also has a unique serial ID also implemented in the same way as the serial ID associated with the `Medication`

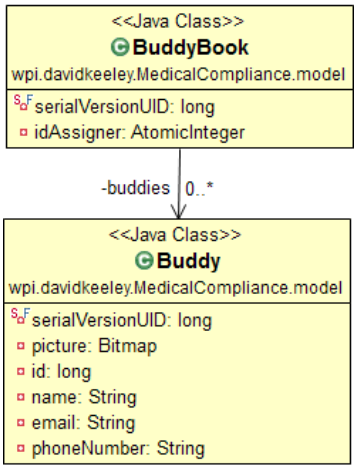

Figure 22: UML diagram of buddy portion of Model

object. A UML diagram of the buddy section is displayed in Figure 24.

History

The `History` portion of the model is used to record the `ComplianceEvents` completed by the user. The objects stored by the history are the `ComplianceHistoryEvent`, the `BGHistoryEvent`, and the `BPHistoryEvent`. The `ComplianceHistoryEvent` is a derivative of the `ComplianceEvent`. It contains an extra field that allows the user to enter some feedback. There is also a rule in the `ComplianceHistoryEvent` class that does not allow the event state to be *PENDING*, because a pending event cannot have been completed yet. The `BGHistoryEvent` records blood pressure readings, and contains rules for critical values that can be used to provide the user with a notification that warns them that the reading is out of the safe range. The `BGHistoryEvent` records blood gravity readings and contains similar safety rule. Figure 25 shows a UML diagram of the History section of the model.

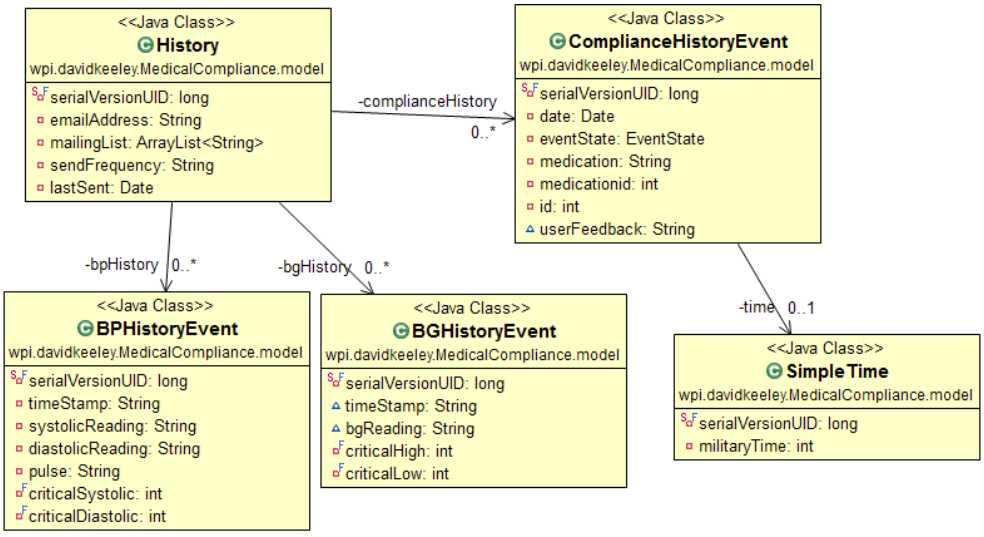

Figure 23: UML diagram of history portion of Model

6. Evaluation Criteria

Before the application can used by the public on a large scale, empirical evidence should exist that verifies that the application functions under stress attributed extreme user use. To provide this evidence via tests, two things must be determined: the volume of use expected in extreme cases and the sets of tests to run that verify that the application indeed supports this extreme use. Should the application fail to function under stress, an `OutOfMemoryError` occurs. This error occurs when the memory required to perform the computations associated with one of the features of the application exceed the amount that can be allocated to the application by the android operating system. This memory allocated is in the form of heap space. This makes the heap allocation during runtime one of the most important factors to consider throughout the evaluation process. The heap space allocated directly depends upon variables like the number of medications that the user has added. The numbers of medications, buddies, doctors, pharmacies, the number of events in the history log, and the number of schedulable timers per medication all have a substantial impact on the performance of the application. To prevent an `OutOfMemoryError`, it is important that the application have bounds on user actions. The application imposes restrictions upon itself based on the following parameters:

- `MEDICATION_MAX` - The maximum number of medications that a user may add.
- `BUDDY_MAX` - The maximum number of buddies that a user may add.
- `DOCTOR_MAX` - The maximum number of doctors that a user may add.
- `PHARMACY_MAX` - the maximum number of pharmacies that a user may add.
- `HISTORY_SIZE` - The maximum number of events saved in the history.
- `MEDICATION_SCHEDULE_MAX` - The maximum number of scheduled times per medication.

To determine the appropriate values of these variables, substantial stress testing is required based on different potential values for these parameters. A wide range of values for tests is necessary to grasp an accurate picture of exactly how the heap space allocation behaves for each user action based upon different parameters. It is also important that testing be conducted without any limits. In this unlimited test, the values of the parameters should be increased exponentially until the application does experience errors associated with heap space.

Knowing exactly when the application fails is important when drawing conclusions about usability. The heap usage of every user function is tested with the following parameter values:

	MEDICATION_MAX	BUDDY_MAX	DOCTOR_MAX	PHARMACY_MAX	HISTORY_MAX	MEDICATION_SCHEDULE_MAX
Test 1	1	1	1	1	1	1
Test 2	5	5	5	5	125	5
Test 3	10	10	10	10	250	10
Test 4	15	25	15	15	500	15
Test 5	25	50	25	25	1000	25
Test 6	50	100	50	50	2000	50
Test 7	unlimited	unlimited	unlimited	unlimited	unlimited	unlimited

Table 1: Shows metrics for each test

Test 1 is the baseline test with the minimum possible values in each category to establish the base memory usage of the application. This test is used to determine exactly how much of the memory used is associated with each feature when not under any stress. Tests 2 through 6 examine the memory usage and heap allocation based on various potential user usages. Test 5 represents the desired constraints for public use. For anticipated standardized user use of the application, it should be able to function properly at those maximum values. These values are derived from reports on prescription medication use published by the Center for Disease Control and Prevention [13]. According to these reports, eighty-eight percent of all individuals over the age of sixty, regardless of gender or ethnicity, take a prescription medication. This age group had the highest prescription medication usage. The application should therefore be able to support these individuals. Thirty-six percent of all individuals that fall into the sixty and over age group use more than five prescriptions medications [13]. The application must at a minimum allow for five medications to be managed. This minimum is reflected by test 2. Statistics published by the American Society of Consultant Pharmacists indicates that the average number of prescription medications increased to eighteen for individuals over the age of eighty who take a prescription medication [14]. The values for constraints in test 5 allow for this extreme usage, making it the proposed goal for public use. An application that performs effectively at these maximum constraints is usable for almost all potential patients. Test 7 is the

aforementioned test and removes the constraints and attempts to determine the limits of the application by exponentially increasing use until an error occurs. This test attempts to deliberately cause an `OutOfMemoryError`.

For all tests of all functions, four measurements are computed: the time required to complete the action, the amount of heap space being used by the action, the amount of allocated heap space that is unused, and the percentage of allocated heap space used. The first metric, time to complete user action is important regardless of memory usage. Even if an acceptable amount of memory is being used and the application is operating within constraints, user experience depends on speed. For optimal user experience, the user should receive feedback within 200 milliseconds of initiating some action. Any wait time beyond 200 milliseconds is perceivable by the user. This is suggested by experiments in timing conducted by psychologists and user experience experts [21]. This metric is loosely used to determine areas for improvement in the application.

The remaining measurements are used to analyze performance of the application with respect to heap usage. The heap size allocated to an application is dynamic and is allocated by the operating system. The heap may grow in size up to a maximum. Should the application exceed this maximum, an `OutOfMemoryError` occurs. The maximum heap size is device dependent [15]. Older devices have very restricted heap space. The device used to test the application allowed a maximum heap size of 64 Mb. Most new devices allow for much larger heap sizes [15]. The chart contained in Appendix A shows the twenty-nine areas of functionality that are analyzed in testing. Areas of functionality refer to the set of operations and tasks necessary for the system to compute in order to get from a state A to a state B. The majority of these functions are associated with the loading of different views. View loading is often the most resource intensive action.

7. Results

Data was collected for Tests 1 through Tests 7. The raw data for each test is contained in Appendices B through G. The data is evaluated with respect to time and memory usage.

7.1 Time

As previously discussed, the experience of the user is largely dependent on speed of the systems response to user actions. Test 1 was the baseline to determine the performance of the applications functionality at minimal use. All the constraints were set to 1 for this test. As shown by Figure 26, all measured areas of functionality, with the exception of the *email_ history* function, completed in fewer than 120 milliseconds. The *email_ history* area of functionality is an exception because the medical compliance application actually launches an external third party application to send emails. The time associated with *email_history* is dependent on this third party application and does not reflect the performance of the medical compliance application. For these reasons, time associated with the *email_history* functionality is shown but not taken into consideration beyond this point.

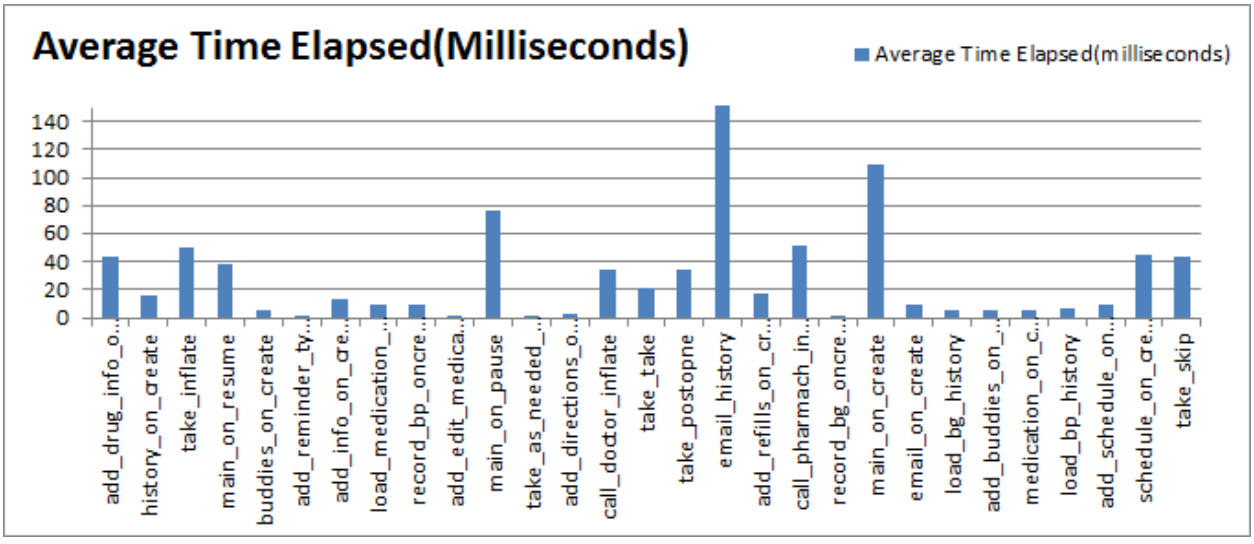

Figure 24: Average time elapsed in milliseconds for each area of functionality for test 1.

Test 5 was conducted with values that were determined to be the ideal constraints for actual patient use. Figure 27 shows the comparison of time elapsed in milliseconds for all areas of functionality operating at the maximum values for all constraints. The results of this test

revealed several areas of concern in the application. The difference between times associated with different areas of functionality was rather large. The majority of all functionality remained below 200 milliseconds but others reached times close to 6000 milliseconds.

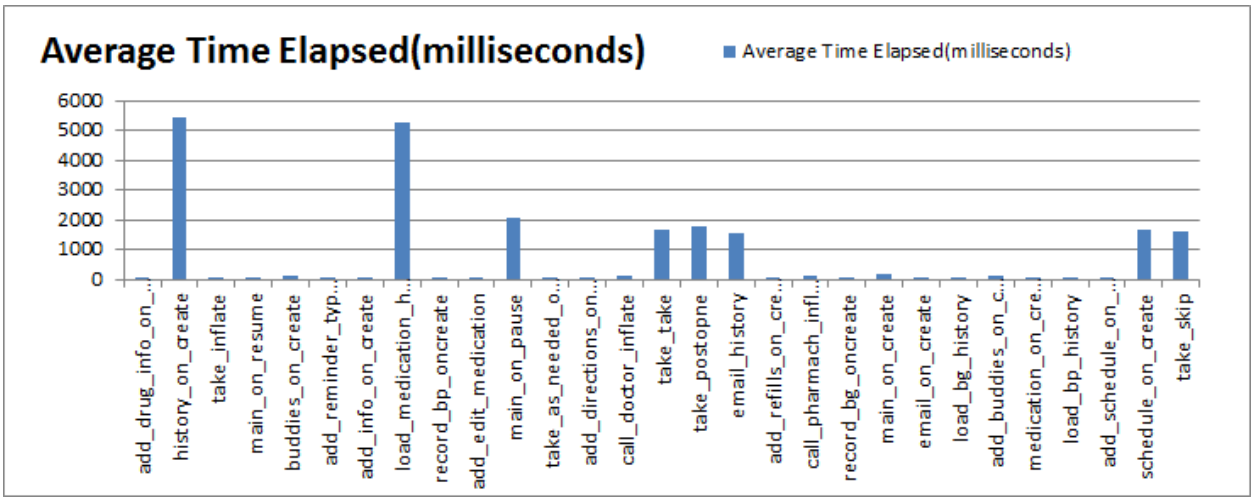

Figure 25: Average time elapsed in milliseconds for each area of functionality for test 5.

The following areas of functionality had times that exceeded expectations and could potentially interfere with user interaction. These areas are:

- Schedule level functionality
  - *schedule_on_create*
  - *take_take*
  - *take_postopone*
  - *take_skipe*
- History level functionality
  - *history_on create*
  - *load_medication_history*
- Application Level functionality
  - *main_on_pause*

The concerning areas of functionality can be grouped into several categories: Schedule, History and Application level functionality. Areas of functionality within each category are often dependent upon each other. Therefore it is important to take a closer look the areas functionality associated with each section across all tests.

Functionality associated with the Schedule is the most complex. The schedule processes compliance events. The number of compliance events is dependent upon the number of

medications and the number of events for each medication. Table 1 shows the number of compliance events that are processed during each of the tests. The time for functionality in the schedule is largely dependent on the number of compliance events per day. Figure 29 shows the relationship of each of the areas on functionality in the schedule section across all tests conducted with explicit maximum constraints enforced.

	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6
Number of Events	1	25	100	225	625	2500

Table 2: Number of compliance events for each test

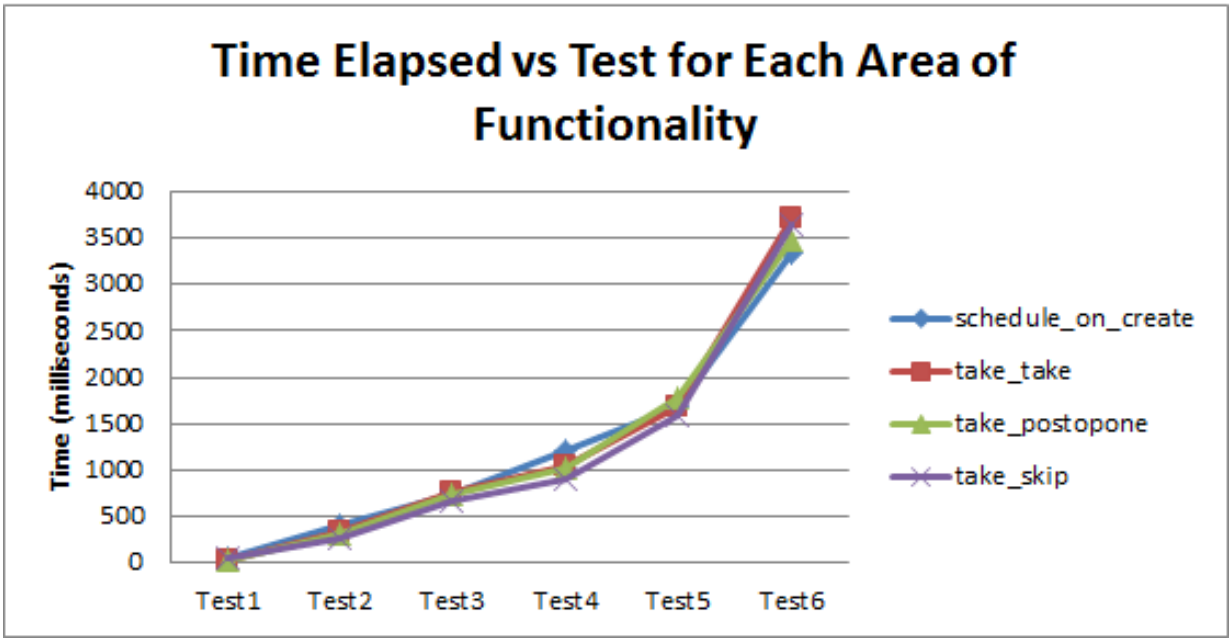

Figure 26: Time elapsed in milliseconds across all tests for select areas of functionality.

When the user takes, postpones or skips a compliance event, large portions of the scheduled must be regenerated. Therefore, the functionality of *scheudle_on_create* is utilized by all areas of functionality in the schedule section. Improving the algorithms associated with the *schedule_on_create* area of functionality would drastically improve the functionality for the entire schedule section. Although the results make it seem as though the system functions poorly, it does not for average use. Previous discussion revealed that an individual of over eighty takes an average eighteen medications per day. The results of Test 2 show twenty-five events being handled in less than 500 milliseconds. The seemingly poor performance of Test 6 is with twenty-five hundred events.

The second area of concern is the history section. Figure 30 shows the number of events logged and displayed in the history section of the application.

	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6
Size of History	1	125	250	500	1000	2000

Table 3: Number of events in history for each test.

Table 2 shows the relationship between the time elapsed and the test conducted for each area of functionality in the history section.

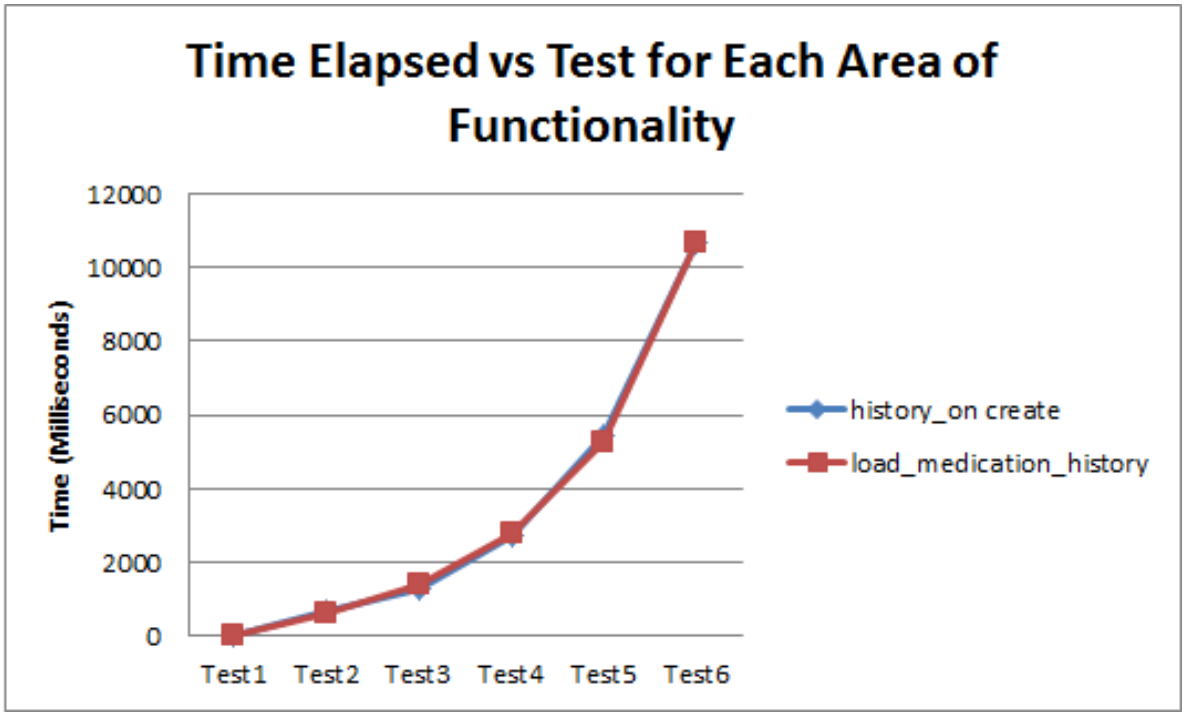

Figure 31: Table showing average time elapsed in milliseconds across all tests for select areas of functionality.

The results are unsurprising. The time associated with *history_on_create* is dependent on *load_medication_history* because every time the user navigates to the history view, the view defaults to the medication history view and not the view for the blood pressure or blood glucose history. Therefore as the time required to load the medication history increases, the time to switch to the history also increases. Unlike the issues associated with the schedule section, this increase in time in the history section is undoubtedly experienced by the average user because even at a rate of five events per day, the user should reach the history limit after some amount of time. Possible solutions to this problem are addressed in the Future Work section.

Test results also revealed that there was a substantial increase in the time associated with the *main_on_pause* functionality. Figure 32 shows this increase across all tests.

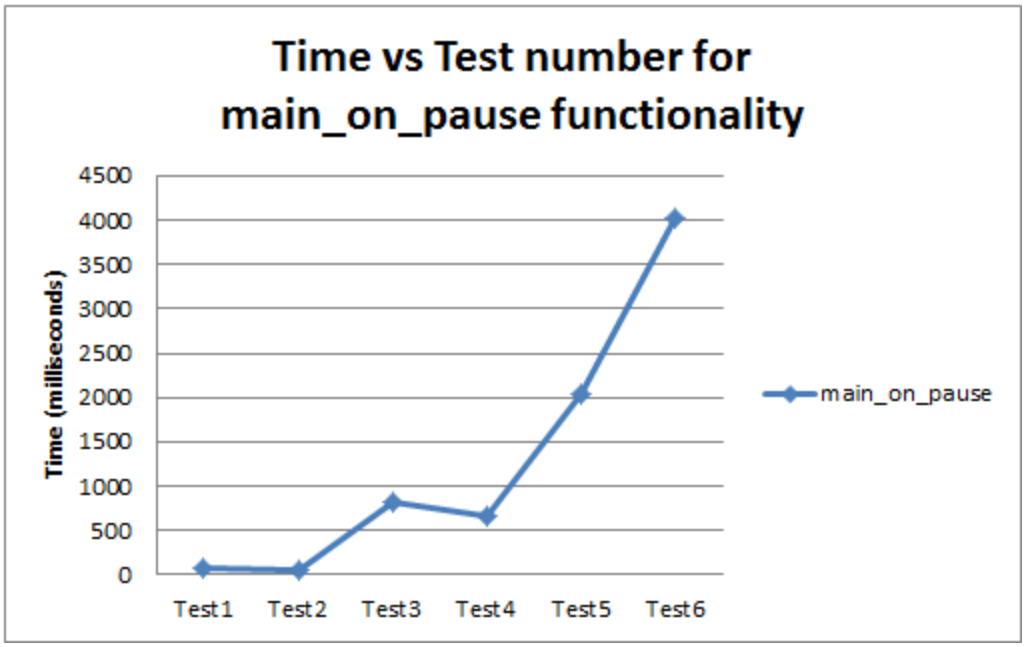

Figure 27: Time elapsed in milliseconds across all tests for the main_on_pause functionality.

This functionality is invoked as the application closes. It is here that the state of the application is saved. The increase in time here is unavoidable. As the amount of information increases, the time required to save this information increases. This increase in time is also not perceivable by the user as this functionality happens after the application has been removed from the users view.

7.2 Memory/Heap Usage

This section analyzes the applications heap usage. To avoid throwing an `OutOfMemoryError`, the heap space used must not have exceeded the heap space allocated. The heap space used by the application is compared to the heap space dynamically allocated by the application and maximum heap space that can be allocated. Analysis here is done primarily with respect to Test 5 as the results from this test reveal memory usage when the application is operating under the ideal constraints. Figure 33 shows the average memory used for each area of functionality. This metric can only be used to make so many inferences about the behavior of the application. Memory usage at the time a particular area of functionality is

invoked is largely dependent on the subsequent invocations of areas of functionality. Memory is managed by the operating system and therefore heap space used by a set of functions is not necessarily freed when the area of functionality has completed its tasks.

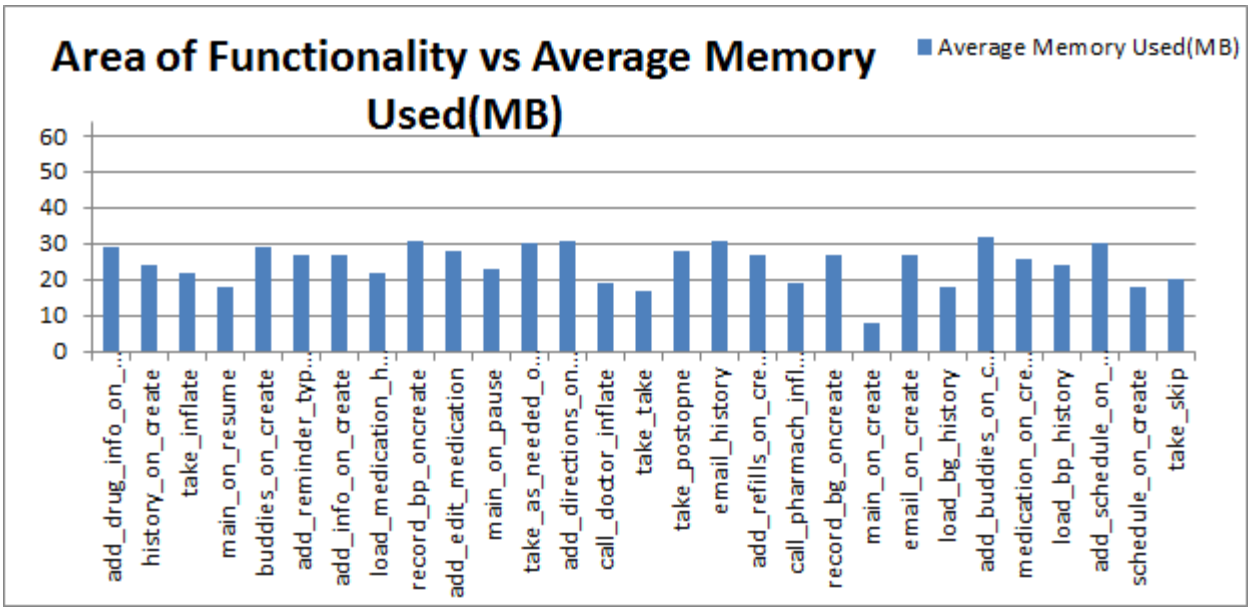

Figure 28: Table showing the avrae memory used by each area of functionlaity for test 5.

Only one important conclusion can be made based on this information. All twenty-nine areas of functionality used an average of 24.55 megabytes of heap space. This equates to an average of 37.5 percent of the 64 available megabytes being used. Not a single area of functionality exceeded 48.44 percent of this max heap size. This suggests that the application functions for users without throwing an `OutOfMemoryError`. Older phones that enforce a max heap size of 48 or even 32 megabytes are able to run the application without reaching this hard memory limit.

Further testing was done to investigate what conditions cause the application to reach this maximum.

MEDICATOIN_MAX	BUDDY_MAX	DOCTOR_MAX	PHARMACY_MAX	HISTORY_MAX	MEDICATION_SCHEDULE_MAX
100	200	100	100	4000	100

Table 4: Test settings at which memory error occurred during unlimited test(test 7)

Table 3 shows the values for the constraints that caused an OutOfMemroyError to be thrown on a device with a 64 megabyte max heap size. No problems occurred until the applications constraints were increased to 400 percent of the values used for Test 5. It would be possible to impose larger constraints like the ones used in Test 6 in the application release; however this could cause problems on older devices with a max heap size of 32 megabytes. Test 5 performed on these devices as well. The proposed values captured by Test 5 are appropriate for public use.

8. Conclusions

An application that leverages mobile technology to address the issues and costs attributed to patient noncompliance has been developed and presented. The smartphone application is the first step in a solution distributable to individuals on a large scale. This application provided an intuitive means of use despite the complicated nature of prescriptions and medication regimens. The application was developed to require minimal user input with features like the QR Code scanning of prescriptions. The intent of this was to streamline use of the application and reduce the burden of excessive user input by unexperienced smartphone users.

The application also serves a means of collecting live data from patients in an effort to increase the synergy between patients and medical professionals. Researchers depend upon accurate information across a large sample of individuals. Throughout use, data corresponding to all user compliance action is passively collected. Although no centralized database for the information collected was developed as part of this project, the foundation exists and data is easily accessible. In addition to compliance information, data about the functionality of the application under use is collected. This functionality data was collected in a series of empirical stress tests and simulations. The evaluation of the test results revealed areas of weakness but also established that the application functions appropriately without failure under extreme use. The application is a dependable means of encouraging medical compliance in patients on an individual basis.

9. Future Work

9.1 Pre-Market Changes

This section summarizes the changes required before the application goes on the android marketplace. These changes are necessary in order for the application to perform its basic functions.

9.1.1 Features to be Implemented

Automatic Emails

This feature provides a means of sending compliance information to all relevant individuals via email automatically every x number of days, where x is set by the user. This again adds convenience and increases compliance accountability for the user. The groundwork for this feature is already present in the application. The broadcast receiver already runs in the background and handles alerts and notifications and the content provider already interfaces with the Gmail application. This feature can be implemented in one of two ways; first, the user's compliance history could be stored in a SQLite database that is accessible to the background process that handles the notifications and text message alerts. This would allow for the excel file containing the compliance information to be generated only when the email is sent, thereby minimizing performance costs. This method requires a major rework of the model and presenter sections of the applications, as well as the addition of an entirely new SQLite database. As a side effect, this method would improve loading times by allowing the application to query only those history events that the user is currently viewing. The second method is to generate the excel file in the history section and update it every time the user completes a compliance event. The excel file would still be available to the content provider and the history information would not need to be available to the broadcast receiver. This method requires minimal changes to the application, with any refactoring of code being limited to the history section of the model. This method could have a negative effect on performance when adding events to the history and does not have the added bonus of improving loading times.

9.1.2 Performance

9.1.2.1 Schedule

Future work can improve scheduling performance under worst case scenarios in a number of ways. First is to improve the efficiency of the scheduling and setAlarm algorithms. Both of these algorithms are currently running at a worst case of roughly $O(n^4)$. This is acceptable in normal and even extreme use cases, but quickly causes issues in stress tests. These algorithms can be improved by consolidating for loops to reduce the number of times each list is iterated through, as well as better partitioning their functionality so that only the necessary pieces of each function are called. Another way to improve performance in the worst case would be to reduce the number of times the schedule onCreate() method is called. This can also be done by better partitioning its functionality.

An alternative approach is to reduce the number of events loaded when the schedule tab is opened. The user only views between ten and twenty compliance events at a time. This allows us to only load the events that are currently under view, and dynamically load the rest as the user scrolls.

9.1.2.2 History

The reason for poor performance in the history loading times is the fact that each history event requires five separate views to display it. This means that when the user has 2,000 history events, the application is actually loading 10,000 views. The way to reduce this is to only load the events currently under view as mentioned in the schedule section. This improvement is more important than in the schedule section because while the scheduled events are added and removed daily, history events are accrued over time.

9.2 Additional Features

These features are not necessary for the application to perform its basic functions and can be added after the application is published on Google Play in the form of downloadable updates.

9.2.1 Compliance Score

The compliance score is a heuristic evaluation of a given users medical compliance. The better they adhere to their schedules the higher the score. This heuristic is based on a number of studies done on patient medical compliance and uses the same statistics as many of them. Although this provides minimal medical benefit to doctor and patient, it is an easy way to

evaluate a user's overall performance and can be an asset when setting compliance goals for patients. The compliance score should be a running estimation of the users overall medical compliance. Things to take into account would be the number of minutes before/after the scheduled time the user completes an event, if the event is missed or skipped, the number of times the event is postponed etc. Implementing the compliance score requires minimal changes to the front end of the application. All calculations related to the compliance score would fit best in the history portion of the model as the events are recorded.

9.2.2 Weekly Medications

When the user creates a new medication, they have the option of scheduling the medication daily, weekly, every x days, and take-as-needed. Currently, when the user selects weekly, the medication is scheduled every 7 days from the start date. This should be updated to allow the user to select one or more days on which the medication should be taken e.g. every Monday and Thursday. This should be done on the user side of the application using a dropdown menu that allows multiple selections. On the back end of the application, the scheduling algorithm would need to be updated to handle a list of days during the week, as well as a list of times.

9.2.3 Cloud Storage

Currently, all user information, including compliance events, medications, and history is stored locally on the users device. This was convenient for development purposes, but should be updated. Information should be stored on a server to allow multiple users to log in on the same device, as well as to allow one user to sync compliance information across multiple devices. Users often share devices in the case of a family tablet, or use multiple devices. Allowing users access to multiple accounts on multiple devices provides an added measure of convenience for the user as well as increased accountability. The implementation of this feature should not be dependent on connectivity. Users should have the option to temporarily store information locally to allow use of the application offline. This information should then sync with the online database when connectivity is restored. Users should also have the option of permanently storing all information locally as some users are not comfortable having their medical information stored on the cloud.

9.2.4 Notification Buttons

Expandable notifications are available in phones running android 4.1 or newer. These notifications allow the user to control the application from the notification pull-down. Buttons can be added to take, postpone, and skip medications, or to call a doctor or pharmacy. Because expandable notifications are not available in older phones, issues regarding backwards compatibility need to be addressed.

10. References

[1] L. Morris, R. Shultz, " Patient Compliance," Journal of Clinical Pharmacy and Therapeutics, Vol. 17, no. 5, 283-295, 1992.

[2] D. West, "How Mobile Devices are Transforming Healthcare ," Issues in Technology Innovation, Vol. 18, no. 1 , 1-11, May 2012.

[3] E. Vermeire, H. Hearnshaw, P. Van Royen, "Patient adherence to treatment," Journal of Clinical Pharmacy and Therapeutics, Vol. 26, no. 5, 331-342, May 2012.

[4] Rosenthal, A , Waste in the Health Care System , 'The New York Times', p.1 – 1

[5] K. Fairman, B. Motheral, 1st Initial. , "Evaluating Medication Adherence:," Academy of Managed Care Pharmacy, Vol. 6, no. 6, pp. 449-459, November 2000.[]. :http://ns.amcp.org/data/jmcp/ce_v6_499-506.pdf. [Accessed January 2014]

[6] Snyder, T , "MVC or MVP Pattern". Retrieved January, 2014 Available: http://www.infragistics.com/

[7] Zelenski, J , "Programming Abstractions". Retrieved January, 2014 Available: http://www.stanford.edu/

[8] Google, "Testing Fundamentals". Retrieved January, 2014 Available: http://developer.android.com/tools/testing/testing_android.htmlSrc 4

[9] Bryant, S , "OO Principles: Encapsulation and Decoupling". Retrieved January, 2014 Available: http://www.bryantwebconsulting.com/blog/index.cfm/2009/7/9/OO-Principles-Encapsulation-and-Decoupling

[10] Google, "Activity". Retrieved January, 2014 Available: http://developer.android.com/reference/android/app/Activity.html

[11] Google, "Fragments". Retrieved January, 2014 Available: http://developer.android.com/guide/components/fragments.html

[12] Google, "Fragment Transaction". Retrieved January, 2014 Available: http://developer.android.com/reference/android/app/FragmentTransaction.html

[13] Q. Gu, C. Dillon, V. Burt , "NCHS Data Brief". Retrieved January, 2014 Available: http://www.cdc.gov/nchs/data/databriefs/db42.htm

[14] "ASCP Fact Sheet". Retrieved January, 2014 Available: https://www.ascp.com/articles/about-ascp/ascp-fact-sheet

[15] Google, "Managing Your App's Memory". Retrieved January, 2014 Available: http://developer.android.com/training/articles/memory.html

[16] "Java Excel API". Retrieved January, 2014 Available: http://jexcelapi.sourceforge.net/

[17] Google, "Content Provider Basics". Retrieved January, 2014 Available: http://developer.android.com/guide/topics/providers/content-provider-basics.html

[18] Google , "BroadcastReciever". Retrieved January, 2014 Available: http://developer.android.com/reference/android/content/BroadcastReceiver.html

[19] "Joda Time API". Retrieved January, 2014 Available: http://www.joda.org/joda-time/

[20] Google, "App Manifest". Retrieved January, 2014 Available: http://developer.android.com/guide/topics/manifest/manifest-intro.html

[21] M. Velmans, "When Perception Becomes Conscious," British Journal of Psychology, Vol. 90, no. 4, pp. 543-566, http://www.goldsmiths.ac.uk/academic/ps/velmans.htm. [Accessed January 2014]

[22] Montuno Software, “About Dosecast”. Retrieved January, 2014 Available: http://www.montunosoftware.com/products/dosecast/about/

[23] Drugs.com. Retrieved January, 2014 Available http://www.drugs.com/

[24] MediSafe, “MediSafe Project”. Retrieved January, 2014 Available: http://medisafeproject.com/

Appendix A: List of functionality tested with respect to performance metrics

This is a list of all of the areas of functionality that were considered in performance testing.

Functionality(name)	**Description of functionality**
`main_on_create`	Launch application
`main_on_resume`	Resume paused application
`main_on_pause`	Pause running instance of application
`schedule_on_create`	Calculate events and display daily schedule
`medication_on_create`	Display medication list
`buddies_on_create`	Display list of users buddies
`history_on_create`	Display history view with medication history.
`call_doctor_inflate`	Display view containing list of user added doctors
`call_pharmacy_inflate`	Display view containing list of user added pharmacies
`take_as_needed_on_create`	Display view containing list of all take as needed medications
`take_inflate`	Display view for individual event on daily schedule
`take_take`	Update daily schedule and record user taking of medication.
`take_postopne`	Update daily schedule and record user postponing of medication
`take_skip`	Update daily schedule and record user skipping medication

record_bp_oncreate	Display view for user to record blood pressure
record_bg_oncreate	Display view for user to record blood glucose reading.
add_edit_medication	Display view to update or add new medication.
email_on_create	Display view to send history as email attachment.
load_medication_history	Load and display medication history.
load_bp_history	Load and display blood pressure reading history
load_bg_history	Load and display blood glucose reading history
email_history	Email history
add_drug_info_on_create	Display view to update drug info of a medication
add_schedule_on_create	Display view to update schedule of a medication
add_directions_on_create	Display view to update directions for taking a medication
add_buddies_on_create	Display view to update buddies associated with a medication
add_reminder_types_on_create	Display view to update reminders for a medication
add_refills_on_create	Display view to update refill schedule for a medication
add_info_on_create	Display view to update doctor and pharmacy info for a medication

Appendix B: Raw results from Test 1.

Test 1 is the baseline test. The values used as maximum constraints were set to one. This was done to determine the base memory usage at the absolute minimum possible usage.

Tag	Average Time Elapsed(milliseconds)	Average Memory Used(MB)	Average Memory Free(MB)	Percentage Memory Used(MB)
add_drug_info_on_create	43	15	1	94
history_on_create	16	16	3	84
take_inflate	50	14	3	82
main_on_resume	38	12	6	67
buddies_on_create	6	14	4	78
add_reminder_types_on_create	2	13	3	81
add_info_on_create	14	13	2	87
load_medication_history	10	17	5	77
record_bp_oncreate	9	15	3	83
add_edit_medication	1	14	1	93
main_on_pause	77	16	4	80
take_as_needed_on_create	2	14	2	88
add_directions_on_create	3	13	3	81

call_doctor_inflate	35	13	3	81
take_take	21	13	4	76
take_postopne	34	15	1	94
email_history	483	15	8	65
add_refills_on_create	17	13	3	81
call_pharmach_inflate	52	13	2	87
record_bg_oncreate	1	15	2	88
main_on_create	109	8	8	50
email_on_create	9	19	3	86
load_bg_history	5	17	5	77
add_buddies_on_create	6	13	3	81
medication_on_create	5	14	3	82
load_bp_history	7	17	6	74
add_schedule_on_create	9	12	4	75
schedule_on_create	45	14	3	82
take_skip	43	14	2	88

Appendix C: Raw results from Test 2:

The values used for the maximum constraints in test 2 were 25% of values expected for average use of the application by the average patient.

add drug_info_on_create	69	15	4	79
history_on_create	682	15	2	88
take_inflate	33	16	2	89
main_on_resume	38	10	6	63
buddies_on_create	21	14	2	88
add_reminder_types_on_create	2	15	4	79
add_info_on_create	17	15	4	79
load_medication_history	632	16	1	94
record_bp_oncreate	16	15	3	83
add_edit_medication	13	13	5	72
main_on_pause	46	15	8	65
take_as_needed_on_create	1	16	3	84
add_directions_on_create	3	15	3	83
call_doctor_inflate	44	15	3	83
take_take	339	13	5	72
take_postopne	298	17	1	94

email_history	0	0	0	0
add_refills_on_create	10	15	4	79
call_pharmach_inflate	58	15	3	83
record_bg_oncreate	1	15	3	83
main_on_create	65	8	8	50
email_on_create	15	15	1	94
load_bg_history	10	15	2	88
add_buddies_on_create	15	15	3	83
medication_on_create	19	14	4	78
load_bp_history	36	15	2	88
add_schedule_on_create	41	14	5	74
schedule_on_create	414	14	4	78
take_skip	271	17	2	89

Appendix D: Raw results from Test 3.

The values used for the maximum constraints in test 3 were 50% of values expected for average use of the application by the average patient.

Tag	Average Time Elapsed(milliseconds)	Average Memory Used(MB)	Average Memory Free(MB)	Percentage Memory Used(MB)
add_drug_info_on_create	30	13	5	72
history_on_create	1266	16	2	89
take_inflate	55	14	3	82
main_on_resume	114	15	6	71
buddies_on_create	27	17	5	77
add_reminder_types_on_create	2	15	3	83
add_info_on_create	13	16	3	84
load_medication_history	1385	16	5	76
record_bp_oncreate	1	16	4	80
add_edit_medication	1	13	5	72
main_on_pause	825	18	8	69
take_as_needed_on_create	1	15	7	68
add_directions_on_create	3	16	3	84

call_doctor_inflate	52	11	6	65
take_take	762	14	3	82
take_postopne	736	12	6	67
email_history	758	20	6	77
add_refills_on_create	17	16	2	89
call_pharmach_inflate	58	13	5	72
record_bg_oncreate	1	18	3	86
main_on_create	154	8	8	50
email_on_create	10	18	2	90
load_bg_history	10	20	1	95
add_buddies_on_create	29	15	3	83
medication_on_create	26	16	4	80
load_bp_history	25	18	3	86
add_schedule_on_create	24	16	3	84
schedule_on_create	738	16	3	84
take_skip	671	15	2	88

Appendix E: Raw results from Test 4.

The values used for the maximum constraints in test 3 were 100% of values expected for average use of the application by the average patient. This was the test case designed to simulate expected user behavior under normal circumstances.

Tag	Average Time Elapsed(milliseconds)	Average Memory Used(MB)	Average Memory Free(MB)	Percentage Memory Used(MB)
add_drug_info_on_create	40	23	3	88
history_on_create	2735	17	4	81
take_inflate	39	21	3	88
main_on_resume	129	15	6	71
buddies_on_create	70	20	6	77
add_reminder_types_on_create	2	25	1	96
add_info_on_create	22	25	1	96
load_medication_history	2802	20	6	77
record_bp_oncreate	1	22	3	88
add_edit_medication	2	22	3	88
main_on_pause	674	18	8	69
take_as_needed_on_create	1	23	8	74
add_directions_on_create	3	24	2	92

call_doctor_inflate	87	21	3	88
take_take	1050	20	4	83
take_postopne	1013	21	3	88
email_history	981	23	8	74
add_refills_on_create	9	25	1	96
call_pharmach_inflate	70	22	3	88
record_bg_oncreate	1	20	6	77
main_on_create	176	8	8	50
email_on_create	11	21	5	81
load_bg_history	4	23	2	92
add_buddies_on_create	58	24	1	96
medication_on_create	40	21	5	81
load_bp_history	50	23	2	92
add_schedule_on_create	7	23	2	92
schedule_on_create	1205	20	6	77
take_skip	904	22	3	88

Appendix F: Raw results from Test 5.

The values used for the maximum constraints in test 5 were 200% of values expected for average use of the application by the average patient. The values used in this test also represent the values expected when the application is put under extreme stress by the far above average user. This test represented our goal for constraint values for public use.

Tag	Average Time Elapsed(milliseconds)	Average Memory Used(MB)	Average Memory Free(MB)	Percentage Memory Used(MB)
add_drug_info_on_create	64	29	4	88
history_on_create	5413	24	4	86
take_inflate	29	22	1	96
main_on_resume	48	18	8	69
buddies_on_create	130	29	6	83
add_reminder_types_on_create	3	27	6	82
add_info_on_create	12	27	6	82
load_medication_history	5259	22	12	65
record_bp_oncreate	1	31	1	97
add_edit_medication	1	28	4	88
main_on_pause	2048	23	9	72
take_as_needed_on_create	1	30	2	94

add_directions_on_create	3	31	2	94
call_doctor_inflate	104	19	2	90
take_take	1685	17	3	85
take_postopne	1772	28	4	88
email_history	1534	31	7	82
add_refills_on_create	15	27	6	82
call_pharmach_inflate	129	19	2	90
record_bg_oncreate	1	27	5	84
main_on_create	197	8	8	50
email_on_create	15	27	7	79
load_bg_history	8	18	15	55
add_buddies_on_create	110	32	1	97
medication_on_create	74	26	4	87
load_bp_history	78	24	9	73
add_schedule_on_create	10	30	3	91
schedule_on_create	1678	18	3	86
take_skip	1591	20	0	100

Appendix G: Raw results from Test 6.

The values used for the maximum constraints in test 6 were 400% of values expected for average use of the application by the average patient. This test was used to determine performance when the application was pushed far beyond necessary for public use. No user is expected to use the application under these circumstances.

Tag	Average Time Elapsed(milliseconds)	Average Memory Used(MB)	Average Memory Free(MB)	Percentage Memory Used(MB)
add_drug_info_on_create	27	12	6	67
history_on_create	10694	26	11	70
take_inflate	36	45	5	90
main_on_resume	81	26	4	87
buddies_on_create	397	12	6	67
add_reminder_types_on_create	2	15	2	88
add_info_on_create	12	17	1	94
load_medication_history	10670	26	14	65
record_bp_oncreate	1	13	2	87
add_edit_medication	1	11	6	65
main_on_pause	4035	42	2	95
take_as_needed_on_create	1	29	4	88

add_directions_on_create	3	13	5	72
call_doctor_inflate	159	13	2	87
take_take	3719	45	4	92
take_postopne	3480	41	8	84
email_history	1949	41	4	91
add_refills_on_create	11	16	1	94
call_pharmach_inflate	149	13	2	87
record_bg_oncreate	1	15	1	94
main_on_create	73	8	8	50
email_on_create	15	36	4	90
load_bg_history	5	35	3	92
add_buddies_on_create	197	15	3	83
medication_on_create	114	14	1	93
load_bp_history	146	35	3	92
add_schedule_on_create	8	12	5	71
schedule_on_create	3327	32	6	84
take_skip	3638	44	5	90

Appendix H: Compliance Database

H.1 Public Medical Information Databases

In an effort to increase synergy between patients and professionals, we looked at ways of using the compliance data that would be collected from mass deployment of the application. One such way to use that data would be to build an open source platform containing the compliance data collected for use in research. This section outlines the necessary pieces required for building a publicly available medical compliance information database. This includes a summary of currently existing databases containing similar information, regulations surrounding the creation of such a database and an overview of the types of information that are most important to compliance researchers.

H.2 Currently Existing Databases

Currently, there exist a fairly large amount of publicly or pseudo-publically available patient information databases, and while none of them focus specifically on compliance, it is still relevant to analyze some of the larger databases as reference points for regulations, information collection, data models and accessibility.

H.2.1 OSHPD of California

The Office of Statewide Health Planning & Development in California maintains a database of patient discharge information pertaining to inpatient care, emergency care, and ambulatory surgeries. This data is collected from a specifically licensed group of hospitals within the state of California. This database includes demographic information such as age, gender, county of residence, and race/ethnicity as well as medical information such as diagnoses, treatment information, total charges, and expected payment method. A running total of patient visits by county is also maintained and can be downloaded from the OSHPD homepage for free. This database falls underneath the pseudo-public category of patient health information. While the information is available, a written request must be submitted and the there is a cost per piece of patient data.

In order to maintain patient anonymity, the OSHPD has revised the information collected and overall structure of their database on a yearly basis since 1999. Some of the most recent

changes include moving from a five digit patient zip code to a three digit patient zip code, generalizing demographic information, and applying filter rules to specific medical conditions that occur only once per county. Information is collected for this database using the Medical Information Reporting for California System (MIRCal) and is collected during the patients visit and reported upon patient discharge.

H.2.2 National Program of Cancer Registries

The National Program of Cancer Registries (NCPR) is a database of patient information for individuals with cancer. It is maintained by the CDC and information is collected in a hierarchical fashion. Individual hospitals maintain a cancer registry that is updated every time a patient with cancer is admitted. The information added to the registry is censored by the hospitals Cancer Registrar and stored in the local registry. Once a year, each hospital sends the information collected to the state registry, and each state registry in turn updates the national registry, also once a year. This information is patient centric and contains a great deal of information pertaining to the patient's demographic, lifestyle, medical history, and occupation. For this reason, this information is made available only to medical professionals and cancer researchers with specific clearance. For those medical professionals who do not have specific clearance, summaries of data that pertain to generalized questions about cancer information can be requested. Examples of these questions include "Are more people getting lung cancer this year than last year" or "Are people with liver transplants more likely to be diagnosed with leukemia". Because this database is maintained by the US Government, the information and query answers are considered public goods and as such are available free of charge.

H.2.3 The State Inpatient Databases

The SID are a set of medical databases maintained by the data organizations in various states. It encompasses ninety-seven percent of inpatient discharges across the US every year. Information is collected from states on a yearly basis. The information is then stored in a standard format that tracks over 100 variables including demographic information, diagnostics, treatment, and financial data. All of this information is patient centric and stored in a patient centric way. Measures are taken at both the state and national levels to protect the anonymity of patients. Forty six states now submit medical information on inpatient discharges. The SID are

publically accessible to anyone who fills out an application and is approved. The files can then be purchased. The data includes records for as far back as 1990.

H.3 Limitations

Because there are no publically available medical compliance databases, the most obvious limitations surrounding those patient centric databases that currently are available is that none of them contain compliance information. This is the issue our project attempts to address. Other limitations of current databases are caused by regulatory policies. Health information is very sensitive in nature and as such is surrounded by legal constraints. These limitations cannot be addressed within the scope of this project, and many of these databases have been designed in such a way as to maximize information while still remaining compliant. These constraints, for the most part, are for the safety of the patient, and are not the focus of this project. Instead, strategies used by these databases to maximize information should be used in during the design process for the compliance database.

H.4 What are Researchers Looking For?

Compliance information comes in many forms, none of which, as we examined earlier, are considered a gold standard for determining compliance [5]. The accuracy of a given metric can often be dependent on the length, type and complexity of the study. Due to the nature of mobile applications and the context of this project, the focus of this section is on indirect measurements of medical compliance. Indirect methods include pill counts, patient reports, physician's opinions and a slew of other, often quite subjective variables. For this project, we narrow the focus down to variables that can be measured objectively and empirically and combined in a way that forms a meaningful metric by which to express compliance.

One such metric that is mentioned in many studies is the Medication Possession Ratio (MPR). This is essentially the number of days on which the patient has possession of prescribed medication divided by the number of days observed. This can be calculated in a number of ways, but in the context of a mobile application, recommended refill dates, actual refill dates, and pill/dose counts can be used to determine whether or not the patient has any medication left to take on a given day. The number of days where the patient has medicine is the divided by the

total number of days since the medication was prescribed [5]. This metric is widely used and is seen as a solid estimate of persistence in long term medical compliance studies.

Using information provided by a mobile medication compliance application, the measure of MPR could be made much more accurate than it already is. MPR is designed to estimate how often a patient is taking their medication, based on when it is available to them. It does not account for patients taking medications late or not according to the prescribed schedule. However, allowing the user to input this information when medicine is taken allows this to factor into a patients overall compliance score.

Compliance studies in the past have had limited access to direct patient input due to cost, time, and regulations. The advent of mobile technology presents a way to gain access to this direct patient input. A secondary goal of this project is take this newly accessible information and combine it in a meaningful and concise way to give an accurate estimate of a patient's overall medical compliance.

H.5 Database Characteristics

Our medical compliance application provides a means to contribute to medical compliance research. Currently there are no publicly available, anonymous medical compliance information databases. The nature of the application presents the opportunity to create such a database. Providing this information to researchers could have a profound impact on the field of medical compliance, but the sensitive nature of the information gives rise to some concerns.

H.5.1 Anonymity

Because of the sensitivity of medical information, and because the database is publically available, there can be no identifying information associated with any patient records. Ideally, it should be impossible to link a patient's medical compliance data to the patient him/herself without express permission from the patient to access his/her compliance records. Possible solutions to this problem could be to associate each patient record with a unique, randomly generated ID number that has nothing to do with the patient and is used only to separate patient records from one another. Another solution is to automatically aggregate any publically available data. That is to say, instead of supplying researchers with individual patient records, we would provide statistics based on medication type, demographic information, compliance score,

or any number of other factors. This would maintain patient anonymity by grouping all similar patients together in a purely statistical representation of collected compliance data.

H.5.2 User Consent

User consent is also a valid concern in this context. A vehicle for users to be expressly informed about the nature and availability of the information collected and the option to consent or decline, as well as an anonymity guarantee, must be provided. This could be implemented in the form of a onetime pop-up shown when the user first opens the application. The user should have the option of viewing this information or changing his/her answer in application settings at any time.

H.5.3 Information Collection

Once the user consents to providing the anonymous compliance data, any information transfers should be completely transparent to the user. They should happen in the background and at night, when the user is least likely to be using his/her device. Also, if the user at first declines to provide information, and then changes his/her answer, only information gathered after consent is given should be recorded.