Doctor Patient Communication System – Android Application

GRADUATE PROJECT REPORT

Submitted to the Faculty of
The School of Engineering & Computing Sciences
Texas A&M University-Corpus Christi
Corpus Christi, TX

In Partial Fulfillment of the Requirements for the Degree of Master of Science in
Computer Science

By
Anvesh Paidipala
Spring 2015

Committee Members

Dr. Ajay Katangur
Committee Chairperson

Dr. Dulal Kar
Committee Member
ABSTRACT

Doctor Patient Communication System (DPCS) is an Android app which sets up online communication between a doctor and a patient. This app is helpful to patients to ask questions and state their concerns to doctors regarding their health condition. This app will facilitate the patients to interact with doctors without making any physical appointments, but the patients are restricted to only one message per day before receiving a reply. If the patient gets a response from the corresponding doctor, the patient is allowed to send another message. In addition using this app, the patient can make an appointment to meet the doctor in clinic/hospital. DPCS app would also facilitate the patient’s pharmacy selection to pick up the medication. Similarly, this app is beneficial to doctors by providing the following functionalities: patient interaction through messaging, sending prescription to pharmacies, confirming appointments, information sharing with other doctors, and patient referrals. Unlike other similar kinds of apps in the android play store, DPCS has unique features such as issuing online prescription to patients, referring patients to a specialist, sending health tips to patients, and effectively, reducing the cost of customer service and providing a vital communication link between doctors and patients.
# TABLE OF CONTENTS

Abstract.................................................................................................................................................. ii

Table of Contents..................................................................................................................................... iii

List of Figures........................................................................................................................................... vii

List of Tables............................................................................................................................................. ix

1. Background and Rationale................................................................................................................1

   1.1 Introduction to DPCS......................................................................................................................3

   1.2 Existing Applications......................................................................................................................5

      1.2.1 Epocrates......................................................................................................................................5

      1.2.2 Doctor @t Work......................................................................................................................5

      1.2.3 Diagnotes..............................................................................................................................6

   1.3 Proposed Solution ........................................................................................................................6

   1.4 Android Operating System.........................................................................................................7

   1.5. Android Studio 1.0.1 ................................................................................................................8

   1.6. Parse Cloud Server.....................................................................................................................8

      1.6.1. Features of Parse ................................................................................................................9

2. Narrative.............................................................................................................................................11

   2.1. Problem Statement......................................................................................................................11

   2.2. Motivation.....................................................................................................................................11
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3</td>
<td>Product Description</td>
<td>12</td>
</tr>
<tr>
<td>2.4</td>
<td>Product Scope</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>Proposed Product Design</td>
<td>13</td>
</tr>
<tr>
<td>3.1</td>
<td>Product Design and Architecture</td>
<td>13</td>
</tr>
<tr>
<td>3.2</td>
<td>Use case Diagram</td>
<td>14</td>
</tr>
<tr>
<td>3.3</td>
<td>Class Diagrams</td>
<td>15</td>
</tr>
<tr>
<td>3.4</td>
<td>Sequence Diagram</td>
<td>18</td>
</tr>
<tr>
<td>3.5</td>
<td>User Interaction</td>
<td>19</td>
</tr>
<tr>
<td>4</td>
<td>Implementation of the Application Modules</td>
<td>23</td>
</tr>
<tr>
<td>4.1</td>
<td>Registration &amp; Login</td>
<td>23</td>
</tr>
<tr>
<td>4.2</td>
<td>User Inbox</td>
<td>25</td>
</tr>
<tr>
<td>4.3</td>
<td>Doctor Dashboard</td>
<td>26</td>
</tr>
<tr>
<td>4.4</td>
<td>Add/Block Patient</td>
<td>26</td>
</tr>
<tr>
<td>4.5</td>
<td>Interaction through messages</td>
<td>27</td>
</tr>
<tr>
<td>4.6</td>
<td>Patient Referrals</td>
<td>28</td>
</tr>
<tr>
<td>4.7</td>
<td>Prescribe Medication</td>
<td>29</td>
</tr>
<tr>
<td>4.8</td>
<td>View Appointments</td>
<td>30</td>
</tr>
<tr>
<td>4.9</td>
<td>Register with Doctor</td>
<td>31</td>
</tr>
</tbody>
</table>
4.10. View Prescriptions

4.11. View Health Tips

4.12. Request for Appointment

4.13. Payment Tracker

5. Testing and Evaluation

5.1. Registration

5.2. Login

5.3. Inbox

5.4. Doctor Dashboard

5.5. Add Patient

5.6. Block Patient

5.7. Send Message

5.8. Send Health Tips

5.9. Patient Referral

5.10. Prescribe Medication

5.11. Search nearby locations on Google maps

5.12. View Appointments

5.13. View Blocked Patients
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.14</td>
<td>Track Payments</td>
<td>49</td>
</tr>
<tr>
<td>5.15</td>
<td>Register with Doctor</td>
<td>50</td>
</tr>
<tr>
<td>5.16</td>
<td>View Prescriptions</td>
<td>51</td>
</tr>
<tr>
<td>5.17</td>
<td>View Health Tips</td>
<td>52</td>
</tr>
<tr>
<td>5.18</td>
<td>Make Appointments</td>
<td>53</td>
</tr>
<tr>
<td>5.19</td>
<td>Card Details</td>
<td>54</td>
</tr>
<tr>
<td>5.20</td>
<td>Payment Tracker</td>
<td>55</td>
</tr>
<tr>
<td>5.21</td>
<td>Sending Email Notifications</td>
<td>56</td>
</tr>
<tr>
<td>5.22</td>
<td>Sending Push Notifications</td>
<td>57</td>
</tr>
<tr>
<td>6</td>
<td>Conclusion &amp; Future Works</td>
<td>58</td>
</tr>
<tr>
<td>7</td>
<td>Bibliography &amp; References</td>
<td>59</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

Figure 1.1: Global Mobile Users Statistics (comSCORE 2014)...........................................1

Figure 1.2: Mobile OS market share as of September 2014 (NetApplications)......................3

Figure 1.3: Parse Cloud Server Pricing & Usage Details......................................................10

Figure 3.1. DPCS Architecture.............................................................................................13

Figure 3.2. DPCS Use case Diagram....................................................................................14

Figure 3.3: Class Diagram for Doctor Module........................................................................15

Figure 3.4: Class Diagram for Doctor Dashboard.................................................................16

Figure 3.5. Class Diagram for Patient Module.......................................................................17

Figure 3.6: Sequence Diagram for User Interaction with application..................................18

Figure 3.7. User Registration Screen......................................................................................19

Figure 3.8: Login page of the application..............................................................................20

Figure 3.9. Landing screen for the doctor..............................................................................21

Figure 3.10. Landing screen for the patient..........................................................................22

Figure 5.1: User Registration Page.......................................................................................36

Figure 5.2: User Login Page..................................................................................................37

Figure 5.3: User Inbox & Detailed Conversation Pages.......................................................38

Figure 5.4: Doctors Dashboard..............................................................................................39
Figure 5.5: Doctor Adding a Patient.................................................................40
Figure 5.6: Doctor Blocking a Patient.............................................................41
Figure 5.7: User Sending Message..................................................................42
Figure 5.8: Doctor Sending Health Tips...........................................................43
Figure 5.9: Doctor Referring a Patient to other Doctor....................................44
Figure 5.10: Doctor Prescribing medicine to a patient......................................45
Figure 5.11: Doctor Searching nearby pharmacies of patient location.............46
Figure 5.12: Patient Appointments Request....................................................47
Figure 5.13: List of Blocked Patients...............................................................48
Figure 5.14: Payment Tracker for Doctor........................................................49
Figure 5.15: Patient registering with a doctor..................................................50
Figure 5.16: View list of prescription & detailed prescription page....................51
Figure 5.17: View Health Tips Page...............................................................52
Figure 5.18: Patient making an Appointment Request.....................................53
Figure 5.19: Patient Adding Card Details.......................................................54
Figure 5.20: Payment Tracker for Patient........................................................55
Figure 5.21: Sending Email Notifications........................................................56
Figure 5.22: Sending Push Notifications........................................................57
List of Tables

Table 4.1 Code Snippet for User Signup & Registering Device in Parse .................................23
Table 4.2 Code Snippet for User Login Authentication..............................................................24
Table 4.3: Code Snippet for loading User Inbox........................................................................25
Table 4.4: Code Snippet for loading the doctor dashboard with patient requests ............26
Table 4.5: Code Snippet for Adding Patient..............................................................................27
Table 4.6: Code Snippet for Interacting through messaging service......................................28
Table 4.7: Code Snippet for Sending Prescriptions to Patient...............................................29
Table 4.8: Code Snippet for loading the appointment requests from patients...................30
Table 4.9: Code Snippet for Registering with a doctor.............................................................31
Table 4.10: Code Snippet for loading prescriptions sent by doctor......................................32
Table 4.11: Code Snippet for requesting an appointment with doctor...................................33
Table 4.12: Code Snippet for charging per message from patient........................................33
1. BACKGROUND AND RATIONALE

Mobile to overtake fixed Internet access by 2014 was the huge headline summarizing the bold prediction from 2008, and the apps business has finally come, and this worldwide success story is now growing fast. Apps generated an impressive USD 12 billion in full-year 2012, and in total 46 billion apps were downloaded in the year. That is certainly a rapid growth in market. In 2013, about 90 per cent of the mobile phones purchased were smart phones, and between 2012 and 2014, the number of smart phone users over the desktop users is increased by about 0.75 billion. The total usage is now estimated (as of 2014) to be around 1.9 billion (comSCORE 2014) [2].

Figure 1.1 shows that there is a rapid growth in the smart phone users. Smart phone is the combination of Personal Digital Assistance (PDA) and cellular phone. Every smart phone has its own mobile operating system which is more advanced in connectivity and computing capabilities than a traditional mobile phone.

![Image: Global Mobile Users Statistics (comSCORE 2014)](image.png)

Figure 1.1: Global Mobile Users Statistics (comSCORE 2014)
The popular mobile operating systems in the market as per 2015 are

i. Apple’s IOS
ii. Google’s Android
iii. Windows Phone
iv. RIM Blackberry.

This project Doctor Patient Communication System (DPCS) is developed for the Android Operating system. Android is a Linux based operating system, primarily designed for touch screen mobile devices such as smart phones and tablets. Android is compatible with multiple hardware and supports various features like Web browser, Email, Java, Video calling, Media streaming, Bluetooth, Wi-Fi, Multitasking, External storage, Screen capturing, and etc. The reasons for choosing Android operating system for developing the DPCS app are as follows:

i. It is an open source technology with lot of online learning available for zero cost.
ii. Large number of users use Android based smart phones, so that our app can serve more people in the mobile industry.
iii. According to the market survey, it has the highest number of applications available for download on Google play store.
iv. It is also popular among the other operating systems which focuses on low cost, customizable, and a readymade operating system.

Figure 1.2 supports our strategy of choosing Android for developing the DPCS app, because as per the statistics in the Figure1.2, out of total smart phone users there are 47.06% of users using Android based smartphones. So developing an application in Android will be reachable to more users [2].
This app serves the purpose of establishing an online interaction between a doctor and patient. Generally when doctors are not available or not reachable to patients and vice versa, this app will facilitate both the patients and doctors to interact or communicate and seek some help from the doctor regarding patient’s health condition, concerns, etc.

Always it may not be possible for a patient, to meet doctor in hospital/clinic due to some busy schedule and similarly for doctor it may not be possible to give an appointment for the patient to clear minor issues or doubts that the patient has. In such cases it will be helpful to have a medium of online interaction where the patient can quickly ask the doctor some questions.

DPCS will facilitate the doctors with the following features:

i. Registration in the app for doctor so that the doctor details are stored on the Parse web server.

ii. Login to the app to use the services provided by the application.

iii. Sending an invitation to the patients to register with the DPCS app.
iv. A Dashboard which will list all the registration requests from the patients, which has to be approved by the doctor.

v. A doctor can directly add the patients to his catalogue, it is like preapproved and don’t need any further approval from the dashboard.

vi. Doctor can view disease details of all the registered patients with him.

vii. Doctor will be having an inbox on the home screen, which is actually a list of messages from different patients, to which he can reply.

viii. Doctor can send health tips to all the patients.

ix. Doctor can give a prescription to the patients as well as he/she can send a fax to the pharmacies close to the patient location.

x. A doctor can refer a patient to other specialists who are specialized in dealing with a particular disease.

xi. A doctor can update his schedule on his calendar, those details will be updated on the Parse web server and accordingly shown to patients for further appointments.

xii. A doctor can block patients from interacting, so the patient will be inactive, even when the patient tries to send a message, he/she gets a message displaying “you are inactive with this doctor”.

Similarly DPCS facilitates the patients with the following functionalities:

i. Patient is able to register with one or more doctors.

ii. Patients can interact with doctors through messaging services.

iii. Patient is allowed to send only one message per doctor per day, but if the patient gets a reply from the doctor within 24 hours then the patient is allowed to send one more
message. Similarly the communication can go beyond the limit cap as long as the
doctor keeps replying to the patient messages.

iv. Patient is charged for each message sent to doctor, only if he receives a reply for that
message within 24 hours.

v. A message without any reply from doctor within 24 hours will be expired. Doctors
are allowed to reply after 24 hours but patients are not going to be charged for late
replies from the doctor.

vi. A patient can view the prescriptions issued by doctor and collect the medicine from
the pharmacy.

vii. A patient is allowed to view the doctor’s availability on calendar and accordingly can
make the appointment to meet him in clinic/hospital.

1.2. Existing Applications

1.2.1. Epocrates

Epocrates enables patient care by delivering the right information, when it is
needed. It has features such as, find providers for consults and referrals in the Provider
Directory, review drug prescriptions and safety information, check for potentially
harmful drug interactions, access timely medical news and research information, disease
information, alternative medications, lab guides, and more clinical tools. [3]

1.2.2. Doctor @t Work

This application manages patient records, appointments, patient visit notes, bill
patients, track customer payments and balance due. This app can be useful for medical
professionals and students that visit patient every now and then. It also helps the patients
to get the appointments with doctors and sends the reminder through SMS or by email,
create itemized bills for patients to track the due amount, maintains the visit history of the patients, etc. [3]

1.2.3. Diagnotes

Diagnotes is a mobile and web-based software system that gives medical groups the tools to improve physician communication with their patients, care teams, and office staff. User must have a Diagnotes account in order to benefit from this app’s functionality. It routes phone calls and text, supports documentation of every encounter for continuity of patient care. [3]

1.3. Proposed Solution

All the existing applications discussed in section 1.2 are kind of more commercial and money making, but DPCS cares more for patient centered approach, and provides an optimal communication between doctors and patients. As discussed above, DPCS has many beneficial features such as sending health tips to patients, patient referrals, sending prescription to the user selected pharmacy, interacting with the doctor through messages, and getting health tips from the doctor. DPCS also serves the patient needs by allowing them to make an appointment with the doctor and seeing the prescriptions. To develop the DPCS app Android framework, Android Studio and Parse web server are used which are discussed in detail in the next sections.

1.4. Android Operating System

Android is the world’s most popular mobile platform, it powers millions of devices in 190 countries around the world. It is the largest installed base of any mobile platform. Everyday more than one million new Android devices are activated around the
world [1]. It is a platform for creating apps and games, as well as an open marketplace for distributing them instantly through Google’s play store. It is a powerful development framework that gives everything needed to build, and it uses a single application model that lets deploy the apps across wide range of devices such as Phones to tablets and beyond [1].

Google Play is the premier market place for selling and distributing the Android apps, as an open marketplace it also controls how to sell products. Android runs on Dalvik Virtual Machine (DVM) which is hosted on the Linux based Kernel, Android was initially released in September 2008. The different versions of Android OS are as follows [1]:

i. Android 1.0 released in September 2008.

ii. Android 1.1 released in February 2009.

iii. Android 1.5 also called as cupcake released in April 2009.

iv. Android 1.6, also called as Donut released in September 2009.

v. Android 2.0, also called as Éclair released in October 2009.

vi. Android 2.2, also called as Froyo released in May 2010.

vii. Android 2.3, also called as Gingerbread released in December 2010.

viii. Android 3.0, also called as Honeycomb released in February 2011.

ix. Android 4.0, also called as Ice Cream Sandwich released in October 2011.

x. Android 4.1, also called as Jelly Bean released in July 2012.

xi. Android 4.4, also called as KitKat released in October 2013.

xii. Android 5.0, also called as Lollipop released in November 2014.
1.5. Android Studio 1.0.1

Android Studio is the official Integrated Development Environment for Android application development, based on IntelliJ IDEA. On top of the capabilities expected from IntelliJ, Android Studio also offers [4]:

i. Flexible Gradle-based build system
ii. Build variants and multiple APK file generation
iii. Code templates to help you build common app features
iv. Rich layout editor with support for drag and drop theme editing
v. Lint tools to catch performance, usability, version compatibility, and other problems
vi. ProGuard and app-signing capabilities

1.6. Parse Cloud Server

Parse's vision is to let developers build any mobile app without dealing with a variety of servers. For complex apps, sometimes we just need a bit of logic that is not running on a mobile device. Parse Cloud Code makes this possible. Parse Cloud Code is easy to use because it's built on the same JavaScript SDK that powers thousands of apps. The only difference is that this code runs in the Parse Cloud rather than running on a mobile device. When we update cloud code, it becomes available to all mobile environments instantly. This lets to change app behavior on the fly and add new features faster [6].
1.6.1. Features of Parse

The following describes some of the features of Parse [6]:

i. Parse web server handles the data storage securely and efficiently in the cloud. It stores the basic data types such as string, number, etc. It also can store location details and photos, and query across them without spinning up a single server.

ii. Parse connects the users via traditional logins or third party social networks like Facebook, Gmail.

iii. Parse Local Data store helps our app to use all the querying and security features of Parse, even when there is no network connection.

iv. Parse facilitates running background jobs, easily scheduling recurring tasks such as sending emails, updating data, and long running computation at any time interval via the dashboard.

v. Parse uses the web-based interface for Dashboard. The all-in-one Parse Dashboard is a powerful data manipulation tool to view analytics, schedule and send push notifications, and more.

vi. Using Parse, data can be shared between different platforms like iOS, Android, Windows, etc.

Figure 1.3 shows the usage limit of Parse provided for each user. Each user application is supported up to 30 requests per second. A user is allowed to store up to 20GB application data. Data transfer rate on Parse webserver is 2TB and also a user can send up to 1 million push notifications to different users. These limits are for the free version of the Parse.
Figure 1.3: Parse Cloud Server Pricing & Usage Details
2. NARRATIVE

2.1. Problem Statement

In the current marketplace there are several apps for communication between doctors and patients but most of their functionalities are commercial, and not really helpful to the users. There are no apps which actually meet the patient’s basic needs and no such apps that fulfill the patient’s expectations from the doctors. Though these apps have publicized themselves as the best service providers but all the service is just limited to money making by making appointments with doctors, tracking the visit history, patient billings, and very minimal interaction related to disease condition of the patients. This project aims at changing this kind of misleading and unnecessary interaction between doctors and patients, with a new initiative which focuses more on patient benefits relating to his/her disease treatment and facilitating the doctors with an easy and optimal way to treat the patients.

2.2 Motivation

It is quite possible that patient may not remember all the questions which they want to ask a doctor when they meet in the hospital. This app overcomes this problem by providing a messaging service between doctors and patients. This app helps the doctors to send health tips to their patients. Apart from the above, DPCS also facilitates making appointments with the doctor, patient referrals to other specialists dealing with similar patients. Through all these functionalities DPCS fulfills the basic needs of the patients which are totally non-commercial and beneficial to both patients and doctors.
2.3. Product Description

From the previous observations in the section 1.2, it is evidence that there is need for a handy app which will serve the patients and doctors needs at any time. Both patients and doctors need to register with app, doctors do not need any registration approval to start communicating with the patients, but for patient, the registration request goes into the doctor’s dashboard where it has to be approved by the doctor. Once approved the patient can start interacting with the doctor. A patient can send a message to a doctor on a limited basis which is one message per day but it can go beyond that if the patient gets a reply from the doctor. Patient disease description data is maintained on the cloud and always available to the doctor. With the prescription functionality built in this app doctors can send prescriptions to the pharmacy quickly. In this way, DPCS is an easy treatment platform for both doctors and patients.

2.4. Product Scope

This product is developed for smart phone users, as we know how the smart phone market has evolved in the past few years, there are many operating systems available for smart phones but we opted the Android OS for developing this product because it has a very good user bank worldwide. Users can only avail the services of the DPCS when they are connected to internet, because the communication between doctors and patients, and data exchange from cloud server needs the internet connection. DPCS is compatible on different versions of Android, such as starting from the minimum sdk version of Android 3.0 (Honeycomb) to recent update Android 5.0 (Lollipop). The app functions well on the recent update of Android, but we support backward compatibility in view of the other users of Android versions.
3. Proposed Product Design

3.1. Product Design and Architecture

Figure 3.1 describes how communication takes place between doctors and patients. The user sets up the application in their Android phone, both the doctors and patients are differentiated based on their logins. All the interactions between a doctor and a patient such as messaging, sending health tips, sending prescriptions, patient referrals and appointments are initially sent to the parse web server, these interactions are RESTFUL API service requests made by the user to store data on the Parse web server. Similarly the data is fetched from the Parse web server and then through the web server to the user. The web server acts as an intermediate for the interaction between the doctors and patients for data exchange. The users should always be connected to the internet in order to interact with the web server through the web services. Hence this way the communication takes place in the DPCS app.

![Figure 3.1: DPCS Architecture]
3.2. Use Case Diagram

In Figure 3.2, all the ovals represent the functionalities of the doctors and patients, there are some common use cases between doctor and patient such as registration, login, and pharmacy selection are initially made by patient, and otherwise the doctor will be choosing the pharmacy near to the patient location. Patient will be requesting for doctor appointment which has to be confirmed by the doctor, similarly for the messages sent by patients, doctor has to reply and vice versa. Apart from these common features between doctors and patients, doctors also have some specific functionalities like, adding or blocking patient & send health tips, send prescriptions to pharmacies and patient referrals.

Figure 3.2: DPCS Use Case Diagram
3.3. Class Diagram

The Class diagram for the application consists of the interfaces, methods, variables and relationship between them. Figure 3.3 is the class diagram for doctor module which describes the major functionalities of the doctors like registering with the application, login authentication, checking inbox messages, sending messages to patients, prescribing medicines, adding and blocking patients. Both registration and authentication are the common classes for doctors and patients but the screens and functionalities vary based on the attribute role.

Figure 3.3: Class Diagram for Doctor Module
Apart from these functionalities, doctors have some more capabilities while dealing with patients. Every new request from patient comes to doctor’s dashboard for approval from the doctor. Once the doctor accepts the request, the doctor can interact with patient and send prescriptions to patient as well as to pharmacies.

As shown in Figure 3.4 patient interaction and pharmacy classes are dependent on dashboard class because without approving the patients request doctors will not be exposed to interact with patients. In this project the pharmacy interaction is a mock service and we do not deal with the real-time pharmacy services for sending prescriptions. This project is a proof of concept between doctors and patients, but this service is achievable, if we extend the scope of this project to real-time users in the Android market.

Figure 3.4: Class Diagram for Doctor Dashboard
Figure 3.5 is the patient class in which the registration and authentication is quite similar, but the functionalities and screens are rendered based on the role of the user. After login patients have little functionality such as checking inbox for the messages from doctors, sending messages to doctors, view prescriptions from the doctors and, request for an appointment with the doctor to meet in the hospital.

Figure 3.5: Class Diagram for Patient Module
3.4. Sequence Diagram

A sequence diagram describes how the communication happens between the user, application, and cloud data. Figure 3.6 describes the user sequence with the application and how the application handles the user request. The application processes the user request and makes a request to the Parse server to get the data from the cloud database.

![Sequence Diagram for User interaction with application](image)

Figure 3.6: Sequence Diagram for User interaction with application
3.5. User Interaction

The main activities in the application are the user registration page, login page, landing page for doctors and patients after login, landing page for patient after registration. The other remaining modules are implemented using fragments which are described in the section 4 of this document.

In order to register with the application details such as first name, last name, social security number, role, preferred username, password, email, mobile number, and address are required as shown in Figure 3.7.

![Figure 3.7. User Registration Screen](image)
The Figure 3.8 shows the landing page of the application which is the login page with a link “Click here to register” with the application.

![Login page of the application](image)

**Figure 3.8: Login page of the application**

From the screen shown in Figure 3.8 users (In this case a doctor) will be authenticated and directed to the screen shown in Figure 3.9 (a) from where he/she can interact with the functionalities of the application as shown in Figure 3.9 (b).
Figure 3.9 shows the landing screen for the doctor which will be initially showing the inbox, and there is a side navigation bar which provides the remaining functionalities to interact further with the patients. Similar to doctors, patients also have a different landing screen, this differentiation happens based on the login type.
Figure 3.10 shows the landing screen for the patient which will initially show the inbox, where the users most recent received messages are shown, and there is a side navigation bar which provides the remaining functionalities to interact further with the doctors.

Figure 3.10: Landing screen for the patient
4. Implementation of the Application Modules

4.1. Registration & Login

This module provides the functionality to register with the application before they login and use the services available. There will be a link provided in the landing screen for registering with the application. After clicking on the register link, the user navigates to signup activity. In this activity the user has to give details such as first name, last name, social security number, email id, contact number, user role, user name, password, and address as shown in Figure 3.7. All these fields are properly validated. After validations, these details are stored on the Parse web server in “_User” class and the user is navigated back to the landing screen or login activity of the application. Table 4.1 shows the code snippet for Signup [7] & registering device on Parse server for sending push notifications [8].

Table 4.1: Code Snippet for User Signup & Registering Device in Parse

```java
user.signUpInBackground(new SignUpCallback() {
    @Override
    public void done(ParseException e) {
        if (e == null) {
            //Registering device for push notifications
            ParseInstallation registerDevice = ParseInstallation.getCurrentInstallation();
            registerDevice.put("UserName",username.getText().toString());
            registerDevice.put("FirstName",fname.getText().toString());
            registerDevice.put("Email",email.getText().toString());
            registerDevice.saveInBackground();
        }
    }
});
```

After successful registration, the user can login into the application with his/her credentials, the application validates both the username and password given by the user and also checks the role of the user and navigates appropriately to the corresponding screen. If the user is a doctor then he will be navigated to “Doctor_AfterLogin.activity”, otherwise he will be navigated to “home_patient.activity”. Both the users will have different functionalities and features based on their role in the application. Table 4.2 shows the code snippet for user authentication [7].

**Table 4.2: Code Snippet for User Login Authentication**

```java
ParseUser.logInInBackground(edtUserName.getText().toString().trim(),edtPassword.getText().toString().trim(),new LogInCallback() {
    @Override
    public void done(ParseUser user, ParseException e) {
        if(user != null){
            if(user.get("role").toString().equals("Doctor")){
                Intent loginIntent = new Intent(getApplicationContext(),Doctor_AfterLogin.class);
                loginIntent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
                startActivity(loginIntent);
            }else if(user.get("role").toString().equals("Patient")){
                Intent loginIntent = new Intent(getApplicationContext(),home_Patient.class);
                loginIntent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
                startActivity(loginIntent);
            }
        }else {
        }
    }
});
```
4.2. User Inbox

User inbox is a fragment in which all the recent messages are shown with the date & time of that particular message. This functionality is similar for both the doctors and patients, here patient will be receiving messages from different doctors and similarly doctor gets messages from different patients. Apart from this message communication between doctors and patients, some notifications in the application like appointment requests and confirmations are also pushed to the inbox. All the conversations between doctors and patients are stored on the Parse server in “User Messages” class with the sender & receiver first names. Accordingly after login authentication these messages are loaded from the “User_Messages” class. All the conversation between doctors and patients are grouped and the most recent message is shown, when the user clicks on particular item then the complete conversation between two users is shown in the “message_details” fragment. User also has a provision to send a reply from the same grouped conversation page. Table 4.3 shows the code snippet for loading inbox [7].

Table 4.3: Code Snippet for loading User Inbox

```java
ParseQuery<ParseObject> loadInbox = ParseQuery.getQuery("User_Messages");
loadInbox.whereEqualTo("MessageTo", ParseUser.getCurrentUser().get("fname"));
loadInbox.orderByDescending("createdAt");
```

4.3. Doctor Dashboard

Dashboard is the page in which all the requests from the patients are shown, these are registration requests from a specific patient, these requests need to be approved by the corresponding doctor. If the request is approved then a push notification is sent to the patient and further the application allows the patient to interact with that doctor. A doctor can also reject
these requests. These requests from the patients are stored in “Patient_Requests” class on Parse server. Table 4.4 shows the code snippet for loading the doctor dashboard with patient requests [7].

**Table 4.4: Code Snippet for loading the doctor dashboard with patient request**

```java
final ParseObject Pat_Req = new ParseObject("Patient_Requests");
Pat_Req.put("PatientName", ParseUser.getCurrentUser().get("fname").toString());
Pat_Req.put("PatientPrefName", edtPatPrefName.getText().toString());
Pat_Req.put("PatientPrefEmail", edtPatPrefEmail.getText().toString());
Pat_Req.put("PatientSSNID", edtPatId.getText().toString());
Pat_Req.put("PatientPrefMobile", edtPatPrefMbl.getText().toString());
Pat_Req.put("DiseaseDescription", edtPatProb.getText().toString());
Pat_Req.put("DoctorName", docList.getSelectedItem().toString());
Pat_Req.put("Status","Pending");
```

### 4.4. Add/Block Patient

Doctors can add patients from the application by entering the patient details in the application, at which point an email is sent to patient with the invitation from the doctor and a link to download the application. Once the patient install the application the patient has to sign up with the application. After sign up this patient will be having direct access to the doctor who has added him/her. In this case when a patient is added directly by a doctor they will get default access to the doctor and can start interacting with the doctor without sending any request. When a doctor adds a patient all the details are stored in “Patient_Requests” class on the Parse server, by default with the status as “Approved”. Table 4.5 shows the code snippet for adding patients [7].
Table 4.5: Code Snippet for Adding Patient

```java
ParseObject addPatientRequestClass = new ParseObject("Patient_Requests");
String currentUser = ParseUser.getCurrentUser().get("fname").toString();
addPatientRequestClass.put("DoctorName", ParseUser.getCurrentUser().get("fname").toString());
addPatientRequestClass.put("PatientName", edtPatientFName.getText().toString());
addPatientRequestClass.put("PatientPrefName", edtPatientFName.getText().toString());
addPatientRequestClass.put("PatientPrefMobile", edtPatientMbl.getText().toString());
addPatientRequestClass.put("PatientPrefEmail", edtPatientEmail.getText().toString());
addPatientRequestClass.put("PatientSSNID", edtPatientSSN.getText().toString());
addPatientRequestClass.put("DiseaseDescription", edtPatientDiseaseDescription.getText().toString());
addPatientRequestClass.put("Status", "Approved");
addPatientRequestClass.saveInBackground();
```

Blocking a patient could arise in some situations such as, after the treatment is done, irritating the doctors with the unnecessary questions/appointment requests or it could be when the doctor is not able to deal such cases. If a doctor blocks a particular patient then that patient will be inactive with the specific doctor and further he/she will not be able to interact with the specific doctor. Every doctor will be having the provision to block and see the list of blocked patients.

4.5. Interaction through messages

In the application there is a feature provided for both the doctors and patients to start a new conversation. This feature is same for both the users, where the user can select the recipient and, then type the message to send. Ideally for the doctors the receivers will be the patients registered with him and for the patients receivers will be the doctors with whom they are registered. Every user is allowed to send message to only one recipient at a time. All the messages are stored in “User_Messages” class (In Parse a class is similar to a sql table which
stores the data) on Parse web server. There are few restrictions imposed on the patients in sending the messages to doctors such as, a patient can only send one message to a doctor in 24 hours. However if the patient gets a reply within 24 hours then they will be allowed to send the next message to the doctor. A patient is normally charged $5.00 for each message, however this can be waived off by the doctor. Patients are not allowed to start new conversation until they enter their credit card details. The payments functionality is discussed in detail in the Payment Tracker section. Table 4.6 shows the code snippet for interacting through messages [7].

**Table 4.6: Code Snippet for Interacting through messaging service**

```java
final ParseObject user_messages = new ParseObject("User_Messages");
user_messages.put("MessageFrom", ParseUser.getCurrentUser().get("fname");
user_messages.put("MessageTo", docSendList.getSelectedItem().toString());
user_messages.put("message", edtMessage.getText().toString());
```

4.6. Patient Referrals

This feature is purely intended to the doctors, when a doctor is not able to deal the patient disease and he/she knows the other doctors using DPCS and specialized in dealing such cases, then in such scenarios the doctors have a provision to refer the patients to another doctor or specialist. When the doctor refers a patient then the patient details are stored in “Patient_Requests” class on Parse server as a new request with the status as pending with the other doctor who needs to approve the referee. This referral request will be shown in the other doctor’s dashboard with the referral message from the doctor who referred the specific patient. The code implementation is similar to registering with doctors as discussed in the section 4.4.
4.7. Prescribe Medication

In this module a doctor can prescribe medication to patients. A notification is sent to the patients and at the same time the prescription is sent to the nearby pharmacy selected by the doctor through fax service (currently all the prescriptions go to a pre-saved fax number). There is also a filter option to select the pharmacy, currently the app supports different pharmacies such as, Walgreens, H-E-B, Walmart, and CVS pharmacy. Initially a doctor should select the patient and, then he/she should pick a pharmacy and search for nearby pharmacies. All the pharmacies nearby 5 miles radius are shown on a map to the doctor from which he can pick any one of the pharmacy for sending the prescription. All the prescriptions are stored in “Patient_Prescriptions” class on Parse web server. Accordingly the prescriptions are shown to the patient. The patient can track the list of all the prescriptions sent to him from different doctors with whom he has registered. Table 4.7 shows the code snippet for sending prescriptions to pharmacies & patients [7].

Table 4.7 (a): Code Snippet for sending prescription to pharmacy

```java
Intent emailReferralIntent = new Intent(Intent.ACTION_SEND);
emailReferralIntent.setData(Uri.parse("mailto:"));
emailReferralIntent.setType("text/html");
emailReferralIntent.putExtra(Intent.EXTRA_EMAIL, new String[]{"+13618252795@metrofax.com"});
emailReferralIntent.putExtra(Intent.EXTRA_CC, new String[]{parseObjects.get(0).get("PatientPrefEmail").toString()});
emailReferralIntent.putExtra(Intent.EXTRA_SUBJECT, "New Prescription Service Request!!");
emailReferralIntent.putExtra(Intent.EXTRA_TEXT, Html.fromHtml(new StringBuilder()
        .append("<p>" + txtPharLoc.getText().toString() + "</p>"
        .append("<p>Patient Name:" + spinPatList.getSelectedItem().toString() + "</p>"
        .append("<p>" + edtPresc.getText().toString() + "</p>").toString()));
```
Table 4.7 (b): Code Snippet for Sending Prescription to Patients

```java
ParseObject patPresc = new ParseObject("Patient_Prescriptions");
patPresc.put("PatientName", spinPatList.getSelectedItem().toString());
patPresc.put("DoctorName", ParseUser.getCurrentUser().get("fname").toString());
patPresc.put("PharmacyAdd", txtPharLoc.getText().toString());
patPresc.put("Prescription", edtPresc.getText().toString());
patPresc.saveInBackground();
```

4.8. View Appointments

A doctor can view the appointment requests from different patients, based on his/her availability the doctor can accept the patient’s request or can reject the request. In both the cases a push notification and a message is sent to the patient. When a doctor accepts the appointment from any patient, the application does not allow appointments from other patients to the same doctor without half an hour time gap between two appointments. If another patient tries to send a request for an appointment within the time gap then the application prompts the user with “time slot unavailable, please try again”. All the patient appointments are stored in “patient_appointments” class on Parse server. Figure 4.8 shows code snippet for loading patient appointments [7].

Table 4.8: Code Snippet for loading the appointment requests from patients

```java
ParseQuery<ParseObject> getPatAppointmentReq = ParseQuery.getQuery("patient_appointments");
getPatAppointmentReq.whereEqualTo("DoctorName", ParseUser.getCurrentUser().get("fname"));
getPatAppointmentReq.whereEqual("Status", "Pending");
getPatAppointmentReq.orderByDescending("createdAt");
```

The database is queried for all appointments on pending status with specific doctor name.
4.9. Register with Doctor

This module is specific to the patients, ideally when the patients want to interact with any doctor they need to register first with that specific doctor. This patient requests goes to doctor’s dashboard, these requests need approval from the doctor for the patient to interact further with the doctor. A doctor can approve or reject the request from the patient, in both the cases the patient will be notified about the status of his request sent to the doctor. DPCS will not allow multiple requests from the patients with the same doctor. All the patient requests are stored in “Patient_Requests” class on Parse web server. Figure 4.9 shows the code snippet for registration with a specific doctor [7].

Table 4.9: Code Snippet for Registering with a doctor

```java
final ParseObject Pat_Req = new ParseObject("Patient_Requests");
Pat_Req.put("PatientName", ParseUser.getCurrentUser().get("fname").toString());
Pat_Req.put("PatientPrefName",edtPatPrefName.getText().toString());
Pat_Req.put("PatientPrefEmail",edtPatPrefEmail.getText().toString());
Pat_Req.put("PatientSSNID", edtPatId.getText().toString());
Pat_Req.put("PatientPrefMobile",edtPatPrefMbl.getText().toString());
Pat_Req.put("DiseaseDescription",edtPatProb.getText().toString());
Pat_Req.put("DoctorName", docList.getSelectedItem().toString());
Pat_Req.put("Status","Pending");
```

4.10. View Prescriptions

A patient can view his/her prescriptions sent by the doctors. Every prescription contains the information about the doctor, pharmacy from where the patient can collect the medications. This feature also serves as a prescription tracker for the patient. These details are loaded from the
“Patient_Prescriptions” class on Parse server. Table 4.10 shows the code snippet for loading prescriptions sent by the doctor [7].

Table 4.10: Code Snippet for loading prescriptions sent by doctor

```java
ParseQuery<ParseObject> getPatPresc = ParseQuery.getQuery("Patient_Prescriptions");
getPatPresc.whereEqualTo("PatientName", ParseUser.getCurrentUser().get("fname");
getPatPresc.orderByDescending("createdAt");
getPatPresc.findInBackground(new FindCallback<ParseObject>() {...
```

4.11. View Health Tips

A patient can view health tips sent by the doctors with whom they are registered. This feature helps the patients to follow some good tips on daily basis if received from doctors and stay healthy. All the health tips are stored in “Health_Tips” class on Parse server.

4.12. Request for Appointment

A patient can request for an appointment with the doctor. In order to schedule an appointment with the doctor, first the patient should select a doctor with whom he/she would like to make an appointment to meet in the hospital. The patient then has to pick a date & time and should confirm the appointment. If there are no appointments already scheduled for the doctor with the other patients within half an hour time gap, then the patient appointment request will be sent to the doctor. Otherwise the patient will be notified to try again with a different time or date to schedule an appointment with the doctor. A patient appointment can be accepted or rejected by the doctor, in both the cases the patient will be notified through push notifications and messages. DPCS will not allow scheduling multiple appointments from a patient with the same doctor until the previous request has been served by the specific doctor. All the appointment
requests are stored in “patient_appointments” class on Parse server. Figure 4.11 shows the code snippet for requesting an appointment with the doctor [7].

**Table 4.11: Code Snippet for requesting an appointment with the doctor**

```java
reqAppointment = new ParseObject("patient_appointments");
reqAppointment.put("PatientName", ParseUser.getCurrentUser().get("fname").toString());
reqAppointment.put("DoctorName", spinDocsList.getSelectedItem().toString());
reqAppointment.put("PatientEmail", ParseUser.getCurrentUser().get("email").toString());
reqAppointment.put("RequestedDate", selectedDate);
reqAppointment.put("RequestedTime", selectedTime);
reqAppointment.put("Status", "Pending");
```

**4.13. Payment Tracker**

Every patient is charged an amount of $5.00 for a reply message from the doctor. However doctors have the provision whether to charge or not while replying to a message from the patient. The payment details module is similar to doctors & patients and the payments received or made are shown to them accordingly. From the doctor’s perspective DPCS will show the payments received from the different patients. Similarly for the patient, total payments made by the patient to different doctors is shown. Every patient has to give the credit/debit card details before sending messages to doctors, else the application does not allow them to send messages. All the amount charged from the patients is stored in “Message_Charges” class on Parse server. Figure 4.12 shows the code snippet to charge $5.00 from the patient for a reply message [7].
Table 4.12: Code Snippet for charging per message from patient

```java
ParseQuery<ParseObject> getMsgCharges = ParseQuery.getQuery("Message_Charges");
getMsgCharges.whereEqualTo("PatientName",docSendList.getSelectedItem().toString());
getMsgCharges.whereEqualTo("DoctorName",ParseUser.getCurrentUser().get("fname").toString());
getMsgCharges.findInBackground(new FindCallback<ParseObject>() {
    @Override
    public void done(List<ParseObject> parseObjects, ParseException e) {
        if(e == null){
            if(parseObjects.size() > 0){
                int total = Integer.parseInt(parseObjects.get(0).get("AmountCharged").toString()) + 5;
                parseObjects.get(0).put("AmountCharged",total+"");
                parseObjects.get(0).saveInBackground();
            }else{
                ParseObject chargeMsg = new ParseObject("Message_Charges");
                chargeMsg.put("PatientName",docSendList.getSelectedItem().toString());
                chargeMsg.put("DoctorName",ParseUser.getCurrentUser().get("fname").toString());
                chargeMsg.put("AmountCharged","5");
                chargeMsg.saveInBackground();
            }
        }else{
        }
    }
});
```
5. Testing and Evaluation

In this phase the functionality is tested on Nexus 7 tablet which is an Android device with Android Version 4.4.4. To install apk we need to enable an option in the device settings to allow installations from other locations. The apk supports minimum Android version 3.0 to Android version 5.0. To use the application device should be connected to WIFI with good connection.

This chapter deals with testing each module of the application with positive and negative test cases, first let’s start from registering with application.
5.1. Registration

a. **Negative Test Case:** Clicking on Register button without entering values in any of the fields and trying to register with the application will generate a dialog by showing registration or validation failure as in Figure 5.1 (a).

b. **Positive Test Case:** Figure 5.1 (b) shows the successful registration, by which we ensure the user is registered successfully with the DPCS app. Before registering the user, we make sure all the fields are validated.

![Figure 5.1: User Registration Page](image-url)
5.2. Login

a. **Negative Test Case:** From Figure 5.2 (a) it can be seen that, if we enter wrong credentials and try to login, then an error message displaying “Invalid Login Parameters” is shown.

b. **Positive Test Case:** By entering valid credentials and clicking on the Login button user is authenticated and navigated to the Inbox screen.

![Figure 5.2: User Login Page](image-url)
5.3. Inbox

a. **Negative Test Case:** When the user logs in for the first time, and if there are no messages initially, the app displays a message “Your Inbox is Empty” as shown in Figure 5.3 (a).

b. **Positive Test Case:** Figure 5.3 (b) shows the messages from two different users. By clicking on any one of the list item, then the detailed conversation between them is shown. In the detailed conversation page, blue color text indicates the messages received and red color text indicates the messages sent.

![Figure 5.3: User Inbox & Detailed Conversation Pages](image)
5.4. Doctor Dashboard

a. **Negative Test Case:** When there are no pending requests in the dashboard, normally it will display a message “No Pending Requests” as shown in Figure 5.4 (a).

b. **Positive Test Case:** When there is a request from the patient, it will be similar to as shown in Figure 5.4 (b). The doctor can either approve and rejects the patients requests by pressing on the appropriate buttons. If the doctor approves the request then the patient will be able to interact further with the doctor. Similarly if the patient request is rejected, patient will not be able to interact with the doctor.

![Figure 5.4: Doctors Dashboard](image)
5.5. Add Patient

a. **Negative Test Case:** If the doctor tries to add a patient without entering all the values then a validation error dialog is displayed as shown in Figure 5.5 (a).

b. **Positive Test Case:** After entering all the fields and clicking on save as shown in Figure 5.5 (b), the patient will be registered with that specific doctor and an invitation email will be sent to the patient. These kind of requests are automatically approved and does not require further approval from doctor’s dashboard.

![Figure 5.5: Doctor Adding a Patient](image-url)
5.6. Block Patient

a. Negative Test Case: When the doctor clicks on block button without selecting patient or without typing the block reason then an error dialog is displayed as shown in Figure 5.6 (a).

b. Positive Test Case: The doctor can block a patient by selecting him from list of patients, provide a reason and clicking on the block button. A dialog will be displayed confirming the action as shown in Figure 5.6 (b). Once blocked the patient will not be able to send any further messages to doctor.

![Figure 5.6: Doctor Blocking a Patient](image-url)
5.7. Send Message

a. **Negative Test Case**: When the user tries to send a message without selecting the recipient or typing the message, then an alert dialog will be displayed as shown in Figure 5.7 (a).

b. **Positive Test Case**: For sending a message the user can select a recipient, type a message and click on the send button as shown in Figure 5.7 (b).

![Figure 5.7: User Sending Message](image)
5.8. Send Health Tips

a. **Negative Test Case:** When the doctor tries to send health tips without typing the message, then an alert dialog will be displayed as shown in Figure 5.8 (a).

b. **Positive Test Case:** When the doctor types the health tip and clicks on send, then that health tip will be sent to all their registered patients as an email as shown in Figure 5.8 (b).

![Figure 5.8: Doctor Sending Health Tips](image-url)
5.9. Patient Referral

a. **Negative Test Case:** When doctor tries to refer a patient without selecting any of the fields an alert dialog is displayed as shown in Figure 5.9 (a).

b. **Positive Test Case:** The doctor has to first select a specialist and then select the patient to whom he/she wants to refer, then type a referral message, and click on the refer button to send a request to the specialist as shown in Figure 5.9 (b), this request will be populated in other doctor’s dashboard.

![Figure 5.9: Doctor Referring a Patient to other Doctor](image)
5.10. Prescribe Medication

a. **Negative Test Case:** When the doctor clicks on search locations without selecting a patient an alert dialog is displayed as shown in Figure 5.10 (a). Also when the doctor clicks on send prescription without selecting a patient and a pharmacy an alert dialog is shown as Figure 5.10 (a).

b. **Positive Test Case:** To send a prescription the doctor has to select the patient and nearby pharmacy of the patient’s address as shown in Figure 5.10 (b). By clicking on the send button a prescription is sent to the patient and to the pharmacy fax machine (Currently we are using pre-saved fax number for all the prescriptions).

![Figure 5.10: Doctor Prescribing medicine to a patient](image)

(a)  
(b)
5.11. Searching nearby locations on Google maps

a. **Negative Test Case:** When the doctor clicks for nearby pharmacies of the patient address, without selecting the patient, an alert will be shown to select a patient before searching for nearby pharmacies as shown in Figure 5.11 (a).

b. **Positive Test Case:** To select the nearby pharmacy, the doctor selects the patient and clicks on search, all the pharmacies within 5 miles radius of the patient address will be displayed on the map as shown in Figure 5.11 (b). By clicking on the marker shown on the map, the doctor can select the address which is displayed in the header field as shown in Figure 5.11 (c).

![Figure 5.11: Doctor Searching nearby pharmacies of patient location](image-url)
5.12. View Appointments

a. **Negative Test Case:** When there are no appointment requests from the patients, then a message is shown to the doctor as in Figure 5.12 (a).

b. **Positive Test Case:** When there are appointment requests from the patients, the doctor has a provision to accept or reject the requests by clicking on approve or reject buttons as shown in Figure 5.12 (b).

![Figure 5.12: Patient Appointments Request](image)
5.13. View Blocked Patients

a. **Negative Test Case:** When there are no blocked patients, a message will be shown to the doctor as in Figure 5.13(a). When the doctor unblocks any patient then DPCS allows that specific patient to interact with the doctor.

b. **Positive Test Case:** The patients blocked by the doctor are shown as in Figure 5.13 (b), the doctor will be able to unblock any patient by clicking on the unblock button.

![Figure 5.13: List of Blocked Patients](image)
5.14. Track Payments

a. **Negative Test Case:** In the payment tracker screen for doctor as shown in Figure 5.14, the total payment and the sum of individual payments should be matching.

b. **Positive Test Case:** The total amounts charged by a doctor for each patient is displayed in the payment tracker screen as shown in Figure 5.14.

![Figure 5.14: Payment Tracker for Doctor](image)
5.15. Register With Doctor

a. **Negative Test Case:** If the patient tries to register with the doctor without entering all the required fields then an alert dialog will be shown as in Figure 5.15 (a).

b. **Positive Test Case:** The patient should enter the data for all fields as shown in Figure 5.15(b) before clicking on the Send Request button for registering with a doctor.

![Figure 5.15: Patient registering with a doctor](image-url)
5.16. View Prescriptions

a. **Negative Test Case:** When there are no prescriptions from doctors, the app displays a message saying “No Prescriptions” as shown in Figure 5.16 (a).

b. **Positive Test Case:** All the prescriptions from doctors are shown as a list as in Figure 5.16 (b), if the patient clicks on any of the prescription in Figure 5.16 (b) the complete prescription details are shown as in Figure 5.16 (c).

![Figure 5.16: View list of prescription & detailed prescription page](a) (b) (c)
5.17. View Health Tips

a. **Negative Test Case:** When there are no health tips from the doctors, the app displays a message saying “No Health Tips” as shown in Figure 5.17 (a).

b. **Positive Test Case:** Health tips from only registered doctors will be shown to the patients as in Figure 5.17 (b).

![Figure 5.17: View Health Tips Page](image_url)
5.18. Make Appointments

a. **Negative Test Case:** When the patient tries to make multiple appointments with the doctor, the app displays an alert saying that multiple appointment requests are not allowed as shown in Figure 5.18 (a).

b. **Positive Test Case:** To make an appointment with doctor, the patient has to first select doctor, date & time as shown in Figure 5.18 (b). By clicking on the confirm button, the app checks the doctor schedule and ensures that the requested time has at least half an hour gap with the other appointments of the doctor before sending a request for appointment. Otherwise the patient request will not be confirmed.

![Figure 5.18: Patient making an Appointment Request](image)
5.19. Card Details

a. **Negative Test Case:** When there is no payment information added, the patient will not be able to send a message to the doctors and the app will display a message “No Cards have been added yet!” as shown in Figure 5.19 (a).

b. **Positive Test Case:** To start interacting with the doctors the patient has to give valid card details as shown in Figure 5.19 (a). Patients will have a provision to change/delete card details as shown in Figure 5.19 (c).

![Figure 5.19: Patient Adding Card Details](image-url)
5.20. Payment Tracker

a. **Negative Test Case:** As shown in Figure 5.20, sum of the individual payments to doctors and total payments made should match.

b. **Positive Test Case:** The payment tracker gets updated when the patient receives a reply from the doctor. This screen shows the total amount paid to a particular doctor as in Figure 5.20.

![Figure 5.20: Payment Tracker for Patient](image-url)
5.21. Sending Email Notifications

a. Negative Test Case: The doctor will not be able to send any health tips if no email client is configured on his/her device.

b. Positive Test Case: When the doctor sends a health tip an intent is generated asking him/her to select an application of choice as shown in Figure 5.21 (a). The doctor can compose and send an email as shown in Figure 5.21 (b) after selecting a client of his choice.

Figure 5.21: Sending Email Notifications
5.22. Sending Push Notifications

When the patient sends a message or registration request for an appointment, the doctor receives a push notification as shown in Figure 5.22 (a). Similarly when the doctor approves a patient request or appointment or replies to a message then it will be notified to patients as a push notification as shown in Figure 5.22 (b).

![Figure 5.22: Sending Push Notifications](image-url)
6. Conclusion & Future Work

The major goal of this application is to create online interaction between doctors and patients, to fulfill the basic needs/problems of the patients. Patients can update their problem to doctors by messaging and get advice from the doctor. Through the messaging services patients can seek 24/7 help from doctors. DPCS serves as the platform for easy and quick treatment, with extended support from doctors to patients. DPCS has unique features such as issuing online prescription to patients, referring patients to a specialist, sending health tips to patients, and finally, reducing the cost of customer service and providing a vital communication link between doctors and patients.

Future Works

This app can be improved in the future by adding the following functionalities:

- Extending to Apple iOS.
- Create two separate apps for doctor & patient.
- Sharing the medical test reports of patients to doctors through this app.
- Supporting video calls to discuss the problems with doctors.
7. Bibliography & References

http://developer.android.com/training/basics/firstapp/index.html

http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/


http://wptrafficanalyzer.in/blog/gps-and-google-map-in-android-applications-series/


https://parse.com/tutorials/android-push-notifications


https://api.genmymodel.com/