Technical Specification Document

Notifyi

IPhone Application

12 September 2012
Rapid value

Document Information and Approvals

Version History

Version #	Date	Revised By	Reason for change
1	12/09/12	Veena Chandran	Initial draft

Document Approvals

Approver Name	Project Role	Signature/Electronic Approval	Date

TABLE OF CONTENTS

EXECUTIVE SUMMARY

This project deals with development of an iPhone application for Notifyi users to communicate effectively in the network through an email system which works on iPhone.

This document is intended to address the following users;

1. Notifyi/ Redshift team, the end user who is supposed to approve this document
2. RapidValue Development team includes the Project Manager, Engagement Manager, developers and testers

This document discusses the technical functionality of the system. This document will be used by Designers, Developers and Testers.

SECTION 1. INTRODUCTION

This document describes the high-level design for developing an iPhone application for Notifyi users which would help its clients [doctors] to manage their patients and related task on the move.

1.1 Purpose

The purpose of the system under development is to develop an iPhone application that would enable the physicians in the Notifyi network to send/receive messages , touch base, view a physician's profile, view his/her schedule etc.

The proposed system is intended to be deployed on iPhone 4.x and iPod 4.x, while users will be able to install and run the application in iPad but the UI will remain same as the iPhone.

1.2 Scope

The following functionalities cover the scope of the application:

- ➢ The application is only targeted for "Apple iOS iPhone only"
- ➢ Only the existing users of the Notifyi web application can use the iPhone app.
- ➢ The web services required for the application shall be developed by Notifyi/Redshift.
- ➢ The management of users is not part of the app and will be an external web application.
- ➢ The iPhone application will have the following modules

SECTION 2. FUNCTIONAL DETAILS

Login:

- ➢ User can login by providing user name and password
- ➢ If the user is an authorized, then the server will response with success status.
- ➢ If the user is not an authorized, then the server will response with failure status.
- ➢ A user with success login can enter into the main screen of the application.
- ➢ 'Forgot Password?' and 'Contact Us' link will redirect to the corresponding webpages.

Main Menu:

- The following menus will appear in the bottom tab bar

  - Inbox
  - Touch Base
  - Directory
  - More (My profile, coverage calendar, mobile settings)

- On clicking any of the above items users will navigate to corresponding pages.

1. Inbox:

- The Inbox tab contains all of the messages the user has sent and received.
- The messages are displayed in the chronological order where the latest messages are displayed first.
- It allows the users to view and manage any incoming and outgoing messages.
- All items in any folder will be removed after 30 days.
- If there is any unread message in the inbox, it will be notified in the inbox icon on the bottom bar.
- Users can read a single message by clicking on that message.
- A user can navigate to any of the folders like **Sent items**, **Drafts** and **Trash** by clicking on the menu which is located on the top bar.

1.1 Sent item folder:

When the user clicks on sent items, all send messages are displayed similar to the inbox.

There is also an option to delete that particular sent item.

1.2 Draft folder:

- When the user clicks Draft, all of their drafts or uncompleted messages are displayed in a paginated list similar to Inbox Folder.
- Viewing Messages from the Draft Folder, the user is directed to the partially filled out “Send a Message” form.

1.3 Trash Folder

- When the user clicks on Trash, all deleted messages are displayed similar to the inbox.
- Details will be displayed when a user selects a message from the trash folder list.
- Details page contains a “Restore” button to restore that particular message.

Inbox Details:

- On clicking a message from the inbox, the user can read that message.
- From this screen the user can reply, reply all, forward and delete the message.
- On clicking on the “To” header the user can see all the recipients of that message.
- On swiping from one message screen, the next message will be loaded .

Compose mail:

- User will be able to compose mail by clicking on the button which is also located at the top bar.
- It contains a standardized form that provides needed information to effectively communicate information about a patient.
- On clicking on the patient information, the form will expand to receive the patient information like patient name and date of birth.

Search option:

- There is a search option which will help the users to get an item quickly.

Delete option:

- And users can easily delete mails from the list.

2. Touch Base :

- Touch Base allows physicians to have ongoing group conversations about patients.
- This section is intended to allow physician to easily collaborate with a number of physicians at once about a shared patient.
- When the user clicks on Touch Base, the latest discussions are displayed in the main view.

- The user can click the “Start a Discussion” button on the top to start a new discussion.

Discussion Details:

- While selecting a particular discussion from the list, we can see the details about that discussion.
- In the discussion details page,there is a text field which helps the user to add comments.
- The user can also stay away from that particular discussion by clicking “Remove me” button.
- And also user can easily add participants to that discussion.

Start new discussion:

- It contains a standardized form that provides needed information to effectively communicate information about a patient.
- When the user clicks “Send“, the discussion will be send.

3. Sender Resolution :

"Sender Resolution" is Notifyi 's method for confirming the delivery of messages and notification to users of those messages. This displays the following columns:

- “To” - displays a recipient (may be multiple recipients, one entry per row).
- “Route” - displays either one or two of the following: Fax In Transit, Fax Delivered, Fax Failed, Notifyi Sent, Notifyi Opened, Notifyi Failed.
- “Alert” - describes how the physician received the message as any or all of the following: Pager, Email, Text.
- “Notified” - shows the date and time of notification.
- Coverage - there is no header for this column but each row will display Coverage or will be

blank.

4. Directory :

- Directory contains list of physicians with name and speciality.
- Users can easily select a particular physician from the list and can navigate to its details page.
- There is an option to search for a particular physician from that list.
- And also an option to delete a particular physician.

Directory Details :

While clicking on a particular physician from the directory user will navigate to the details page which includes the following data.

- Name
- Practice
- Specialty
- City
- State
- Status
- Phone

There will be an option to send text message and touch bases from this screen.

5. More:

There are mainly three sections comes under More menu.

- My Profile
- Coverage Calendar
- Mobile settings

5.1 My Profile:

-
- From the my profile menu, the user can see his/ her profile.
- However, for changing the profile the user needs to login to the web version.

- From the profile screen, the user can go back to the 'more' menu on clicking the back button.

5.2 Coverage Calendar :

- The coverage calendar will be a read only page which displays the schedule of the particular user.
- The calendar will display the schedule of that user for 2-3 weeks which can be scrolled and seen.
- On clicking a calendar entry the detailed page will appear.
- The addition or update of a calendar entry can be done only at the web side.

5.3 Mobile settings

The users mobile preferences can be set from the mobile settings page. This page will have two sections

- Mobile alert settings

The message alerts like the vibration alert and the ring on message are set from the mobile alert settings section.

- Push Notifications

From the push notification section we can turn the push notifications on or off.

SECTION 3. DATABASE DESIGN

INBOX:

Field Name	Data Type	Default Value
messageId	Integer 32	
MessageType	Integer 32	SentItem – 0 InboxItem – 1 DraftItem – 2 TrashItem – 3
ReadStatus	Integer 32	Read -0 Unread – 1

Date	Date	
subject	String	
TextMessageBody	String	
patientFirstName	String	
patientLastName	String	
patientDOB	Date	

TOUCH BASE:

Field Name	Data Type	Default Value
discussionId	Integer 32	
TextDiscussion	String	
patientName	String	
comments	String	

MESSAGE RECIPIENTS:

Field Name	Data Type	Default Value
UserName	String	
UserId	Integer 32	
MessageId	Integer 32	
messageType	Integer 32	SentItem – 0 InboxItem – 1

DISCUSSION PARTICIPANTS:

Field Name	Data Type	Default Value
participantName	String	

discussionId	Integer 32	

COMMENTS:

Field Name	Data Type	Default Value
discussionId	Integer 32	
commentsId	Integer 32	
comments	String	
CommentStatus	Integer 32	ReadComment – 0 unreadComment – 1

DIRECTORY:

Field Name	Data Type	Default Value
DoctorName	String	
DoctorId	Integer 32	
practice	String	
Speciality	String	
city	String	
state	String	
status	String	
phone	String	

MY PROFILE:

Field Name	Data Type	Default Value
userName	String	
speciality	String	
hospitals	String	

practice	String	
contactInfo	String	

COVERAGE CALENDAR:

Field Name	Data Type	Default Value
date	Date	
startTime	String	
endTime	String	
title	String	
details	String	

TIME STAMP:

Field Name	Data Type	Default Value
operationType	Integer 32	Inbox – 0 touchBase – 1 directory – 2 myProfile – 3 coverageCalendar – 4
lastUpdatedDate	Date	

SECTION 4. API INFORMATIONS

First syncing will happen when the app get launched. Then the syncing will continue when each view gets loaded. The user has the option to refresh the content manually.

While the app is in background, the user will get a push notification alert (if there any updates).For this the push notifications must be enabled. Here the user has the option to enable or disable push notification in Notifyi settings inside More tab. By default this will be off and we will provide an alert to user to enable the push notification by switch on the notification in application settings tab. This setting will tell the server whether to send the push notifications to this user. Besides this, the push notification should be enabled in the application specific settings in the iPhone as well. User will get the notification only if it is enabled in the device settings.

1. **Login API details:**-

input : username, password, operationType

operationType : Which is the name of the api call e.g: LoginDetails, InboxDetails etc. The same should be returned with the output for the confirmation. And this is common with all API input and output.

The operation types for the various API calls are mentioned below

Login -1
Inbox-2
Read Message - 3
Delete a message-4
Restore a message from trash-5
Compose-6
Touch base-7
Remove Me-8
Add Participant-9
Start Discussion-10
Directory -11
Delete a physician from directory-12
My Profile-13
Coverage Calendar -14
Sender Resolution - 15

output :

- service response – Service response is common with all API responses. It contains three following fields,

1. Response code : 600 – success, 604 – failure and 605 - internal error.
2. message : details about that service response. e.g. : for 604 response code, the message will be some thing like “Entered information is wrong”.
3. operationType : Should be same as the input operationType.

- Details – Contains userId, userName, operationType.

The request format will be like

{"UserName":"name","Password":"password","operationype":1}

1. UserId : - to uniquely identify each user.
2. UserName : Name of the user.

Eg : -

```
            {
  "service response":
  {
       "response code": "600",
        "Message": “Login Success”,
       “operationType” : “1”
   },

  "Details":
  {
     "UserId" :  “46t5678j”,
       “UserName” : “John Von”,
   }
          }
```

2. **Inbox API details:**

PageNumber and LastModified fields are used for getting the synching details

INPUTS :

1. UserId : An id returned from the server when a user is successfully logged in. e.g : “46t5678j”

1. OperationType : Same as above.

2. lastUpdatedDate :

In the first syncing case, this date will be a date at 30th day before the current date. Next time onwards, will use a date which is the locally stored from last response. The last synched date from the server will be stored locally and the same will be sending along with the next request. If there is no latest updates from server side, the response should contain "lastUpdatedDate" same as the input date. Otherwise, it should return another date which means there is an updates. (same as the mechanism called **time stamp**). This "lastUpdatedDate" should return back with output.

3. PageNumber :

We will request a particular amount of data (say 15) from the server. You can determine the amount of data for a single transfer.

when the "pageNumber" is 1, the response should contains first 15(say) data. if the page number is 2, response should be next 15 data. so on.

if the timer calls when the synching is in progress, it will automatically kill that call. When syncing starts it will show a label with text “Syncing In Progress..” and an indicator on the top of the screen.

The **Inbox** API response should contains the following details as output.

- service response – Same as above.
- Details - It contains following fields

1. senderName – message sender name.
2. Recipients – may contains an array of recipientNames.
3. Subject – message subject.
4. Date – message received date.
5. MessageId – a unique identifier for a message.
6. MessageType – returns an integer (i.e. SentItem - 0, InboxItem – 1, DraftItem - 2 and TrashItem – 3)
7. TextMessageBody – contains the body of the message.
8. patientFirstName – contains patient's firstName.
9. patientLastName – contains patient;s lastName.
10. patientDOB – contains patient's DOB.

11. balanceCount – In this Notifyi iPhone application synching,at a time we take only 15 (say) responses from the server in order to avoid the lagging. In such case we have to keep a count of balance response. In the first case, balance count will be total number of responses or data (say 100). In the second time, the balance count will be decremented by 15 (i.e. 100-15=85).

12. lastUpdatedDate - It helps to recognize whether there is any updates or not. (synching is needed or not). if there is no updates, the lastUpdatedDate should be same as the input. Else any other date.

13. ReadStatus : Returns an integer value (i.e. Read – 0 and Unread – 1). Which helps to identify whether the mail is read or not.

Eg: -

```
{
   "service response":
   {
      "response code": "600",
          "Message": “”,
        “operationType” : “2”
   },

{
   "Details": [
   {
        "senderName": "John Von",
        "Recipients": [{“RecipientName” : “Recipient1”,
“RecipientId”:”1”},{ “RecipientName” :Recipient2, “RecipientId”:”2”}],
        “Subject” : “Meeting”,
          “ReadStatus” : “0”,
          “TextMessageBody” : “Sample body”,
          “patientFirstName” : “Francis”,
          “patientLastName” : “John”,
          “patientDOB” : “02/04/1985”,
          “Date” : “02/09/2012”,
          “MessageId” : “001”,
          “MessageType” : “0”,
          “balanceCount” “ “100”,
          “lastUpdatedDate” : “02/09/2012”
    }
             ]
}
}
```

3. Read Message API :

This API will fire when user reads a particular message. (i.e. When clicking on a particular message from the inbox list). The inbox API output should contain the ReadStatus.

input : userId, operationType, MessageId , MessageType, ReadStatus (an integer i.e. read - 0 and unread 1).

output : service response, MessageType,MessageId

Eg :

```
{
   "service response":
   {
      "response code": "600",
         "Message": “Successfully Read the message”,
        “operationType” : “3”
   },
    "Details":
    {
         "MessageType" : "0",
         "MessageId" : "M001"
     }
}
```

4. Read Comment API :

This API will fire when user reads a particular comment. (i.e. When clicking on a particular discussion from the touch Base list). The TouchBase API response should contain the commentStatus (read or not).

input : userId, operationType, discussionId , commentsId, ReadStatus (an integer i.e. read - 0 and unread 1).

output : service response, discussionId , commentsId

Eg :

```
{
   "service response":
   {
      "response code": "600",
         "Message": “Successfully Read the comment”,
        “operationType” : “4”
   },
    "Details":
    {
         "MessageType" : "0",
         “discussionId”  : “D001”,
         "MessageId" : "C001"
     }
}
```

5. Delete a message from Inbox :

input : userId, messageId, messageType, operationType

output: service response,messageId

6. Restore a message from Trash :

input : userId, messageType, messageId, operationType

output: service response, messageId

7. Compose :

input : userId, lastUpdatedDate, operationType, sendersName, patientFirstName, patientLastName, patientDOB, subject, TextMessageBody

output : service response , messageId

8. Touch Base :

input : userId, lastUpdatedDate, operationType, PageNumber

output : service response + ParticipantsName, ParticipantsId, patientName, patientId lastUpdatedDate, TextDiscussions, discussionId, comments, commentsId, commentStatus, balanceCount

This section is actually intended to allow physician to easily collaborate with a number of physicians at once about a shared **patient**.

ParticipantsName : List of physicians who are actively participating in that discussion.

PatientName : should contains the name of the patient.

CommentStatus : Returns an integer value (i.e ReadComment – 0 and unreadComment – 1)

Eg:

```
{
   "service response":
   {
      "response code": "600",
          "Message": “”,
        “operationType” : “8”
   },
{
   "Details": [
   {
        "senderName": "John Von",
        "Participants": [{ “ParticipantName” : “Participant1”, “ParticipantId” : “p001” },
{ “ParticipantName” : “Participant2”, “ParticipantId” : “p002” }],
        “patientName” : “John Samuel”,
        “patientId” : “"
          “lastUpdatedDate” : “12/08/2012,
          “TextDiscussion” : “Text message discussion contents....”,
          “discussionId” : “d001”,
          "Comments": [{ “CommentDescription” : “Comment description ..”, “CommentId” : “c001” ,
“CommentStatus” : “0”}, { “CommentDescription” : “Comment description ..”, “CommentId” :
“c002” , “CommentStatus” : “1”}],
          “balanceCount” “ “100”
    }
              ]
}
}
```

9. Remove me:

input : userId, operationType, discussionId

output : service response , discussionId

10. Add Participants:

input : userId, operationType, discussionId, participantsId

output : service response + participantsId

11. Start a discussion:

input : userId, operationType, sendersName, participantsName,patientFirstName, patientLastName, patientDOB, TextDiscussionMessage

output : service response + discussionId

12. Directory :

input : userId, lastUpdatedDate, operationType, PageNumber, searching(1 or 0) (will be null if the user need not using search feature)

Output : service response , physicianName, physicianId, practice, speciality, city, state, status, phone, balanceCount

Eg:
```
{
   "service response":
   {
      "response code": "600",
          "Message": “”,
         “operationType” : “12”
   },

{
   "Details": [
   {
        "physicianName": "Adelsberg, Michael P.A.",
        “physicianId” : “p001”,
        “practice” : “Connecticut Medical Group”,
        “speciality” : “Internal Medicine”,
        “city” : “Guilford”,
        “state” : “CT”,
        “phone” : “212-683-8993”,
        “balanceCount” “ “100”
    }
            ]
}
}
```

On searching a directory, the lists will be saved to the local database and next time when the search is done, it searches the local database first and then the server (api). This will help to search the recently used items before synching the new items from the server

13. Delete physicians from Directory:

input : userId, operationType, physicianId

Output : service response + physicianId

14. My Profile :

input : userId, operationType

output: service response , userName, speciality, hospitals, practice, contactInfo,

15. Coverage Calendar :

input : userId, operationType

output : service response , date, startTime, endTime, title, details

16. Push Notification

This will fire when user click on Login Button.

input : userId, operationType, deviceToken

output : service response

17. sender Resolution API

input : userId, operationType

output : service response ,recipients, route, Alert, notified, coverage.

Eg :

```
        {
   "service response":
   {
      "response code": "600",
          "Message": "",
        "operationType" : "17"
   },

{
   "Details": [
   {
        "Recipients": [{"RecipientName" : "Recipient1", "RecipientId":"1"},
{ "RecipientName" :Recipient2, "RecipientId":"2"}],
        "route" : "Fax in Transit",
        "Alert" : "Email",
       "notified" : { "date": "12/08/2012","Time" : "08 : 31 : 19 PM" }
        "coverage" : ""
```

```
            } ]
}
}
```

SECTION 5. APN (Apple Push Notifications)

Push notifications which allow messages to be sent directly to an individual device relevant to the application.

Need to communicate with the Apple Push Notification Service (APNS) to send the messages that are then pushed to the phone.

The device needs to maintain a connection to the APNS server

Basic Structure for APNS

Connect to the APNS using a unique SSL certificate.

Construct the payload for each message.

Send message to APNS server.

Disconnect from APNS.

The payload is limited to 256 bytes in total – this includes both the actual body message and all of the optional and additional attributes you might wish to send. Push notifications are not designed for large data transfer, only for small alerts.

For APNS Notification. Mainly need

1 - Device Token

Each push message must be "addressed" to a specific device. This is achieved by using a unique device Token generated by APNS.

2 - Payload Contents

The payload is formatted in JSON. It consists of

Alert – the text string to display on the device

Badge – the integer number to display as a badge by the application icon on the device home screen

Sound – the text string of the name of the sound to accompany the display of the message on the device.

For the implementation we are using third party API (JDSoft).

And also we use a p12 certificate in the server side. So for each APNS Notification we need this p12 certificate. Initially we need to validate p12 certificate by using JDSoft.

NotificationService service = new NotificationService("development/production", "p12 certificate server path", "certificate password", "connection")

When a message and a device token is received in the server, server can create a payload and send this payload and a particular device token to the APNS server by using third party API JDSoft.

Notification alert = new Notification("device token");

JDSoft sample

http://apns-sharp.googlecode.com/svn-history/r21/trunk/JdSoft.Apple.Apns.Notifications.Test/Program.cs

SECTION 6. KEY DESIGN ASSUMPTIONS

- Application would be developed on iPhone SDK 5.0.
- User interface is not designed for iPad and thus user will only be able to install and run the application on iPad but with application screen size same as that of iPhone

SECTION 7. APPLICATION FRAMEWORK

iPhone SDK provides a collection of frameworks for accomplishing our development activities like UIKit, messageUI, CoreData etc. The application would be developed with the aid of these frameworks. The architectural pattern used in designing these frameworks is the Model View Controller [MVC] architecture. The advantage of using this architecture is that the presentation logic and business logic are separated from each other. This makes the software system flexible to changes and also helps in dividing responsibilities between development team. Here Model contains the business logic of the system, View renders the data in Model into a visual format for the users and Controller manages the presentation logic of the View and provides feedback to Model regarding user inputs.

SECTION 8. FUNCTIONAL ARCHITECTURE

5.1 System Architecture Overview

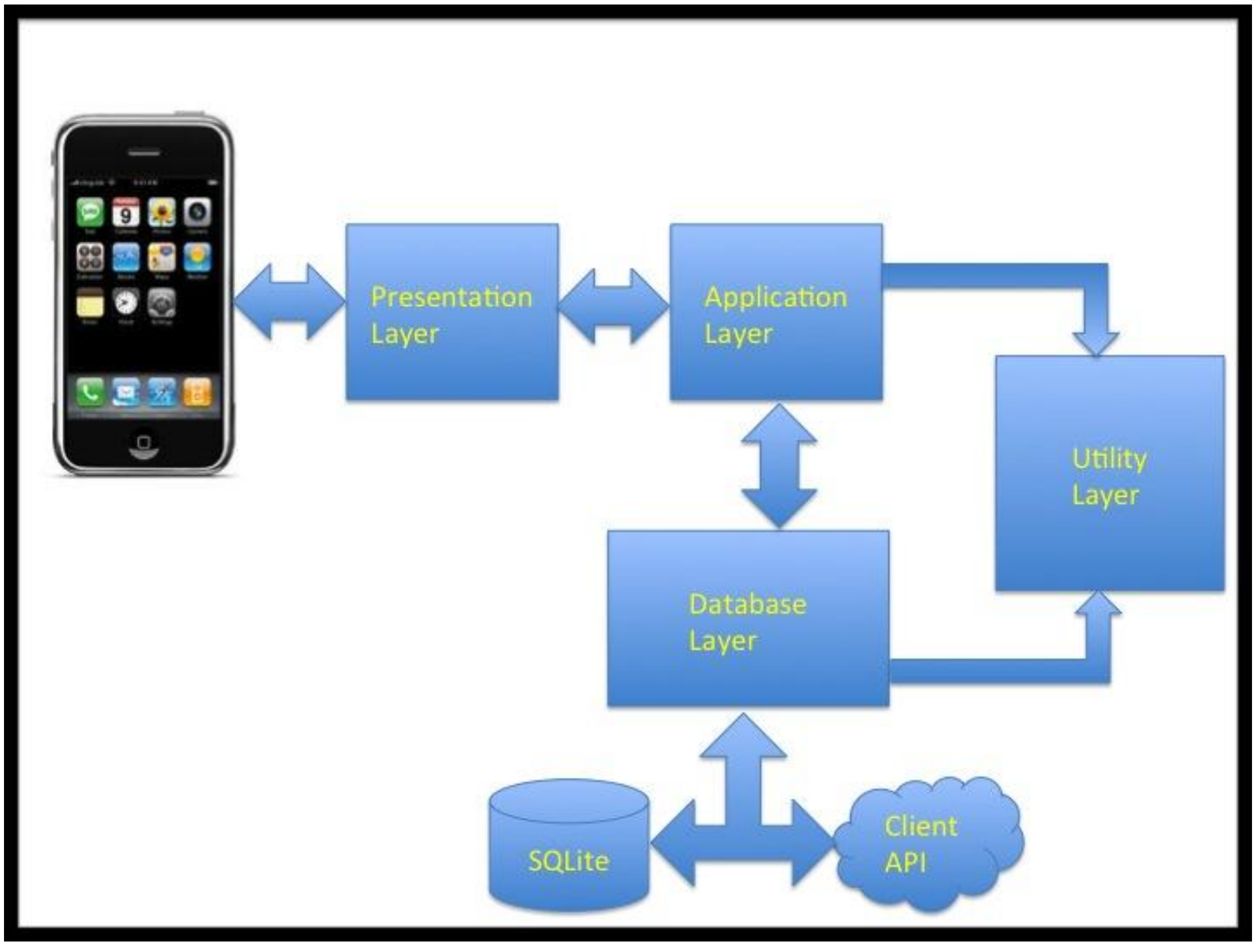

Fig. System architecture

Detailed description of each component in the architecture view:

a) **Presentation Layer:** This layer implements the functionality of *controller* in the MVC architecture. It will manage the presentation logic for the view and track user input to the system. Based on the user input, this layer will trigger appropriate application layer class for handling the messages.

b) **Application Layer:** Business logic of the application is contained in this layer. It sets the rule for manipulating the application data. This layer would form a part of the *Model* unit in the MVC architecture. The layer manages the flow of data through Database layer. Utility layer classes support this layer.

c) **Database Layer:** This layer forms the interface of our application to the data storage resource like SQLite and client side API's [web services]. In this layer data is not manipulated but simply retrieved or saved back to their resource based on request.

d) **Utility Layer:** Utility layer implements most of the common functionalities across the layers. This helps in easy maintenance and availability of the utilities to other layers. Few examples of functionalities that can be grouped into utility layer are XML parsing, database connectivity, string formatting etc.

e) **SQLite:** It is an internal data storage resource available in iPhone. It is light weighted and can be used for the purpose of storing mostly static data or for caching frequently used data.

f) **Client API:** Since the client data is large and dynamic; we would be utilizing the API's provided by client for retrieving and storing data into client database.

SECTION 9. OTHER REQUIREMENTS

Non-functional requirements focus on the qualities that must be applied to design and implement the system.

Type of Requirement	Description
Reliability	The application should give an up-time of 99%.
Operational Recovery	Manual/Automatic backup facility of data should be provided. On the event of a System/Application crashes, recovery should be quick with minimum process. If possible, automate the recovery wherever possible.
Performance and Response time	Every data should load without fail and incase of error, appropriate messages should be displayed like if unable to send a prescription to pharmacy or unable to change a schedule etc.
Scalability	The application architecture should allow any number of users to be added to the user groups, for instance currently only physicians can login to the application but later clinic staffs could also be able to login to the application with minimal changes.
Security and privacy	Users should not be able to view content/data restricted by clinic or patient.
Interoperability	The application should be able to integrate and exchange data or services with other systems.

SECTION 10. DEVELOPMENT ENVIRONMENTS

Technology	**Version**
iPhone SDK	5.0+
Xcode	4.3+
Web services	REST

Appendix A: Acronyms and Abbreviations

Acronym	Definition
XML	*Extended Mark-up Language*
DB	*Database*
Rx	Prescription
UI	User Interface
SDK	Software Development Kit