**Florida International University**

**School of Computing and Information Sciences**

**Senior Project**

**CIS 4911 (U01)**

**Professor: Masoud Sadjadi**

**Mentor: Steven Luis**

**Mobile Clinic**



**Team Members:**

Steven Berlanga

Rigoberto Hernandez

Sebastian Zanlongo

Carlos Corvaia

Michael Montaque

**April 23, 2013**

**Copyrights, trademarks and restrictions**

Copyright (C) Mobile Clinic-Electronic Medical Records

Permission is granted to copy, distribute and/or modify this document

under the terms of the GNU Free Documentation License, Version 1.3

or any later version published by the Free Software Foundation;

with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.

A copy of the license is included in the section entitled "GNU

Free Documentation License".

**Executive Summary**

*The main purpose of this document is to present the detail design for the application Mobile Clinic: Electronic Medical Record. This document will cover the purpose of the application and, the details of the system such as the system decomposition, design patterns, important classes and objects, and their interactions.*

**Table of Contents**

1 Introduction 7

1.1 Problem definition 7

1.2 Scope of system 7

1.3 Over all development methodology 8

1.4 Definitions, acronyms, and abbreviations (at most one page) 8

1.5 Overview of document 8

2 Feasibility Study 10

2.1 Current system 10

i. Limitations and Constraints 10

2.2 Alternative solutions 11

ii. Description of Alternatives 11

iii. Selection Criteria 11

iv. Operation 11

v. Technical Criteria 12

vi. Economic 12

vii. Schedule 12

viii. Deadlines 12

i. Analysis of Alternatives 13

2.3 Recommendation 14

3 Project Plan 15

3.1 Project Organization 15

ii. Project Personnel 15

iii. Hardware and Software Resources 15

3.2 Identification of Tasks, Milestones and Deliverables 15

i. Deliverables 15

ii. Milestones 17

iii. Tasks 18

3.3 Cost of the Project 19

4 System Requirements 20

4.1 Functional and Nonfunctional Requirements 20

4.2 Requirements Analysis 21

i. Use Case Model 21

ii. Static Model 24

iii. Dynamic Model 25

5 System Design (i.e., overall system design) 26

5.1 Overview 26

5.2 Subsystem Decomposition 27

5.2.1 Field Application 27

5.2.2 Cloud Application 27

5.3 Hardware and Software Mapping 28

5.4 Persistent Data Management 28

5.5 Security and Privacy 30

6 Detailed Design 31

6.1 Overview 31

6.2 Static model 32

6.3 Dynamic model 33

6.4 Code Specification 35

7 System Validation 36

7.1 Subsystem 36

7.2 System Tests 45

7.3 Evaluation of Tests 48

8 Glossary 51

9 Appendix 52

9.1 Appendix A - Project schedule (Gantt chart or PERT chart) 52

9.2 Appendix B – All use cases with nonfunctional requirements 53

9.3 Appendix C – User Interface designs 79

9.4 Appendix D – Analysis models (static and dynamic) 85

9.5 Appendix E – Design models (static and dynamic) 87

9.6 Appendix F – Documented Class interfaces (code) and constraints 90

9.7 Appendix G – Documented code for test drivers and stubs 91

9.8 Appendix H – Diary of meeting and tasks for the entire semester 92

10 References 97

10.1 Open Source Code 97

10.2 Licensing 98

# Introduction

Currently, volunteer physicians working for Orant Charities lack a robust method of tracking patients through their clinic. They also need a way to improve their care by studying what the most common ailments and prescribed medicines are. Orant require a method of keeping a history of patient visits and outcomes to improve their patient tracking, and to streamline record keeping in order to improve patient processing rates.

The aim of this project is to provide this charity with a simple, effective, and robust method of managing patients and patient records with the goal of improving the effectiveness and quality of the care Orant provides.

## Problem definition

The creation of an open source mobile app that can provide users with the ability to register and track patients and their visits, and allow for later analysis of these visits. The mobile client will implement a “triage app” that allows nurses, doctors, and pharmacists to manage patients. The app will provide users with clean, flexible forms to replace their existing paper-based forms. There will also be a local server and cloud server providing backups and synchronization of the data using a custom API.

## Scope of system

The scope of the project is to develop an iPad based system for managing three triage stations necessary for providing patient care in rural conditions. Each system will be in charge of a particular task oriented around patient care:

* Triage System - provide the ability for the triage staff to be able to take patients personal information and vitals.
* Physician System - allow a doctor to review vitals, conduct exams, render diagnosis and if necessary prescribe medication.
* Pharmacy System - allow a pharmacist to filled the prescriptions given to a patient by a doctor.

There will be a Local server that will allow the iPads to pass information between each other. This information will be available through a web application that will allow administrators to access/edit patient information.

## Over all development methodology

**Triage:** A nurse that checks in patients and obtains personal information such as name, age, and village. Triage will also record patient vitals such as heart rate, temperature, and complaints. For minor issues, the triage nurse can also hand out medication and check out the patient, removing strain on the doctor and pharmacy.

**Physician:** The doctor. The physician reviews patient vitals collected from triage, conducts exams, and renders a diagnosis. If necessary, they will also prescribe medication.

**Pharmacy:** The pharmacy consists of a person that dispenses prescriptions, usually on a one-time visit.

## Definitions, acronyms, and abbreviations (at most one page)

**MC-EMR:** Mobile Clinic Electronic Medical Record System

**API:** Application Programming Interface

**NFC:** Near Field Communication

## Overview of document

Section one includes the introduction to this project and describes the existing problem and why a solution is required to fill this need. This section also presents the problem definition and relevant background information. In order to facilitate this discussion, section one also provides the relevant definitions, acronyms, and terminology that were used throughout this project, and subsequently throughout this document. This section then ends with a brief overview of what will follow in this document.

Section two will review the feasibility study, which describes the current system employed by Orant Charities, and the limitations and constraints of this system. It then covers the purpose of the new application, combined with a definition of the requirements set forth by the user, and alternative solutions. The section concludes with the selection criteria used in analyzing the various alternative solutions.

Section three goes over the project plan, and the hardware and software requirements. This section also covers how the project itself is organized, along with the personnel organization and the identification of tasks, milestones, and deliverables for this project.

Section four contains the appendices with a project schedule and diary of meetings. The section also has a feasibility and cost matrix for this project.

Section five includes the references.

# Feasibility Study

In this section, we will evaluate the different alternative solutions to this problem. The first subsection will review the current system being employed, allowing readers to better understand what the goal is for this project. Next, the purpose of the replacement system will be described, along with the definition of user requirements. We then move on to the evaluation of the feasibility of the proposed system compared to other alternatives. Finally, the evaluation criteria that are being used will be explained.

## Current system

Orant Charities currently employs two paper-based forms in order to track the events in their clinics. Triage nurses start by taking down a new patient’s name, family name, village, age, sex, and weight. They can also record what the patient is complaining of, and any symptoms they exhibit. The patient and this form then move on to the Physician, which will exam the patient and diagnose them with a condition and possible a prescription. This prescription is written down on a second form, with details such as tablets per day, time of day to ingest, and any other additional instructions. Finally, the patient will take these forms to the Pharmacist, which dispenses medication.

The issue of managing patients and patient records is one that has constantly affected clinics and hospitals. Orant cannot afford nor requires the conventional record management systems being employed in hospitals today.

### Limitations and Constraints

The current system suffers from many of the issues that affect any paper based system. There is the loss of patient records during normal operation or traveling to and from the United States. The largest issue is that paper based records are difficult to analyze and data mine. Orant cannot refine its processes and planning because it relies mostly on patient and doctor feedback, rather than on collected data.

Moreover, the paper based records are sometimes difficult to pair with a returning patient, which can also compromise patient privacy.

## Alternative solutions

This section will review the existing alternatives to the problem. It will also cover the alternatives to each section of the solution. There are currently several alternatives to our proposed system. However, these patient record management systems are intended for use in large hospitals, and not mobile clinics in rural areas. The next subsection will compare each solution against selection criteria.

### Description of Alternatives

Current medical record alternatives are very expensive, requiring large servers for the processing and storage of thousands or millions of patients, combined with a large amount of information about each patient.

Because of the large amount of information that these systems are meant to collect, Orant would not have a use for a large part of these applications, and so would be paying for something they aren’t going to use. Moreover, the multitude of features can be overwhelming to new users, which could discourage volunteers.

These systems are also not meant to be mobile, and require constant internet connections in order to properly synchronize data across mobile devices, or perform backups. We can see an example of this with CloudCare, which requires an internet connection in order to function.

Moreover, many of these systems do not allow for biometric authentication of patients. We looked at several alternatives for authentication, including retina and fingerprint scanners, and NFC bracelets.

For the server, we looked at different providers, including Heroku and Amazon web services. For the mobile application, we were limited to a tablet, leaving us with either Android or iOS.

The local server will be a laptop capable of providing a local hotspot for the tablets to work on. This will be closely related to the choice of tablet. We were particularly interested in the MacBook Air, and the Lenovo ThinkPad.

### Selection Criteria

This section covers the selection criteria used to pick the best alternative solution. We used operation, technical criteria, schedule, and economic cost to make decisions.

### Operation

* + - 1. Performance
         1. System should have an adequate response time.
      2. Information
         1. Users should receive only relevant information.
      3. Scalability
         1. Allow for multiple iPads to run simultaneously.
         2. Allow for the addition of more iPads to a local server.
         3. Allow for the addition of more clinics and more local servers.
      4. Interface
         1. Easy to learn.
         2. Simple forms.
         3. Flexible input.
      5. Efficiency
         1. System should increase the throughput of clinics.

### Technical Criteria

* + - 1. Availability and Maturity
         1. The solutions used must be reliable enough to operate for extended periods of time.
         2. Technology must be available for the client.

### Economic

* + - 1. Operation
         1. Final solution should have a low maintenance cost.
      2. Infrastructure
         1. Equipment used should be within an acceptable budget for the charity.

### Schedule

* + - 1. Training
         1. Time needed to learn tools required for solution.
         2. Time to implement solution.
         3. Time needed to train users of the application.

### Deadlines

* + - 1. Consequences of not meeting deadlines.
      2. Required to ship before next visit to Malawi.

### Analysis of Alternatives

|  |  |  |  |
| --- | --- | --- | --- |
| **Cloud Server** | **Pros** | **Cons** | **Rating** |
| Amazon Web Services | Reliable Fast Easily Scalable | Expensive | 3 |
| Heroku | Economic Support for Ruby | More difficult to scale | 3.2 |
|  |  |  |  |
| **Biometry** | **Pros** | **Cons** | **Rating** |
| Tactivo | Commercially available Developer-friendly Cannot be forged | Expensive | 4 |
| Flomio | Inexpensive | Not in commercial production Can be lost | 2 |
|  |  |  |  |
| **Tablet** | **Pros** | **Cons** | **Rating** |
| iPad | Physicians already own them Could be more compatible with local server Long battery life | Expensive | 4 |
| Android | Less Expensive | Could have compatibility issues with server Many different versions | 3 |
|  |  |  |  |
| **Local Server** | **Pros** | **Cons** | **Rating** |
| MacBook Air | Should be more compatible with iPads Long battery life | Expensive | 3 |
| Lenovo ThinkPad | Economic Long battery life | Could be incompatible with tablets | 2.5 |

## Recommendation

The server chosen was Heroku, as it provides a reliable and simple setup for Ruby, upon which we will build the cloud server. Heroku is also provides much more affordable pricing options which are in line with the relatively small amount of traffic the servers will be handling.

For biometric authentication, the Tactivo fingerprint reader will be used. Tactivo is one of the few iPad-compatible fingerprint readers on the market, and appears to be the most developer-friendly, with an SDK and support.

iPads were chosen to run the mobile application, since the majority of the volunteer physicians already own one, allowing us to reduce the cost to the clinic. iPads should also be more compatible with the local server.

The local server that was chosen is a MacBook Air. This provides one of the longest battery lives, and should provide a simple connection for the iPads.

The local server will communicate with the cloud server through an AirTel 3G hotspot. This will be provided by the charity.

# Project Plan

In this section, we describe the project plan. This will include the project organization and the personnel organization. After this we will review the hardware and software requirements for the solution. Finally, there is the identification of the tasks, milestones, and deliverables.

## Project Organization

This section explains each person's role and their accompanying responsibilities.

### Project Personnel

This subsection reviews the personnel roles throughout the project.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Deliverable 1** | **Deliverable 2** | **Deliverable 3** | **Deliverable 4** |
| Steven Berlanga | System Analyst | Architect | Developer | Minute Taker |
| Carlos Corvaia | Architect | Leader | Developer | Developer |
| Rigoberto Hernandez | Leader | Developer | Minute Taker | Architect |
| Michael Montaque | Developer | Minute Taker | Developer | Leader |
| Sebastian Zanlongo | Minute Taker | Architect | Leader | Developer |

### Hardware and Software Resources

For this project, we will be developing an iOS mobile application. This requires XCode running on OSX. Most of the team has their own MacBook, allowing them to work from home and to use third-party tools to assist them. However, there is a team member that doesn’t own one, and will be required to work in the university’s computer lab to work on the project. In addition, there are different versions of iOS and XCode, so we had to verify that all iPad’s run 6.0.1, and the same version of XCode.

## Identification of Tasks, Milestones and Deliverables

Here, we will review the work breakdown of the project. Table one covers the deliverables that need to be written. Table two shows the milestones for each deliverable. Table three indicates which tasks must be met for each deliverable.

### Deliverables

|  |  |
| --- | --- |
| **Deliverable** | **Description** |
| 1 | Feasibility Study |
| 2 | Requirements |
| 3 | Design |
| 4 | Final Document |

### Milestones

|  |  |  |
| --- | --- | --- |
| **Milestone** | **Name** | **Deliverable** |
| 1 | Requirements Elicitation | 1 |
| 2 | Analysis of System Requirements | 1 |
| 3 | Feasibility Study | 1 |
| 4 | Setup of Server | 1 |
| 5 | Setup of Development Environment | 1 |
| 6 | Creation of Schemas | 1 |
| 7 | Mockup App | 1 |
| 8 | Completion of Triage | 2 |
| 9 | Completion of Doctor | 2 |
| 10 | Completion of Pharmacy | 2 |
| 11 | Submit beta to Orant | 2 |
| 12 | Basic Cloud Server | 2 |
| 13 | Basic Local Server | 2 |
| 14 | Creation of API | 2 |
| 15 | Integration of Photos | 3 |
| 16 | Integration of Biometrics | 3 |
| 17 | Completion of Cloud Server | 3 |
| 18 | Finalized App | 4 |
| 19 | Finalized Local Server | 4 |
| 20 | Finalized Cloud Server | 4 |
| 21 | Finalized API | 4 |
| 22 | Submit to Orant | 4 |

### Tasks

|  |  |
| --- | --- |
| **ID** | **Title** |
| US-103 | As an administrator I want the server to connect with the physical Device so that information can be persistent throughout the system |
| US-44 | As a triage nurse I want to login into the system so that I can work securely |
| US-37 | As a potential user I want to create my own user profile so I can be properly identified in the system |
| US-2 | As a triage nurse I want to manually register a new patient so that I can Check-In a patient. |
| US-70 | As a triage nurse I want to link family members together so that I can quickly bring up other patients |
| US-5 | As a physician I want to diagnose a patient to provide the patient care. |
| US-11 | As a pharmacist I to want finalize and check out the patient so that the staff knows that patient's treatment is complete. |
| US-36 | As a pharmacist I want to fulfill the patient's prescription so that I can confirm they were given medication |
| US-13 | As a pharmacist I want see the doctor’s prescription so that I know what drug to dispense. |
| US-21 | As an application administrator I want to verify when the local system backs up to the Cloud so that I can verify the data's status |
| US-15 | As a Triage nurse I want to prioritize the patient so that the critical patients get attention first |
| US-25 | As an application administrator I want easily startup, shutdown and reset the local server so that adequate operating & troubleshooting measures can be executed |
| US-43 | as a physician I want to manually identify a patient so that I can find their records |
| US-67 | As a Triage Nurse I want to be able to check in a patient so that I can take their vitals and collect their family history. |
| US-68 | As a Triage Nurse I want to be able to search for an existing patient to record their vitals for the current visit |
| US-73 | As a Triage nurse I want to quickly discharge a patient so that patients with major issues can quickly see the doctor |
| US-81 | As a triage / doctor / pharmacist, I want to check in and search for patients using their fingerprints. |
| US-82 | As a doctor, I want to assign patient's medication from a table. |
| US-91 | As a triage / doctor / pharmacist, I want to search for a patient. |
| US-92 | As a doctor I want to be able to checkout a patient if no medication needs to be prescribed. |
| US-93 | As a triage, I want to assign patients a priority in queue. |
| US-94 | As a pharmacist, I want to select patients from a queue. |
| US-95 | As a doctor, I want to select patients from a queue based on priority. |
| US-96 | As a triage, I want to assign patient vitals. |
| US-97 | As a doctor, I want to save a patient's diagnosis to their current visit in queue. |
| US-99 | As a developer, I want to be able to test production on my rails application on a staging server before being live. |
| US-104 | As a triage nurse I want to be able to quickly abort the patient I am working on so that I can address other patients |
| US-105 | As a triage nurse I want to be aware of all the patients that are currently in the system so that I can monitor their progress |
| US-107 | As a Physician I want to be able to quickly select multiple medication for a patient |
| US-109 | As an Application Administrator I want to be able to sync (upload information) between the Local Server and the Cloud app |
| US-110 | As an Application Administrator I want to be able to sync (download information) between the Local Server and the Cloud app |
| US-112 | As a triage nurse I want to be able to login so that I can access my dashboard |
| US-113 | As a Pharmacist I want to be able to login so that I can access my patient queue |
| US-114 | As a Doctor I want to be able to login so that I can access my patient queue |
| US-115 | As an app user I want to be able to see the total number of tablets/fl oz. available for any particular medication |
| US-116 | As a triage, I want to have a dashboard to manage my account. |
| US-117 | As a triage, I want to assign patients temperature and brief explanation (title) of a patient's visit. |
| US-120 | As a physician, I want to see relevant patient information in the queue. |

## Cost of the Project

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Item** | **Quantity** | | **Cost** | | **Total** | |
| MacBook Air | 1 | 999 | | 999 | |
| iPad 2 | 3 | N/A | | 0 | |
| Heroku | 1 | 150 | | 150 | |
| Total |  |  | | 1149 | |

# System Requirements

In this chapter we created a walkthrough of the functionality using the functional requirements from the use cases we implemented. The use case diagram along with a brief description of the relationships between the use cases and actors was also included. The chapter ends with an analysis of the requirements. The analysis of the system requirements is broken down into Use Case Models, that will describe the proposed functionality of the system. Then the Static Models which will contain object diagrams and class diagrams. finally the dynamic models will be made up of sequence diagrams and state machines.

## Functional and Nonfunctional Requirements

We will be implementing a total of 15 use cases for MC-EMR. Which will be broken on how the system is comprised of:

1. The Mobile IPad Application shall provided a RIU with the ability to login (use case is MC-EMR001) as seen in appendix B
2. The Mobile IPad Application shall provided a RIU with the ability to log out (use case is MC-EMR002) as seen in appendix B
3. The Mobile IPad Application shall provided a RIN with the ability to create a new patient
4. The Mobile IPad Application shall provided a RIN with the ability to register a patients fingerprint
5. The Mobile IPad Application shall provided a RIN with the ability to check in a patient into a queue
6. The Mobile IPad Application shall provided a RIN with the ability to quick check-out a patient
7. The Mobile IPad Application shall enable a RIU with the ability to search for existing patient data
8. The Mobile IPad Application shall also give the capacity to a RIU to add a new entries to existing patients
9. The Mobile IPad Application shall provide to the RIU to modify existing patient data
10. The Mobile IPad Application shall give the capacity to a RIU to register medication given to patients
11. The Mobile IPad Application shall enable a RID the ability to diagnose a patient.
12. The Mobile IPad Application shall enable a RIP the ability to prescribe medication.
13. The Local Server shall provide a RLSA with the ability to push local data to the cloud
14. The Local Server shall provide a RLSA with the ability to pull cloud data
15. The Local Server shall provide a RLSA with the ability to sync local iPad with the most recent information

## Requirements Analysis

### Use Case Model

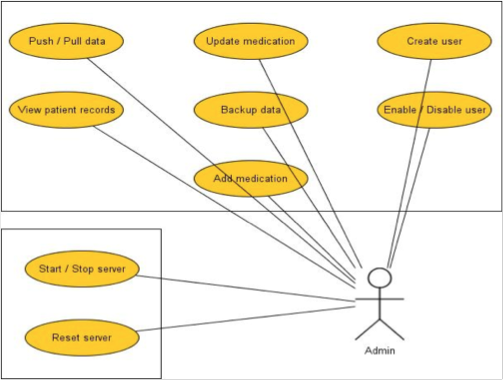


Figure 4.2.1

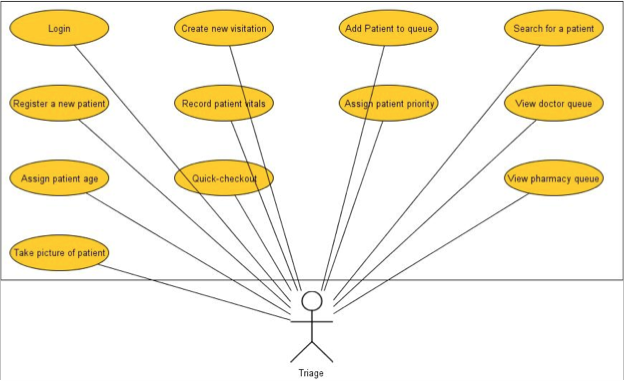


Figure 4.2.2

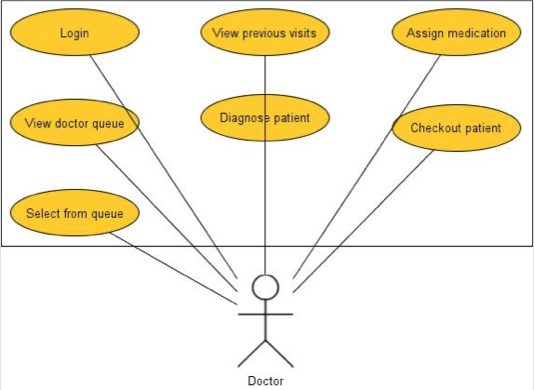


Figure 4.2.3

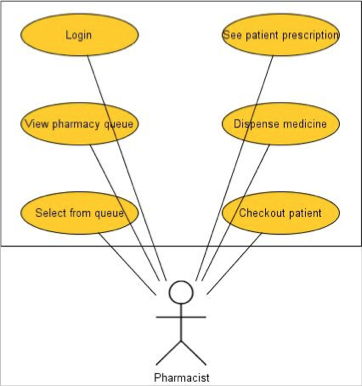


Figure 4.2.4

The fours actors in the system are a Registered IPad Nurse (RIN), Registered IPad Doctor (RID), Registered IPad Pharmacy (RIP), Registered Local Server User (RLSU). An RIN, RID and a RIP is a user that has gone through the registration process and can be authenticated by the iPad application only. A RLSU is a user that can authenticated by the iPad and Local Server applications.

A RIN can interact with every use case seen in Figure 4.2.2. A RID can interact with every use case seen in Figure 4.2.3. A RIP can interact with every use case seen in Figure 4.2.4. Lastly, a RLSU can interact with every use case seen in 4.2.1. There are similar use cases that are shared between different users such as Searching for a Patient, Logging into and out of the system and sending a patient into the waiting queue.

### Static Model

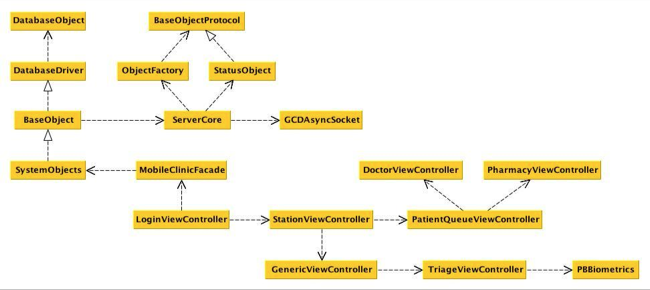


Figure 4.3.1

The diagram above (Figure 4.3.1) is the minimal class diagram that represents the structure of MC-EMR. This structure is replicated both in the iOS client and inside the Local server. The MobileClinicFacade is the entry point for any communication between the controllers and the database. The facade speaks to the specific object that is going to be modified. The ServerCode class sends a message to the GCDAAsyncSocket which communicates with another GCDAAsyncSocket class sitting on the Local Server which goes to the ServerCore to the Facade to the specific Controller.

### Dynamic Model

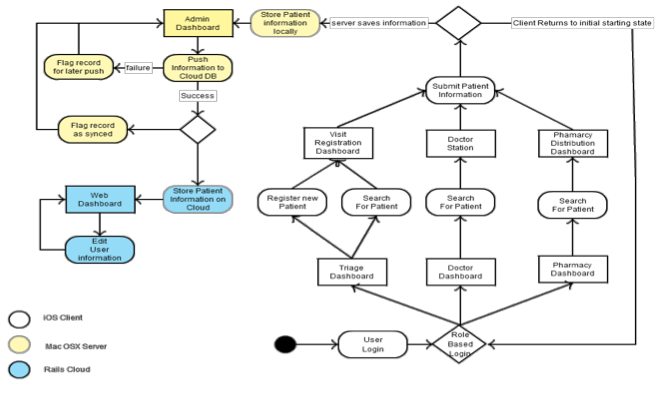


Figure 4.3.2

The diagram above (Figure 4.3.2) is the state diagram for the system. The System can have three paths after a user logging in. If a triage nurse logs in, he/she can either search for a patient or register a new patient. At this point, the triage nurse can register a new visit for the new patient.

If a Doctor/Pharmacy logs into the system, they can search for a patient in the queue. After selecting a patient they can write up a patient diagnosis and/or prescribe patient medication. After submitting the information, the user is sent back to their main screen.

When the information is sent to the Local Server, the server saves the information locally and sends the information to the cloud. The cloud application stores the information internally.

# System Design (i.e., overall system design)

This section will describe in detail the MC-EMR in high-level abstraction. It will describe the system’s primary and secondary architectures and its benefits, while breaking the system down into distinguishable subsystems. This section will also show the validity of each subsystem by tracing its functionality to its proper user story.

## Overview

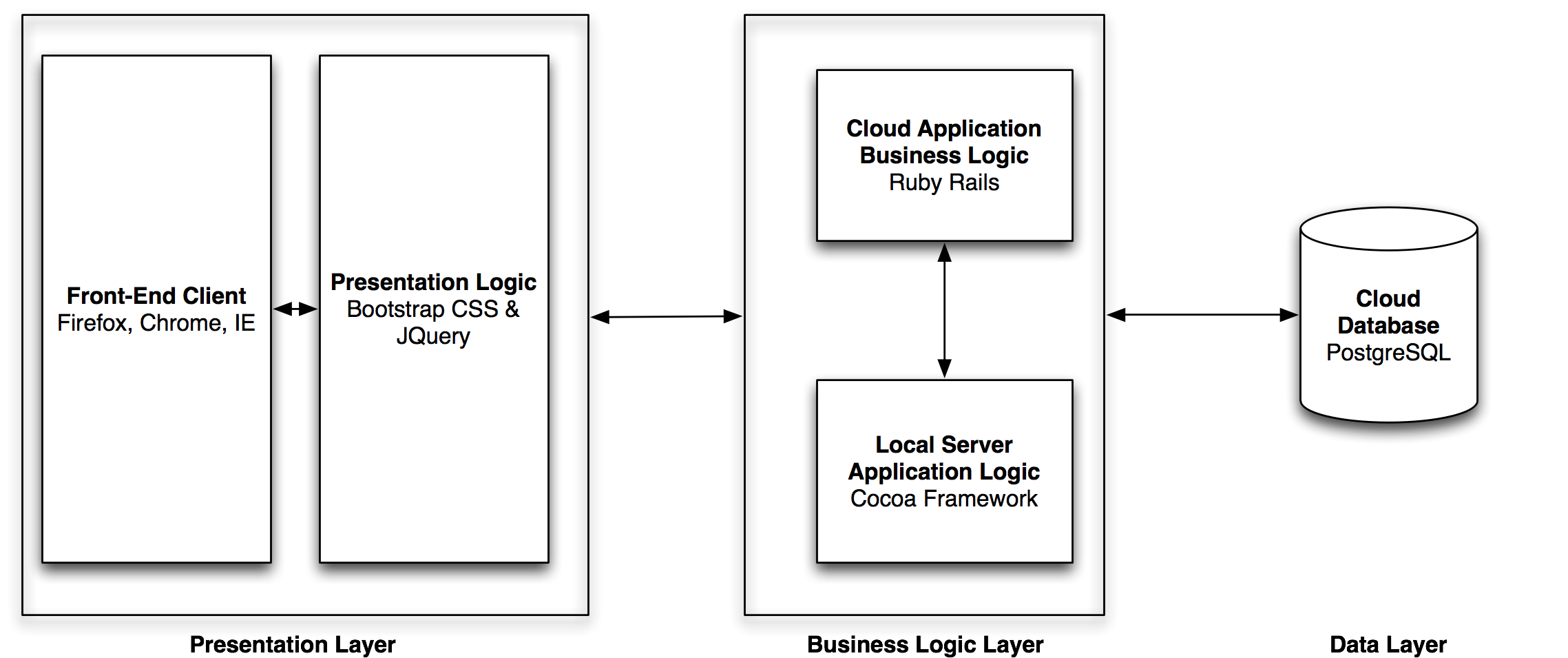
MC-EMR is broken into two major subsystems. The cloud application uses a 3T architecture. The other is a client server architecture, using iPads as clients and a MacBook Air as a server. 

Figure 1 Cloud Application Architecture

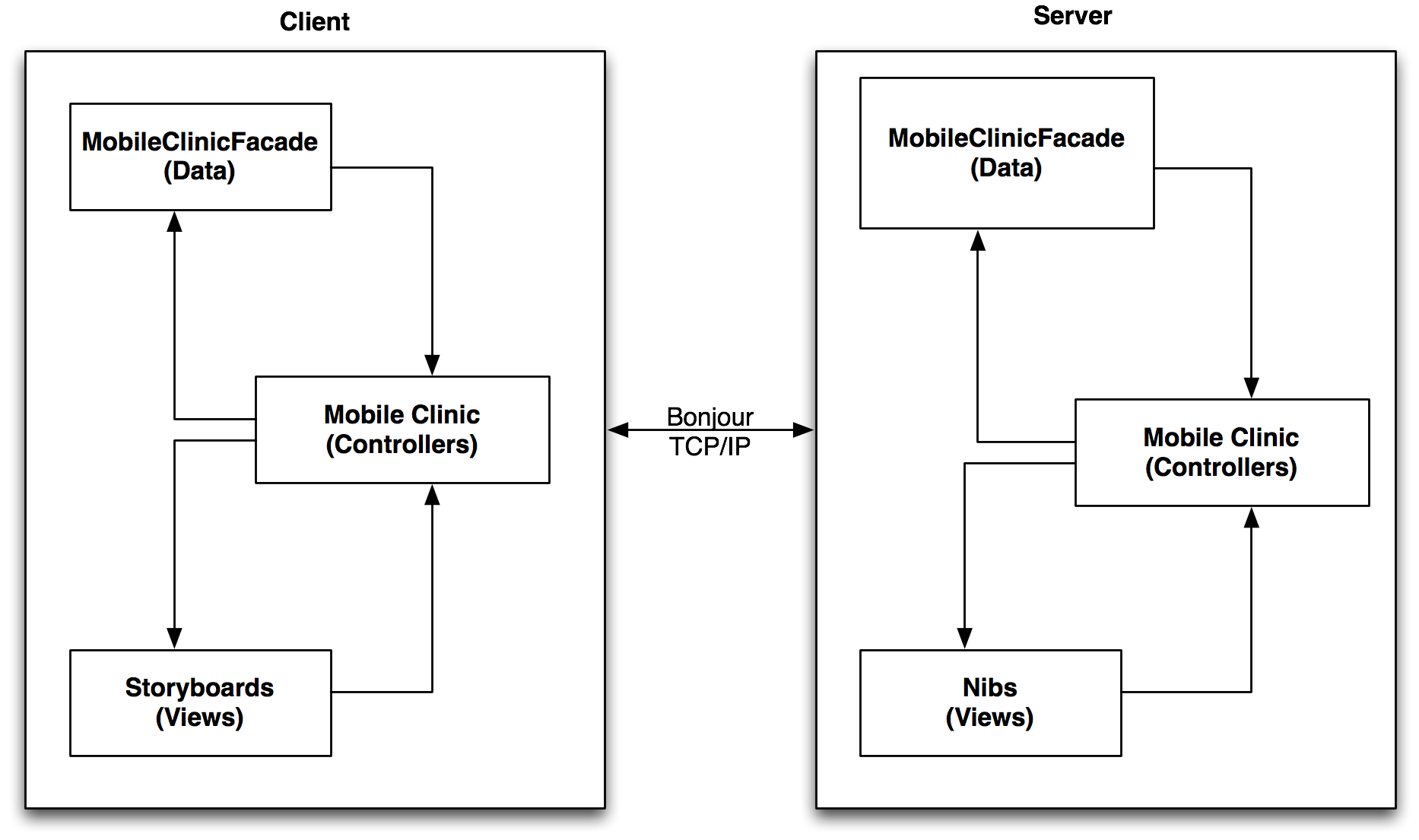


Figure 2 Field Application Architecture

The field application uses a MVC secondary architecture on both the client and the server. This is due to Apple’s guidelines and implantation of their framework. Unlike traditional client server’s, the client will connect on an “As Needed” basis. Every connection to the server, the client will update from the server to hold the most up-to-date information. This implementation is due to the specification of the customer.

## Subsystem Decomposition

### 5.2.1 Field Application

The Field application employs a client server with a MVC to govern internal logic flow. On the client, the controllers govern all the views, which are contained in the storyboard. The controllers dictate the flow the users take while they navigate the application. The data is split into two portions: the persistent data design and the network protocols. The persistent data uses the repository pattern. See section 2.4 for more details. The network protocols allow objects generated by the client (or server) to be sent over TCP/IP to and from the server.

### 5.2.2 Cloud Application

The cloud application operates on a 3T system. Data subsystem is hosted on Heroku server, which runs Linux. Ruby on Rails acts as the controller and also provides the end-points for the local server to connect to the cloud. The front end uses Twitter bootstrap as well as JavaScript. The cloud application is a semi-responsive system and can be viewed effectively on major web browsers such as Chrome, Firefox, and IE as well as the iPad’s mobile browser.

## Hardware and Software Mapping

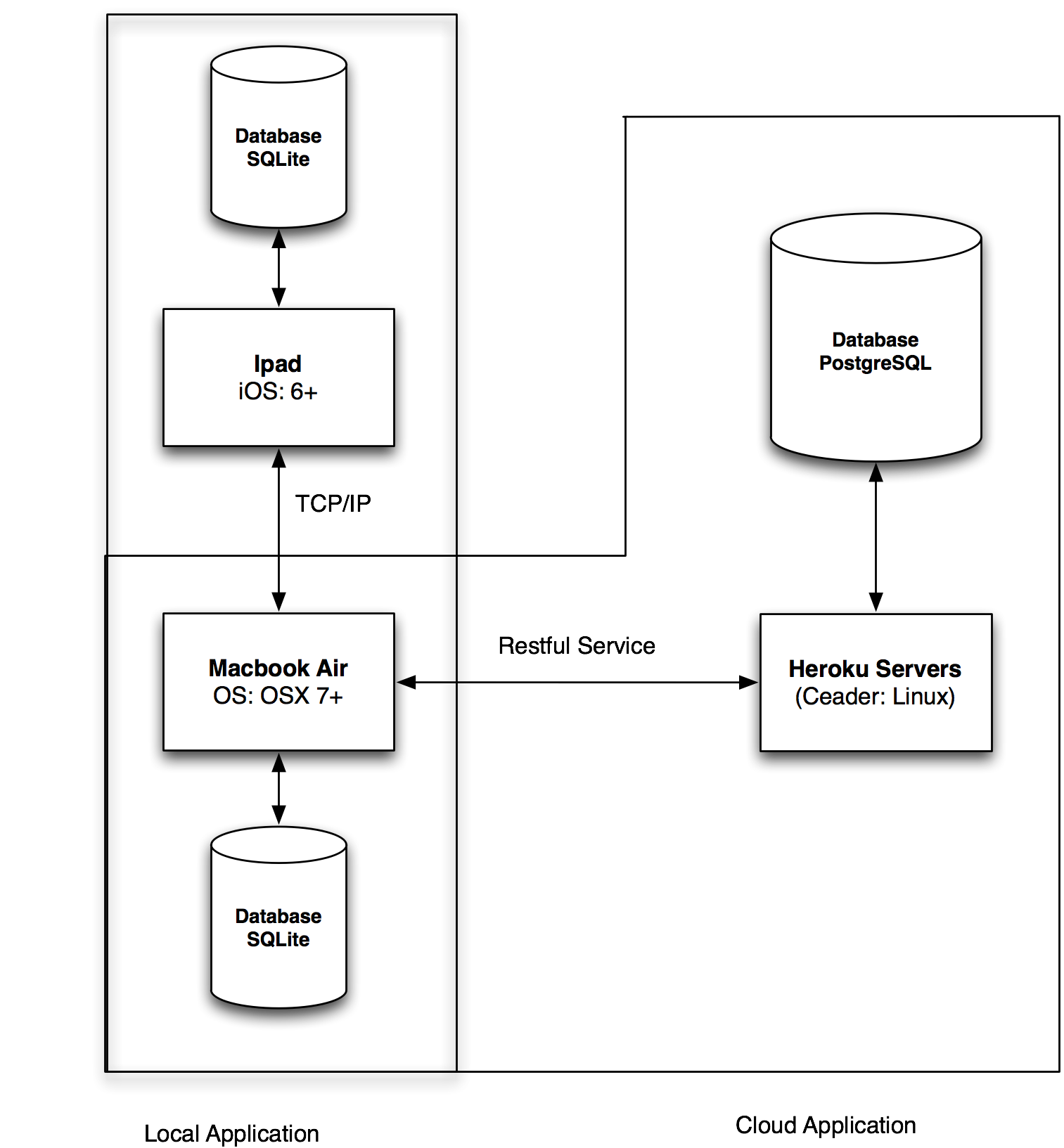


Figure Deployment Diagram

MC-EMR can operate under the following minimal specs:

|  |  |
| --- | --- |
| Hardware | Software |
| iPad 2nd Gen | iOS 6 |
| MacBook Air | 2Gb RAM, 802.11b/g 128Gb Harddrive |
| Heroku Server | Ruby on Rails 1.9.3 |
|  | PostgreSQL Server |

## Persistent Data Management

The data models the repository pattern. This pattern allows the clients to access the database through a controller, which will abstract the implementation of the database. This provides cohesion and reduces coupling and unnecessary dependencies. The system objects represents several classes that abstract the entities in the database. The PatientObject, VisitationObject, PrescriptionObject, MedicationObject, and UserObject are custom objects that allow the users to save and manipulate objects in the database without coupling the user’s implementations to a specific database.

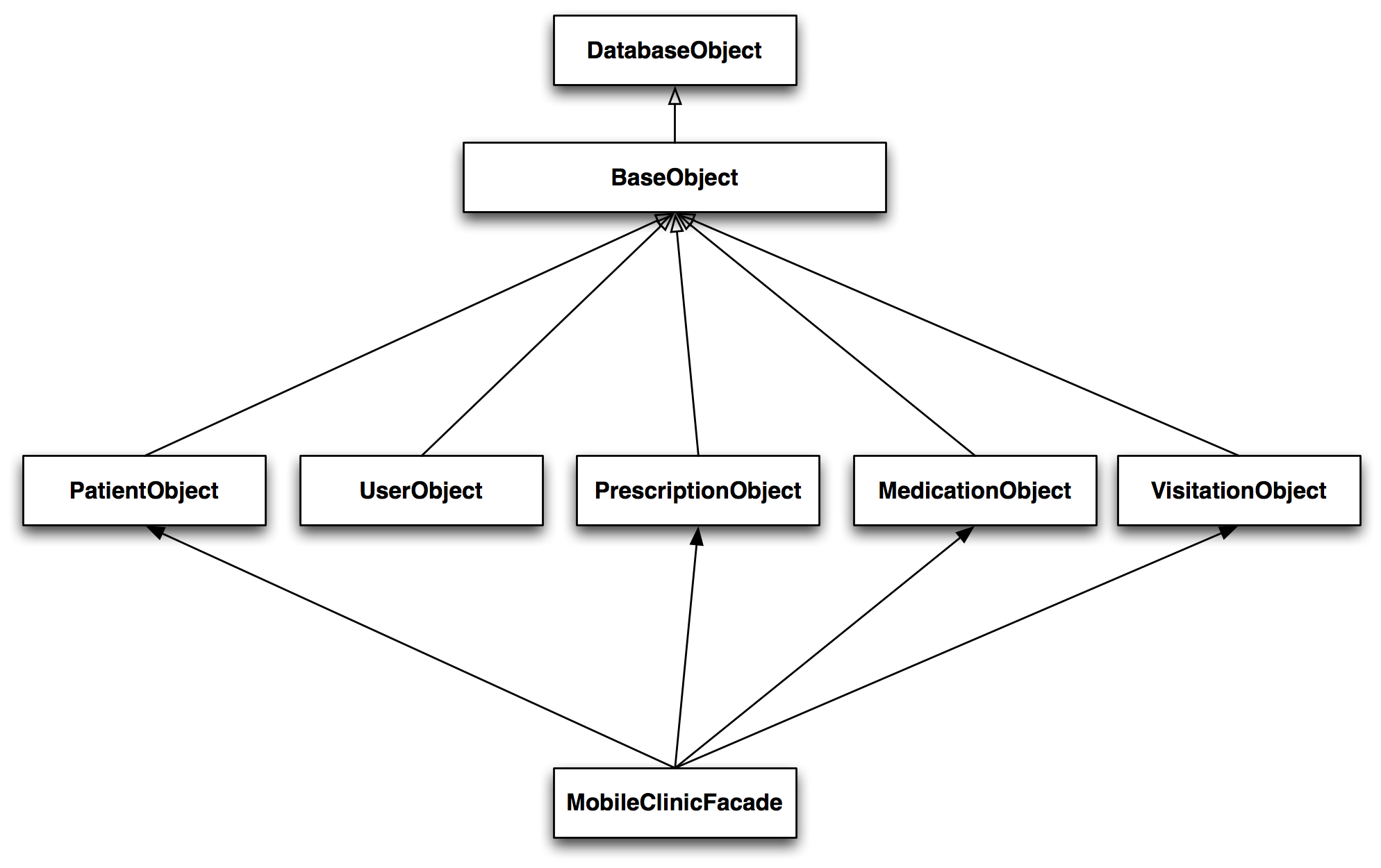


Figure System Objects

The system objects mentioned above all extend from the BaseObject, which also extends from the DatabaseDriver. The DatabaseDriver uses the DatabaseObject, which holds the actual connections to the database. This layer provides a layer of abstraction that not only decouples the system from the DataBaseObject but also from the network protocol implementation that allows the objects to be sent back and forth between the client and the server.

Figure System Object Entities

The information is also mirrored in the local server and only these objects are synced back with the cloud application. Every 15 minutes the local server backs up all its data in a separate JSON file to ensure that a physical form of the data is available should the system encounter some disaster.

## Security and Privacy

MC-EMR primary purpose is to operate without Internet connection. This significantly reduces the chance of cyber attack. The system is self-contained so that in order to steal or manipulate data the attacker must be on the same ad-hoc or local area network as local server. These connections will be secure use WPA Encryption. The system’s most vulnerable point is the cloud entry points. Cloud calls must be authenticated by presenting an authorized token. All patient information cannot be altered once it is sent to the local server and the visit is closed. This prevents data tampering. The users must have a username and password in order to use the iPads. To obtain access requires that the users register on the cloud application. After which, the local server must sync to update and be running in order for the user to login for the first time.

# Detailed Design

This chapter expands on the design of the entire system in terms of its implemented use cases and classes. Sections 3.1 contain class descriptions for major classes in the system, as well as the minimal class diagram. Section 3.2 describes the object interactions and contains sequence diagrams. Section 3.3 contains the detailed class diagram for the subsystems.

## Overview

The following is a description of the major classes and the respective subsystem to which they belong.

|  |  |  |
| --- | --- | --- |
| Subsystem | Class | Description |
| Data | UserObject | Represents a user of the system |
| PatientObject | Represents a patient and their information |
| VisitationObject | Represents a visit for a given patient |
| PrescriptionObject | Represents a prescription for a given visit |
| MedicationObject | Represents a medication that can be prescribed |
| BaseObject | Abstracts the interaction between the database and network |
| DatabaseDriver | Serves as a controller between the database and the client |
| DatabaseObejct | Connects directly with the database |
| Network | StatusObject | Generates and interprets the system’s status |
| ServerCore | Interfaces between the concrete network implementation |
| ObjectFactory | Generates the appropriate objects based on given values |
| BaseObjectProtocol | Decouples the system and allows for polymorphism |
| GCDAsyncSocket | Concrete implementation of network protocols |
| BaseObject | Abstracts the interaction between the database and network |
| Controller | LoginView | Provides visual interface for logging in |
| DoctorView | Provides visual interface for diagnosing patients |
| PatientQueueView | Provides visual interface for seeing patients that are up next |
| TriageView | Provides visual interface for registering and adding new patients |
| PBBiometrics | Provides visual interface for registering fingerprints |
| PharmacyView | Provides visual interface for administering prescriptions |
| DashboardView | Provides visual interface for showing all patients in the system |

## Static model

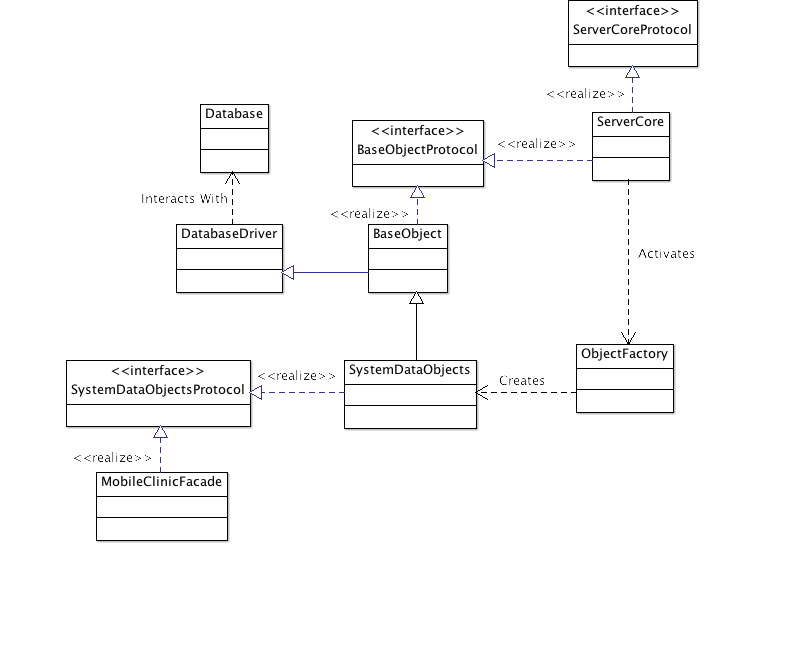
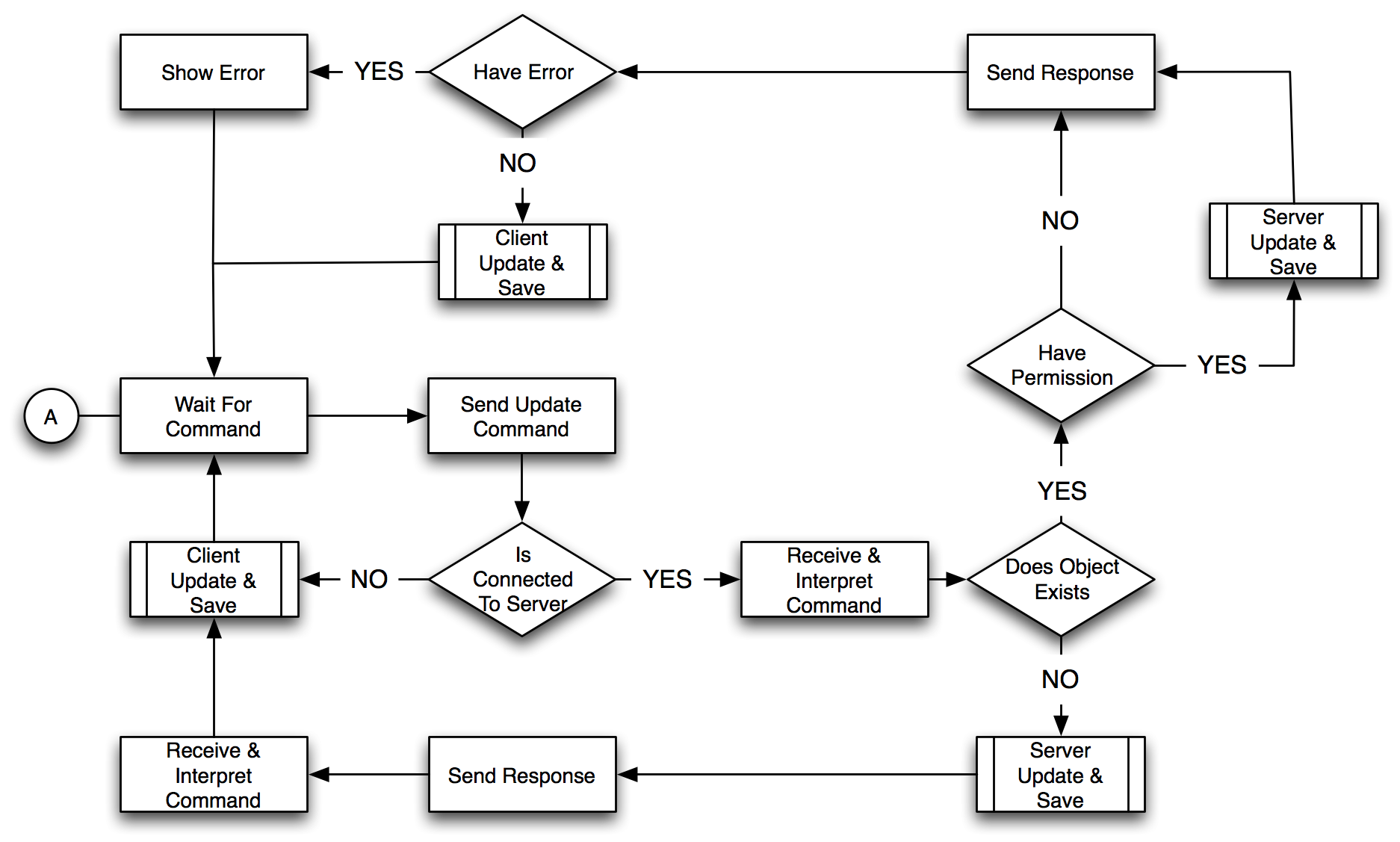
****

Figure Class Diagram of Data Subsystem

The data subsystem employs all of the design patterns mentioned above. The ServerCore and DatabaseObject are singletons to prevent duplication of data and threads. The ObjectFactory uses the factory pattern to retrieve the proper object based on the data it receives. The ServerCore also uses the observer pattern to alert any listeners that clients have connected to the device. The BaseObject uses the command pattern to delay the execution of commands to the proper object and finally, the MFC uses the façade pattern to simplify the interface as well as increase cohesion.

## Dynamic model

**Update & Save** (Appendix D.12)

****

This state diagram represents all saving and updating stories of the system. This workflow is designed to prevent data from being overwritten or duplicated. After the server receives the command, it verifies

1. If object exists
2. If the object is locked by another user

It then saves the object if it is unlocked or updates the object if it already exists.

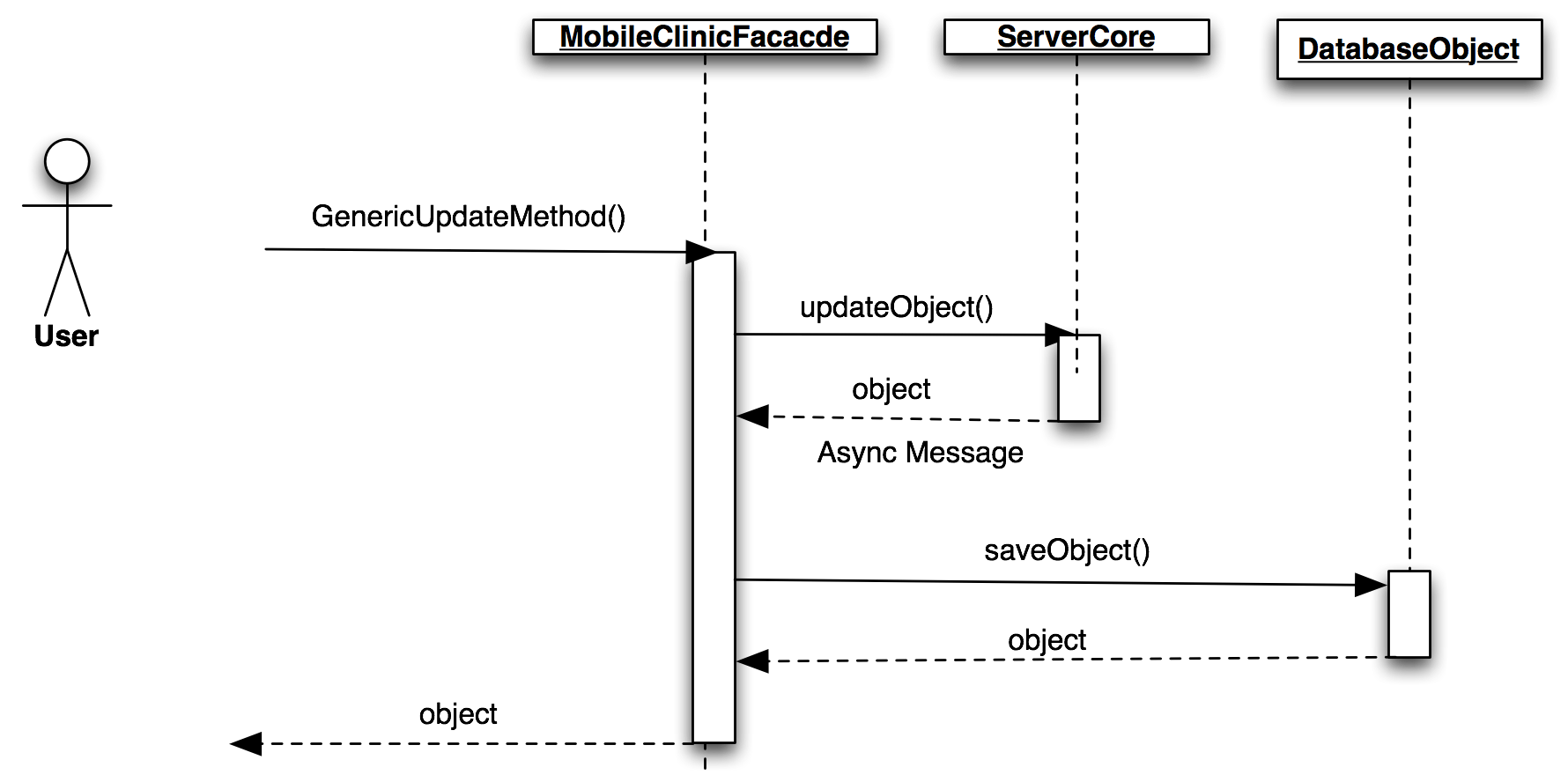


Figure Sequence Diagram of Update Visit (Client-Side)

On the client, when the user updates any object it is sent to the ServerCore. The server core can only be accessed efficiently through the MFC. Once the server sends the updated object back the ServerCore receives the information and saves the object if it was a success. The system prevents objects that already exist from being duplicated so long as the object contains a unique id.

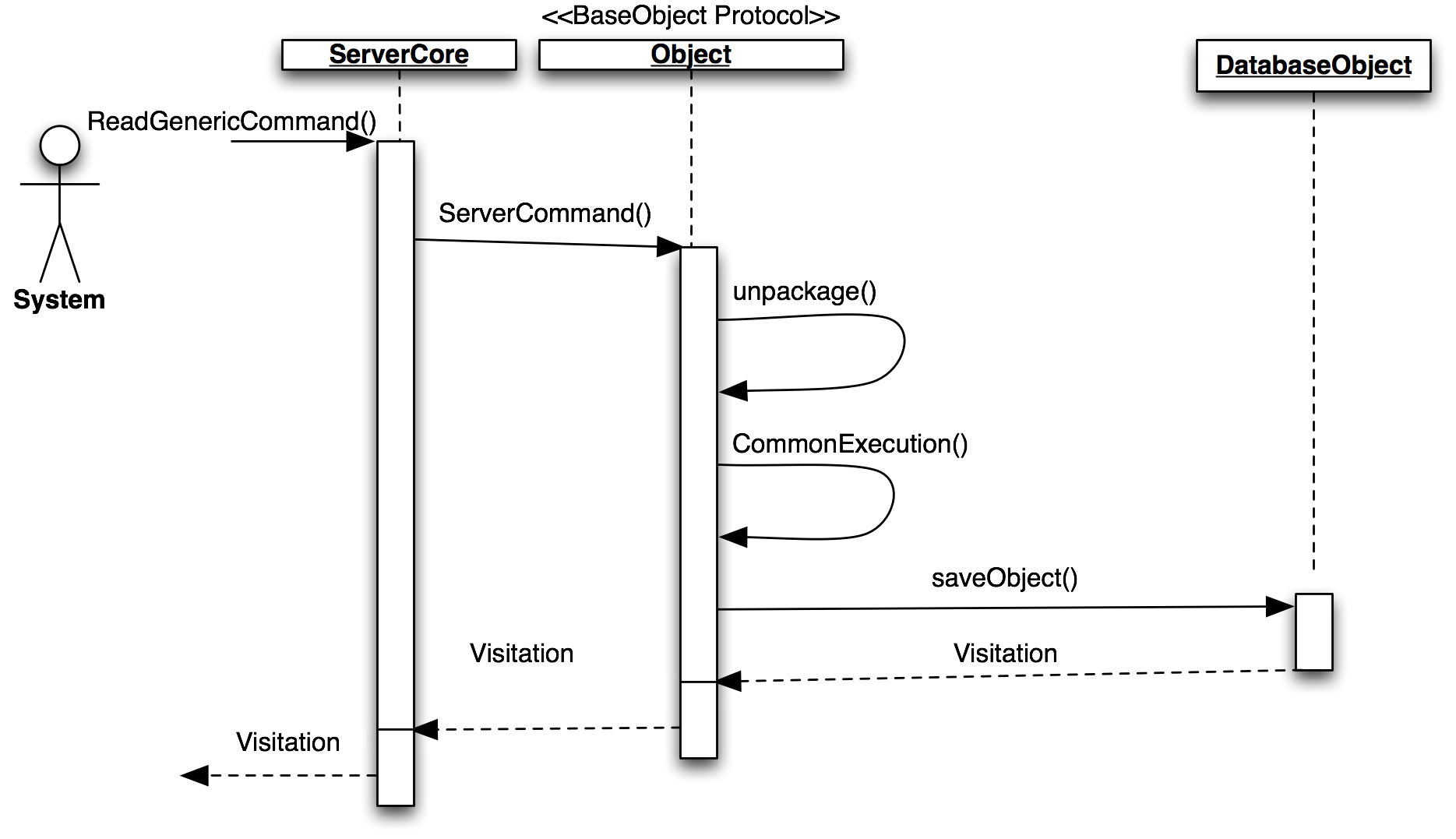


Figure Sequence Diagram Update Visit (Server-Sided)

The local server never initiates a communication with the clients. To further emphasis this point, the clients only connect to the server on an “As needed” basis. This means that the server will not know that the client exist until the client connects to the server and gives it a command. After the server responds to the client, the client will sever the connection. When the local server receives an update command, the ObjectFactory creates the appropriate object to handle the command. Object has a method that models the command pattern that allows the object to execute the correct method once the server demands it. All of these interactions happen on a background thread to keep the main thread clear of the communication of all the clients. On completion the server sends the updated object and a status back to the client. The status indicates whether the execution of the command was a success or failure.

Should the client or server take more than 5 seconds to connect or 10 seconds to finish a command, a timeout will occur and place the system back into a stable state.

## Code Specification

**Controller Subsystem** manipulates the views and controls the visual workflow. The controllers interact only with the MobileClinicFacade of the data subsystem. This reduces coupling and minimizes refactoring if implementations or objects on the lower level changes.

**Data Subsystem** follows the repository pattern to buffer any clients trying to access the database or the network. This subsystem allows for maximum flexibility by using standard data structures such as the dictionary and array to interact with outside classes. This allows any connecting system to immediately compatibility and high cohesion. The data system acts as the gatekeeper to the Networking subsystem.

**Networking subsystem** is as complex system with a simple interface. By taking advantage of Apple’s Block callback structure and Grand Central Dispatch, the networking subsystem is able to asynchronously send messages and continue at a later time without blocking the main thread. The implementation of the networking API is a 3rd party open source code call GCDAysncSocket.

# System Validation

In software engineering, system verification or validation is the process of checking that the software we as developers produce meets those specifications we put forth as the project was in the design stages of its creation and that it fulfills its intended purpose. In this section of the final deliverables, we will present details on the testing and validation of the system. The first portion of this section will illustrate how the subsystem tests were performed, such as the creation of test drivers and stubs. These tests will cover three portions of our system, the iPad application, the local server and a web application. In the following subsection, we will show how the system tests were performed. This testing is conducted on a complete, integrated system to evaluate the system’s compliance with its specified requirements. Lastly, we will conclude the section with an evaluation on how successful all the tests were in tabular form.

## Subsystem

|  |  |
| --- | --- |
| **Test Case ID** | MCEMR\_Subtest\_Retrieve\_Users\_001 |
| **Purpose** | To test that the application is retrieving all the users in the database that are active/inactive. |
| **Test Setup Environment** | Local server must be running. |
| **Test Inputs** | - Create a BaseObject  - Call query database method with “user” entity name |
| **Expected Outputs** | Results from call should equal true |

|  |  |
| --- | --- |
| **Test Case ID** | MCEMR\_Subtest\_Create\_User\_001 |
| **Purpose** | To test that an application user can be created in the server |
| **Test Setup Environment** | Local server must be running. |
| **Test Inputs** | - Create a BaseObject  - Create a NSDictionary  - Set username to “user”  - Set password to “trudat”  - Set firstname to “Bob”  - Set lastname to “Johnson”  - Set email to test@test.com  - Set usertype to “1”  - Call query database method with the NSDictionary |
| **Expected Outputs** | Results from call should equal true |

|  |  |
| --- | --- |
| **Test Case ID** | MCEMR\_Subtest\_Create\_User\_002 |
| **Purpose** | To test that duplicate application users are not allowed to be created in the server |
| **Test Setup Environment** | - Local server must be running.  - There exists a user with a username of “user” |
| **Test Inputs** | - Create a BaseObject  - Create an NSDictionary  - Set username to “user”  - Set password to “trudat”  - Set firstname to “Bob”  - Set lastname to “Johnson”  - Set email to test@test.com  - Set usertype to “1”  - Call query database method with the NSDictionary |
| **Expected Outputs** | Results from call should equal true |

|  |  |
| --- | --- |
| **Test Case ID** | MCEMR\_Subtest\_User\_for\_Id\_001 |
| **Purpose** | To test that the correct user is returned when we search for a user by their id. |
| **Test Setup Environment** | - Local server must be running.  - A list of users should exist in the database |
| **Test Inputs** | - Create a BaseObject  - Create an NSDictionary  - Set username to “John”  - Call query database method with the NSDictionary |
| **Expected Outputs** | Results from call should equal true |

|  |  |
| --- | --- |
| **Test Case ID** | MCEMR\_Subtest\_API\_User\_For\_Id\_001 |
| **Purpose** | Test that a specific user exists after pull from the cloud server. |
| **Test Setup Environment** | - Local server must be running.  - There must be internet access.  - Cloud server must be running.  - Cloud server must have a user with a username of “Frederick” in its database. |
| **Test Inputs** | - Generate a random string  - Create an access token for authentication  - Create an NSDictionary  - Set username as “Frederick”  - Call CloudService request method with “user\_by\_id” parameter |
| **Expected Outputs** | Results is equal to True |

|  |  |
| --- | --- |
| **Test Case ID** | MCEMR\_Subtest\_API\_Patients\_001 |
| **Purpose** | Test if the patients were retrieved from the cloud server properly. |
| **Test Setup Environment** | - Local server must be running.  - There must be internet access.  - Cloud server must be running.  - Cloud server must have a list of patients in its database. |
| **Test Inputs** | - Generate a random string  - Create an access token for authentication  - Call CloudServer request method for “Patients” entity |
| **Expected Outputs** | Results is equal to True |

|  |  |
| --- | --- |
| **Test Case ID** | MCEMR\_Subtest\_API\_Create\_Patients\_001 |
| **Purpose** | To test that we can create a new user on the cloud server. |
| **Test Setup Environment** | - Local server must be running.  - There must be internet access.  - Cloud server must be running. |
| **Test Inputs** | - Generate a random string  - Create an access token for authentication  - Create an NSDictornary  - Set firstname, familyname and villagename with a random string.  - Set sex to male  - Set age to 42  - Set patient name equal to firstname  - Call CloudService request method with “update\_patient” parameter. |
| **Expected Outputs** | Result is equal to True. |

|  |  |
| --- | --- |
| **Test Case ID** | MCEMR\_Subtest\_API\_Update\_Patient\_001 |
| **Purpose** | To test that we can update a patient on the cloud. |
| **Test Setup Environment** | - Local server must be running.  - There must be internet access.  - Cloud server must be running.  - Cloud server must have a user with the details below. |
| **Test Inputs** | - Generate a random string  - Create an access token for authentication  - Create an NSDictornary  - Set firstname, familyname and villagename with a random string.  - Set sex to male  - Set age to 42  - Set patient name equal to firstname  - Create a NSArray with dictionary  - Create a new dictionary and set Patient’s key with NSArray  - Call CloudService request method with “update\_patient” param. |
| **Expected Outputs** | Result is equal to True |

|  |  |
| --- | --- |
| **Test Case ID** | MCEMR\_Subtest\_API\_Update\_Photo\_001 |
| **Purpose** | To test that we can update a patient’s photo on the cloud. |
| **Test Setup Environment** | - Local server must be running.  - There must be internet access.  - Cloud server must be running.  - Cloud server must have a user with the details listed below. |
| **Test Inputs** | - Generate a random string  - Create an access token for authentication  - Create an NSDictornary  - Set firstname, familyname and villagename with a random string.  - Set sex to male  - Set age to 42  - Set patient name equal to firstname  - Set photo with new NSData  - Create a NSArray with dictionary  - Create a new dictionary and set Patient’s key with NSArray  - Call CloudService request method with “update\_patient” param. |
| **Expected Outputs** | Result is equal to True |

|  |  |
| --- | --- |
| **Test Case ID** | MCEMR\_Subtest\_API\_Create\_Visit\_001 |
| **Purpose** | To test that we can create a new visit and sync it with the cloud server. |
| **Test Setup Environment** | - Local server must be running.  - There must be internet access.  - Cloud server must be running. |
| **Test Inputs** | - Generate a random string  - Create an access token for authentication  - Create an NSDictionary  - Set triage-in/out, weight, condition, blood pressure to a random string  - Set doctorIn, doctorOut, doctorId, nurseId, patientId, and observation to “test”  - Create an NSArray with that dictionary  - Create a new NSDictionary with that array  - Call CloudService request method with “update\_visit” param. |
| **Expected Outputs** | Result is equal to True |

|  |  |
| --- | --- |
| **Test Case ID** | MCEMR\_Subtest\_Patient\_Object\_Test\_001 |
| **Purpose** | To test that unpackaging a file for a user is working properly |
| **Test Setup Environment** | Local server must be running. |
| **Test Inputs** | - Create a PatientObject  - Create an NSDictionary  - Set firstname to “John”  - Set familyname to “Doe”  - Set village to “village”  - Set patientId to “John.Doe.1234”  - Set sex to “1”  - Call unpackageFileForUser method with patient object |
| **Expected Outputs** | Patient object with the same values as test input |

|  |  |
| --- | --- |
| **Test Case ID** | MCEMR\_Subtest\_Create\_Find\_New\_Objects\_Sunny\_001 |
| **Purpose** | To test that we can create and find a patient object given a certain set on patient information. |
| **Test Setup Environment** | Local server must be running. |
| **Test Inputs** | - Set systemUser = “user”  - Create a date object  - Create a patientObject  - Create an NSArray  - Set firstname to “Tiffany”  - Set familyname to “Cinder”  - Set village to DOB  - Set sex to 0  - Call createNewObject method  - Save results to NSArray  - Create a local variable with patient object |
| **Expected Outputs** | Patient object should be the same. |

|  |  |
| --- | --- |
| **Test Case ID** | MCEMR\_Subtest\_Create\_Find\_New\_Objects\_Sunny\_002 |
| **Purpose** | To test that we can create and find a patient id given a certain set on patient information. |
| **Test Setup Environment** | Local server must be running. |
| **Test Inputs** | - Set systemUser = “user”  - Create a date object  - Create a patientObject  - Create an NSArray  - Set firstname to “Tiffany”  - Set familyname to “Cinder”  - Set village to DOB  - Set sex to 0  - Call createNewObject method  - Save results to NSArray  - Create a local variable with patient’s id |
| **Expected Outputs** | Patient id must exist and not be nil. |

|  |  |
| --- | --- |
| **Test Case ID** | MCEMR\_Subtest\_Create\_Find\_New\_Objects\_Rainy\_002 |
| **Purpose** | Test that given a patient object that contains an illegal value, that an error is thrown. |
| **Test Setup Environment** | Local server must be running. |
| **Test Inputs** | - Set systemUser = “user”  - Create a date object  - Create a patientObject  - Create an NSArray  - Set firstname to “Tiffany”  - Set familyname to “Cinder”  - Set village to DOB  - Set sex to 0  - Set dob value with a string  - Call createNewObject method  - Save error to NSInteger value |
| **Expected Outputs** | An error is thrown reading “Misconfigured object error” |

|  |  |
| --- | --- |
| **Test Case ID** | MCEMR\_Subtest\_Save\_Without\_PrimaryID\_001 |
| **Purpose** | Test that given a patient object and a dictionary with no primary id, an error is thrown CreateNewObject is called. |
| **Test Setup Environment** | Local server must be running. |
| **Test Inputs** | - Set systemUser = “user”  - Create a date object  - Create a patientObject  - Create an NSArray  - Set firstname to “Tiffany”  - Set familyname to “Cinder”  - Set village to DOB  - Set sex to 0  - Set value in dictionary to a patient object with no id  - Call saveObject  - Save error to an integer |
| **Expected Outputs** | System should display an error that the object is misconfigured. |

|  |  |
| --- | --- |
| **Test Case ID** | MCEMR\_Subtest\_Save\_Without\_PrimaryID\_002 |
| **Purpose** | Test that given a patient object and a dictionary with no primary id, an error is thrown CreateNewObject is called. |
| **Test Setup Environment** | Local server must be running. |
| **Test Inputs** | - Set systemUser = “user”  - Create a date object  - Create a patientObject  - Create an NSArray  - Set firstname to “Tiffany”  - Set familyname to “Cinder”  - Set village to DOB  - Set sex to 0  - Set value in dictionary to a patient object with no id  - Call saveObject  - Save error to an integer |
| **Expected Outputs** | The system prompts that the “Object was not assigned a primary key ID”. |

|  |  |
| --- | --- |
| **Test Case ID** | MCEMR\_Subtest\_Illegal\_Values\_Patient\_Object\_001 |
| **Purpose** | To test that an error is thrown when an illegal value is set and passed in a patient object. |
| **Test Setup Environment** | Local server must be running. |
| **Test Inputs** | - Set systemUser = “user”  - Create a date object  - Create a patientObject  - Create an NSArray  - Set firstname to “Tiffany”  - Set familyname to “Cinder”  - Set village to DOB  - Set sex to 0  - Set DOB with NSData  - Create a local boolean variable and set results of testPatient’s |
| **Expected Outputs** | Boolean value should be false |

|  |  |
| --- | --- |
| **Test Case ID** | MCEMR\_Subtest\_Register\_User\_001 |
| **Purpose** | To test that each field of the in registration is accessible and saved properly for a new patient. |
| **Test Setup Environment** | Local server must be running. |
| **Test Inputs** | - Create a new RegisterPatientViewController  - Set patient name field to “John”  - Set patient family name field to “Doe”  - Set village patient field to “Miami”  - Set segmentControl to 1  - Retrieve patient first name field  - Save to the database along with the remaining patient info. |
| **Expected Outputs** | Patient first name is “John” |

|  |  |
| --- | --- |
| **Test Case ID** | MCEMR\_Subtest\_Register\_User\_002 |
| **Purpose** | To test that each field of the in registration is accessible and saved properly for a new patient. |
| **Test Setup Environment** | Registration view controller fields must be done. |
| **Test Inputs** | - Create a new RegisterPatientViewController  - Set patient name field to “John”  - Set patient family name field to “Doe”  - Set village patient field to “Miami”  - Set segmentControl to 1  - Retrieve patient family name field  - Save to the database along with the remaining patient info. |
| **Expected Outputs** | Patient first name is “Doe” |

|  |  |
| --- | --- |
| **Test Case ID** | MCEMR\_Subtest\_Register\_User\_003 |
| **Purpose** | To test that each field of the in registration is accessible and saved properly for a new patient. |
| **Test Setup Environment** | Registration view controller fields must be done. |
| **Test Inputs** | - Create a new RegisterPatientViewController  - Set patient name field to “John”  - Set patient family name field to “Doe”  - Set village patient field to “Miami”  - Set segmentControl to 1  - Retrieve village name field  - Save to the database along with the remaining patient info. |
| **Expected Outputs** | Patient first name is “Miami” |

|  |  |
| --- | --- |
| **Test Case ID** | MCEMR\_Subtest\_Register\_User\_004 |
| **Purpose** | To test that each field of the in registration is accessible and saved properly for a new patient. |
| **Test Setup Environment** | Registration view controller fields must be done. |
| **Test Inputs** | - Create a new RegisterPatientViewController  - Set patient name field to “John”  - Set patient family name field to “Doe”  - Set village patient field to “Miami”  - Set segmentControl to 1  - Retrieve segment control value  - Save to the database along with the remaining patient info. |
| **Expected Outputs** | Patient first name is “1” |

## System Tests

|  |  |
| --- | --- |
| **Test Case ID** | MCEMR\_Systest\_Cloud\_Create\_Appusers\_001 |
| **Purpose** | To test that an app user can be created on the cloud server. |
| **Test Setup Environment** | Instantiate FactoryGirl gem for validation. |
| **Test Inputs** | - User enters firstname  - User enters email  - User enters password and confirms password  - User selects a user type  - User selects a status  - User clicks on submit |
| **Expected Outputs** |  |

|  |  |
| --- | --- |
| **Test Case ID** | MCEMR\_Systest\_Cloud\_Authencation\_001 |
| **Purpose** | To test that a user is properly authenticated. |
| **Test Setup Environment** | Instantiate the FactoryGirl gem for validation. |
| **Test Inputs** | User enters a username  User enters a password  User clicks on Sign-in |
| **Expected Outputs** |  |

|  |  |
| --- | --- |
| **Test Case ID** | MCEMR\_Systest\_Cloud\_Authencation\_002 |
| **Purpose** | To test that a user is rejected if the password is entered incorrectly. |
| **Test Setup Environment** | Instantiate the FactoryGirl gem for validation. |
| **Test Inputs** | User enters a username  User enters the wrong password  User clicks on Sign-in |
| **Expected Outputs** |  |

|  |  |
| --- | --- |
| **Test Case ID** | MCEMR\_Systest\_Cloud\_Create\_User\_001 |
| **Purpose** | To test that a user is properly created on the cloud server. |
| **Test Setup Environment** | Instantiate the FactoryGirl gem for validation |
| **Test Inputs** | Set name to “Example User”  Set email to user@example.com  Set password & confirmation password to “foobar”  Set email to “test@exmaple”  Set username to “test001”  Set first name to “User”  Set last name to “Example”  Set userType to “admin”  Set status to “active”  Initiate click action on submit button |
| **Expected Outputs** |  |

|  |  |
| --- | --- |
| **Test Case ID** | MCEMR\_Subtest\_Deactivate\_001 |
| **Purpose** | To test that a user is properly deactivated in the system. |
| **Test Setup Environment** | Local server must be running.  A user with the name “John” must exist in the database. |
| **Test Inputs** | - Create a BaseObject  - Create an NSDictionary  - Set username to “John”  - Call query database method with the NSDictionary |
| **Expected Outputs** | Results from call should equal true |

|  |  |
| --- | --- |
| **Test Case ID** | MCEMR\_Subtest\_API\_Users\_001 |
| **Purpose** | Test that users are retrieved from the cloud server. |
| **Test Setup Environment** | - Local server must be running.  - There must be internet access.  - Cloud server must be running.  - Cloud server must have a list of users in its database. |
| **Test Inputs** | - Generate a random string  - Create an access token for authentication  - Call CloudService request method with “Users” entity |
| **Expected Outputs** | Results is equal to True |

|  |  |
| --- | --- |
| **Test Case ID** | MCEMR\_Subtest\_API\_Deactivate\_User\_001 |
| **Purpose** | To test that a user can be deactivate locally after pulling from the server. |
| **Test Setup Environment** | - Local server must be running.  - There must be internet access.  - Cloud server must be running.  - Cloud server must have a user with the username of “Frederick”. |
| **Test Inputs** | - Generate a random string  - Create an access token for authentication  - Create an NSDictionary  - Set username as “Frederick”  - Create a NSArray with the dictionary object created  - Create a new dictionary with that NSArray for key “Users”  - Call CloudService request method with “deactivate\_user” parameter |
| **Expected Outputs** | Result is equal to True |

|  |  |
| --- | --- |
| **Test Case ID** | MCEMR\_Subtest\_API\_Test\_Visitation\_001 |
| **Purpose** | To test that we can pull visitation information from the cloud server. |
| **Test Setup Environment** | - Local server must be running.  - There must be internet access.  - Cloud server must be running.  - Cloud server must have a visitation in the database. |
| **Test Inputs** | - Generate a random string  - Create an access token for authentication  - Call CloudService request method with “Visitations” as a param. |
| **Expected Outputs** | Result is equal to True |

|  |  |
| --- | --- |
| **Test Case ID** | MCEMR\_Subtest\_Lock\_Patient\_Locally\_001 |
| **Purpose** | To test that a patient is locked properly when some is access their records. |
| **Test Setup Environment** | - Local server must be running.  - Patient with the details below should exist in the local server. |
| **Test Inputs** | - Set systemUser = “user”  - Create a date object  - Create a patientObject  - Create an NSArray  - Set firstname to “Tiffany”  - Set familyname to “Cinder”  - Set village to DOB  - Set sex to 0  - UpdateObjectAndShouldLock method is called  - Results are saved to an NSArray  - Patient’s id is saved to a local variable  - Save the username of the user who locked the patient  - Save any error codes to an NSInteger |
| **Expected Outputs** | LockedBy value should be equal to the systemUser id |

|  |  |
| --- | --- |
| **Test Case ID** | MCEMR\_Subtest\_Lock\_Patient\_Locally\_002 |
| **Purpose** | To test that a patient is locked properly when some is access their records while disconnected. |
| **Test Setup Environment** | - Local server must be running.  - Patient must exist in the local server. |
| **Test Inputs** | - Set systemUser = “user”  - Create a date object  - Create a patientObject  - Create an NSArray  - Set firstname to “Tiffany”  - Set familyname to “Cinder”  - Set village to DOB  - Set sex to 0  - UpdateObjectAndShouldLock method is called  - Results are saved to an NSArray  - Patient’s id is saved to a local variable  - Save the username of the user who locked the patient  - Save any error codes to an NSInteger |
| **Expected Outputs** | System error should be equal the kErrorDiconnected error. |

## Evaluation of Tests

|  |  |
| --- | --- |
| **Test Case ID** | **Result** |
| MCEMR\_Subtest\_Retrieve\_Users\_001 | **PASS** |
| MCEMR\_Subtest\_Create\_User\_001 | **PASS** |
| MCEMR\_Subtest\_Create\_User\_002 | **PASS** |
| MCEMR\_Subtest\_User\_for\_Id\_001 | **PASS** |
| MCEMR\_Subtest\_Deactivate\_001 | **PASS** |
| MCEMR\_Subtest\_API\_Users\_001 | **PASS** |
| MCEMR\_Subtest\_API\_UserForId | **PASS** |
| MCEMR\_Subtest\_API\_Deactivate\_User\_001 | **PASS** |
| MCEMR\_Subtest\_API\_Patients\_001 | **PASS** |
| MCEMR\_Subtest\_API\_Create\_Patients\_001 | **PASS** |
| MCEMR\_Subtest\_API\_Update\_Patient\_001 | **PASS** |
| MCEMR\_Subtest\_API\_Update\_Photo\_001 | **PASS** |
| MCEMR\_Subtest\_API\_Test\_Visitation\_001 | **PASS** |
| MCEMR\_Subtest\_API\_Create\_Visit\_001 | **PASS** |
| MCEMR\_Subtest\_PatientObjectTest\_001 | **PASS** |
| MCEMR\_Subtest\_Create\_Find\_New\_Objects\_Sunny\_001 | **PASS** |
| MCEMR\_Subtest\_Create\_Find\_New\_Objects\_Sunny\_002 | **PASS** |
| MCEMR\_Subtest\_Create\_Find\_New\_Objects\_Rainy\_002 | **PASS** |
| MCEMR\_Subtest\_Lock\_Patient\_Locally\_001 | **PASS** |
| MCEMR\_Subtest\_Lock\_Patient\_Locally\_002 | **PASS** |
| MCEMR\_Subtest\_Save\_Without\_PrimaryID\_001 | **PASS** |
| MCEMR\_Subtest\_Save\_Without\_PrimaryID\_002 | **PASS** |
| MCEMR\_Subtest\_Illegal\_Values\_Patient\_Object\_001 | **PASS** |
| MCEMR\_Subtest\_Register\_User\_001 | **PASS** |
| MCEMR\_Subtest\_Register\_User\_002 | **PASS** |
| MCEMR\_Subtest\_Register\_User\_003 | **PASS** |
| MCEMR\_Subtest\_Register\_User\_004 | **PASS** |
| MCEMR\_Systest\_Cloud\_Create\_Appusers\_001 | **PASS** |
| MCEMR\_Systest\_Cloud\_Authencation\_001 | **PASS** |
| MCEMR\_Systest\_Cloud\_Authencation\_002 | **PASS** |
| MCEMR\_Systest\_Cloud\_Create\_User\_001 | **PASS** |

# Glossary

**Cloud**: Is the delivery of computing and storage capacity as a service to a heterogeneous community of end-recipients. The name comes from the use of a cloud-shaped symbol as an abstraction for the complex infrastructure it contains in system diagrams. Cloud computing entrusts services with a user's data, software and computation over a network.

**iOS**: (previously iPhone OS) is a mobile operating system developed and distributed by Apple Inc.

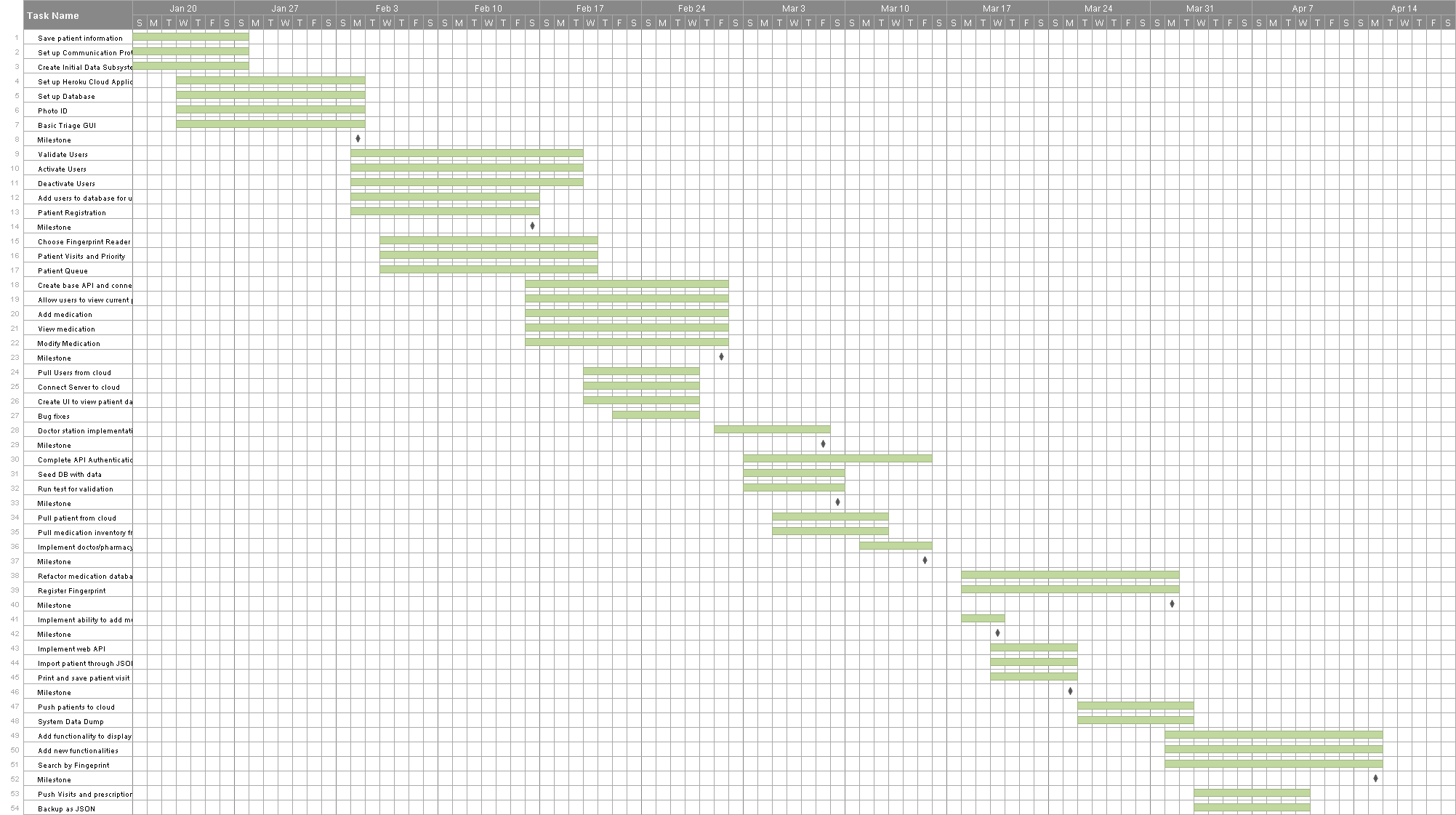
**iPad**: Is a line of tablet computers designed and marketed by Apple Inc., primarily as a platform for audio-visual media including books, periodicals, movies, music, games, apps and web content. Its size and weight fall between those of contemporary smartphones and laptop computers. The iPad runs on iOS.

**Xcode**: Software design by apple used for programming in Objective-C/ C.

**SDK**: Software Development Kit

# Appendix

## Appendix A - Project schedule (Gantt chart or PERT chart)



## Appendix B – All use cases with nonfunctional requirements

Use Case ID: MC-EMR001 – Login

* + Details:
    - Actor:
      * RIU
    - Pre-conditions:
      * The application must be open.
      * The server must be active
    - Description:
      1. Use case begins when the user presses the Login button.
      2. The information is encrypted and passed to the server
      3. A username and password match is found, login flag is set
      4. Use case end as soon as the system receives the flag and displays personal RIU information.
    - Post-conditions:
      * RIU should be logged in and should see his/her main screen.
  + Alternative Courses of Action
    - In step D.3 (step 3 of Description section) the server does not find a match and the login flag is not set.
  + Extensions:
    - None.
  + Exceptions:
    - If the system cannot establish connection with the Local Server an exception will be throw letting know the user that the action was not performed
  + Concurrent Uses:
    - None
  + Related Use Cases:
    - None

---------------------------------------------------------------------------------------------------------

Decision Support

* + Frequency: On average 4 requests are made daily by RIUs.
  + Criticality: High. RIUs cannot access their accounts without being able to login.
  + Risk: Low. Implementation of this use case employs standard web-based technology as well as mobile device.

Constraints:

* + Usability
    - Intuitive steps.
    - No help required.
  + Reliability
    - 4% failures for every twenty-four hours of operation is acceptable.
  + Performance
    - 5 seconds of delay is expected.
  + Supportability
    - The use case should be handled correctly by any iPad running iOS 6.0 or greater.
  + Implementation
    - iOS (Objective-C).

Use Case ID: MC-EMR002 – Logoff

* + Details:
    - Actor:
      * RIU
    - Pre-conditions:
      * The user must have been logged into the system.
    - Description:
      1. Use case begins when the user presses the Logout buton
      2. The system sends the usertoken over to the local server
      3. The local server destroys the user token
      4. Use case ends when the Local server sends a signal to the iPad that the user has logged off.
    - Post-conditions:
      * RIU should be in the login screen
  + Alternative Courses of Action
    - None
  + Extensions:
    - None.
  + Exceptions:
    - If the system cannot establish connection with the user’s token on the local server is not destroyed.
  + Concurrent Uses:
    - None
  + Related Use Cases:
    - None

---------------------------------------------------------------------------------------------------------

Decision Support

* + Frequency: On average 2 requests are made daily by RIUs.
  + Criticality: Medium. If a RIU cannot logout, this poses a security threat where a different user may use the system without having to login.
  + Risk: Low. Implementation of this use case employs standard web-based technology as well as mobile device.

Constraints:

* + Usability
    - Intuitive steps.
    - No help required.
  + Reliability
    - 3% failures for every twenty-four hours of operation is acceptable.
  + Performance
    - 3 seconds of delay is expected.
  + Supportability
    - The use case should be handled correctly by any iPad running iOS 6.0 or greater.
  + Implementation
    - iOS (Objective-C).

Use Case ID: MC-EMR003 – Create a New Patient

* + Details:
    - Actor:
      * RIN
    - Pre-conditions:
      * The RIN must have been logged into the system.
      * The RIN must be in the Registered Patient screen.
    - Description:
      1. Use case begins when a RIN fills in all the patient information and hits the “Register Patient” button
      2. The Client will store the patient information locally
      3. The Client will send the appropriate patient information to the local server
      4. The local server will store the appropriate patient information
      5. Use case ends when the Local server sends a signal to the iPad that the patient information has been saved.
    - Post-conditions:
      * RIN should be in the login screen
  + Alternative Courses of Action
    - None
  + Extensions:
    - None.
  + Exceptions:
    - If the system cannot establish connection with the Local Server, RIN will receive a notification that the patient information could not be saved in the Local Server
  + Concurrent Uses:
    - None
  + Related Use Cases:
    - None

---------------------------------------------------------------------------------------------------------

Decision Support

* + Frequency: On average 125 requests are made daily by RIUs.
  + Criticality: High. If a RIN cannot register a new patient then the RID or RIP cannot see the new patients and will not be able to use the system effectively.
  + Risk: Medium. Implementation of this use case employs a intricate manner of communicate between an iPad and a Macbook.

Constraints:

* + Usability
    - No previous training time
    - On average the user should take less than 1 minute to complete the request
  + Reliability
    - 6% failures for every twenty-four hours of operation is acceptable.
  + Performance
    - Request should be sent and saved withint 3 seconds
    - System should handle 175 requests in 1 minute
  + Supportability
    - The use case should be handled correctly by any iPad running iOS 6.0 or greater.
  + Implementation
    - iOS (Objective-C).

Use Case ID: MC-EMR004 – Register a Patients Fingerprint

* + Details:
    - Actor:
      * RIN
    - Pre-conditions:
      * The RIN must have been logged into the system.
      * The RIN must be in the Registered Patient screen.
    - Description:
      1. Use case begins when a RIN hits the “Register Patient Fingerprint” button
      2. The RIN presses the button corresponding to the appropriate finger the RIN is registering for the patient.
      3. The patient will swipe their finger on the biometric device
      4. The RIN will hit the “Save” button
      5. The Client will save the Fingerprint information on the Client and send the information to the Local Server
      6. Use case ends when the Local server sends a signal to the iPad that the Fingerprint information has been saved.
    - Post-conditions:
      * RIN should be returned to the patient registration page.
  + Alternative Courses of Action
    - None
  + Extensions:
    - None.
  + Exceptions:
    - If the system cannot establish connection with the Local Server, RIN will receive a notification that the Fingerprint information could not be saved in the Local Server
  + Concurrent Uses:
    - None
  + Related Use Cases:
    - None

---------------------------------------------------------------------------------------------------------

Decision Support

* + Frequency: On average 60 requests are made daily by RIUs.
  + Criticality: Low. A RIN can still properly use and identify a patient without registration their fingerprints.
  + Risk: Medium. Implementation of this use case utilises a third party API to interact with the biometric hardware.

Constraints:

* + Usability
    - Previous training is required to perform this operation
    - On average the user should take less than 2 minute to complete the request
  + Reliability
    - 10% failures for every twenty-four hours of operation is acceptable.
  + Performance
    - Request should be sent and saved within 6 seconds
  + Supportability
    - The use case should be handled correctly by any iPad running iOS 6.0 or greater.
  + Implementation
    - iOS (Objective-C).

Use Case ID: MC-EMR005 – Check a Patient Into Doctor Queue

* + Details:
    - Actor:
      * RIN
    - Pre-conditions:
      * The RIN must have been logged into the system.
      * The RIN must be in the Create New Visitation Page
    - Description:
      1. Use case begins when a RIN hits the “Send to Doctor” button
      2. The Client will send the patient information to the Local Server
      3. The Local Server will set the Patient’s inWaiting state to the Doctor Queue Flag
      4. Use case ends when the Local server sends a signal to the iPad verifying that the patient has been put into the queue.
    - Post-conditions:
      * RIN should be returned to the patient registration page.
  + Alternative Courses of Action
    - None
  + Extensions:
    - None.
  + Exceptions:
    - If the system cannot establish connection with the Local Server, RIN will receive a notification that the Patient was not moved into the Doctor Queue.
  + Concurrent Uses:
    - None
  + Related Use Cases:
    - None

---------------------------------------------------------------------------------------------------------

Decision Support

* + Frequency: On average 125 requests are made daily by RIUs.
  + Criticality: High. If a RIN cannot move a patient into the Doctor Queue, the Doctor or Pharmacist will not be able to use the system.
  + Risk: Low. Implementation of this use case utilises a third party API to communicate between the Local Server and the Client.

Constraints:

* + Usability
    - No previous training required
    - On average the user should take less than 20 seconds to complete the request
  + Reliability
    - 8% failures for every twenty-four hours of operation is acceptable.
  + Performance
    - Request should be sent and saved within 4 seconds
  + Supportability
    - The use case should be handled correctly by any iPad running iOS 6.0 or greater.
  + Implementation
    - iOS (Objective-C).

Use Case ID: MC-EMR006 – Quick Checkout a Patient

* + Details:
    - Actor:
      * RIN
    - Pre-conditions:
      * The RIN must have been logged into the system.
      * The RIN must be in the Create New Visitation Page
    - Description:
      1. Use case begins when a RIN hits the “Quick Checkout” button
      2. The Client will send the patient information to the Local Server
      3. The Local Server will close the Patients visitation.
      4. Use case ends when the Local server sends a signal to the iPad verifying that the patient’s visitation has been closed and the patient has been checked out.
    - Post-conditions:
      * RIN should be returned to the patient registration page.
  + Alternative Courses of Action
    - None
  + Extensions:
    - None.
  + Exceptions:
    - If the system cannot establish connection with the Local Server, RIN will receive a notification that the Patient was not moved into the Doctor Queue.
  + Concurrent Uses:
    - None
  + Related Use Cases:
    - None

-------------------------------------------------------------------------------------------------------

Decision Support

* + Frequency: On average 25 requests are made daily by RIUs.
  + Criticality: Low. A RIN can still pass a patient through the system without checking out a patient quickly.
  + Risk: Low. Implementation of this use case utilises a third party API to communicate between the Local Server and the Client.

Constraints:

* + Usability
    - No previous training required
    - On average the user should take less than 5 seconds to complete the request
  + Reliability
    - 3% failures for every twenty-four hours of operation is acceptable.
  + Performance
    - Request should be sent and saved within 4 seconds
  + Supportability
    - The use case should be handled correctly by any iPad running iOS 6.0 or greater.
  + Implementation
    - iOS (Objective-C).

Use Case ID: MC-EMR007 – Search Patient

* + Details:
    - Actor:
      * RIU
    - Pre-conditions:
      * The RIU must have been logged into the system.
      * The RIU must be in the Search Patient screen
    - Description:
      1. Use case begins when a RIU fills in the Patient Name and hits the “Search” button
      2. The Client sends the patient name over to the Local Server
      3. The Local Server looks for the patient information
      4. Use case ends when the Local server sends a signal to the iPad that the patient information has been saved.
    - Post-conditions:
      * RIU should display the patient information
  + Alternative Courses of Action
    - None
  + Extensions:
    - None.
  + Exceptions:
    - If the system cannot establish connection with the Local Server, RIU will receive a notification that the patient information could not be found in the Server
  + Concurrent Uses:
    - None
  + Related Use Cases:
    - None

---------------------------------------------------------------------------------------------------------

Decision Support

* + Frequency: On average 5 requests are made daily by RIUs.
  + Criticality: Low. The likelihood of the Triage administering a patient is low due to the fact they they are a moving clinic throughout Africa
  + Risk: Low. Implementation of this use case employs a intricate manner of communicate between an iPad and a Macbook.

Constraints:

* + Usability
    - No previous training time
    - On average the user should take less than 1 minute to complete the request
  + Reliability
    - 2% failures for every twenty-four hours of operation is acceptable.
  + Performance
    - System should handle 80 requests in 1 minute
  + Supportability
    - The use case should be handled correctly by any iPad running iOS 6.0 or greater.
  + Implementation
    - iOS (Objective-C).

---------------------------------------------------------------------------------------------------------

Use Case ID: MC-EMR008 – Add visit to existing patient

* + Details:
    - Actor:
      * RIN
    - Pre-conditions:
      * The RIN must have been logged into the system.
      * The RIN must be in the Search Results screen
    - Description:
      1. Use case begins when a RIN selects a patient from the Search Results screen
      2. The New Patient Visit screen will be shown
      3. The RIN fills in the appropriate patient information
      4. The RIN selects either Checkout or Check-In
      5. The iPad saves the new visit and attempts to sync with the Local Server
    - Post-conditions:
      * iPad will now display the New Patient screen.
  + Alternative Courses of Action
    - Select the Triage back button
  + Extensions:
    - None.
  + Exceptions:
    - If the system cannot establish connection with the Local Server, RIN will receive a notification that the patient information could not be saved on the server.
  + Concurrent Uses:
    - None
  + Related Use Cases:
    - None

Use Case ID: MC-EMR009 – Edit Patient Information

* + Details:
    - Actor:
      * RIN
    - Pre-conditions:
      * The RIN must have been logged into the system.
      * The RIN must have searched for an existing
    - Description:
      1. Use case begins when a RIN selects a patient image
      2. The Client will enable all the text Fields
      3. Upon the RIN clicking “Save”, the Client will save the Patient information and send it to the Local Server
      4. The Local Server will save the Patient information
      5. Use case ends when the Local server sends a signal to the iPad that the patient information has been edited.
    - Post-conditions:
      * RIN will be in the patient detail screen.
  + Alternative Courses of Action
    - None
  + Extensions:
    - None.
  + Exceptions:
    - If the system cannot establish connection with the Local Server, RIN will receive a notification that the patient information could not be saved in the Local Server
  + Concurrent Uses:
    - None
  + Related Use Cases:
    - None

---------------------------------------------------------------------------------------------------------

Decision Support

* + Frequency: On average 35 requests are made daily by RINs.
  + Criticality: Medium. If a RIN cannot edit a patient, the system can still run but there may be a high chance for error.
  + Risk: Medium. Implementation of this use case employs a intricate manner of communicate between an iPad and a Macbook.

Constraints:

* + Usability
    - No previous training time
    - On average the user should take less than 5 minute to complete the request
  + Reliability
    - 8% failures for every twenty-four hours of operation is acceptable.
  + Performance
    - Request should be sent and saved within 3 seconds
    - System should handle 45 requests in 1 minute
  + Supportability
    - The use case should be handled correctly by any iPad running iOS 6.0 or greater.
  + Implementation
    - iOS (Objective-C).

-------------------------------------------------------------------------------------------------------

Modification History

Owner: Steven Berlanga

Initiation date: 05/23/2012

Date last modified: 05/23/2012

Use Case ID: MC-EMR010 – Register medication given to patients

* + Details:
    - Actor:
      * RID
    - Pre-conditions:
      * The RID must have been logged into the system.
      * The RID must be at the Current Diagnosis screen
    - Description:
      1. Use case begins when a RID selects the Submit button
      2. The RID selects the Find Drug button
      3. After being presented with the Choose Medicine screen, the user selects the medication to prescribe
      4. RID presses the Select button
      5. After being presented with the updated Choose Medication screen, the RID selects the checkout button
      6. The iPad will save the patient visitation and attempt to sync with the Local Server
      7. Use case ends when the iPad displays the Patient Queue screen.
    - Post-conditions:
      * RID will be in the Patient Queue.
  + Alternative Courses of Action
    - RID can selected the Doctor back button to return to the Patient Queue
  + Extensions:
    - None.
  + Exceptions:
    - None
  + Concurrent Uses:
    - None
  + Related Use Cases:
    - None

Use Case ID: MC-EMR011 – Diagnose a Patient

* + Details:
    - Actor:
      * RID
    - Pre-conditions:
      * The RID must have been logged into the system.
      * The RID must be in the patient Visitation screen
    - Description:
      1. Use case begins when a RID fills in the diagnosis information and hits “Add Prescription” button
      2. The Client saves the visitation information
      3. The Client passes the visitation information over to the Local Server
      4. The Local Server saves the visitation information
      5. Use case ends when the Local server sends a signal to the iPad that the visitation information has been saved.
    - Post-conditions:
      * RID will be in the prescribe medicine screen.
  + Alternative Courses of Action
    - None
  + Extensions:
    - None.
  + Exceptions:
    - If the system cannot establish connection with the Local Server, RIN will receive a notification that the patient information could not be saved in the Local Server
  + Concurrent Uses:
    - None
  + Related Use Cases:
    - None

---------------------------------------------------------------------------------------------------------

Decision Support

* + Frequency: On average 130 requests are made daily by RINs.
  + Criticality: High. If a patient visitation can’t be saved, then Pharmacist can’t use the system..
  + Risk: Medium. Implementation of this use case employs a intricate manner of communicate between an iPad and a Macbook.

Constraints:

* + Usability
    - No previous training time
    - On average the user should take less than 10 minute to complete the request
  + Reliability
    - 4% failures for every twenty-four hours of operation is acceptable.
  + Performance
    - Request should be sent and saved within 6 seconds
    - System should handle 45 requests in 1 minute
  + Supportability
    - The use case should be handled correctly by any iPad running iOS 6.0 or greater.
  + Implementation
    - iOS (Objective-C).

Use Case ID: MC-EMR012 – Prescribe Medicine to Patient

* + Details:
    - Actor:
      * RIP
    - Pre-conditions:
      * The RIP must have been logged into the system.
      * The RIP must be in the patient Prescription screen
    - Description:
      1. Use case begins when a RIP selects the prescription and clicks the “Prescribe Now” button
      2. The Client saves the prescription information
      3. The Client passes the prescription information over to the Local Server
      4. The Local Server saves the prescription information
      5. Use case ends when the Local server sends a signal to the iPad that the prescription information has been saved.
    - Post-conditions:
      * RIP will be in the patient prescription medicine screen.
  + Alternative Courses of Action
    - None
  + Extensions:
    - None.
  + Exceptions:
    - If the system cannot establish connection with the Local Server, RIP will receive a notification that the prescription information could not be saved in the Local Server
  + Concurrent Uses:
    - None
  + Related Use Cases:
    - None

---------------------------------------------------------------------------------------------------------

Decision Support

* + Frequency: On average 115 requests are made daily by RIPs.
  + Criticality: High. If a prescription cannot be filled there is no way to now what medicine the doctors have left.
  + Risk: Medium. Implementation of this use case employs a intricate manner of communicate between an iPad and a Macbook.

Constraints:

* + Usability
    - No previous training time
    - On average the user should take less than 3 minute to complete the request
  + Reliability
    - 6% failures for every twenty-four hours of operation is acceptable.
  + Performance
    - Request should be sent and saved within 4 seconds
    - System should handle 75 requests in 1 minute
  + Supportability
    - The use case should be handled correctly by any iPad running iOS 6.0 or greater.
  + Implementation
    - iOS (Objective-C).

Use Case ID: MC-EMR013 – Pull Information to Local Server

* + Details:
    - Actor:
      * RLSU
    - Pre-conditions:
      * The RLSU must have been logged into the system.
    - Description:
      1. Use case begins when a RLSU presses the “Pull Cloud Information”
      2. The Local Server sends a signal to the Cloud API
      3. The Cloud API packages the appropriate database information into a json response.
      4. Use case ends when the Local Server parses and receives the JSON response from the Cloud API
    - Post-conditions:
      * The Local Server now has the JSON response stored into the it’s SQLite DB.
  + Alternative Courses of Action
    - None
  + Extensions:
    - None.
  + Exceptions:
    - None
  + Concurrent Uses:
    - None
  + Related Use Cases:
    - None

---------------------------------------------------------------------------------------------------------

Decision Support

* + Frequency: On average 230 requests are made daily by a RLSU.
  + Criticality: High. If information cannot be pulled down into the Local Server then the Server cannot maintain the most recent information
  + Risk: Medium. Implementation of this use case employs a intricate manner of communicate between a Mac OSX system and a rails API.

Constraints:

* + Usability
    - No previous training time
    - On average the user should take less than 13 seconds to complete the request
  + Reliability
    - 1% failures for every twenty-four hours of operation is acceptable.
  + Performance
    - Request should be sent and saved within 4 seconds
    - System should handle 130 requests in 1 minute
  + Supportability
    - The use case should be handled correctly by any iPad running iOS 6.0 or greater.
  + Implementation
    - iOS (Objective-C).

Use Case ID: MC-EMR014 – Push Information to Local Server

* + Details:
    - Actor:
      * RLSU
    - Pre-conditions:
      * The RLSU must have been logged into the system.
    - Description:
      1. Use case begins when a RLSU presses the “Push Cloud Information”
      2. The Local Server packages up an object into a JSON object and sends a signal to the Cloud API
      3. The Cloud API receives the information, parses it and saves the object
      4. Use case ends when the Local Server receives and ok signal from the Cloud API
    - Post-conditions:
      * The Local Server now has the JSON response stored into the it’s SQLite DB.
  + Alternative Courses of Action
    - None
  + Extensions:
    - None.
  + Exceptions:
    - If the JSON post is not what the API expects, the API will return and error code.
  + Concurrent Uses:
    - None
  + Related Use Cases:
    - None

---------------------------------------------------------------------------------------------------------

Decision Support

* + Frequency: On average 230 requests are made daily by a RLSU.
  + Criticality: High. If information cannot be pushed up into the Cloud then the Cloud will not hold any of the most recent information.
  + Risk: Medium. Implementation of this use case employs a intricate manner of communicate between a Mac OSX system and a rails API.

Constraints:

* + Usability
    - No previous training time
    - On average the user should take less than 13 seconds to complete the request
  + Reliability
    - 1% failures for every twenty-four hours of operation is acceptable.
  + Performance
    - Request should be sent and saved within 4 seconds
    - System should handle 130 requests in 1 minute
  + Supportability
    - The use case should be handled correctly by any iPad running iOS 6.0 or greater.
  + Implementation
    - iOS (Objective-C).

Use Case ID: MC-EMR015 – Sync iPad information with Local Server

* + Details:
    - Actor:
      * RLSU
    - Pre-conditions:
      * The RLSU must have been logged into the system.
    - Description:
      1. Use case begins when a RLSU presses the “Sync iPad Information”
      2. The Local Server sends a signal to the Client application
      3. The Client application will check if there is any dirty information.
      4. If there is any dirty information, the Client application packages up the information
      5. Use case ends when the Local Server receives the Client information and parses it.
    - Post-conditions:
      * The Local Server now has the Client response stored into the it’s SQLite DB.
  + Alternative Courses of Action
    - None
  + Extensions:
    - None.
  + Exceptions:
    - None
  + Concurrent Uses:
    - None
  + Related Use Cases:
    - None

---------------------------------------------------------------------------------------------------------

Decision Support

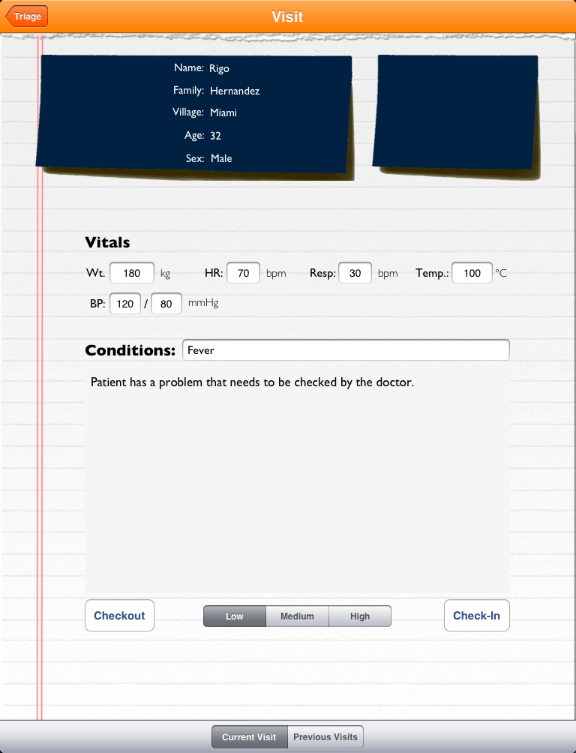
* + Frequency: On average 50 requests are made daily by a RLSU.
  + Criticality: High. If information cannot be synced between the Client applications then there will be an inconsistency between the information.
  + Risk: Medium. Implementation of this use case employs a third party API to communicate between iOS and Mac OSX applications

Constraints:

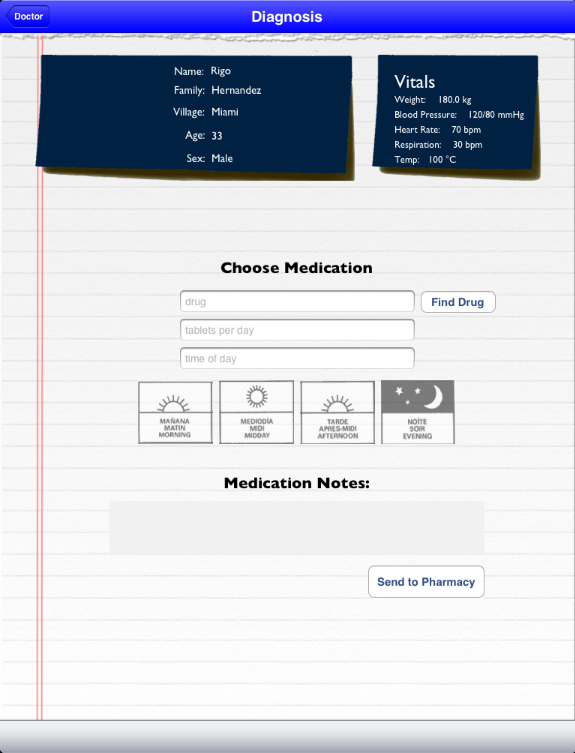
* + Usability
    - No previous training time
    - On average the user should take less than 13 seconds to complete the request
  + Reliability
    - 1% failures for every twenty-four hours of operation is acceptable.
  + Performance
    - Request should be sent and saved within 4 seconds
    - System should handle 130 requests in 1 minute
  + Supportability
    - The use case should be handled correctly by any iPad running iOS 6.0 or greater.
  + Implementation
    - iOS (Objective-C).

## Appendix C – User Interface designs

## Macintosh HD:private:var:folders:l_:63gtxm9n6md0z__23yvr805c0000gn:T:Messages:Transfers:Messages Image(423717536).png











## Appendix D – Analysis models (static and dynamic)

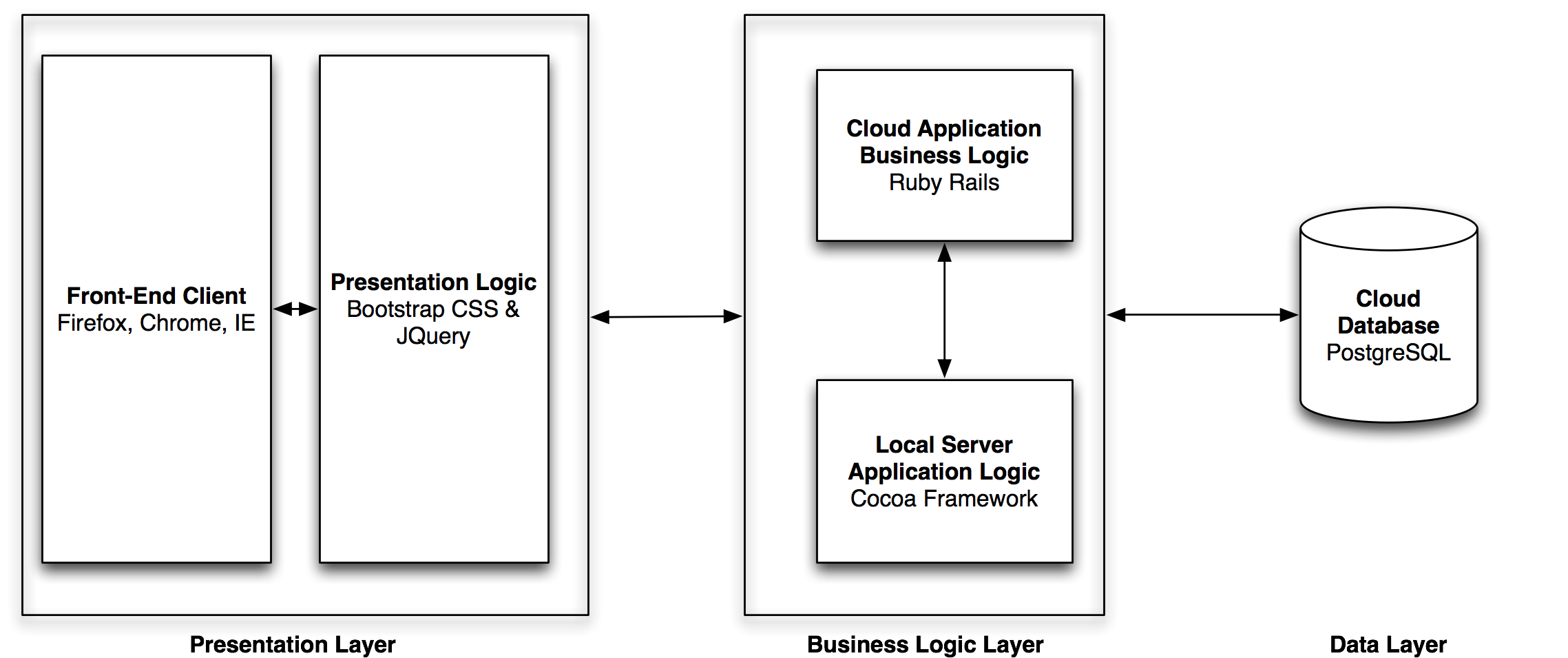


Figure 9 Cloud Application Architecture

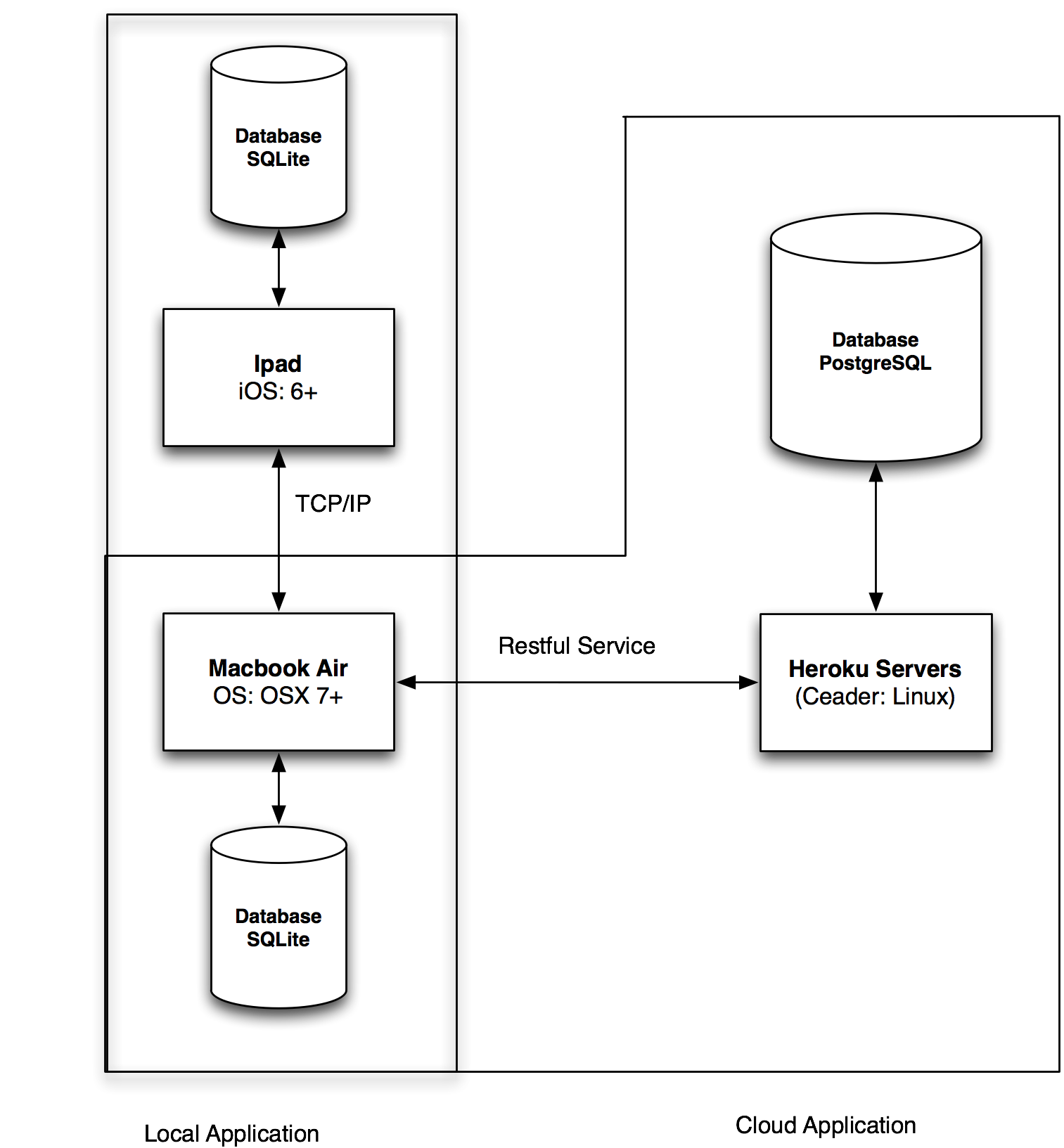


Figure Deployment Diagram

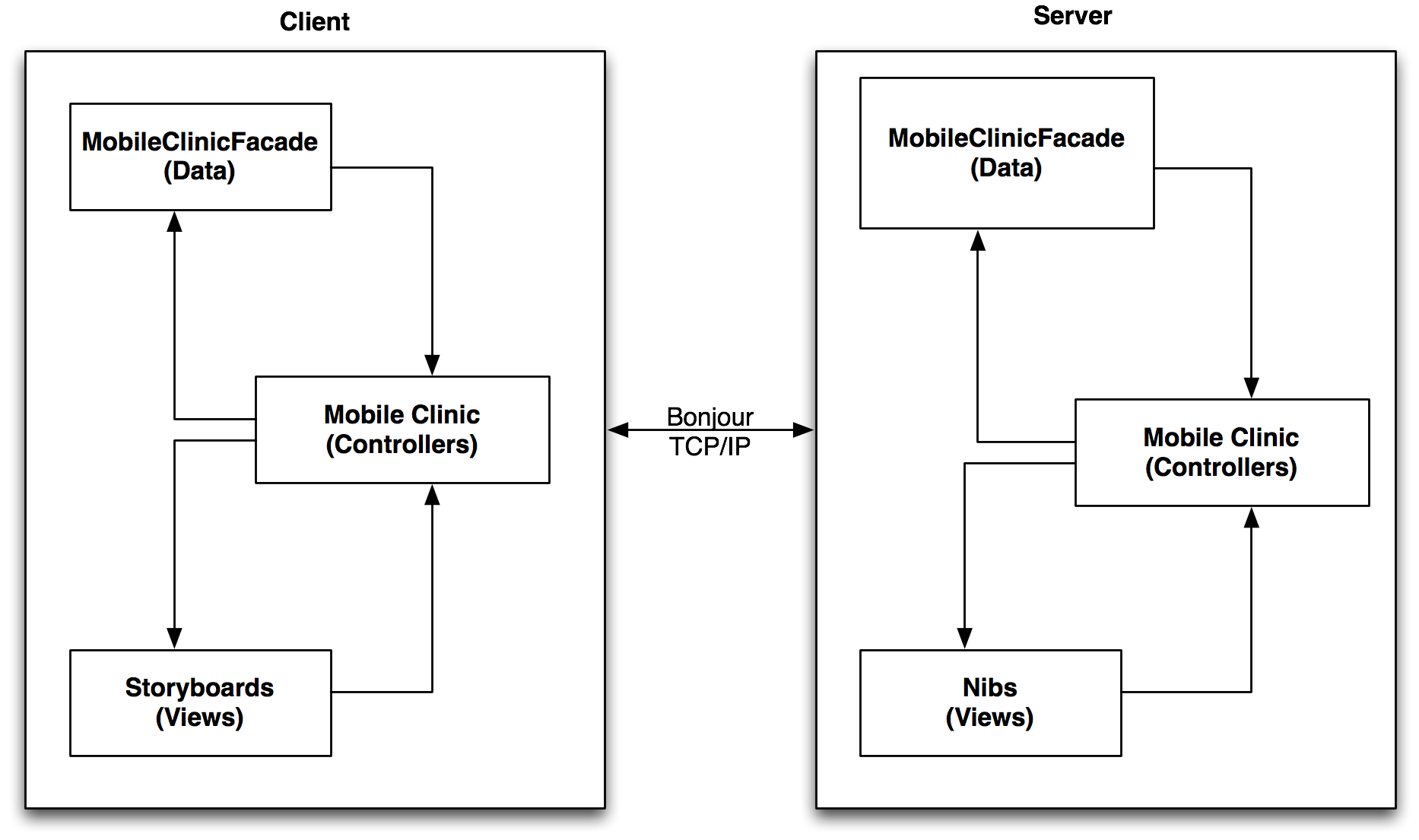


Figure 11 Field Application Architecture

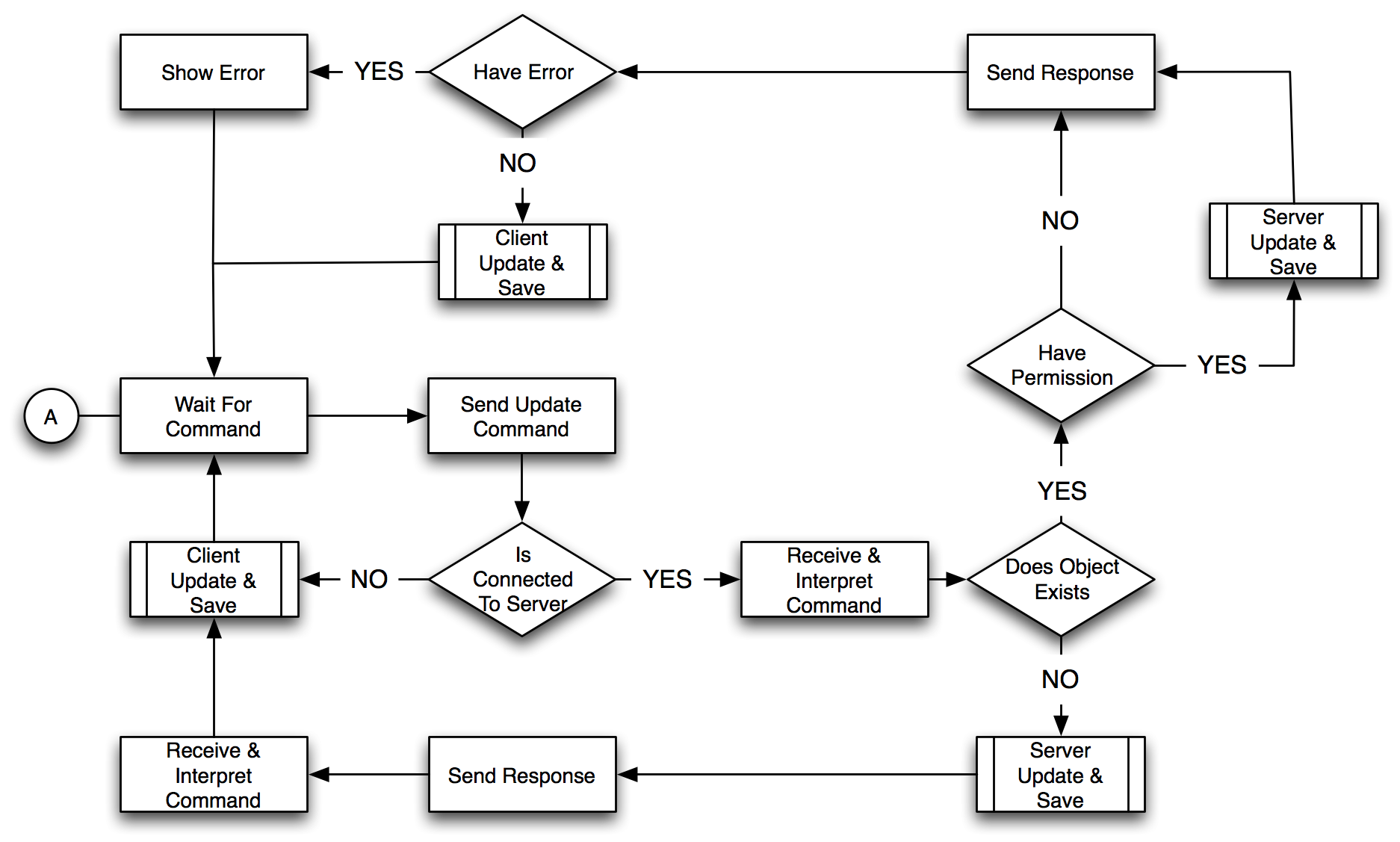
****

Figure State Diagram of Object Update Algorithm

## Appendix E – Design models (static and dynamic)

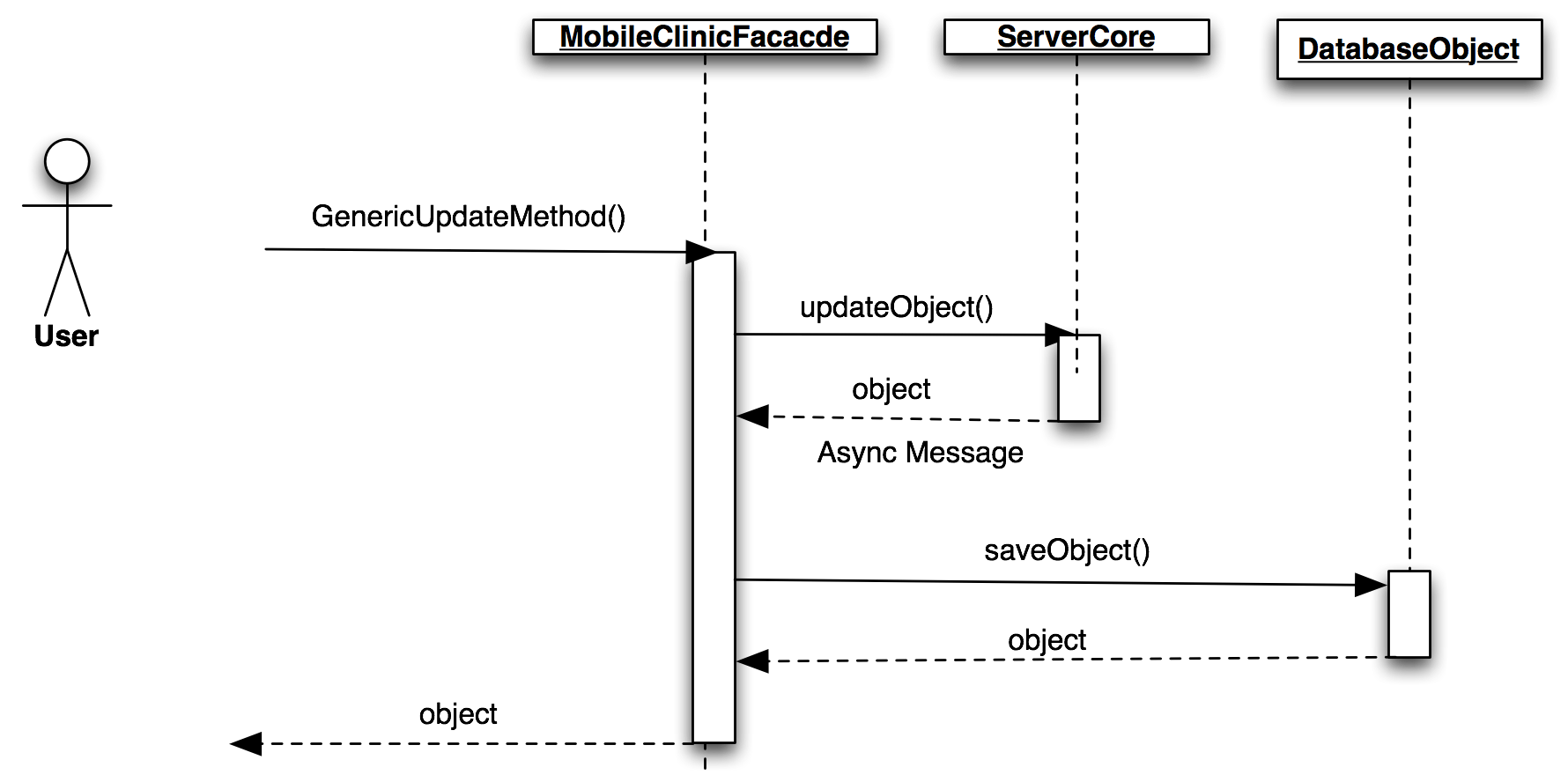


Figure Sequence Diagram of Update Visit (Client-Side)

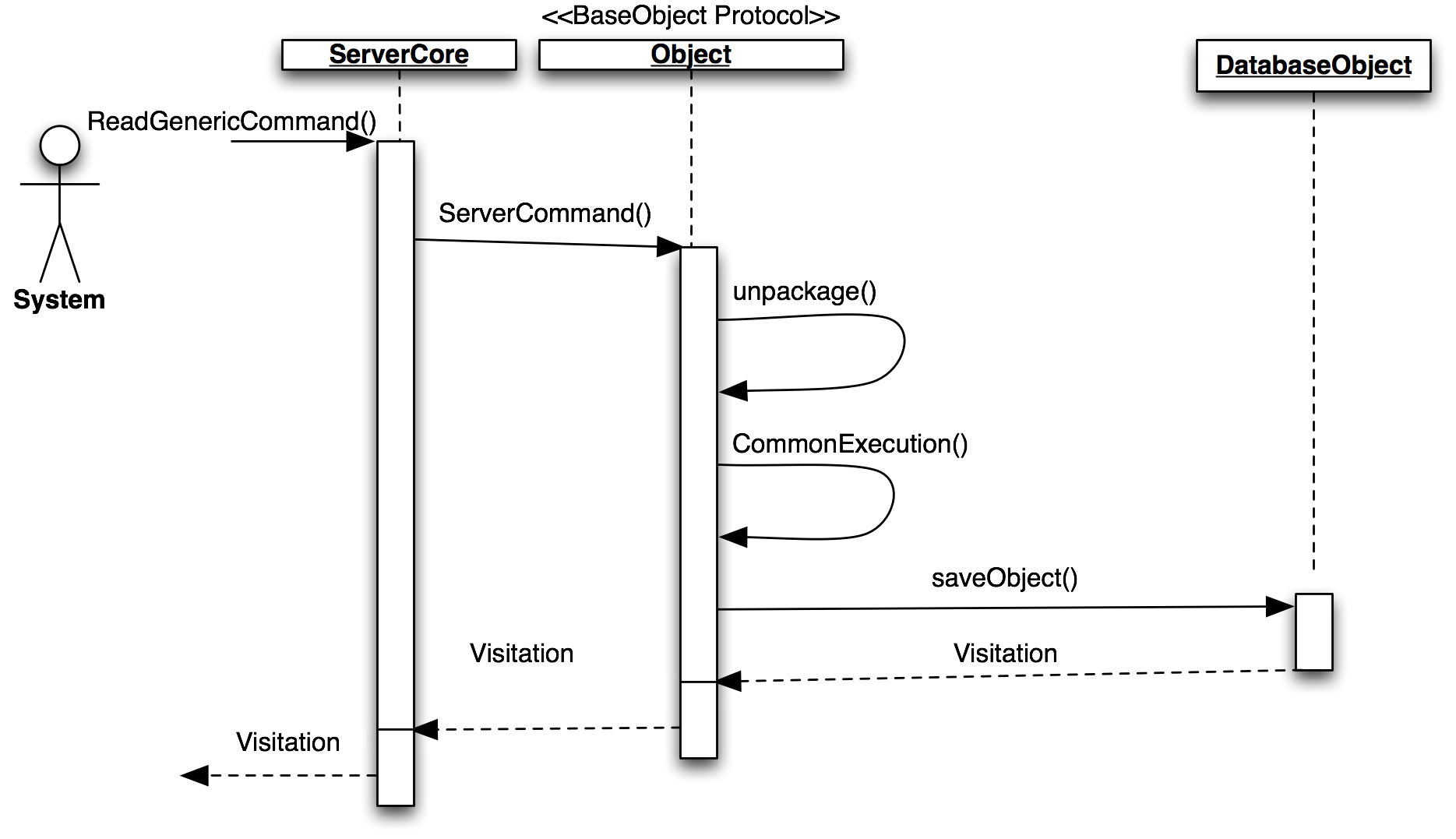


Figure Sequence Diagram Update Visit (Server-Sided)

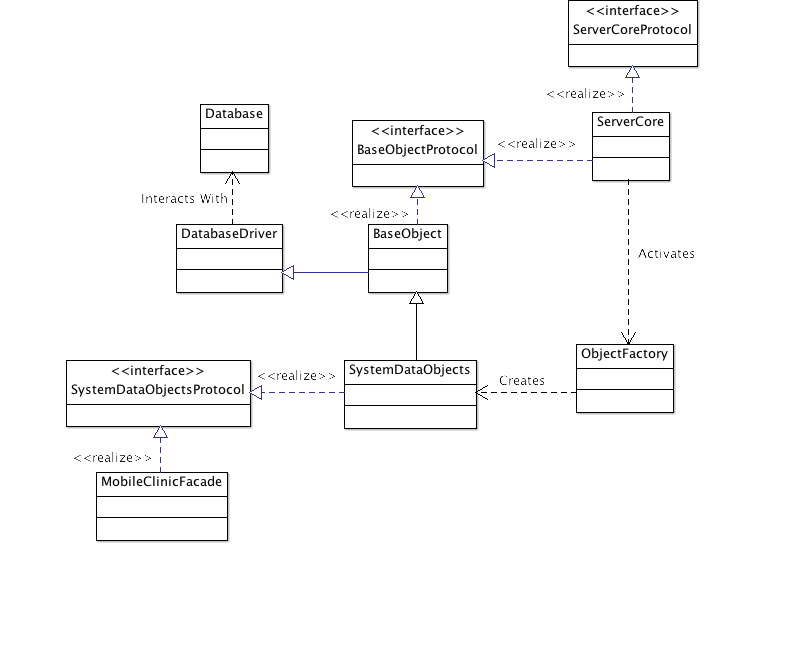
****

Figure Class Diagram of Data Subsystem

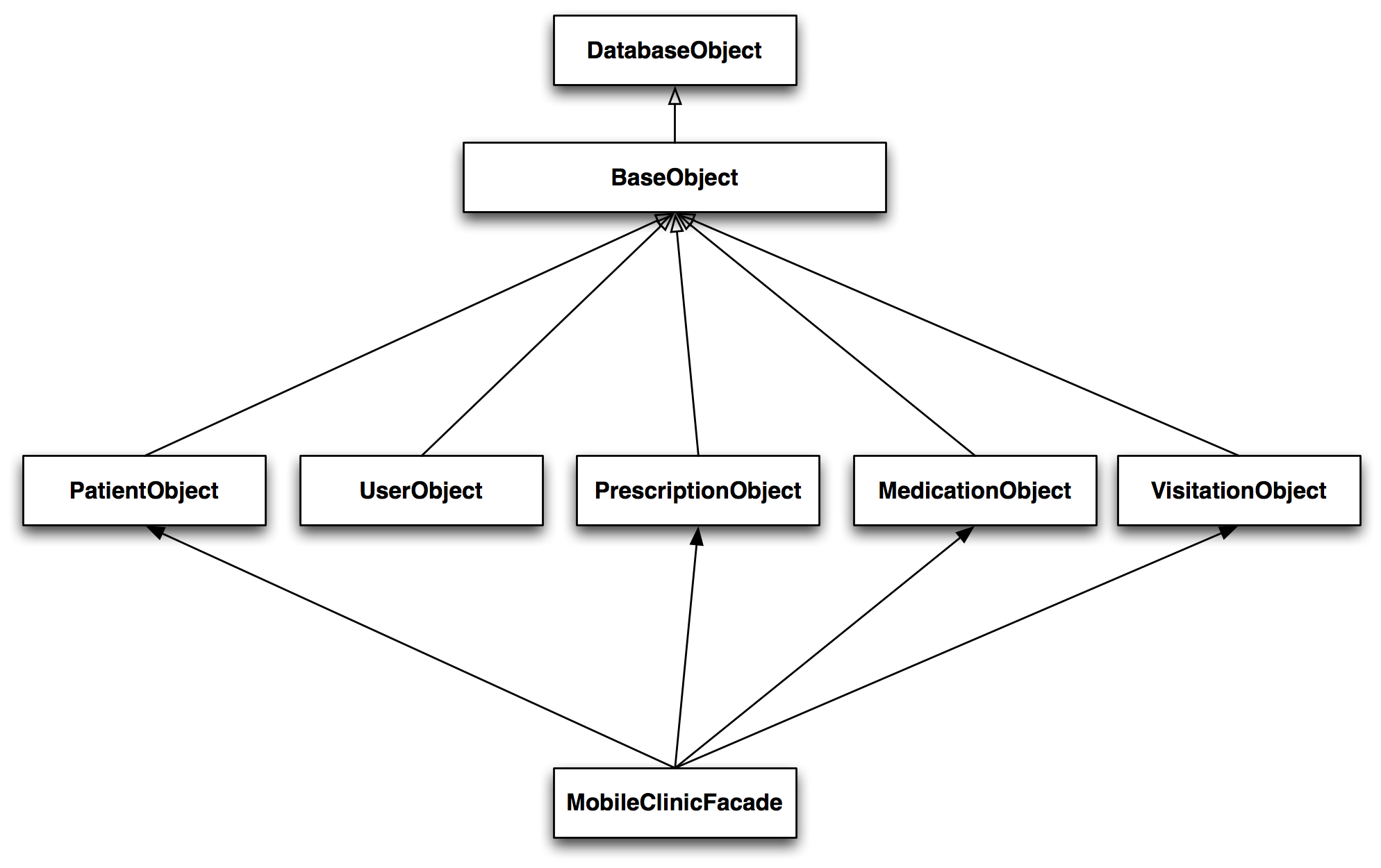


Figure System Objects

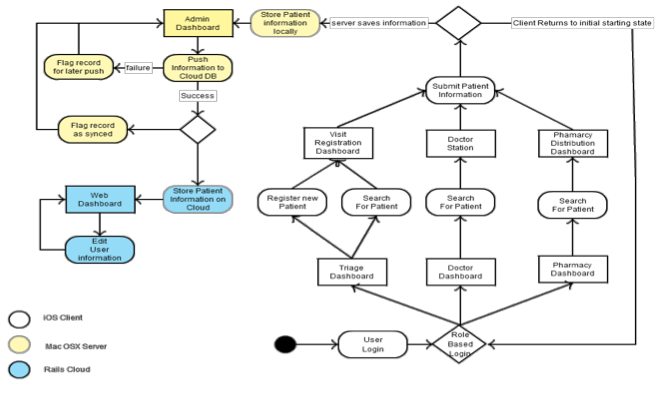


Figure 4.3.2

## Appendix F – Documented Class interfaces (code) and constraints

[Click Here For Code Documentation](../Documentation/rtf/refman.rtf)

## Appendix G – Documented code for test drivers and stubs

[Click Here For Code Documentation](../Documentation/rtf/refman.rtf)

## Appendix H – Diary of meeting and tasks for the entire semester

|  |  |  |
| --- | --- | --- |
| Meeting | 1 | 1/29/2013 |
| Attendance | Everyone |  |
| Review | Accepted Stories and Tasks |  |
|  | Unacceptable Stories |  |
|  | Time limit |  |
|  | Limited communication |  |
|  | Synchronization |  |
|  | Learning curve |  |
|  | Client-Server Implementation |  |
|  | Flojack API |  |
|  | Architecture Review |  |
|  | Acceptability Tests |  |
|  | Automated Tests |  |
|  | Story Map Refactoring |  |
| Next | Allow synchronization |  |
|  | Client and Server |  |
|  | iPad GUI |  |
|  | MVP of Triage |  |
|  | Login/Logout |  |
|  | Patient Creation |  |

|  |  |  |
| --- | --- | --- |
| Meeting | 2 | 2/5/2013 |
| Attendance | Everyone |  |
| Review | Accepted Stories and Tasks |  |
|  | Unacceptable Stories |  |
|  |  | Detailed Vitals |
|  |  | Sending patient data |
|  | Problems |  |
|  |  | Time |
|  |  | Synchronization |
|  | Solution |  |
|  |  | Git |
|  |  | Standard Schema |
|  | Architecture Review |  |
|  | iPad Client Architecture |  |
|  | OSC Architecture |  |
|  | WebApp Architecture |  |
|  | Story map Refactoring |  |
| Next | Diagnosis |  |
|  |  | Assign diagnosis |
|  | Sync iPad and Server |  |
|  | iPad GUI |  |
|  | MVP of Diagnosis |  |

|  |  |  |
| --- | --- | --- |
| Meeting | 3 | 2/12/2013 |
| Attendance | Everyone |  |
| Review | Storied attempted |  |
|  |  | Stories Removed |
|  |  | Stories Completed |
|  |  | In Progress |
|  | Tasks |  |
|  |  | Completed |
|  |  | In progress |
|  |  | Not initiated |
|  |  | Deleted |
|  |  | Removed |
|  | Architecture |  |
|  |  | Web Services |
|  |  | Client |
|  |  | Server |
|  | Internal Architecture |  |
|  |  | Presentation Layer |
|  |  | Objects |
|  |  | Base Object |
|  |  | Database |
|  | Class Diagram |  |
|  | Acceptability Tests |  |
|  | User Stories |  |
|  |  | I want to see all patient in system |
|  |  | Add medication to webApp |
|  |  | Update medication |

|  |  |  |  |
| --- | --- | --- | --- |
| Meeting | 4 | 2/26/ |  |
| Attendance | Everyone |  |  |
| Review | Stories Burn down |  |  |
|  | Tasks Run down |  |  |
|  | System Architecture |  |  |
|  |  | Client |  |
|  |  |  | iPad |
|  |  |  | Bonjour |
|  |  |  | CoreData |
|  |  | Web |  |
|  |  |  | Bootstrap CSS |
|  |  |  | jSQuery |
|  |  | API |  |
|  |  |  | Restful |
|  |  |  | Heroku |
|  |  | Server |  |
|  |  |  | MacBook |
|  |  |  | CoreData |
|  |  |  | OSX |
|  |  |  | Bonjour TCP |
|  |  |  | PostgreSQL |
|  | Acceptability Tests |  |  |
|  |  | Add visitation |  |
|  |  | Easy access to forms |  |
|  |  | See visitation records |  |
|  | Stories |  |  |
|  |  | Add medication through web |  |
|  |  | Update medication as pharmacist |  |

For a more detailed review or the meetings please view the products in ‘Documentation > Meetings’ folder.

# References

## Open Source Code

|  |  |
| --- | --- |
| System | Open Source Code Name |
| Client & Server | GCDAsyncSocket |
| Client | AJNotification |
| MBProgressHUD |

## Licensing

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

**0. PREAMBLE**

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

**1. APPLICABILITY AND DEFINITIONS**

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

**2. VERBATIM COPYING**

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

**3. COPYING IN QUANTITY**

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

**4. MODIFICATIONS**

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

* A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
* B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
* C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
* D. Preserve all the copyright notices of the Document.
* E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
* F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
* G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
* H. Include an unaltered copy of this License.
* I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
* J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
* K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
* L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
* M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
* N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
* O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

**5. COMBINING DOCUMENTS**

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

**6. COLLECTIONS OF DOCUMENTS**

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

**7. AGGREGATION WITH INDEPENDENT WORKS**

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

**8. TRANSLATION**

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

**9. TERMINATION**

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

**10. FUTURE REVISIONS OF THIS LICENSE**

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

**11. RELICENSING**

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.