**Design Document**

**Florida International University**

**CIS 4911 Senior Project (U01) – Spring 2013**

**Professor: Masoud Sadjadi**

**Mentor: Steven Luis**

**Client: Orant Charities**

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

**Steven Berlanga**

**Carlos Corvaia**

**Rigoberto Hernandez**

**Michael Montaque**

**Sebastian Zanlongo**

**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**

Table of Contents

1. Introduction 1

1.1 Purpose of the System 1

1.2 Functional and Non-Functional Requirements 1

1.3 Design Methodology 2

1.4 Definitions, Acronyms, and Abbreviations 1

1.5 Overview of Document 1

2. Proposed Software Architecture 2

2.1 Architecture Overview 2

2.2 Subsystem Decomposition 3

2.2.1 Field Application 3

2.2.2 Cloud Application 3

2.3 Hardware and Software Mapping 4

2.4 Persistent Data Management 4

2.5 Security/Privacy 6

3. Detail Design 7

3.1 Overview 7

3.2 Design Patterns 8

3.4 Dynamic Model 10

3.5 Code Specification 12

4. Glossary 13

Appendix A 14

Admin Stories 14

Triage Stories 14

Doctor Stories 15

Pharmacist Stories 15

Appendix B 16

Implemented Stories 16

Appendix C 18

Appendix D 19

Meetings 19

Appendix E 20

Open Source Code 20

# 1. Introduction

This section will introduce the purpose and scope of the system, as well as the requirements design methodology and acronyms and abbreviations that will be seen throughout the documentation.

## 1.1 Purpose of the System

The purpose of MC-EMR is to assist medical personnel in triaging, tracking and processing patients in Internet void areas. This system will allow the field users to stay in sync while they gather and manipulate patient information and, when Internet connection is available, allow field users to upload their information and all home users in a separate remote area to see the information collected.

## 1.2 Functional and Non-Functional Requirements

|  |  |
| --- | --- |
| ID | Title |
| US-2 | The system shall manually register a patient |
| US-5 | The system shall allow users to Diagnose a patient |
| US-11 | The system shall allow pharmacist to checkout patients |
| US-13 | The system shall show the pharmacist the doctor’s prescription |
| US-15 | The system shall sort by priority |
| US-25 | The system shall allow users to shutdown restart and startup the local server |
| US-37 | As a potential user I want to create my own user profile so I can be properly identified in the system |
| US-43 | as a physician i want to manually identify a patient so that i can find their records |
| US-44 | As a triage nurse I want to login into the system so that I can work securely |
| 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-70 | As a triage nurse i want to link family members together so that i can quickly bring up other patients |
| 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-103 | As an administrator I want the server to connect with the physical Device so that information can be persistent throughout the system |
| 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-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-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-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. |

## 1.3 Design Methodology

We are using the Agile Scrum to complete this system. We collaborate with the client to establish the requirements and from those requirements the project manager creates the user stories. Also, during this phase, the system and object design, hardware & software, database tables, and design goals are established. These stories are then prioritized and the team creates appropriate tasks for the stories to be completed.

## 1.4 Definitions, Acronyms, and Abbreviations

**3T:** 3 Tier Architecture

**MVC:** Model View Controller

**MC-EMR:** Mobile Clinic-Electronic Medical Records

**SQLite**: database residing on the iOS device

**MFC:** MobileClinicFacade – a class responsible for providing a simple interface to interact with the network and database protocols

## 1.5 Overview of Document

This document will present the organization of the existing system. This document will also describe how the problems faced, solutions, how the roles are distribute for this phase, and the software tools need it make the system. It will describe in detail the system’s design, features, constraints and limitations, the organization of the code, and the safety of the system. Finally this document will also provide graphical documents to illustrate certain concepts, and an appendix and glossary for further understanding.

# 2. Proposed Software Architecture

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.

## 2.1 Architecture Overview

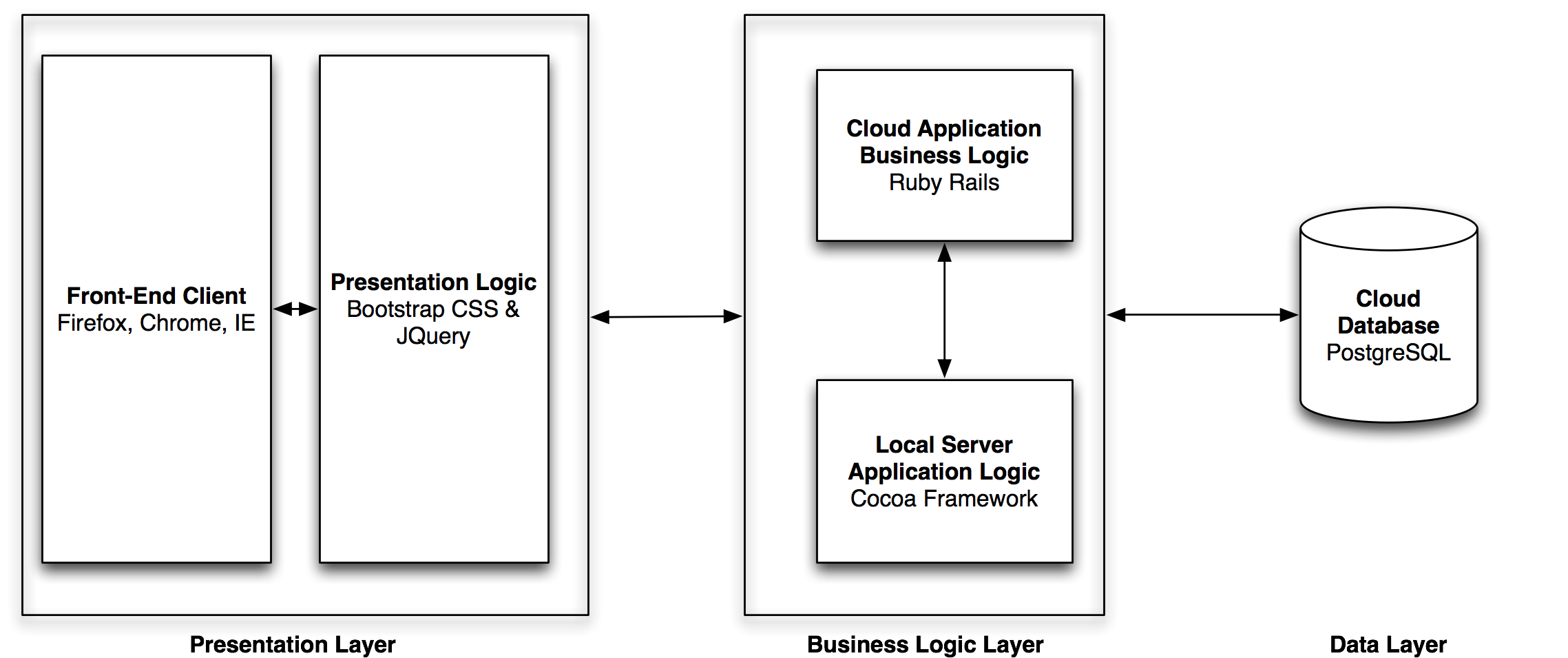
MC-EMR is broken into two major subsystems. The cloud application uses a 3T architecture. The other is a client server architecture, using iPads as clients and a MacBook Air as a server. 

Figure Cloud Application Architecture

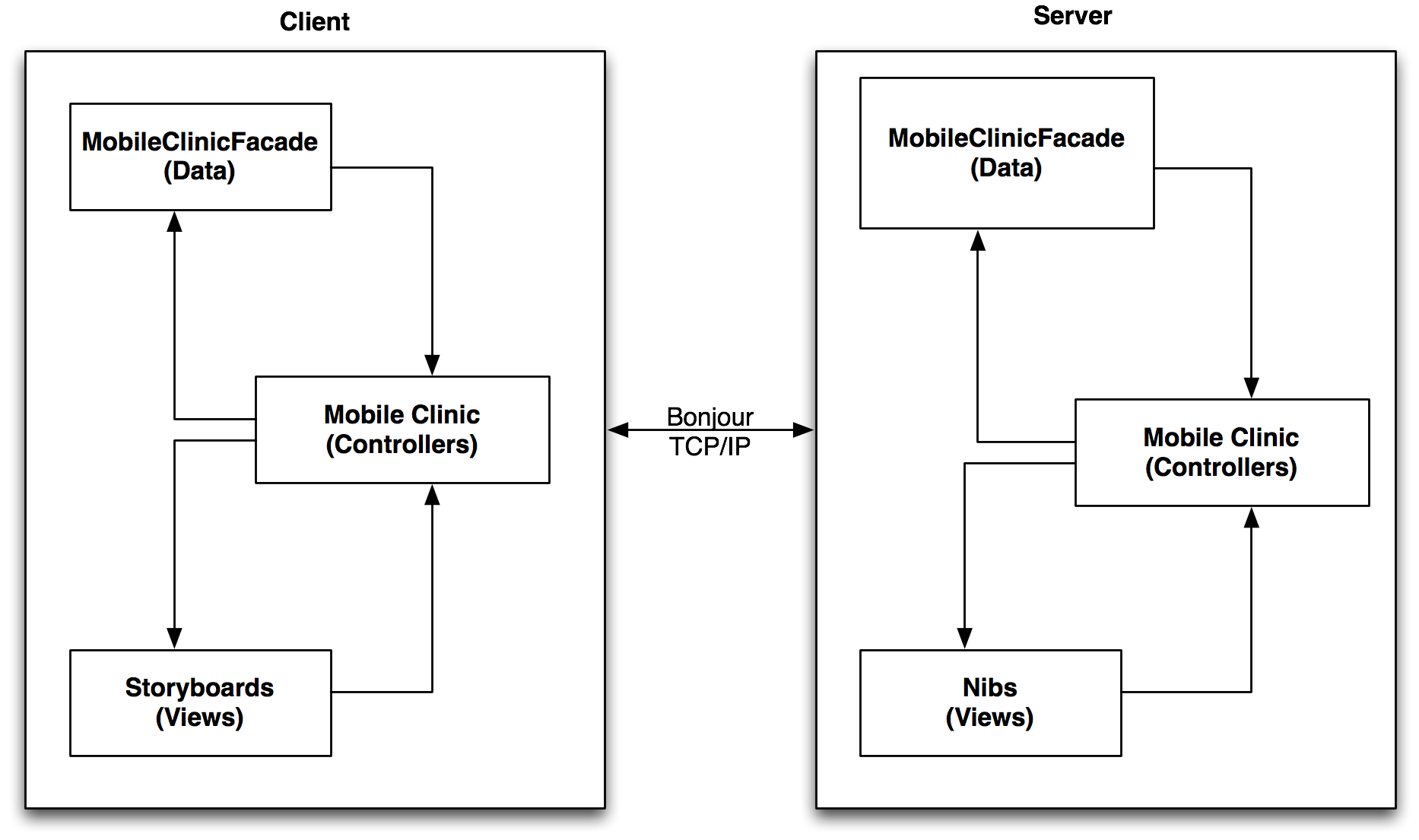


Figure Field Application Architecture

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.

## 2.2 Subsystem Decomposition

### 2.2.1 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.

### 2.2.2 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.

## 2.3 Hardware and Software Mapping

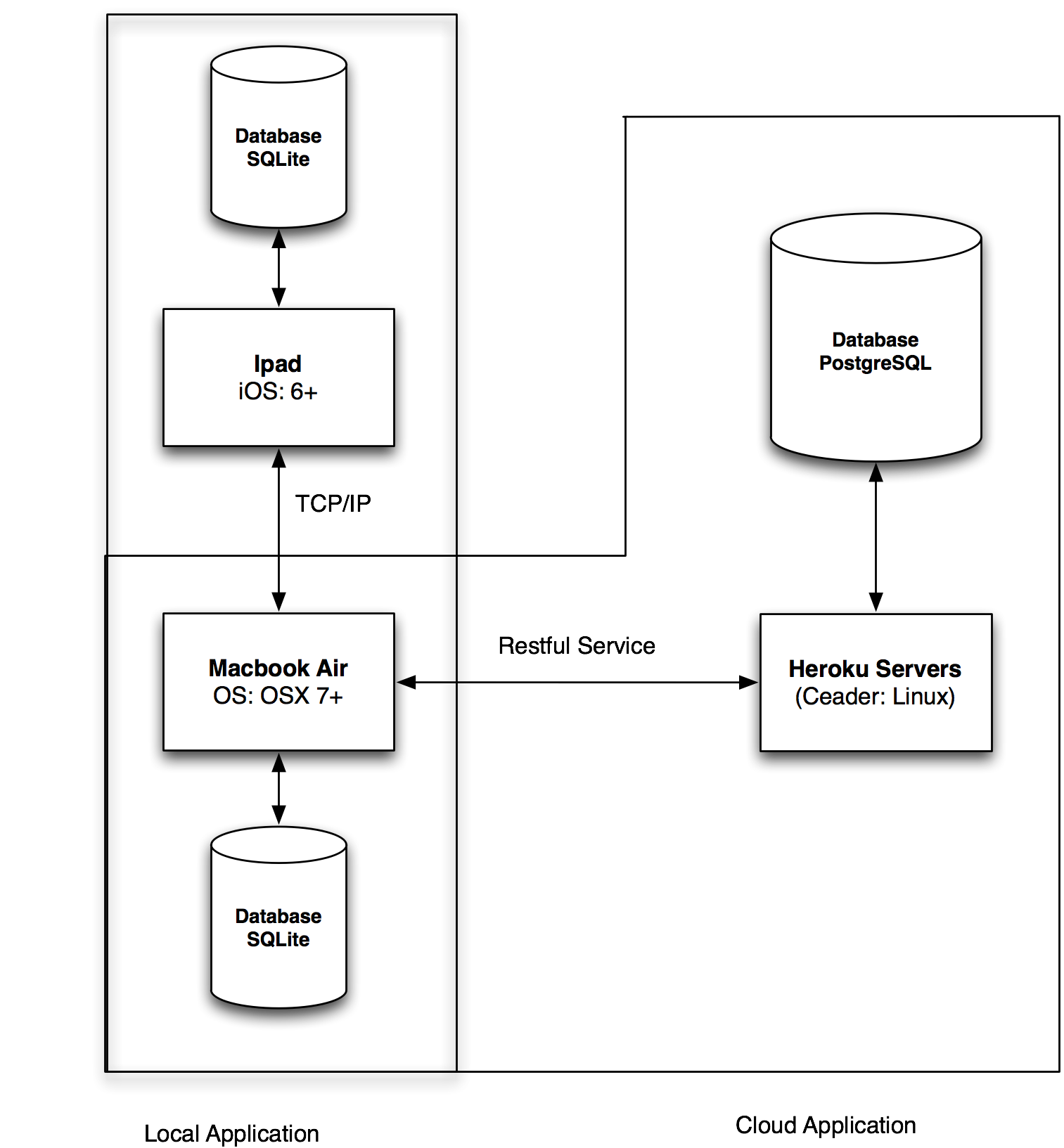


Figure Deployment Diagram

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 |

## 2.4 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.

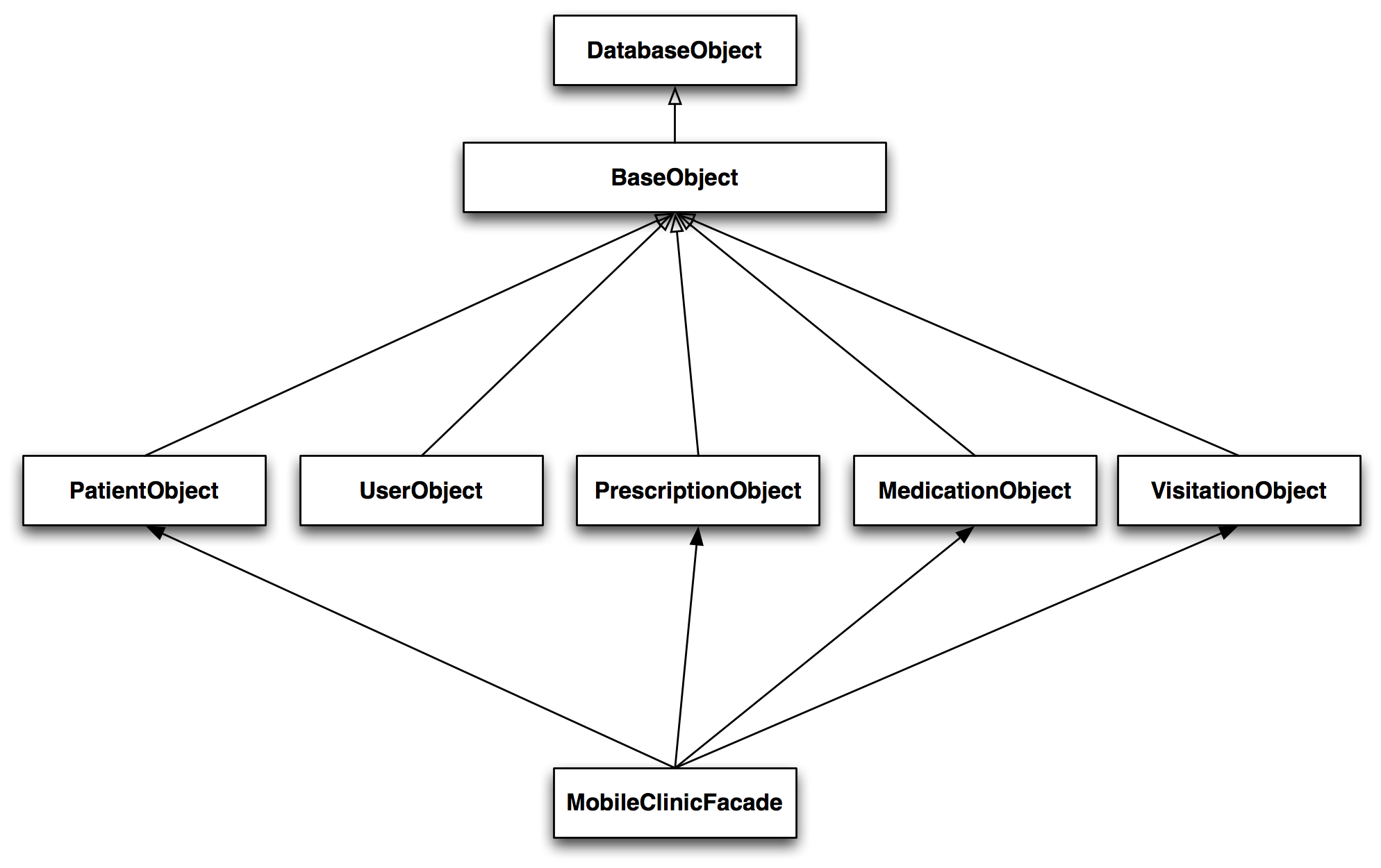


Figure System Objects

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.

## 2.5 Security/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.

# 3. Detail 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.

## 3.1 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 |

## 3.2 Design Patterns

**Façade**

The MobileClinicFacade allows users to interact with the Server and database. It minimizes coupling yet neatly couples commonly used commands to make avoid redundancies.

**Singleton**

The ServerCore and DatabaseObject are singletons. The ServerCore is responsible for generating threads to handle connection between client and server. As a result only one object should generate threads. The DatabaseObject needs to be a singleton is to avoid objects from creating duplicate data.

**Command**

The ServerCore needs to keep a light implementation and must adapt quickly to new methods and objects. The Command pattern allows the server to call on method and defers the execution of the command till the appropriate object is capable of handling it. This allows the ServerCore to be multi-functional and protect against coupling and large unmanageable code.

**Factory**

The ServerCore uses the factor to create objects of the correct type. When information is received by the Core it needs to first figure out which object the information belongs to. The generic information given to the Core will contain the key for the factory to create the appropriate object.

**Observer**

The both the local server and client uses the observer patterns to send messages to objects that are too far away or to help decouple the system. This pattern allows the objects that are created early in the system interface with other objects that are created way later down the line and have no dependency on each other.

**3.3 Static Model**

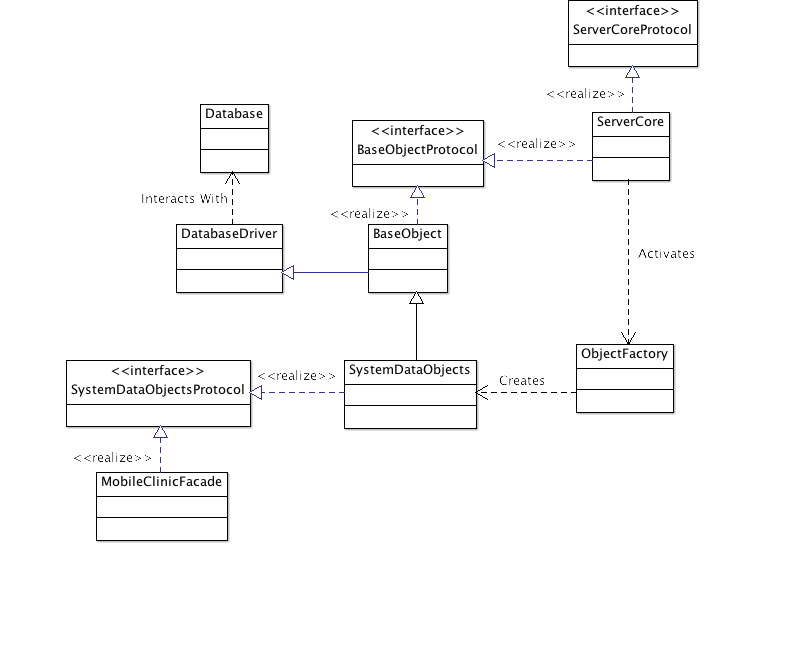
****

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.

## 3.4 Dynamic Model

**Update & Save**

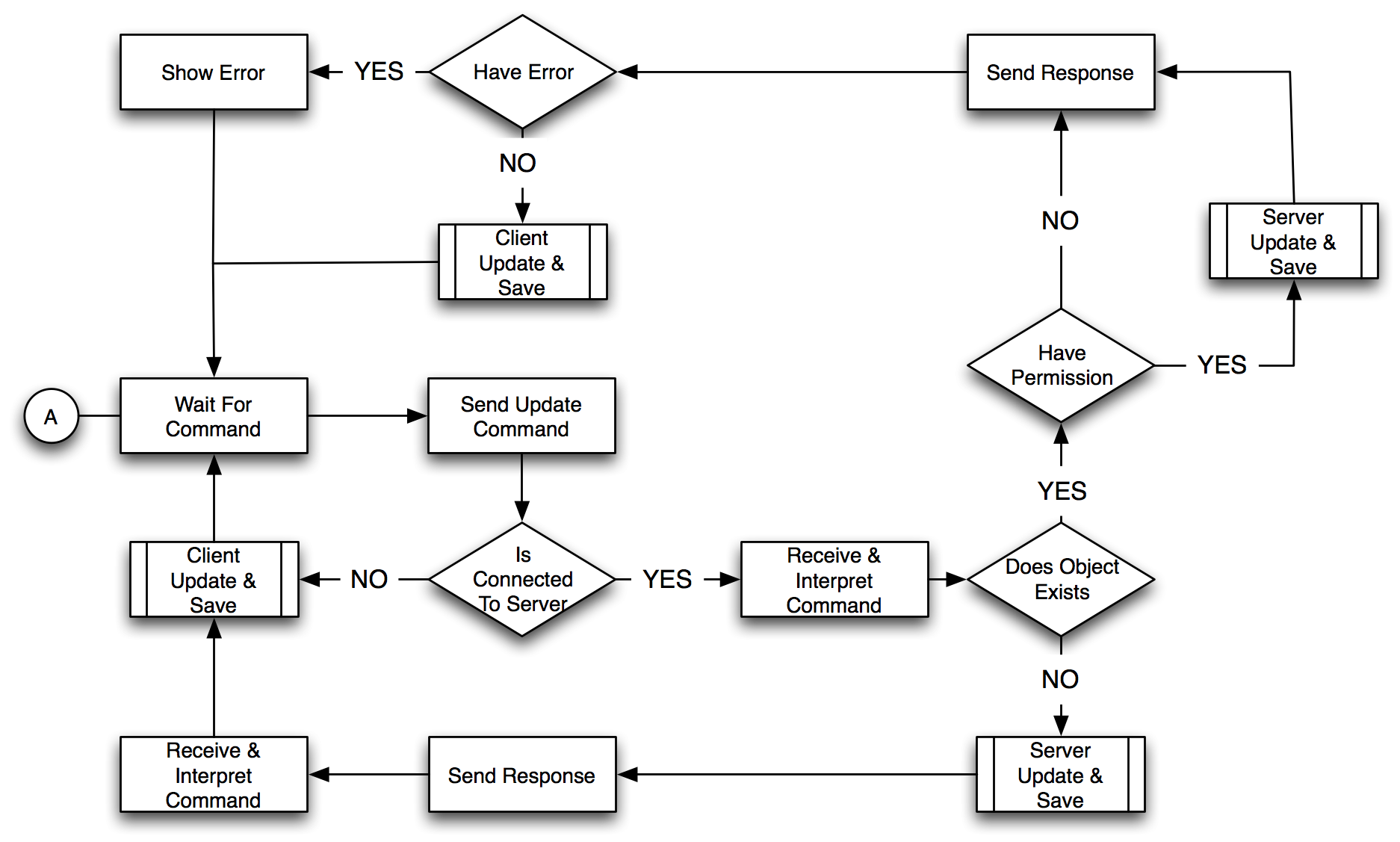
****

Figure 7 State Diagram of Object Update Algorithm

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.

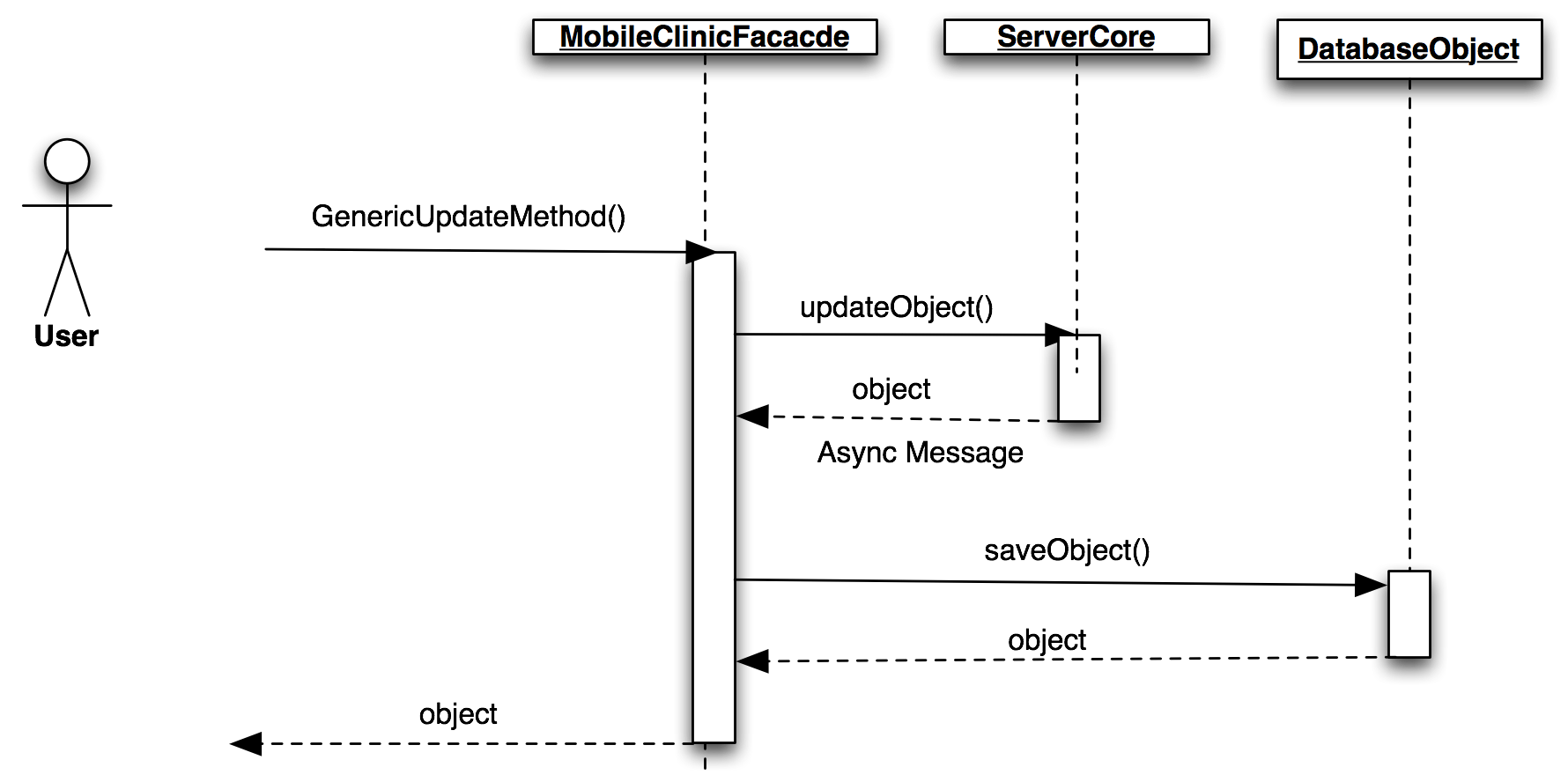


Figure Sequence Diagram of Update Visit (Client-Side)

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.

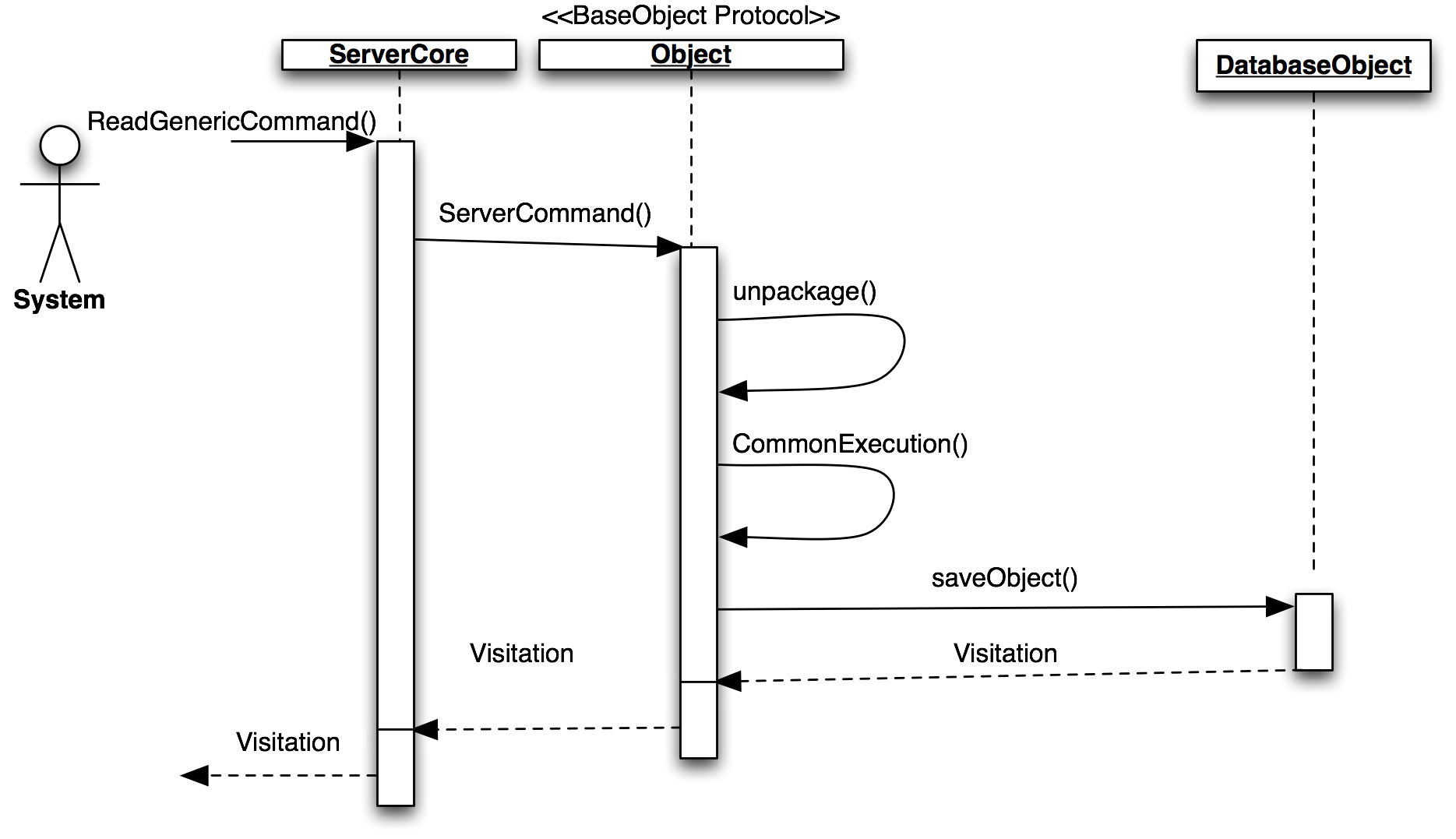


Figure Sequence Diagram Update Visit (Server-Sided)

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.

## 3.5 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.

# 4. 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 A

## Admin Stories

## Triage Stories

## Doctor Stories

## Pharmacist Stories

# Appendix B

## Implemented Stories

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

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

# Appendix D

## Meetings

|  |  |  |
| --- | --- | --- |
| 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.

# Appendix E

## Open Source Code

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