

Capstone Project: Physician Portal Application

CSC450 Projects for Computer Science Majors

Michael A Crowell

Carroll University

August 22, 2013

**TABLE OF CONTENTS**

<b>Project Proposal.....</b>	<b>3</b>
<b>Feasibility Study.....</b>	<b>5</b>
<b>Work Plan.....</b>	<b>7</b>
<b>A Brief Explanation of HL7 and Healthcare Integration .....</b>	<b>8</b>
<b>Software Engineering Artifacts .....</b>	<b>10</b>
Software Requirements Specification .....	10
Risk Management .....	15
Software Architecture Document .....	16
Software Project Management Plan .....	37
<b>Test Plans .....</b>	<b>54</b>
<b>Code Summary .....</b>	<b>57</b>
<b>User Manual .....</b>	<b>80</b>
<b>Reflection .....</b>	<b>88</b>
<b>Self Evaluation .....</b>	<b>91</b>

**PROJECT PROPOSAL**

Physicians groups consist of a group of physicians that may provide services through a number of different clinics. They may also provide services within a hospital setting. When they order tests or treatments for their patients they will be performed by various ancillary departments, such as laboratory, radiology, or physical therapy departments. Each ancillary department will transmit the patient results back to the hospital or clinic that they are associated with. For a physician group that services multiple organizations this can create a complex environment of disparate systems.

In this type of environment the physicians need to have access to each of the different systems in order to view their patient results. Where the patient results are delivered and stored will depend on where the service was performed. Managing this type of environment creates a level of confusion and complexity for the physicians and their staff that can lead to delays and affect how they are able to provide patient care. The proposed solution is to provide a single portal for the physicians that will store all of their results from any of the entities that provide their services. This will improve efficiency in the office and it will also help to avoid mistakes caused by the current need to look for information in multiple systems. Having all of the patient data in a single database will also allow the physicians group to perform statistical analysis against the data, such as providing the group a list of the most common tests that the physicians order. Such analysis again will help to improve the efficiency of the office. The physician portal will receive the data from the disparate systems via HL7 integration methodologies, which is the common method of integration in the healthcare industry. Providing HL7 import and parsing capabilities for the physician portal will be a key

component of this project. The portal itself will be developed as a web application to remove system and platform dependencies so the physicians can access their patient information from remote locations.

**FEASABILITY STUDY****Overview:**

Physicians groups consist of a group of physicians that may provide services through a number of different clinics. They may also provide services within a hospital setting.

When they order tests or treatments for their patients they will be performed by various ancillary departments, such as laboratory, radiology, or physical therapy departments.

Each ancillary department will transmit the patient results back to the hospital or clinic that they are associated with. For a physician group that services multiple organizations this can create a complex environment of disparate systems.

In this type of environment the physicians need to have access to each of the different systems in order to view their patient results. Where the patient results are delivered and stored will depend on where the service was performed. Managing this type of environment creates a level of confusion and complexity for the physicians and their staff that can lead to delays and affect how they are able to provide patient care.

**Recommendation:**

The proposed solution is to provide a single portal for the physicians that will store all of their results from any of the entities that provide their services.

**Expected Benefits:**

This will improve efficiency in the office and it will also help to avoid mistakes caused by the current need to look for information in multiple systems. Having all of the patient data in a single database will also allow the physicians group to perform statistical analysis against the data, such as providing the group a list of the most common tests that

the physicians order. Such analysis again will help to improve the efficiency of the office.

**Market Analysis:**

There are many different companies that offer various types of physician portal solutions. Many of the larger healthcare software developers, such as Epic and Cerner, offer a portal solution that is an overlay to their own clinic EHR (Electronic Health Record) products. These solutions can be costly, but also limited, in that they are designed to integrate with a single specific software product. Although it is a competitive market, I feel there is room for a low cost alternative that has the ability to integrate with software products from different vendors.

**WORK PLAN****Project Plan Overview**

Below are the tasks associated with the development of this project.

Task	Completion Date	Description
Project proposal	05/15/2013	Delivered online to Professor Konemann.
Feasibility study, detailed project requirement specification, and the work plan.	05/22/2013	Delivered online to Professor Konemann.
Technology research	05/22/2013	I will need time to research some of the technology needed for this project that I am less familiar with, such as MVC development, AJAX, and JQuery.
Architectural design	05/29/2013	System design, class diagram, design patterns to be used.
Database design, build database	05/31/2013	ER diagram and build of the database.
HL7 import process	06/12/2013	The routine to parse and import the HL7 formatted file.
Unit test HL7 import process	06/12/2013	
System security	06/12/2013	
Unit test security	06/12/2013	
Update on progress	06/17/2013	Meet with Professor Konemann to discuss project status.
Web forms design	06/30/2013	
Patient lookup, results display, LINQ queries	07/20/2013	
Update on progress	07/22/2013	Meet with Professor Konemann to discuss project status.
Statistical reports	07/31/2013	
Unit test statistical reports	07/31/2013	
Data maintenance screens	08/05/2013	
Unit test data maintenance screens	08/05/2013	
Final release version of application	08/10/2013	
Integration testing	08/14/2013	Testing of full integrated application.
Final documentation	08/17/2013	Completion of all deliverables needed for capstone project.
Final presentation	08/21/2013	Presentation of capstone project

**A BRIEF EXPLANATION OF HL7 AND HEALTHCARE INTEGRATION**

HL7, which stands for Health Level 7, is a standardized message format used by the health care industry. The HL7 protocol defines the rules by which two applications will exchange data in an orderly way. The protocol architecture is hierarchical, moving from high-level groupings and structures to a set of several hundred data fields. Each level of the hierarchy serves a different organizing purpose. Areas of the protocol are grouped according to common application function; for example, ADT, Order Entry, Finance, Control, and Ancillary Reporting all represent groups described in the standard. Different functional groups are typically given individual chapters in the HL7 specification document. Within a functional group are defined one or more message types that can be implemented in various combinations to support high-level business rules for the applications involved. For example, ADT only specifies one message type while Order Entry describes more than a dozen. Segments provide a logical grouping for data elements. For example, the Patient Identification segment (PID) includes fields for such identifying information as patient name, Social Security number, medical record number, account number, and miscellaneous demographic details. How fields are grouped in segments forms part of the HL7 implied data model. Segments can be required or optional, can be nested, and can repeat. Each segment then contains a nested hierarchy of data elements in the form of fields, components, and sub components that are also optional or can repeat.

**Here is an example ADT message, which transmits patient demographic information.**

```
MSH|^~\&|Hospital System|Hospital Test|Physician Portal|Physician  
Group|200912071648||ADT^A08|20091207164815862|P|2.3  
EVN|A08|200912071648  
PID|1|003|15862^^^PRMCARESP||TEST003^NUMBER3||19450103|M||5833 N.WEST
```



CIRCLE AVE.^CHICAGO^IL^60631|(773)763-8403||M||360-36-9744  
 PV1||O|^PRMCARESP  
 GT1|1|4744|RUE^SHERWIN^|5833 N.WEST CIRCLE  
 AVE.^CHICAGO^IL^60631|(773)763-8403||19390321|M||SPO|||R  
 IN1|1|EDI00003|B008|BC/BS OF ILLINOIS PPO|P.O. BOX  
 805107^CHICAGO^IL^60680|(800)972-  
 8088|P16602||20050101||RUE^SHERWIN^|SPO|19390321|5833 N.WEST CIRCLE  
 AVE.^CHICAGO^IL^60631||1|||CTY000338393|||M  
 IN1|2|DEF|DEF|^|TEST^TIMMY^D|SEL|19451024|5833 N.WEST CIRCLE  
 AVE.^CHICAGO^IL^60631||2|||F  
 IN1|3|DEF|DEF|^|TEST^TIMMY^D|SEL|19451024|5833 N.WEST CIRCLE  
 AVE.^CHICAGO^IL^60631||3|||F  
 IN2|1|R  
 IN2|2|360-36-9744|  
 IN2|3|360-36-9744|

**Here is an example ORM message, which transmits the information for a patient order.**

MSH|^~\&|LH81|AA302||20120524005334||ORM^O01|9416615|P|2.3  
 PID|1|007|00545982||TEST007^NUMBER7^|19460628|F||^  
 ^|||1684881251|||C  
 PV1||A||H81146^ALEKSANDR GALPERIN, M.D.|||||Q 0405977|  
 GT1|1||KARNAUCHOVA^NIJOLE|^|U||OT||  
 ORC|NW|^ACL|Q 0405977THINDNA|Q  
 0405977|IP||00001^ONCE^201205240057^R||20120524005334||0000582A|1  
 OBR|1|007444||BIOPSY||201205231800|||201205230039||1^ALEKSANDR  
 GALPERIN, M.D.||Q 0405977AA302||20120524005334||1||^R

**Here is an example ORU message, which transmits the patient's result report.**

MSH|^~\&|COMLAB|ACL|IDX|ACERN|20091027113620||ORU^R01|31|P|2.2|||  
 PID|1||80393010||MOCK^HMP||19850909|F|||325322222A||11621|546315455|||  
 PV1||O|HMP^N/A^N/A^N/A||225511|||  
 ORC|RE|1186542719|1186542719C991201122|||225511|||  
 OBR|1|006222|006222|PDF^PDF^||200910071515||999|MC||200910071615|BRLA^  
 BRONCHOALVEOLAR LAVAGE|225511|IDX||C991201122||MB|P|^R|||  
 OBX|1|TX|SOURCE^SPECIMEN DESCRIPTION:|1|BRONCHOALVEOLAR  
 LAVAGE|||P||200910071515|MAIN||  
 OBX|2|TX|RES^RESULT:|1|POSITIVE FOR PNEUMOCYSTIS JIROVECI (CARINII)  
 BY DFA||P||P||200910071515|MAIN||  
 OBX|2|TX|RES^RESULT:|1|GIARDIA LAMBLIA (CRITICAL/ALERT  
 VALUE)||C||F||200910071515|MAIN||  
 OBX|2|TX|RES^RESULT:|1|LEVELS PASSED||P||P||200910071515|MAIN||

More information about HL7 can be found at <http://www.hl7.org>

SOFTWARE ENGINEERING ARTIFACTS

# **Software Requirements Specification**

**for**

## **Physician Portal**

## Revision History

Name	Date	Reason For Changes	Version
Mike Crowell	5/22/2013	Initial Document	1.0
Mike Crowell	6/2/2013	Requirement Revision	2.0
Mike Crowell	6/28/2013	Requirement Revision	3.0

## Introduction

### Purpose

This document will outline an application that will be used within a physician group office to allow physicians to access their patient results received from the various institutions where they provide services.

### Project Scope

This project will provide a web portal solution for the physicians group office that will provide them with access to patient results received from authorized institutions. The web portal will communicate with the different systems by means of a batch HL7 interface. The HL7 interface will automatically update the database. The application will allow for login restrictions so physicians can only see the results of those patients where they are listed as the attending provider. Statistical analysis reporting will be provided.

## Overall Description

### Product Perspective

The physician group needs a centralized means to access patient results that currently exist in a handful of different software applications from different software vendors.

### Product Features

This application will consist of an HL7 interface. HL7 (Health Level 7) is a standardized format used in the integration of data between disparate systems within a healthcare environment. The interface will import data from a file in HL7 format and perform an add or update to the database with the data from that file. Providing an HL7 interface will allow the application to communicate with many of healthcare software products from different vendors. The HL7 interface will import ORU (observational results) transactions. The interface will be automated to run on a timed interval that can be set by the user. The application will feature security that will only display those patients that are assigned to the physician that is currently logged in. Physicians will be able to view order status and finalized results for each test that has been ordered for each of their patients. General patient demographic information can also be viewed. Statistical analysis reports will be available to all users. A complete list of those reports is still to be determined but will include: Most common tests ordered by physician group, most common tests ordered by each physician, list of sites where each physician practices.

### **User Classes and Characteristics**

The primary target audience of this application is the physicians within the physicians group. Other office staff at the physicians group office will also be users of this application. Because of this, the users will have a wide range of web and computer skills.

### **Operating Environment**

This application will be a web portal designed to allow the physicians to access the application from remote locations on any platform. It will be used in Internet Explorer, Firefox, Chrome, Safari, and potentially Opera web browsers.

### **Design and Implementation Constraints**

Cross browser, support may cause issues with the development of this application since it will be a web application. This could have an effect on the style and potentially some of the functionality of the application.

### **User Documentation**

A manual is included outlining the functionality of the application.

## **System Features**

- Import external file and validate that it is in HL7 format.
- Parse HL7 file and add or update patient information to database.
- Automated HL7 import process with interval defined by user.
- Create a log file of the status of each import, and an error file with any issues that occur during the import process.

Add, edit, physicians in database.  
Add, edit, locations that physicians group services.  
Add, edit, list of test codes.  
Security to restrict data that is displayed based on user that is logged in.  
Allow physicians to display a list of their patients.  
Allow physicians to view a list of tests that have been ordered for each patient and the status of each order.  
Allow physicians to view the detailed results for each order that has been resulted.  
Statistical analysis reporting.

## **External Interface Requirements**

### **User Interfaces**

This will be a web application with a login splash screen to require user credentials before accessing the rest of the application.

### **Software Interfaces**

This is a .NET web application that connects to an MSSQL 2008 database, which serves as the storage place for all of the patient data and physician group tables.

### **Communications Interfaces**

This application will use a web browser for the user interface. Users will communicate with the web server using HTTP.

## **Other Nonfunctional Requirements**

### **Security Requirements**

Each account will be password protected. Along with that administrators will have the ability to make accounts for users, along with activate and deactivate those accounts as needed.

### **Software Quality Attributes**

This application will display a simple interface that will aid the most basic of users, while supplying the administrators with the control they need.

## SOFTWARE ENGINEERING ARTIFACTS

Risk Management				
Step 1: Risk Identification	Step 2: Risk Assessment		Step 3: Risk Management	
List of Possible Risks	Likelihood H/M/L	Impact H/M/L	What are we already doing about it? (mitigating factors)	What more can we do about it?
Time constraints lead to rushed work which causes bugs and functionality problems	M	H	Plan in advance and schedule around conflicts	Continue to look ahead to what else needs to be done so that time can be allocated to the most important aspects of the project and avoid getting off schedule
Insufficient understanding of security leads to a security problem	L	H	Research how to implement web security	Continue researching security as it pertains to web applications and database
Misunderstanding of the requirements leads to incorrect program functionality, or missing functionality	M	M	Continuing to review the requirements specifications as work on the project progresses	Ask for clarification when there is confusion about the requirements and make sure that requirements document is continuously updated
Changes in requirements lead to work having to be redone	L	M	Make sure all requirements are understood before significant work is started	Continue to regularly communicate with clients about requirements
Insufficient technical knowledge leads to wasted time	H	M	Researched technologies that are new and unfamiliar	Continue to research, make use of all resources to obtain knowledge
Scope of project is too great and project can't be completed	L	H	Continually review the scope and assess the work to be done against the time remaining	Focus on the main parts of the application and those that meet the client's primary needs first.

**SOFTWARE ENGINEERING ARTIFACTS**

# **Software Architecture Document**

**for**

## **Physician Portal**



Physician Portal

Software Architecture Document

Mike Crowell

Version 3.0

## Revision History

<b>Date</b>	<b>Version</b>	<b>Description</b>	<b>Author</b>
6/11/2013	1.0		Mike Crowell
6/25/2013	2.0		Mike Crowell
7/12/2013	3.0		Mike Crowell

## Software Architecture Document

**1. Introduction**

This Software Architecture Document describes the architecture of the Physician Portal application using different architectural views to illustrate different aspects of the system and to explain the significant architectural decisions. The Software Architecture Document includes a high-level description of the goals of the architecture, the use cases support by the system and architectural styles and components that have been selected to best achieve the use cases.

**1.1 Purpose**

The Software Architecture Document provides a comprehensive architectural overview of the Physician Portal Application, designed to fill a need within the health care industry for remote access to physician's patient information.

**1.2 Scope**

The scope of this Software Architecture Document is to depict the architecture of the Physician Portal Application.

**1.4 References**

[KRU41]: The "4+1" view model of software architecture, Philippe Kruchten, November 1995,

<http://www3.software.ibm.com/ibmdl/pub/software/rational/web/whitepapers/2003/Pbk4p1.pdf>

[RSA]: IBM Rational Software Architect

<http://www-306.ibm.com/software/awdtools/architect/swarchitect/index.html>

[RUP]: The IBM Rational Unified Process :

<http://www-306.ibm.com/software/awdtools/rup/index.html>

**1.5 Overview**

Section 2: Architectural Representation

Section 3: Physician Portal

Section 4: Use-Case View

Section 5: Logical View

Section 6: Process View

Section 7: Deployment View

Section 8: Implementation View  
Section 9: Data View  
Section 10: Size and Performance  
Section 11: Quality

## **2. Architectural Representation**

This document describes the architecture of Physician Portal Application using standard UML and ERD diagrams, including use case, component, package, and deployment diagrams. Readers of this document need to be familiar with the UML and ERD notation.

The below views will be specified in the document:

### **Logical view**

Audience: Designers.

Area: Functional Requirements: describes the design's object model. Also describes the most important use-case realizations.

Related Artifacts: Design model

### **Process view**

Audience: Integrators.

Area: Non-functional requirements: describes the design's concurrency and synchronization aspects.

Related Artifacts: (no specific artifact).

### **Implementation view**

Audience: Programmers.

Area: Software components: describes the layers and subsystems of the application.

Related Artifacts: Implementation model, components

### **Deployment view**

Audience: Deployment managers.

Area: describes the mapping of the software onto the hardware and shows the system's distributed aspects.

Related Artifacts: Deployment model.

### **Use Case view**

Audience: all the stakeholders of the system, including the end-users.

Area: describes the set of scenarios and/or use cases that represent some significant, central functionality of the system.

Related Artifacts : Use-Case Model, Use-Case documents

### **Data view (optional)**

Audience: Data specialists, Database administrators

Area: Persistence: describes the architecturally significant persistent elements in the data model

Related Artifacts: Data model.



### **3. Architectural Goals and Constraints**

There are some crucial requirements and systems constraints that significantly impact the architecture:

The Physician Portal Application will be deployed into Carroll IIS Server

The Physician Portal Application development framework is .NET 4.0 and using Microsoft SQL Server 2008.

Data persistence will be addressed using relational database.

Query results must be returned within five seconds.

For application access, there will be different permission criteria for admin access and user access.

Authentication: Login using at least a user name and a password

Authorization: according to their profile, online users must be granted or not to perform some specific actions depending whether they are admin, physician, or staff users.

#### 4. Use-Case View

The following use cases are involved in the Physician Portal application.

UC-1: User Login

UC-2: View Physician's Orders

UC-3: View Patient Results

UC-4: HL7 Import

UC-5: View Interface Log

UC-6: Add Site

UC-7: Add Test Code

UC-8: View Report

#### UC-1: User Login

##### Use Case Overview

<b>Description</b>	This is the process of an Administrator, Physician or staff level user logging into their account.
<b>Actor(s)</b>	User, System
<b>Pre-Conditions</b>	User has already been added to the system.
<b>Post-Conditions</b>	<u>Success end condition</u> User will be logged into their account and able to function in the system with their given privileges.
<b>Trigger</b>	User clicks the log in button

##### Main Flow

Main	
1	User enters their user name.
2	User enters their password.
3	System validates user name and password.
4	System displays application home page with either administrator, physician, or staff level user functionality.

## Alternate Flows

Alt 1	User not in the system
1	System does not find the entered user name.
2	User will be prompted to submit a request to be set up with a user account.
3	Use case ends

## UC-2: View Physician Orders

### Use Case Overview

<b>Description</b>	This case focuses on searching the database for orders based on physician id.
<b>Actor(s)</b>	User, System
<b>Pre-Conditions</b>	User has already been added to the system.
<b>Post-Conditions</b>	Success end condition Orders for the logged in Physician will e displayed
<b>Trigger</b>	User logs in with Physician privileges.

### Main Flow

Main	
1	User logs into the system with physician credentials
2	System queries database using the id of the logged in physician to select from the order table.
3	A list of patients with orders by the logged in physician are displayed in the center panel of the web page.
4	If no match is found in the database the System will display a message indicating that there are no active orders for that physician.
5	User selects on a single patient row to display all orders for that patient.
6	System queries database using the id of the selected patient to select from the order table.
7	A list of orders for the selected patient is displayed below the patient row in the table.

## UC-3: View Patient Results

### Use Case Overview

<b>Description</b>	This case focuses on displaying results for a selected patient order
<b>Actor(s)</b>	User, System
<b>Pre-Conditions</b>	User is logged into their account with physician credentials. A successful query has been performed with at least one patient with an order returned as the result.



<b>Post-Conditions</b>	<u>Success end condition</u> User is able to view a result report for the selected patient order.
<b>Trigger</b>	User clicks on an order to view its corresponding result.

## Main Flow

Main	
1	From the physician's patient list, user selects a patient row to view orders for that patient.
2	System queries database using the id of the selected patient to select from the order table.
3	A list of orders for the selected patient is displayed below the patient row in the table.
4	If the order has results available the order status will display as "Resulted".
5	User selects the row of an order that has a result report available to view.
6	System queries database using the id of the selected order to select from the result table.
7	The result report is displayed below the selected order in the table.

## UC-4: HL7 Import

### Use Case Overview

<b>Description</b>	This case focuses on the system updating the database with data from an HL7 message received through the HL7 Import web service.
<b>Actor(s)</b>	System
<b>Pre-Conditions</b>	An HL7 message has been successfully received through the web service.
<b>Post-Conditions</b>	<u>Success end condition</u> Database will be updated accordingly with either patient demographic, order, or result information.
<b>Trigger</b>	Outside entity calls getHL7Message method of HL7 Import web service.

## Main Flow

Main	
1	System parses first line of string message that was received to determine if it is a valid HL7 message.
2	If the message is not a valid HL7 message then system writes an entry to the interfacelog table to indicate that an invalid message was received. System takes no further action on the message.

3	If the string is a valid HL7 message, system will further parse the first line to determine if the message contains patient demographic, order, or result information.
3	If the message contains patient demographic information, the message will be passed to the ADT processor.
4	If the message contains order information, the message will be passed to the order processor.
5	If the system contains result information, the message will be passed to the result processor.
6	System will parse and process the message based on the respective ADT, order, and result rules and update the patient, order, and result tables accordingly
7	System updates the interfacelog table with the result of the message parsing and table updates.

### UC-5: View Interface Log

#### Use Case Overview

<b>Description</b>	This case focuses on a user viewing the interface log.
<b>Actor(s)</b>	User, System
<b>Pre-Conditions</b>	User is logged into their account. User is logged in with appropriate privileges to view the interface log.
<b>Post-Conditions</b>	<u>Success end condition</u> The interface log is displayed.
<b>Trigger</b>	User selects the option to view the interface log

#### Main Flow

Main	
1	User selects the interface log tab from the menu.
2	System queries database for interface log information.
3	Interface log is displayed in the center panel of the web page.

### UC-6: Add Site

#### Use Case Overview

<b>Description</b>	This case focuses on a user adding a new site
<b>Actor(s)</b>	User, System
<b>Pre-Conditions</b>	User is logged into their account. User is logged in with an administrator account.
<b>Post-Conditions</b>	<u>Success end condition</u> A new site is added.

<b>Trigger</b>	User selects Add New Site option.
----------------	-----------------------------------

## Main Flow

Main	
1	User types new site ID into the Site ID field..
2	User types in a site name into the Site Name field.
3	Optional: User fills in the address, city, state, and zip field.
4	User clicks submit button.
5	System verifies that site id is unique.
6	If the site ID is unique, system adds new site to the database.
7	System displays a message indicating that the new site has been successfully added.

## Alternate Flows

Alt 1	Site ID already exists in database
1	System prompts user that the entered ID already exists and a new unique site ID must be entered.
2	Use case resumes at step 1.

## UC-7: Add Test Code

### Use Case Overview

<b>Description</b>	This case focuses on a user adding a new test code
<b>Actor(s)</b>	User, System
<b>Pre-Conditions</b>	User is logged into their account. User is logged in with an administrator account.
<b>Post-Conditions</b>	<u>Success end condition</u> A new test code is added.
<b>Trigger</b>	User selects Add New Test Code option.

## Main Flow

Main	
1	User types new test code into the Test Code field.
2	User types in a description into the Description field.
4	User clicks submit button.
5	System verifies that the test code is unique.
6	If the test code is unique, system adds new test code to the database.
7	System displays a message indicating that the new test code has been successfully added.

**Alternate Flows**

Alt 1	Test Code already exists in database
1	System prompts user that the entered Test Code already exists and a new unique test code must be entered.
2	Use case resumes at step 1.

**UC-7: View Report****Use Case Overview**

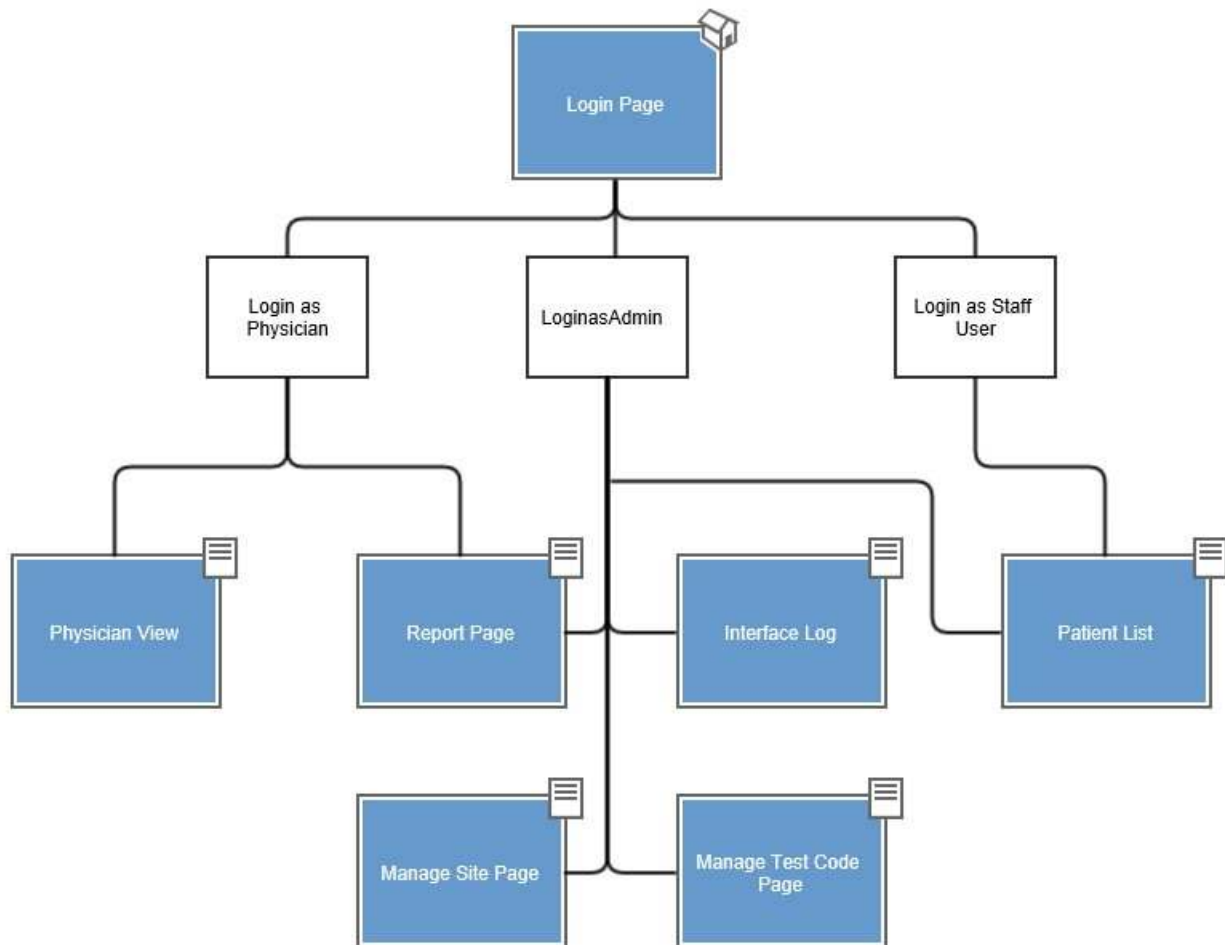
<b>Description</b>	This case focuses on a user viewing a report.
<b>Actor(s)</b>	User, System
<b>Pre-Conditions</b>	User is logged into their account. User is logged in with an administrator or physician account.
<b>Post-Conditions</b>	<u>Success end condition</u> The report is displayed.
<b>Trigger</b>	User selects the option to view a report.

**Main Flow**

Main	
1	User selects one of the available reports. The following reports are available: Physician Activity. Test Code Usage.
2	System queries database for report information.
3	System builds report.
4	Report is displayed in the center panel of the web page.
5	Optional: User may choose to export the report by clicking the export button. System then transfers file to user's local machine using HTTP.

## 5. Logical View

### 5.1 Overview



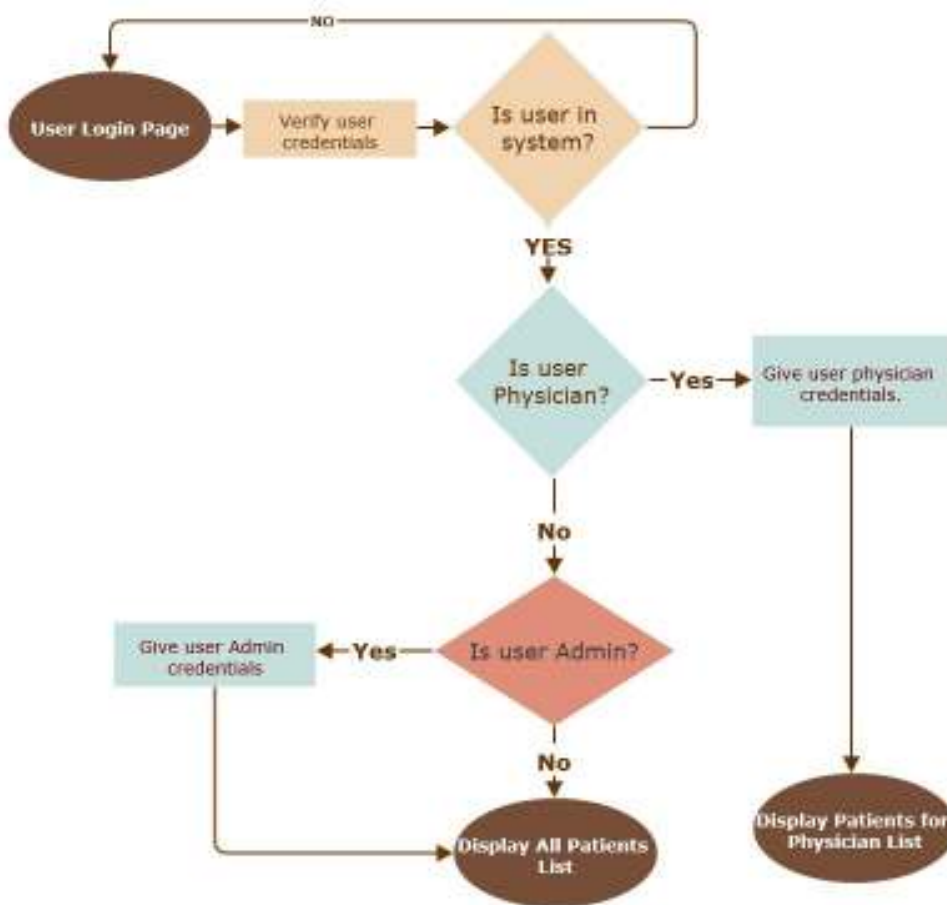
### 5.2 Architecturally Significant Design Considerations

.DataAccess.cs	This class provides a single repository for all of the business logic. It provides the LINQ queries that access the data layer. The DataAccess class is referenced in the view classes, providing a layer of separation between view and data.
HL7ImportWebService.asmx	Web service that exposes a method which allows the application to receive messages from external systems.
IParser.cs	Interface that provides the contract for parsing of the messages being imported into the application. This layer of abstraction provides the application to import messages of different formats.
HL7Parserv2.cs	This class implements the IParser and provides the specific logic needed for the Physician Portal application to read HL7 messages and allow it to integrate in the health care world.

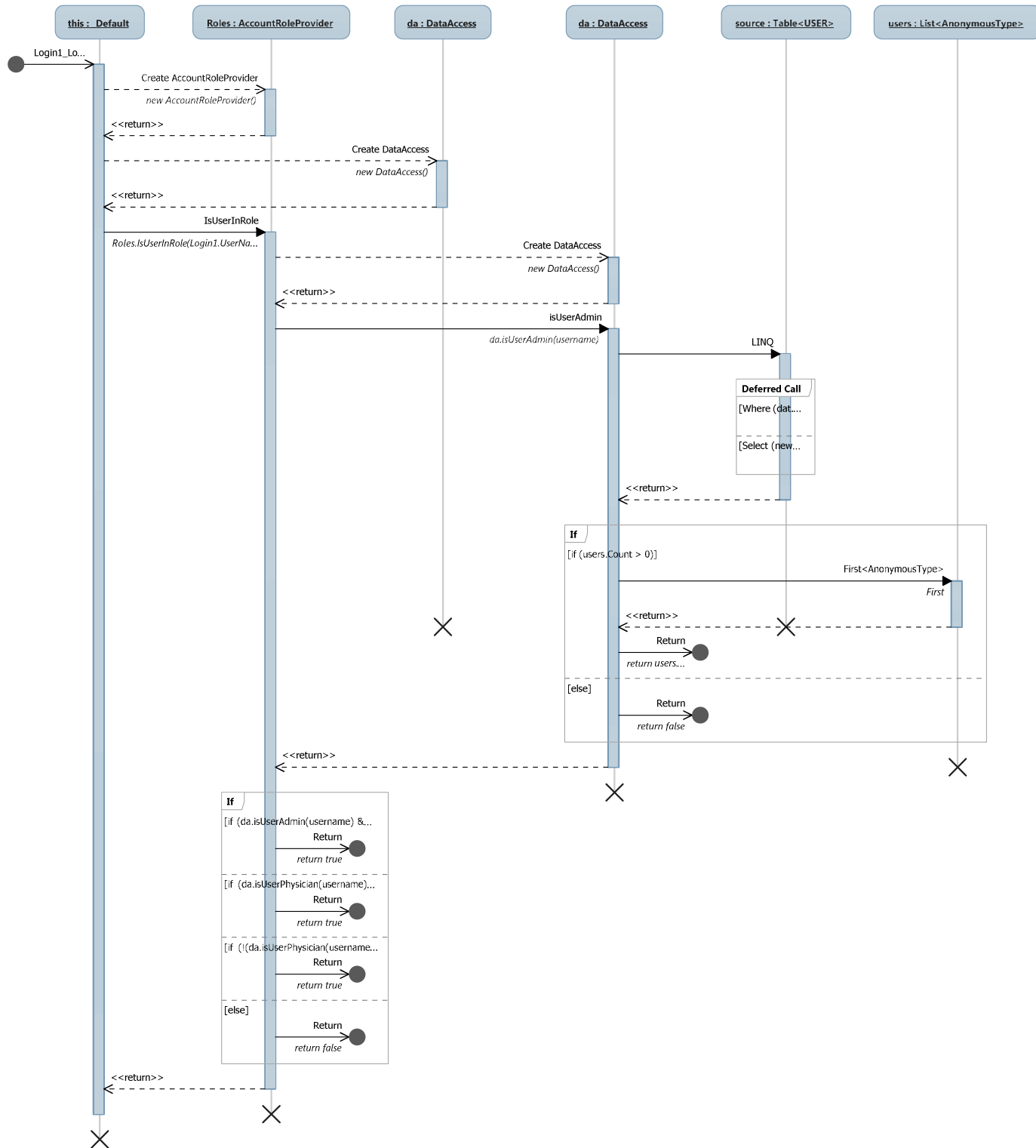
AccountMembershipProvider.cs and AccountRoleProvider.cs	These two classes are overridden to provide custom code that will allow the .NET security features to make use of the external Physician Portal SQL database so the security rules will be driven by the design in that database.
---	---

## 6. Process View

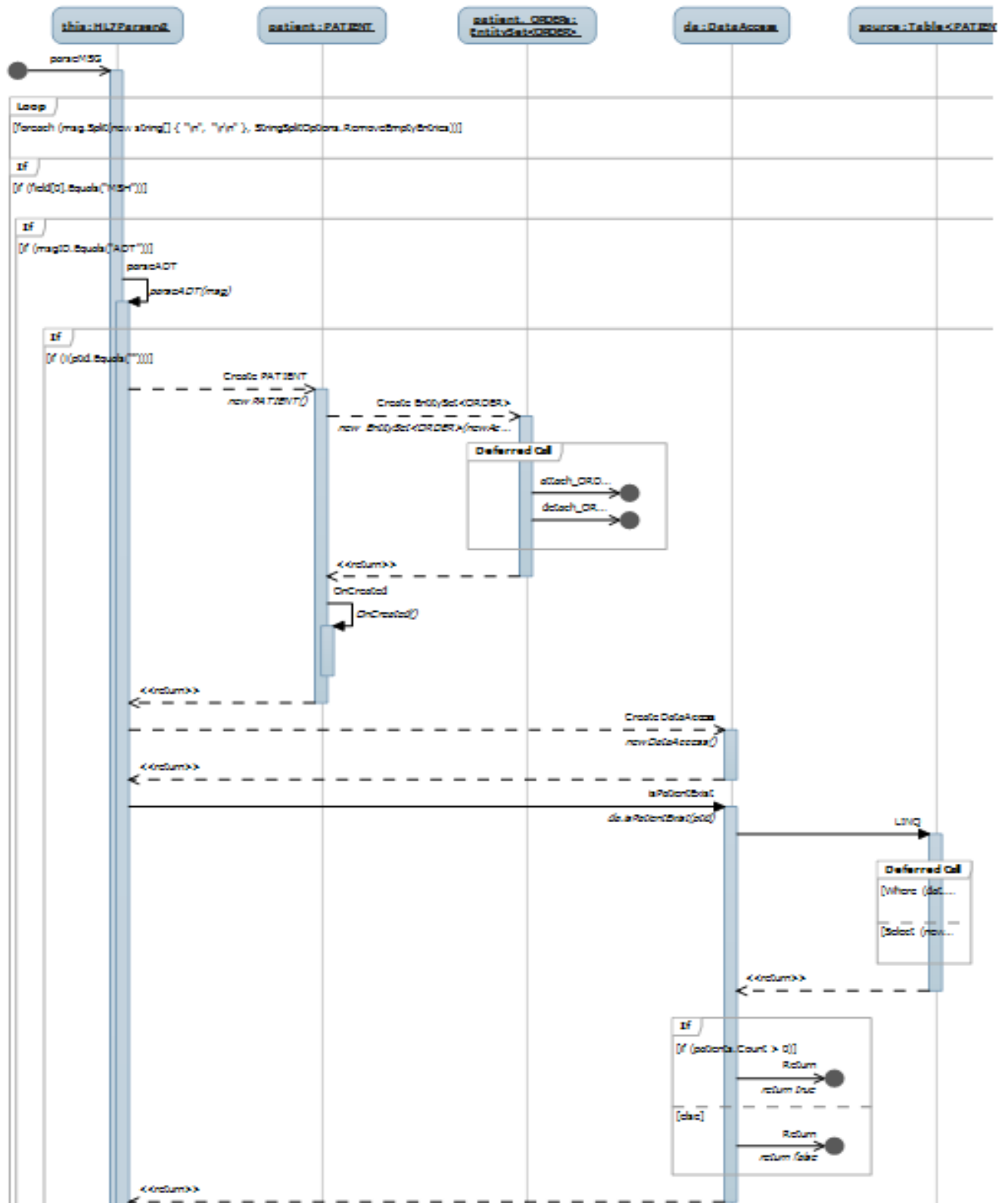
This section consists of three process diagrams. The first is a flow chart representation of the user login process. The second is a sequence diagram representation of the user login process. The third is a sequence diagram representation of the HL7 parsing process.



Login Flow Chart



Login Sequence Diagram

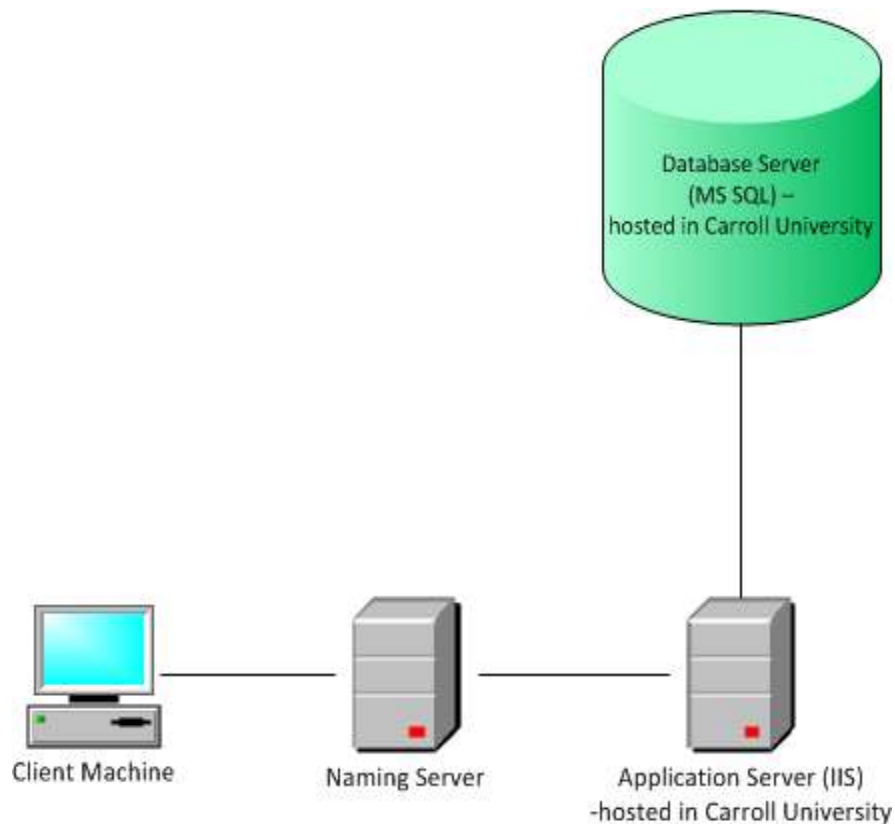


HL7 Parser Sequence Diagram



## 7. Deployment View

Both the Physician Portal application website and the Physician Portal database will be hosted in Carroll University servers. Clients will have access to website using any browser in their personal computer.



## 8. Implementation View

### 8.1 Overview

Although MVC isn't used in the Physician Portal application, it was designed to provide the same separation between the view, business logic, and data access layers. This design will allow for seamless changes in one layer that will not impact the code in the other layers.

### 8.2 Layers

**View Layer:**

This layer provides interfaces with users through different .aspx pages. Through the interface, users input are processed by back-end C# codes behind. To communicate with the database, this layer use different model classes written in C#.

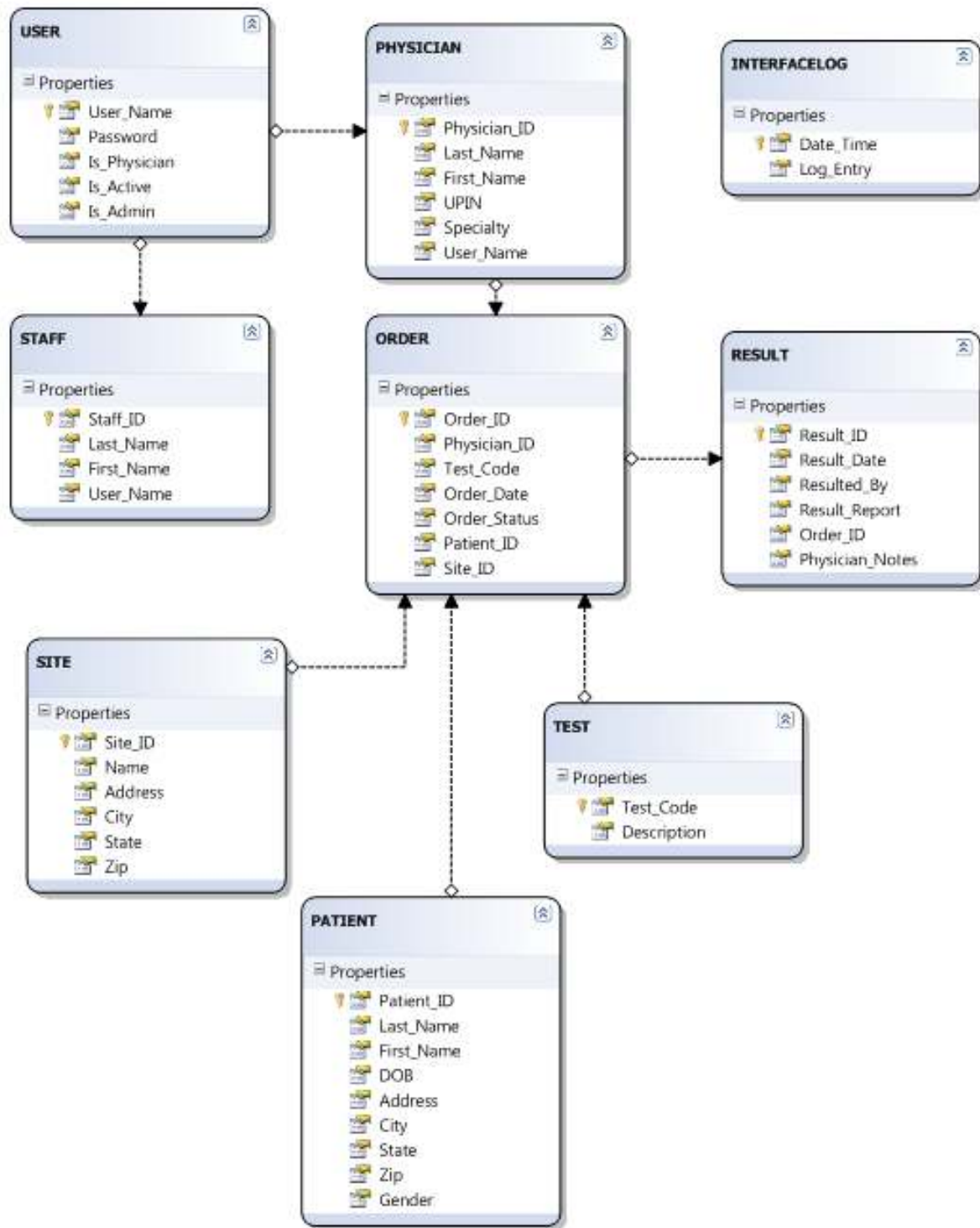
**Logic Layer:**

This layer connects view layer with data source layer. More specifically, it processes, user inputs, get access to data source, and manipulate the data based on user input, and output results to the View layer. Linq was used to create a collection classes, each class represents a table in the Microsoft SQL database. The querying brain is DataAccess.cs. Back-end codes in View Layer use this class to query information from the Linq Tables.

**Data Access Layer:**

This layer is represented by Linqtoalldata.dbml. The physical connection to the SQL server database is managed by LINQ.

## 9. Data View



**10. Size and Performance**

During the first deployment, it will be available to admin users for testing and uploading data. After that, it will be accessible to a wider audience for further integration testing. Page response times will be less than 10 seconds and ideally should be returned within a second. The system will be backed up on in timely manner.

**11. Quality**

Only registered physician, admin, and staff users can access the system. The application will be available 24/7. Users will be notified for scheduled downtime for backup and maintenance. Support will be contacted if admin requires additional information about the application. . Any changes done within the architecture will have less/none effect on other layers of the architecture thus maintaining high portability of the system.

SOFTWARE ENGINEERING ARTIFACTS

# **Software Project Management Plan**

**for**

## **Physician Portal**

**Change History**

<b>Digital Asset Management Application</b>		
SPMP Version 1.0	Release date: 6/1/13	Baseline Version
SPMP Version 2.0	Release date: 6/20/13	
SPMP Version 3.0	Release date: 7/10/13	
SPMP Version 4.0	Release date: 8/15/13	Final Version

## **Preface**

The purpose of this document is to specify the project plan to develop the Physician Portal Application. This document outlines a brief plan about how the project is to be shaped and also includes the milestones and deliverables. The SPMP document will serve as a guide for the project team, developing the product as part of the project. Updates of this document will serve to record the progress of the project.

## **List of Figures**

Figure.1 Schedule

Figure 2. Gantt Chart



## **1.0. Overview**

### ***Project Summary***

#### **Scope, Purpose and Objectives**

Physicians groups consist of a group of physicians that may provide services through a number of different clinics. They may also provide services within a hospital setting. When they order tests or treatments for their patients they will be performed by various ancillary departments, such as laboratory, radiology, or physical therapy departments. Each ancillary department will transmit the patient results back to the hospital or clinic that they are associated with. For a physician group that services multiple organizations this can create a complex environment of disparate systems.

In this type of environment the physicians need to have access to each of the different systems in order to view their patient results. Where the patient results are delivered and stored will depend on where the service was performed. Managing this type of environment creates a level of confusion and complexity for the physicians and their staff that can lead to delays and affect how they are able to provide patient care.

The proposed solution is to provide a single portal for the physicians that will store all of their results from any of the entities that provide their services. This will improve efficiency in the office and it will also help to avoid mistakes caused by the current need to look for information in multiple systems. Having all of the patient data in a single database will also allow the physicians group to perform statistical analysis against the data, such as providing the group a

list of the most common tests that the physicians order. Such analysis again will help to improve the efficiency of the office.

The physician portal will receive the data from the disparate systems via HL7 integration methodologies, which is the common method of integration in the healthcare industry. Providing HL7 import and parsing capabilities for the physician portal will be a key component of this project. The portal itself will be developed as a web application to remove system and platform dependencies so the physicians can access their patient information from remote locations.

### **Assumptions and Constraints**

The author of this document is expected to complete the project within one semester. This project will use resources in the form of time and effort that will be spent developing the project deliverables.

### **Project Deliverables**

The list of project deliverables is:

Project Management Plan

Software Requirements Specification

Software Architecture document

Testing plans

Working .NET application with an MSSQL 2008 database

User Manual

Final Report

Refer to section 1.1.4 for the expected delivery dates of the project deliverables.

## Schedule and Budget Summary

**Budget Summary:** “No budget required”.

A tentative schedule is as shown below in table 1.

**Figure 1: Schedule**

Item	Due date
Software Requirements Specifications	May 22, 2013
Software Architecture Design Document	June 1, 2011
Software Test Plan	June 10, 2013
Software Project Management Plan (this document)	June 15, 2013
Database	July 1, 2013
Software Testing Description	July 30, 2013
System Integration	August 15, 2013
Final Paper	August 22, 2013
Application presentation	August 22, 2013

***Evolution of the Plan***

The preliminary drafts of the SPMP will be developed based on the initial requirements gathered and will be updated as the project progresses and the requirements needs are reassessed.

## References

Braude, Eric J., Software Engineering: An Object Oriented Perspective. Wiley, 2001.

Case Study: SPMP for Encounter video game ref: <http://www.wiley.com/college/braude>.

IEEE Document Standards published in “IEEE Standards Collection”. 2001 edition.

Major field test administered by the ETS: <http://www.ets.org/hea/mft/index.html>

Martin, Dr. Dennis S., Documentation Standards. Jacksonville State University, 2003:

<http://mcis.jsu.edu/faculty/dmartin/cs521.html>

Pressman, Roger S., Software Engineering: A Practitioner’s Approach. McGraw-Hill, 2001.

Health Level 7 International, HL7 Standards.

<http://www.hl7.org/implement/standards/index.cfm?ref=nav>

## Glossary

Term	Description
AUP	Agile Unified Process
CIS	Computer Information Systems program within the MCIS Dept
CS	Computer Science program at Carroll University
CSS	Cascading Style Sheets
Extensible	Software that is designed for future growth
HL7	Health Level 7
HTML	Hyper Text Markup Language
RDBMS	Relational Database Management System.
SCMP	Software Configuration Management Plan
SDD	Software Design Document
SQAP	Software Quality Assurance Plan
SRS	Software Requirement Specification.
Stakeholder	A person, group or organization with a stake in the outcome of an application that is being developed.
TBD	To be decided

## **Project Organization**

### ***External Interfaces***

The external interfaces for the project are the members of the health care community and the designers and resources for the HL7 specification.

### ***Internal Interfaces***

Internal interfaces used for this project are:

Google Drive, to share project artifacts and documents.

Email, to update professor Konemann as issues arise with the project.

Monthly status update meetings with professor Konemann.

### ***Roles and Responsibilities***

Term Member	Responsibility
Mike Crowell	Software development , testing, project management plan, system architecture, risk analysis, requirement specification.

## Managerial Process Plans

### Work Plan

#### Work Activities

Refer to section 5.1.2 Gantt chart.

#### Schedule Allocation

**Figure 2: Gantt chart**

#### Physician Portal Application Project Schedule

Status date: August 22, 2013

Due Date: August 22, 2013

Activity	% Complete	Status	Date Ending						
			5/22	6/1	6/22	7/5	7/19	8/2	8/20
User Analysis	0	S	SSSSS						
Full Specification	0	S	SSSSSSSS						
Architectural Design	100	S	SSSSSS						
Detailed Design	100	S			SSSSSSSS				
Database	100	S			SSSSSSSSSSSSSS				
Physician Module	80	S			SSSSSSSSSSSSSSSSSSSSSSSS				
Order import	100	S			SSSSSSSSSSSSSSSS				
Result reporting	100				SSSSSSS				
Test Design	100	S			SSSSSSS				
Security	90				SSSSSS				
Web Services	100	S				SSSSSSS			
Test Design	100	S				SSSSSSS			
Admin Module	80	S				SSSSSSS			
Report Features	70	S					SSSSSSS		
Test Design	90	S					SSSSSSS		
Implementation	100	S						SSSSSSSSS	
Module Testing	80	S						SSSSSSSSSS	
Integration Testing	80	S						SSSSSSS	

#### Project Status Symbols

- S Schedule
- A Satisfactory
- C Caution
- F Critical

#### Planning / Progress Symbols

- B Work before Schedule Time
- S Scheduled Activity Time



**A** Work after Scheduled Time

**X** Scheduled But Not Worked

### **Resource Allocation**

This project will use resources in the form of time and effort that I shall spend developing the project deliverables as mentioned in section 1.1.3.

### **Budget Allocation**

None.

## ***Control Plan***

### **Requirements Control Plan**

When changes are to be made in the requirements after the SRS has been released, the changes shall be fully documented and implemented within the constraints of the project mentioned in section 1.1.2, and resources in terms of knowledge and skill of the developer required. Once the changes have been made to the SRS document, an updated version of the SRS shall be released. However, no changes shall be made to the requirements once the SDD is completed.

### **Schedule Control Plan**

If the work scheduled in section 1.1.4 is gets behind, the development team is ready to spend extra time on the project in between and after the schedules.

### **Budget Control Plan**

None.

### **Quality Control Plan**

Please refer to Software Quality Assurance Plan regarding Quality Control.

### **Reporting Plan**

Each updated revision of the SPMP will be made available for review.

**Metrics Collection Plan**

None.

***Risk Management Plan***

See Risk Management document.

***Closeout Plan***

All the details about the final project will be delivered to professor Konemann as part of the presentation at the end of the semester, on the due date as outlined in section 1.1.4. A user manual will be provided for use and reference after product deployment. All user requests for this application were met. Further development is still needed to complete the reporting and physician entry portions of the application. Parts of the application were designed to be extensible and can be easily modified to meet future needs.

## **Technical Process**

### ***Process Model***

The Physician Portal Application will be implemented using the Agile Unified Process. The AUP involves the following disciplines: Model, Implementation, Test, Deployment, Configuration Management, Project Management, Environment. The use of AUP in this project will provide a simple iterative process that can easily adapt to changes in user requirements and will effectively make use of all available resources.

An explanation of the Unified Process can be found here:

<http://www.methodsandtools.com/archive/archive.php?id=32>

### ***Methods, Tools, and Techniques***

The Physician Portal Application will be developed as a .NET 4.0 web application in Visual Studio 2010 with C# as the primary programming language. HTML, CSS, and JavaScript will also be used. It will use MSSQL 2008 as its database.

### ***Infrastructure Plan***

The hardware resources of the Microsoft web server and database server are unknown.

### ***Product Acceptance Plan***

## **Support Process Plans**

### ***Configuration Management Plan***

All the project deliverables are to be considered as configuration items. The project will be developed in Dev/Test/Prod environment, with each code iteration being tested and promoted up the ladder.

### ***Validation & Verification Plan***

A Verification and Validation Plan as a part of the Software Quality Assurance and Verification & Validation Plan will be developed following recommended standards. See section 1.1.4 for its delivery date.

### ***Documentation Plan***

All documents for this project will be by Mike Crowell.

### ***Quality Assurance Plan***

Software quality will be assured through the following means:

Thorough unit testing within Visual Studio

Review by outside resources when available.

### ***Review and Audits***

Review and Audits would be addressed as a part of the Software Quality Assurance and Verification & Validation Plan that would be developed following recommended standards. See section 1.1.4 for its delivery date.

***Problem Resolution Plan***

A communication system is in place where professor Konemann can be reached as problems arise. The ITS department at Carroll University is also available for technical issues.

***Subcontractor Management Plan***

None.

***Process Improvement Plan***

Project management documentation will be maintained throughout the project. A journal of programmer's tasks will also be maintained. Both documents will be examined to identify areas where work and available resources can be better managed and aligned.

**TEST PLANS****LOGIN**

<b>Test Item</b>	<b>Expected Result</b>	<b>Actual Result</b>
Test 1: Leave User Name field blank	Prompt should display stating User name is required	Prompt was displayed
Test 2: Leave Password field blank	Prompt should display stating Password is required	Prompt was displayed
Test 3: Enter invalid user name	Unsuccessful login message should be displayed	Message was displayed
Test 4: Enter valid user name and invalid password	Unsuccessful login message should be displayed	Message was displayed
Test 5: Enter valid login name and password for regular user	Welcome message should display. User should have access to search page and no other pages.	Welcome message was displayed. Only the search page was available to the user.
Test 6: Logout	Welcome message should no longer be displayed. User should be directed to the login page.	User was logged out and directed to the login page
Test 7: Enter valid user name and password for an admin user	Welcome message should display. User should have tabs available with access to all admin pages.	Welcome message displayed. Tabs for all admin pages were available.
Test 8: Logout	Welcome message should no longer be displayed. User should be directed to the login page.	User was logged out and directed to the login page
Test 9: Enter valid user name and password for a physician	Welcome message should display. User should have tabs available with access to all physician pages. User should see only those patients assigned to that physician	Welcome message displayed. Tabs for all physician pages were available. Patients displayed were only those assigned to that physician

**HL7 IMPORT**

<b>Test Item</b>	<b>Expected Result</b>	<b>Actual Result</b>
Test 1: Send a message to the web service that is not in valid HL7 format	Log should be updated with message indicating invalid message was received	Invalid message received was written to interface log
Test 2: Send a valid HL7 ADT message to the web service	New patient should be added to the database	Patient was added
Test 3: Send a valid HL7 order message to the web service	New order should be added to the database	New order was added
Test 4: Send a valid HL7 result message to the web service	New result should be added to the database	New result was added
Test 5: Send an ADT message for a patient that already exists in the database	Existing patient record should be updated	Patient record was updated
Test 6: Send an order message for an order that already exists in the database	Existing order record should be updated	Order record was updated
Test 7: Send a result record for a result that already exists in the database	Result record should be updated	Result record was updated
Test 8: Send an order message to the web service. Log into physician account for physician on that order.	Order should appear on that physician's list and the status should say pending	Order was displayed on the list for the correct physician. Status said pending.
Test 9: Send a result message in the web service for the order in the previous step	Order status on the physician list should say Resulted	Order status said Resulted
Test 10: Click on the order row in the physician list for the order with the result that was just sent	Result report should display below the order row in the table	Result report was displayed

**REPORTS**

<b>Test Item</b>	<b>Expected Result</b>	<b>Actual Result</b>
Test 1: Select Test Code Usage Report	Report should display usage count for all test codes	Report displayed correctly
Test 5: Import a new order, go back to reports page and select Test Code Usage Report	Count should increase for test code in the new order	Test code count incremented correctly
Select download to Excel	Report data should be saved to Excel file	Excel file was created successfully

**TABLE MAINTENANCE**

<b>Test Item</b>	<b>Expected Result</b>	<b>Actual Result</b>
Test 1: Go to add site section and add a new site.	New site should be added to the database	Site was successfully added
Test 2: Go to update site section, select a site from the drop down menu.	Fields should populate with information for the selected site.	All fields were populated with the correct information
Test 3: Modify the information for the selected site.	Updated information should be stored in the database	Database was updated with the new information
Test 4: Go to add test code section and add a new test code.	New test code should be added to the database	Test code was successfully added
Test 5: Go to update test code section, select a test code from the drop down menu.	Fields should populate with information for the selected test code.	All fields were populated with the correct information
Test 6: Modify the information for the selected test code.	Updated information should be stored in the database	Database was updated with the new information



## CODE SUMMARY

### HL7ImportWebService.asmx

The HL7ImportWebService.asmx web service exposes the getHL7Message method to external systems so they can send messages to the Physician Portal application which can then be imported into the Physician Portal database. HL7ImportWebService.asmx is a web service that uses the Simple Object Access Protocol.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Services;

namespace PhysicianPortal2
{
    /// <summary>
    /// Web service method to receive HL7 formatted String message
    /// </summary>
    [WebService(Namespace = "http://tempuri.org/")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    [System.ComponentModel.ToolboxItem(false)]
    public class HL7ImportWebService : System.Web.Services.WebService
    {
        [WebMethod]
        public void GetHL7Message(String msg)
        {
            IParser h17;
            h17 = new HL7Parsev2();
            h17.parseMSG(msg.ToString());
        }
    }
}
```

### Web service WSDL XML file

In order for external systems to be able to successfully send messages to the Physician Portal application, it needs to understand how the Physician Portal application expects to receive messages. This information is shared by providing an XML WSDL file that tells other systems what the Physician Portal application expects to receive.

```

<?xml version="1.0" encoding="UTF-8"?>
- <wsdl:definitions xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://tempuri.org/" xmlns:tns="http://tempuri.org/" xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" xmlns:soapenc="http://schemas
- <wsdl:types>
  - <s:schema targetNamespace="http://tempuri.org/" elementFormDefault="qualified">
    - <s:element name="GetHL7Message">
      - <s:complexType>
        - <s:sequence>
          <s:element name="msg" type="s:string" maxOccurs="1" minOccurs="0"/>
        </s:sequence>
      </s:complexType>
    </s:element>
    - <s:element name="GetHL7MessageResponse">
      <s:complexType/>
    </s:element>
  </s:schema>
</wsdl:types>
- <wsdl:message name="GetHL7MessageSoapIn">
  <wsdl:part name="parameters" element="tns:GetHL7Message"/>
</wsdl:message>
- <wsdl:message name="GetHL7MessageSoapOut">
  <wsdl:part name="parameters" element="tns:GetHL7MessageResponse"/>
</wsdl:message>
- <wsdl:portType name="HL7ImportWebServiceSoap">
  - <wsdl:operation name="GetHL7Message">
    <wsdl:input message="tns:GetHL7MessageSoapIn"/>
    <wsdl:output message="tns:GetHL7MessageSoapOut"/>
  </wsdl:operation>
</wsdl:portType>
- <wsdl:binding name="HL7ImportWebServiceSoap" type="tns:HL7ImportWebServiceSoap">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http"/>
  + <wsdl:operation name="GetHL7Message">
  </wsdl:operation>
</wsdl:binding>
- <wsdl:binding name="HL7ImportWebServiceSoap12" type="tns:HL7ImportWebServiceSoap">
  <soap12:binding transport="http://schemas.xmlsoap.org/soap/http"/>
  - <wsdl:operation name="GetHL7Message">
    <soap12:operation style="document" soapAction="http://tempuri.org/GetHL7Message"/>
    - <wsdl:input>
      <soap12:body use="literal"/>
    </wsdl:input>
    - <wsdl:output>
      <soap12:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
- <wsdl:service name="HL7ImportWebService">
  - <wsdl:port name="HL7ImportWebServiceSoap" binding="tns:HL7ImportWebServiceSoap">
    <soap:address location="http://cscserver2.carrollu.edu/csc450mcrowell/HL7ImportWebService.asmx"/>
  </wsdl:port>
  - <wsdl:port name="HL7ImportWebServiceSoap12" binding="tns:HL7ImportWebServiceSoap12">
    <soap12:address location="http://cscserver2.carrollu.edu/csc450mcrowell/HL7ImportWebService.asmx"/>
  </wsdl:port>
</wsdl:service>
</wsdl:definitions>

```

### IParser.cs and HL7Parserv2.cs

The IParser.cs interface provides an abstraction to the message import process. This the Physician Portal application extensible and will allow different message types to be added than can be easily imported without the need to modify other parts of the application. The HL7Parserv2.cs class implements the IParser interface and provides the details needed to parse and an understand an HL7 formatted message.

```

interface IParser
{

```

```

        void parseMSG(string msg);
    }

namespace PhysicianPortal2
{
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Web;
    using System.Collections;
    using System.IO;

    public class HL7Parsev2 : IParser
    {
        // Public method to receive a string message that is expected to be in HL7
        // format.
        // If one of the HL7 IDs are not identified in the message then it is logged as
        // an invalid HL7 message. Otherwise the message is passed to the appropriate
        // method.
        public void parseMSG(String msg)
        {
            String msgID = "";
            foreach (string segment in msg.Split(new string[] { "\n", "\r\n" },
                StringSplitOptions.RemoveEmptyEntries))
            {
                //Get field
                String[] field = segment.ToString().Split('|');
                if (field[0].Equals("MSH"))
                {
                    String[] component = field[8].ToString().Split('^');
                    msgID = component[0].ToString();
                    if (msgID.Equals("ADT"))
                    {
                        parseADT(msg);
                    }
                    else if (msgID.Equals("ORM"))
                    {
                        parseOrder(msg);
                    }
                    else if (msgID.Equals("ORU"))
                    {
                        parseResult(msg);
                    }
                    else
                    {
                        //invalid HL7 message
                        INTERFACELOG log = new INTERFACELOG();
                        log.Date_Time = DateTime.Now;
                        log.Log_Entry = "Invalid HL7 message received. \n" + msg;
                        DataAccess da = new DataAccess();
                        da.writeToInterfaceLog(log);
                    }
                }
            }
        }
    }
}

```

```

    }

    // This method handles messages with Patient demographic information.
    // If the patients exists in the database then the existing record is updated.
    // Otherwise a new patient record is added.
    private void parseADT(String msg)
    {
        String ptid = "";
        String lastName = "";
        String firstName = "";
        String dob = "";
        String gender = "";
        String address = "";
        String city = "";
        String state = "";
        String zip = "";

        foreach (string segment in msg.Split(new string[] { "\n", "\r\n" },
StringSplitOptions.RemoveEmptyEntries))
        {
            String[] field = segment.ToString().Split('|');
            if (field[0].Equals("PID"))
            {
                ptid = field[2].ToString();
                String[] nameComponent = field[5].ToString().Split('^');
                lastName = nameComponent[0].ToString();
                firstName = nameComponent[1].ToString();
                dob = field[7].ToString();
                gender = field[8].ToString();
                String[] addressComponent = field[11].ToString().Split('^');
                address = addressComponent[0].ToString();
                city = addressComponent[2].ToString();
                state = addressComponent[3].ToString();
                zip = addressComponent[4].ToString();
            }
        }
        if (!(ptid.Equals("")))
        {
            PATIENT patient = new PATIENT();
            patient.Patient_ID = ptid;
            patient.Last_Name = lastName;
            patient.First_Name = firstName;
            String year = dob.Substring(0, 4);
            String month = dob.Substring(4, 2);
            String day = dob.Substring(6, 2);
            String birthdate = month + "/" + day + "/" + year;
            patient.DOB = Convert.ToDateTime(birthdate);
            patient.Gender = gender;
            patient.Address = address;
            patient.City = city;
            patient.State = state;
            patient.Zip = zip;
            DataAccess da = new DataAccess();
            if (da.isPatientExist(ptid))
            {
                da.modifyPatient(patient);
                INTERFACELOG log = new INTERFACELOG();
                log.Date_Time = DateTime.Now;
            }
        }
    }

```

```

        log.Log_Entry = "Patient id " + ptid + " modified.";
        da.writeToInterfaceLog(log);
    }
    else
    {
        da.addPatient(patient);
        INTERFACELOG log = new INTERFACELOG();
        log.Date_Time = DateTime.Now;
        log.Log_Entry = "Patient id " + ptid + " added.";
        da.writeToInterfaceLog(log);
    }
}
else
{
    //HL7 error
    INTERFACELOG log = new INTERFACELOG();
    log.Date_Time = DateTime.Now;
    log.Log_Entry = "Null patient ID received.\n" + msg;
    DataAccess da = new DataAccess();
    da.writeToInterfaceLog(log);
}
}

// This method handles messages with Patient order information.
// If the order exists in the database then the existing record is updated.
// Otherwise a new order is added.
private void parseOrder(String msg)
{
    String orderid = "";
    String physicianid = "";
    String testcode = "";
    String orderdate = "";
    String ptid = "";
    String siteid = "";

    foreach (string segment in msg.Split(new string[] { "\n", "\r\n" },
StringSplitOptions.RemoveEmptyEntries))
    {
        String[] field = segment.ToString().Split('|');
        if (field[0].Equals("PID"))
        {
            ptid = field[2].ToString();
        }
        else if (field[0].Equals("PV1"))
        {
            siteid = field[3].ToString();
        }
        else if (field[0].Equals("OBR"))
        {
            orderid = field[2].ToString();
            testcode = field[4].ToString();
            orderdate = field[7].ToString();
            String[] physicianComponent = field[16].ToString().Split('^');
            physicianid = physicianComponent[0].ToString();
        }
    }
    if (!(orderid.Equals("")))
    {

```

```

ORDER order = new ORDER();
order.Order_ID = orderid;
order.Physician_ID = physicianid;
order.Test_Code = testcode;
String year = orderdate.Substring(0, 4);
String month = orderdate.Substring(4, 2);
String day = orderdate.Substring(6, 2);
String newdate = month + "/" + day + "/" + year;
order.Order_Date = Convert.ToDateTime(newdate);
order.Order_Status = "Pending";
order.Patient_ID = ptid;
order.Site_ID = siteid;
DataAccess da = new DataAccess();
if (da.isOrderExist(orderid))
{
    da.modifyOrder(order);
    INTERFACELOG log = new INTERFACELOG();
    log.Date_Time = DateTime.Now;
    log.Log_Entry = "Order id " + orderid + " modified.";
    da.writeToInterfaceLog(log);
}
else
{
    da.addOrder(order);
    INTERFACELOG log = new INTERFACELOG();
    log.Date_Time = DateTime.Now;
    log.Log_Entry = "Order id " + orderid + " added.";
    da.writeToInterfaceLog(log);
}
}
else
{
    //HL7 error
    INTERFACELOG log = new INTERFACELOG();
    log.Date_Time = DateTime.Now;
    log.Log_Entry = "Null order ID received.\n" + msg;
    DataAccess da = new DataAccess();
    da.writeToInterfaceLog(log);
}
}

// This method handles messages with result information.
// If the result exists in the database then the existing record is updated.
// Otherwise a new result is added.
// The method will also update the order status of the order record.
private void parseResult(String msg)
{
    String resultid = "";
    String resultdate = "";
    String resultedby = "";
    String resultreport = "";
    String orderid = "";

    foreach (string segment in msg.Split(new string[] { "\n", "\r\n" },
StringSplitOptions.RemoveEmptyEntries))
    {
        String[] field = segment.ToString().Split('|');
        if (field[0].Equals("OBR"))

```

```

        {
            resultid = field[2].ToString();
            orderid = field[3].ToString();
            resultdate = field[7].ToString();
            resultedy = field[11].ToString();
        }
        else if (field[0].Equals("OBX"))
        {
            resultreport += field[5].ToString() + ", ";
        }
    }
    if (!(resultid.Equals("")))
    {
        DataAccess da = new DataAccess();
        ORDER order = da.getORDERByOrderID(orderid);
        if (order == null)
        {
            RESULT result = new RESULT();
            result.Result_ID = resultid;
            String year = resultdate.Substring(0, 4);
            String month = resultdate.Substring(4, 2);
            String day = resultdate.Substring(6, 2);
            String newdate = month + "/" + day + "/" + year;
            result.Result_Date = Convert.ToDateTime(newdate);
            result.Resulted_By = resultedy;
            result.Result_Report = resultreport;
            result.Order_ID = orderid;
            INTERFACELOG log = new INTERFACELOG();
            if (da.isResultExist(resultid))
            {
                da.modifyResult(result);
                log.Date_Time = DateTime.Now;
                log.Log_Entry = "Result id " + resultid + " modified.";
                da.writeToInterfaceLog(log);
            }
            else
            {
                da.addResult(result);
                log.Date_Time = DateTime.Now;
                log.Log_Entry = "Result id " + resultid + " added.";
                da.writeToInterfaceLog(log);
            }
            order.Order_Status = "Resulted";
            da.modifyOrder(order);
            log.Date_Time = DateTime.Now;
            log.Log_Entry = "Order id " + orderid + " modified.";
            da.writeToInterfaceLog(log);
        }
        else
        {
            INTERFACELOG log = new INTERFACELOG();
            log.Date_Time = DateTime.Now;
            log.Log_Entry = "Result contained an order ID that does not exist.\n
Results can only be applied to an existing order. \n" + msg;
            da.writeToInterfaceLog(log);
        }
    }
}

```

```

else
{
    //HL7 error
    INTERFACELOG log = new INTERFACELOG();
    log.Date_Time = DateTime.Now;
    log.Log_Entry = "Null result ID received.\n" + msg;
    DataAccess da = new DataAccess();
    da.writeToInterfaceLog(log);
}
}
}
}

```

### **AccountMembershipProvider.cs and AccountRoleProvider.cs**

The AccountMembershipProvider and AccountRoleProvider classes, which .NET uses to manage security and users/roles, were overridden to provide custom functionality to allow the Physician Portal application to define how the users and roles should be managed. The custom AccountMembershipProvider and AccountRoleProvider classes will rely and the Physician Portal database to authenticate users and roles.

```

public override string GetPassword(string username, string answer)
{
    DataAccess da = new DataAccess();
    return da.getUserPassword(username);
    return "";
}

public override System.Web.Security.MembershipUser GetUser(string username, bool
userIsOnline)
{
    DataAccess da = new DataAccess();
    if (da.isValidUser(username))
    {
        User newUser = new User(username);
        return newUser;
    }
    else
    {
        MembershipUser newMember = new MembershipUser(
            providerName: "AccountMembershipProvider",
            name: "",
            providerUserKey: null,
            email: "",
            passwordQuestion: "",
            comment: "",
            isApproved: false,
            isLockedOut: false,

```



```
        creationDate: DateTime.UtcNow,
        lastLoginDate: DateTime.UtcNow,
        lastActivityDate: DateTime.UtcNow,
        lastPasswordChangedDate: DateTime.UtcNow,
        lastLockoutDate: DateTime.UtcNow);
    return newMember;
}
}

public override bool ValidateUser(string username, string password)
{
    DataAccess da = new DataAccess();
    if (da.isValidUser(username) &&
da.getUserPassword(username).Equals(password))
    {
        return true;
    }
    else
    {
        return false;
    }
}

// Determine roles based on values from the SQL database.
public override string[] GetRolesForUser(string username)
{
    String[] roles;
    DataAccess da = new DataAccess();
    if (da.isUserAdmin(username))
    {
        roles = new String[] { "Admin" };
    }
    else if (da.isUserPhysician(username))
    {
        roles = new String[] { "Physician" };
    }
    else
    {
        roles = new String[] { "User" };
    }
    return roles;
}

public override bool IsUserInRole(string username, string roleName)
{
    DataAccess da = new DataAccess();
    if (da.isUserAdmin(username) && roleName.Equals("Admin"))
    {
        return true;
    }
    else if (da.isUserPhysician(username) && roleName.Equals("Physician"))
    {
        return true;
    }
}
```

```

        else if (!(da.isUserPhysician(username)) && !(da.isUserAdmin(username)) &&
roleName.Equals("User"))
        {
            return true;
        }
        else
        {
            return false;
        }
    }
}

```

### Physician.aspx.cs

The Physician.aspx.cs code behind class uses the identity of the logged in user to display a custom list of only those patients that are being seen by the physician that is logged into the Physician Portal application. The results are displayed in a three layered nested gridview control that allows the user to drill down from a list of patients to orders for each patient and finally to the result report for each order.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace PhysicianPortal2.PhysicianPages
{
    public partial class Physician : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            if (!IsPostBack)
            {
                DataAccess da = new DataAccess();
                String physID =
da.getPhysicianIDByUserID(Page.User.Identity.Name);
                gvPatient.DataSource = da.getPatientsByPhysicianID(physID);
                gvPatient.DataBind();
            }
        }

        protected void gvPatient_OnRowDataBound(object sender, GridViewRowEventArgs
e)
        {
            if (e.Row.RowType == DataControlRowType.DataRow)
            {
                Label lblPtID = (Label)e.Row.FindControl("lblPtID");
                GridView gvOrders = (GridView)e.Row.FindControl("gvOrders");

                string txtptid = lblPtID.Text;

                DataAccess da = new DataAccess();

                gvOrders.DataSource = da.getPatientOrdersByPatientID(txtptid);
                gvOrders.DataBind();
            }
        }
    }
}

```

```

    }
}

protected void gvOrders_OnRowDataBound(object sender, GridViewRowEventArgs
e)
{
    if (e.Row.RowType == DataControlRowType.DataRow)
    {
        Label lblOrderID = (Label)e.Row.FindControl("lblOrderID");
        GridView gvResult = (GridView)e.Row.FindControl("gvResult");

        string txtorderid = lblOrderID.Text;

        DataAccess da = new DataAccess();

        gvResult.DataSource = da.getResultListByOrderID(txtorderid);
        gvResult.DataBind();
    }
}
}

```

### Patient.cs

A Patient.cs class was created to represent the PATIENT table in the database and to implement the IComparable interface to allow patients to be sorted by first and last name. The User.cs class represents the USER table in the database. It implements the IComparable interface and extends MembershipUser. Extending MembershipUser allows the data from the user table to be used in the .NET management of user authentication and roles.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace PhysicianPortal2
{
    public class Patient : IComparable<Patient>
    {
        public string ptid { get; set; }
        public string lastName { get; set; }
        public string firstName { get; set; }
        public string dob { get; set; }
        public string address { get; set; }
        public string city { get; set; }
        public string state { get; set; }
        public string zip { get; set; }

        #region IComparable<DataFile> Members

```

```

        public int CompareTo(Patient other)
        {
            return String.Compare(this.ptid, other.ptid);
        }

#endregion

public override string ToString()
{
    return "ID: " + ptid + "\n" +
        "Patient Name: " + lastName + ", " + firstName + "\n" +
        "DOB: " + dob + "\n" +
        "Address: " + address + "\n"
        + city + ", " + state + " " + zip + "\n";
}

public override int GetHashCode()
{
    return this.ptid.GetHashCode();
}
}

}

namespace PhysicianPortal2
{
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Web;

    public class User : System.Web.Security.MembershipUser, IComparable<User>
    {
        public string username { get; set; }
        public string password { get; set; }
        public bool isPhysician { get; set; }
        public string userLogin { get; set; }

        public User()
        {
        }

        public User(String name)
            : base(
                providerName: "AccountMembershipProvider",
                name: name,
                providerUserKey: null,
                email: "",
                passwordQuestion: "",
                comment: "",
                isApproved: true,
                isLockedOut: false,
                creationDate: DateTime.UtcNow,
                lastLoginDate: DateTime.UtcNow,
            )
        {
        }
    }
}

```

```

        lastActivityDate: DateTime.UtcNow,
        lastPasswordChangedDate: DateTime.UtcNow,
        lastLockoutDate: DateTime.UtcNow)
    {
        userLogin = name;
    }

    #region IComparable<User> Members

    public int CompareTo(User other)
    {
        return String.Compare(this.username, other.username);
    }

    #endregion
}
}

```

### DataAccess.cs

The DataAccess.cs class provides all of the Linq queries to represent the business logic that deliver the necessary datasets. The view classes will call the methods in the DataAccess class so they are not tied directly to the data layer. This provides a separation between the data and view layers so changes can be made in one layer without having an impact on the other.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace PhysicianPortal2
{
    public class DataAccess
    {
        private LinqToAllDataDataContext linqDat = new LinqToAllDataDataContext();

        public List<USER> getListofAllUsers()
        {
            List<USER> uList = new List<USER>();
            var u = from usr in linqDat.USERs select usr;
            foreach (var v in u)
            {
                USER newU = new USER();
                newU.User_Name = v.User_Name;
                newU.Password = v.Password;
                newU.Is_Physician = v.Is_Physician;
                uList.Add(newU);
            }
            return uList;
        }

        public bool isValidUser(String name)
        {

```

```
var users = (from dat in linqDat.USERS
              where dat.User_Name.Equals(name)
              select new { dat }).ToList();

if (users.Count > 0)
{
    if (users.First().dat.Is_Active)
    {
        return true;
    }
    else
    {
        return false;
    }
}
else
{
    return false;
}
}

public String getUserPassword(String name)
{
    var users = (from dat in linqDat.USERS
                  where dat.User_Name.Equals(name)
                  select new { dat }).ToList();

    if (users.Count > 0)
    {
        return users.First().dat.Password;
    }
    else
    {
        return null;
    }
}

public bool isUserAdmin(String name)
{
    var users = (from dat in linqDat.USERS
                  where dat.User_Name.Equals(name)
                  select new { dat }).ToList();

    if (users.Count > 0)
    {
        return users.First().dat.Is_Admin;
    }
    else
    {
        return false;
    }
}

public bool isUserPhysician(String name)
{
    var users = (from dat in linqDat.USERS
```

```

        where dat.User_Name.Equals(name)
        select new { dat }).ToList();

    if (users.Count > 0)
    {
        return users.First().dat.Is_Physician;
    }
    else
    {
        return false;
    }
}

public void addPatient(Patient patient)
{
    PATIENT newPatient = new PATIENT();
    newPatient.Patient_ID = patient.ptid;
    newPatient.Last_Name = patient.lastName;
    newPatient.First_Name = patient.firstName;
    DateTime dt = Convert.ToDateTime(patient.dob);
    newPatient.DOB = dt;
    newPatient.Address = patient.address;
    newPatient.City = patient.city;
    newPatient.State = patient.state;
    newPatient.Zip = patient.zip;
    linqDat.PATIENTs.InsertOnSubmit(newPatient);
    linqDat.SubmitChanges();
}

public Patient getPatientByID(String id)
{
    var patients = (from dat in linqDat.PATIENTs
        where dat.Patient_ID.Equals(id)
        select new { dat }).ToList();

    if (patients.Count > 0)
    {
        Patient patient = new Patient();
        patient.ptid = patients.First().dat.Patient_ID;
        patient.firstName = patients.First().dat.First_Name;
        patient.lastName = patients.First().dat.Last_Name;
        patient.address = patients.First().dat.Address;
        patient.city = patients.First().dat.City;
        patient.state = patients.First().dat.State;
        patient.zip = patients.First().dat.Zip;
        patient.dob = patients.First().dat.DOB.ToString();
        return patient;
    }
    else
    {
        return null;
    }
}

public List<ORDER> getPatientOrdersByPatientID(String id)
{
    List<ORDER> orderList = new List<ORDER>();

```

```

        var ord = from dat in linqDat.PATIENTs
                   join orders in linqDat.ORDERS on dat.Patient_ID equals
orders.Patient_ID
                   where dat.Patient_ID.Equals(id)
                   select new { orders };

        foreach (var v in ord)
        {
            ORDER o = new ORDER();
            o.Order_ID = v.orders.Order_ID;
            o.Physician_ID = v.orders.Physician_ID;
            o.Test_Code = v.orders.Test_Code;
            o.Order_Date = v.orders.Order_Date;
            o.Order_Status = v.orders.Order_Status;
            orderList.Add(o);
        }
        return orderList;
    }

    public List<PATIENT> getAllPatients()
    {
        List<PATIENT> patientList = new List<PATIENT>();

        var p = from pat in linqDat.PATIENTs select pat;
        foreach (var v in p)
        {
            PATIENT newPat = new PATIENT();
            newPat.Patient_ID = v.Patient_ID;
            newPat.Last_Name = v.Last_Name;
            newPat.First_Name = v.First_Name;
            newPat.DOB = v.DOB;
            newPat.Address = v.Address;
            newPat.City = v.City;
            newPat.State = v.State;
            newPat.Zip = v.Zip;
            patientList.Add(newPat);
        }
        return patientList;
    }

    public List<PATIENT> getPatientsByPhysicianID(String id)
    {
        List<PATIENT> patientList = new List<PATIENT>();

        var dat = from pat in linqDat.PATIENTs
                   join orders in linqDat.ORDERS on pat.Patient_ID equals
orders.Patient_ID
                   where orders.Physician_ID.Equals(id)
                   select new { pat};

        foreach (var v in dat)
        {
            PATIENT newPat = new PATIENT();
            newPat.Patient_ID = v.pat.Patient_ID;
            newPat.Last_Name = v.pat.Last_Name;
            newPat.First_Name = v.pat.First_Name;
            newPat.DOB = v.pat.DOB;

```



```

        newPat.Address = v.pat.Address;
        newPat.City = v.pat.City;
        newPat.State = v.pat.State;
        newPat.Zip = v.pat.Zip;
        patientList.Add(newPat);
    }
    return patientList;
}

public RESULT getResultByOrderID(String id)
{
    var result = (from res in linqDat.RESULTs
                  where res.Order_ID.Equals(id)
                  select new { res }).ToList();

    if (result.Count > 0)
    {
        return result.First().res;
    }
    else
    {
        return null;
    }
}

public List<RESULT> getResultListByOrderID(String id)
{
    var result = (from res in linqDat.RESULTs
                  where res.Order_ID.Equals(id)
                  select new { res }).ToList();

    List<RESULT> resultList = new List<RESULT>();

    foreach (var v in result)
    {
        RESULT newRes = new RESULT();
        newRes.Result_ID = v.res.Result_ID;
        newRes.Order_ID = v.res.Order_ID;
        newRes.Result_Date = v.res.Result_Date;
        newRes.Resulted_By = v.res.Resulted_By;
        newRes.Result_Report = v.res.Result_Report;
        resultList.Add(newRes);
    }
    return resultList;
}

public String getPhysicianIDByUserID(String id)
{
    var result = (from res in linqDat.PHYSICIANS
                  where res.User_Name.Equals(id)
                  select new { res }).ToList();

    if (result.Count > 0)
    {
        return result.First().res.Physician_ID;
    }
    else
    {

```

```
        return null;
    }
}

public bool isPatientExist(String id)
{
    var patients = (from dat in linqDat.PATIENTs
                     where dat.Patient_ID.Equals(id)
                     select new { dat }).ToList();

    if (patients.Count > 0)
    {
        return true;
    }
    else
    {
        return false;
    }
}

public void addPatient(PATIENT patient)
{
    linqDat.PATIENTs.InsertOnSubmit(patient);
    linqDat.SubmitChanges();
}

public void modifyPatient(PATIENT patient)
{
    PATIENT modPatient = getPATIENTByPtID(patient.Patient_ID.ToString());
    modPatient.Patient_ID = patient.Patient_ID;
    modPatient.Last_Name = patient.Last_Name;
    modPatient.First_Name = patient.First_Name;
    modPatient.DOB = patient.DOB;
    modPatient.Address = patient.Address;
    modPatient.City = patient.City;
    modPatient.State = patient.State;
    modPatient.Zip = patient.Zip;
    modPatient.Gender = patient.Gender;
    linqDat.SubmitChanges();
}

public PATIENT getPATIENTByPtID(String id)
{
    var patients = (from dat in linqDat.PATIENTs
                     where dat.Patient_ID.Equals(id)
                     select new { dat }).ToList();

    if (patients.Count > 0)
    {
        return patients.First().dat;
    }
    else
    {
        return null;
    }
}

public bool isOrderExist(String id)
```

```
{
    var orders = (from dat in linqDat.ORDERs
                   where dat.Order_ID.Equals(id)
                   select new { dat }).ToList();

    if (orders.Count > 0)
    {
        return true;
    }
    else
    {
        return false;
    }
}

public void addOrder(ORDER order)
{
    linqDat.ORDERs.InsertOnSubmit(order);
    linqDat.SubmitChanges();
}

public void modifyOrder(ORDER order)
{
    ORDER modOrder = getORDERByOrderID(order.Order_ID.ToString());
    modOrder.Order_ID = order.Order_ID;
    modOrder.Physician_ID = order.Physician_ID;
    modOrder.Test_Code = order.Test_Code;
    modOrder.Order_Date = order.Order_Date;
    modOrder.Order_Status = order.Order_Status;
    modOrder.Patient_ID = order.Patient_ID;
    modOrder.Site_ID = order.Site_ID;
    linqDat.SubmitChanges();
}

public ORDER getORDERByOrderID(String id)
{
    var orders = (from dat in linqDat.ORDERs
                   where dat.Order_ID.Equals(id)
                   select new { dat }).ToList();

    if (orders.Count > 0)
    {
        return orders.First().dat;
    }
    else
    {
        return null;
    }
}

public bool isResultExist(String id)
{
    var results = (from dat in linqDat.RESULTs
                   where dat.Result_ID.Equals(id)
                   select new { dat }).ToList();

    if (results.Count > 0)
    {

```

```
        return true;
    }
    else
    {
        return false;
    }
}

public void addResult(RESULT result)
{
    linqDat.RESULTs.InsertOnSubmit(result);
    linqDat.SubmitChanges();
}

public void modifyResult(RESULT result)
{
    RESULT modResult = getResultByResultID(result.Result_ID.ToString());
    modResult.Result_ID = result.Result_ID;
    modResult.Result_Date = result.Result_Date;
    modResult.Resulted_By = result.Resulted_By;
    modResult.Result_Report = result.Result_Report;
    modResult.Order_ID = result.Order_ID;
    linqDat.SubmitChanges();
}

public RESULT getResultByResultID(String id)
{
    var results = (from dat in linqDat.RESULTs
                   where dat.Result_ID.Equals(id)
                   select new { dat }).ToList();

    if (results.Count > 0)
    {
        return results.First().dat;
    }
    else
    {
        return null;
    }
}

public void writeToInterfaceLog(INTERFACELOG log)
{
    linqDat.INTERFACELOGs.InsertOnSubmit(log);
    linqDat.SubmitChanges();
}

public List<INTERFACELOG> getInterfaceLog()
{
    List<INTERFACELOG> logList = new List<INTERFACELOG>();
    var l = from log in linqDat.INTERFACELOGs select log;
    foreach (var v in l)
    {
        INTERFACELOG newLog = new INTERFACELOG();
        newLog.Date_Time = v.Date_Time;
        newLog.Log_Entry = v.Log_Entry;
        logList.Add(newLog);
    }
}
```

```
        return logList;
    }

    public void addSite(SITE site)
    {
        linqDat.SITEs.InsertOnSubmit(site);
        linqDat.SubmitChanges();
    }

    public void modifySite(SITE site)
    {
        SITE modSite = getSiteByID(site.Site_ID.ToString());
        modSite.Site_ID = site.Site_ID;
        modSite.Name = site.Name;
        modSite.Address = site.Address;
        modSite.City = site.City;
        modSite.State = site.State;
        modSite.Zip = site.Zip;
        linqDat.SubmitChanges();
    }

    public SITE getSiteByID(String id)
    {
        var sites = (from dat in linqDat.SITEs
                     where dat.Site_ID.Equals(id)
                     select new { dat }).ToList();

        if (sites.Count > 0)
        {
            return sites.First().dat;
        }
        else
        {
            return null;
        }
    }

    public void addTestCode(TEST test)
    {
        linqDat.TESTs.InsertOnSubmit(test);
        linqDat.SubmitChanges();
    }

    public void modifyTestCode(TEST test)
    {
        TEST modTest = getTestCodeByID(test.Test_Code.ToString());
        modTest.Test_Code = test.Test_Code;
        modTest.Description = test.Description;
        linqDat.SubmitChanges();
    }

    public TEST getTestCodeByID(String id)
    {
        var tests = (from dat in linqDat.TESTs
                     where dat.Test_Code.Equals(id)
                     select new { dat }).ToList();

        if (tests.Count > 0)
```

```

        {
            return tests.First().dat;
        }
        else
        {
            return null;
        }
    }

    public List<SITE> getListofAllSites()
    {
        List<SITE> sList = new List<SITE>();
        var s = from site in linqDat.SITEs select site;
        foreach (var v in s)
        {
            SITE newS = new SITE();
            newS.Site_ID = v.Site_ID;
            newS.Name = v.Name;
            newS.Address = v.Address;
            newS.City = v.City;
            newS.State = v.State;
            newS.Zip = v.Zip;
            sList.Add(newS);
        }
        return sList;
    }

    public List<TEST> getListofAllTests()
    {
        List<TEST> tList = new List<TEST>();
        var t = from test in linqDat.TESTs select test;
        foreach (var v in t)
        {
            TEST newT = new TEST();
            newT.Test_Code = v.Test_Code;
            newT.Description = v.Description;
            tList.Add(newT);
        }
        return tList;
    }

    public List<TestCodeReport> getTestCodeUseage()
    {
        var testcodes = from order in linqDat.ORDERs
                        join test in linqDat.TESTs on order.Test_Code equals
test.Test_Code
                        group new
                        {
                            order,
                            test
                        } by new
                        {
                            test.Test_Code
                        } into g
                        select new
                        {
                            testCount = g.Count(),
                            testCode = g.Key.Test_Code

```

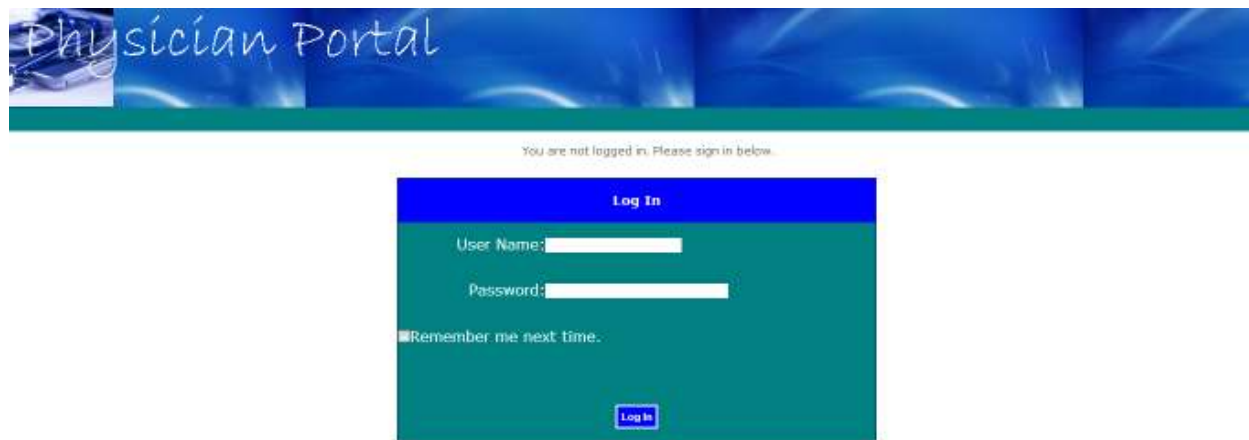
```
};

if (testcodes.Count() > 0)
{
    List<TestCodeReport> tList = new List<TestCodeReport>();
    foreach (var v in testcodes)
    {
        TestCodeReport tr = new TestCodeReport();
        tr.tetCode = v.testCode;
        tr.useageCount = v.testCount;
        tList.Add(tr);
    }
    return tList;
}
else
{
    return null;
}
}
}
```

## USER MANUAL

### Introduction

This is an application that can integrate with various health care systems and provide a centralized place for physicians to keep track of their patients. The application is designed to be seamless and easy for the physicians to use. All they have to do is log in and all of their patients will be available to use. The application also provides reporting functionality to track the performance of the physician group.



### Staff Level User:


You are not logged in. Please sign in below.

A screenshot of the Staff Level User login page. The form has a blue header with the text "Log In". Below the header, there are two input fields: "User Name:" and "Password:". Below these fields is a checkbox labeled "Remember me next time." and a blue "Log In" button at the bottom.



(Figure 1)


An administrator will supply you with login information. Visiting the site will prompt you to enter the supplied username and password to log in. (Figure 1)



	Patient ID	Last Name	First Name	DOB	Address	City	State	Zip
<input type="checkbox"/>	003	TEST003	NUMBER3	3/1/1945 12:00:00 AM	5833 N.WEST CIRCLE AVE.	CHICAGO	IL	60631
<input type="checkbox"/>	004	TEST004	NUMBER4	1/4/1945 12:00:00 AM	5833 N.WEST CIRCLE AVE.	CHICAGO	IL	60631
<input type="checkbox"/>	005	TEST005	NUMBER5	1/5/1945 12:00:00 AM	5833 N.WEST CIRCLE AVE.	CHICAGO	IL	60631
<input type="checkbox"/>	006	TEST006	NUMBER6	1/5/1945 12:00:00 AM	5833 N.WEST CIRCLE AVE.	CHICAGO	IL	60631
<input type="checkbox"/>	007	TEST007	NUMBER7	1/5/1945 12:00:00 AM	5833 N.WEST CIRCLE AVE.	CHICAGO	IL	60631
<input type="checkbox"/>	008	TEST008	NUMBER8	1/5/1945 12:00:00 AM	5833 N.WEST CIRCLE AVE.	CHICAGO	IL	60631
<input type="checkbox"/>	123	Webservice	JOHNNY	7/23/1967 12:00:00 AM	111 Webservice Lane	Web city	WI	11111
<input type="checkbox"/>	1234Carroll	Webservice	JIMMY	7/23/1967 12:00:00 AM	111 Webservice Lane	Web city	WI	11111
<input type="checkbox"/>	123Carroll	Webservice	JOHNNY	7/23/1967 12:00:00 AM	111 Webservice Lane	Web city	WI	11111
<input type="checkbox"/>	456Carroll	Webservice	TIMMY	7/23/1967 12:00:00 AM	111 Webservice Lane	Web city	WI	11111

(Figure 2)

As a staff user, you will have access to a patient list and the interface log. The patient list (figure 2) will display all patients that are being seen by physicians in the physician group.



The interface shows a header with a stethoscope icon and the text "Welcome, joe" followed by "Patients" and "Interface" tabs. Below is a log table with two columns: "Date\_Time" and "Log\_Entry".

Date_Time	Log_Entry
8/22/2013 9:30:09 AM	Patient id 005 modified.
8/22/2013 9:31:25 AM	Null patient ID received. MSH ^~\& Hospital System Hospital Test Physician Portal Physician Group 200912071648  ADT^A08 20091207164815862 P 2.3 EVN A08 200912071648 PID 1  15862^^^PRMCARESP  TEST005^NUMBER5  19450105 F  5833 N.WEST CIRCLE AVE.^CHICAGO^IL^60631  ((773)763-8403  M  360-36-9744 PV1  O ^PRMCARESP GT1 1 4744 RUER^SHERWIN^  5833 N.WEST CIRCLE AVE.^CHICAGO^IL^60631  ((773)763-8403  19390321 M  SPO    R IN1 1 EDI00003 B008 BC/BS OF ILLINOIS PPO P.O. BOX 805107^CHICAGO^IL^60680  ((800)972-8088 P16602   20050101  RUER^SHERWIN^ SPO 19390321 5833 N.WEST CIRCLE AVE.^CHICAGO^IL^60631  1     CTY000338393    M IN1 2 DEF DEF ^CHICAGO^IL^60631  2     TEST^TIMMY^D SEL 19451024 5833 N.WEST CIRCLE AVE.^CHICAGO^IL^60631  2     F IN1 3 DEF DEF ^CHICAGO^IL^60631  3     TEST^TIMMY^D SEL 19451024 5833 N.WEST CIRCLE AVE.^CHICAGO^IL^60631  3     F IN2 1  R IN2 2 360-36-9744  IN2 3 360-36-9744
8/22/2013 3:11:34 PM	Patient id 005 modified.
8/22/2013 3:50:46 PM	Patient id 005 modified.
8/22/2013 3:50:58 PM	Patient id 006 added.
8/22/2013 3:51:06 PM	Patient id 007 added.
8/22/2013 3:51:13 PM	Patient id 007 modified.
8/22/2013 3:51:22 PM	Patient id 008 added.
8/22/2013 3:51:36 PM	Order id 005444 added.
8/22/2013 3:51:47 PM	Order id 005333 added.
8/22/2013 3:52:03 PM	Order id 005222 added.
8/22/2013 3:52:20 PM	Order id 005222 modified.
8/22/2013 3:52:57 PM	Order id 005111 added.
8/22/2013 3:53:08 PM	Order id 005111 modified.
	Result contained an order ID that does not exist. Results can only be applied to an existing order.

(Figure 3)

The interface log (figure 3) will list all activity that the application has received through the web service interface. If there are problems with an imported message, the error will be written to the interface log along with the raw HL7 transaction so the issue can be researched.

### Administrator:

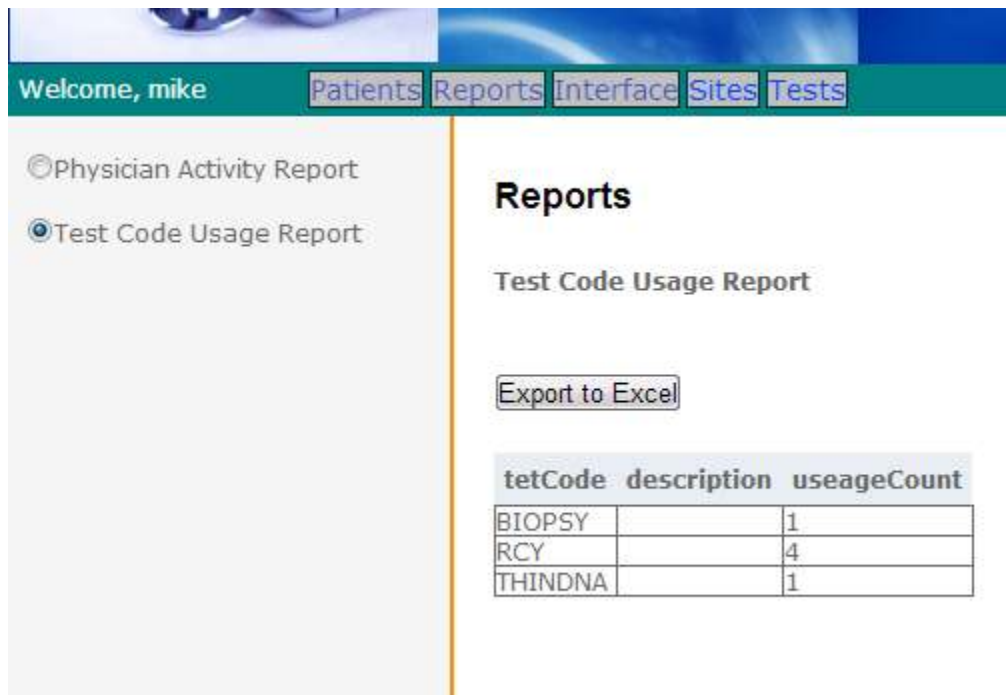
Administrators will be added to the application on deployment. These accounts have more functionality than the basic user accounts. The administrative user will be able to view the same patient list and interface log as the staff level users. In addition, the administrative user will also have access to the reports and table maintenance pages.



The screenshot shows a web application interface for a physician portal. At the top, there is a header with a stethoscope image and the text 'PHYSICIAN PORTAL'. Below this is a navigation bar with the text 'Welcome, mike' and several menu items: 'Patients', 'Reports', 'Interface', 'Sites', and 'Tests'. The main content area displays a table of patient data.

	Patient ID	Last Name	First Name	DOB	Address	City	State	Zip
<input type="checkbox"/>	003	TEST003	NUMBER3	3/1/1945 12:00:00 AM	5833 N.WEST CIRCLE AVE.	CHICAGO	IL	60631
<input type="checkbox"/>	004	TEST004	NUMBER4	1/4/1945 12:00:00 AM	5833 N.WEST CIRCLE AVE.	CHICAGO	IL	60631
<input type="checkbox"/>	005	TEST005	NUMBER5	1/5/1945 12:00:00 AM	5833 N.WEST CIRCLE AVE.	CHICAGO	IL	60631
<input type="checkbox"/>	006	TEST006	NUMBER6	1/5/1945 12:00:00 AM	5833 N.WEST CIRCLE AVE.	CHICAGO	IL	60631
<input type="checkbox"/>	007	TEST007	NUMBER7	1/5/1945 12:00:00 AM	5833 N.WEST CIRCLE AVE.	CHICAGO	IL	60631
<input type="checkbox"/>	008	TEST008	NUMBER8	1/5/1945 12:00:00 AM	5833 N.WEST CIRCLE AVE.	CHICAGO	IL	60631
<input type="checkbox"/>	123	Web service	JOHNNY	7/23/1967 12:00:00 AM	111 Web service Lane	Web city	WI	11111
<input type="checkbox"/>	1234Carroll	Web service	JIMMY	7/23/1967 12:00:00 AM	111 Web service Lane	Web city	WI	11111
<input type="checkbox"/>	123Carroll	Web service	JOHNNY	7/23/1967 12:00:00 AM	111 Web service Lane	Web city	WI	11111
<input type="checkbox"/>	456Carroll	Web service	TIMMY	7/23/1967 12:00:00 AM	111 Web service Lane	Web city	WI	11111

(Figure 4)



(Figure 5)

The application will come delivered with two reports (figure 5), the Physician Activity Report and the Test Code Usage Report. Additional reports can be requested from development. Each report can be exported to an Excel formatted file.

The screenshot shows a web application interface for a physician portal. At the top, there is a navigation bar with a welcome message 'Welcome, mike' and several menu items: 'Patients', 'Reports', 'Interface', 'Sites', and 'Tests'. Below the navigation bar, the page is divided into two main sections. On the left, there is a sidebar with two radio buttons: 'Add Site' (which is selected) and 'Update Site'. On the right, the main content area is titled 'Site Management'. Under this title, there is a section labeled 'Add Site'. This section contains a form with six input fields: 'Site ID:', 'Site Name:', 'Address:', 'City:', 'State:', and 'Zip:'. Each field is represented by a text label followed by a rectangular input box. At the bottom of the form, there is a 'Submit' button.

(Figure 6)

Table maintenance can be done from any of the maintenance pages (figure 6). You will have the option to either add a new record to the database or to modify an existing record.

## Physician

The physician is a specific user level designed to provide the physician with quick and easy access to the information that is most important to them. The physician will have access to the reports page along with a personalized patient list.

Welcome, john								
Physician Reports								
Patient ID	Last Name	First Name	DOB	Address	City	State	Zip	
+ 003	TEST003	NUMBER3	3/1/1945 12:00:00 AM	5833 N.WEST CIRCLE AVE.	CHICAGO	IL	60631	
+ 004	TEST004	NUMBER4	1/4/1945 12:00:00 AM	5833 N.WEST CIRCLE AVE.	CHICAGO	IL	60631	
+ 007	TEST007	NUMBER7	1/5/1945 12:00:00 AM	5833 N.WEST CIRCLE AVE.	CHICAGO	IL	60631	

(Figure 7)

Upon logging in, the physician will immediately see all patients that are in his care. The patient list can be expanded to display detailed information for each patient.

Welcome, john

Physician Reports

Patient ID	Last Name	First Name	DOB	Address	City	State	Zip								
+ 003	TEST003	NUMBER3	3/1/1945 12:00:00 AM	5833 N.WEST CIRCLE AVE.	CHICAGO	IL	60631								
= 004	TEST004	NUMBER4	1/4/1945 12:00:00 AM	5833 N.WEST CIRCLE AVE.	CHICAGO	IL	60631								
<table><tr><th>Order Id</th><th>Date</th><th>Test Code</th><th>Status</th></tr><tr><td>+ 004111</td><td>5/23/2012 12:00:00 AM</td><td>RCY</td><td>Resulted</td></tr></table>								Order Id	Date	Test Code	Status	+ 004111	5/23/2012 12:00:00 AM	RCY	Resulted
Order Id	Date	Test Code	Status												
+ 004111	5/23/2012 12:00:00 AM	RCY	Resulted												
+ 007	TEST007	NUMBER7	1/5/1945 12:00:00 AM	5833 N.WEST CIRCLE AVE.	CHICAGO	IL	60631								

(Figure 8)

Clicking on a row in the patient list will display a list of all orders for that patient (figure 8). If the order does not have a result report available yet the order status will say Pending. If the order does have a result report available to view the order status will say Resulted.

Welcome, john										Physician Reports					
Patient ID	Last Name	First Name	DOB	Address	City	State	Zip								
+	003	TEST003	NUMBER3	3/1/1945 12:00:00 AM	5833 N.WEST CIRCLE AVE.	CHICAGO	IL	60631							
=	004	TEST004	NUMBER4	1/4/1945 12:00:00 AM	5833 N.WEST CIRCLE AVE.	CHICAGO	IL	60631							
Order Id										Date		Test Code		Status	
=	004111		5/23/2012 12:00:00 AM			RCY		Resulted							
Result Date										Resulted By		Result Report			
10/7/2009 12:00:00 AM										MC		BRONCHOALVEOLAR LAVAGE, POSITIVE FOR PNEUMOCYSTIS JIROVECI (CARINII) BY DFA, GIARDIA LAMBLIA (CRITICAL/ALERT VALUE),			
+	007	TEST007	NUMBER7	1/5/1945 12:00:00 AM	5833 N.WEST CIRCLE AVE.	CHICAGO	IL	60631							

(Figure 9)

Clicking on the row for an order with an order status of resulted will display the result report for that order (figure 9).

## REFLECTION

The goal of this project was to develop a centralized means for physicians to access and view all of their patient data with an application that could be fully integrated into any healthcare environment. Tools were also to be developed that would allow the physician's group to analyze its performance. The goals of the project were accomplished but time constraints prevented some details from being implemented. Time management could have been better early in the project to manage and limit scope so a better quality finished product could have been delivered.

One main design consideration with this application was keeping a separation between the view and data layers. No controls on the view forms are bound directly to a database object. All forms reference the DataAccess layer, which manages the database queries. Another important design consideration was to make sure that the application had control over how the security was managed. This was accomplished by implementing custom AccountMember and AccountRole provider classes so the security would rely on the external SQL database and not the built in .NET functionality. My initial design hoped to implement more abstractions and design patterns. I was not able to accomplish that with the way I managed my time. However I did make sure to provide an abstract layer to the import process through the use of the IParser interface. I felt this was a key component to making the application extensible.

I spent a lot of time researching some new technologies for this project. A couple of the things that I spent a lot of time on but ultimately failed to include in the final project were MVC design and AJAX controls. They were both new concepts that I wanted to learn through this project. I did learn about each subject and I feel I know more about them now than I did a couple months ago, but I needed more time to be able to successfully incorporate them into the final application. I didn't compromise my reasons for wanting to use the MVC pattern and I feel



I still accomplished the goal of keeping a separation between data and view. While I did not include any AJAX controls in the final product I did implement a multi-nested GridView control. That is something that I hadn't used before and it required a bit of research to figure out how to accomplish the nested GridView where each layer is linked. Now that I understand how that type of control works that is something I will use again in future projects.

One of the key new components that I was successful in including in this application was web services. I had never worked with web services before. I was looking for a solution for how the HL7 messages from external systems would be delivered to the Physician Portal web server. Web services seemed like the natural solution. I spent some time researching web services, SOAP versus REST, and how to develop each. The SOAP service that was included in this project ended being the perfect solution to the message import need.

This application requires further development to fully implement some of the missing details. Additional reports need to be added, forms need to be added for additional table maintenance, search functionality needs to be included, and there needs to be a means for physicians to add their notes to the patient record. The items were missed in this version of the product due to inadequate time and not due to a lack of technical understanding. As time limitations were becoming obvious as the project progressed, priority was given to what was determined to be the key components necessary to meet the main goals of the application. Those goals were met. The items that were missed are important though and need to be added to make this a better quality product.

I enjoyed working on this project. While I regret that I didn't have the time to complete this application to my fullest capability, I am happy that I succeeded in meeting the original

stated goal and I was successful and learning and implementing some new technologies along the way.

**SELF EVALUATION**

Project Appropriateness:	5
Project Planning:	3
Software Analysis and Design:	4
Code Quality:	5
Software Usability:	5
Project Verification:	4
Effort:	5
Binder:	5
Presentation:	3