Table of Contents

TABLE OF CONTENTS .........................................................................................................................2

DOCUMENT REVISION HISTORY ......................................................................................................3

LIST OF FIGURES .................................................................................................................................4

LIST OF TABLES ....................................................................................................................................6

1. INTRODUCTION ..............................................................................................................................8

2. ARCHITECTURE OVERVIEW ..........................................................................................................11

3. CLIENT LAYER ..................................................................................................................................18

4. TRANSPORT LAYER ..........................................................................................................................37

5. BUSINESS LOGIC LAYER .................................................................................................................51

6. DATA STORAGE LAYER .....................................................................................................................65

7. QUALITY ASSURANCE ......................................................................................................................74

8. REQUIREMENTS MAPPING ...............................................................................................................81

9. ACCEPTANCE PLAN ..........................................................................................................................86

10. APPENDIX ......................................................................................................................................88
## Document Revision History

<table>
<thead>
<tr>
<th>Revision Number</th>
<th>Revision Date</th>
<th>Description</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>02/13/2014</td>
<td>First Draft</td>
<td>Version to be reviewed by peers</td>
</tr>
<tr>
<td>2.0</td>
<td>02/28/2014</td>
<td>Baseline</td>
<td>Revisions based off peer and sponsor feedback</td>
</tr>
</tbody>
</table>
List of Figures

Figure 1-1: General System Diagram .............................................................................................................. 9
Figure 2-1: Architecture Diagram .................................................................................................................. 12
Figure 2-2: Module Decomposition Diagram .................................................................................................. 16
Figure 2-3: Producer-Consumer Matrix ........................................................................................................ 17
Figure 3-1: Browser Display Module ............................................................................................................. 18
Figure 3-2: Web Validation Module ............................................................................................................... 22
Figure 3-3: Chart Generation Module .......................................................................................................... 25
Figure 3-4: Application UI Module ............................................................................................................... 28
Figure 3-5: Bluetooth Communication Module ........................................................................................... 31
Figure 3-6: Data Packet Processor Module .................................................................................................. 33
Figure 3-7: BPM Validation Module ............................................................................................................ 35
Figure 4-1: Request Encryption Module ....................................................................................................... 38
Figure 4-2: Response Decryption Module ................................................................................................... 40
Figure 4-3: Client/Server Communication Module ....................................................................................... 42
Figure 4-4: Request Decryption Module ....................................................................................................... 45
Figure 4-5: Permission Authentication Module .......................................................................................... 47
Figure 4-6: Response Encryption Module ................................................................................................... 49
Figure 5-1: Request Delegation Module ....................................................................................................... 52
Figure 5-2: Analysis Controller Module ....................................................................................................... 55
Figure 5-3: Analysis Controller Module ....................................................................................................... 58
Figure 5-4: Data Preparer Module ............................................................................................................... 60
Figure 5-5: Page Generator Module ............................................................................................................ 63
Figure 6-1: Data Controller Module .................................................................65
Figure 6-2: Cryptography Module .................................................................68
Figure 6-3: Database Get Module .................................................................70
Figure 6-4: Database Update Module .............................................................72
List of Tables

Table 3- 1: Browser Display Interfaces ........................................................................................................19
Table 3- 2: WebValidation Module Interfaces ..............................................................................................22
Table 3- 3: Chart Generation Module ........................................................................................................25
Table 3- 4: Application UI Module Interfaces ..............................................................................................29
Table 3- 5: Bluetooth Communication Module Interfaces .............................................................................31
Table 3- 6: Data Packet Processor Module Interfaces ..................................................................................33
Table 3- 7: BPM Validation Module Interfaces ............................................................................................36
Table 4- 1: Request Encryption Module Interfaces .......................................................................................39
Table 4- 2: Response Decryption Module Interfaces ....................................................................................41
Table 4- 3: Client/Server Communication Module Interfaces ........................................................................43
Table 4- 4: Request Decryption Module Interfaces .......................................................................................45
Table 4- 5: Permission Authentication Module Interfaces .............................................................................47
Table 4- 6: Response Encryption Module Interfaces ....................................................................................49
Table 5- 1: Request Delegation Module Interfaces .......................................................................................53
Table 5- 2: Analysis Controller Module Interfaces ........................................................................................56
Table 5- 3: Analysis Controller Module Interfaces ........................................................................................58
Table 5- 4: Data Preparer Module Interfaces .................................................................................................61
Table 5- 5: Page Generator Module Interfaces ..............................................................................................63
Table 6- 1: Data Preparer Module Interfaces .................................................................................................66
Table 6- 2: Cryptography Module Interfaces .................................................................................................69
Table 6- 3: Database Get Module Interfaces .................................................................................................70
Table 6- 4: Database Update Module Interfaces ............................................................................................72
Table 8- 1: Client Layer Module Requirements Mapping .............................................................................82
Table 8-2: Transport Layer Module Requirement Mapping ................................................................. 83
Table 8-3: Business Logic Layer Module Requirement Mapping ..................................................... 84
Table 8-4: Data Storage Module Requirement Mapping ..................................................................... 85
1. Introduction

The Detailed Design Specification Document will provide a low-level description of the wireless blood pressure monitoring system project. The document as a whole will detail how the system will be implemented by extending the architectural layers and subsystems defined in the Architecture Design Document. Beginning with a decomposition of the architecture’s subsystems into individual modules, the DDS will progress into defining the modules in terms of their specific functionality, interfaces and interactions, data structures, and pseudo code that transcribes their process. It will conclude by describing the team's plans for testing the product to ensure that it functions properly and that it meets the requirements specified in the System Requirements Specification document.

1.1 Project Scope

The goal of the project is to create a simple way for patients to monitor their blood pressure with the guidance of their primary physician. The patient will be able to take periodic blood pressure readings using the Bluetooth-enabled blood pressure monitor that comes as part of the system. The BPM will transmit the patient's blood pressure readings to a mobile application that will then transport the data to the system's web server.

A web service will be provided that will allow patients and their doctors to monitor the patient's blood pressure as it changes with time. The service will be able to display information about the patient in various tables and graphs, give general advice to the patient based on the data provided, and provide a way for patients to make notes and annotations on specific results, so that their doctor can get a better grasp of the situation. It will also provide an alert system, which will allow patients to get the attention of their doctor by setting an alert. In addition, the alert system will be able to automatically send an alert to the doctor if it detects a concerning trend in the patient's recent health.

1.2 Project Description

The wireless blood pressure monitor project is comprised of two separate components: a wireless blood pressure monitor and a web service. The blood pressure monitor and the web service will communicate wirelessly over a secure network.

The blood pressure monitor will handle authenticating the patient, taking blood pressure readings, and transmitting the readings over Wi-Fi to the system's web server. Users will need to establish a connection from the monitor to their Wi-Fi service in order for the monitor to begin transmitting data. The patient’s account will have to be created and their monitor authentication credentials established on the web service in order to authenticate successfully and begin storing their blood pressure data.
After the monitor measures their blood pressure, the patient can view their results from the monitor, or they can log into their web account to see them.

The web service will allow doctors and patients to set up accounts so as to monitor a patient's data. The doctor will create their account, and then be able to create accounts for their patients. Those registered under a Doctor account will be able to keep tabs on their patient(s) by letting them see their patients' information and readings. Doctors will receive an alert when a patient thinks something is wrong, or when the system believes that it has detected something concerning about a patient's blood pressure results, so that they can immediately check on the patient to ensure that everything is alright or to get more information about the problem.

Those with a Patient account will have a profile page that graphically displays all their results in various tables and graphs, the ability to make annotations on any specific blood pressure reading, the ability to create alerts for their doctor, and an advice page that will give the patient general advice based on their blood pressure results. The figure below illustrates the doctor and patient interaction with the system.

![Figure 1-1: General System Diagram](image_url)
1.3 Acronyms and Interchangeable Terms

**Acronyms:**
- BP - blood pressure
- BPM - blood pressure monitor
- PHI – protected health information
- HIPAA - Health Insurance Portability and Accountability Act
- ANSI - American National Standards Institute
- AAMI - Association for the Advancement of Medical Instrumentation

**Interchangeable Terms:**
- User, Patient
- Physician, Doctor
- Web Service, Web Application, Website
- BPM, Device
- Mobile, Android
2. Architecture Overview

This section will describe the architecture of the wireless blood pressure monitor system, as determined by the Architecture Design Document. The defined subsystems, for each layer, will be broken down into modules that perform specific tasks for each respective subsystem. Each module will be defined and described in terms of functionality and internal and external interactions.

2.1 Architecture

The architecture of the wireless blood pressure monitoring system was designed with three major considerations in mind: it must be simple and easy-to-follow, it must be secure, and it must be modular and easy to extend. With these factors in mind, the team developed a four layer architecture that is based off of the Client-Server model. The layers stack upon each other structurally such that the interaction with the system starts at the top, descends down to the bottommost layer, and then ascends back to the top stack. From top to bottom, the layers are: Client Layer, Transport Layer, Business Logic Layer, and Data Storage.

The Client Layer is where users will interact with the system, as well as where the BPM will send its data to. It encompasses everything that happens on the client side of the system, from input taking to data processing and so on. The layer has four subsystems: a Web UI subsystem that allows users to get information from and interact with the website, a Web IO Processing subsystem that takes care of input validation and the processing of certain web features, a BPM UI subsystem that lets patients send input to the phone application, and a BPM IO Processing subsystem that validates user input and data from the BPM.

The Transport Layer is in charge of getting data from the client to the server and back the other way. It also deals with making the data secure so that it cannot be corrupted or stolen by any unauthorized sources. The layer has three subsystems: a Client Security subsystem that encrypts data being sent to the server and decrypts data coming from the server, a Network Transport subsystem that handles the actual transfer, and a Server Security subsystem that does the exact opposite of the Client Security.

The Business Logic Layer does all of the processing that the server is tasked with. Three subsystems reside here: a System Controller subsystem that acts as a central hub for the various parts of the server and delegates tasks to those components, an Analysis subsystem that looks through various sets of data to find patterns, more detailed information, and other things of interest, and a Response Controller that takes the data gathered by the server and prepares a response for the client.

The lowest level of the architecture, the Data Storage layer, is what connects the server processing to the data stored on the system’s database. It contains three subsystems: a Storage Controller that acts as an interface between the System Controller and the actual database commands and breaks general data requests into smaller database commands to process, a Database Security subsystem that encrypts key data going into the database and decrypts the same data going out, and a Database Interface that handles the actual SQL commands that need to be used to talk to the database.
Figure 2-1: Architecture Diagram
2.2 Module Decomposition
Each subsystem in the architecture plays a fundamental role in the functionality of the entire system as a whole. Here the modules of the subsystems will be identified so that they implement a single task regarding the functionality of the subsystem. Depending on the functionality, a subsystem can be composed of a single module or multiple modules. The modules for each subsystem of every architectural layer are listed below along with a brief description.

1. Client Layer

1.1 Web UI
   Browser Display
   The graphical elements constructed from HTML, CSS, and JavaScript that will be displayed on a user’s browser to capture user interaction.

1.2 Web IO Processing
   Web Validation
   Verification of user input based off individual criteria for each input type.

   Chart Generation
   Generates patient data charts that are requested by the user.

1.3 BPM UI
   Application UI
   The graphical elements constructed for an Android device to capture user interaction.

1.4 BPM IO Processing
   Bluetooth Communication
   The Bluetooth communication protocol that would exchange data between the blood pressure monitor and the Android device.

   Data Packet Processor
   The Processing of data packets received from the BPM to be used by BPM Validation module.

   BPM Validation
   Verification of input based off individual criteria for each input type.
2. Transport Layer
   2.1 Client Security
       Request Encryption
       The encryption of requests made by the Client Layer to be sent over the network.

       Response Decryption
       The decryption of the server’s response passed to the Client Layer.

   2.2 Network Transport
       Client/Server Communication
       The libraries responsible for transmitting the data from client to server and back over SSL.

   2.3 Server Security
       Response Encryption
       The encryption of the server’s response to be sent to client

       Request Decryption
       The decryption of the request from client to be handed to server

       Permission Authentication
       The authentication checks made on a client request to determine the permissions allowed for the user.

3. Business Logic Layer
   3.1 System Controller
       Request Delegation
       The controller for the server, handles data distribution between the various other modules of the server and delegates tasks to generate a response.

   3.2 Data Analysis
       Analysis Controller
       Handles the general steps of an analysis, not the actual calculations and checks. Stores the results of the analysis.

       Analysis Command
       Does calculations and checks based on the tasks sent from the Analysis Controller.
3.3 Response Controller

Data Preparer
Does final preparations for responding, including reformatting data if need be, loading needed data into the correct object, and getting the correct page or data to respond with.

Page Generator
Translates JTSL, EL, and other server-side code elements into the data the Data Preparer has provided.

4. Data Storage Layer

4.1 Storage Controller

Data Controller
The delegation of requests made to the database. Breaks up data into specific segments to pass through specific modules.

4.2 Database Security

Cryptography
Handles the encryption and decryption of data passing to and from the database.

4.3 Database Interface

Database Get
Module that handles finding and getting specific data from the database.

Database Update
Module that handles finding and modifying specific data in the database.
2.3 Module Data Flow Diagram

The diagram below shows each of the modules inside of their correct subsystems, and how the modules interact and send data between each other.

Figure 2- 2: Module Decomposition Diagram
2.3 Producer Consumer Matrix

The table below shows the relationships between modules from a different perspective. Here you can clearly see who talks to who, and who produces data for others. This also shows that most of the modules do not rely on a great deal of other modules, and that no module is having to produce for many others. The most interactive modules appear to be the Request Delegator (which was intentionally developed this way) and some of the client modules (which have to handle browsers, phones, BPMs, and a variety of languages, which limits how customizable they can be).

![Producer-Consumer Matrix Diagram]

**Figure 2-3: Producer-Consumer Matrix**
3. Client Layer

The Client Layer modules are meant to handle all functions relating to user interaction. This includes creating the graphical layout for the web browser and BPM, capturing input, and processing the data. After the input is processed it’s relayed to the Transport Layer to be sent to the server. And in turn, the Client layer modules receive the server’s response through the Transport layer. The server’s response is displayed to the user through the UI in a user friendly manner.

3.1 Web User Interface

The Web UI subsystem will allow patients and doctors to interact with the web server through a supported web browser. This subsystem will handle producing the graphical elements responsible for user interaction. From the Web UI subsystem patients will be able to view their blood pressure readings, basic advice, alerts, or messages, and be able to make certain changes to their profile information and send messages to their doctor. The doctors will be able to view their patients profile, BP data, and alerts as well as be able to send them messages. This subsystem will interface with the user through various web pages on a web browser.

3.1.1 Browser Display

Figure 3- 1: Browser Display Module
**Prologue**

The Browser Display Module will consist of the HTML, CSS, JavaScript animation scripts, and other elements that make up the graphical interface that will be displayed on a user’s web browser. This module is where users will interact with the web portion of the system.

**Interfaces**

<table>
<thead>
<tr>
<th>Source</th>
<th>Sink</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>Browser Display</td>
<td>User Input</td>
<td>Visual response</td>
</tr>
<tr>
<td>Browser Display</td>
<td>Web Validation</td>
<td>HTML element (form, button, etc.)</td>
<td>Boolean for Invalid, or N/A for Valid</td>
</tr>
<tr>
<td>Response Decryption</td>
<td>Browser Display</td>
<td>Web page (HTML, JavaScript, CSS, etc.)</td>
<td>N/A</td>
</tr>
<tr>
<td>Chart Generation</td>
<td>Browser Display</td>
<td>JavaScript Chart object</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 3-1: Browser Display Interfaces

**External Data Dependencies**

- Keyboard Input
- Mouse Events
- HTML5 Library
- Firefox version 27.0
- Internet Explorer 10.0
- Google Chrome 32.0

**Internal Data Dependencies**

N/A
Detailed Design Specification

Pseudocode

<!DOCTYPE html>
<head>
<!-- all external java script functions will need to be link -->
<script src="javascriptfile.js"></script>
<!-- all external style/theme will need to be link -->
<link rel="stylesheet" type="text/css" href="theme.css" />
</head>
<title>Web Page</title>
</head>
<body>
<!-- Top header division for banner and login portal. -->
<div id="banner"></div>
<!-- Tool bar division to hold navigation and other quick access points. -->
<div id="toolbar"></div>
<!-- Content division to hold all of the main information of the current page for display, can change depending on the current web page. -->
<div id="content">
<!-- More charts can be added to the chartbox for more variety. Each chart can have different types of interaction base on what the developer wishes to implement when adding the charts. -->
<div id="chartbox">
<div>
<span>Line Chart</span>
<div id="linechart" class="chart"></div>
</div>
<div>
<span>Range Chart</span>
<div id="rangechart" class="chart"></div>
</div>
<div>
<span>Pie Chart</span>
<div id="piechart" class="chart"></div>
</div>
</div>
<!-- Information regarding the company, methods of contact, and various other details. -->
<div id="metaData"></div>
</body>
</html>

<!-- All standard css formatting for the theme of the Web UI will be contain into one file and shared throughout the html pages.
Any css style that is specific to a page must be separated into different css and link to the appropriate pages.
function onLoad(id) //Each id of the html element will be passed in for processing.
{
    if (!document.getElementById || !document.getElementsByTagName)
        return false;

    initialize();
}

function initialize()
{
    for (each item in the html element)
    {
        //add actionEvent(input)
    }
}

//actionEvent will process the event when trigger and execute the animation.
function actionEvent(input)
{
    //Algorithm for execution of the animation.
}

/* All general css theme for the website will be condense and separate into a theme.css file. Any specific styling will be separate into its own css file and linked to appropriate web pages.*/
{
    Background-color: white;
}

h1
{
    font-size: 40px;
    font-family: "Times New Roman", Georgia, Serif;
}

Input { border: 5px solid red; }
3.2 Web IO Processing

The Web IO Processing subsystem will be responsible for processing the user input received from the Web UI subsystem. The processing that this subsystem will perform includes validating the input data for correct format, encoding, or range. Based on the validation check this subsystem will give feedback to the Web UI subsystem, on the event that the validation failed, or forward the data to the Client Security subsystem in the Transport layer for handling the security of the data.

3.2.1 Web Validation

![Web Validation Module]

**Prologue**

The Web Validation Module evaluates the inputs sent from the Browser Display module. Each input element will be verified for valid character input – only alpha numerical characters allowed, valid language – English UTF-8 encoding, and valid username sequence – email sequence. After validation has been completed an error message will be returned back to the Browser Display if any error is detected. If no error is detected the request will be sent to the Request Encryption Module for encryption and transport to the server for further processing.

**Interfaces**

<table>
<thead>
<tr>
<th>Source</th>
<th>Sink</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Browser Display</td>
<td>Web Validation</td>
<td>HTML element (form, button, etc.)</td>
<td>Boolean for Invalid, or N/A for Valid</td>
</tr>
<tr>
<td>Web Validation</td>
<td>Request Encryption</td>
<td>HttpConnection Object</td>
<td>N/A</td>
</tr>
</tbody>
</table>

*Table 3-2: WebValidation Module Interfaces*
External Data Dependencies

- JQuery library version 1.x
- JavaScript library version 1.x

Internal Data Dependencies

N/A

Pseudocode

```javascript
function validateForm() {
    /* errorCounter is used to count the number of invalid inputs. Counter must be 0 for form to be submitted. */
    var errorCount = 0;

    /*
    Each field is extracted from the form and have their values stored into respective variables.
    */
    var element = document.forms["Form"]["elementName"].value;

    /*
    Any form of regex used to check against certain particular test case.
    The regex for java script will be using ECMA-262 Standard.
    */
    var regex = new RegExp(\w+);

    /*
    Each input field will be tested for valid/invalid input. Standards for input will vary depending on the types of input field given by the form. Each field will raise the errorCounter if it contains any form of invalid inputs. Each input field will be marked red for invalid input or green for valid input after processing.
    */
    if (element contains error) {
        document.getElementById("elementID").style.border = "2px solid red";
        errorCounter++;
    } else {
        document.getElementById("elementID").style.border = "2px solid green";
    }
}```
/*
   errorCounter will be evaluated at the end and return a value back to the webpage.
   */

   if (errorCounter > 0) {
       return false;
   } else {
       return true;
   }
3.2.2 Chart Generation

Prologue

The Chart Generation module will generate charts requested from the Browser Display module. The separation made between the charts and the web pages was made because not all web pages will need to generate charts; the pages that will directly handle chart generation are handled by this module. When a request is received from a web page this module will send a request to the Request Encryption module for user information from the database. Upon retrieval of the data object, this module will generate chart element base on the data, and insert them into the web page.

Interfaces

<table>
<thead>
<tr>
<th>Source</th>
<th>Sink</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chart Generation</td>
<td>Browser Display</td>
<td>JavaScript Chart object</td>
<td>N/A</td>
</tr>
<tr>
<td>Response Decryption</td>
<td>Chart Generation</td>
<td>Data in JSON format</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 3- 3: Chart Generation Module

External Data Dependencies

- JQuery library version 1.x
- JavaScript library version 1.x
- Highcharts library

Internal Data Dependencies

- Data series
**Pseudocode**

```
/*
Basic format of how each of the chart variables is to be constructed. These are the bare
bone minimum of the charts creation.
Each chart will call for a different types of data needed for construction from the
database. Each data request will be formatted into data series for the highcharts to take in
and create each graph.
*/

var dataseries1;
var dataseries2;
var chart = {

/*
Each chart will be render to a specific region on the webpage.
*/

chart : { renderTo : 'chartArea', },

/*
Axis min and max will be calculated by the data object provided from the database.
*/

xAxis : {
    min : xMin,
    max : xMax,
    title : {
        text : 'X Axis Title'
    },
},

yAxis : {
    min : yMin,
    max : yMax,
    title : {
        text : 'Y Axis Title'
    }
},

/*
Each chart will be customize with different plotOptions depending on what type of chart
you are and how you wish to interact with the chart. Please refer to the highcharts library
and tutorial for further examples.
*/

plotOptions : { },

/*
beta blockers
```
Information pertaining to a series can be viewed by hovering over the data series and a tool tip will display information corresponding to the series. Each of the series has its own name and will return tool tip data corresponding to its respective name. Tool tips may change depending on what type of chart is currently being created.

```javascript
tooltip: {
  enabled: true,
  formatter: function() {
    if (this.series.name == "Data Name") {
      return "information regarding this data series";
    }
  }
},

legend: { enabled: true },

series: [
  {
    type: 'arearange',
    color: '#00BFFF',
    name: 'Low',
    data: dataseries1
  },
  {
    type: 'line',
    color: '#00CC00',
    name: 'Normal',
    data: dataseries2
  }
];
```
3.3 BPM User Interface

The BPM UI subsystem handles the interaction between the user and the mobile application. It allows users to see their blood pressure results once they have been transferred, to send the results to the server, to alter settings, and everything else the patient will need.

3.3.1 Application UI

Application UI captures user input from an Android device through a touch screen constructed from various XML layout files and event listeners. The XML layout files construct the buttons, titles, text fields, and checkboxes for user interaction. The event listeners correspond to each UI element to ‘listen’ to button presses, text input, or check marks that signify the user interaction. The layout will be constructed so that it is user friendly, flexible so that it is compatible with all mobile devices supporting Android, responsive to user interaction, and visually and target-ability comfortable. This will be made possible by following Android UI metrics for different displays.
Detailed Design Specification

Wireless Blood Pressure Monitor

**Interfaces**

<table>
<thead>
<tr>
<th>Source</th>
<th>Sink</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>Application UI</td>
<td>User Input (Login Credentials, BPM Mac Address, etc.)</td>
<td>Operation Success</td>
</tr>
<tr>
<td>Application UI</td>
<td>User</td>
<td>UI Display Output</td>
<td>N/A</td>
</tr>
<tr>
<td>Application UI</td>
<td>BPM Validation</td>
<td>User Input (Login Credentials, BPM Mac Address, etc.)</td>
<td>Boolean Valid</td>
</tr>
<tr>
<td>BPM Validation</td>
<td>Application UI</td>
<td>String Result(s) (for BPM packets)</td>
<td>N/A</td>
</tr>
</tbody>
</table>

*Table 3- 4: Application UI Module Interfaces*

**External Data Dependencies**

- Android SDK API level 11 to 19
- Java SDK 1.7

**Internal Data Dependencies**

N/A
Pseudocode

```xml
<LinearLayout
    xmlns:android=http://schemas.android.com/apk/res/android
    tools:context=".LoginPage">
    <TextView
        android:id="@+id/error_warning" />
    <EditText
        android:id="@+id/username"
        android:inputType="text" />
    <EditText
        android:id="@+id/password"
        android:inputType="textPassword" />
    <CheckBox
        android:id="@+id/remember_username"
        android:text="@string/remember_username" />
    <Button
        android:id="@+id/login"
        android:text="@string/login_button" />
    <Button
        android:id="@+id/cancel"
        android:text="@string/cancel_button" />
</LinearLayout>
```
3.4 BPM IO Processing

The BPM IO Processing subsystem handles local processing of patient input. The processing will validate input, handle the Bluetooth connection to the monitor, deal with packets sent by the monitor, and translate the details of the packet so that they are able to be read by the patient.

3.4.1 Bluetooth Communication

![Bluetooth Communication Module Diagram]

Figure 3-5: Bluetooth Communication Module

**Prologue**

Bluetooth Communication module will handle initializing and setting up the Bluetooth communication protocol for the Android device, including the mobile’s hardware adapter. The module will be able to accept BP data from the blood pressure monitor device over the protocol initialized in the form of data packets.

**Interfaces**

<table>
<thead>
<tr>
<th>Source</th>
<th>Sink</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPM</td>
<td>Bluetooth Communication</td>
<td>BP Data Packets</td>
<td>N/A</td>
</tr>
<tr>
<td>Bluetooth Communication</td>
<td>Data Packet Processor</td>
<td>BP Data Packets</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 3-5: Bluetooth Communication Module Interfaces
External Data Dependencies

- Android SDK API level 11 to 19
- Java JDK 1.7
- Android Bluetooth library

Internal Data Dependencies

- BluetoothAdapter _adapter;
- BluetoothDevice _device;
- BluetoothSocket _socket;
- ArrayList _queue;

Pseudocode

// Create Data Structures
// The BluetoothDevice is Class found in the Android Bluetooth library required to setup a connection to the remote Bluetooth adapter found on BPM device.
BluetoothAdapter _adapter;

// The BluetoothAdapter is Class found in the Android Bluetooth library required to instantiate the devices Bluetooth adapter.
BluetoothDevice _device;

// The BluetoothSocket is Class found in the Android Bluetooth library required to open a connection to the BPM. The connection will allow input and output streams to form so data can be exchanged between the Android device and the BPM.
BluetoothSocket _socket;

// Store the data packets recieved from BPM
List queue = new ArrayList();

// Enable Android device BlueTooth
_adapter = BluetoothAdapter.getDefaultAdapter();

// Pair Android with BPM using BT MAC address(Entered by Patient)
_device = mBTAdapter.getRemoteDevice(bt_address);

// Create BluetoothSocket for a connection with BPM
_socket = _device.createRfcommSocketToServiceRecord(UUID);
Method m = _device.getClass().getMethod("createRfcommSocket", new Class[]{int.class});
_socket = (BluetoothSocket) m.invoke(_device, 1);

// start communication stream
outStream = _socket.getOutputStream();
inStream = _socket.getInputStream();

// Read BP data packets
if(inStream != null) {
    num bytes read = inStream.read(buffer);
    queue buffer; }
### 3.4.2 Data Packet Processor

![Diagram of Data Packet Processor Module]

**Figure 3-6: Data Packet Processor Module**

#### Prologue

Data Packet Processor module handles the processing of the data packets received from the BPM through the Bluetooth Communication module. The processing includes determining the type of packet received – blood pressure data or general data packet, detecting corrupt data, and decrypting the data to pass to the BPM Validation module.

#### Interfaces

<table>
<thead>
<tr>
<th>Source</th>
<th>Sink</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bluetooth Communication</td>
<td>Data Packet Processing</td>
<td>Data Packet</td>
<td>N/A</td>
</tr>
<tr>
<td>Data Packet Processing</td>
<td>BPM Validation</td>
<td>String BP Result</td>
<td>N/A</td>
</tr>
</tbody>
</table>

*Table 3-6: Data Packet Processor Module Interfaces*

#### External Data Dependencies

- Android SDK API level 11 to 19
- Java JDK 1.7
Internal Data Dependencies

- Array List for data BP Packet
- Array List for data General Packet

Pseudocode

// Continuously process the data packets received from Bluetooth Communication module
while(data packet queue -> has Packet)
{
    // Check packet length
    if (packet.length <= 0 || packet.length < MINIMUM_LENGTH || packet.length > MAXIMUM_LENGTH)
        throw new Exception("Incorrect size. Packet Invalid");

    // Check for packet header
    else if (packet[0] != Start_Code)
        throw new Exception("Packet Without Header. Invalid");

    // Check packet CRC for uncorrupted data
    byte crc = packet[crc_position];
    else if (crc8.Calculate(packet) != crc)
        throw new Exception("Wrong CRC. Invalid Packet.");

    // Check for correct packet format
    else if ((packet_end != ACK) && (packet_end != NAK) && (packet_end != END_OF_TEXT))
        throw new Exception("Wrong end. Invalid Packet");

    // Determine if BP data or General Data, Decrypt
    else
        if (packet.msgID == BP_DATA_ID)
            dataBP_Pkt.add(packet);
            decryptPacket(dataBP_Pkt);
        else
            dataGeneral_Pkt.add(packet);
            decryptPacket(dataBP_Pkt);
}
### 3.4.3 BPM Validation

#### Prologue

BPM Validation module handles the authentication of all data passing through the BPM UI and BPM IO Processing Subsystems. User input is verified for valid character input – only alpha numerical characters allowed, valid language – English UTF-8 encoding, and valid username sequence – email sequence. Strings from the Data Packet Processing module are verified for valid characters – only alpha numerical characters allowed. Response from the Response Decryption will be validated for Boolean response – any other form of input will not be accepted. After validating the input this module either returns an error (if invalid error occurs) to the Application UI or Data Packet Processing modules; passes the objects (user input or BP data) to the Request Encryption module for encryption and transport to server; or passes the Boolean (valid|invalid) response from server to Application UI.

#### Interfaces

<table>
<thead>
<tr>
<th>Source</th>
<th>Sink</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application UI</td>
<td>BPM Validation</td>
<td>User Input String(s)</td>
<td>Boolean Valid</td>
</tr>
<tr>
<td>Data Packet Processing</td>
<td>BPM Validation</td>
<td>BP Data Packets</td>
<td>N/A</td>
</tr>
<tr>
<td>BPM Validation</td>
<td>Request Encryption</td>
<td>HttpConnection object</td>
<td>N/A</td>
</tr>
<tr>
<td>BPM Validation</td>
<td>Application UI</td>
<td>Response String(s)</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Table 3-7: BPM Validation Module Interfaces

External Data Dependencies

- Android SDK API level 11 to 19
- Java JDK 1.7
- Java String Processing Libraries

Internal Data Dependencies

N/A

Pseudocode

if(String.chars < 0x30 || (String.chars >= 0x3a && String.chars <= 0x40) ||
    (String.chars > 0x5a && String.chars <= 0x60) || String.chars > 0x7a)
    return "Invalid. Not Alphanumeric";
if(String.encoding != encoding("utf-8"))
    return "Invalid. Not English";

if(! Patterns.EMAIL_ADDRESS.matcher(String).matches()) {
    return "Invalid. Not Email"

if(!parseBoolean(String) || !getBoolean(String).equal('true') || !getBoolean(String).equal('false'))
    return "Invalid. Not Boolean"
4. Transport Layer

The transport layer is the interface for communications between the Client and Business layer in the system architecture. It is meant to prepare data that is to be sent and received between the client and the server where the data is to be handled outside of the client and server devices, i.e. over the internet. Therefore the data must be prepared in a way that makes it obscure to external parties. The data, when received on the server and the client, must be able to be decrypted and decoded. It is intended to use as many available, renowned robust solutions instead of attempting to develop one from scratch due to the project’s resource constraints as their designs are very complex. This is referring to the existing HTTPS API libraries available for both the client and the server to transport the data outside of the client and server machines as well as any additional symmetric encryption, “same key” encryption, to add additional layers of data obfuscation.

This layer is split up into 3 subsystems: client security, network transport, and server security. Both the client security and the server security subsystems are similar in functionality in that they handle any security functions needed to be done on the data that is to be sent or received between the client and the server. In particular, they handle the encryption of data before it is sent and decryption of the data as it is received, either by the server or the client. Also, they will perform any additional security related functionality such as verifying data integrity and client/host authentication and authorization. The network transport encapsulates the process of communications over a network or the internet. This basically embodies the entire OSI network model from the application to physical layers. Most of the network communications at this time is intended to use the HTTPS protocol for simplicity and conformance since it is very widely used and maintained amongst a vast variety of API libraries.
4.1 Client Security

The Client Security subsystem is responsible for handling the security and integrity of data before it is transmitted to the web server and when it is received from the web server. The subsystem encrypts the data received from the Web IO and BPM IO Processing subsystems and passes the now encrypted data to the Network Transport subsystem for transmission. When this subsystem receives data from the Network Transport subsystem it decrypts it and passes the decrypted data to either the Web or BPM UI subsystems.

4.1.1 Request Encryption

![Request Encryption Module](image)

Figure 4-1: Request Encryption Module

**Prologue**

The Request Encryption module prepares data that is to be sent from the respective client modules to the server across the internet. This data will be encrypted using a symmetric encryption approach based off of the AES-128 encryption. This will be handled automatically when sending through HTTPS.
### Interfaces

<table>
<thead>
<tr>
<th>Source</th>
<th>Sink</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web Validation</td>
<td>Request Encryption</td>
<td>HttpConnection object</td>
<td>N/A</td>
</tr>
<tr>
<td>BPM Validation</td>
<td>Request Encryption</td>
<td>HttpConnection object</td>
<td>N/A</td>
</tr>
<tr>
<td>Request Encryption</td>
<td>Client/Server Communication</td>
<td>HttpConnection object (now encrypted)</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 4-1: Request Encryption Module Interfaces

### External Data Dependencies

- JQuery library version 1.x
- JavaScript library version 1.x
- Android SDK API level 11 to 19
- Java JDK 1.7
- java.net.*
- java.net.ssl.*

### Internal Data Dependencies

N/A

### Pseudocode

N/A
4.1.2 Response Decryption

Prologue

This module decrypts the encrypted data received from the server as part of a HTTP request/response communication pattern. The encryption method is expected to be known or derived from the encrypted data’s packaging. This will be handled automatically when sending through HTTPS.
Interfaces

<table>
<thead>
<tr>
<th>Source</th>
<th>Sink</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client/Server Communication</td>
<td>Response Decryption</td>
<td>Web page or data to be decrypted</td>
<td>N/A</td>
</tr>
<tr>
<td>Response Decryption</td>
<td>Chart Generation</td>
<td>Data in JSON format</td>
<td>N/A</td>
</tr>
<tr>
<td>Response Decryption</td>
<td>BPM Validation</td>
<td>Boolean response</td>
<td>N/A</td>
</tr>
<tr>
<td>Response Decryption</td>
<td>Browser Display</td>
<td>Web page or data (now decrypted)</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 4-2: Response Decryption Module Interfaces

External Data Dependencies

- JQuery library version 1.x
- JavaScript library version 1.x
- Android SDK API level 11 to 19
- Java SDK 1.7
- Java.net.*
- Java.net.ssl.*

Internal Data Dependencies

N/A

Pseudocode

N/A
4.2 Network Transport

The Network Transport subsystem is responsible for handling the lower-level communications between the Client and the Business Logic layers. This subsystem mainly consists of the libraries that handle the http communications that will be transmitted over the Secure Sockets Layer (SSL) network protocol.

4.2.1 Client/Server Communication

![Client/Server Communication Module](image)

**Figure 4-3: Client/Server Communication Module**

**Prologue**

This module is responsible for handling the transmission of data between the client and server across a public network, i.e. the internet. It basically encapsulates capabilities provided by API libraries, implementing the OSI network model, on any two communicating systems across a network. For the scope of this project, only API’s exposing the application-level layer of the OSI model would be used. The data will be transported through HTTPS which is the use of the HTTP protocol on top of the SSL/TLS to ensure a secure transfer.
Detailed Design Specification

Wireless Blood Pressure Monitor

Interfaces

<table>
<thead>
<tr>
<th>Source</th>
<th>Sink</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request Encryption</td>
<td>Client/Server Communication</td>
<td>HttpClientConnection object</td>
<td>N/A</td>
</tr>
<tr>
<td>Response Encryption</td>
<td>Client/Server Communication</td>
<td>HttpServletRequest and HttpServletResponse objects</td>
<td>N/A</td>
</tr>
<tr>
<td>Client/Server Communication</td>
<td>Request Decryption</td>
<td>HttpServletRequest and HttpServletResponse objects</td>
<td>N/A</td>
</tr>
<tr>
<td>Client/Server Communication</td>
<td>Response Decryption</td>
<td>Web page or data to be decrypted</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 4-3: Client/Server Communication Module Interfaces

External Data Dependencies

- JQuery library version 1.x
- JavaScript library version 1.x
- Android SDK API level 11 to 19
- Java SDK 1.7.x.
- Java EE
- Java.net.*
- Java.net.ssl.*

Internal Data Dependencies

N/A
Pseudocode

// Load CAs from an InputStream
// (could be from a resource or ByteArrayInputStream or ...)
CertificateFactory cf = CertificateFactory.getInstance("X.509");
InputStream caInput = new BufferedInputStream(new FileInputStream("site_certification.crt"));
Certificate ca;
try {
    ca = cf.generateCertificate(caInput);
    System.out.println("ca=" + ((X509Certificate) ca).getSubjectDN());
} finally {
    caInput.close();
}

// Create a KeyStore containing our trusted CAs
String keyStoreType = KeyStore.getDefaultType();
KeyStore keyStore = KeyStore.getInstance(keyStoreType);
keyStore.load(null, null);
keyStore.setCertificateEntry("ca", ca);

// Create a TrustManager that trusts the CAs in our KeyStore
String tmfAlgorithm = TrustManagerFactory.getDefaultAlgorithm();
TrustManagerFactory tmf = TrustManagerFactory.getInstance(tmfAlgorithm);
tmf.init(keyStore);

// Create an SSLContext that uses our TrustManager
SSLContext context = SSLContext.getInstance("TLS");
context.init(null, tmf.getTrustManagers(), null);

// Tell the URLConnection to use a SocketFactory from our SSLContext
URL url = new URL("https://amphiscope.com/test");
HttpsURLConnection urlConnection = (HttpsURLConnection)url.openConnection();
urlConnection.setSSLSocketFactory(context.getSocketFactory());
InputStream in = urlConnection.getInputStream();

// do stuff with InputStream
4.3 Server Security

The Server Security subsystem is functionally similar to the Client Security Subsystem. It will also be responsible for handling the security and integrity of the data that is being transmitted. Here the subsystem will encrypt data that is passed in from the Response Controller subsystem in the Business Logic layer and decrypt data received from the Network Transport layer to pass to the System Controller Subsystem.

4.3.1 Request Decryption

![Diagram of Request Decryption Module]

**Figure 4-4: Request Decryption Module**

**Prologue**

This module decrypts the encrypted payload within HTTP requests received from clients. The encryption method is expected to be known or derived from the encrypted data’s packaging. This module is expected to be called after the HTTP request is received in order to be able to decode the received data. This will be handled automatically when sending through HTTPS.

**Interfaces**

<table>
<thead>
<tr>
<th>Source</th>
<th>Sink</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client/Server Communication</td>
<td>Request Decryption</td>
<td>HttpServletRequest and HttpServletResponse objects</td>
<td>N/A</td>
</tr>
<tr>
<td>Request Decryption</td>
<td>Permission Authentication</td>
<td>HttpServletRequest and HttpServletResponse objects</td>
<td>N/A</td>
</tr>
</tbody>
</table>

*Table 4-4: Request Decryption Module Interfaces*
External Data Dependencies

- Java SDK 1.7.x.
- Java EE

Internal Data Dependencies

N/A

Pseudocode

N/A
4.3.2 Permission Authentication

Figure 4-5: Permission Authentication Module

Prologue

This module ensures that users have permission to send a certain type of request before the request is processed and a response generated. This will provide another level of security to the system, so that if the user somehow was able to send a request that they should not have had access to, they will still be stopped by this module before anything happens.

Interfaces

<table>
<thead>
<tr>
<th>Source</th>
<th>Sink</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request Decryption</td>
<td>Permission Authentication</td>
<td>HttpServletRequest and HttpServletResponse</td>
<td>N/A</td>
</tr>
<tr>
<td>Permission Authentication</td>
<td>Request Delegation</td>
<td>HttpServletRequest and HttpServletResponse (if valid)</td>
<td>N/A</td>
</tr>
<tr>
<td>Permission Authentication</td>
<td>Response Encryption</td>
<td>HttpServletRequest and HttpServletResponse (with error page, if invalid)</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 4-5: Permission Authentication Module Interfaces
External Data Dependencies:

- Java SDK 1.7.x.
- Java EE
- The XML containing the users and their permissions

Internal Data Dependencies:

N/A

Pseudocode

```java
// returns true if user is allowed to access a certain URL, which is obtained through the
// HttpServletRequest object
public boolean authenticate(…, HttpServletRequest request, user_credentials)
    throws RuntimeException // rethrows exceptions for process control
{
    allowed = false;
    role = request.getUserRole();
    if (role == DOCTOR){
        allowed = checkAuthList(doctor_permissions);
    }
    else if (role == ADMIN){
        allowed = checkAuthList(admin_permissions);
    }
    else if (role == PATIENT){
        allowed = checkAuthList(patient_permissions);
    }
    else{
        allowed = false;
    }

    if(allowed == false){
        //Return error page
        return allowed;
    }
```

### 4.3.3 Response Encryption

![Response Encryption Module Diagram](image)

**Figure 4-6: Response Encryption Module**

#### Prologue

This module encrypts the payload of the server’s response to a client’s HTTP request before forwarding it to the Transport subsystem to send over the network. Symmetric encryption with a strength of AES-128 or greater will be used. This will be handled automatically when sending through HTTPS.

#### Interfacing

<table>
<thead>
<tr>
<th>Source</th>
<th>Sink</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Permission Authentication</td>
<td>Response Encryption</td>
<td>HTTPServletRequest and HTTPServletResponse objects (error page)</td>
<td>N/A</td>
</tr>
<tr>
<td>Data Preparer</td>
<td>Response Encryption</td>
<td>HTTPServletRequest and HTTPServletResponse objects (data, no page)</td>
<td>N/A</td>
</tr>
<tr>
<td>Page Generator</td>
<td>Response Encryption</td>
<td>HTTPServletRequest and HTTPServletResponse objects (response page)</td>
<td>N/A</td>
</tr>
<tr>
<td>Response Encryption</td>
<td>Client/Server Communication</td>
<td>HTTPServletRequest and HTTPServletResponse objects</td>
<td>N/A</td>
</tr>
</tbody>
</table>

*Table 4-6: Response Encryption Module Interfaces*
External Data Dependencies

- Java SDK 1.7.x.
- Java EE

Internal Data Dependencies

N/A

Pseudocode

N/A
5. Business Logic Layer

The Business Logic Layer is meant to handle the major processing that is done by the server. Requests that make it past the Server Security subsystem of the Transport layer (the gateway to the server) will then be passed to this layer, where the web server will begin processing the request and determining how to respond to the client. The Business Logic Layer is composed of three subsystems: the System Controller (who handles the delegating of requests for the rest of the server), the Analysis subsystem (who looks at the data passed to it and makes conclusions and generates information based off of these analyses), and the Response Controller (who generates the page or readies the data that will be sent to the client).

The Business Logic Layer will be designed in such a way that it makes full use of some key design patterns. Some modules will be modeled after the Controller pattern, in that they will take a general request and break it down to a series of smaller requests, which will then be delegated to other objects. The Expert pattern will allow the system to store critical information and data into a specialized object, ensuring that no object has to handle a great deal of information by itself. The command pattern will allow objects of similar type to execute methods that seem the same, but may act very differently. These will all play a part in allowing the layer to make use of Java's polymorphic nature, which will be detailed below. This design will allow for an incredibly modular design, where adding or altering processing features will be as simple as creating new "request" objects similar to the ones that the core controllers use, and adding or manipulating an overloaded function in that core controller.

5.1 System Controller

The System Controller is designed to give the various core components of the server (data access, data analysis, response generation, and the network connection) one central hub to connect to, instead of having to connect all of these various parts together. The System Controller will take requests from both the client and from within the server, and will pass the data from the request on to the different subsystems, which in turn will help process what the client wants done and what the appropriate response would be. It is a rather simple job, but an incredibly important job, because this subsystem is the heart of the server's infrastructure.
5.1.1 Request Delegation

Prologue

This module is in charge of receiving requests from the client and from within the server and delegating the tasks associated with that request to the various other portions of the server. It is meant to provide a layer of abstraction for the modules it connects to, so that they can do their job without having to know how the other modules do theirs, making the modules less coupled and allowing for a separation of concerns. Client requests will be handled by Java Servlets, with every request having its own servlet to use. Server requests will be handled by one Java object (ServerRequestController) that will contain several overloaded static methods that will provide a layer of abstraction between the various portions of the server.
Interfaces

<table>
<thead>
<tr>
<th>Source</th>
<th>Sink</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Permission Authorization Module</td>
<td>Request Delegation</td>
<td>HttpServletRequest and HttpServletRequestResponse objects</td>
<td>N/A</td>
</tr>
<tr>
<td>Request Delegation</td>
<td>Data Controller</td>
<td>StorageDataPool Object</td>
<td>DataResponse object</td>
</tr>
<tr>
<td>Data Controller</td>
<td>Request Delegation</td>
<td>StorageDataPool object</td>
<td>Updated StorageDataPool object</td>
</tr>
<tr>
<td>Request Delegation</td>
<td>Analysis Controller</td>
<td>AnalysisDataPool object</td>
<td>AnalysisResponse object</td>
</tr>
<tr>
<td>Analysis Controller</td>
<td>Request Delegation</td>
<td>AnalysisDataPool object</td>
<td>Updated AnalysisDataPool object</td>
</tr>
<tr>
<td>Request Delegation</td>
<td>Data Preparer</td>
<td>HttpServletRequest, HttpServletRequestResponse, DataResponse, and AnalysisResponse objects</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 5-1: Request Delegation Module Interfaces

External Data Dependencies

- Java SDK 1.7.x
- Java EE
- HttpServletRequest, HttpServletResponse, ServerRequest and Web.xml

Internal Data Dependencies

- @WebServlet annotation
Pseudocode:

/*
Example of a servlet that will make up part of the Request Delegator. This
one is designed to handle the advice request.
*/
@WebServlet("/advice") //Linked to the /advice URL
public class ChartServ extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException,
            IOException {
        StorageDataPool dreq = new AdviceStorageDataPool(request, response);
        DataResponse dresp = new AdviceDataResponse();
        dresp.exec(dreq);
        AnalysisDataPool areq = new AdviceAnalysisDataPool(dresp);
        AnalysisResponse aresp = new AdviceAnalysisResponse();
        aresp.exec(areq);
        ResponsePreparer.sendResponse(dresp, aresp, request, response);
    }
}

/*
A barebones example of the ServerRequestDelegator, with
two overloaded methods. This will be expanded as the
development becomes more clear. The first one handles a
request from the Analysis Controller to get information about
a patient's medication from the DataController. Note how it
translates the AnalysisDataPool into a StorageDataPool, so
that the data flows remain consistent.
*/
public class ServerRequestDelegator {
    public static AdviceAnalysisDataPool delegate(AdviceAnalysisDataPool aadp) {
        DataResponse mdre = new MedicalDataResponse();
        StorageDataPool msdp = new MedicineStorageDataPool(aadp);
        mdre.exec(msdp);
        aadp.update(mdre);
        return aadp;
    }

    public static OtherAnalysisDataPool delegate(OtherAnalysisDataPool oadp) {
        //run code
    }
}
5.2 Data Analysis

The Data Analysis is meant to do the data processing and calculations that the system requires in order to generate a response for the user. The analyses performed by this subsystem range from simple checks that require a couple of inputs, to complex algorithms that require many steps and lots of data (based on previous blood pressure results, does the latest result show a concerning spike? If so, are there recent changes to the patient's medication, diet, or exercise patterns have could have influenced this? If not, does the result show the patient's blood pressure to be trending towards a healthier level, is it heading the opposite way, or has it remained relatively the same?). This subsystem is the calculator of the system, and its only job is to take in data and report back answers.

5.2.1 Analysis Controller

![Analysis Controller Module](image)

Prologue

The Analysis Controller is in charge of handling the steps and processes of a given analysis or algorithm, and storing the resulting data. It determines what formulas and algorithms to run based on the analysis requested and the results of other calculations and builds an object from the Analysis Command module that handles the actual calculation, which allows for separation between the steps of an algorithm and the calculations for an algorithm. The Analysis Controller module is made up of a collection of Java objects that implement the Analysis Controller interface, each object encapsulating one analysis to execute.
Interfaces

<table>
<thead>
<tr>
<th>Source</th>
<th>Sink</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request Delegation</td>
<td>Analysis Controller</td>
<td>AnalysisDataPool object</td>
<td>AnalysisResponse object with result data.</td>
</tr>
<tr>
<td>Analysis Controller</td>
<td>Analysis Command</td>
<td>AnalysisDataPool object</td>
<td>AnalysisCommand object with a set string, boolean, int, or float response attribute.</td>
</tr>
<tr>
<td>Analysis Controller</td>
<td>Request Delegation</td>
<td>Same AnalysisDataPool object</td>
<td>Same AnalysisDataPool object, now with more data.</td>
</tr>
</tbody>
</table>

Table 5- 2: Analysis Controller Module Interfaces

External Data Dependencies

- Java SDK 1.7.x
- AnalysisDataPool Object (from Request Delegator)

Internal Data Descriptors

- AnalysisController Object made up of string, int, float, and Boolean fields.
Pseudocode

/*
A rough example of the Advice object, which handles the process of looking at a patient's data for trends, changes, and other things to produce advice for the patient on ways to get or stay healthy. Note that this only handles the general steps, meaning that changing the algorithm without affecting the individual calculations, or vice-versa, is easy.
The results are stored in the object, so other modules can pass it around without having to understand the internals.

This example looks for spikes in the latest blood pressure readings, as well as looks at the patient's general health in relation to others
*/
public class AdviceAnalysisResponse implements AnalysisResponse
{
    String spike;
    String against;

    @Override
    public void exec(AdviceAnalysisDataPool dp)
    {
        BloodPressureAverageCmd bpa = new BloodPressureAverageCmd(dp);
        bpa.exec();
        dp.setAverage(bpa.getAverage);
        BloodSpikeCmd bsc = new BloodSpikeCmd(dp);
        bsc.exec();
        if(bsc)
        {
            dp = ServerRequestDelegator.delegate(
                new MedicineHistoryAnalysisDataPool(
                    dp));
            MedSpikeCheckCmd msc = new MedSpikeCheckCmd(dp);
            msc.exec();
            spike = msc.getResult();
        }
        CheckAgainstAverageCmd caa = new CheckAgainstAverageCmd(dp);
        caa.exec();
        against = caa.getResult();
    }
}
5.2.2 Analysis Command

![Figure 5-3: Analysis Controller Module]

**Prologue**

The Analysis Command module is meant to do calculations, data assessments, and other functions that are needed to do a proper analysis on a given data set. The module will consist of a collection of Java objects that extend the AnalysisCommand interface. Each object will encapsulate a single calculation, so that they are very simple objects, and so that individual calculations can be easily altered without affecting others. To keep the objects to a standard, and to simplify the Analysis Controller's job, the constructors will all accept a full AnalysisDataPool object, instead of individual data elements.

**Interfaces**

| Source          | Sink                | Input                   | Output                                                |
|-----------------|---------------------|-------------------------|                                                      |
| Analysis Controller | Analysis Command     | AnalysisDataPool object | AnalysisCommand object with a set string, boolean, int, or float response attribute. |

*Table 5-3: Analysis Controller Module Interfaces*
External Data Dependencies

- Java SDK 1.7.x.
- AnalysisDataPool object (from the Analysis Controller)

Internal Data Dependencies

Each object in the module will have a result attribute, which will vary in type depending on the job the object is in charge of. The various objects will each hold different data that is needed to perform the calculation, which will be filled when the AnalysisRequest is received.

Pseudocode

```java
/*
An example of one of the Analysis Command objects that make up the AnalysisCommander.
This one simple calculates the patient's average systolic and diastolic levels,
and sets a string that holds both the values
*/
public class BloodPressureAverageCmd {
    private ArrayList<Float> lastFiveSys;
    private ArrayList<Float> lastFiveDia;
    private String result

    public BloodPressureAverageCmd(AdviceAnalysisDataPool dp)
    {
        lastFiveSys = dp.getSys();
        lastFiveDia = dp.getDia();
    }

    @Override
    public void exec()
    {
        float sysAv = //calculate average systolic reading
        float diaAv = //calculate average diastolic reading
        result = "" + sysAv + "," + "diaAv";
    }

    public String get()
    {
        return result;
    }
}
```
5.3 Response Controller

The Response Controller will be the final subsystem to work on the response before it is sent back to the client. The Response Controller will look at the results of the data and the analyses, find the correct web page, reformat any data that needs to be put into a particular format for the page, load the data back into the HTTPRequest object, and replace any JSTL, EL, and other server-side language commands on the page with correct data. Its purpose is simply to get the data ready to send back to client.

5.3.1 Data Preparer

![Data Preparer Module Diagram](image)

**Figure 5-4: Data Preparer Module**

**Prologue**

The Data Preparer is in charge of getting the response ready for transfer. It finds the web page or data to send, reformats any data that needs to be (like making Java objects into JSON objects), and loads the data needed by the page into the HttpRequest and HTTPResponse objects. The module is made up of a collection of Java objects that implement the ResponseGenerator interface, each object encapsulating the appropriate response to a given client request.
Interfaces

<table>
<thead>
<tr>
<th>Source</th>
<th>Sink</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request Delegation</td>
<td>Data Preparer</td>
<td>AdviceResponse, StorageResponse, HttpServletRequest, and HttpServletResponse objects</td>
<td>N/A</td>
</tr>
<tr>
<td>Data Preparer</td>
<td>Page Generator</td>
<td>HttpServletRequest and HttpServletResponse objects (for page generation)</td>
<td>N/A</td>
</tr>
<tr>
<td>Data Preparer</td>
<td>Server Response Encryption</td>
<td>Data requested by client (for pure data transfer).</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 5-4: Data Preparer Module Interfaces

External Data Dependencies

- Java SDK 1.7.x.
- GSON Library
- AdviceResponse, a StorageResponse, an HttpServletRequest, and an HttpServletResponse (from Request Delegator)

Internal Data Dependencies
N/A
Pseudocode

/*
The advice Response Generator, which pulls data from the
various objects, puts them into the request object, and
gets the page that the PageGenerator will generate.
*/

public class AdviceDataPreparer implements ResponseGenerator{
    public void exec(AdviceDataResponse dresp,
                     AdviceAnalysisResponse aresp,
                     HttpServletRequest request,
                     HttpServletResponse response)
    {
        String spike = aresp.getSpike();
        request.setParameter("isSpike", spike);
        String against = aresp.getSpike();
        request.setParameter("against", against);
        String doctor = dresp.getDoctor();
        request.setParameter("doctorName", doctor);

        request.getRequestDispatcher("/WEB-INF/pages/advice.jsp").forward(request, response);
    }
}

/*
The Chart Response Generator. This one uses the GSON library
to convert the object to a JSON format. It also passes data,
not a page, so it doesn't go through the PageGenerator, it
just passes straight to the Server Encryption module
*/

public class ChartDataPreparer implements ResponseGenerator {
    public void exec(AdviceDataResponse dresp,
                     AdviceAnalysisResponse aresp,
                     HttpServletRequest request,
                     HttpServletResponse response)
    {
        Gson gson = new Gson();
        String bloodObject = gson.toJson(dresp.getBloodObject);
        response.getWriter().write(bloodObject);
    }
}

public static FinalResponse prepareData(PatientResponseRequest areq){}
5.3.2 Page Generator

![Diagram](image.png)

**Figure 5-5: Page Generator Module**

**Prologue**

The Page Generator is the final stop for any response requiring a web page. Here, JSTL, EL, and any other server-side code that is placed in the page is translated into HTML, JavaScript, and CSS (depending on how the data is used) so that the client's browser can understand it. The team will not be responsible for the actual translation of this code, but these server-side web languages will play a big part in the system's web pages.

**Interfaces**

<table>
<thead>
<tr>
<th>Source</th>
<th>Sink</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Preparer</td>
<td>Page Generator</td>
<td>HttpServletRequest and HttpServletResponse</td>
<td>N/A</td>
</tr>
<tr>
<td>Page Generator</td>
<td>Server Encryption</td>
<td>Translated web page</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 5-5: Page Generator Module Interfaces

**External Data Dependencies**

- HttpServletRequest and HttpServletResponse (from Data Preparer Module)

**Internal Data Dependencies**

N/A
Pseudocode

<!--
Snippet of Javascript with EL code inside,
which will be replaced with JSON Objects
-->
setChart(${sys},${dia});

<!--
Code snippet wil JSTL and EL code that makes the
page dynamic. These will be replaced with HTML,
CSS, JavaScript, and so on.
-->
<c:choose>
 <c:when test="${sessionScope.user != null}">
  <c:if test="${notFound != null}">
   User not found in database <br>
  </c:if>
  <form action="/login" method="post">
   Username: <input name="user"/></input>
   <br>
   Password: <input name="pass"/></input>
   <br>
   <input type="submit" value="login"></input>
  </form>
 </c:when>
 <c:otherwise>
  ${sessionScope.user}
 </c:otherwise>
</c:choose>
6. Data Storage Layer

The data storage layer is responsible for allowing the system to communicate securely with an external database. This layer consists of a data distributor, database security, and database interface subsystems. The data distributor processes requests for data from the other layers. The data security subsystem encrypts and decrypts those requests so that they can securely move outside of the system. The interface will take those requests and translate them into talk that the database can understand.

6.1 Storage Control Subsystem

This is in charge of processing the requests that come in from the other layers asking for data. Receives a request data pool and breaks it down into database command objects based on what flags are set in the data pool. These command objects are then sent to the security subsystem. Once it receives the response from the database, it packages the new data into a data response object and sends it back up.

6.1.1 Data Controller

![Diagram of Data Controller Module]

Figure 6-1: Data Controller Module
Prologue
This module receives a StorageDataPool object, looks at the flags, and then breaks the data into individual database command objects. These commands are sent to the security module, when the response is received, the controller packages the data into a DataResponse object and sends it back.

<table>
<thead>
<tr>
<th>Source</th>
<th>Sink</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request Delegation</td>
<td>Data Controller</td>
<td>StorageDataPool Object</td>
<td>Data Response object</td>
</tr>
<tr>
<td>Data Controller</td>
<td>Cryptography</td>
<td>Database Command object</td>
<td>Updated Database Command object</td>
</tr>
<tr>
<td>Data Controller</td>
<td>Request Delegation</td>
<td>Same StorageDataPool object</td>
<td>Same StorageDataPool object, now with more data.</td>
</tr>
</tbody>
</table>

Table 6-1: Data Preparer Module Interfaces

External Data Dependencies

- Java SDK 1.7.x.
- StorageDataPool Object (from Request Delegator)

Internal Data Dependencies

- Various Boolean, int, float, and string fields, depending on the data being returned
Pseudocode

/*
Extremely simple example of a StorageResponse object
that gets a patient's last five blood pressure
results.
*/

public class AdviceStorageResponse implements StorageResponse {
    ArrayList<Float> lastFiveSys;
    ArrayList<Float> lastFiveDia;

    @Override
    public void exec(AdviceStorageDataPool dp) {
        getLastFiveSysCmd glfs = new getLastFiveSysCmd(dp);
        lastFiveSys = Crypto.cryptog(glfs).getResults;
        getLastFiveDiaCmd glfd = new getLastFiveSysCmd(dp);
        lastFiveDia = Crypto.cryptog(glfd).getResults;
    }
}
6.2 Data Security Subsystem

The Data Security Subsystem is in charge of database security. Any data that is passing in or out of the database has to go through security – data going into the database has to be encrypted and data that is retrieved must be decrypted so that it may be brought to the system.

6.2.1 Cryptography

![Cryptography Module Diagram]

Figure 6-2: Cryptography Module

Prologue
This module takes in the command objects and encrypts the data inside of them. The database interface subsystem executes the commands and returns the command object with the encrypted response. The cryptography module decrypts the data and passes it on.
Interfaces

<table>
<thead>
<tr>
<th>Source</th>
<th>Sink</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Controller</td>
<td>Cryptography</td>
<td>Database command object</td>
<td>Updated Database command object</td>
</tr>
<tr>
<td>Cryptography</td>
<td>DB Get /DB Update</td>
<td>Database command object w/encrypted data</td>
<td>Database Command object with encrypted result</td>
</tr>
</tbody>
</table>

Table 6-2: Cryptography Module Interfaces

External Data Dependencies

- Java SDK 1.7.x
- StorageDataPool Object

Internal Data Dependencies

N/A

Pseudocode

```java
public class Crypto {
    public static getLastFiveSysCmd cryptog(getLastFiveSysCmd gls) {
        gls.setUser = libraryCall.encrypt(gls.getUsername);
        gls.exec();
        gls.decrResult = libraryCall.decrypt(gls.getResult);
        return gls;
    }

    public static getLastFiveDiaCmd cryptog(getLastFiveDiaCmd gld) {
        gld.setUser = libraryCall.encrypt(gld.getUsername);
        gld.exec();
        gld.decrResult = libraryCall.decrypt(gld.getResult);
        return gld;
    }
}
```
6.3 Database Interface Subsystem

The Database Interface Subsystem is in charge of executing the requests to the database. It will be designed using the bridge pattern so that this subsystem can be easily changed to accommodate any type of database. For this implementation the team will be using SQL.

6.3.1 Database Get

![Diagram of Database Get Module]

**Figure 6- 3: Database Get Module**

**Prologue**

This module executes the commands that involve retrieving data from the database.

**Interfaces**

<table>
<thead>
<tr>
<th>Source</th>
<th>Sink</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cryptography</td>
<td>DB Get</td>
<td>Database command object w/encrypted data</td>
<td>Database Command object with encrypted result</td>
</tr>
<tr>
<td>DB Get</td>
<td>Database</td>
<td>SQL Statement</td>
<td>ResultSet object</td>
</tr>
</tbody>
</table>

*Table 6- 3: Database Get Module Interfaces*
External Data Dependencies

- Java SDK 1.7.x.
- Java EE
- StorageDataPool Object
- Libraries to connect to database

Internal Data Structure:

- A result field of varying type
- Various String, Boolean, Float, and Int fields based on any needed data

Pseudocode

```java
public class BloodPressureSysCmd {

    private String username;
    private ArrayList<Float> result;

    public BloodPressureAverageCmd(AdviceAnalysisDataPool dp) {
        username = dp.getUser();
    }

    @Override
    public void exec() {
        Database.connect;
        connection.exec("GET Sys FROM Patient WHERE" +
        "patientName = " + username + ",");
        //Parse result set to make arraylist out of
        //values
        result = bpSysArray;
    }

```
6.3.2 Database Update

This module executes the commands that involve updating (changing data or deleting data) in the database.

### Interfaces

<table>
<thead>
<tr>
<th>Source</th>
<th>Sink</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cryptography</td>
<td>DB Update</td>
<td>Database command object w/encrypted data</td>
<td>Database Command object w/encrypted result</td>
</tr>
<tr>
<td>DB Update</td>
<td>Database</td>
<td>SQL Statement</td>
<td>ResultSet object</td>
</tr>
</tbody>
</table>

Table 6- 4: Database Update Module Interfaces
External Data Dependencies

- Java SDK 1.7.x.
- Java EE
- StorageDataPool Object
- Libraries to connect to database

Internal Data Structure

- A String result for errors or confirmation
- Various string, Boolean, float, and int fields base on any needed data

Pseudocode

```java
public class BloodPressureSysCmd {

    private String username;
    private String password;
    private Float newBloodRead;
    private String result;

    public BloodPressureAverageCmd(AdviceAnalysisDataPool dp) {
        username = dp.getUser();
        password = dp.getPass();
        newBloodRead = dp.getNewReading();
    }

    @Override
    public void exec() {
        Database.connect;
        connection.exec("INSERT ..."); //Insert attributes into db;
        //Get response
        result = responseString;
    }
}
```
7. Quality Assurance

7.1 Module/Unit Testing

7.1.1 Client Layer

- The Browser Display module will be able to take input from the keyboard and display them on the html element.
- The Browser Display module will be able to recognize mouse actions and respond to each action upon any html element.
- The Browser Display module shall be able to recognize action events and execute proper animation based on the html.
- The Browser Display module will be able to receive the response data or page from the Response Decryption module after a requested has been completed.
- The Web Validation module will be able to take in html elements (form, button, etc) and validate the integrity of the input. Return a boolean for invalid or N/a for valid.
- The Chart Generation module will be able to create charts elements based on user data retrieved from the database, and insert it into the web page.
- The Application UI module will be able take input from an Android touch screen and display interaction on the display.
- The Application UI module will be able to respond to button clicks, check marks, scrolls, or text field according to the intent of the interaction.
- The Bluetooth Communication module will be able to instantiate and power on the Bluetooth adapter of the Android device, and be able to receive data streamed from the BPM device.
- The Data Packet Processing module will be able to extract the BP pressure data correctly, as displayed by the BPM display.
- The BPM Validation module will be able to catch errors from input, if and only if input has errors.
7.1.2 Transport Layer

- The Request Encryption module will be able to take in a HttpConnection object sent in from the Client Layer and encrypt the object and send it to the Client/Server Communication module.

- The Request Decryption module will be able to receive encrypted HttpServletRequest/HttpServletResponse objects decrypt and send it to the Permission Authentication module.

- The Client/Server Communication module will be able to receive both encrypted and decrypted request and response object and send them to both the Server Security modules and Client Security modules.

- The Response Decryption module will be able to receive and encrypted object(web page, data, etc) and decrypt it and send it to the Client Layer.

- The Response Encryption module will be able to receive encrypted HttpServletRequest/HttpServletResponse objects encrypt it and send it to the Client/Server Communication module.

- The Permission Authentication module will be able to receive the HttpServletRequest/HttpServletResponse objects validate the permission of the request and send it back to the Response Encryption module on and error else if it is valid send it to the Request Delegation Module.
7.1.3 Business Logic Layer

- Request Delegation - Each time a new servlet is created, the team will verify that the requests assigned to that servlet are reaching it, that the correct function is being run, that the servlet sends data to the other modules, and that a page is being sent back.

- Analysis Controller - Each time a new method is added to handle a new request type, the team will verify that the request is reaching the correct method, that the command objects are being created correctly, and that the response being generated by the method accurately reflects the data that was being passed in. Different data sets will be passed in to verify that all branches are covered. The AnalysisResponse object that corresponds with that method will be tested to ensure that it's attributes are easy to get and set, and that it works well with the command objects that it pulls from.

- Analysis Command - Each time a new Command object is added to the system, the team will verify that the object receives all the data it needs from the Analysis Factory's given request, that the calculation or analysis works correctly, and that it's result data does not cause any conflicts with the Response object being created. Depending on the command being executed, various values will be inserted to test all branches, ensure that special cases are handled appropriately, and so on.

- Data Preparer - Each time a new method is created to prepare a new page or piece of data, the team will verify that all of the data needed by the page is being loaded into it appropriately, that the data being formatted is actually in the correct format, and that the correct page is actually being loaded. In cases where different pages can be loaded, it will be fed data so that each should be pulled up at one point.

- Page Generator - Every time a new page is added that has server-side code elements in it, the team will ensure that the data is being translated correctly so that the browser can use it, by viewing the areas that should have been affected by the changes in the data being fed.
7.1.4 Data Storage Layer

- Data Controller – Data pools will be successfully broken into command objects. Results shall be properly packaged into a response object.

- Cryptography – Data inside of the command objects will be successfully encrypted and will be tested that they function properly inside of the database. Encrypted data shall be properly decrypted so the data can be used by the system.

- Database Get – The requests shall retrieve the proper data.

- Database Update – The requests shall perform the correct operations on the database.
7.2 Component Testing

7.2.1 Web Testing

- System will accept user keyboard input and send it to the web validation module. And receive a message of validation
- System can retrieve user reading data and display it in a chart of the web page.

7.2.2 BPM Testing

- User can log into mobile application, take reading on BPM, accept readings from BPM, and send readings to the database.

7.2.3 Transport Testing

- System shall be able to encrypt both server and client side packets, and transport them between the client and business logic layers.
- System can verify user request via profile permissions.

7.2.4 Business Logic Testing

- The system shall take in different types of client requests with different sets of data, and respond with the correct page containing the correct information

7.2.5 Storage Testing

- System shall be able to receive data requests. Translat into database code, then encrypt the commands and send them to the database.
7.3 Integration Testing

- System shall be able to receive encrypted data from the database. Decrypt the data of the fulfilled request.

7.4 System Verification Testing

- The system verification testing shall be used to verify that all high-level requirements defined in the SRS have been properly implemented
### 7.5 Test Cases

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Expected Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>User shall input information into Registration page and hits enters.</td>
<td>User Profile is created.</td>
</tr>
<tr>
<td>User shall input login credentials into login page/screen.</td>
<td>User is login into their account and sent to the profile page.</td>
</tr>
<tr>
<td>User shall click on a chart and add an Annotation.</td>
<td>User sees the annotation on the chart.</td>
</tr>
<tr>
<td>User shall click on the Advise button.</td>
<td>User will be sent to advice page and view advice for their reading.</td>
</tr>
<tr>
<td>User shall set an Alert.</td>
<td>Doctor will receive an alert message.</td>
</tr>
<tr>
<td>User shall take a reading and hit send button the application.</td>
<td>System shall store the blood pressure reading in the database.</td>
</tr>
</tbody>
</table>

**Table 7-1: Test Cases**
8. Requirements Mapping

The tables below show which modules help address which of the project’s key requirements. This helps to show both which modules play a bigger role in the system (though each module has its purpose) and which requirements are more complex. This can help to evaluate which modules and which requirements to focus on first, both based on their relative complexity and based on which requirements are seen as more important (as stated in the SRS).

Based on the tables below, you can see that more of our requirements pertain to the website than to the BPM, due to the wide variety of features being provided by the web service. It also shows that there are some modules that help address just about every requirement listed. Just about every requirement discusses a function that needs to connect a client to the server, needs to be able to reach the various inner components of the server, needs to store or access data in some way, and needs to get a response back. The project relies on a lot of input, which means the system has to validate what is sent in, and because of the sensitive data being collected and used by the system, certain security modules become incredibly important at all times. In addition, the tables clearly show how the modules work together to address a requirement, instead of having single modules handle certain requirements.
## 8.1 Client Layer

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3.5</td>
<td>User Authentication</td>
<td>X</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.7</td>
<td>View Results</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.8</td>
<td>Multiple Readings</td>
<td>x</td>
<td>X</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.9</td>
<td>Result Transmission and Storage</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>3.1</td>
<td>Web Service</td>
<td>X</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.11</td>
<td>Account Types</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.12</td>
<td>Doctor Account Creation</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.13</td>
<td>Patient Account Creation</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.14</td>
<td>Patient's Doctor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.15</td>
<td>Doctor Limit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.16</td>
<td>Doctor Editing</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.17</td>
<td>Patient Editing</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.18</td>
<td>View Charts and Graphs</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.19</td>
<td>Annotations</td>
<td>X</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.2</td>
<td>Patient Viewing Profile</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.21</td>
<td>Create Alerts</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.22</td>
<td>View Alerts</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.23</td>
<td>Messaging System</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.24</td>
<td>Multiple Users</td>
<td>X</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>3.25</td>
<td>Login to Access Data</td>
<td>X</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.26</td>
<td>HIPAA</td>
<td>X</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.27</td>
<td>Advice System</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.28</td>
<td>Manual Upload</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.1</td>
<td>Data Transport Encryption</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.2</td>
<td>Authorization</td>
<td>X</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.3</td>
<td>Integrity</td>
<td>X</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.4</td>
<td>Storage Encryption</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.5</td>
<td>Disposal Guarantee</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.6</td>
<td>WLAN Connection Security</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.7</td>
<td>Data Backup</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 8-1: Client Layer Module Requirements Mapping
### 8.2 Transport Layer

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>3.5</strong></td>
<td>User Authentication</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>3.7</strong></td>
<td>View Results</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>3.8</strong></td>
<td>Multiple Readings</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>3.9</strong></td>
<td>Result Transmission and Storage</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>3.1</strong></td>
<td>Web Service</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>3.11</strong></td>
<td>Account Types</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>3.12</strong></td>
<td>Doctor Account Creation</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>3.13</strong></td>
<td>Patient Account Creation</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>3.14</strong></td>
<td>Patient's Doctor</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>3.15</strong></td>
<td>Doctor Limit</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>3.16</strong></td>
<td>Doctor Editing</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>3.17</strong></td>
<td>Patient Editing</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>3.18</strong></td>
<td>View Charts and Graphs</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>3.19</strong></td>
<td>Annotations</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>3.2</strong></td>
<td>Patient Viewing Profile</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>3.21</strong></td>
<td>Create Alerts</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>3.22</strong></td>
<td>View Alerts</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>3.23</strong></td>
<td>Messaging System</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>3.24</strong></td>
<td>Multiple Users</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>3.25</strong></td>
<td>Login to Access Data</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>3.26</strong></td>
<td>HIPAA</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>3.27</strong></td>
<td>Advice System</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>3.28</strong></td>
<td>Manual Upload</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>8.1</strong></td>
<td>Data Transport Encryption</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>8.2</strong></td>
<td>Authorization</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>8.3</strong></td>
<td>Integrity</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>8.4</strong></td>
<td>Storage Encryption</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>8.5</strong></td>
<td>Disposal Guarantee</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>8.6</strong></td>
<td>WLAN Connection Security</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>8.7</strong></td>
<td>Data Backup</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Table 8-2: Transport Layer Module Requirement Mapping
### 8.3 Business Logic Layer

<table>
<thead>
<tr>
<th>Req. No.</th>
<th>Requirement Name</th>
<th>Request Delegation</th>
<th>Analysis Controller</th>
<th>Analysis Command</th>
<th>Data Preparer</th>
<th>Page Generator</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.5</td>
<td>User Authentication</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>3.7</td>
<td>View Results</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.8</td>
<td>Multiple Readings</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.9</td>
<td>Result Transmission and Storage</td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>3.1</td>
<td>Web Service</td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>3.11</td>
<td>Account Types</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.12</td>
<td>Doctor Account Creation</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>3.13</td>
<td>Patient Account Creation</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>3.14</td>
<td>Patient's Doctor</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.15</td>
<td>Doctor Limit</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.16</td>
<td>Doctor Editing</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>3.17</td>
<td>Patient Editing</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>3.18</td>
<td>View Charts and Graphs</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>3.19</td>
<td>Annotations</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>3.2</td>
<td>Patient Viewing Profile</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>3.21</td>
<td>Create Alerts</td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>3.22</td>
<td>View Alerts</td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>3.23</td>
<td>Messaging System</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>3.24</td>
<td>Multiple Users</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.25</td>
<td>Login to Access Data</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.26</td>
<td>HIPAA</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>3.27</td>
<td>Advice System</td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>3.28</td>
<td>Manual Upload</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.1</td>
<td>Data Transport Encryption</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.2</td>
<td>Authorization</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.3</td>
<td>Integrity</td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>8.4</td>
<td>Storage Encryption</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.5</td>
<td>Disposal Guarantee</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.6</td>
<td>WLAN Connection Security</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.7</td>
<td>Data Backup</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 8-3: Business Logic Layer Module Requirement Mapping**
### 8.4 Data Storage Layer

<table>
<thead>
<tr>
<th>Req. No.</th>
<th>Requirement Name</th>
<th>Data Controller</th>
<th>Cryptography</th>
<th>DB Get</th>
<th>DB Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.5</td>
<td>User Authentication</td>
<td></td>
<td>x</td>
<td>x</td>
<td>X</td>
</tr>
<tr>
<td>3.7</td>
<td>View Results</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.8</td>
<td>Multiple Readings</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.9</td>
<td>Result Transmission and Storage</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>3.1</td>
<td>Web Service</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.11</td>
<td>Account Types</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.12</td>
<td>Doctor Account Creation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.18</td>
<td>View Charts and Graphs</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.19</td>
<td>Annotations</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.2</td>
<td>Patient Viewing Profile</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.21</td>
<td>Create Alerts</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.22</td>
<td>View Alerts</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.23</td>
<td>Messaging System</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.24</td>
<td>Multiple Users</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.25</td>
<td>Login to Access Data</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.26</td>
<td>HIPAA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.27</td>
<td>Advice System</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.28</td>
<td>Manual Upload</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.1</td>
<td>Data Transport Encryption</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.2</td>
<td>Authorization</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.3</td>
<td>Integrity</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.4</td>
<td>Storage Encryption</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.5</td>
<td>Disposal Guarantee</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.6</td>
<td>WLAN Connection Security</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.7</td>
<td>Data Backup</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 8-4: Data Storage Module Requirement Mapping
9. Acceptance Plan

This section outlines the plan that will be followed that will ensure that the wireless blood pressure monitor project will be accepted by the sponsor and by the customer. The plan includes the criteria that will be met, the acceptance testing, and how the system should be packaged and installed.

9.1 Acceptance Criteria

The Blood Pressure Monitor must meet the following criteria.

- Personal Data is secure
- The blood pressure reading is accurate
- The system is user friendly
- The system can authenticate user logins
- The system is safe and noninvasive for the user.
- The system gives proper advice
- Account creation works and can handle errors
- Charts are properly generated
- Alert system works correctly
- Data transfers properly.

9.2 Acceptance Testing

The Acceptances Testing will ensure that each of the acceptance criteria is correctly met. These criteria are the high level requirements stated in the System Requirement Specification. All test plan shall be documented and reported in the System Test Plan.
9.3 Package and Installment

The blood pressure monitoring system will come with preassembled parts and will be ready to use. User will be able to easily configure the device and start taking readings. All readings will be automatically sent to the website when Wi-Fi is enabled or available. Users will have an account created for them by their primary doctor to view and log past readings. The blood pressure monitor will come package with the following.

- Blood Pressure cuff
- Blood Pressure Monitor
- Android Application (available via online download)
- Web Services
- Connection Cable
- User Manual
10. Appendix

10.1 Blood Pressure Monitor
The Blood pressure monitor the user uses to take blood pressure readings and that interfaces with the Bluetooth Communication module is external to the system. Specifically, the BPM that the system uses and will be tested with is the HPL-108 wrist monitor manufactured by MyTech. It is a Bluetooth enabled monitor that captures user input from five on-device buttons. The specific task of each button are: power on/off device, turn Bluetooth on/off, set time/date, take BP reading, delete single/all blood pressure records, send BP reading through Bluetooth, or scroll through existing records. From the device the user will be able to see there systolic and diastolic blood pressures, as well as see their heart rate. The monitor will transmit the users taken blood pressure reading through its Bluetooth adapter and will be captured by the Bluetooth Communication module to be processed further.

10.2 Server-Side Objects
Due to the complex nature of our server setup, which will make use of various design patterns and general good coding practices, many different types of objects will be created and used during the execution of a given request handling. Below are high-level descriptions of these objects, to more clearly show how these objects will be used.

10.2.1 Servlet
A special set of objects that are meant to accept an HTTP request from a client. They are tied to a URL by the web.xml file that is set up when creating the server. They receive two objects from the client, an HttpServletRequest object and an HttpServletResponse object. These objects hold information about the request, including data needed to handle the request.

10.2.2 ServerRequestDelegator
An object holding several static, overloaded methods that is meant to work in the same way as a servlet, but which will handle server requests, instead of client requests. It will receive a DataPool object that acts in a similar way to the HTTP objects, and which will help the ServerRequestDelegator understand which method to use and how to process it.
10.2.3 DataPool

DataPool objects are meant to provide a uniform way to pass data between different parts of the server. The Analysis Controller and Data Controller could be called either because they are needed by client, or because they are needed by another part of the server. The object that the Request Delegation receives in order to make this request will differ depending on where the request came from. In order to avoid having the components have to know how to handle a variety of different objects, the DataPool object was created. It will take in these different types of objects and pull the data from them to store, so that the data is now stored in the DataPool object and can be passed around. This simplifies the jobs of some of the modules, as they don't have to worry about handling different types of objects, instead handling only one specific object.

10.2.4 Response

Objects that are meant to handle a high-level plan to execute a given algorithm. These come in two different types, a DataResponse object and an AnalysisResponse object (which in turn have several children). They know what to do to achieve something (such as storing user data or looking for problems in a set of blood pressure readings), but not exactly how to do them. The how will be handled by the Command objects. This object will delegate the individual tasks to the Command objects, and will know how to handle the result (do I store this as more input for future Command objects, do I use this as a Boolean to check for certain things, or is this part of the final result?). The Response object will also hold the final result of the algorithm, and thus can be passed around with the stored result for later use. The Response objects need to receive a DataPool in order to actually do their jobs, as the DataPool holds the data that the Response objects need.

10.2.5 Command

Objects that handle individual steps to a given algorithm. This includes calculating averages, updating individual tables, and so on. The Command objects take in a DataPool in order to do their jobs, and execute their method to generate a result, which is then stored internally for the Response object to retrieve when needed. A series of these Command objects make up an algorithm.
10.2.6 Cryptography

One object with several static, overloaded methods, similar to the ServerRequestDelegator. This intercepts the DatabaseCommand objects before they execute, and encrypts any data that is going to be stored and needs encryption. It also decrypts any data coming out of the database, so that the system can process it properly. This is to make sure that the database is secure and is not allowing data to go into the database without a proper level of encryption being placed on it. Not all data needs to be encrypted or decrypted, in which case the DatabaseCommand objects will simply execute and the finished object will be sent back to the Response object. In order to make things uniform, however, and to make sure that the code maintains the same general structure regardless of situation, all DatabaseCommand objects will go through the Cryptography object before they are executed.