UAH Fit Vault Software Design Specification

CPE 656/658 Software Studio

Timothy R. Wilkins

Whit J. Sisulak

Glen L. Riden

James J. Duggan IV

12/6/2015

Revision History

Revision #	**Revision Date**	Description of Change	Author
0.1	10/12/15	Initial Draft	J. Duggan W. Sisulak
0.2	10/25/15	Updated scope and architecture. Changed title and file name.	J. Duggan W. Sisulak
0.3	11/07/15	Added application framework section. Created detail design section.	W. Sisulak
0.4	11/13/15	Update scope and architecture overview.	J. Duggan
0.5	11/14/15	Added detailed design for SelectDataController and SelectActivityController. Added detailed design for Activity Model and ActivityType enumeration.	W. Sisulak
0.6	11/15/15	Added several new enumerations. Added detailed design for all the various system user objects. Added experiment object design. Created structure and relationship section with description of data ingestion feature.	J. Duggan
0.7	11/22/15	Added database schema and associated detailed designs of the various objects shown in the schema. Added glossary.	J. Duggan W. Sisulak
0.8	11/22/15	Added account controller	T. Wilkins
0.9	11/22/15	Added Administrator Tools features and corresponding detailed design of the classes involved.	W. Sisulak J. Duggan
0.10	11/29/15	Added Physician detailed design and feature descriptions. Added feature description for account management.	G. Riden T. Wilkins
0.11	11/29/15	Added patient delete notes to admin tools feature section 2.2.2. Added experiment detailed design and feature description.	J. Duggan

0.12	11/29/15	Updated database schema to add new data type field to PatientData table. Removed unnecessary timestamp fields from tables. Added initial version of data repository class.	J. Duggan
0.13	12/2/15	Added patient feature information and detailed design.	W. Sisulak
0.14	12/2/15	Updated activity list per customer request. Added export feature and added new functions for handling exports to the patient, physician, and experiment controllers.	J. Duggan G. Riden
0.15	12/2/15	Added data dictionary for user inputted information.	W. Sisulak
0.16	12/4/15	Updated SQL repository class with remaining data functions.	J. Duggan
0.17	12/6/15	Adding references to prototypes.	G. Riden
0.18	12/6/15	Added missing functions to the admin controller. Added traceability appendix.	J. Duggan

Table of Contents

Software Design Specification

1 Introduction

1.1 Purpose

The purpose of this document is to provide a detailed design of the UAH Fit Vault software projects. This document should be used as a reference for the software system architecture and detailed design descriptions of the system components. The intended audience for this document includes system developers, testers, customers, and any other stakeholders.

1.1 Scope

The UAH Fit Vault software package will be a web application that will accept medical data from users and display the data in a meaningful way. There are two major components to this software. The first is the data collection tool that is used by the users to upload their medical data that is recorded by one of the supported wearable medical devices. There are three different medical devices supported for this project that record various types of data. The data provided by these devices consists of different file formats, and the data is different from device to device. The software will have to determine the contents of each file and how to process them. The software needs to able to take in files that a user has downloaded from their medical devices, process those files, and store the data in a database. The software should have the ability to process multiple files at a time as well as individual files and allow for an activity to be assigned to them by date and time.

The other major component of the web application is the data analysis tools used to analyze the data that is captured from the data collection tool mentioned above. The software needs to perform data analysis over different intervals of time such as one week, one month, etc. There will need to be some way to manage user access to the various medical data that has been inserted into the database that this software will access. Below are some proposed data analysis ideas that can be incorporated into the project.

- Simple Moving Average
- Data correlation discovery between the multiple devices.
- Simply display data that was uploaded to the customer in a graphical format.
- Calculate the user’s activity.

The data analysis possibilities will likely not fully be realized until the project team understands the different types of data that are available. Also, there will need to be collaboration with the customer for additions or changes to the data measurements provided by this software. The web application will have to have different levels of user access which will be defined later in this document.

1.2 Definitions, Acronyms, and Abbreviations

Account	**Roles and permissions assigned to a user and stored electronically**
Activity	Action performed by patient (ex. running, walking, sleeping)
ASP.NET	Microsoft web based software architecture
C#	C “Sharp” Microsoft .NET programming language
Controller	Middle layer logic, interaction between the view and data model
csv	Comma-separated values
Customer	A person or group requesting the software to be built
dat	Data file
Data	Patient information
Database	Storage medium for patient data
Database Service	Database functionality accessible via the internet
ECG	Electro Cardio Gram
Experiment	Combining of data to generate a report based on input criteria
Experiment Admin	User of the system, able to run experiments on patient data
GSR	Galovanic Skin Resistance
GUI	Graphical User Interface
HTTP(S)	Hypertext Transfer Protocol (Secure)
HRV	Heart Rate Variability
IEEE	Institute of Electrical and Electronics Engineers
Interface	Medium in which a user interacts with the system
Medical Device	A device worn by a patient to collect medical data
Model	Data base representation
MVC	Model View Controller
Patient	User of the system, under the care of a physician
Physician	User of the system, treats patients
Query	Request for information based on parameters
Query Builder	Aids in building queries
RSSI	Bluetooth signal strength
SDD	Software Design Document/Software Design Specification
SQL	Structured Query Language
Stakeholder	Anyone who has a stake in the outcome of the project
System Admin	User of the system, able to manage other user accounts
System Developer	An individual involved in the design and construction of a system
Tester	An individual involved in the testing of a system
UAH	The University of Alabama in Huntsville
User	Operator of the system
View	User presentation GUI
Web Application	An application hosted on a server and accessible via the internet most commonly through a web browser
Web Based	Accessible via the internet utilizing HTTP/HTTPS
Web Browser	An interface to view web applications over the internet (example: Firefox,

	Internet Explorer)
Web Service	A software service accessible over the internet

1.3 References

IEEE Std 1016-1998, IEEE Standard for Software Design Specification

Definitions for New Race and Ethnicity Categories (https://nces.ed.gov/ipeds/reic/definitions.asp)

UAH Fit Vault Sofware Requirements.docx

1.4 Overview

The remainder of this design specification document addresses the software system architecture, detailed design information for the various system components, and the database schema design. Each major section will be broken into two pieces each detailing the design criteria for the two pieces of software the make the UAH Fit Vault.

1.5 Application Framework

The application is designed to work with the Microsoft ASP.NET MVC web application framework. This framework uses the Microsoft .NET Framework and Common Language Runtime languages (specifically C#)

Using ASP.NET MVC helps guide the design process. The framework takes ideas from the traditional design patterns of Model-View-Controllers and bootstraps a web application allowing for simple design and allows us to leverage a lot of hard work done by people with far more expertise in web application framework development.

2 System Architecture Description

2.1 Overview of Components

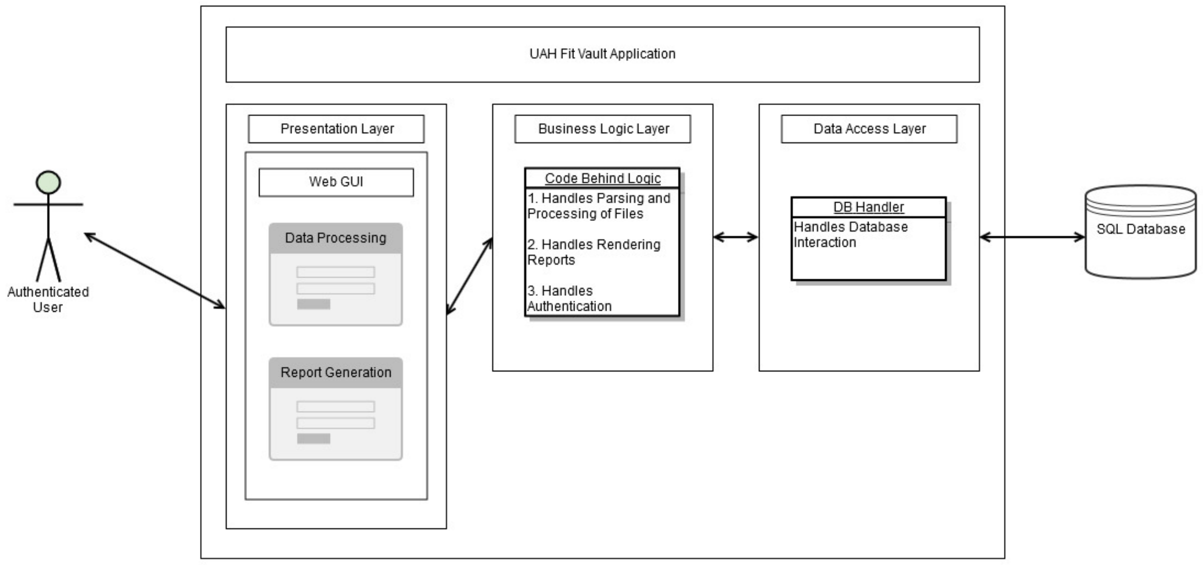


Figure 2.1: Architectural Overview

The UAH Fit Vault application is a Web application that is comprised of and encompasses the presentation, business logic, and data access layers. The system interacts with the end user via a web interface and with an external SQL database hosted on a remote server. It is assumed that the user is authenticated to use the application.

- The presentation layer consists of a web interface for data processing and reporting. The various views and controllers used in this system will exist in the presentation layer. The views function as the user interface that gives the user a way to select the data files they wish to upload as well as a means for assigning activity information to the data. Other views will be used for account management and for displaying reports to the different users. The controllers process the data that is submitted to the system through the views from the user. The controllers are also responsible for determining the data that is used by the views to display the request information to the user.

- The business logic layer will consist of all the back end code comprised of classes to handle the parsing and processing of files as well as provide the logic to render metrics and reports. Business logic functions are those that are used to edit the data elements in the model classes in the system.

- The data access layer is structured using a singular repository object that provides a uniform means for persisting and retrieving data models from the database. This is intentionally done in order to abstract away other objects ability to communicate directly with the data source. The repository contains the necessary create, read, update, and delete functions necessary to change objects and processes requests from service objects. The data access layer will consist of all the back end code needed to interact with an external SQL database utilizing the data access library of choice. This system will use Entity Framework to manage the data access layer.

2.2 Structure and Relationships

The following section is intended to describe the major features of the UAH Fit Vault system. Each feature shall contain a brief summary of the feature and information about how the feature will be implemented within the system. There will also be an accompanying sequence diagram to help visualize the relationship between the various system components needed to satisfy the feature's functionality.

2.2.1 Data Ingestion

Data ingestion is a feature that encapsulates the process of the patient selecting data files, assigning activities to the data, the system processing the data, and finally uploading the data and saving it to the system's SQL database.

This process begins with a patient navigating to the **SelectDataView** and selecting the data that they would like to upload to the system. The data selection can occur in one of two ways. The system shall give the patient the ability to click a browse button on the view that will prompt the user with a modal that can traverse the file system structure of the machine the patient is using. The patient can use this tool to select a directory that contains the files that they wish to upload. The other option available for the patient is to drag and drop any files they wish to upload into the screen where the files would normally be listed. After either of these processes is performed the files will be listed in a list element on the view for review. The patient shall be given the opportunity to remove files from the list that they do not wish to upload. When the patient is satisfied the patient can click a "Next" button to proceed to assigning activities to the data.

After clicking the "Next" button the **SelectActivityView** will be loaded. This view allows the patient to enter any number of activities that occurred during the time of data capture. The patient will be provided with an interface to add, edit, and delete activities. Adding an activity will display date picker boxes to select the time period of the activity. There will also be a drop down select list containing preset activities that the user may choose from to label their data with an activity. Once the patient is satisfied with the activity list, the patient can click a "Upload" button to begin processing the data. After the system processes the data it will be uploaded and saved to the SQL database.

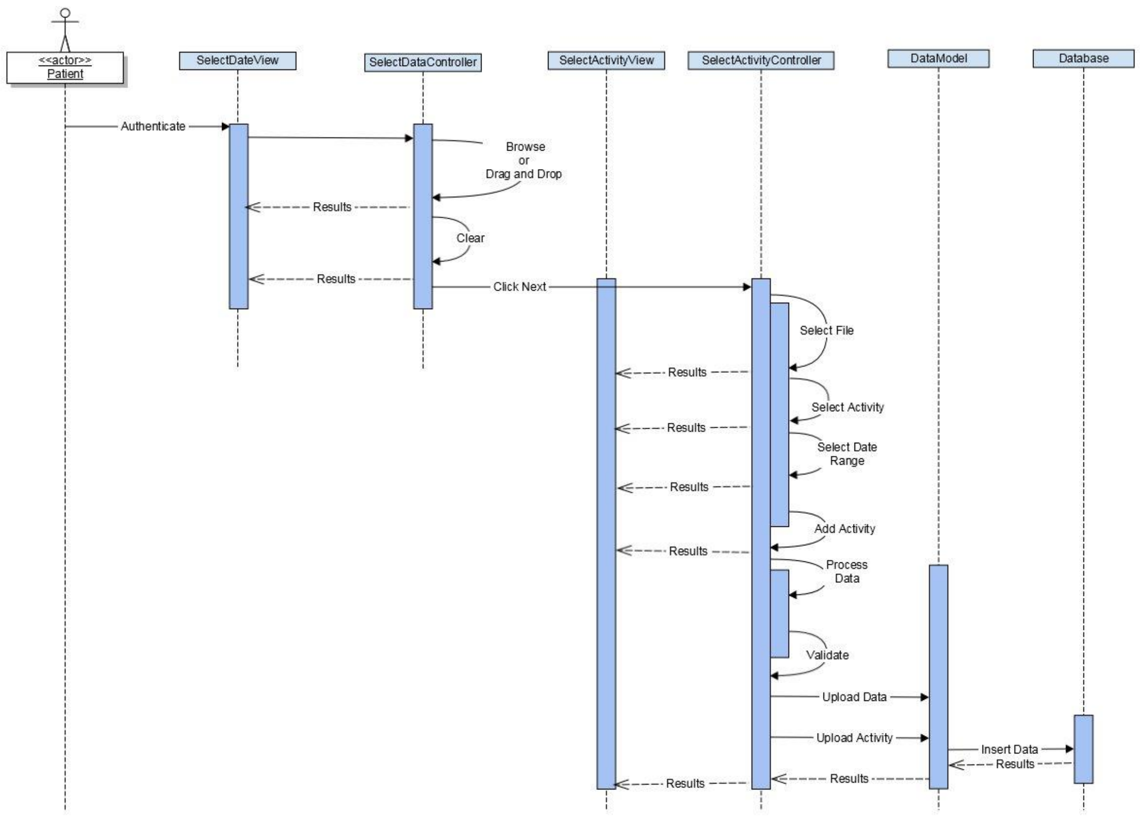


Figure 2.2: Sequence Diagram of Data Ingestion Feature

Prototype References for Data Ingestion: "Patient: Upload Data" (pg. 18)

2.2.2 Administrator Tools

The administrator shall have access to a number of tools used for user management. The first of these tools is the ability to approve or reject a user. When a new user registers as a Physician or an Experiment Administrator a System Administrator will have to approve their account in order for them to use the system. Upon creation of a new Physician or Experiment Administrator account, when a System Administrator views the **AdminHomeView**, they will see a list of user accounts that are pending approval. From this point the Admin will have the option of either clicking an approve button or a reject button. Based on that choice the user account will either update to an active or inactive status.

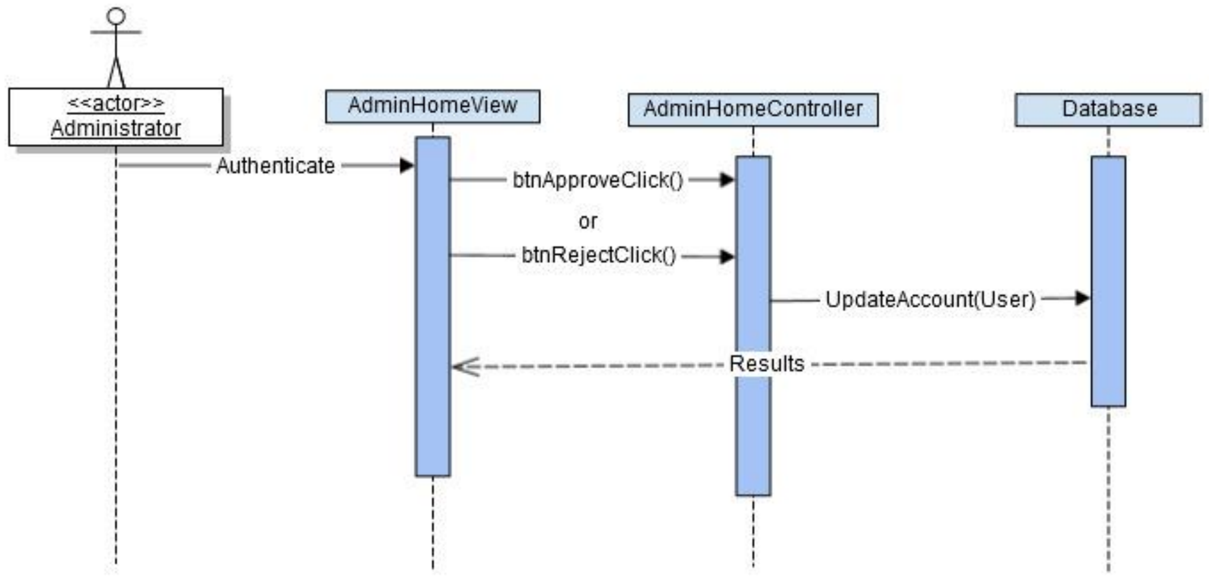


Figure 2.3: Approve Reject Account Sequence Diagram

The second set of Administrator tools involves editing the existing users in the database. These functions include enabling, disabling, editing, resetting a user's password, and deleting a user. When coming to the AdminHomeView the admin will see a list of Physicians and Experiment Administrators. When selecting a Physician, the Administrator will have the option to retrieve a list of the Physician's patients from the database. Once a user has been selected the Administrator will press a submit button and will navigate to the AdminEditView. There will be five options available: enable, disable, edit, reset password and delete. After the action is selected the user record will be updated in the database.

Regarding account deletion, when a system administrator deletes a user account that is a patient, all of the data associated with the patient will be deleted from the database. If in the future it is decided that this is not the desired functionality, it is suggested that the ability to delete a patient be removed from the system. A patient that would normally have been deleted shall be disabled instead. This will leave the patient record in the database allowing all the patient data to remain as well.

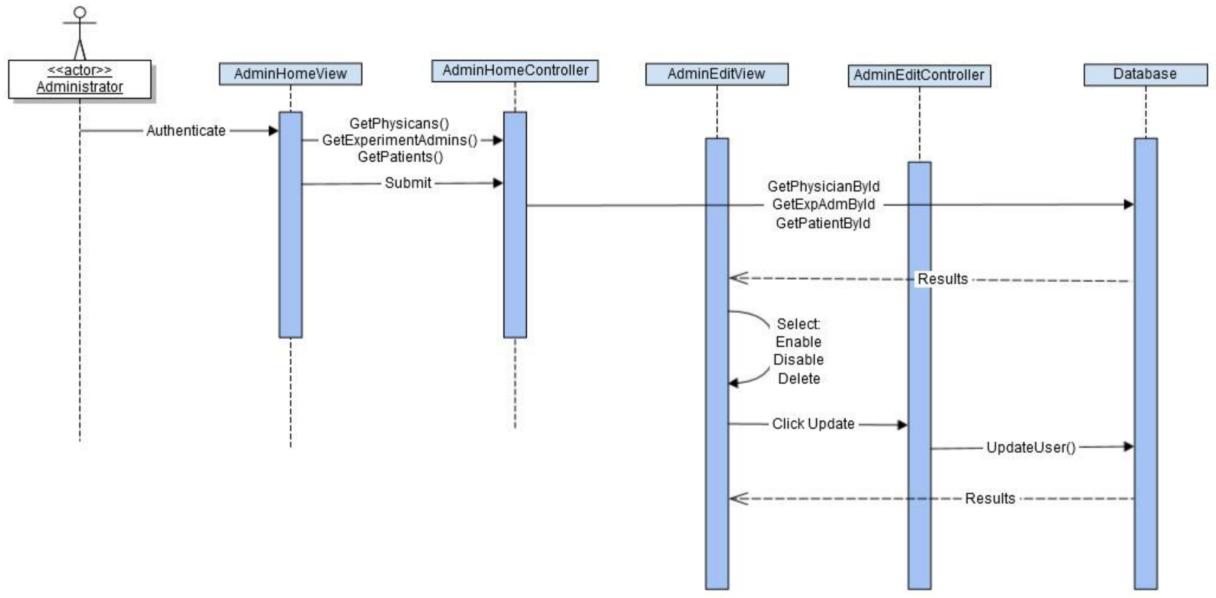


Figure 2.4: Approve/Reject/Delete Account Sequence Diagram

Prototype References for Administrator Tools: “System Admin: Welcome Screen” (pg. 34), “System Admin: Look-up User 1” (pg. 36), “System Admin: Look-up User 2” (pg. 37), “System Admin: Jsmith1 Patients” (pg. 38), “System Admin: Smith0001 User” (pg. 39)

2.2.3 Account Management

The account management feature encompasses the various tools available to the user for creating and managing their accounts. Patients will have access to the account management tools but will not be allowed to create their own accounts. Patient account creation will be described in the Physician Tools section 2.2.4.

The first tool shown below in figure 2.5 is account creation. The AccountController will provide a Register(model) procedure. The AccountController will create a new ApplicationUser when the Register() procedure is called. Then the AccountController will then call the Microsoft provided UserManager's Create(ApplicationUser,Password) procedure. This procedure will return a Success if the creation was successful. If a Success is returned, the AccountController will redirect the user back to the Home page.

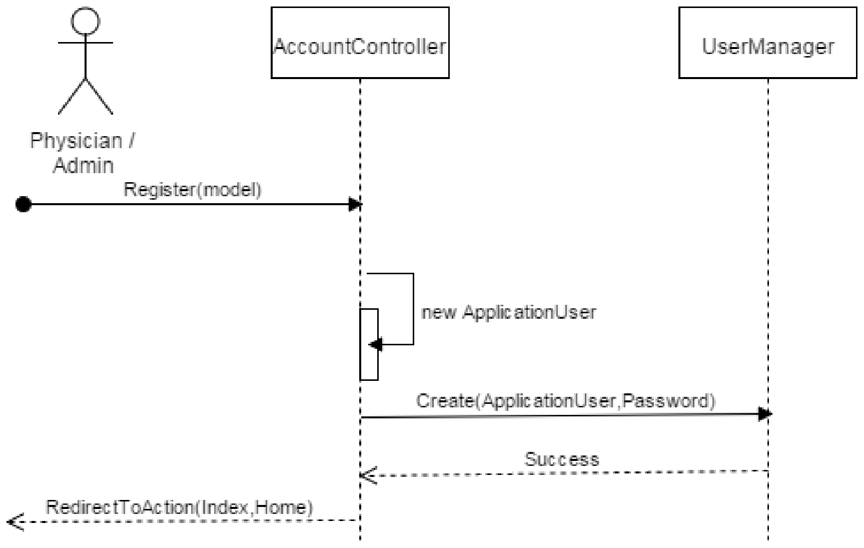


Figure 2.5: Account Creation Sequence Diagram

Next in figure 2.6 the edit account tool is detailed. The AccountController will offer a Edit(model) procedure. This procedure will interact with the Database object. When the Edit() procedure is called, the AccountController will get the username by using the Microsoft provided attribute User.Identity.Name. The AccountController will then call the Database's UserProfiles.FirstOrDefault(Username). This procedure will respond with the UserProfile corresponding to the Username. The AccountController will then set all the UserProfile attributes to the values provided in the model that was passed in with the Edit(model) call. Then the AccountController will set the Database's Entry(UserProfile).State to EntityState.Modified and call the Database's SaveChanges() procedure. The AccountController can then redirect the User to the Home page again.

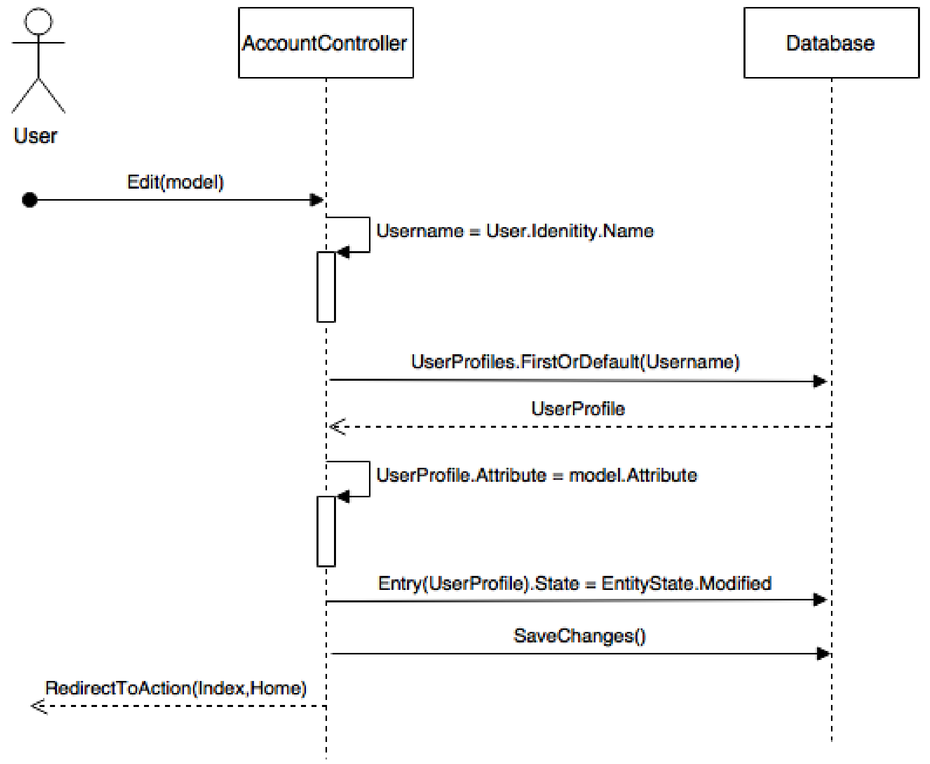

User
AccountController
Database
Edit(model)
Username = User.Idenitity.Name
UserProfiles.FirstOrDefault(Username)
UserProfile
UserProfile.Attribute = model.Attribute
Entry(UserProfile).State = EntityState.Modified
SaveChanges()
RedirectToAction(Index,Home)

Figure 2.6: Edit Account Sequence Diagram

Two other key elements involved in user account management are the ability securely login to your account and to logout of your account. Figure 2.6 details the login in functionality, and figure 2.7 details the logout function.

The AccountController will offer a Login() procedure. It will interact with the Microsoft provided UserManager and SignInManager objects. When the Login is called it will call the UserManager's FindByName(Username) procedure to make sure the user exists in the database. Then the Login() procedure will call the SignInManager's PasswordSignIn(Username,Password,RememberMe) procedure. This will return a SignInStatus that tells the AccountController if the signin was successful. If the signin was successful, the AccountController will redirect the user to the retURL.

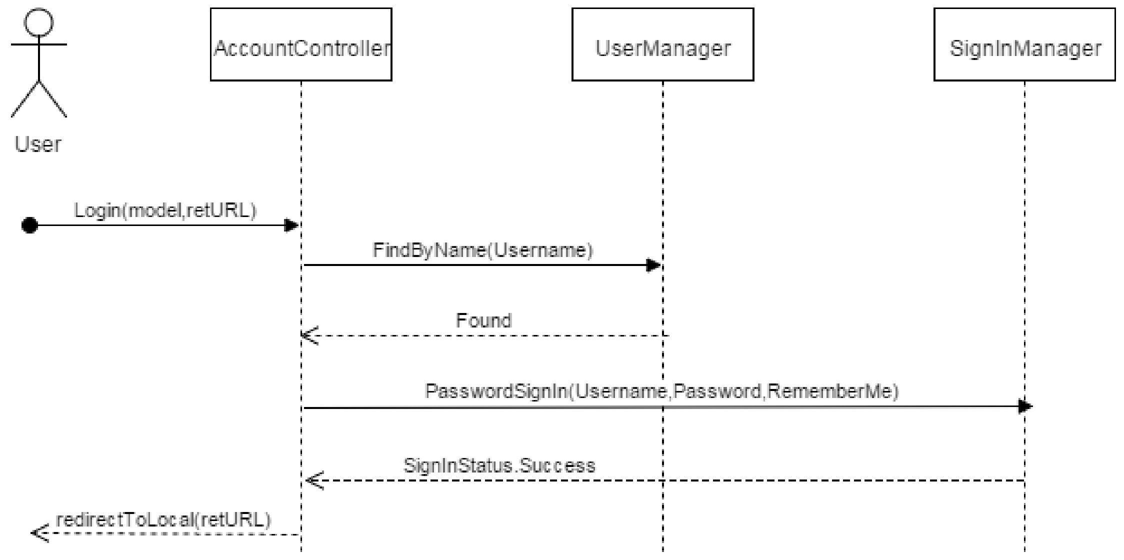


Figure 2.7: Login Sequence Diagram

The AccountController has a public procedure called Logoff(). This procedure when called by the user shall call the formAuthentication.SignOut() procedure provided by Microsoft in the MVC library. Once the SignOut() procedure has finished, the Logoff() procedure will redirect the user back to the Login screen.

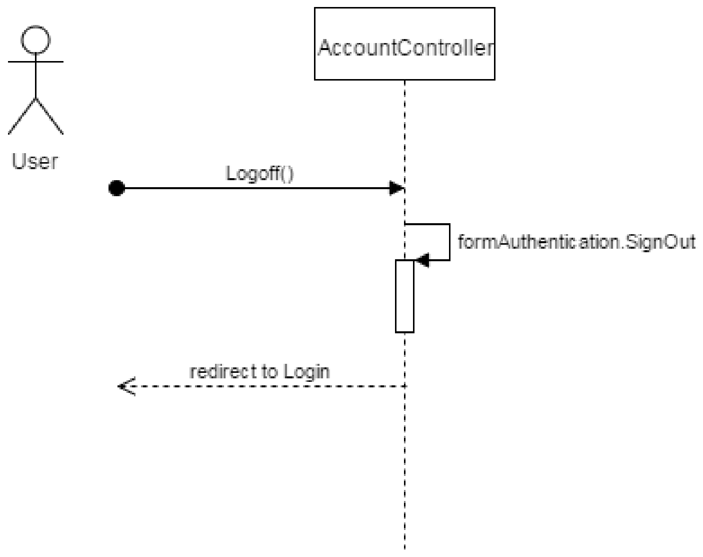


Figure 2.8: Logoff Sequence Diagram

Prototype References for Account Management: “Main Screen” (pg. 4), “Request Account 1” (pg. 9), “Logout” (pg. 11), “Patient: Edit Account Settings” (pg. 16), “Physician: Edit Account Settings (pg. 22), “Experimenter: Edit Account Settings” (pg. 31), “System Admin: Edit Account Settings” (pg. 35)

2.2.4 Physician Actions

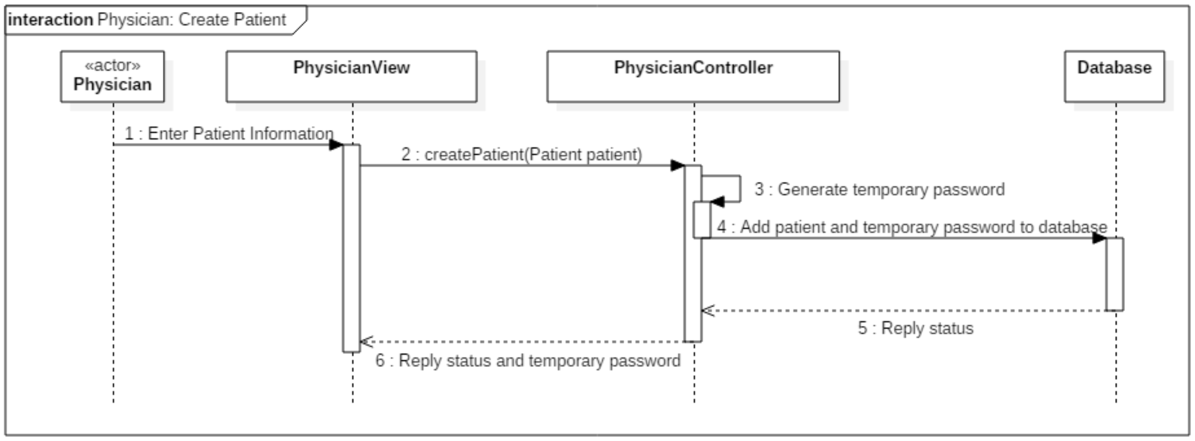

interaction Physician: Create Patient
«actor» Physician
PhysicianView
PhysicianController
Database
1 : Enter Patient Information
2 : createPatient(Patient patient)
3 : Generate temporary password
4 : Add patient and temporary password to database
5 : Reply status
6 : Reply status and temporary password

Figure 2.9: Create Patient Sequence Diagram

The "Physician: Create Patient" sequence diagram, as seen in figure 2.9, details the interactions when a physician adds a patient to the system. The physician will supply the details of the patient to the system, and the system will generate a password for the user and add the user to the database. If successful, the system will reply with a confirmation that the user was added to the database and the generated password. If unsuccessful, the system will display error information.

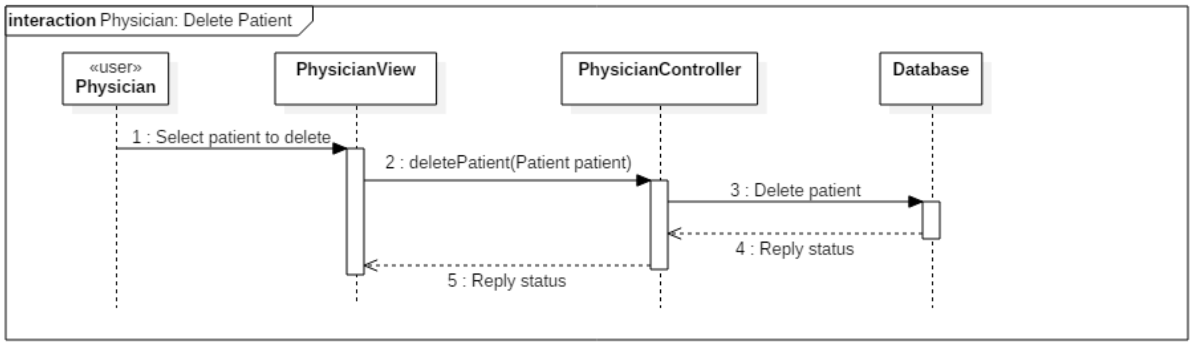


Figure 2.10: Delete Patient Sequence Diagram

The "Physician: Delete Patient" sequence diagram, as seen in figure 2.10, details the interactions when a physician removes a patient from the system. The physician must select one of their patients to remove from the system. After the physician has selected a patient, the system will attempt to delete the patient. If successful, the system will reply that the patient was deleted or else display error information. As per section 2.2.2, deleting a patient will also remove all the patient data from the system.

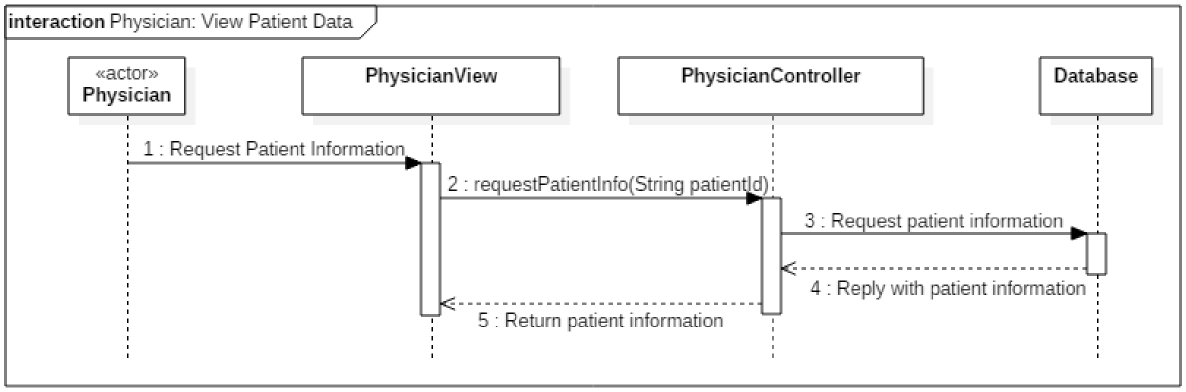

interaction Physician: View Patient Data
«actor» Physician
PhysicianView
PhysicianController
Database
1 : Request Patient Information
2 : requestPatientInfo(String patientId)
3 : Request patient information
4 : Reply with patient information
5 : Return patient information

Figure 2.11: View Patient Data Sequence Diagram

The “Physician: View Patient Data” sequence diagram, as seen in figure 2.11, details the interactions when a physician wants to view the data for one of their patients. The physician selects a patient to view the data for. After the physician makes their selection, the PhysicianController will query the database for the patient information. The PhysicianView will display the patient information.

Prototype References for Physician Actions: “Physician: Create Patient” (pg. 24), “Physician: View Patient Data” (pg. 26), “Physician: Select Session” (pg. 27), “Physician: View Session” (pg. 28)

2.2.5 Experiment Tools

This section describes the various experiment tools made available to experiment administrators and physicians by the system. The experiment administrator will be the only user allowed to create, modify, and delete experiments. The physicians using this system will have access to view the experiments that have been created by the experiment administrators using the system. Experiments will be created based on various criteria such as age, height, weight, ethnicity, race, location, and gender.

The create experiment function in figure 2.12 functions as follows. As experiment administrator navigates to the experiment module in the web application. Pressing a create experiment button on the ExperimentView will display a modal form with criteria for creating the experiment. The form will include the criteria list above as well as drop down selection list containing various types of medical data that can be used in the experiments. Once the experiment administrator is satisfied with the experiment criteria, a create button on the modal can be pressed to call the CreateExperiment function in the ExperimentController. The controller will issue an InsertExperiment call passing the newly created Experiment object to the database to be added.

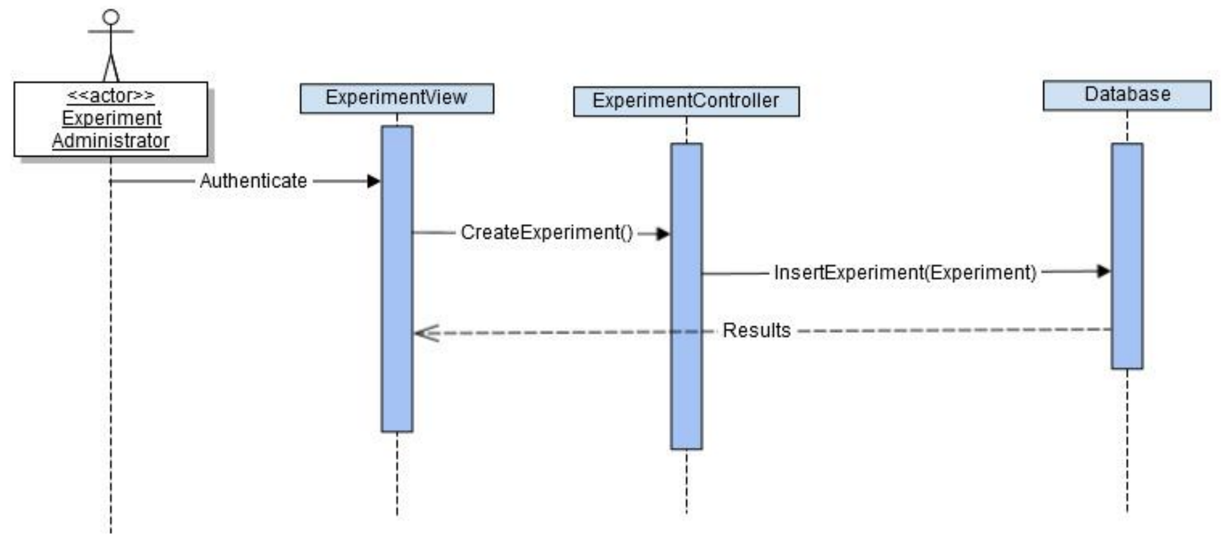


Figure 2.12: Create Experiment Sequence Diagram

The edit experiment function shown in figure 2.13 operates as follows. The experiment administrator shall have a list of their current experiments viewable when they access the ExperimentView. Selecting the edit option next to the experiment will display the same modal that was filled out when creating the experiment. This time the form will already be filled out with the current experiment criteria. The experiment administrator may now change any element of the experiment he or she wishes. Pressing an update button now on the modal will call the EditExperiment function in the ExperimentController. The controller will then tell the database to update the experiment record by calling the UpdateExperiment function passing the updated Experiment object as a parameter.

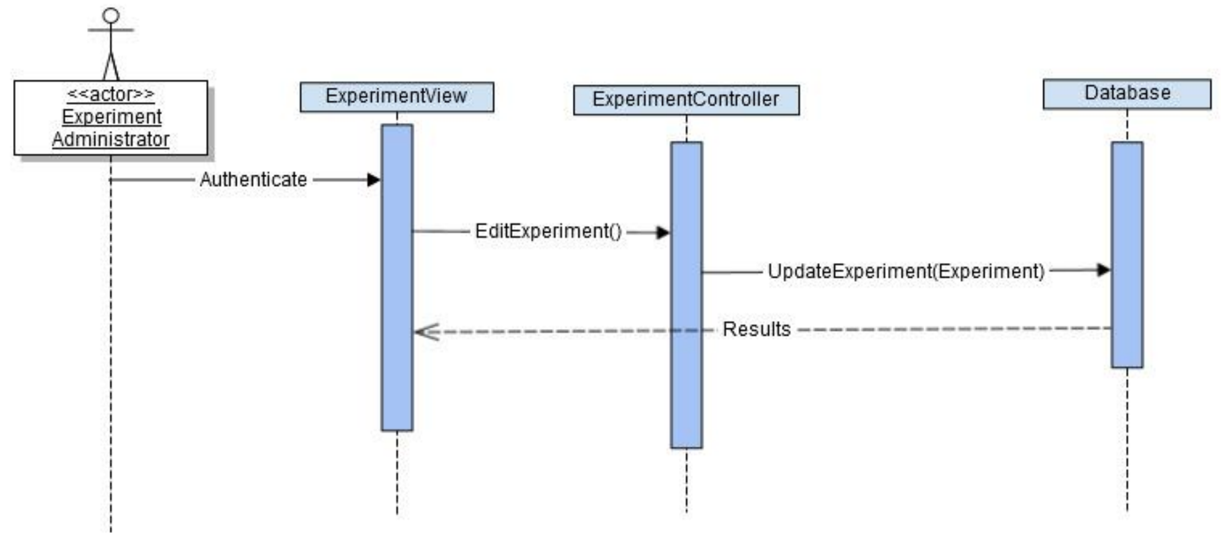


Figure 2.13: Edit Experiment Sequence Diagram

Deleting an experiment involves an experiment administrator navigating to the ExperimentView and pressing the delete button next to one of their experiments. A modal dialog asking if the experiment administrator is sure that they wish to delete the experiment will appear. Clicking the "OK" button will call the DeleteExperiment function in the ExperimentController. The controller will inform the database that the experiment needs to be deleted. This process is illustrated in figure 2.14 below.

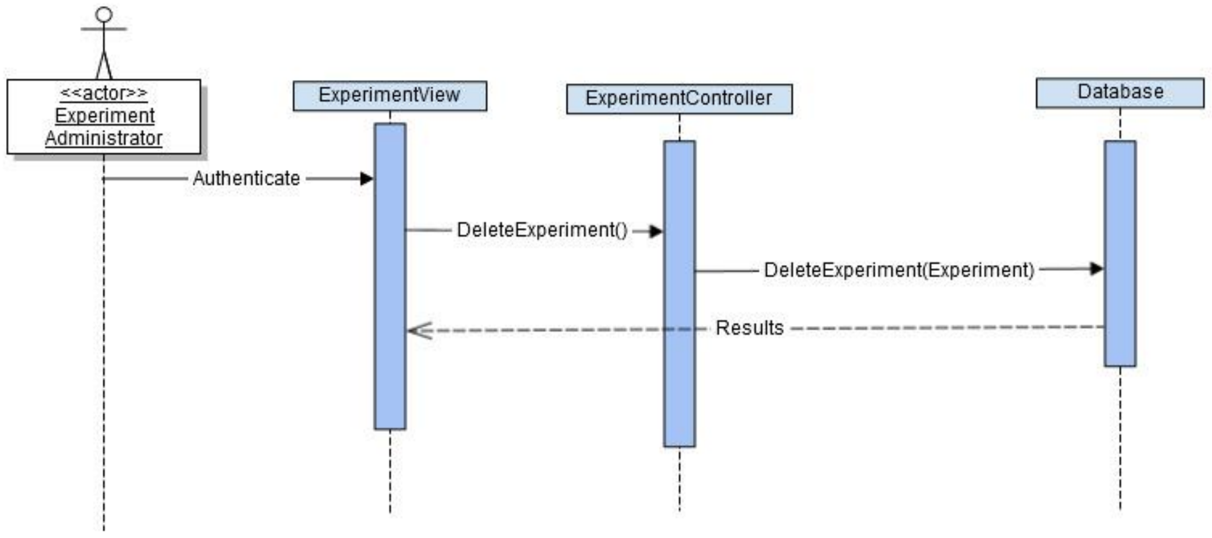


Figure 2.14: Delete Experiment Sequence Diagram

Both the physicians and experiment administrators will have access to viewing the experiments created in the system. Upon navigating to the ExperimentView, they physician will be able to view a list of all experiments in the system. The experiment administrator however will only have a list of the experiments they have created. Clicking a view button next to the experiment will call the RunExperiment function in the ExperimentController. This will get the experiment query information from the database and execute the query. The controller will collect all of the patients that meet the experiment criteria. The controller will then request the relevant medical data for each patient based on the experiment criteria. Figure 2.15 shows a function called CollectData(). This function is merely a placeholder for the various calls to the database that are required to collect all of the medical data. Once all the data is collect the controller will process the data in order to return in back to the ExperimentView where the results of the experiment are now displayed.

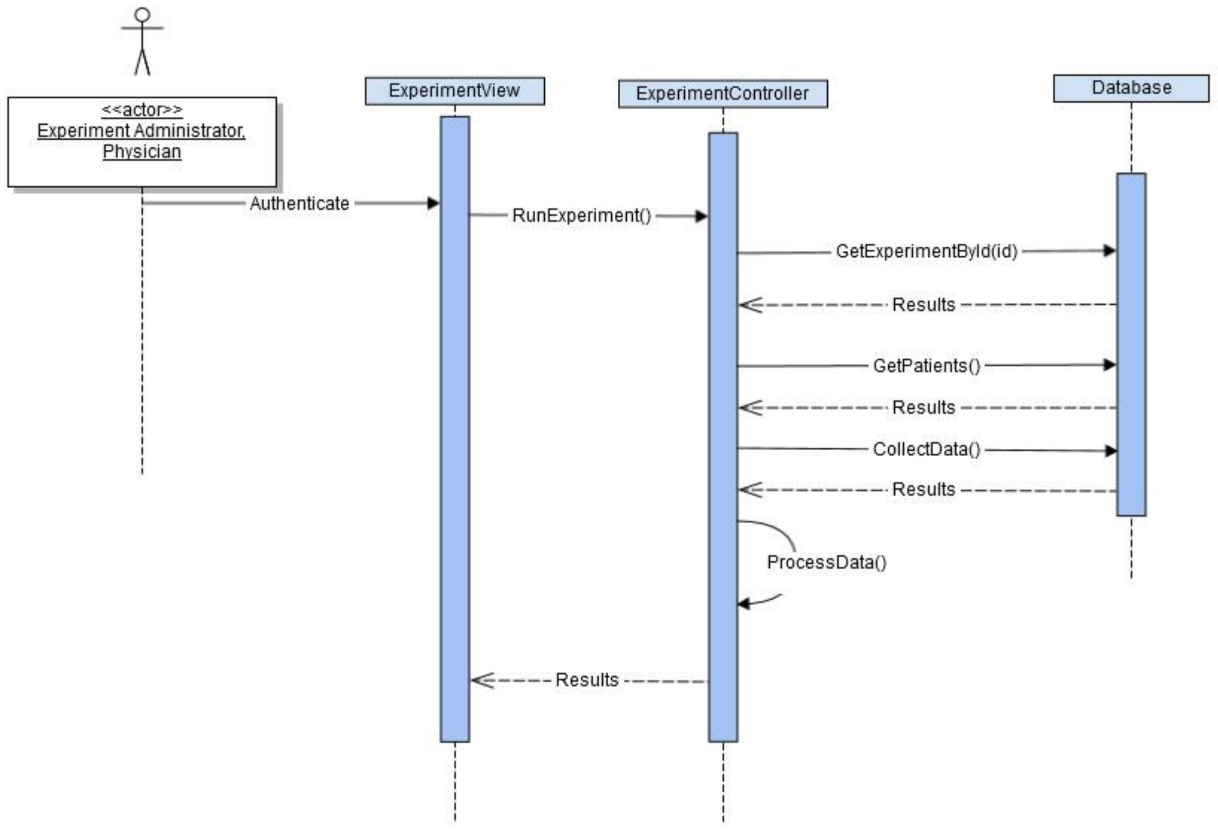


Figure 2.15: View Experiment Sequence Diagram

Prototype References for Experimenter Tools: “Experimenter: Create Experiment” (pg. 30), “Experimenter: View Experiments” (pg. 32), “Experimenter: View Results” (pg. 33)

2.2.6 Patient Actions

The “Patient: View Patient Data” sequence diagram, as seen in figure 2.16, details the interactions when a patient wants to view their own data. The PatientController will query the database for their information. The PatientView will display their information. The request data interaction from the controller to the database represents all of the get calls to the various database tables that contain medical data that the patient may have selected to view.

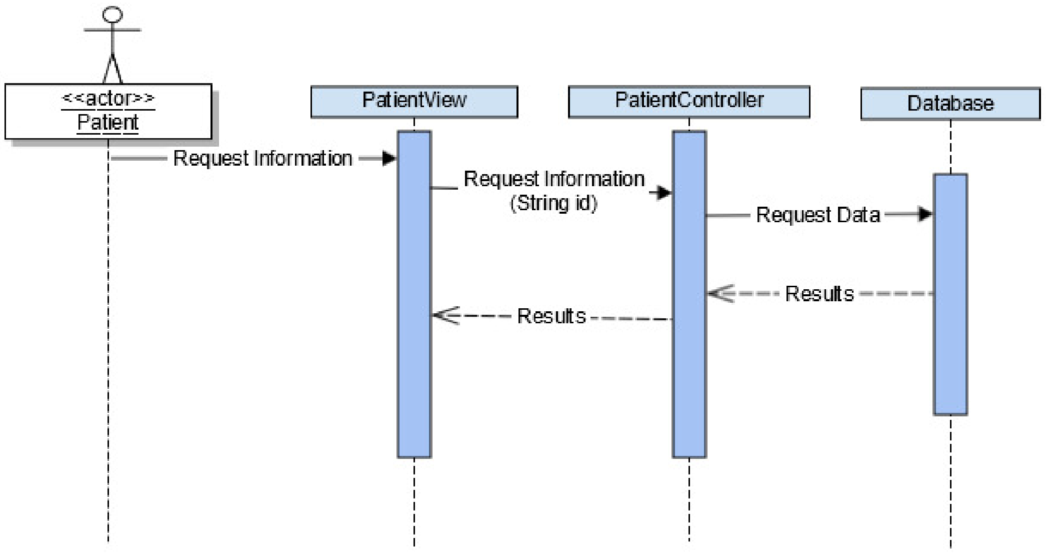
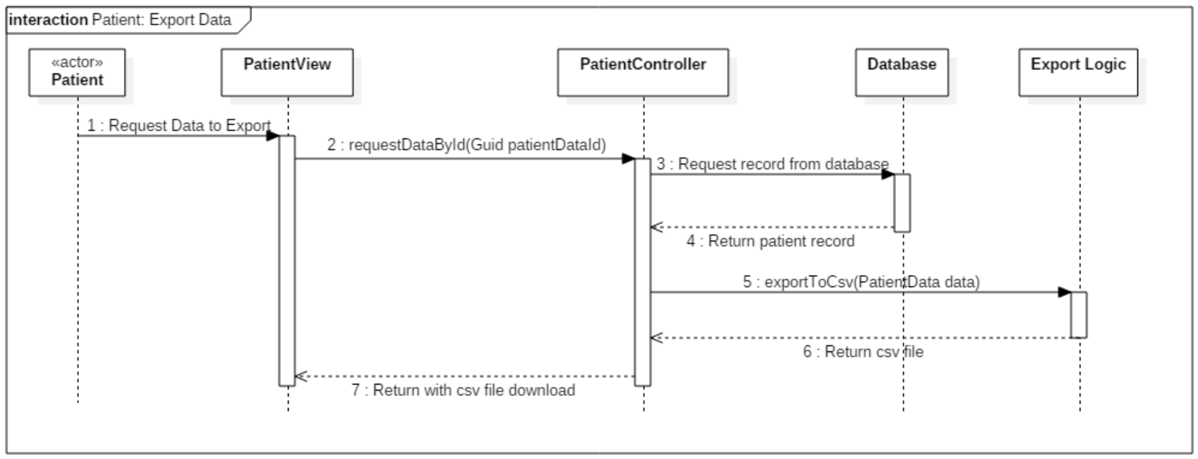


Figure 2.16: View Own Data Sequence Diagram

Prototype References for Patient Actions: “Patient: View Data” (pg. 14), “Patient: View Session” (pg. 15)

2.2.7 Export Data

There are two different exports available in this web application. The first export is available to both patients and physicians. The export is used for saving a patient data record locally to the user’s hard drive. These exports are described in figures 2.17 and 2.18.



interaction Patient: Export Data
«actor» Patient
PatientView
PatientController
Database
Export Logic
1 : Request Data to Export
2 : requestDataById(Guid patientDataId)
3 : Request record from database
4 : Return patient record
5 : exportToCsv(PatientData data)
6 : Return csv file
7 : Return with csv file download

Figure 2.17: Export Own Data Sequence Diagram

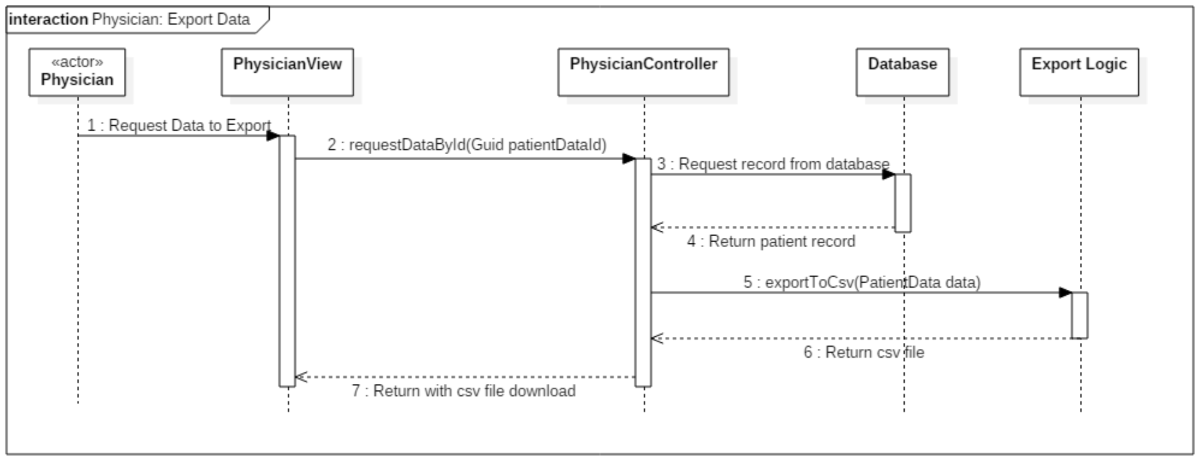

interaction Physician: Export Data
«actor» Physician
PhysicianView
PhysicianController
Database
Export Logic
1 : Request Data to Export
2 : requestDataById(Guid patientDataId)
3 : Request record from database
4 : Return patient record
5 : exportToCsv(PatientData data)
6 : Return csv file
7 : Return with csv file download

Figure 2.18: Export Patient Data Sequence Diagram

In both cases the user will have to access the view available to them to use the system. A patient will have access to the PatientView where the physician will have access to the Physician view. After selecting a patient record, each user will have the option to select an export option instead of viewing the data in a graphical formal. Clicking the export will call the requestDataById function in the corresponding controller depending on the user executing the export. The controller will then request the patient record from the database. After receiving the record it will look up all the relevant medical data from the database. The controller will then make a call to a static class to handle the exporting of this data to a .csv file. After the data is serialized into the correct format a download option will be prompted to the user.

The next export option is available to both the experiment administrator and the physician. The system will allow either of those types of users to export the results of an experiment. This process is detailed in figure 2.19. Either user from the ExperimentView selects an experiment from the list of available experiments. The experiment will have a button available to the user to export the experiment when pressed instead of the run experiment button. Clicking the export button will call the requestDataById function in the controller. The controller will request all of the data that is used to generate the export report. That data is then sent to the static class called ExportLogic. The data is then processed into a .csv file format. The user is prompted with an option to download the file after the data available for download.

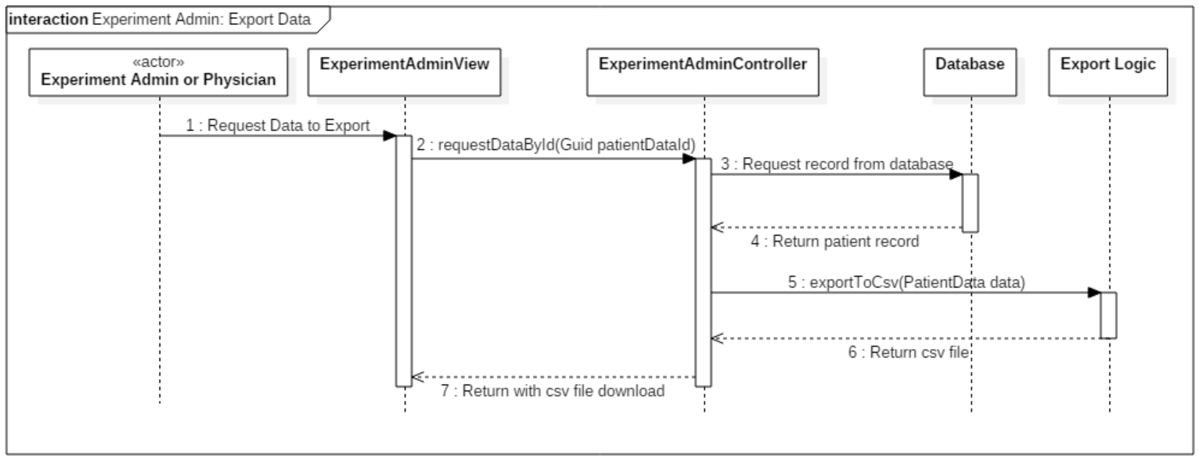

interaction Experiment Admin: Export Data
«actor» Experiment Admin or Physician
ExperimentAdminView
ExperimentAdminController
Database
Export Logic
1 : Request Data to Export
2 : requestDataById(Guid patientDataId)
3 : Request record from database
4 : Return patient record
5 : exportToCsv(PatientData data)
6 : Return csv file
7 : Return with csv file download

Figure 2.19: Export Experiment Data Sequence Diagram

Prototype References for Export Tools: “Patient: View Session” (pg. 15), “Physician: View Session” (pg. 28)

2.3 Database Schema

Figure 2.20 below shows the SQL database schema that will be implemented with this system. The database that will be used is SQL Server 2014. The database will be developed using the Entity Framework 6.0 package through Visual Studio 2015. The diagram shows all the tables and columns in each table. Primary keys for each table are denoted with (PK), and foreign keys are denoted with (FK). The links show not only the association information but also point to the columns in each table the foreign key refers to. Each class is further described in the detailed design section 3.2.1.

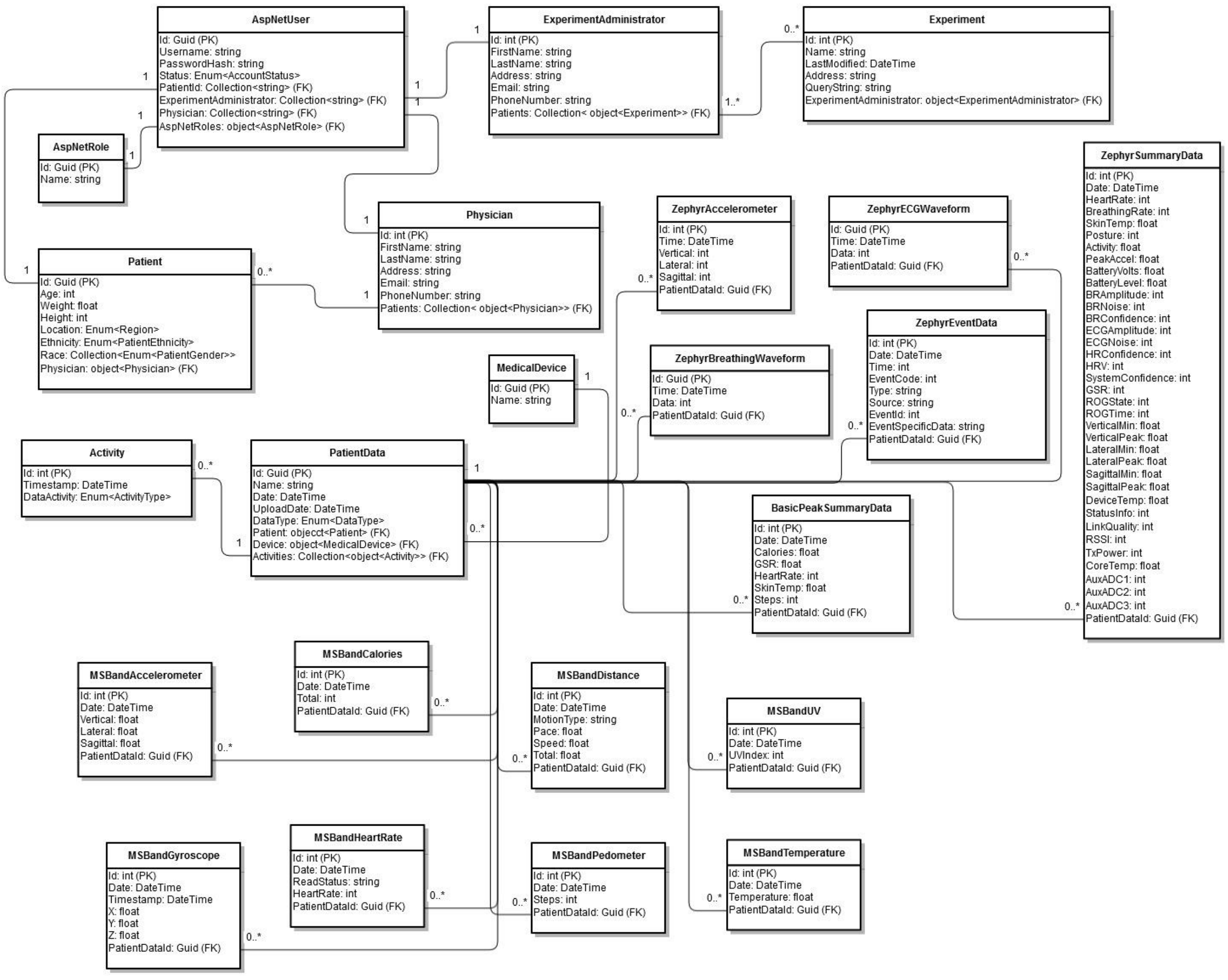


Figure 2.20: Database Schema

3 Detailed Description of Components

3.1 Presentation Layer Overview

3.1.1 Controllers

3.1.1.1 SelectDataController

This class is responsible for selecting the data files for processing. This class provides the backend for the **SelectDataView**. See Figure 2.2 sequence diagram for this controller.

<<Task>>

Attributes:

- **Private Collection<int, String>** DataFiles
  - Dictionary containing the paths to the data files and a unique id.

- **Private String** DirectoryPath
  - Path to the directory containing the data files to be processed.

Operations:

- Sets and gets for all attributes.

- **Public ActionResult** btnFolderBrowseClick()
  - When the folder browser button is clicked a folder browser dialog object is created, a directory is selected by the user, and the DirectoryPath is set.
  - The results are returned to the **SelectDataView.**

- **Public ActionResult** DragAndDropFiles()
  - When the desired files are selected and dropped into the GUI window.
  - The results are returned to the **SelectDataView.**

- **Public ActionResult** btnClearClick()
  - When the clear button is clicked any stored data and file path(s) will be removed.
    - DirectoryPath variable reset
    - DataFiles collection emptied
    - The results are returned to the **SelectDataView.**

- **Private ActionResult** btnNextClick()
  - When the next button is clicked the file locations of the files in the DirectoryPath are added to the DataFiles collection.
  - The DataFiles collection is passed to the **SelectActivityController.**

3.1.1.2 SelectActivityController

This class is responsible for assigning an activity to a date and time portion of a data file. This class provides the backend for the **SelectActivityView**. See Figure 2.2 sequence diagram for this controller.

<<Task>>

Attributes:

- **Private Collection<Activity>** Activities
  - Activities is a collection used to hold all **Activity** objects.

Operations:

- Sets and gets for all attributes.

- **Public ActionResult** SelectFile()
  - A data file is selected (id).
  - The results are returned to the **SelectActivityView**

- **Public ActionResult** SelectActivity()
  - An activity is selected.
  - The results are returned to the **SelectActivityView**

- **Public ActionResult** AddDateRange()
  - A date and time range is selected.
  - The results are returned to the **SelectActivityView**

- **Public ActionResult** AddActivity()
  - An activity object is created and assigned the values from the selections
    - File id
    - Activity
    - Date time range
  - The results are returned to the **SelectActivityView**

- **Public ActionResult** ProcessData()
  - Iterates through the data files.
    - Calls the Validate method to validate each data file
      - If a file is invalid it is not process and any associated activates are dropped.
      - If a file is valid the UploadDataFile method is called and passed the data file object
        - Any activities associated with the data file is uploaded via the UploadDataActivity method.
  - When processing is complete the results will be returned to the **SelectActivityView**

- **Public ActionResult** Validate(**File file**)
  - Returns if the file is valid or invalid.

- **Public ActionResult** UploadDataFile(**File file**)
  - This method uploads the data file into the database.

- **Public ActionResult** UploadActivity(**Activity activity**)
  - This method uploads the activity into the database.

3.1.1.3 AccountController

This class is responsible for user accounts. This class provides the backend for the **AccountView**.

<<Task>>

Operations:

- **Public ActionResult** Login(LoginViewModel model, String returnURL)
    - Finds the user by name (username).
    - Calls the sign in manager to sign in by password.
    - Returns the returnURL.

- **Public ActionResult** Logoff()
    - Calls the form authentication's sign out procedure.

- **Public Task<ActionResult>** Register(RegisterVIewModel model)
    - Calls user manager's create
    - Returns success

3.1.1.4 AdminHomeController

This class is responsible for accepting or rejecting a user account request. This class provides the backend for the **AdminHomeView**. The AdminHomeView has a link to navigate to look-up a user in order to delete, enable, or disable their account. See Figure 2.3 and 2.4 sequence diagram for this controller.

<<Task>>

Attributes:

- **Private Collection<String>** UserNames
    - Collection of usernames of accounts to be approved or rejected.

Operations:

- **Public string** GetPhysicians()
    - Gets a list of all the physicians in the system.

- **Public string** GetExperimentAdministrators()
    - Gets a list of all the experiment administrators in the system.

- **Public string** GetPatients(int PhysicianId)
    - Gets a list of all the patients for the selected physician in the system.

- **Public ActionResult** btnApproveClick()
    - When the approve button is clicked the user account request is approved.
    - The results are returned to the **AdminHomeView.**

- **Public ActionResult** btnRejectClick()

- When the reject button is clicked the user account request is denied.
- The results are returned to the **AdminHomeView.**

- **Public ActionResult** btnSubmit()
  - A query by username is sent to the database and the user account information is returned.
  - The user account information is passed to the **AdminEditController.**

Constructors:

- **Public** AdminHomeController()
  - Default constructor

3.1.1.5 AdminEditController

This class is responsible for allowing the system administrator to enable, disable, or delete a user account. This class provides the backend for the **AdminEditView**. See Figure 2.4 sequence diagram for this controller.

<<Task>>

Attributes:

- **Private String** Username
  - The username of the account to modify.

Operations:

- **Public ActionResult** btnUpdate()
  - An update is sent to the database based on the selected option:
    - Enable
    - Disable
    - Delete
    - Edit
    - Reset Password
  - The results are returned to the **AdminEditView.**

Constructors:

- **Public** AdminEditController()
  - Default constructor

3.1.1.6 PhysicianController

This class is responsible for handling the various functions available to a physician. This class provides the backend for the **PhysicianView**. The actions available to the physician are detailed in the operations section below.

<<Task>>

Attributes:

- **Private Physician** physician
  - The physician associated with the class.

Operations:

- **Public ActionResult** CreatePatient(Patient patient)
  - The function to add a patient to the system.

- **Public ActionResult** DeletePatient(Patient patient)
  - The function to remove a patient from the system.

- **Public ActionResult** RequestPatientInformation(String patientId)
  - The function to request patient information from the system.

- **Public JsonResult** RequestDataById(String id)
  - The function to request an export of a patient data record.

Constructors:

- **Public** PhysicianController()
  - Default constructor.

3.1.1.7 ExperimentController

This class is responsible for handling experiment functions called from the **ExperimentView**. The actions are available to the experiment administrators with physicians having access to the view experiment functionality.

<<Task>>

Attributes:

- **Private Collection<Experiment>** experiments
  - A collection of experiments available to the user.

Operations:

- **Public ActionResult** CreateExperiment(Experiment experiment)
  - The function used to add an experiment to the system.

- **Public ActionResult** DeleteExperiment(Experiment experiment)
  - The function used to remove an experiment from the system.

- **Public ActionResult** EditExperiment(Experiment experiment)
  - The function used to edit an experiment in the system.

- **Public ActionResult** RunExperiment(int experimentId)
  - The function used to run the selected experiment.

- **Private string** ProcessData(Collection<object>)
    - The function is here in the event that the data returned from the database will need to be processed in some way prior to be sending it to the view.

- **Public JsonResult** RequestDataById(String id)
    - The function to request an export of experiment data record.

Constructors:

- **Public** ExperimentController()
    - Default constructor.

3.1.1.8 PatientController

This class is responsible for handling the various functions available to a patient. This class provides the backend for the **PatientView**. The actions available to the patient are detailed in the operations section below.

<<Task>>

Operations:

- **Public ActionResult** RequestInformation(String id)
    - The function to request the patient's own information from the system.

- **Public JsonResult** RequestDataById(String id)
    - The function to request an export of a patient data record.

Constructors:

- **Public** PatientController()
    - Default constructor.

3.2 Business Logic Layer Overview

3.2.1 Entities

3.2.1.1 LoginViewModel

This class contains data associated with a user's login information. The **LoginViewModel** class is used by the **AccountController**.

<<Entity>>

Attributes:

- **Public string** UserName
    - Username of the user that is used to authenticate their login.

- **Public string** Password
  - Password of the user that is used to authenticate their login.

- **Public bool** RememberMe
  - Used to inform the browser to remember the login and password information in a cookie for future logins by that user.

Operations:

- Sets and gets for all attributes.

3.2.1.2 RegisterViewModel

This class contains data associated registering a new user account. The **RegisterViewModel** class is used by the **AccountController**.

<<Entity>>

Attributes:

- **Public string** UserName
  - Username of the user that is used to authenticate their login.

- **Public string** Password
  - Password of the user that is used to authenticate their login.

- **Public bool** ConfirmPassword
  - Password of the user that is used to authenticate their login. Must match the Password field in order to successfully register.

Operations:

- Sets and gets for all attributes.

3.2.1.3 Activity

This class contains data associated with an activity tied to a date and time portion of a data file. The **Activity** class is used by the **ActivitySelectionController**.

<<Entity>>

Attributes:

- **Public int** fileId
  - The id of the file to be assigned an activity.

- **Public DateTime** TimeStamp
  - The date and time of the activity.

- **Public ActivityType** DataActivity
    - The activity of the data.

Operations:

- Sets and gets for all attributes.

Constructors:

- **Public** Activity()
    - Default constructor

3.2.1.4 AspNetUser

This class describes the attributes common to all users. This will act as a base class for **Patients**, **Physicians**, and **Experiment Administrators**.

<<Entity>>

Attributes:

- **Public Guid** Id
    - Unique id for each user.

- **Public string** UserName
    - The user name used by each user that will be used to log into the system.

- **Public string** PasswordHash
    - Hash string representing the user's password used to log into the system.

- **Public AccountStatus** Status
    - Enumeration value that describes the current account status of the user.

- **Public Nullable<string>** PatientId
    - If the user is a patient this field will be used as a foreign key to the **Patient** table Id field.

- **Public Nullable<int>** Physician
    - If the user is a physician this field will be used as a foreign key to the **Physician** table Id field.

- **Public Nullable<int>** ExperimentManager
    - If the user is a experiment manager this field will be used as a foreign key to the **ExperimentManager** table Id field.

- **Public virtual AspNetRole** AspNetRoles
    - Link to the user role in the **AspNetRole** table

Operations:

- Sets and gets for all attributes.

Constructors:

- **Public** AspNetUser()
    - Default constructor

3.2.1.5 Patient

This class describes the attributes and construction of a **Patient** user object.

<<Entity>>

Attributes:

- **Public string** Id
    - Unique patient id created by the patient's physician used to identify their patients.

- **Public int** Age
    - The age of the patient.

- **Public float** Weight
    - The weight of the patient.

- **Public int** Height
    - The height of the patient given in inches.

- **Public Region** Location
    - The state where the patient is located.

- **Public PatientEthnicity** Ethnicity
    - Enumeration value used to describe the patient's ethnicity.

- **Public Collection<PatientRace>** Race
    - Collection of enumeration values used to describe the patient's race.

- **Public PatientGender** Gender
    - Enumeration value used to describe the patient's gender.

- **Public virtual Physician** Physician

- Foreign key link to the patient's physician.

Operations:

- Sets and gets for all attributes.

Constructors:

- **Public** Patient()
  - Default constructor

3.2.1.6 Physician

This class describes the attributes and construction of a **Physician** user object.

<<Entity>>

Attributes:

- **Public int** Id
  - Unique id generated by the system for each physician.
- **Public string** FirstName
  - Physician's first name.
- **Public string** LastName
  - Physician's last name.
- **Public string** Address
  - Physician's office address information.
- **Public string** Email
  - Email address for the physician
- **Public string** PhoneNumber
  - Phone number for the physician
- **Public virtual Collection<Parent>** Patients
  - Collection of all of the physician's patients.

Operations:

- Sets and gets for all attributes.

Constructors:

- **Public** Physician()
  - Default constructor

3.2.1.7 ExperimentAdministrator

This class describes the attributes and construction of an **Experiment Administrator** user object.

<<Entity>>

Attributes:

- **Public int** Id
  - Unique id generated by the system for each experiment administrator.
- **Public string** FirstName
  - Experiment administrator's first name.
- **Public string** LastName
  - Experiment administrator's last name.
- **Public string** Address
  - Experiment administrator's address information.
- **Public string** Email
  - Email address for the experiment administrator
- **Public string** PhoneNumber
  - Phone number for the experiment administrator.
- **Public virtual Collection<Experiment>** Experiments
  - Collection of all of the experiment administrator experiments.

Operations:

- Sets and gets for all attributes.

Constructors:

- **Public** ExperimentAdministrator()
  - Default constructor

3.2.1.8 Experiment

This class describes the attributes and construction of an **Experiment** object.

<<Entity>>

Attributes:

- **Public int** Id
  - Unique id generated by the system for each experiment.

- **Public string** Name
  - Experiment name.

- **Public DateTime** LastModified
  - Date information for when experiment was last modified.

- **Public string** QueryString
  - Query information used to collect the experiment data.

- **Public virtual ExperimentAdministrator** ExperimentAdministrator
  - Foreign key relationship to the experiment administrator that created the experiment.

Operations:

- Sets and gets for all attributes.

Constructors:

- **Public** ExperimentAdministrator()
  - Default constructor

3.2.1.9 Medical Device

This class describes the attributes and construction of an **MedicalDevice** object.

<<Entity>>

Attributes:

- **Public int** Id
  - Unique id generated by the system for each medical device.

- **Public string** Name
  - Medical device name.

Operations:

- Sets and gets for all attributes.

Constructors:

- **Public** MedicalDevice()
  - Default constructor

3.2.1.10 PatientData

This class describes the attributes and construction of a **PatientData** object.

<<Entity>>

Attributes:

- **Public Guid** Id
    - Unique id generated by the system for each patient data record.

- **Public string** Name
    - Name of the patient data record

- **Public DateTime** Date
    - Date information for when the patient collected the data.

- **Public DateTime** UploadDate
    - Date when the patient uploaded the data record to the system.

- **Public DataType** DataType
    - Enumeration detailing what type of data is stored in the patient data record. This will be used to determine what database table should be queried to get the correct data for this data record.

- **Public virtual Patient** Patient
    - Foreign key relationship to the Patient that the data record belongs to.

- **Public virtual MedicalDevice** Device
    - Foreign key relationship to the MedicalDevice that the data record was collected from.

- **Public virtual Collection<Activities>** Activities
    - Foreign key relationship to a list of Activities that the patient used to describe the data record.

Operations:

- Sets and gets for all attributes.

Constructors:

- **Public** PatientData()
    - Default constructor

3.2.1.11 MSBandAccelerometer

This class describes the attributes and construction of a **MSBandAccelerometer** object.

<<Entity>>

Attributes:

- **Public int** Id
    - Unique id generated by the system for each Microsoft Band Accelerometer data record.

- **Public DateTime** Date
    - Date information for when the patient collected the data.

- **Public float** Vertical
    - Vertical axis of the accelerometer data.

- **Public float** Lateral
    - Lateral axis of the accelerometer data.

- **Public float** Sagittal
    - Sagittal axis of the accelerometer data.

- **Public Guid** PatientDataId
    - Foreign key relationship to the id of the patient data record from the **PatientData** table that this data record corresponds to.

Operations:

- Sets and gets for all attributes.

Constructors:

- **Public** MSBandAccelerometer()
    - Default constructor

3.2.1.12 MSBandCalories

This class describes the attributes and construction of a **MSBandCalories** object.

<<Entity>>

Attributes:

- **Public int** Id
    - Unique id generated by the system for each Microsoft Band Calorie data record.

- **Public DateTime** Date
    - Date information for when the patient collected the data.

- **Public int** Total

- Total calories burned at the moment of each data sample

- **Public Guid** PatientDataId
    - Foreign key relationship to the id of the patient data record from the **PatientData** table that this data record corresponds to.

Operations:

- Sets and gets for all attributes.

Constructors:

- **Public** MSBandCalories()
    - Default constructor

3.2.1.13 MSBandGyroscope

This class describes the attributes and construction of a **MSBandGyroscope** object.

<<Entity>>

Attributes:

- **Public int** Id
    - Unique id generated by the system for each Microsoft Band Gyproscope data record.

- **Public DateTime** Date
    - Date information for when the patient collected the data.

- **Public DateTime** Timestamp
    - Timestamp detailing the when in the record each data point was collected.

- **Public float** X
    - X axis of the gyroscope data record.

- **Public float** Y
    - Y axis of the gyroscope data record.

- **Public float** Z
    - Z axis of the gyroscope data record.

- **Public Guid** PatientDataId
    - Foreign key relationship to the id of the patient data record from the **PatientData** table that this data record corresponds to.

Operations:

- Sets and gets for all attributes.

Constructors:

- **Public** MSBandGyroscope()
    - Default constructor

3.2.1.14 MSBandHeartRate

This class describes the attributes and construction of a **MSBandHeartRate** object.

<<Entity>>

Attributes:

- **Public int** Id
    - Unique id generated by the system for each Microsoft Band Heart Rate data record.
- **Public DateTime** Date
    - Date information for when the patient collected the data.
- **Public string** ReadStatus
    - Read status of the heart rate monitor on the device.
- **Public int** HeartRate
    - Heart rate at the moment of each data sample
- **Public Guid** PatientDataId
    - Foreign key relationship to the id of the patient data record from the **PatientData** table that this data record corresponds to.

Operations:

- Sets and gets for all attributes.

Constructors:

- **Public** MSBandHeartRate()
    - Default constructor

3.2.1.15 MSBandDistance

This class describes the attributes and construction of a **MSBandDistance** object.

<<Entity>>

Attributes:

- **Public int** Id
    - Unique id generated by the system for each Microsoft Band Distance data record.

- **Public DateTime** Date
    - Date information for when the patient collected the data.

- **Public string** MotionType
    - Describes the type of motion that is occurring at the time of each data sample

- **Public float** Pace
    - Describes the pace of the movement in min/km at the time of each data sample

- **Public float** Speed
    - Describes the speed of the user in km/hr at the time of each data sample

- **Public string** Total
    - Describes the total distance that has been travelled at the time of each data sample

- **Public Guid** PatientDataId
    - Foreign key relationship to the id of the patient data record from the **PatientData** table that this data record corresponds to.

Operations:

- Sets and gets for all attributes.

Constructors:

- **Public** MSBandDistance()
    - Default constructor

3.2.1.16 MSBandUV

This class describes the attributes and construction of a **MSBandUV** object.

<<Entity>>

Attributes:

- **Public int** Id

  - Unique id generated by the system for each Microsoft Band UV data record.

- **Public DateTime** Date
  - Date information for when the patient collected the data.

- **Public int** UVIndex
  - UV index measured by the device at the moment of each data sample

- **Public Guid** PatientDataId
  - Foreign key relationship to the id of the patient data record from the **PatientData** table that this data record corresponds to.

Operations:

- Sets and gets for all attributes.

Constructors:

- **Public** MSBandUV()
  - Default constructor

3.2.1.17 MSBandTemperature

This class describes the attributes and construction of a **MSBandTemperature** object.

<<Entity>>

Attributes:

- **Public int** Id
  - Unique id generated by the system for each Microsoft Band Calorie data record.

- **Public DateTime** Date
  - Date information for when the patient collected the data.

- **Public int** Temperature
  - Outside temperature (Celsius) measured by the device at the moment of each data sample

- **Public Guid** PatientDataId
  - Foreign key relationship to the id of the patient data record from the **PatientData** table that this data record corresponds to.

Operations:

- Sets and gets for all attributes.

Constructors:

- **Public** MSBandTemperature()
    - Default constructor

3.2.1.18 BasisPeakSummaryData

This class describes the attributes and construction of a **BasisPeakSummaryData** object.

<<Entity>>

Attributes:

- **Public int** Id
    - Unique id generated by the system for each BasicPeak Summary data record.

- **Public DateTime** Date
    - Date information for when the patient collected the data.

- **Public float** Calories
    - Total calories burned at the moment of each data sample

- **Public float** GSR
    - GSR data at the moment of each data sample.

- **Public int** HeartRate
    - Heart rate at the moment of each data sample.

- **Public float** SkinTemp
    - Patient skin temperature at the moment of each data sample.

- **Public int** Steps
    - Number of steps the patient has taken at the moment of each data sample.

- **Public Guid** PatientDataId
    - Foreign key relationship to the id of the patient data record from the **PatientData** table that this data record corresponds to.

Operations:

- Sets and gets for all attributes.

Constructors:

- **Public** MSBandCalories()

  - Default constructor

3.2.1.19 ZephyrAccelerometer

This class describes the attributes and construction of a **ZephyrAccelerometer** object.

<<Entity>>

Attributes:

- **Public int** Id
  - Unique id generated by the system for each Zephyr Accelerometer data record.
- **Public DateTime** Time
  - Date information for when the patient collected the data.
- **Public int** Vertical
  - Vertical axis data from the device accelerometer
- **Public int** Lateral
  - Lateral axis data from the device accelerometer
- **Public int** Sagittal
  - Sagittal axis data from the device accelerometer
- **Public Guid** PatientDataId
  - Foreign key relationship to the id of the patient data record from the **PatientData** table that this data record corresponds to.

Operations:

- Sets and gets for all attributes.

Constructors:

- **Public** ZephyrAccelerometer()
  - Default constructor

3.2.1.20 ZephyrBreathingWaveform

This class describes the attributes and construction of a **ZephyrBreathingWaveform** object.

<<Entity>>

Attributes:

- **Public int** Id
    - Unique id generated by the system for each Zephyr Breathing Waveform data record.

- **Public DateTime** Time
    - Date information for when the patient collected the data.

- **Public int** Data
    - Waveform data point

- **Public Guid** PatientDataId
    - Foreign key relationship to the id of the patient data record from the **PatientData** table that this data record corresponds to.

Operations:

- Sets and gets for all attributes.

Constructors:

- **Public** ZephyrBreathingWaveform ()
    - Default constructor

3.2.1.21 ZephyrECGWaveform

This class describes the attributes and construction of a **ZephyrECGWaveform** object.

<<Entity>>

Attributes:

- **Public int** Id
    - Unique id generated by the system for each Zephyr ECG Waveform data record.

- **Public DateTime** Time
    - Date information for when the patient collected the data.

- **Public int** Data
    - Waveform data point

- **Public Guid** PatientDataId
    - Foreign key relationship to the id of the patient data record from the **PatientData** table that this data record corresponds to.

Operations:

- Sets and gets for all attributes.

Constructors:

- **Public** ZephyrECGWaveform ()
  - Default constructor

3.2.1.22 ZephyrEventData

This class describes the attributes and construction of a **ZephyrEventData** object.

<<Entity>>

Attributes:

- **Public int** Id
  - Unique id generated by the system for each Zephyr Event data record.
- **Public DateTime** Date
  - Date information for when the patient collected the data.
- **Public int** Time
  - Time in ms at the moment of each data sample.
- **Public int** EventCode
  - Event code defined by the device when the event is triggered by the device.
- **Public string** Type
  - Type of event that was triggered by the device
- **Public string** Source
  - Source of the device that caused the event to occur.
- **Public int** EventId
  - Id given to the event.
- **Public string** EventSpecificData
  - Detail of what occurred that triggered the event.
- **Public Guid** PatientDataId
  - Foreign key relationship to the id of the patient data record from the **PatientData** table that this data record corresponds to.

Operations:

- Sets and gets for all attributes.

Constructors:

- **Public** ZephyrEventData ()
  - Default constructor

3.2.1.23 ZephyrSummaryData

This class describes the attributes and construction of a **ZephyrSummaryData** object.

<<Entity>>

Attributes:

- **Public int** Id
  - Unique id generated by the system for each Zephyr Summary data record.

- **Public DateTime** Date
  - Date information for when the patient collected the data.

- **Public int** HeartRate
  - Patient heart rate measured by the device at the moment of each data sample.

- **Public int** BreathingRate
  - Patient breathing rate measured by the device at the moment of each data sample.

- **Public float** SkinTemp
  - Patient skin temperature measured by the device at the moment of each data sample.

- **Public int** Posture
  - Patient posture measured by the device at the moment of each data sample.

- **Public float** Activity
  - Patient activity measured by the device at the moment of each data sample.

- **Public float** PeakAccel
  - Patient peak acceleration measured by the device at the moment of each data sample.

- **Public float** BatteryVolts
  - Current voltage being supplied by the device battery.

- **Public float** BatterLevel
    - Current battery level of the device battery.

- **Public int** BRAmplitude
    - Patient breathing rate amplitude measured by the device at the moment of each data sample.

- **Public int** BRNoise
    - Patient breathing rate noise measured by the device at the moment of each data sample.

- **Public int** BRConfidence
    - Device confidence in the breathing rate measurement.

- **Public int** ECGAmplitude
    - Patient ECG amplitude measured by the device at the moment of each data sample.

- **Public int** ECGNoise
    - Patient ECG noise measured by the device at the moment of each data sample.

- **Public int** HRConfidence
    - Device confidence in the heart rate measurement.

- **Public int** HRV
    - HRV measured by the device at the moment of each data sample.

- **Public int** SystemConfidence
    - Measurement of overall system confidence in the measurements.

- **Public int** GSR
    - GSR measured by the device at the moment of each data sample.

- **Public int** ROGState
    - Some device state at the moment of each data sample

- **Public int** ROGTime
    - An incremental time value representing how many samples occurred inside the current ROGState.

- **Public float** VerticalMin
    - Minimum value of the vertical axis measured by the device accelerometer between summary sample points.

- **Public float** VerticalPeak
  - Maximum value of the vertical axis measured by the device accelerometer between summary sample points.

- **Public float** LateralMin
  - Minimum value of the lateral axis measured by the device accelerometer between summary sample points.

- **Public float** LateralPeak
  - Maximum value of the lateral axis measured by the device accelerometer between summary sample points.

- **Public float** SagittalMin
  - Minimum value of the sagittal axis measured by the device accelerometer between summary sample points.

- **Public float** SagittalPeak
  - Maximum value of the sagittal axis measured by the device accelerometer between summary sample points.

- **Public float** DeviceTemp
  - Device temperature at the moment of each data sample.

- **Public int** StatusInfo
  - A status code provided by the device.

- **Public int** LinkQuality
  - Measurement of the quality of the device link.

- **Public int** RSSI
  - RSSI value measured by the device at the time of each data sample.

- **Public int** TxPower
  - Transmit power measured by the device at the time of each data sample.

- **Public float** CoreTemp
  - Device core temperature at the moment of each data sample.

- **Public int** AuxADC1
  - Auxillary ADC value.

- **Public int** AuxADC2
  - Auxillary ADC value.

- **Public int** AuxADC3
    - Auxillary ADC value.

- **Public Guid** PatientDataId
    - Foreign key relationship to the id of the patient data record from the **PatientData** table that this data record corresponds to.

Operations:

- Sets and gets for all attributes.

Constructors:

- **Public** ZephyrSummaryData ()
    - Default constructor

3.2.2 Enumerations

3.2.2.1 ActivityType

This is the enumeration that contains the various activities a user could be performing while wearing the medical device and capturing data.

<<Enumeration>>

Attributes:

- Home: Cooking
- Home: Doing housework
- Home: Eating
- Home: Watching TV
- Home: Napping
- Home: Sleeping
- Home: Sitting
- Home: Gardening
- Driving
- Using transportation (somebody else is driving)
- Shopping
- Office Work
- Workplace Activities
- Walking
- Running
- Climbing Stairs
- Exercise – Low Intensity
- Exercise – Medium Intensity
- Exercise – High Intensity
- Playing Sports

- Swimming
- Hiking
- Biking
- Miscellaneous

3.2.2.2 AccountStatus

This is the enumeration that contains the different account statuses.

<<Enumeration>>

Attributes:

- Pending
- Active
- Inactive

3.2.2.3 PatientEthnicity

This is the enumeration that contains the ethnicity categories defined in the National Center for Education Statistics that a patient may belong to.

<<Enumeration>>

Attributes:

- Hispanic or Latino
- Not Hispanic or Latino

3.2.2.4 PatientRace

This is the enumeration that contains the racial categories defined in the National Center for Education Statistics that a patient may belong to.

<<Enumeration>>

Attributes:

- American Indian or Alaska Native
- Asian
- Black or African American
- Native Hawaiian or Other Pacific Islander
- White
- Other

3.2.2.5 PatientGender

This is the enumeration that contains the different gender types.

<<Enumeration>>

Attributes:

- Female

- Male

3.2.2.6 Region

This is the enumeration that contains the names of the fifty states that make up the United States plus other major regions in the world.

<<Enumeration>>

Attributes:

- Alabama
- Alaska
- American Samoa
- Arizona
- Arkansas
- California
- Colorado
- Connecticut
- District of Columbia
- Delaware
- Florida
- Georgia
- Guam
- Hawaii
- Idaho
- Illinois
- Indiana
- Iowa
- Kansas
- Kentucky
- Louisiana
- Maine
- Maryland
- Massachusetts
- Michigan
- Minnesota
- Mississippi
- Missouri
- Montana
- Nebraska
- Nevada
- New Hampshire
- New Jersey
- New Mexico
- New York

- North Carolina
- North Marianas Islands
- North Dakota
- Ohio
- Oklahoma
- Oregon
- Pennsylvania
- Puerto Rico
- Rhode Island
- South Carolina
- South Dakota
- Tennessee
- Texas
- Utah
- Vermont
- Virginia
- Virgin Islands
- Washington
- West Virginia
- Wisconsin
- Wyoming
- European Union
- Eastern Europe
- Africa
- South America
- Central America
- Asia
- Oceania
- North America
- The Caribbean

3.2.2.7 DataType

This is the enumeration that contains the different data types stored in the system.

<<Enumeration>>

Attributes:

- HeartRate
- Breathing
- Calorie
- Accelerometer
- Gyroscope
- Pedometer

- Distance
- UV
- Temperature
- Summary
- Event

3.3 Data Access Layer Overview

3.3.1 SQL Client

3.3.1.1 SQLRespository

This class is a repository client that contains all of the functions needed to interface with the system's SQL database. This class will implement the **IDisposable** interface.

<<Repository>>

Attributes:

- **Private DBContext** context
  - Entity Framework database context used to connect to the SQL database.

- **Private bool** disposed
  - Default value is false.
  - Used to signal application garbage collector whether the db context needs to be disposed or already has been.

Operations:

- **Public Patient** GetPatientById(string id)
  - This function is used to get a patient from the database using the patient Id.

- **Public Collection<Patient>** GetPatientsByPhysician(int physicianId)
  - This function is used to get a collection of patients belonging to a physician using the physician's id.

- **Public Collection<Patient>** GetPatients()
  - This function is used to get all the patients in the system.

- **Public bool** InsertPatient(Patient patient)
  - This function is used to add a patient to the database.

- **Public bool** UpdatePatient(Patient patient)
  - This function is used to update a patient record in the database.

- **Public bool** DeletePatient(string patientId)

- This function is used to delete a patient from the database.

- **Public Experiment** GetExperimentById(string id)
  - This function is used to get an experiment from the database using the experiment Id.

- **Public Collection<Experiment>** GetExperimentsByAdmin(int experimentAdminId)
  - This function is used to get a collection of experiments belonging to an experiment administrator using the experiment administrator’s id.

- **Public Collection<Experiment>** GetExperiments()
  - This function is used to get all the experiments in the system.

- **Public bool** InsertExperiment(Experiment experiment)
  - This function is used to add an experiment to the database.

- **Public bool** UpdateExperiment(Experiment experiment)
  - This function is used to update an experiment record in the database.

- **Public bool** DeleteExperiment(int experimentId)
  - This function is used to delete an experiment from the database.

- **Public Physician** GetPhysicianById(int id)
  - This function is used to get a physician from the database using the physician Id.

- **Public Collection<Physician>** GetPhysicians()
  - This function is used to get all the physicians in the system.

- **Public bool** InsertPhysician(Physician physician)
  - This function is used to add a physician to the database.

- **Public bool** UpdatePhysician(Physician physician)
  - This function is used to update a physician record in the database.

- **Public bool** DeletePhysician(int physicianId)
  - This function is used to delete a physician from the database.

- **Public ExperimentAdministrator** GetExperimentAdminById(int id)
  - This function is used to get an experiment administrator from the database using the experiment administrator Id.

- **Public Collection<ExperimentAdministrator>** GetExperimentAdmins()

  - This function is used to get all the experiment administrators in the system.

- **Public bool** InsertExperimentAdmin (ExperimentAdmin experimentAdmin)
  - This function is used to add an experiment administrator to the database.

- **Public bool** UpdateExperimentAdmin (ExperimentAdmin experimentAdmin)
  - This function is used to update an experiment administrator record in the database.

- **Public bool** DeleteExperimentAdmin (int experimentAdminId)
  - This function is used to delete an experiment administrator from the database.

- **Public bool** GetPatientDataById (string patientDataId)
  - This function is used to get a patient data record from the database using its id.

- **Public bool** GetPatientDataByPatient(Patient patient)
  - This function is used to get all the patient data records from the database for a patient.

- **Public bool** InsertPatientData (PatientData patientData)
  - This function is used to add a new patient data record to the database.

- **Public bool** DeletePatientData (string patientDataId)
  - This function is used to delete a patient data record from the database.

- **Public bool** GetActivityById (int activityId)
  - This function is used to get an activity record from the database using its id.

- **Public bool** GetActivitiesByDataRecord(PatientData patientData)
  - This function is used to get all the activity records from the database for a patient data record.

- **Public bool** InsertActivity (Activity activity)
  - This function is used to add a new activity to the database.

- **Public bool** DeleteActivity (int activityId)
  - This function is used to delete an activity from the database.

- **Public bool** GetZephyrAccel (string patientDataId)
  - This function is used to get all the Zephyr accelerometer data records from the database using the corresponding patient data record id.

- **Public bool** InsertZephyrAccel (ZephyrAccelerometer zephyrAccelerometer)
    - This function is used to add a new Zephyr Accelerometer record to the database.

- **Public bool** DeleteZephyrAccel (int zephyrAccelerometerId)
    - This function is used to delete an Zephyr Accelerometer record from the database.

- **Public bool** GetZephyrECG (string patientDataId)
    - This function is used to get all the Zephyr ECG data records from the database using the corresponding patient data record id.

- **Public bool** InsertZephyrECG (ZephyrECGWaveform zephyrECGWaveform)
    - This function is used to add a new Zephyr ECG wareform record to the database.

- **Public bool** DeleteZephyrECG (int zephyrECGWaveformId)
    - This function is used to delete a Zephyr ECG wareform record from the database.

- **Public bool** GetZephyrBreathing (string patientDataId)
    - This function is used to get all the Zephyr Breathing data records from the database using the corresponding patient data record id.

- **Public bool** InsertZephyrBreathing (ZephyrBreathingWaveform zephyrBreathingWaveform)
    - This function is used to add a new Zephyr Breathing wareform record to the database.

- **Public bool** DeleteZephyrBreathing (int zephyrBreathingWaveformId)
    - This function is used to delete a Zephyr ECG wareform record from the database.

- **Public bool** GetZephyrEventData (string patientDataId)
    - This function is used to get all the Zephyr event data records from the database using the corresponding patient data record id.

- **Public bool** InsertZephyrEventData (ZephyrEventData zephyrEventData)
    - This function is used to add a new Zephyr event data record to the database.

- **Public bool** DeleteZephyrEventData (int zephyrEventDataId)

  - This function is used to delete a Zephyr event data record from the database.

- **Public bool** GetZephyrSummaryData (string patientDataId)
  - This function is used to get all the Zephyr summary data records from the database using the corresponding patient data record id.

- **Public bool** InsertZephyrSummaryData (ZephyrSummaryData zephyrSummaryData)
  - This function is used to add a new Zephyr summary data record to the database.

- **Public bool** DeleteZephyrSummaryData (int zephyrSummaryDataId)
  - This function is used to delete a Zephyr summary data record from the database.

- **Public bool** GetBasicPeakSummaryData (string patientDataId)
  - This function is used to get all the Basic Peak summary data records from the database using the corresponding patient data record id.

- **Public bool** InsertBasicPeakSummaryData (BasicPeakSummaryData basisSummaryData)
  - This function is used to add a new Basic Peak summary data record to the database.

- **Public bool** DeleteBasicPeakSummaryData (int basisSummaryDataId)
  - This function is used to delete a Basic Peaksummary data record from the database.

- **Public bool** GetMSBandAccel (string patientDataId)
  - This function is used to get all the MS Band accelerometer data records from the database using the corresponding patient data record id.

- **Public bool** InsertMSBandAccel (MSBandAccelerometer msBandAccelerometer)
  - This function is used to add a new MS Band Accelerometer record to the database.

- **Public bool** DeleteMSBandAccel (int msBandAccelerometerId)
  - This function is used to delete a MS Band Accelerometer record from the database.

- **Public bool** GetMSBandCalories (string patientDataId)
  - This function is used to get all the MS Band calorie data records from the database using the corresponding patient data record id.

- **Public bool** InsertMSBandCalorie (MSBandCalorie msBandCalorie)
  - This function is used to add a new MS Band calorie record to the database.

- **Public bool** DeleteMSBandCalorie (int msBandCalorieId)
  - This function is used to delete a MS Band calorie record from the database.

- **Public bool** GetMSBandGyroscrope (string patientDataId)
  - This function is used to get all the MS Band gyroscope data records from the database using the corresponding patient data record id.

- **Public bool** InsertMSBandGyroscope (MSBandGyroscope msBandGyroscope)
  - This function is used to add a new MS Band gyroscope record to the database.

- **Public bool** DeleteMSBandGyroscope (int msBandGyroscopeId)
  - This function is used to delete a MS Band gyroscope record from the database.

- **Public bool** GetMSBandHeartRate (string patientDataId)
  - This function is used to get all the MS Band heart rate data records from the database using the corresponding patient data record id.

- **Public bool** InsertMSBandHeartRate (MSBandHeartRate msBandHeartRate)
  - This function is used to add a new MS Band heart rate record to the database.

- **Public bool** DeleteMSBandHeartRate (int msBandHeartRateId)
  - This function is used to delete a MS Band heart rate record from the database.

- **Public bool** GetMSBandDistance (string patientDataId)
  - This function is used to get all the MS Band distance data records from the database using the corresponding patient data record id.

- **Public bool** InsertMSBandDistance (MSBandDistance msBandDistance)
  - This function is used to add a new MS Band distance record to the database.

- **Public bool** DeleteMSBandDistance (int msBandDistanceId)
  - This function is used to delete a MS Band distance record from the database.

- **Public bool** GetMSBandPedometer (string patientDataId)
    - This function is used to get all the MS Band pedometer data records from the database using the corresponding patient data record id.

- **Public bool** InsertMSBandPedometer (MSBandPedometer msBandPedometer)
    - This function is used to add a new MS Band pedometer record to the database.

- **Public bool** DeleteMSBandPedometer (int msBandPedometerId)
    - This function is used to delete a MS Band pedometer record from the database.

- **Public bool** GetMSBandUVIndex (string patientDataId)
    - This function is used to get all the MS Band UV index data records from the database using the corresponding patient data record id.

- **Public bool** InsertMSBandUVIndex (MSBandPedometer msBandPedometer)
    - This function is used to add a new MS Band UV index record to the database.

- **Public bool** DeleteMSBandUVIndex (int msBandUVIndexId)
    - This function is used to delete a MS Band UV index record from the database.

- **Public bool** GetMSBandTemperature (string patientDataId)
    - This function is used to get all the MS Band temperature data records from the database using the corresponding patient data record id.

- **Public bool** InsertMSBandTemperature (MSBandTemperature msBandTemperature)
    - This function is used to add a new MS Band temperature record to the database.

- **Public bool** DeleteMSBandTemperature (int msBandTemperatureId)
    - This function is used to delete a MS Band temperature record from the database.

Constructors:

- **Public** SQLRespository ()
    - Default constructor

- **Public** SQLRespository (string url)
    - Constructor that accepts the database url to construct db context with.

- **Public** SQLRespository (DbContext context)
    - Constructor used if db context as already been created.

Appendix A: Data Dictionary

Name/Data Type	**Description**
Address/string	Min: 1, Max: 200 Alphanumeric characters
Age(years)/int	Min: 1, Max 200 Integers
Dates, Times/DateTime	All dates and times are validated against the DateTime data type
Email/string	Min: 1, Max: 100 Alphanumeric characters (format [*@*.*])
Ethnicity/PatientEthnicity	A list will be provided to the user.
FirstName/string	Min: 1, Max: 50 Alphanumeric characters
Gender/PatientGender	Male or Female
Height(inches)/int	Min: 1, Max 200 Integers
ID/GUID	Globally Unique Identifier (system generated)
LastName/string	Min: 1, Max: 50 Alphanumeric characters
***Location/* Region**	A drop down of choices will be provided.
Name/string	Min: 1, Max: 50 Alphanumeric characters
Password/string	Min: 12, Max: 50 Alphanumeric characters; must contain (1 upper case letter, 1 lower case letter, one special character, 1 number)
PhoneNumber/string	Min: 1, Max: 15 Alphanumeric characters limited to digits and the following character set [‘(’, ’)’, ’ ’, ’-’, ‘.’]
Race/PatientRace	A list will be provided to the user.
Username/string	Min: 12, Max: 50 Alphanumeric characters
Weight(pounds)/float	Min: 1.0 Max: 1,000 Floating point numbers

Appendix B: Traceability Matrix

TraceabilityMatrix.xlsx