5-2017

Remote Health Service System based on Struts2 and Hibernate

Asma Saeed
asaeed@stcloudstate.edu

Remote Health Service System Based on Struts2 and Hibernate

by

Asma Saeed

A Starred Paper

Submitted to the Graduate Faculty of

St. Cloud State University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in Computer Science

May, 2017

Abstract

Traditional web applications were developed using Servlets or Java Server Pages (JSP). Maintainability and extensibility became a problem with this approach as the business logic and presentation logic were mixed in a single file. Model View Controller (MVC) is a popular and powerful design pattern for developing web applications as it clearly separates the main responsibilities (business logic, presentation and request handling), which facilitates the scalability and extensibility of the applications. Struts2 framework has gained popularity in the recent times as it realized the MVC architecture. It is an elegant, extensible framework for developing enterprise level Java web applications. Data persistence is a significant task in any web application. Many persistence frameworks have gained importance recently. Hibernate is one such framework that can be easily integrated with Struts2. In this project, a Remote Health Service System is implemented that can be used by patients and medical experts for remote medical diagnosis. Patient registration, medical query submission, query response and query search are the main functionalities of this system. The application is modular, complete and flexible as its architecture is based on the integration of Struts2 and Hibernate frameworks. It has a multi-layered architecture with the Presentation, Controller and Service layers implemented using Struts2 and the Data Access layer implemented using Hibernate.

Table of Contents

CHAPTER 1: INTRODUCTION 10

CHAPTER 2: TECHNOLOGY OVERVIEW 12

2.1 Struts2 Framework 12

2.1.1 Struts2 Architecture 12

2.1.2 Struts2 - Request Processing Workflow 14

2.1.3 Advantages of Struts2 framework 16

2.2 Hibernate 17

2.2.1 Hibernate Architecture 18

2.2.2 Advantages of Hibernate 19

CHAPTER 3: REMOTE HEALTH SERVICE SYSTEM 21

3.1 System Functionality 21

3.2 System Design 23

3.2.1 Use Case Diagrams 23

3.2.2 Statecharts 34

3.2.3 Use Case Realization 38

3.2.4 Class Diagram: 68

3.2.5 Application Architecture 75

3.3 Application Screenshots 77

CHAPTER 4: CONCLUSION 89

References 91

List of Tables

Table 1: Login use case description for the Remote Health Service System 24

Table 2: Create Account use case description for the Remote Health Service System................ 25

Table 3: View Medical Expert Information use case description for the Remote
Health Service System 26

Table 4: Edit Profile use case description for the Remote Health Service System 27

Table 5: Submit Query use case description for the Remote Health Service System 28

Table 6: View Response use case description for the Remote Health Service System................ 29

Table 7: View Past Medical Queries use case description for the Remote Health
Service System.. 30

Table 8: Respond to Query use case description for the Remote Health Service System........... 31

Table 9: Transfer Query use case description for the Remote Health Service System 32

Table 10: Manage Medical Experts use case description for the Remote Health Service System33

Table 11: 'Admin Login – Login success' scenario of Login use case....................................... 40

Table 12: 'Admin Login – Login failure' scenario of Login use case... 41

Table 13: 'Patient Login – Login success' scenario of Login use case....................................... 42

Table 14: 'Patient Login – Login failure' scenario of Login use case... 43

Table 15: 'Medical Expert Login – Login success' scenario of Login use case 44

Table 16: 'Medical Expert Login – Login failure' scenario of Login use case............................ 45

Table 17: 'Create a new account' scenario of Create Account use case 47

Table 18: 'View Medical Expert details' scenario of View Medical Expert
Information use case 48

Table 19: ‘Patient – Edit Profile Information’ scenario of Edit Profile use case 50

Table 20: ‘Medical Expert – Edit Profile Information’ scenario of Edit Profile use case............ 51

Table 21: ‘Submit query to Medical Expert’ scenario of Submit Query use case........................ 53

Table 22: ‘View query response’ scenario of View Response use case...................................... 54

Table 23: ‘Patient – View past queries’ scenario of View Past Medical Queries use case 56

Table 24: ‘Medical Expert – View past queries’ scenario of View Past Medical Queries use case 57

Table 25: ‘Respond to patient query’ scenario of Respond to Query use case............................ 59

Table 26: ‘Medical Expert – Transfer query’ scenario of Transfer Query use case..................... 61

Table 27: ‘Admin – Transfer query’ scenario of Transfer Query use case 62

Table 28: ‘Add a medical expert’ scenario of Manage Medical Experts use case 64

Table 29: ‘Edit a medical expert’ scenario of Manage Medical Experts use case 65

Table 30: ‘Activate a medical expert’ scenario of Manage Medical Experts use case................. 66

Table 31: ‘Deactivate a medical expert’ scenario of Manage Medical Experts use case 68

List of Figures

Figure 1: Struts2 Architecture 13
Figure 2: Request Processing Workflow in Struts2 Framework 15
Figure 3: Role of ORM in Java Application 17
Figure 4: Hibernate Architecture 18
Figure 5: Use Case diagram for the Remote Health Service System 23
Figure 6: Login use case for the Remote Health Service System 24
Figure 7: Create Account use case for the Remote Health Service System 25
Figure 8: View Medical Expert Information use case for the Remote Health Service System 26
Figure 9: Edit Profile use case for the Remote Health Service System 27
Figure 10: Submit Query use case for the Remote Health Service System 28
Figure 11: View Response use case for the Remote Health Service System 28
Figure 12: View Past Medical Queries use case for the Remote Health Service System 30
Figure 13: Respond to Query use case for the Remote Health Service System 31
Figure 14: Transfer Query use case for the Remote Health Service System 32
Figure 15: Manage Medical Experts use case for the Remote Health Service System 33
Figure 16: Complete Statechart of the Remote Health Service System 35
Figure 17: Statechart of the Admin System 36
Figure 18: Statechart of the Patient System 37
Figure 19: Statechart of the Medical Expert System 38
Figure 20: Classes for the Remote Health Service System 39
Figure 21: Interaction of classes for the realization of the Login use case 40

Figure 22: Sequence diagram for the ‘Admin Login – Login success’ scenario 41

Figure 23: Sequence diagram for the ‘Admin Login – Login failure’ scenario 42

Figure 24: Sequence diagram for the ‘Patient Login – Login success’ scenario 43

Figure 25: Sequence diagram for the ‘Patient Login – Login failure’ scenario 44

Figure 26: Sequence diagram for the ‘Medical Expert Login – Login success’ scenario 45

Figure 27: Sequence diagram for the ‘Medical Expert Login – Login failure’ scenario 46

Figure 28: Interaction of classes for the realization of the Create Account use case 46

Figure 29: Sequence diagram for the ‘Create a new account’ scenario 47

Figure 30: Interaction of classes for the realization of the ‘View Medical Expert Information’ use case 48

Figure 31: Sequence diagram for the ‘View Medical Expert details’ scenario 49

Figure 32: Interaction of classes for the realization of the Edit Profile use case 50

Figure 33: Sequence diagram for the ‘Patient – Edit Profile Information’ scenario 51

Figure 34: Sequence diagram for the ‘Medical Expert – Edit Profile Information’ scenario 52

Figure 35: Interaction of classes for the realization of the Submit Query use case 52

Figure 36: Sequence diagram for the ‘Submit query to Medical Expert’ scenario 53

Figure 37: Interaction of classes for the realization of the View Response use case 54

Figure 38: Sequence diagram for the ‘View query response’ scenario 55

Figure 39: Interaction of classes for the realization of the View Past Medical Queries use case 55

Figure 40: Sequence diagram for the ‘Patient – View past queries’ scenario 57

Figure 41: Sequence diagram for the ‘Medical Expert – View past queries’ scenario 58

Figure 42: Interaction of classes for the realization of the Respond to Query use case 59

Figure 43: Sequence diagram for the ‘Respond to patient query’ scenario 60

Figure 44: Interaction of classes for the realization of the Transfer Query use case 61

Figure 45: Sequence diagram for the ‘Medical Expert – Transfer query’ scenario...................... 62

Figure 46: Sequence diagram for the ‘Admin – Transfer query’ scenario 63

Figure 47: Interaction of classes for the realization of the Manage Medical Experts use case 64

Figure 48: Sequence diagram for the ‘Add a medical expert’ scenario....................................... 65

Figure 49: Sequence diagram for the ‘Edit a medical expert’ scenario 66

Figure 50: Sequence diagram for the ‘Activate a medical expert’ scenario 67

Figure 51: Sequence diagram for the ‘Deactivate a medical expert’ scenario 68

Figure 52: Complete Class diagram of the Remote Health Service System................................ 69

Figure 53: Part of the Class diagram for Admin System 70

Figure 54: Part of the Class diagram for Patient System 71

Figure 55: Part of the Class diagram for Medical Expert System 72

Figure 56: Part of the Class diagram showing Model Classes... 74

Figure 57: Architecture of Remote Health Service System.. 75

Figure 58: Login Screen of the Remote Health Service System 77

Figure 59: Create Account Screen of the Remote Health Service System 78

Figure 60: Patient Screen of the Remote Health Service System.. 78

Figure 61: Edit Profile – Patient Screen 79

Figure 62: Edit Profile – Patient Screen (Success message)... 79

Figure 63: View Medical Expert Information – Patient Screen 80

Figure 64: Submit Query – Patient Screen 80

Figure 65: Submit Query – Patient Screen (Success message).. 81

Figure 66: Medical Expert Screen of the Remote Health Service System 81

Figure 67: Edit Profile – Medical Expert Screen.. 82

Figure 68: Respond to Query – Medical Expert Screen 82

Figure 69: Respond to Query – Medical Expert Screen (Success message) 83

Figure 70: View Response – Patient Screen 83

Figure 71: View Past Medical Queries – Patient Screen 84

Figure 72: View Past Medical Queries – Medical Expert Screen 84

Figure 73: Transfer Query – Medical Expert.. 85

Figure 74: Transfer Query – Medical Expert (Success message)... 85

Figure 75: Admin Screen of the Remote Health Service System 86

Figure 76: Manage Medical Experts (Add a new Medical Expert) – Admin Screen 86

Figure 77: Manage Medical Experts (Edit a Medical Expert) – Admin Screen........................... 87

Figure 78: Manage Medical Experts (Deactivate a Medical Expert) – Admin Screen 87

Figure 79: Manage Medical Experts (Activate a Medical Expert) – Admin Screen 88

Figure 80: Transfer Query – Admin Screen 88

Chapter 1: INTRODUCTION

A web application is an application program that can be hosted on a server and clients can request the application services using web browsers. A web application can also be termed as an online application. Initially, the web applications were developed using Servlets or Java Server Pages which consisted of both the Java code (for business logic) and HTML code (for rendering the view part) in a single program. This not only made them unappealing but also caused maintenance and extensibility problems. As the complexity of enterprise level applications started to scale up, this approach proved inefficient. So, the idea of Model View Controller (MVC) was proposed. MVC stands for Model View Controller. MVC pattern is composed of the following three parts:

Model – The Model represents the application’s data.

View – The View is a visual representation of the Model data.

Controller – The Controller responds to the user input and handles the interactions between the Model and the View. It controls the flow of data into the Model object and updates the View whenever the data changes.

MVC supports the separation of concerns as it isolates the business logic from the presentation layer. It promotes better code organization, extensibility, scalability and code re-use. Whenever a user makes a request, the controller receives this request, processes it and prepares the Model data. The View then renders this data and the result (HTML, JSP page) is displayed in the browser.

Struts2 framework gained popularity as it implements the MVC design pattern. Struts2 is an elegant framework that can be used to build enterprise level Java web applications. It can be easily integrated with other frameworks like Spring, Hibernate and Tiles. It is flexible and has support for multiple view technologies like JSP, Velocity and Freemarker. The theme based User Interface (UI) tags help the developers create appealing interfaces with minimum effort.

As databases are used to store data in almost all the web applications, efficiency during the database interaction was a critical factor to improve the overall application performance. To optimize the database interactions, the Object Relational Mapping (ORM) tools were introduced. ORM tools introduce a direct mapping between the Java code objects and the relational database tables. Hibernate is one such ORM tool. It provides Hibernate Query Language (HQL) which is database independent, thus eliminating the need to write database specific queries.

In this project, a Remote Health Service system is implemented based on the ideas presented in the IEEE paper 'Design of Java EE-Based Remote Health Service System' [1]. The system consists of three main modules: *Patient* system, *Medical Expert* system and *Admin* system. Profile management, medical expert information lookup, medical expert selection, query submission, view response and query search are different functionalities available in the *Patient* system. Profile management, query reply, query transfer (to another medical expert) are the functionalities available in the *Medical Expert* system. Medical expert management and query transfer functionality is available in the *Admin* system.

The contents of this document are organized as follows: Chapter 2 gives an overview of the Struts2 and Hibernate frameworks. Chapter 3 presents the system design and functionalities of the Remote Health Service System along with the application screenshots. Chapter 4 concludes the project and provides possible future enhancements. Finally, the project references are listed.

Chapter 2: TECHNOLOGY OVERVIEW

This chapter presents detailed information about the Struts2 and Hibernate frameworks – their overview, architecture and advantages. The first section of this chapter explains the Struts2 framework and the second section explains the Hibernate framework.

2.1 Struts2 Framework

A software framework is a platform for developing software applications that uses the design patterns that are commonly agreed in the industry. It introduces an elegant architectural solution to automate all tedious tasks of application development. Struts2 is an elegant and flexible web application framework based on the MVC (Model View Controller) design pattern. It is used for developing enterprise-level Java web applications. The power of Struts2 lies in its model component by which it can be integrated with other frameworks like Spring, Hibernate etc. Struts2 makes web application development easier and faster by providing an extensible and flexible architecture. The core component of Struts2 is ‘Action’, which is responsible for handling user requests and executing the business logic. The result returned by the Action class helps the framework determine the view page (HTML or JSP page) that will be sent as a response to the user.

2.1.1 Struts2 Architecture

Struts2 framework implements the MVC architecture which divides the application into three components: Model, View and Controller. In Struts2, the Model is implemented by Action, the View is implemented by Result and the Controller is implemented by Filter Dispatcher. The architecture of Struts2 is shown in Figure 1. This figure is adapted from ‘Struts 2 Architecture’ [2].

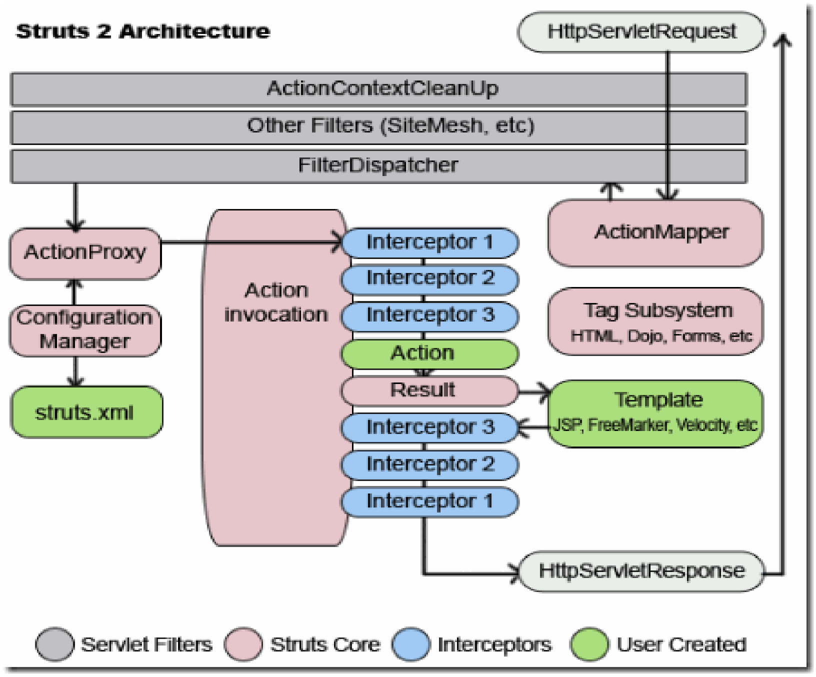


Figure 1: Struts2 Architecture

Struts2 framework consists of the following components [3]:

ActionContextCleanUp filter: This filter is optional and is useful when integrating Struts2 application with other technologies like SiteMash Plugin, Tiles Plugin etc.

ActionMapper: This class contains the action mapping information used to invoke a Struts2 action corresponding to a given request. It helps the FilterDispatcher determine whether to invoke an Action or not. When a request comes, the ActionMapper tries to match an appropriate action invocation request. If no action invocation request matches, it returns null otherwise it returns an ActionMapping that contains details such as the Action class and method to execute.

FilterDispatcher: FilterDispatcher plays the role of controller in Struts2. It maps the incoming requests to actions. It uses the ActionMapper to determine which Struts2 Action should be invoked. If the ActionMapper finds an action corresponding to that request, the FilterDispatcher transfers the control to the ActionProxy.

ActionProxy: This uses the Configuration manager, which is initialized from the struts.xml file to get the information of action and interceptor stack. The ActionProxy creates an ActionInvocation, which processes and invokes the Interceptors (if configured) for the execution of pre-processing logic and finally invokes the Action.

Interceptor: Most of the request processing in Struts2 is done by interceptors. They are called both before and after the Action is invoked and are responsible for executing the pre-processing and post-processing logic. Interceptors perform tasks such as validation, file upload, exception handling, logging etc.

Struts.xml: This is the configuration file of struts2 that contains user defined mappings for actions, interceptors and results.

ActionInvocation: This invokes any interceptors before invoking the Action itself. When the Action executes the business logic, the ActionInvocation determines the appropriate result (JSP or HTML page) associated with the Action result code using the mapping defined in struts.xml file.

Action: The model component of MVC is implemented by Struts2 Action. It generates the result by executing the business logic. The Interceptors are executed again in reverse order to perform any post-processing logic. Once the result is generated, it is responsible to match the view page (JSP or FreeMarker template) which will render the response.

Result: It translates the application data into the JSP code that can be sent to the user’s browser as response.

2.1.2 Struts2 - Request Processing Workflow

The request processing workflow in Struts2 is shown in Figure 2. This figure is adapted from ‘Struts 2 Architecture’ [4].

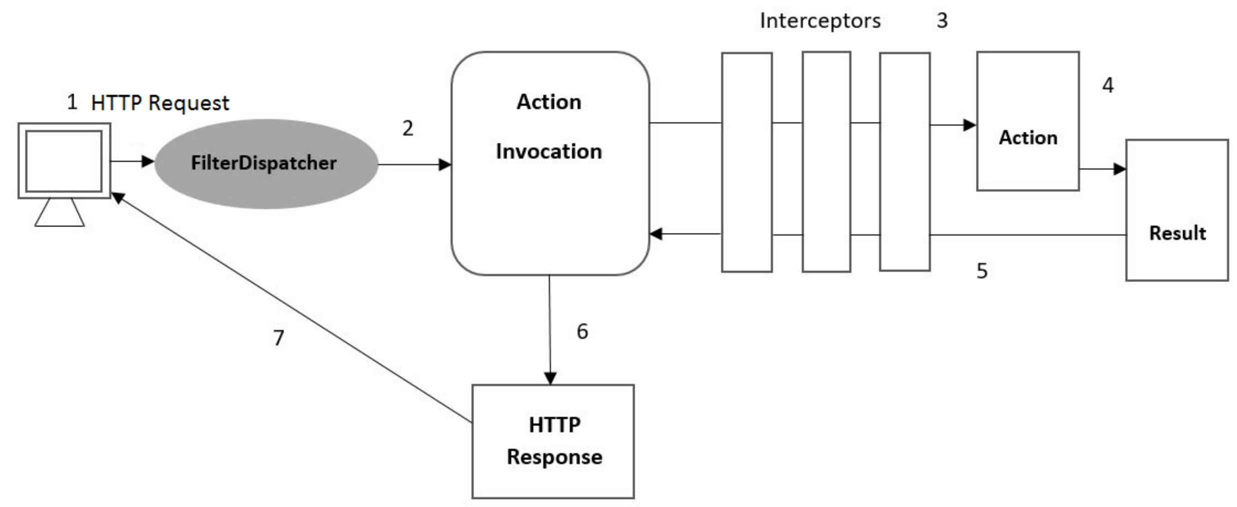


Figure 2: Request Processing Workflow in Struts2 Framework

A request from the user is handled in Struts2 as follows [4]:

1. User sends an HTTP request to the server for a resource (i.e. web page). This request is handled by the FilterDispatcher.
2. FilterDispatcher looks at this request and determines the appropriate Action.
3. Configured interceptors are applied one by one before calling the Action. Interceptors perform functionalities such as validation, file upload, exception handling etc.
4. The selected Action is executed to perform the requested operation and the Result is generated.
5. The interceptors are again applied in the reverse order on the generated result to perform any post-processing logic if required.
6. The result is then rendered in the view to generate the response for the user.
7. This response is sent back to the user.

2.1.3 Advantages of Struts2 framework

Struts2 framework has many advantages which makes web development easier for programmers. Some of these advantages are mentioned below [5]:

- **Simplified Design**: Code is not tightly coupled as Struts2 framework Action is implemented using interfaces and their implementation classes.
- **Plugin support**: Readymade plugins are available for JQuery, JSF, Junit, Spring and Hibernate which can be used to enhance the functionality of Struts2.
- **Lightweight**: ActionForms can be implemented using Plain Old Java Objects (POJO).
- **Adaptability and Flexibility**: Actions can be configured using annotations like *@Action*, which reduces the code complexity.
- **Easy Integration**: Struts2 can be easily integrated with other frameworks like Spring, Hibernate and Tiles etc.
- **Multiple View Options**: Different view technologies like JSP, Velocity and FreeMarker are supported in Struts2 web application.
- **Simplified Testability**: Struts2 Actions can be tested by instantiating the Action and setting the properties using Dependency Injection features.
- **Tag support:** Struts2 provides theme based and Ajax enabled tags which saves time for the developers as they do not need to write lengthy code.
- **AJAX support**: AJAX stands for Asynchronous JavaScript and XML. Struts2 taglib includes several AJAX tags using which a part of the page can be changed without reloading the entire page. This enhances the application performance.

2.2 Hibernate

In high level languages like C# or Java, the information is represented in the form of Objects (instances of Classes). The information in relational databases is represented in the form of tables, so there is a need for direct mapping between the classes and the tables to enhance the application performance. This task is done by Object Relational Mapping (ORM) tools. The role played by an ORM tool is shown in Figure 3. This figure is adapted from 'Introduction to hibernate framework' [6]. ORM maps Java classes to database tables and class member variables to table columns. Each row in the table represents a class object. This mapping allows the developers to perform database operations such as create, update and delete using model objects instead of writing raw SQL queries. Hibernate is one such ORM tool which integrates well with the Struts2 applications.

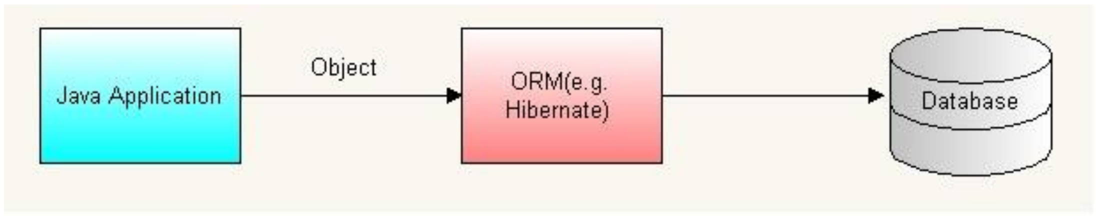


Figure 3: Role of ORM in Java Application

Hibernate is an open source, high-performance object-relational mapping (ORM) and persistence framework for Java applications. It not only maps Java classes to relational database tables and Java data types to SQL data types, but also facilitates the storage and retrieval of data from the database. The main goal of hibernate is to relieve the developers from common data persistence related programming tasks. Hibernate solves the object-relational impedance mismatch problems by replacing direct persistence-related database accesses with high-level object handling functions. Hibernate allows developers to focus more on the business logic by eliminating the need to write complex SQL or JDBC code for data persistence.

2.2.1 Hibernate Architecture

Hibernate acts like a bridge between java application and relational database. Figure 4 shows the architecture of Hibernate framework. This figure is adapted from ‘Hibernate Architecture’ [7].

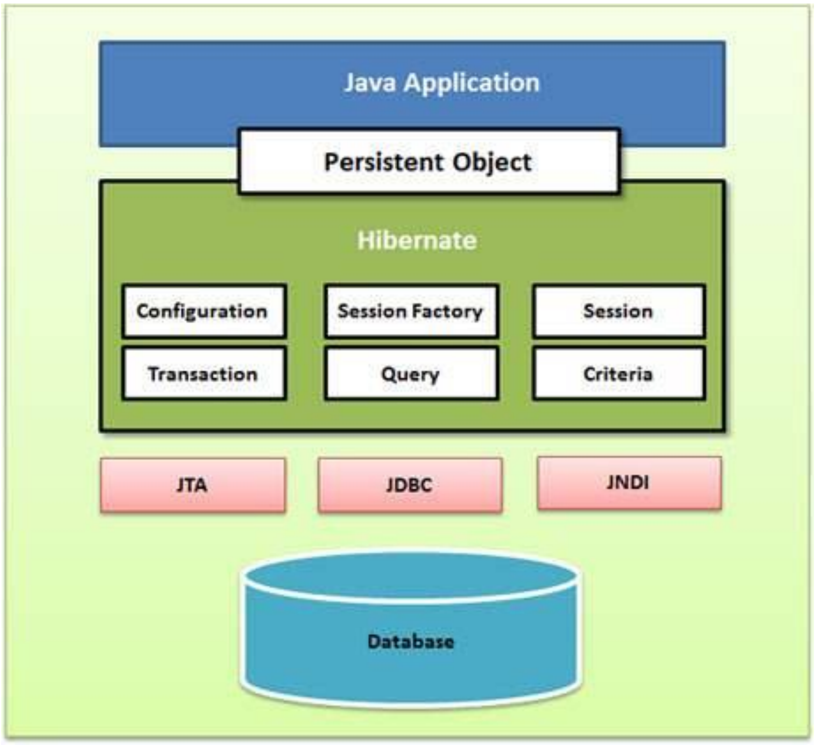


Figure 4: Hibernate Architecture

Hibernate framework consists of the following elements [7][8]:

- **Configuration:** The Configuration object is the first object created in any Hibernate application. It represents the properties file required by Hibernate. The database connection properties and the class mapping setup are defined in the configuration files. The class mapping setup component is responsible for creating the connection between Java classes and database tables.
- **Session Factory:** It is the factory class through which sessions are obtained to perform database operations. The different databases used in an application are mapped in the

Session Factory objects. It is a heavyweight object and is usually built at the time of application start up. One SessionFactory object is required per database. So, an application using multiple databases will have a SessionFactory object for each database.

- **Session:** This object provides an interface between the Java application and the database tables. When the session object is created, a connection is made to the database. It provides methods to insert, update and delete an object.
- **Transaction:** This represents the amount of work that should be committed to the database or rolled back completely to keep the database in a consistent state. This object is optional. Transaction is available under 'org.hibernate.Transaction' package, which provides methods for managing transactions.
- **Query:** This object uses SQL or Hibernate Query Language (HQL) to fetch data from the database.
- **Criteria:** This object is used to execute object oriented queries to retrieve the data from database.

2.2.2 Advantages of Hibernate

The important advantages of Hibernate are mentioned below [9]:

- **HQL**: Hibernate has its own query language 'Hibernate Query Language' which is database independent. This eliminates the need to write database specific queries.
- **Automatic table creation:** Hibernate creates the database tables automatically if they do not exist.
- **Primary Key Generation**: Hibernate can generate the primary keys automatically while storing the data in the database.

- **High Performance**: Hibernate has two levels of cache which results in high performance during the database interactions.
- **Easy Data Fetching**: Hibernate simplifies the fetching of data from multiple tables as there is no need to write complex SQL queries manually.
- **XML Mapping:** The Java classes are mapped to the database tables using XML files. So, any changes in the database can be handled just by changing the XML file properties.

Chapter 3: REMOTE HEALTH SERVICE SYSTEM

This chapter gives a detailed description of the Remote Health Service System. The first section explains the different functionalities of the system. The second section presents the system design – Use Case diagrams, Statecharts, Use Case realization, Sequence diagrams, Class diagrams and Application architecture. The third section presents the application screenshots.

3.1 System Functionality

The Remote Health Service System can be used by patients and medical experts for remote medical diagnosis. This system provides an interactive platform where patients can send queries to medical experts regarding their health symptoms. The medical experts can review their symptoms to diagnose the symptoms and respond to the patients.

A Patient can login into the system and view the specialization and educational background of different medical experts. The Patient can then select a medical expert for treatment, write down the symptoms and submit the query. This query is then sent to the medical expert selected by the patient. The Medical Expert can login into the system and search for any open medical queries assigned to him/her. If there is any open query, he can respond to that query. This response is sent back to the Patient who had sent the query. The Medical Expert can also assign the patient query to another medical expert. The Admin can manage the information of medical experts and can also transfer a patient query from one medical expert to another. The system consists of three main modules – **Patient** system, **Medical Expert** system and **Admin** system.

The *Patient* system provides the following functionalities:

Patient Registration: The Patient can create an account to be able to request the medical services.

Patient Login: The Patient can login into the system using the registered username and password.

Viewing Medical Expert Information: The Patient can select a specialization and view the list of different medical experts in that medical field. The patient can also view the educational background of the medical experts.

Personal Information Management: The Patient can edit his/her profile information like name, address, email, phone number, username and password.

Query Submission: The Patient can write down his/her symptoms and submit the data. This query is then sent to the medical expert selected by the patient.

Viewing Response: The Patient can view the response from a medical expert regarding his/her medical query.

Searching Past Medical Queries: The Patient can search his/her past medical queries. The search can be performed by date range, query type and query status. The patient can then view the query details.

The *Medical Expert* system provides the following functionalities:

Personal Information Management: The Medical Expert can edit his/her profile information like name, address, email, phone number, username and password.

Responding to Query: The Medical Expert can search for open queries assigned to him/her and respond to those queries. This response is sent back to the patient who had sent the query.

Viewing and Reassigning Queries: The Medical Expert can view the past queries of the patients. Also, transfer the query to another medical expert if needed.

The *Administrator* system provides the following functionalities:

Medical Experts Management: The Admin can add or update the information of medical experts. Also, the Admin can activate or deactivate a medical expert in case if that medical expert is on leave or not available for some reason. The Admin can also transfer a patient query from one medical expert to another.

3.2 System Design

3.2.1 Use Case Diagrams

A use case is a graphic representation of the interactions between a user (actor) and the system to achieve a particular goal. The Use Case diagram for the Remote Health Service System is shown in Figure 5. The system has three actors – Admin, Patient and Medical Expert.

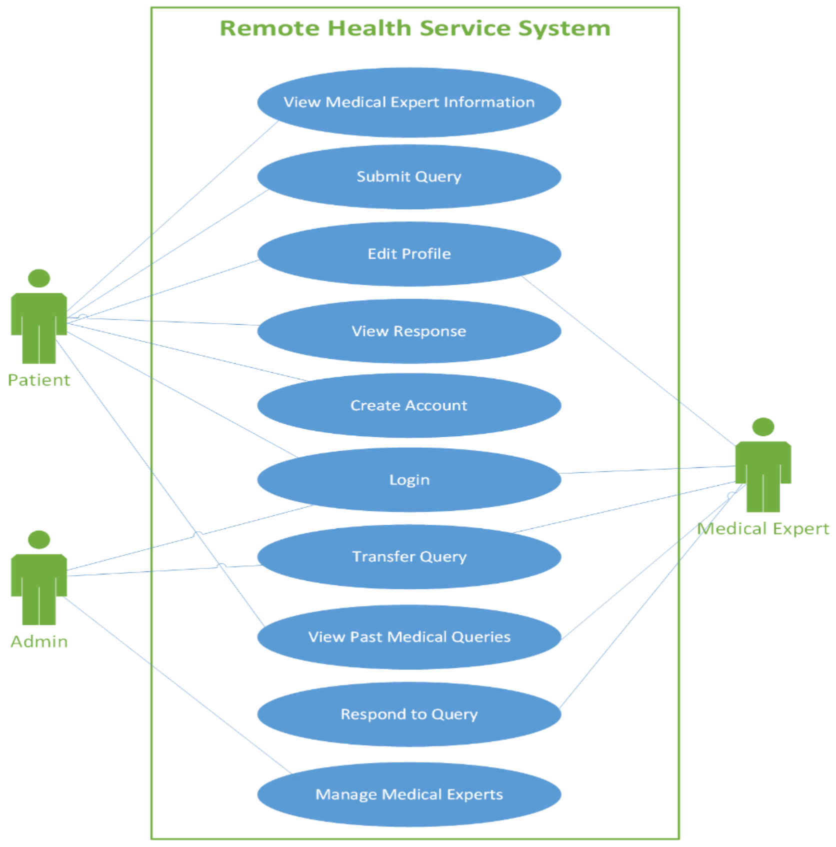


Figure 5: Use Case diagram for the Remote Health Service System

The different Use Cases of the Remote Health Service System are described below:

Login **Usecase:**

Use Case *Login* is shown in Figure 6.

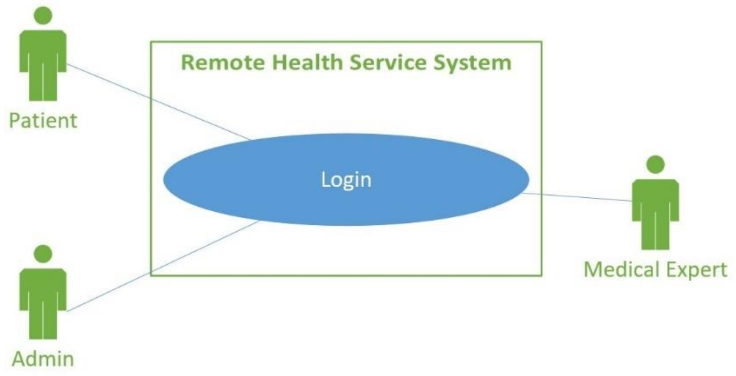


Figure 6: *Login* use case for the Remote Health Service System

Table 1: *Login* use case description for the Remote Health Service System

Brief Description
The *Login* use case enables the Remote Health Service System Admin/Patient/Medical Expert to login into the system.
Step-by-Step Description 1. Enter the Username and Password at the login screen. 2. Validate the Username and Password entered by the user to display Admin/Patient/ Medical Expert screen based on the user type.

Create Account **Usecase:**

Use Case *Create Account* is shown in Figure 7.

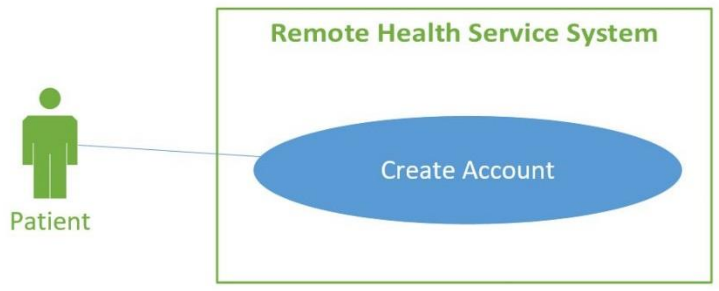


Figure 7: *Create Account* use case for the Remote Health Service System

Table 2: *Create Account* use case description for the Remote Health Service System

Brief Description
The *Create Account* use case enables the Remote Health Service System user to create an account (as Patient) to be able to request the medical services.
Step-by-Step Description 1. Enter the below details to create an account: • First Name • Last Name • Email • Address • Phone Number • Username • Password

View Medical Expert Information **Usecase:**

Use Case *View Medical Expert Information* is shown in Figure 8.

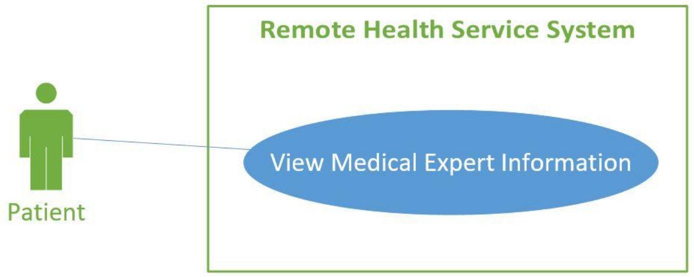


Figure 8: *View Medical Expert Information* use case for the Remote Health Service System

Table 3: *View Medical Expert Information* use case description for the Remote Health Service System

Brief Description The *View Medical Expert Information* use case enables the Remote Health Service System Patient to view the list of medical experts along with their specialization and education background.
Step-by-Step Description 1. A list of medical experts must appear based on the selected medical specialization. 2. For the selected medical expert, information must be displayed with the following attributes: • Education Details • Years of Experience

Edit Profile **Usecase**:

Use Case *Edit Profile* is shown in Figure 9.

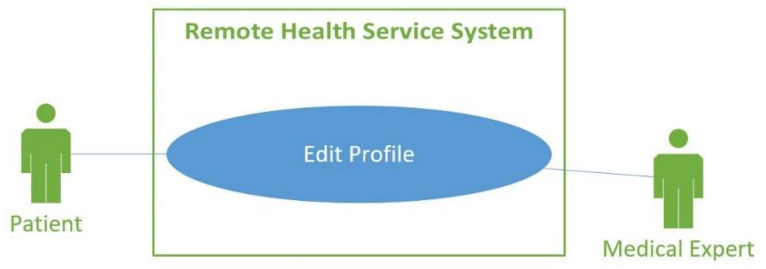


Figure 9: *Edit Profile* use case for the Remote Health Service System

Table 4: *Edit Profile* use case description for the Remote Health Service System

Brief Description The *Edit Profile* use case enables the Remote Health Service System Patient/Medical Expert to update his/her profile information.
Step-by-Step Description 1. Allow updates to the following attributes: • First Name • Last Name • Email • Address • Phone Number • Username • Password

Submit Query **Usecase:**

Use Case *Submit Query* is shown in Figure 10.

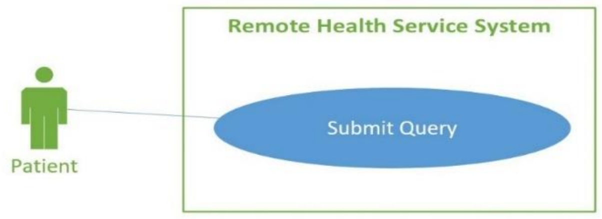


Figure 10: *Submit Query* use case for the Remote Health Service System

Table 5: *Submit Query* use case description for the Remote Health Service System

Brief Description The *Submit Query* use case enables the Remote Health Service System Patient to write down his/her symptoms and submit the query to a Medical Expert.
Step-by-Step Description 1. Enter the medical query and submit it. 2. The submitted query is sent to the Medical Expert selected by the Patient.

View Response **Usecase:**

Use Case *View Response* is shown in Figure 11.

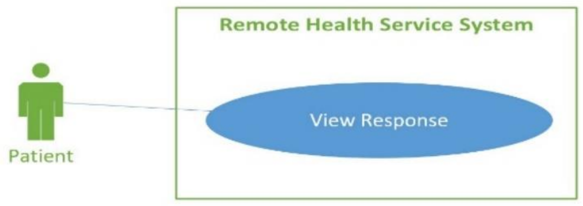


Figure 11: *View Response* use case for the Remote Health Service System

Table 6: *View Response* use case description for the Remote Health Service System

Brief Description

The *View Response* use case enables the Remote Health Service System Patient to view the response from a medical expert regarding his/her query.

Step-by-Step Description

1. Search for medical queries using date range, query type or query status. For each query, the following attributes are displayed:
    - Short Description
    - Assigned to
    - Query Status
    - Query Type
    - Created Date
    - Last Updated by
    - Last Updated Date
2. Select the query to view the response. The query details must be displayed with the following attributes:
    - Assigned To
    - Query Type
    - Created Date
    - Last Updated Date
    - Last Updated by
    - Assigned by

• Comments

View Past Medical Queries **Usecase:**

Use Case *View Past Medical Queries* is shown in Figure 12.

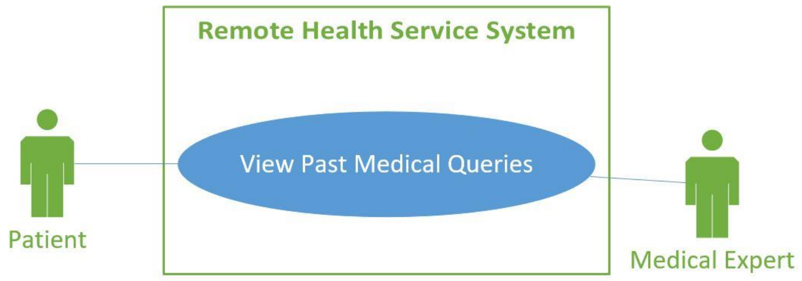


Figure 12: *View Past Medical Queries* use case for the Remote Health Service System

Table 7: *View Past Medical Queries* use case description for the Remote Health Service System

Brief Description The *View Past Medical Queries* use case enables the Remote Health Service System Patient/Medical Expert to view the past medical queries. The Patient can view the queries created by him/her and the Medical Expert can view the queries that were assigned to him/her.
Step-by-Step Description 1. Search for medical queries using date range, query type or query status. 2. The list of queries must be displayed based on the search criteria. For each query, the following attributes are displayed: • Short Description • Assigned to • Query Status

• Query Type • Created Date • Last Updated by • Last Updated Date 3. Selecting a query from the above list should print the detailed information of that query.

Respond to Query **Usecase:**

Use Case *Respond to Query* is shown in Figure 13.

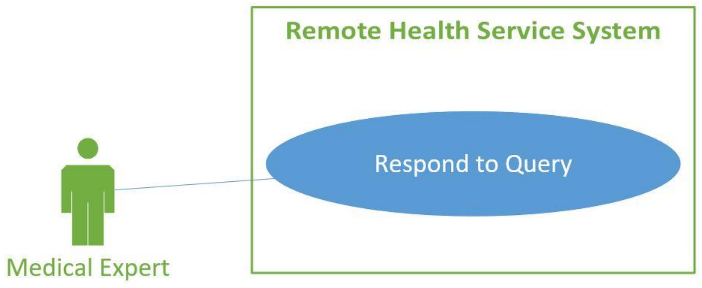


Figure 13: *Respond to Query* use case for the Remote Health Service System

Table 8: *Respond to Query* use case description for the Remote Health Service System

Brief Description
The *Respond to Query* use case enables the Remote Health Service System Medical Expert to respond to the query sent by the Patient.
Step-by-Step Description 1. Select a query from the list of queries with the query status as "Open". 2. Write a response and submit the reply. This reply is sent to the Patient who had submitted the query.

Transfer Query **Usecase:**

Use Case *Transfer Query* is shown in Figure 14.

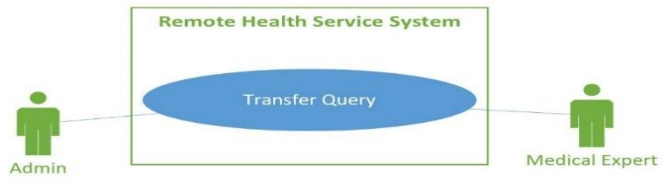


Figure 14: *Transfer Query* use case for the Remote Health Service System

Table 9: *Transfer Query* use case description for the Remote Health Service System

Brief Description The *Transfer Query* use case enables the Remote Health Service System Medical Expert/Admin to transfer the patient query from one Medical Expert to another.
Step-by-Step Description 1. Select the medical query to be assigned to another Medical Expert. 2. Select the medical expert to whom the query should be transferred and save the information. 3. The system transfers the query to the selected medical expert.

Manage Medical Experts **Usecase:**

Use Case *Manage Medical Experts* is shown in Figure 15.

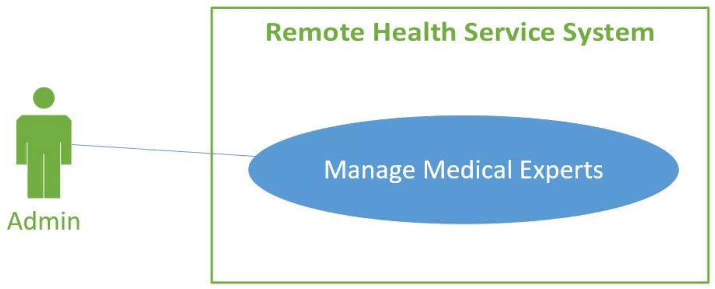


Figure 15: *Manage Medical Experts* use case for the Remote Health Service System

Table 10: *Manage Medical Experts* use case description for the Remote Health Service System

Brief Description

The *Manage Medical Experts* use case enables the Remote Health Service System Admin to create a new Medical Expert and manage the existing Medical Experts.

Step-by-Step Description

1. Create a new Medical Expert.
2. Search for the existing Medical Experts using first name, last name, username, specialization or status. For each Medical Expert, the following attributes are displayed:
    - First Name
    - Last Name
    - Username
    - Specialization
    - Active
3. For the existing Medical Experts, the following information can be updated:
    - First Name

- Last Name
- Email Id
- Address
- Phone Number
- User Name
- Password
- Specialty
- Years of Experience
- Graduation Credentials
- Active or Inactive

3.2.2 Statecharts

A Statechart is used to describe the behavior of a system. It shows the flow of control from one state to another. It reflects all the operations performed by or to that system, indicating the events that cause the transition from state to state. The Statechart of the complete Remote Health Service System is shown in Figure 16. The system is represented as a combination of three modules – *Admin* system, *Patient* system and *Medical Expert* system.

The solid circle on the top left represents the initial state of the Statechart. The arrow from the initial state leads to the state labeled **Remote Health Service System Event Loop**. From this state, the user can login into the system, create a new account or quit the system. The trigger 'application's main screen displayed' takes the Remote Health Service System Event Loop to the state 'Providing Login Details'. In this state, the user can login into the system by entering his/her username and password. The trigger 'application's main screen displayed and new user signup selected' takes the Remote Health Service System Event Loop to the state 'Creating an Account' where the user can create a new account. The trigger 'admin credentials entered' leads to the state 'Performing Admin Events'. In this state, user can perform events like *Manage Medical Experts* and *Transfer Query*. The trigger 'patient credentials entered' leads to the state 'Performing Patient Events' where operations like *View Medical Expert Information, Edit Profile, Submit Query, View Response* and *View Past Medical Queries* can be performed. The trigger 'medical expert credentials entered' leads to the state 'Performing Medical Expert Events' where user can perform operations like *Edit Profile, View Past Medical Queries, Respond to Query* and *Transfer Query.*

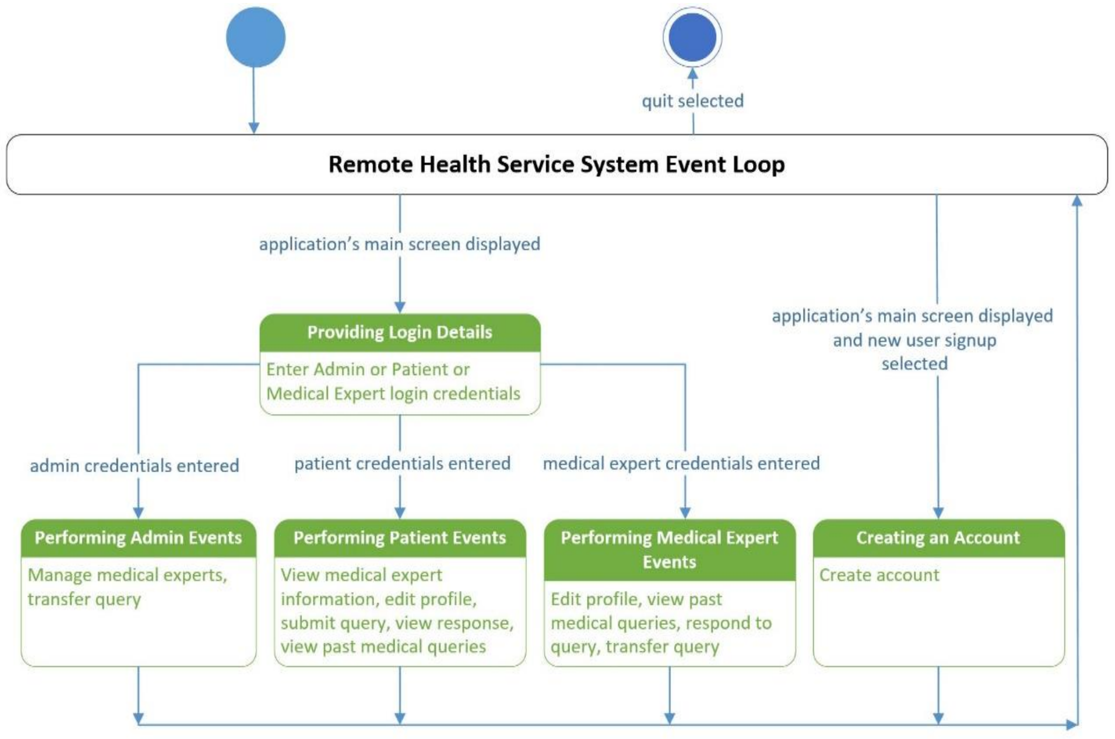

quit selected
Remote Health Service System Event Loop
application's main screen displayed
Providing Login Details
Enter Admin or Patient or Medical Expert login credentials
application's main screen displayed and new user signup selected
admin credentials entered
patient credentials entered
medical expert credentials entered
Performing Admin Events
Manage medical experts, transfer query
Performing Patient Events
View medical expert information, edit profile, submit query, view response, view past medical queries
Performing Medical Expert Events
Edit profile, view past medical queries, respond to query, transfer query
Creating an Account
Create account

Figure 16: Complete Statechart of the Remote Health Service System

The Statechart for the Admin system is shown in Figure 17. The trigger 'manage user selected' leads the system to the state 'Managing Medical Experts'. In this state, the Admin can add/edit/activate/deactivate a medical expert. The trigger 'reassign query selected' leads the system to the state 'Transferring Query' where the Admin can transfer a patient query from one medical expert to another.

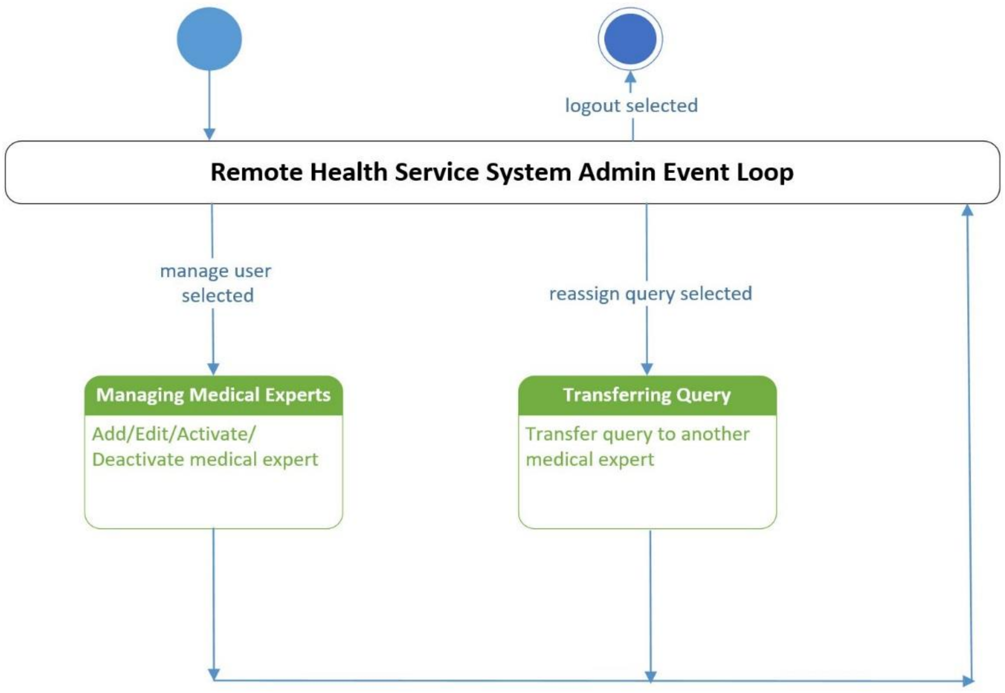

logout selected
Remote Health Service System Admin Event Loop
manage user selected
reassign query selected
Managing Medical Experts
Add/Edit/Activate/ Deactivate medical expert
Transferring Query
Transfer query to another medical expert

Figure 17: Statechart of the Admin System

The Statechart for the Patient System is shown in Figure 18. The trigger 'medical specialization selected' leads the system to the state 'Viewing Medical Expert Information'. In this state, the Patient can view the list of medical experts and their educational background based on the selected specialization. The trigger 'profile management selected' leads to the state 'Editing Profile' where the Patient can edit his/her profile information. The trigger 'submit query selected' takes the system to the state 'Submitting Query' where the Patient can submit a query to a medical expert. The trigger 'view query response selected' leads to the state 'Viewing Response' where the Patient can view the response sent by the medical expert regarding the query. The trigger 'query search selected' leads the system to the state 'Viewing Past Medical Queries' where the Patient can search for past medical queries and view details of the queries.

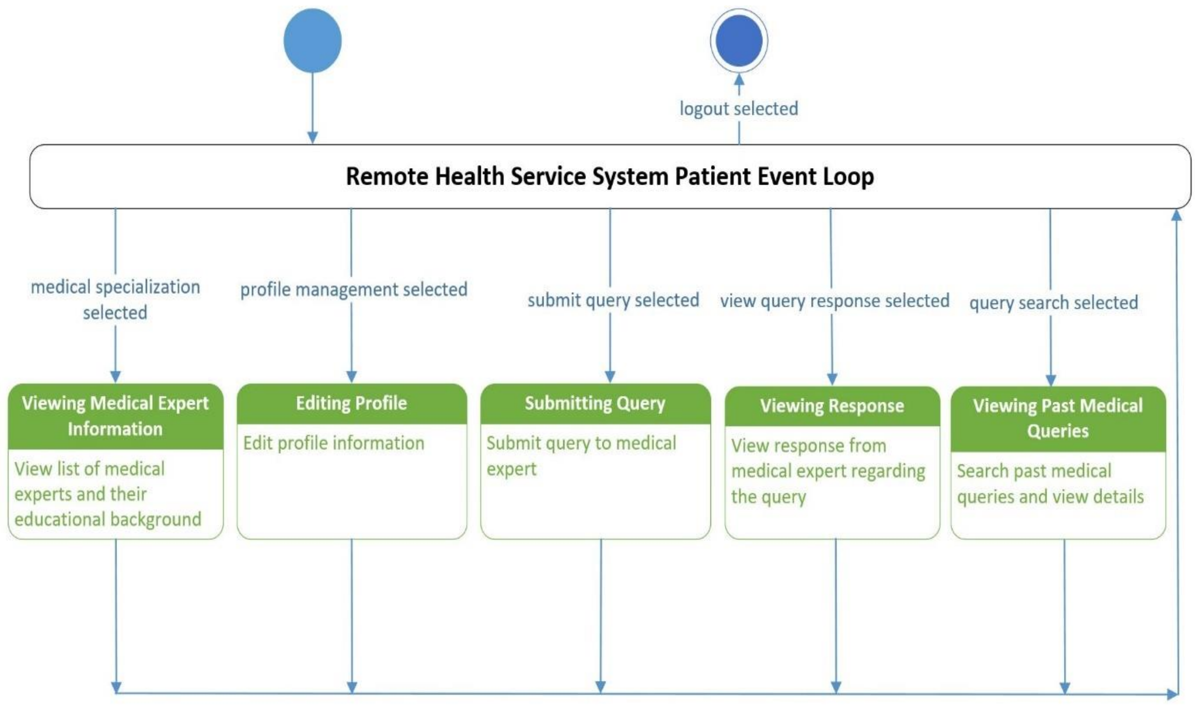

logout selected
Remote Health Service System Patient Event Loop
medical specialization selected
profile management selected
submit query selected
view query response selected
query search selected
Viewing Medical Expert Information
View list of medical experts and their educational background
Editing Profile
Edit profile information
Submitting Query
Submit query to medical expert
Viewing Response
View response from medical expert regarding the query
Viewing Past Medical Queries
Search past medical queries and view details

Figure 18: Statechart of the Patient System

The Statechart for the Medical Expert System is shown in Figure 19. The trigger 'profile management selected' leads the system to the state 'Editing Profile'. In this state, the Medical Expert can edit his/her profile information. The trigger 'reassign query selected' leads to the state 'Transferring Query' where the Medical Expert can transfer a patient query to another Medical Expert. The trigger 'respond to query selected' leads the system to the state 'Responding to Query' where the Medical Expert can respond to a patient query. The trigger 'query search selected' leads the system to the state 'Viewing Past Medical Queries' where the Medical Expert can search for past medical queries and view details of the queries.

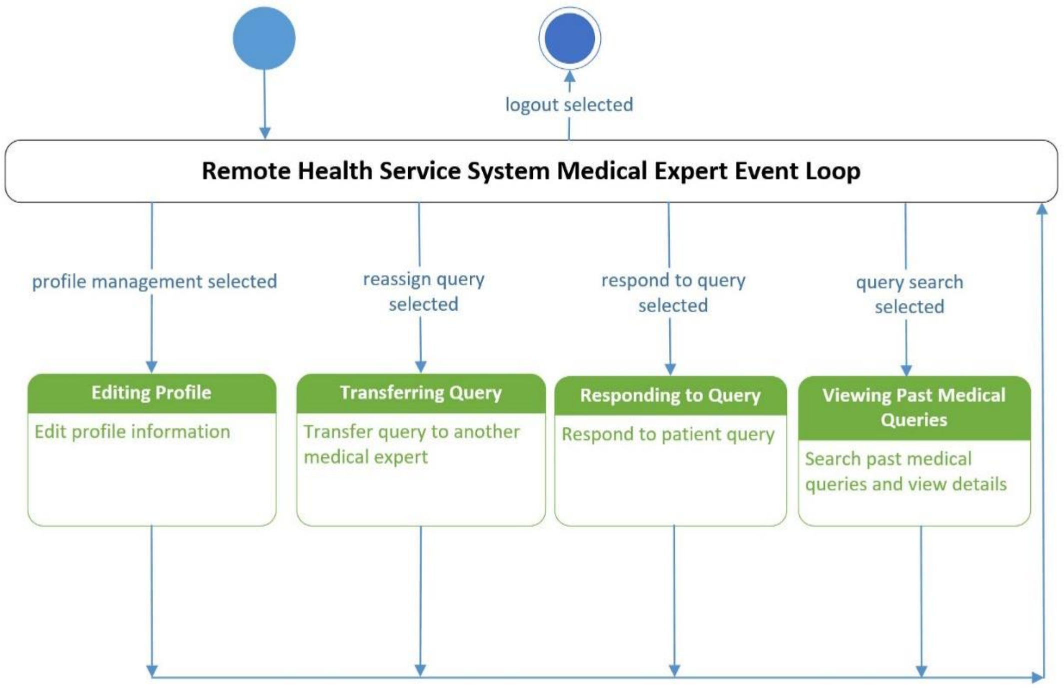
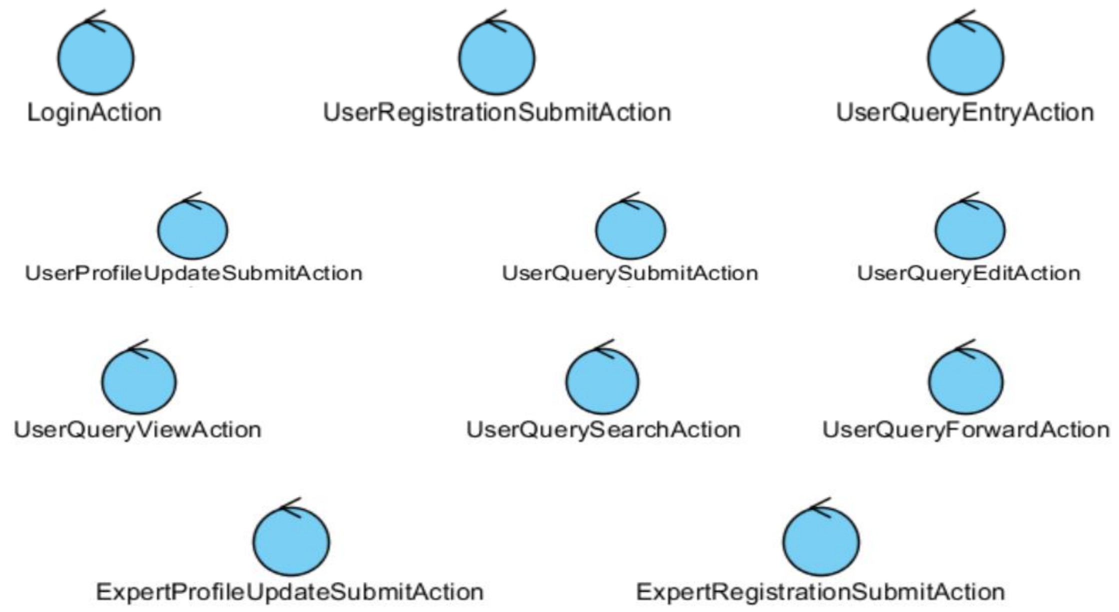

logout selected
Remote Health Service System Medical Expert Event Loop
profile management selected
reassign query selected
respond to query selected
query search selected
Editing Profile
Edit profile information
Transferring Query
Transfer query to another medical expert
Responding to Query
Respond to patient query
Viewing Past Medical Queries
Search past medical queries and view details

Figure 19: Statechart of the Medical Expert System

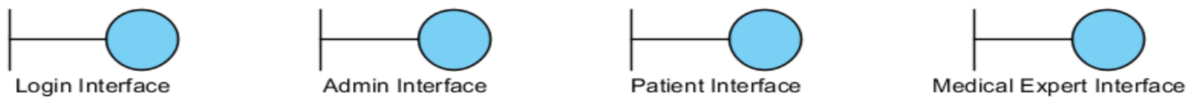
3.2.3 Use Case Realization

Use Case realization is the process of extending and refining use cases. It describes how a particular use case is accomplished in terms of classes and collaborating objects. The use case realization of each use case in the Remote Health Service System is done using class diagram and sequence diagram. A class diagram shows the interrelationships and interaction between the classes. A sequence diagram depicts the objects and the messages sent between them in the realization of a specific scenario of the use case. The different classes for the Remote Health Service System are shown in Figure 20. These consist of Control Classes, Boundary Classes and Entity Classes. A *Control Class* models complex computations and algorithms. A *Boundary Class* models the interaction between the system and its actors. An *Entity Class* models long lived information.

Control Classes



Boundary Classes



Entity Classes

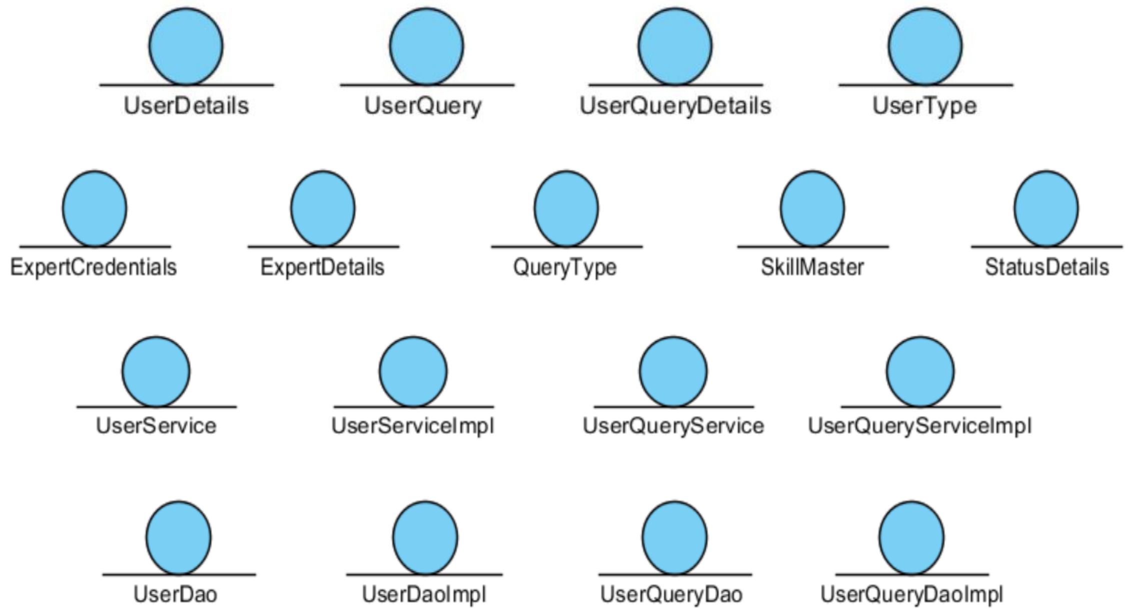


Figure 20: Classes for the Remote Health Service System

Use Case Realization for the *Login* use case

The classes that participate in the realization of the *Login* use case and their relationships are shown in Figure 21.

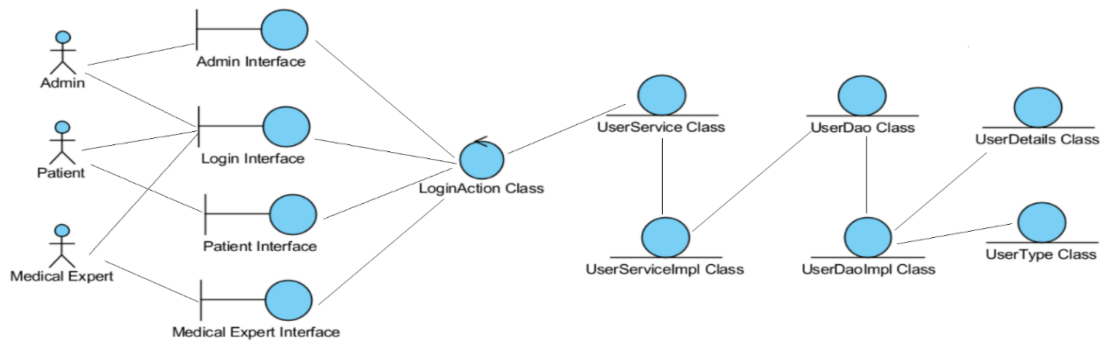


Figure 21: Interaction of classes for the realization of the *Login* use case

Scenarios and Sequence diagrams for the *Login* use case:

Scenario 1 (Admin Login – Login success)

The scenario 'Admin Login – Login success' of *Login* use case is shown in Table 11.

Table 11: 'Admin Login – Login success' scenario of *Login* use case

The Remote Health Service System Admin wants to login into the system. 1. The Admin enters the username and password at the login screen. 2. The system validates the username and password entered by the user. 3. The system displays the Admin screen options after successful authentication.

The sequence diagram of the realization of the 'Admin Login – Login success' scenario of the *Login* use case is shown in Figure 22.

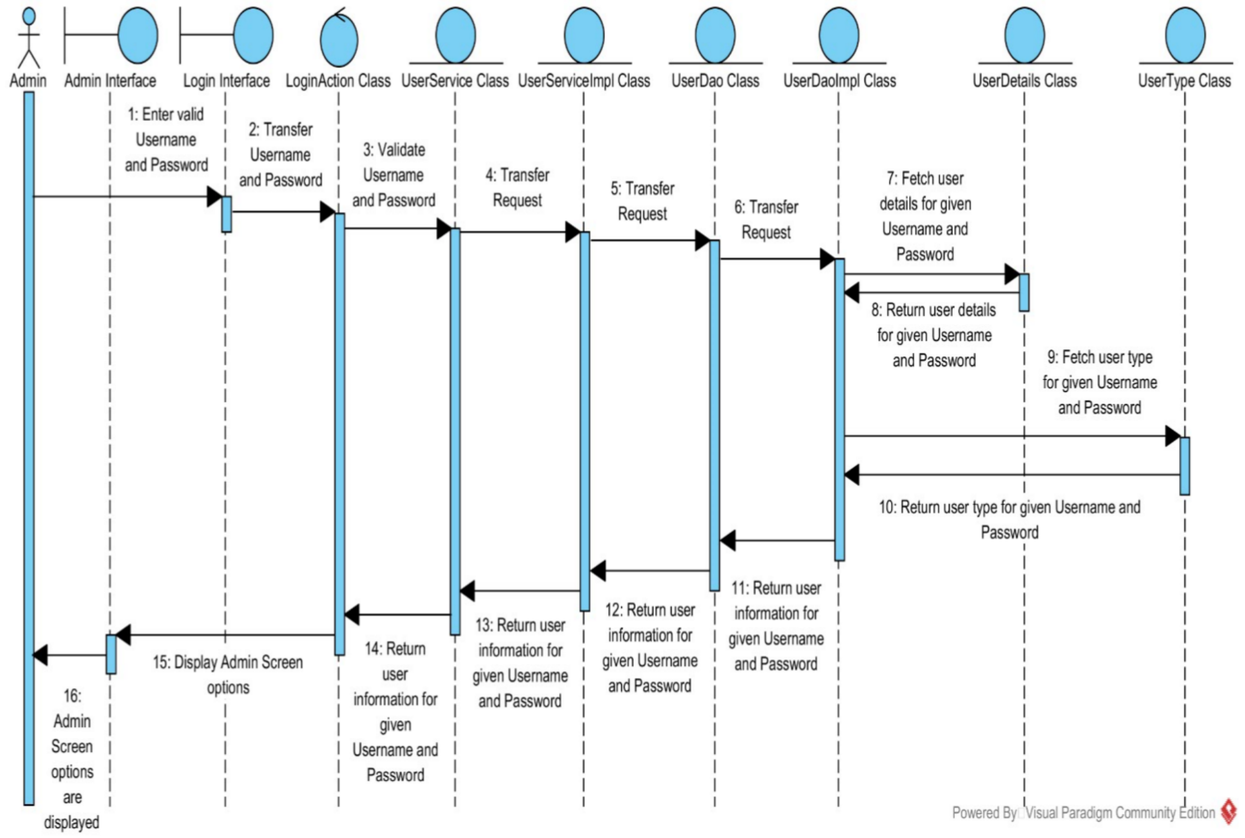

Admin
Admin Interface
Login Interface
LoginAction Class
UserService Class
UserServiceImpl Class
UserDao Class
UserDaoImpl Class
UserDetails Class
UserType Class
1: Enter valid Username and Password
2: Transfer Username and Password
3: Validate Username and Password
4: Transfer Request
5: Transfer Request
6: Transfer Request
7: Fetch user details for given Username and Password
8: Return user details for given Username and Password
9: Fetch user type for given Username and Password
10: Return user type for given Username and Password
11: Return user information for given Username and Password
12: Return user information for given Username and Password
13: Return user information for given Username and Password
14: Return user information for given Username and Password
15: Display Admin Screen options
16: Admin Screen options are displayed
Powered By Visual Paradigm Community Edition

Figure 22: Sequence diagram for the 'Admin Login – Login success' scenario

Scenario 2 (Admin Login – Login failure)

The scenario 'Admin Login – Login failure' of *Login* use case is shown in Table 12.

Table 12: 'Admin Login – Login failure' scenario of *Login* use case

The Remote Health Service System Admin wants to login into the system. 1. The Admin enters the username and password at the login screen. 2. The system validates the username and password entered by the user. 3. The system displays an error message after authentication failure.

The sequence diagram of the realization of the 'Admin Login – Login failure' scenario of the *Login* use case is shown in Figure 23.

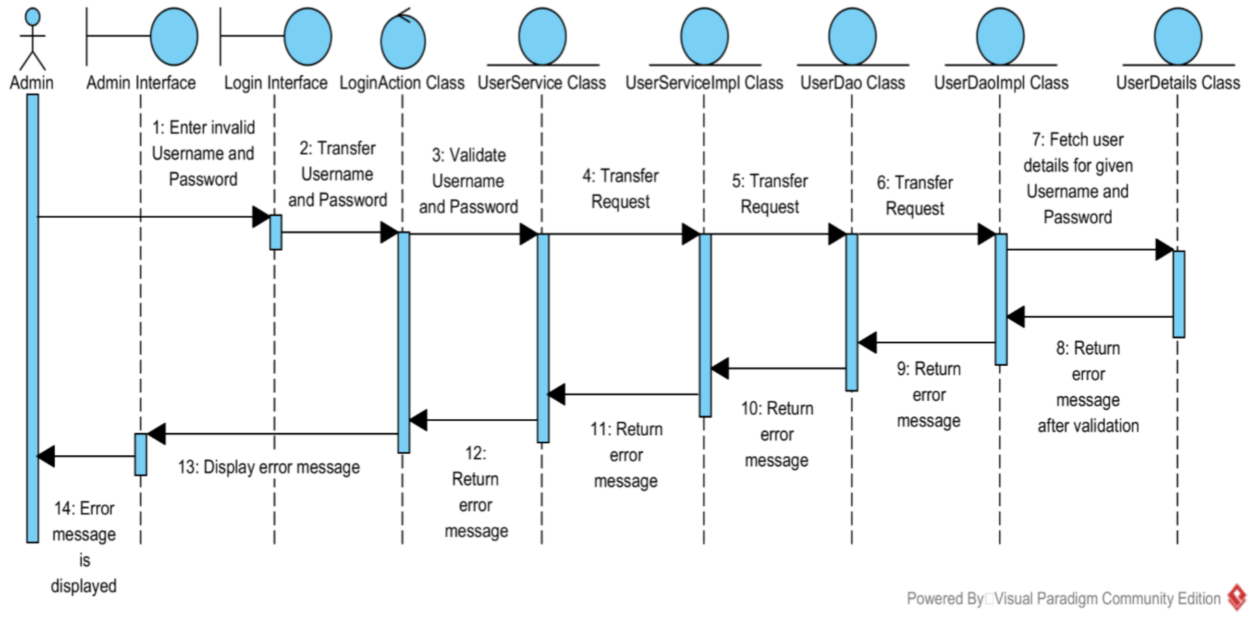

Admin
Admin Interface
Login Interface
LoginAction Class
UserService Class
UserServiceImpl Class
UserDao Class
UserDaoImpl Class
UserDetails Class
1: Enter invalid Username and Password
2: Transfer Username and Password
3: Validate Username and Password
4: Transfer Request
5: Transfer Request
6: Transfer Request
7: Fetch user details for given Username and Password
8: Return error message after validation
9: Return error message
10: Return error message
11: Return error message
12: Return error message
13: Display error message
14: Error message is displayed
Powered By Visual Paradigm Community Edition

Figure 23: Sequence diagram for the 'Admin Login – Login failure' scenario

Scenario 3 (Patient Login – Login success)

The scenario 'Patient Login – Login success' of *Login* use case is shown in Table 13.

Table 13: 'Patient Login – Login success' scenario of *Login* use case

The Remote Health Service System Patient wants to login into the system. 1. The Patient enters the username and password at the login screen. 2. The system validates the username and password entered by the user. 3. The system displays the Patient screen options after successful authentication.

The sequence diagram of the realization of the 'Patient Login – Login success' scenario of the *Login* use case is shown in Figure 24.

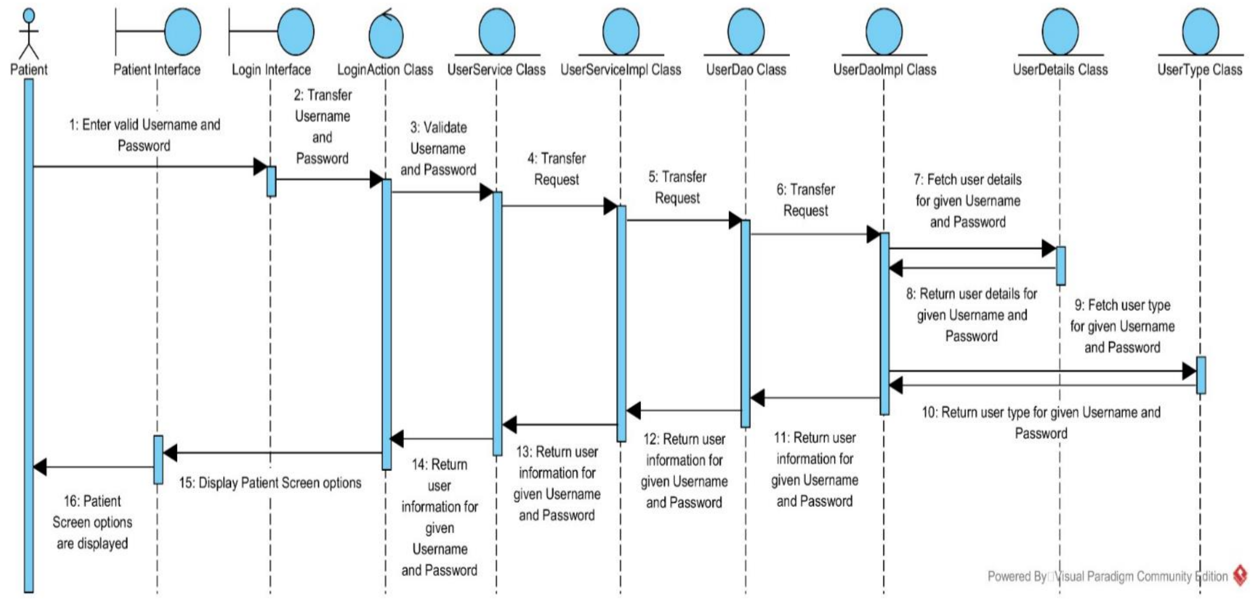

Patient
Patient Interface
Login Interface
LoginAction Class
UserService Class
UserServiceImpl Class
UserDao Class
UserDaoImpl Class
UserDetails Class
UserType Class
1: Enter valid Username and Password
2: Transfer Username and Password
3: Validate Username and Password
4: Transfer Request
5: Transfer Request
6: Transfer Request
7: Fetch user details for given Username and Password
8: Return user details for given Username and Password
9: Fetch user type for given Username and Password
10: Return user type for given Username and Password
11: Return user information for given Username and Password
12: Return user information for given Username and Password
13: Return user information for given Username and Password
14: Return user information for given Username and Password
15: Display Patient Screen options
16: Patient Screen options are displayed
Powered By Visual Paradigm Community Edition

Figure 24: Sequence diagram for the ‘Patient Login – Login success’ scenario

Scenario 4 (Patient Login – Login failure)

The scenario ‘Patient Login – Login failure’ of *Login* use case is shown in Table 14.

Table 14: ‘Patient Login – Login failure’ scenario of *Login* use case

The Remote Health Service System Patient wants to login into the system. 1. The Patient enters the username and password at the login screen. 2. The system validates the username and password entered by the user. 3. The system displays an error message after authentication failure.

The sequence diagram of the realization of the ‘Patient Login – Login failure’ scenario of the *Login* use case is shown in Figure 25.

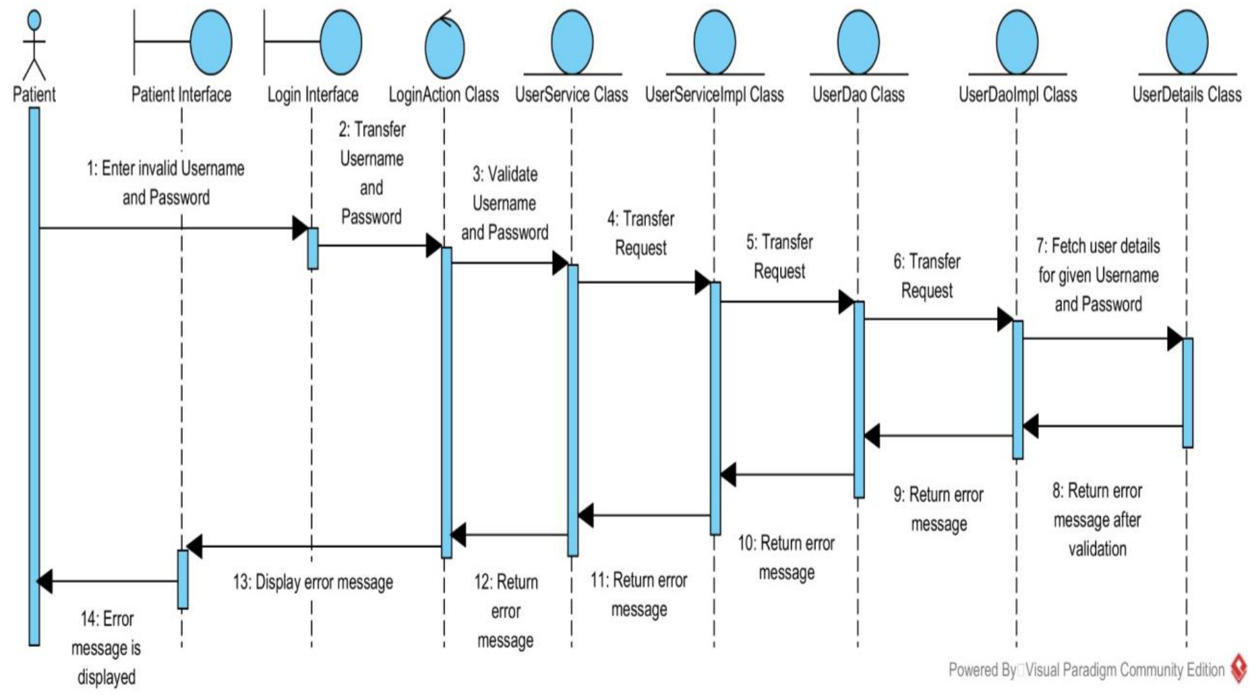

Patient
Patient Interface
Login Interface
LoginAction Class
UserService Class
UserServiceImpl Class
UserDao Class
UserDaoImpl Class
UserDetails Class
1: Enter invalid Username and Password
2: Transfer Username and Password
3: Validate Username and Password
4: Transfer Request
5: Transfer Request
6: Transfer Request
7: Fetch user details for given Username and Password
8: Return error message after validation
9: Return error message
10: Return error message
11: Return error message
12: Return error message
13: Display error message
14: Error message is displayed
Powered By Visual Paradigm Community Edition

Figure 25: Sequence diagram for the ‘Patient Login – Login failure’ scenario

Scenario 5 (Medical Expert Login – Login success)

The scenario ‘Medical Expert Login – Login success’ of *Login* use case is shown in Table 15.

Table 15: ‘Medical Expert Login – Login success’ scenario of *Login* use case

The Remote Health Service System Medical Expert wants to login into the system. 1. The Medical Expert enters the username and password at the login screen. 2. The system validates the username and password entered by the user. 3. The system displays the Medical Expert screen options after successful authentication.

The sequence diagram of the realization of the ‘Medical Expert Login – Login success’ scenario of the *Login* use case is shown in Figure 26.

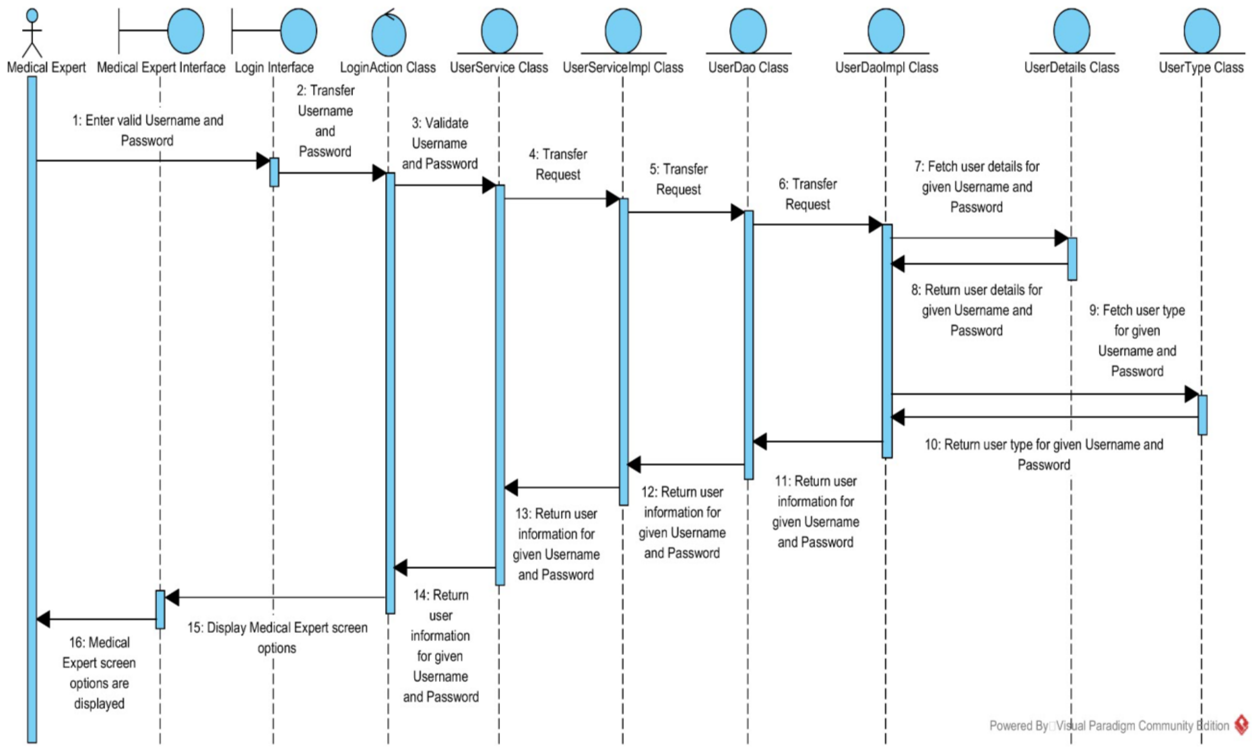

Medical Expert
Medical Expert Interface
Login Interface
LoginAction Class
UserService Class
UserServiceImpl Class
UserDao Class
UserDaoImpl Class
UserDetails Class
UserType Class
1: Enter valid Username and Password
2: Transfer Username and Password
3: Validate Username and Password
4: Transfer Request
5: Transfer Request
6: Transfer Request
7: Fetch user details for given Username and Password
8: Return user details for given Username and Password
9: Fetch user type for given Username and Password
10: Return user type for given Username and Password
11: Return user information for given Username and Password
12: Return user information for given Username and Password
13: Return user information for given Username and Password
14: Return user information for given Username and Password
15: Display Medical Expert screen options
16: Medical Expert screen options are displayed
Powered By Visual Paradigm Community Edition

Figure 26: Sequence diagram for the 'Medical Expert Login – Login success' scenario

Scenario 6 (Medical Expert Login – Login failure)

The scenario 'Medical Expert Login – Login failure' of *Login* use case is shown in Table 16.

Table 16: 'Medical Expert Login – Login failure' scenario of *Login* use case

The Remote Health Service System Medical Expert wants to login into the system. 1. The Medical Expert enters the username and password at the login screen. 2. The system validates the username and password entered by the user. 3. The system displays an error message after authentication failure.

The sequence diagram of the realization of the 'Medical Expert Login – Login failure' scenario of the *Login* use case is shown in Figure 27.

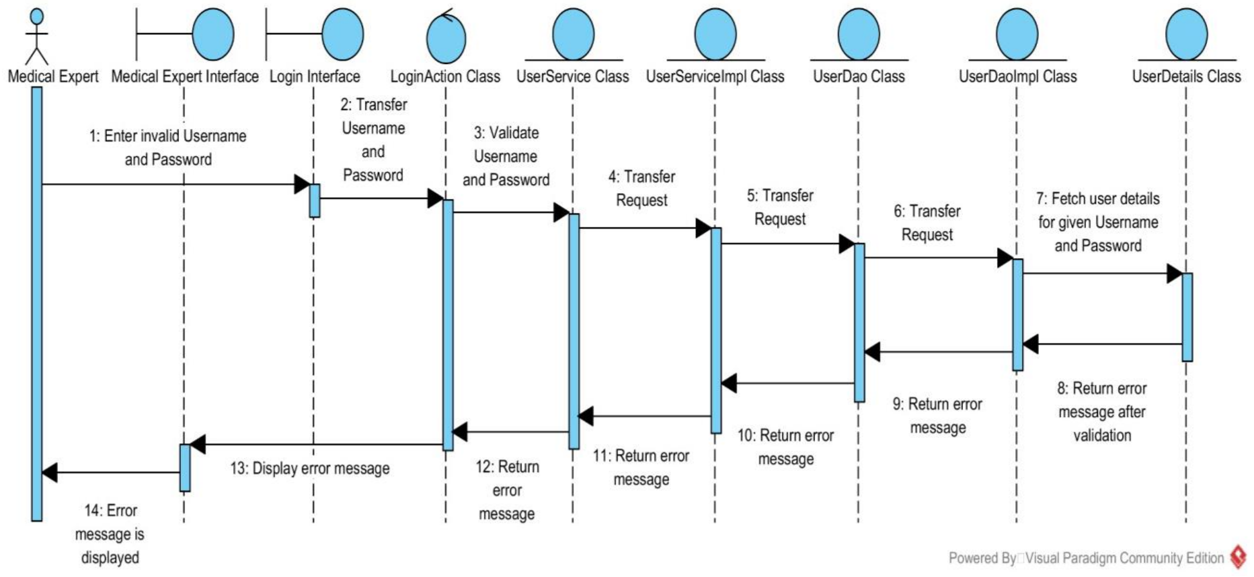


Figure 27: Sequence diagram for the ‘Medical Expert Login – Login failure’ scenario

Use Case Realization for the ***Create Account*** **use case**

The classes that participate in the realization of the *Create Account* use case and their relationships are shown in Figure 28.

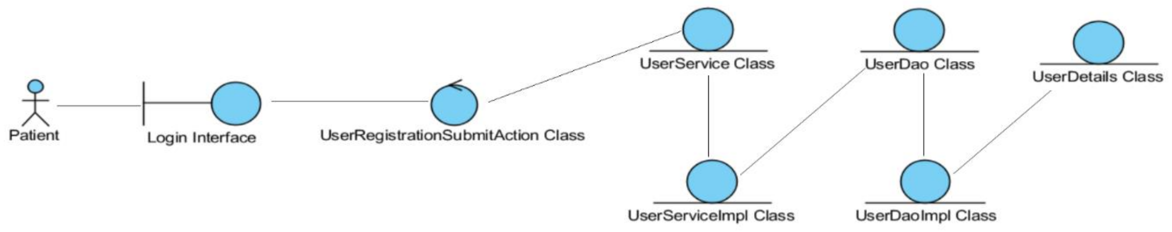


Figure 28: Interaction of classes for the realization of the *Create Account* use case

Scenario and Sequence diagram for the ***Create Account’*** **use case:**

Scenario 1 (Create a new account)

The scenario ‘Create a new account’ of *Create Account* use case is shown in Table 17.

Table 17: ‘Create a new account’ scenario of *Create Account* use case

The Remote Health Service System Patient wants to create a user account. 1. The Patient selects ‘New User Signup’ option at the application screen. 2. The Patient enters his/her details in the registration form and submits the data. 3. The system creates the user account and displays the success message to the user.

The sequence diagram of the realization of the ‘Create a new account’ scenario of the *Create Account* use case is shown in Figure 29.

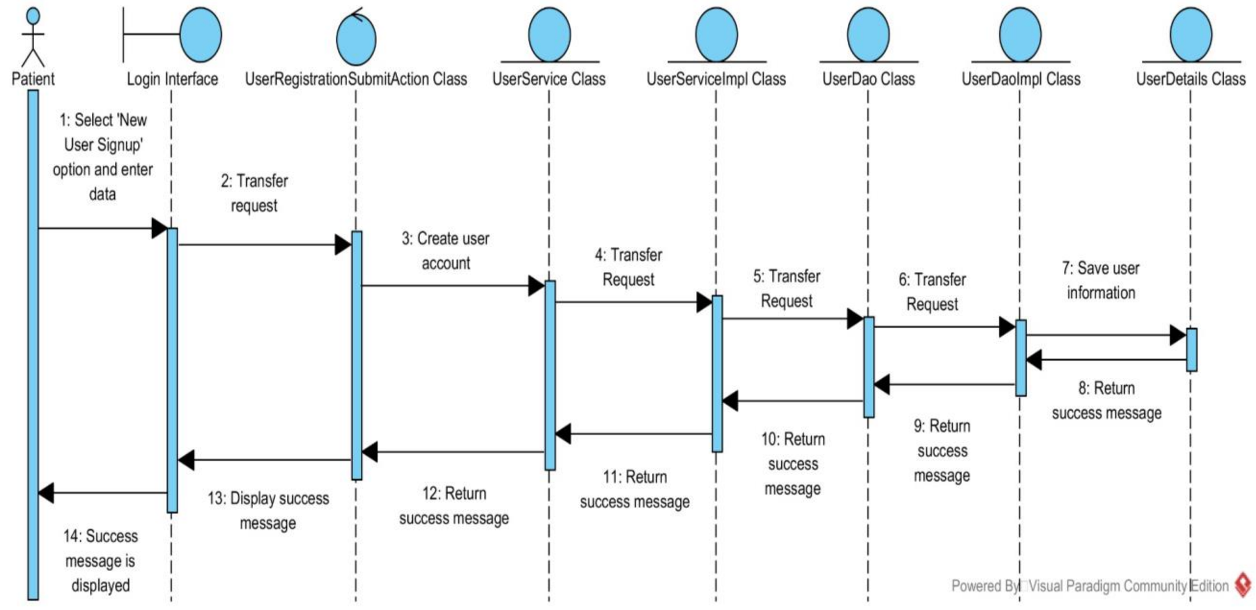

Patient
Login Interface
UserRegistrationSubmitAction Class
UserService Class
UserServiceImpl Class
UserDao Class
UserDaoImpl Class
UserDetails Class
1: Select 'New User Signup' option and enter data
2: Transfer request
3: Create user account
4: Transfer Request
5: Transfer Request
6: Transfer Request
7: Save user information
8: Return success message
9: Return success message
10: Return success message
11: Return success message
12: Return success message
13: Display success message
14: Success message is displayed
Powered By Visual Paradigm Community Edition

Figure 29: Sequence diagram for the ‘Create a new account’ scenario

Use Case Realization for the ‘*View Medical Expert Information’* use case

The classes that participate in the realization of the *View Medical Expert Information* use case and their relationships are shown in Figure 30.

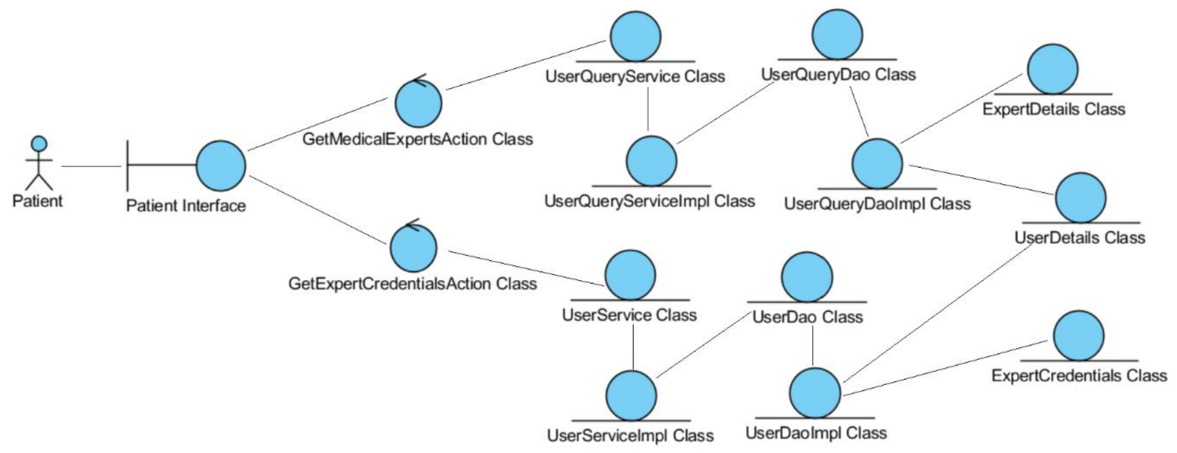


Figure 30: Interaction of classes for the realization of the 'View Medical Expert Information' use case

Scenario and Sequence diagram for the '*View Medical Expert Information'* use case:

Scenario 1 (View Medical Expert details)

The scenario 'View Medical Expert details' of *View Medical Expert Information* use case is shown in Table 18.

Table 18: 'View Medical Expert details' scenario of *View Medical Expert Information* use case

The Remote Health Service System Patient wants to view the different medical experts in the system and their specializations. 1. The Patient selects the desired medical specialization at the Patient screen. 2. The system provides the list of medical experts based on the selected specialization. 3. The Patient selects a medical expert from the list to view the educational background of that medical expert.

The sequence diagram of the realization of the 'View Medical Expert details' scenario of the *View Medical Expert Information* use case is shown in Figure 31.

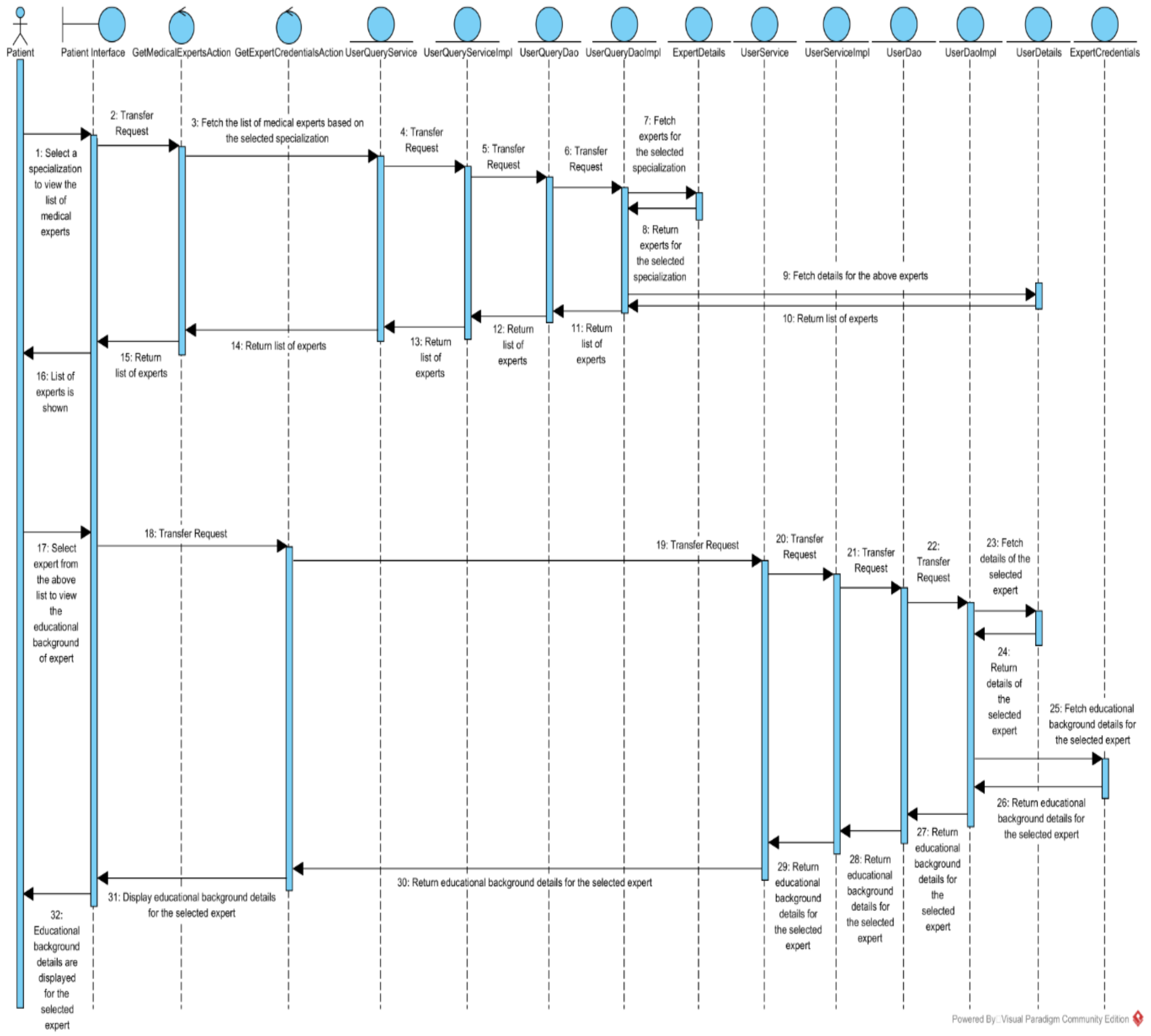

Patient
Patient Interface
GetMedicalExpertsAction
GetExpertCredentialsAction
UserQueryService
UserQueryServiceImpl
UserQueryDao
UserQueryDaoImpl
ExpertDetails
UserService
UserServiceImpl
UserDao
UserDaoImpl
UserDetails
ExpertCredentials
1: Select a specialization to view the list of medical experts
2: Transfer Request
3: Fetch the list of medical experts based on the selected specialization
4: Transfer Request
5: Transfer Request
6: Transfer Request
7: Fetch experts for the selected specialization
8: Return experts for the selected specialization
9: Fetch details for the above experts
10: Return list of experts
11: Return list of experts
12: Return list of experts
13: Return list of experts
14: Return list of experts
15: Return list of experts
16: List of experts is shown
17: Select expert from the above list to view the educational background of expert
18: Transfer Request
19: Transfer Request
20: Transfer Request
21: Transfer Request
22: Transfer Request
23: Fetch details of the selected expert
24: Return details of the selected expert
25: Fetch educational background details for the selected expert
26: Return educational background details for the selected expert
27: Return educational background details for the selected expert
28: Return educational background details for the selected expert
29: Return educational background details for the selected expert
30: Return educational background details for the selected expert
31: Display educational background details for the selected expert
32: Educational background details are displayed for the selected expert
Powered By: Visual Paradigm Community Edition

Figure 31: Sequence diagram for the 'View Medical Expert details' scenario

Use Case Realization for the *Edit Profile* use case

The classes that participate in the realization of the *Edit Profile* use case and their relationships are shown in Figure 32.

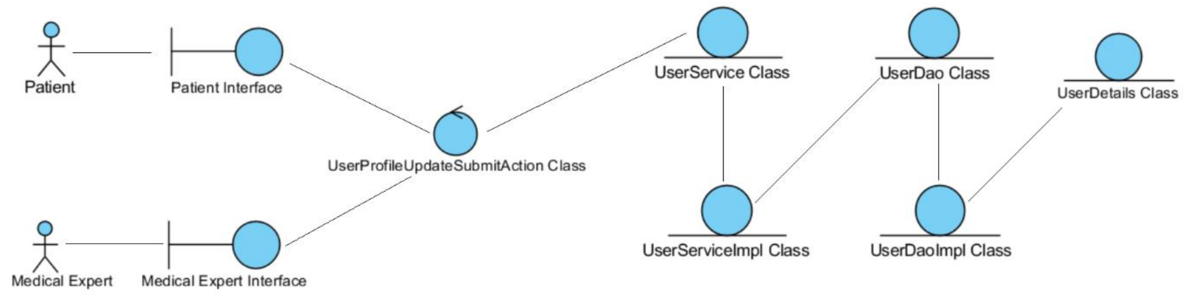


Figure 32: Interaction of classes for the realization of the *Edit Profile* use case

Scenarios and Sequence diagrams for the *Edit Profile* **use case:**

Scenario 1 (Patient – Edit Profile Information)

The scenario 'Patient – Edit Profile Information' of *Edit Profile* use case is shown in Table 19.

Table 19: 'Patient – Edit Profile Information' scenario of *Edit Profile* use case

The Remote Health Service System Patient wants to edit his/her profile information. 1. The Patient selects the 'Profile Management' option at the patient screen. 2. The Patient edits and submits the information. 3. The system updates the changes and displays the success message.

The sequence diagram of the realization of the 'Patient – Edit Profile Information' scenario of the *Edit Profile* use case is shown in Figure 33.

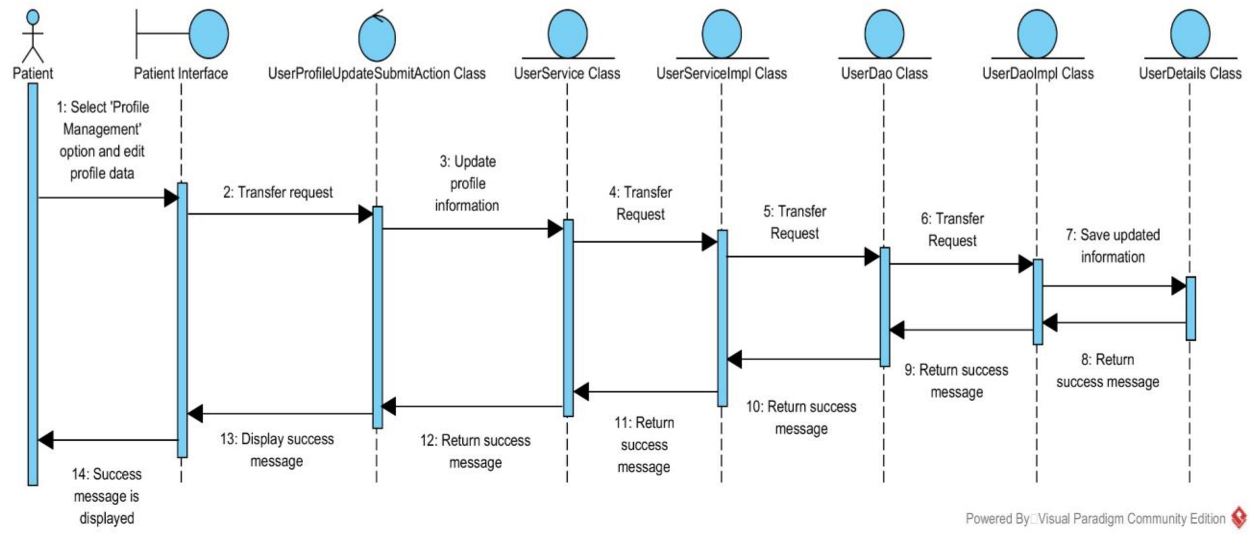

Patient
Patient Interface
UserProfileUpdateSubmitAction Class
UserService Class
UserServiceImpl Class
UserDao Class
UserDaoImpl Class
UserDetails Class
1: Select 'Profile Management' option and edit profile data
2: Transfer request
3: Update profile information
4: Transfer Request
5: Transfer Request
6: Transfer Request
7: Save updated information
8: Return success message
9: Return success message
10: Return success message
11: Return success message
12: Return success message
13: Display success message
14: Success message is displayed
Powered By Visual Paradigm Community Edition

Figure 33: Sequence diagram for the 'Patient – Edit Profile Information' scenario

Scenario 2 (Medical Expert – Edit Profile Information)

The scenario 'Medical Expert – Edit Profile Information' of *Edit Profile* use case is shown in Table 20.

Table 20: 'Medical Expert – Edit Profile Information' scenario of *Edit Profile* use case

The Remote Health Service System Medical Expert wants to edit his/her profile information. 1. The Medical Expert selects the 'Profile Management' option at the medical expert screen. 2. The Medical Expert edits and submits the information. 3. The system updates the changes and displays the success message.

The sequence diagram of the realization of the 'Medical Expert – Edit Profile Information' scenario of the *Edit Profile* use case is shown in Figure 34.

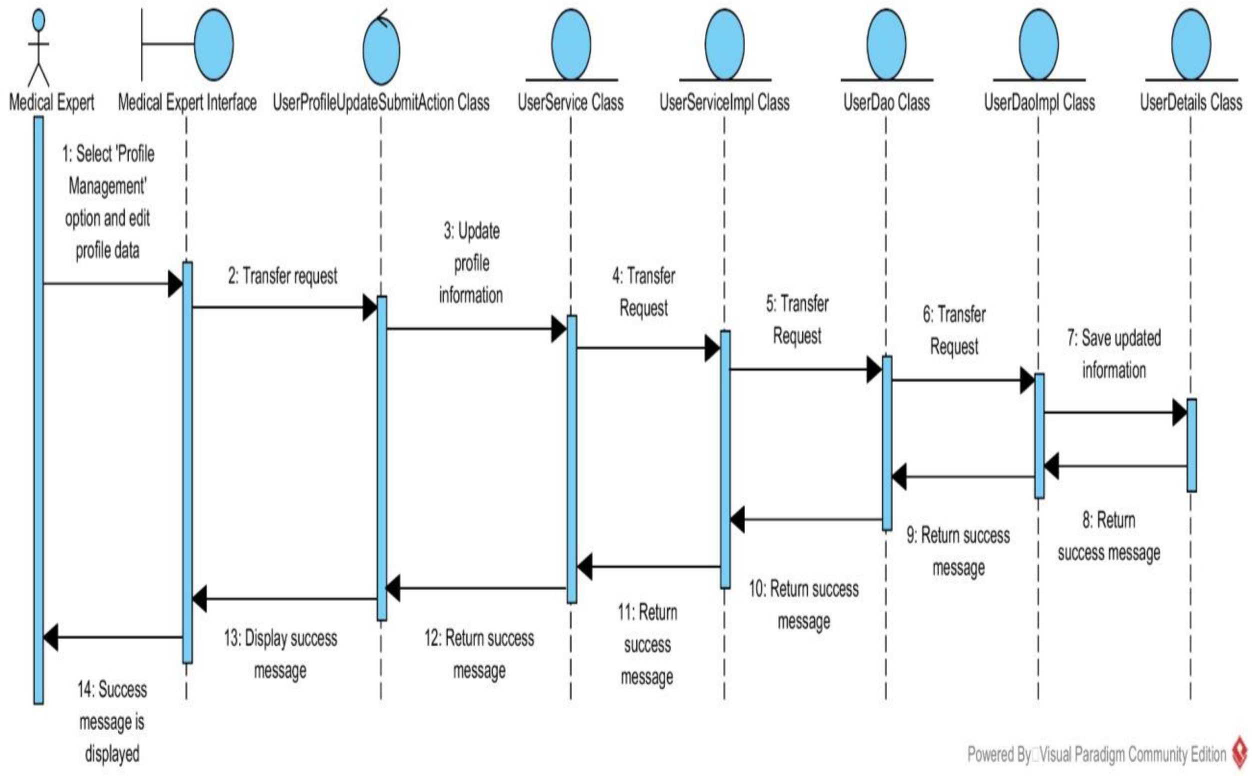

Medical Expert
Medical Expert Interface
UserProfileUpdateSubmitAction Class
UserService Class
UserServiceImpl Class
UserDao Class
UserDaoImpl Class
UserDetails Class
1: Select 'Profile Management' option and edit profile data
2: Transfer request
3: Update profile information
4: Transfer Request
5: Transfer Request
6: Transfer Request
7: Save updated information
8: Return success message
9: Return success message
10: Return success message
11: Return success message
12: Return success message
13: Display success message
14: Success message is displayed
Powered By Visual Paradigm Community Edition

Figure 34: Sequence diagram for the 'Medical Expert – Edit Profile Information' scenario

Use Case Realization for the *Submit Query* use case

The classes that participate in the realization of the *Submit Query* use case and their relationships are shown in Figure 35.

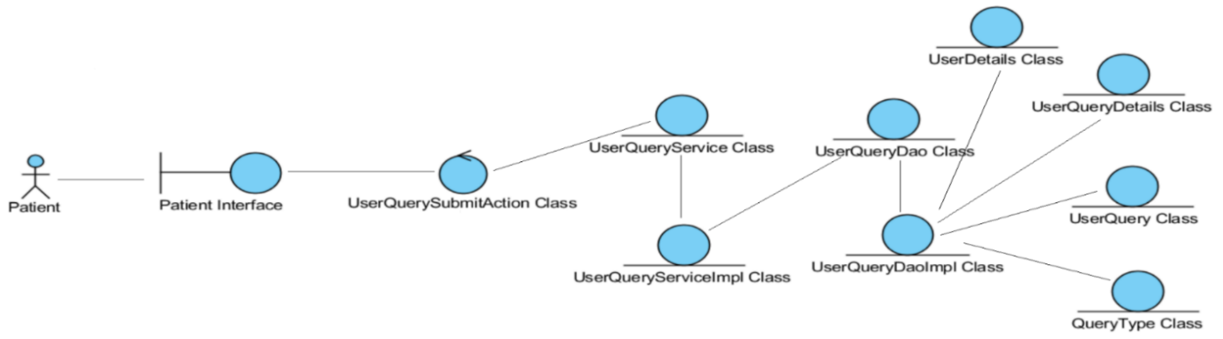


Figure 35: Interaction of classes for the realization of the *Submit Query* use case

Scenario and Sequence diagram for the *Submit Query* use case:

Scenario 1 (Submit query to Medical Expert)

The scenario ‘Submit query to Medical Expert’ of *Submit Query* use case is shown in Table 21.

Table 21: ‘Submit query to Medical Expert’ scenario of *Submit Query* use case

The Remote Health Service System Patient wants to submit a new query to a medical expert.

1. The Patient selects the ‘Submit Query’ option and picks a medical expert.
2. The Patient enters the query and submits it.
3. The system sends the query to the selected medical expert and displays a success message.

The sequence diagram of the realization of the ‘Submit query to Medical Expert’ scenario of the *Submit Query* use case is shown in Figure 36.

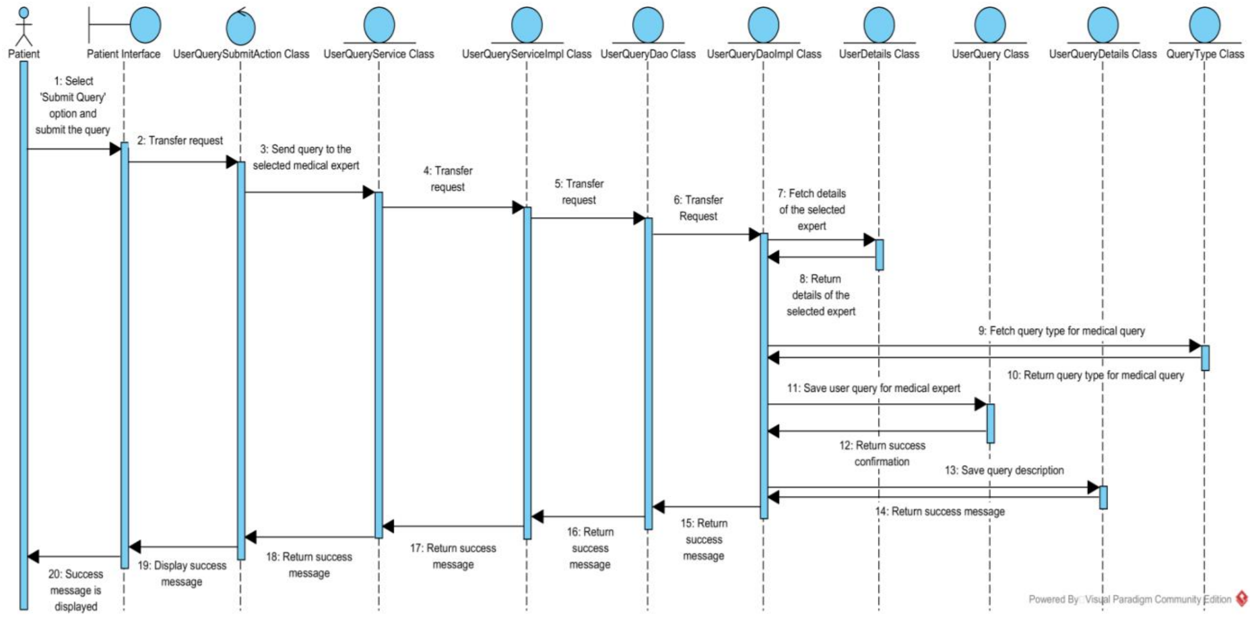

Patient
Patient Interface
UserQuerySubmitAction Class
UserQueryService Class
UserQueryServiceImpl Class
UserQueryDao Class
UserQueryDaoImpl Class
UserDetails Class
UserQuery Class
UserQueryDetails Class
QueryType Class
1: Select 'Submit Query' option and submit the query
2: Transfer request
3: Send query to the selected medical expert
4: Transfer request
5: Transfer request
6: Transfer Request
7: Fetch details of the selected expert
8: Return details of the selected expert
9: Fetch query type for medical query
10: Return query type for medical query
11: Save user query for medical expert
12: Return success confirmation
13: Save query description
14: Return success message
15: Return success message
16: Return success message
17: Return success message
18: Return success message
19: Display success message
20: Success message is displayed
Powered By: Visual Paradigm Community Edition

Figure 36: Sequence diagram for the ‘Submit query to Medical Expert’ scenario

Use Case Realization for the *View Response* use case

The classes that participate in the realization of the *View Response* use case and their relationships are shown in Figure 37.

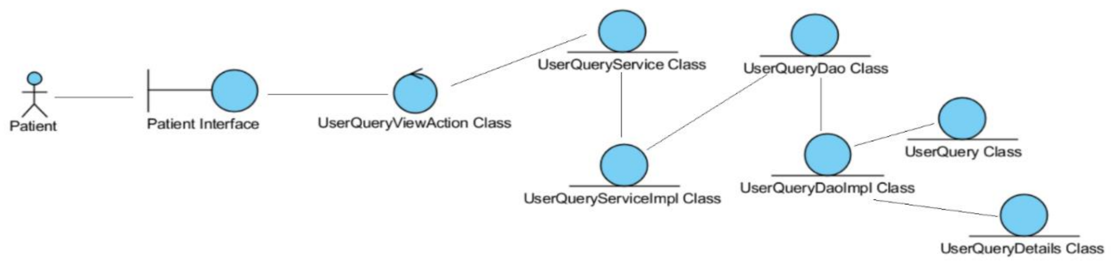


Figure 37: Interaction of classes for the realization of the *View Response* use case

Scenario and Sequence diagram for the *View Response* use case:

Scenario 1 (View query response)

The scenario 'View query response' of *View Response* use case is shown in Table 22.

Table 22: 'View query response' scenario of *View Response* use case

The Remote Health Service System Patient wants to view the response for his/her query from the medical expert. 1. The Patient selects the 'Query Search' option and searches for a previously submitted query. 2. The Patient selects the query to view the response from the medical expert. 3. The system displays the response sent by the medical expert.

The sequence diagram of the realization of the 'View query response' scenario of the *View Response* use case is shown in Figure 38.

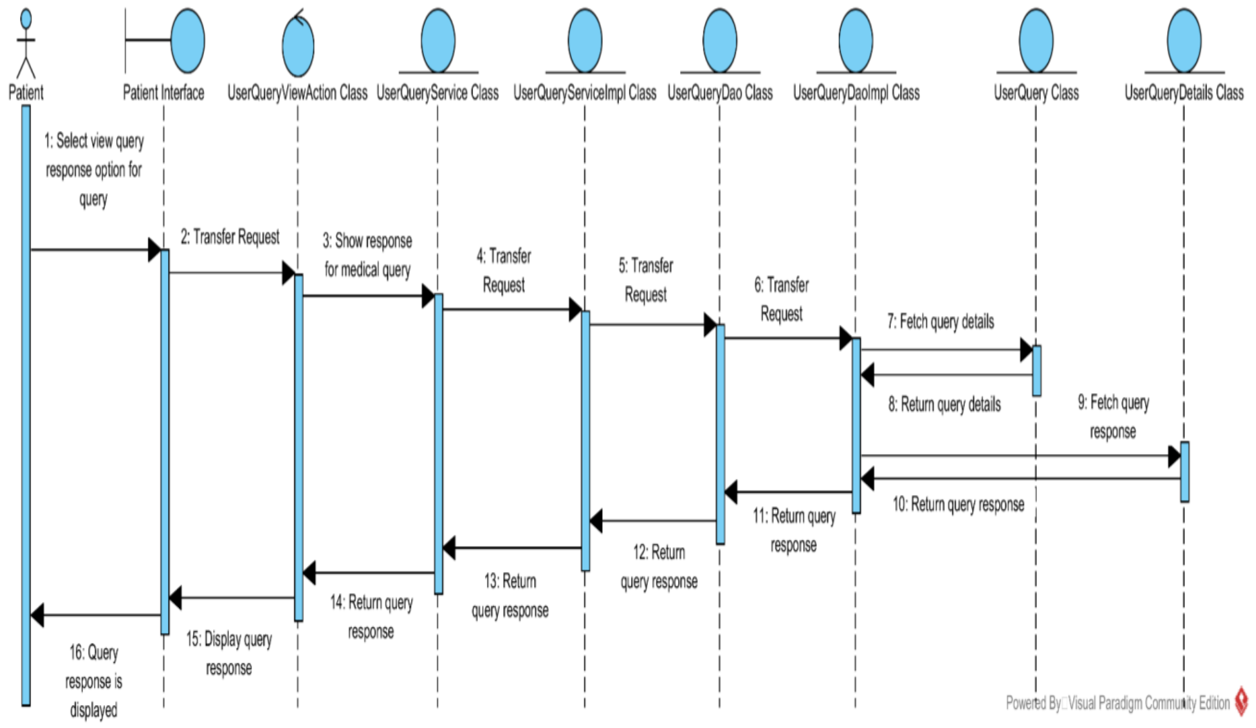

Patient
Patient Interface
UserQueryViewAction Class
UserQueryService Class
UserQueryServiceImpl Class
UserQueryDao Class
UserQueryDaoImpl Class
UserQuery Class
UserQueryDetails Class
1: Select view query response option for query
2: Transfer Request
3: Show response for medical query
4: Transfer Request
5: Transfer Request
6: Transfer Request
7: Fetch query details
8: Return query details
9: Fetch query response
10: Return query response
11: Return query response
12: Return query response
13: Return query response
14: Return query response
15: Display query response
16: Query response is displayed
Powered By□Visual Paradigm Community Edition

Figure 38: Sequence diagram for the ‘View query response’ scenario

Use Case Realization for the *View Past Medical Queries* use case

The classes that participate in the realization of the *View Past Medical Queries* use case and their relationships are shown in Figure 39.

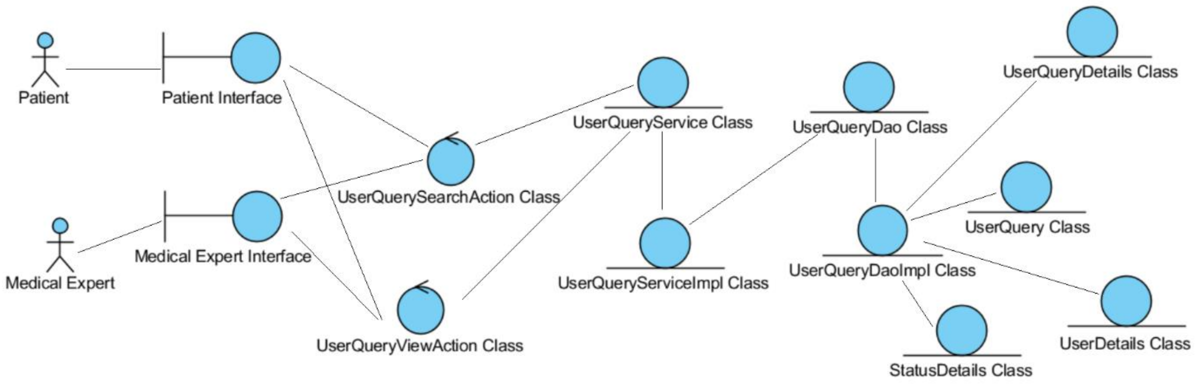


Figure 39: Interaction of classes for the realization of the *View Past Medical Queries* use case

Scenarios and Sequence diagrams for the *View Past Medical Queries* **use case:**

Scenario 1 (Patient – View past queries)

The scenario ‘Patient – View past queries’ of *View Past Medical Queries* use case is shown in Table 23.

Table 23: ‘Patient – View past queries’ scenario of *View Past Medical Queries* use case

The Remote Health Service System Patient wants to view a previously submitted query. 1. The Patient selects the ‘Query Search’ option and enters the search criteria. 2. The system displays the medical queries matching the search criteria. 3. The Patient can now select the query to see its details.

The sequence diagram of the realization of the ‘Patient – View past queries’ scenario of the *View Past Medical Queries* use case is shown in Figure 40.

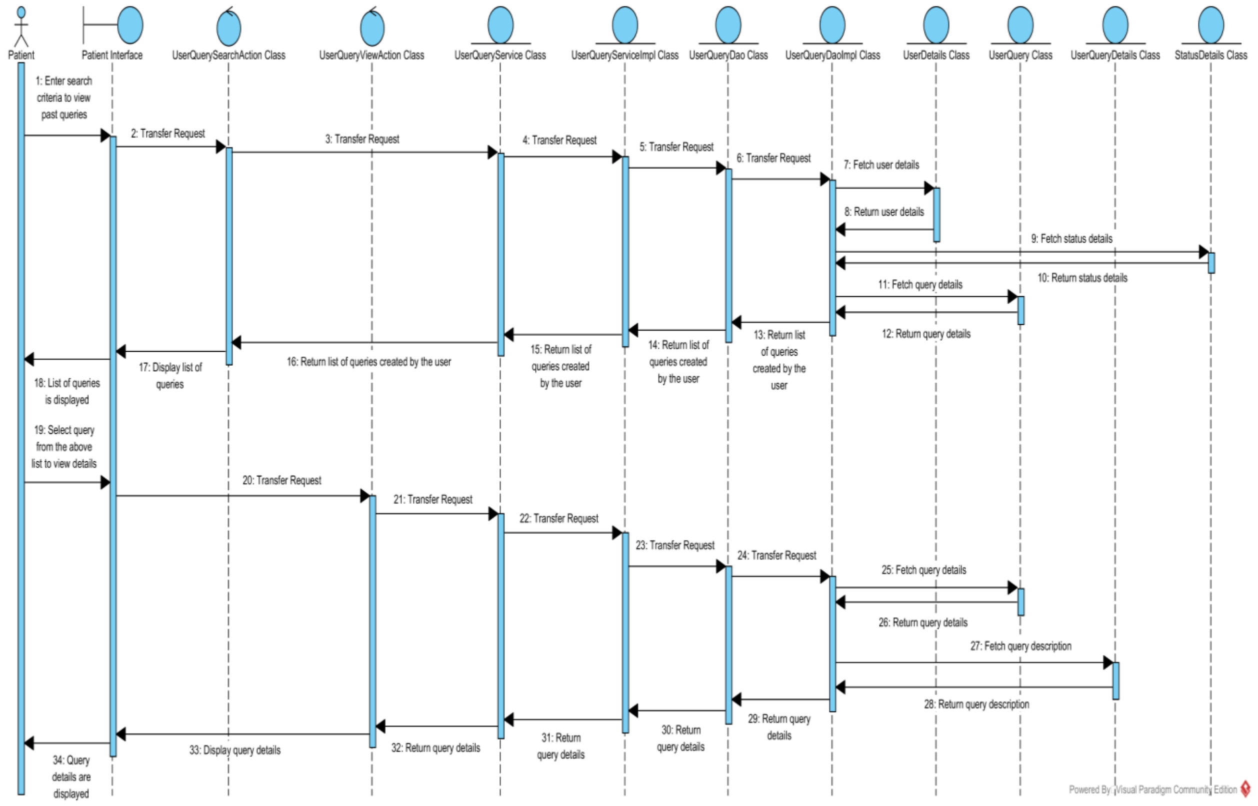


Figure 40: Sequence diagram for the 'Patient – View past queries' scenario

Scenario 2 (Medical Expert – View past queries)

The scenario 'Medical Expert – View past queries' of *View Past Medical Queries* use case is shown in Table 24.

Table 24: 'Medical Expert – View past queries' scenario of *View Past Medical Queries* use case

The Remote Health Service System Medical Expert wants to view a particular query from the list of queries assigned to him/her. 1. The Medical Expert selects the 'Query Search' option and enters the search criteria. 2. The system displays the medical queries matching the search criteria. 3. The Medical Expert can now select the query to see its details.

The sequence diagram of the realization of the ‘Medical Expert – View past queries’ scenario of the *View Past Medical Queries* use case is shown in Figure 41.

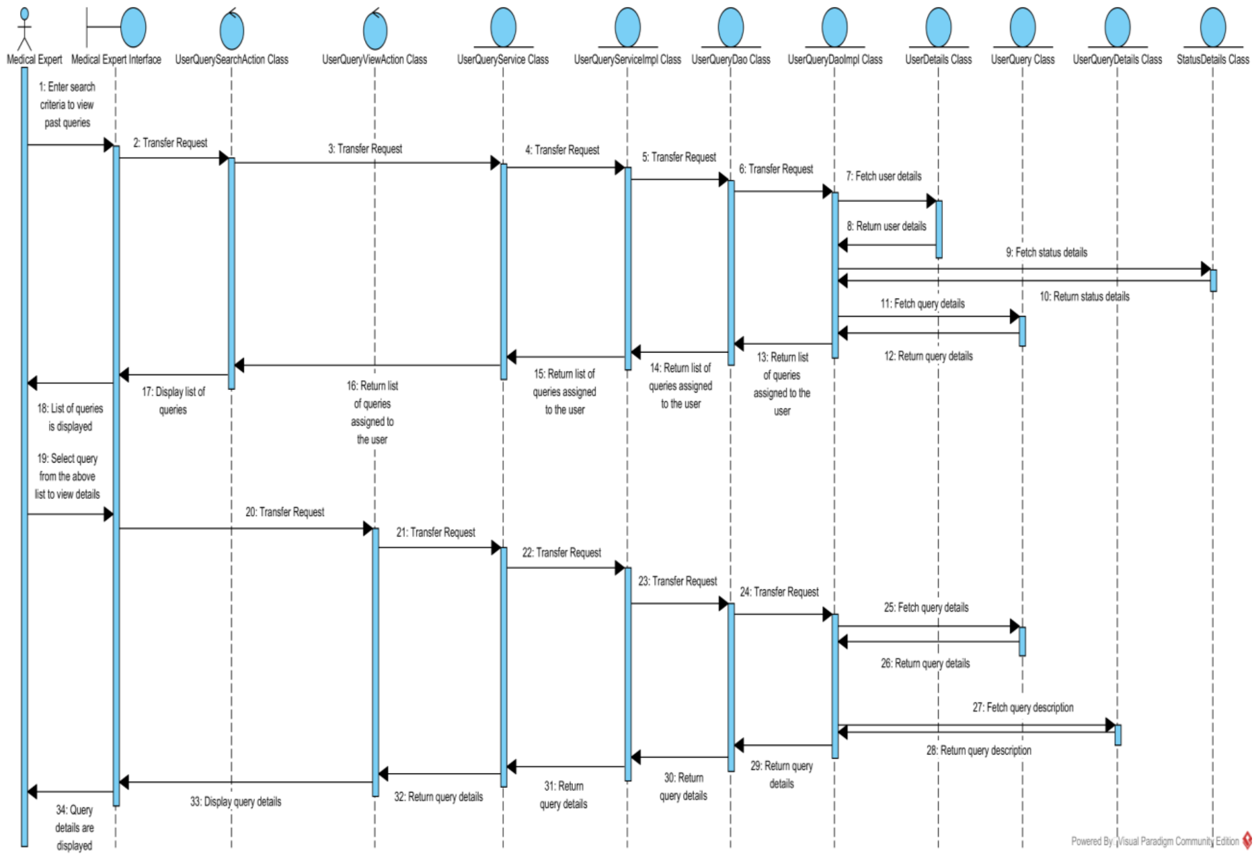


Figure 41: Sequence diagram for the ‘Medical Expert – View past queries’ scenario

Use Case Realization for the *Respond to Query* use case

The classes that participate in the realization of the *Respond to Query* use case and their relationships are shown in Figure 42.

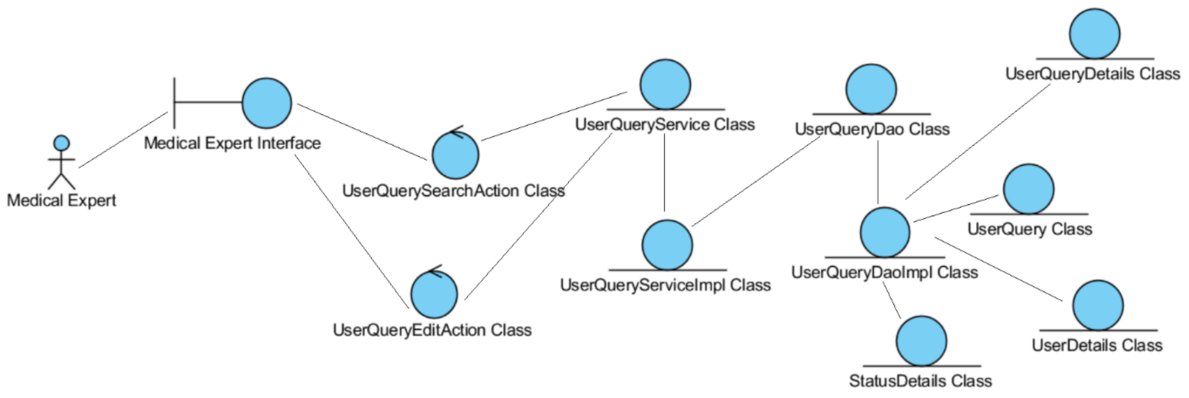


Figure 42: Interaction of classes for the realization of the *Respond to Query* use case

Scenario and Sequence diagram for the *Respond to Query* **use case:**

Scenario 1 (Respond to patient query)

The scenario ‘Respond to patient query’ of *Respond to Query* use case is shown in Table 25.

Table 25: ‘Respond to patient query’ scenario of *Respond to Query* use case

The Remote Health Service System Medical Expert wants to respond to a patient query.
1. The Medical Expert selects the ‘Query Search’ option and searches for queries with ‘Open’ status. 2. The Medical Expert selects the query, writes the response and saves it. Also, the query status can be changed. 3. The system sends the response to the patient and displays a success message.

The sequence diagram of the realization of the ‘Respond to patient query’ scenario of the *Respond to Query* use case is shown in Figure 43.

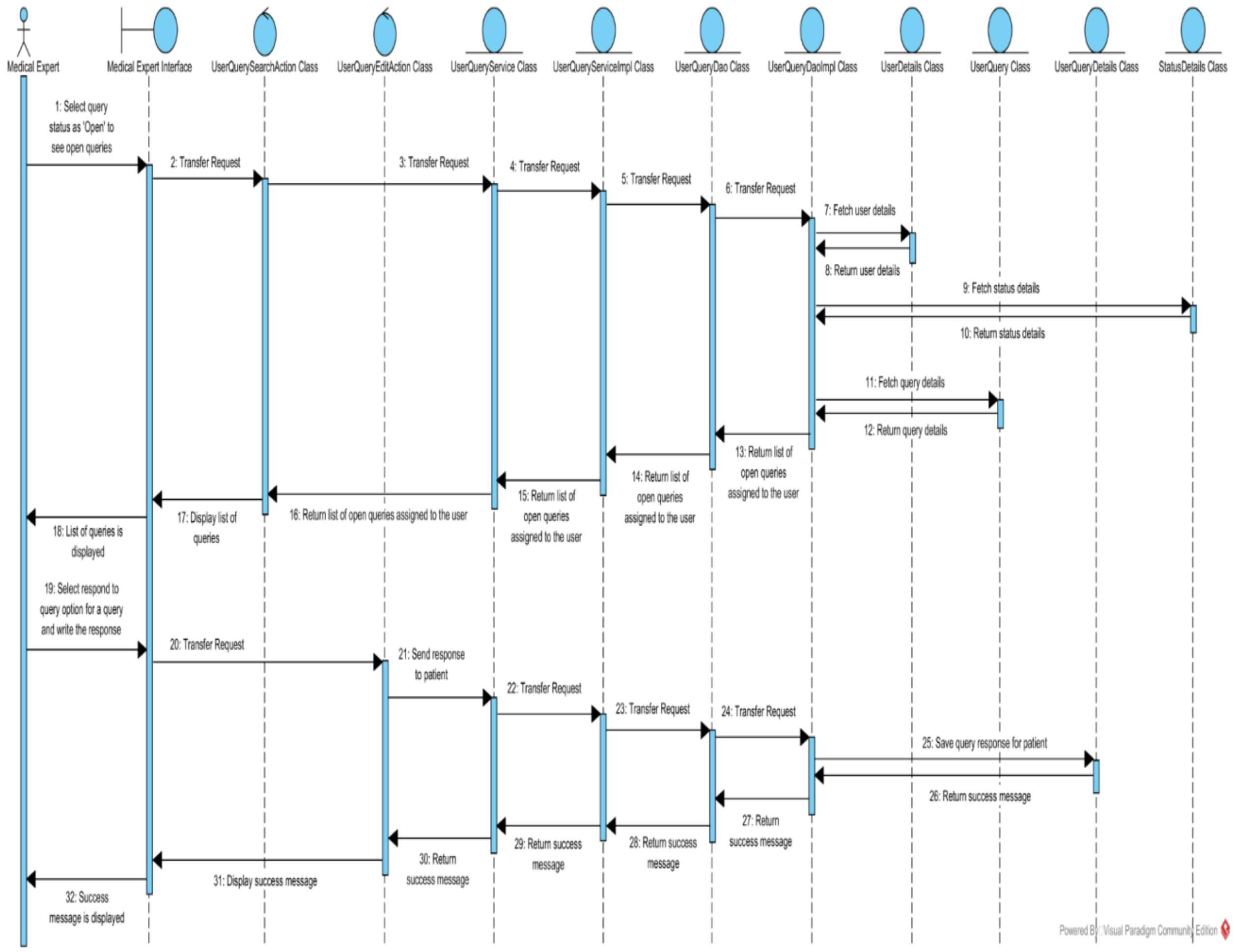

Medical Expert
Medical Expert Interface
UserQuerySearchAction Class
UserQueryEditAction Class
UserQueryService Class
UserQueryServiceImpl Class
UserQueryDao Class
UserQueryDaoImpl Class
UserDetails Class
UserQuery Class
UserQueryDetails Class
StatusDetails Class
1: Select query status as 'Open' to see open queries
2: Transfer Request
3: Transfer Request
4: Transfer Request
5: Transfer Request
6: Transfer Request
7: Fetch user details
8: Return user details
9: Fetch status details
10: Return status details
11: Fetch query details
12: Return query details
13: Return list of open queries assigned to the user
14: Return list of open queries assigned to the user
15: Return list of open queries assigned to the user
16: Return list of open queries assigned to the user
17: Display list of queries
18: List of queries is displayed
19: Select respond to query option for a query and write the response
20: Transfer Request
21: Send response to patient
22: Transfer Request
23: Transfer Request
24: Transfer Request
25: Save query response for patient
26: Return success message
27: Return success message
28: Return success message
29: Return success message
30: Return success message
31: Display success message
32: Success message is displayed
Powered By: Visual Paradigm Community Edition

Figure 43: Sequence diagram for the ‘Respond to patient query’ scenario

Use Case Realization for the *Transfer Query* use case

The classes that participate in the realization of the *Transfer Query* use case and their relationships are shown in Figure 44.

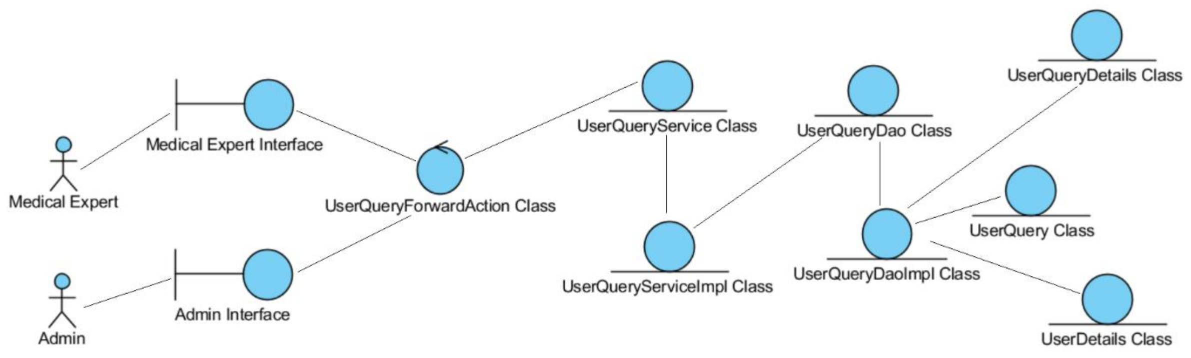


Figure 44: Interaction of classes for the realization of the *Transfer Query* use case

Scenarios and Sequence diagrams for the *Transfer Query* **use case:**

Scenario 1 (Medical Expert – Transfer query)

The scenario 'Medical Expert – Transfer query' of *Transfer Query* use case is shown in Table 26.

Table 26: 'Medical Expert – Transfer query' scenario of *Transfer Query* use case

The Remote Health Service System Medical Expert wants to transfer a patient query to another Medical Expert. 1. The Medical Expert selects the 'Query Search' option and searches for queries with 'Open' status. 2. The Medical Expert selects the query, selects the medical expert to whom he/she wants to transfer the query and saves the data. 3. The system sends the query to the selected medical expert and displays a success message.

The sequence diagram of the realization of the 'Medical Expert – Transfer query' scenario of the *Transfer Query* use case is shown in Figure 45.

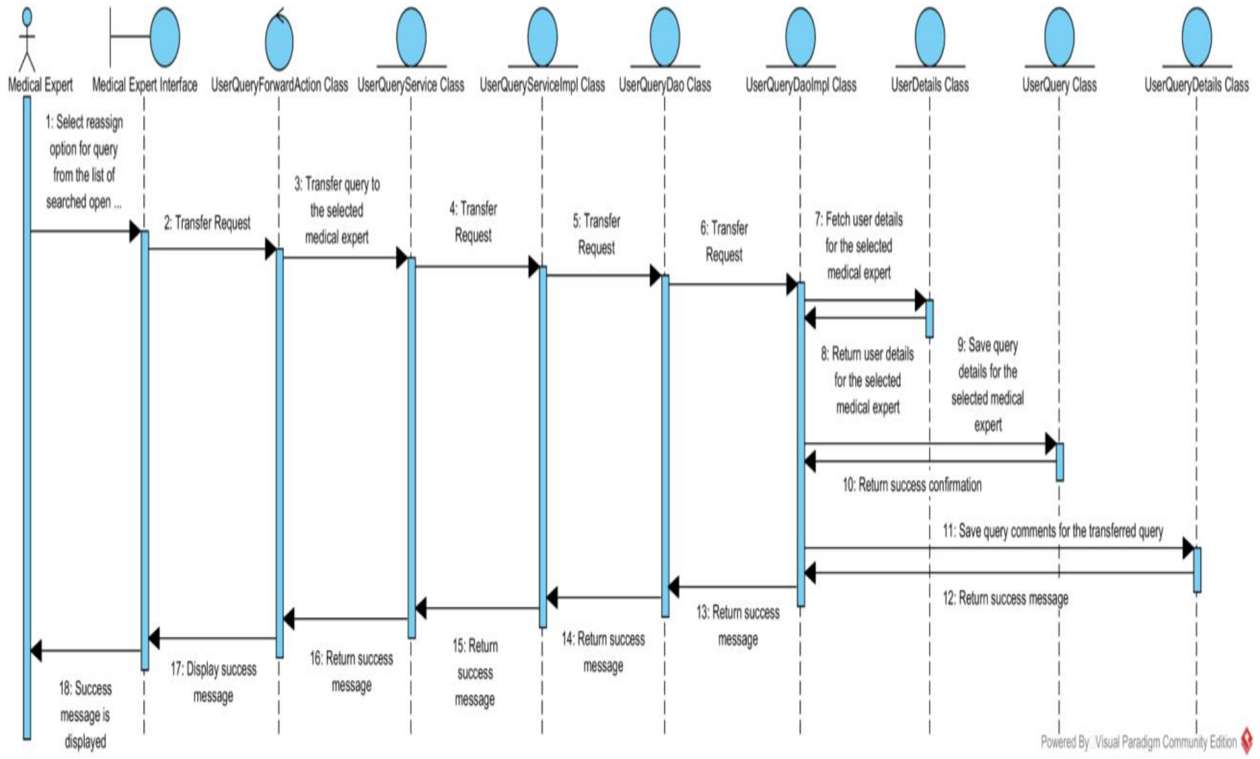

Medical Expert
Medical Expert Interface
UserQueryForwardAction Class
UserQueryService Class
UserQueryServiceImpl Class
UserQueryDao Class
UserQueryDaoImpl Class
UserDetails Class
UserQuery Class
UserQueryDetails Class
1: Select reassign option for query from the list of searched open ...
2: Transfer Request
3: Transfer query to the selected medical expert
4: Transfer Request
5: Transfer Request
6: Transfer Request
7: Fetch user details for the selected medical expert
8: Return user details for the selected medical expert
9: Save query details for the selected medical expert
10: Return success confirmation
11: Save query comments for the transferred query
12: Return success message
13: Return success message
14: Return success message
15: Return success message
16: Return success message
17: Display success message
18: Success message is displayed
Powered By Visual Paradigm Community Edition

Figure 45: Sequence diagram for the 'Medical Expert – Transfer query' scenario

Scenario 2 (Admin – Transfer query)

The scenario 'Admin – Transfer query' of *Transfer Query* use case is shown in Table 27.

Table 27: 'Admin – Transfer query' scenario of *Transfer Query* use case

The Remote Health Service System Admin wants to transfer a patient query from one Medical Expert to another. 1. The Admin selects the 'Query Search' option and searches for queries with 'Open' status. 2. The Admin selects the 'Reassign Query' option for a query and then selects the medical expert to whom he/she wants to transfer the query and saves the data.

3. The system sends the query to the selected medical expert and displays a success message.

The sequence diagram of the realization of the 'Admin – Transfer query' scenario of the *Transfer Query* use case is shown in Figure 46.

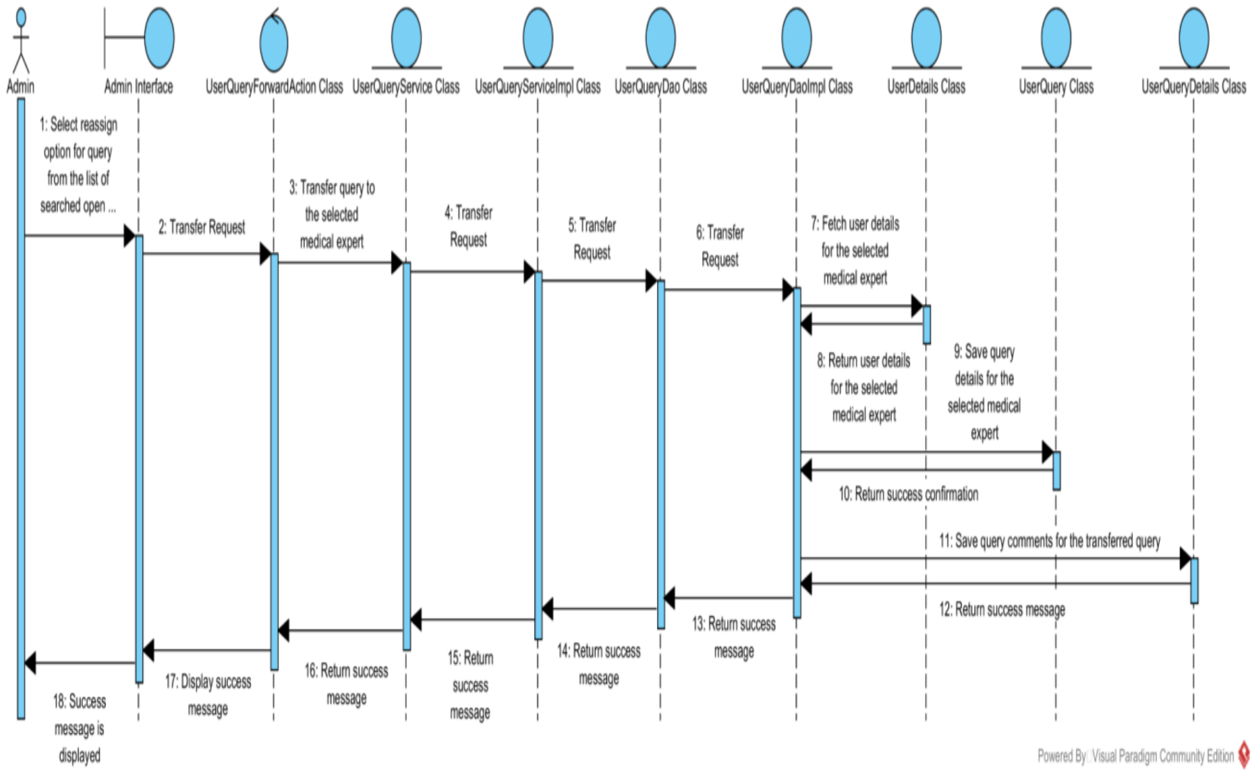

Admin
Admin Interface
UserQueryForwardAction Class
UserQueryService Class
UserQueryServiceImpl Class
UserQueryDao Class
UserQueryDaoImpl Class
UserDetails Class
UserQuery Class
UserQueryDetails Class
1: Select reassign option for query from the list of searched open ...
2: Transfer Request
3: Transfer query to the selected medical expert
4: Transfer Request
5: Transfer Request
6: Transfer Request
7: Fetch user details for the selected medical expert
8: Return user details for the selected medical expert
9: Save query details for the selected medical expert
10: Return success confirmation
11: Save query comments for the transferred query
12: Return success message
13: Return success message
14: Return success message
15: Return success message
16: Return success message
17: Display success message
18: Success message is displayed
Powered By: Visual Paradigm Community Edition

Figure 46: Sequence diagram for the 'Admin – Transfer query' scenario

Use Case Realization for the *Manage Medical Experts* use case

The classes that participate in the realization of the *Manage Medical Experts* use case and their relationships are shown in Figure 47.

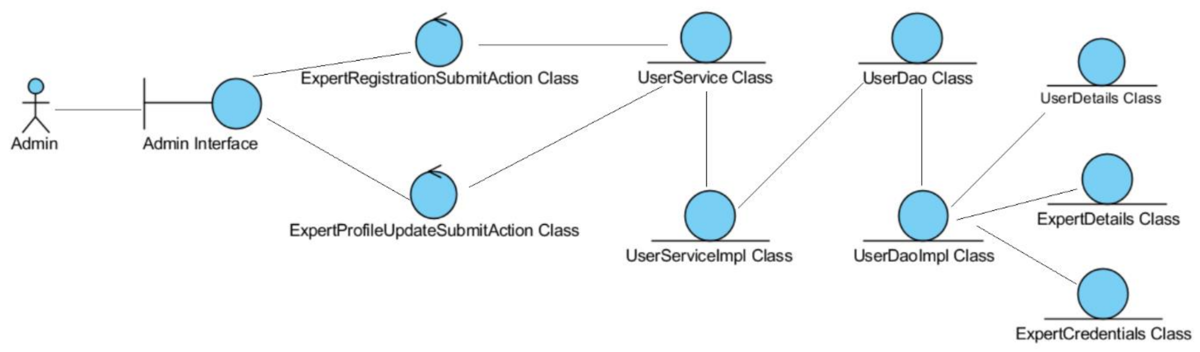


Figure 47: Interaction of classes for the realization of the *Manage Medical Experts* use case

Scenarios and Sequence diagrams for the *Manage Medical Experts* **use case:**

Scenario 1 (Add a medical expert)

The scenario ‘Add a medical expert’ of *Manage Medical Experts* use case is shown in Table 28.

Table 28: ‘Add a medical expert’ scenario of *Manage Medical Experts* use case

The Remote Health Service System Admin wants to add a new Medical Expert to the system. 1. The Admin selects the ‘Enroll Expert’ option at the Admin screen. 2. The Admin enters the details of the new expert and saves the data. 3. The system creates a new medical expert and displays a success message to the user.

The sequence diagram of the realization of the ‘Add a medical expert’ scenario of the *Manage Medical Experts* use case is shown in Figure 48.

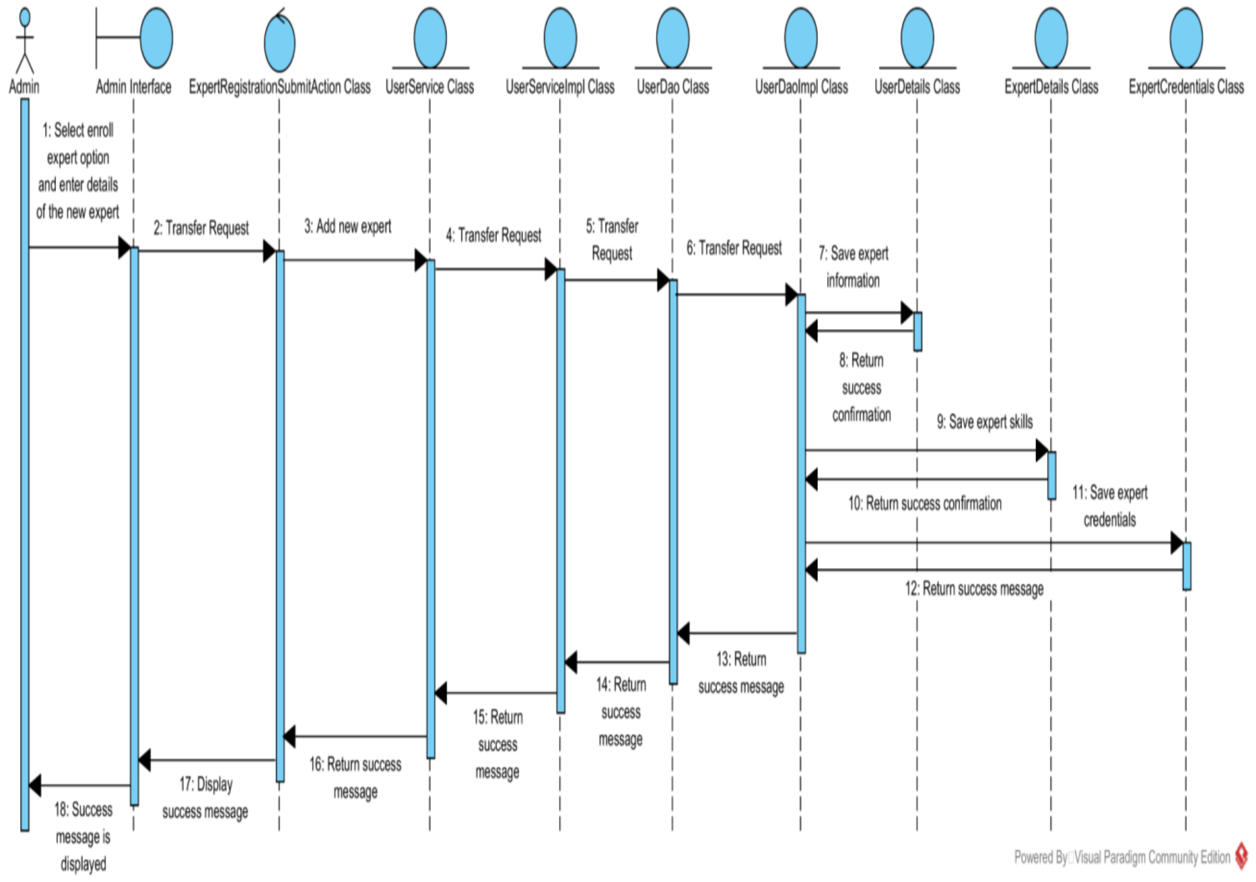

Admin
Admin Interface
ExpertRegistrationSubmitAction Class
UserService Class
UserServiceImpl Class
UserDao Class
UserDaoImpl Class
UserDetails Class
ExpertDetails Class
ExpertCredentials Class
1: Select enroll expert option and enter details of the new expert
2: Transfer Request
3: Add new expert
4: Transfer Request
5: Transfer Request
6: Transfer Request
7: Save expert information
8: Return success confirmation
9: Save expert skills
10: Return success confirmation
11: Save expert credentials
12: Return success message
13: Return success message
14: Return success message
15: Return success message
16: Return success message
17: Display success message
18: Success message is displayed
Powered By: Visual Paradigm Community Edition

Figure 48: Sequence diagram for the 'Add a medical expert' scenario

Scenario 2 (Edit a medical expert)

The scenario 'Edit a medical expert' of *Manage Medical Experts* use case is shown in Table 29.

Table 29: 'Edit a medical expert' scenario of *Manage Medical Experts* use case

The Remote Health Service System Admin wants to edit the details of an existing Medical Expert. 1. The Admin selects the 'Manage User' option at the Admin screen. 2. The Admin selects the medical expert to be updated, edits the details and saves it.

3. The system updates the medical expert details and displays a success message to the user.

The sequence diagram of the realization of the 'Edit a medical expert' scenario of the *Manage Medical Experts* use case is shown in Figure 49.

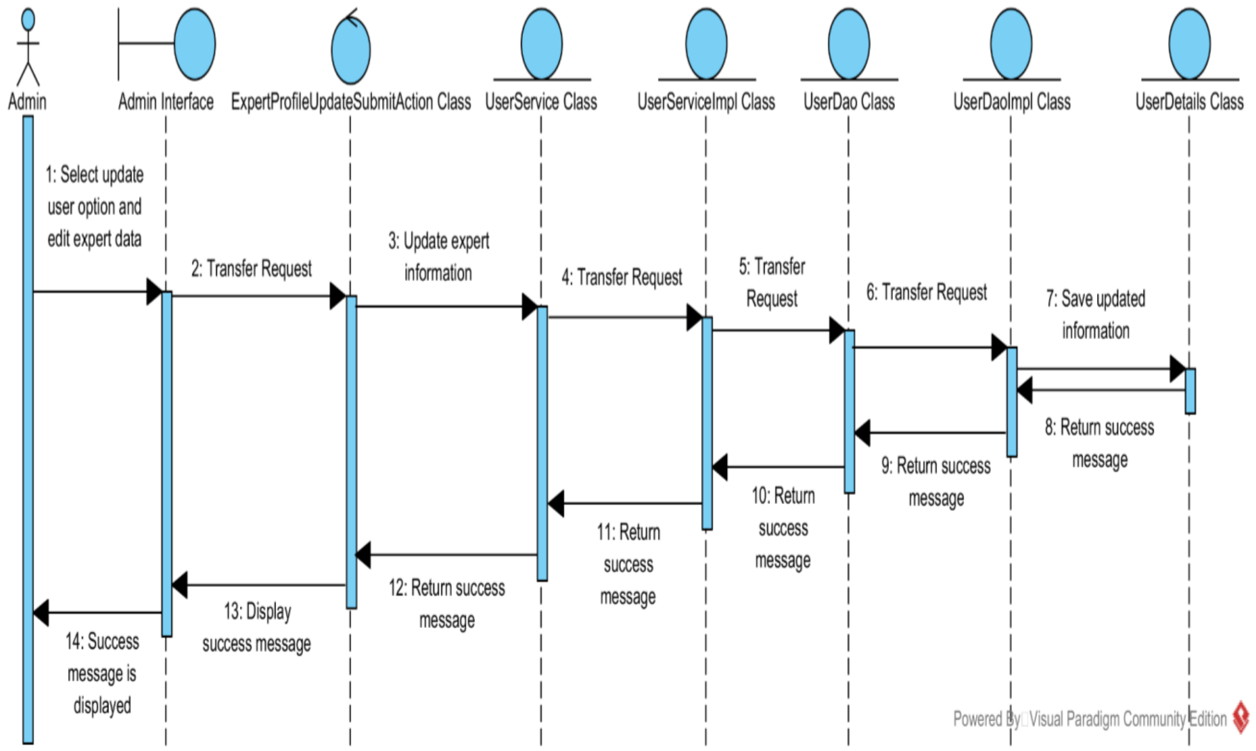

Admin
Admin Interface
ExpertProfileUpdateSubmitAction Class
UserService Class
UserServiceImpl Class
UserDao Class
UserDaoImpl Class
UserDetails Class
1: Select update user option and edit expert data
2: Transfer Request
3: Update expert information
4: Transfer Request
5: Transfer Request
6: Transfer Request
7: Save updated information
8: Return success message
9: Return success message
10: Return success message
11: Return success message
12: Return success message
13: Display success message
14: Success message is displayed
Powered By Visual Paradigm Community Edition

Figure 49: Sequence diagram for the 'Edit a medical expert' scenario

Scenario 3 (Activate a medical expert)

The scenario 'Activate a medical expert' of *Manage Medical Experts* use case is shown in Table 30.

Table 30: 'Activate a medical expert' scenario of *Manage Medical Experts* use case

The Remote Health Service System Admin wants to activate a previously deactivated medical expert.

1. The Admin selects the ‘Manage User’ option at the Admin screen.
2. The Admin selects the medical expert to be activated in the system, selects the option to activate and saves the data.
3. The system activates the medical expert and displays a success message to the user.

The sequence diagram of the realization of the ‘Activate a medical expert’ scenario of the *Manage Medical Experts* use case is shown in Figure 50.

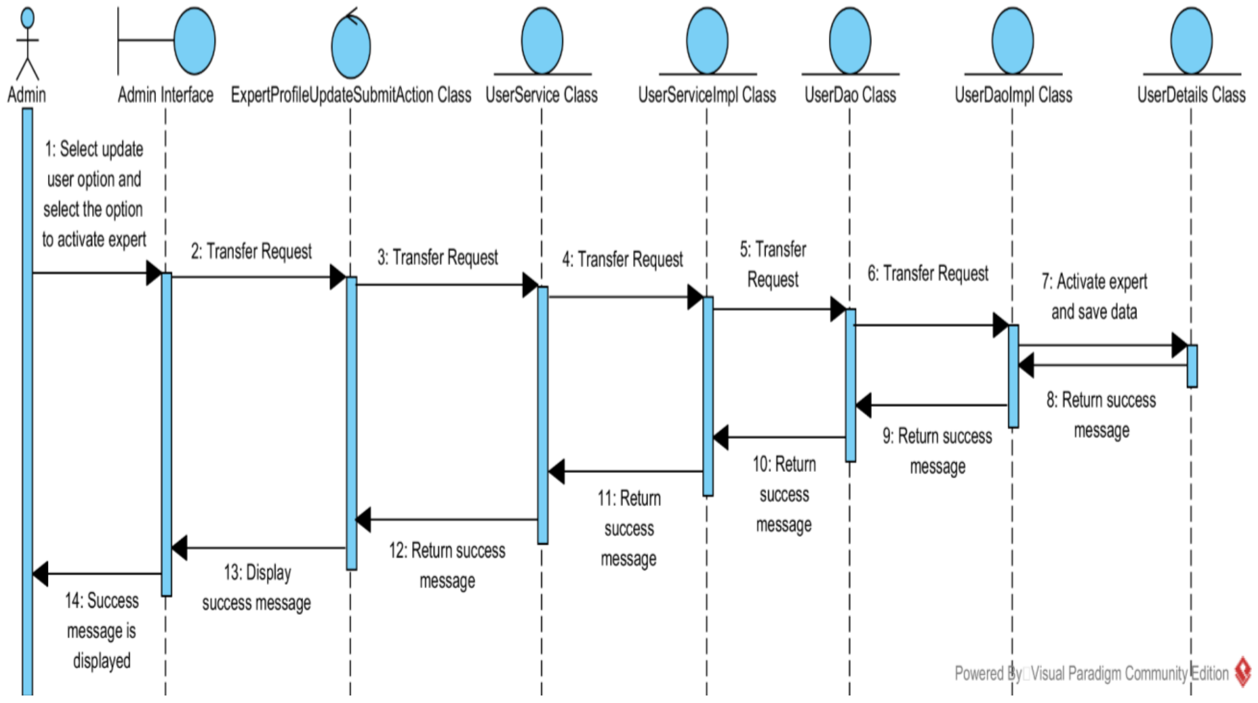

Admin
Admin Interface
ExpertProfileUpdateSubmitAction Class
UserService Class
UserServiceImpl Class
UserDao Class
UserDaoImpl Class
UserDetails Class
1: Select update user option and select the option to activate expert
2: Transfer Request
3: Transfer Request
4: Transfer Request
5: Transfer Request
6: Transfer Request
7: Activate expert and save data
8: Return success message
9: Return success message
10: Return success message
11: Return success message
12: Return success message
13: Display success message
14: Success message is displayed
Powered By Visual Paradigm Community Edition

Figure 50: Sequence diagram for the ‘Activate a medical expert’ scenario

Scenario 4 (Deactivate a medical expert)

The scenario ‘Deactivate a medical expert’ of *Manage Medical Experts* use case is shown in Table 31.

Table 31: ‘Deactivate a medical expert’ scenario of *Manage Medical Experts* use case

The Remote Health Service System Admin wants to deactivate a medical expert. 1. The Admin selects the ‘Manage User’ option at the Admin screen. 2. The Admin selects the medical expert to be deactivated in the system, selects the option to deactivate and saves the data. 3. The system deactivates the medical expert and displays a success message to the user.

The sequence diagram of the realization of the ‘Deactivate a medical expert’ scenario of the *Manage Medical Experts* use case is shown in Figure 51.

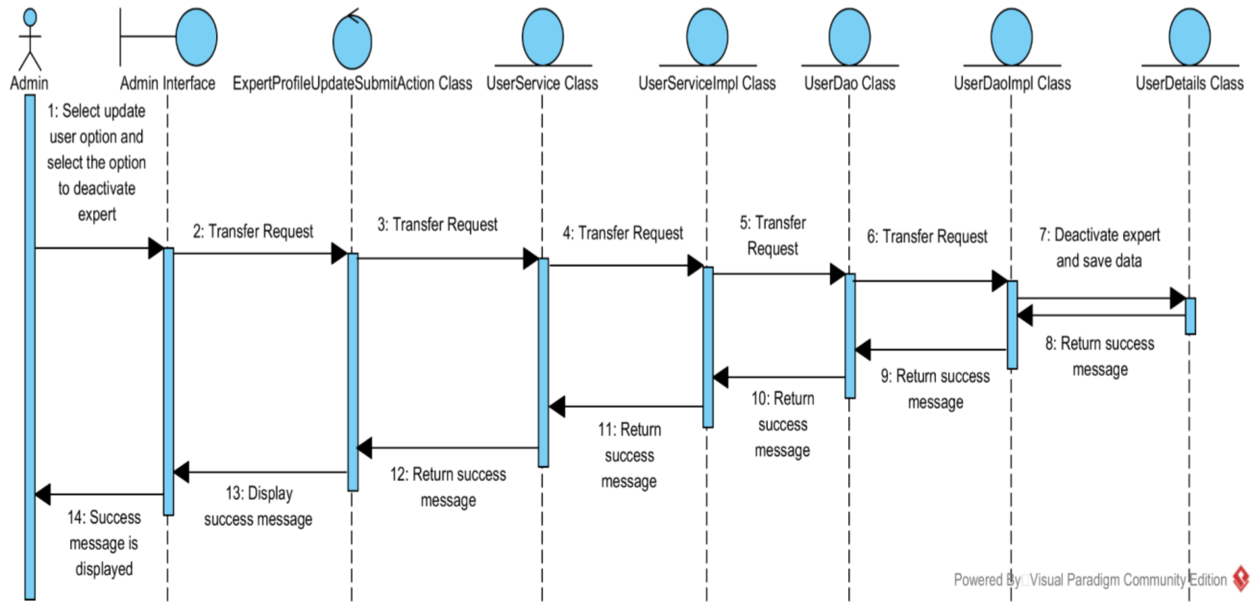

Admin
Admin Interface
ExpertProfileUpdateSubmitAction Class
UserService Class
UserServiceImpl Class
UserDao Class
UserDaoImpl Class
UserDetails Class
1: Select update user option and select the option to deactivate expert
2: Transfer Request
3: Transfer Request
4: Transfer Request
5: Transfer Request
6: Transfer Request
7: Deactivate expert and save data
8: Return success message
9: Return success message
10: Return success message
11: Return success message
12: Return success message
13: Display success message
14: Success message is displayed
Powered By Visual Paradigm Community Edition

Figure 51: Sequence diagram for the ‘Deactivate a medical expert’ scenario

3.2.4 Class Diagram:

Complete Class Diagram:

The complete class diagram of the Remote Health Service System is shown in Figure 52.

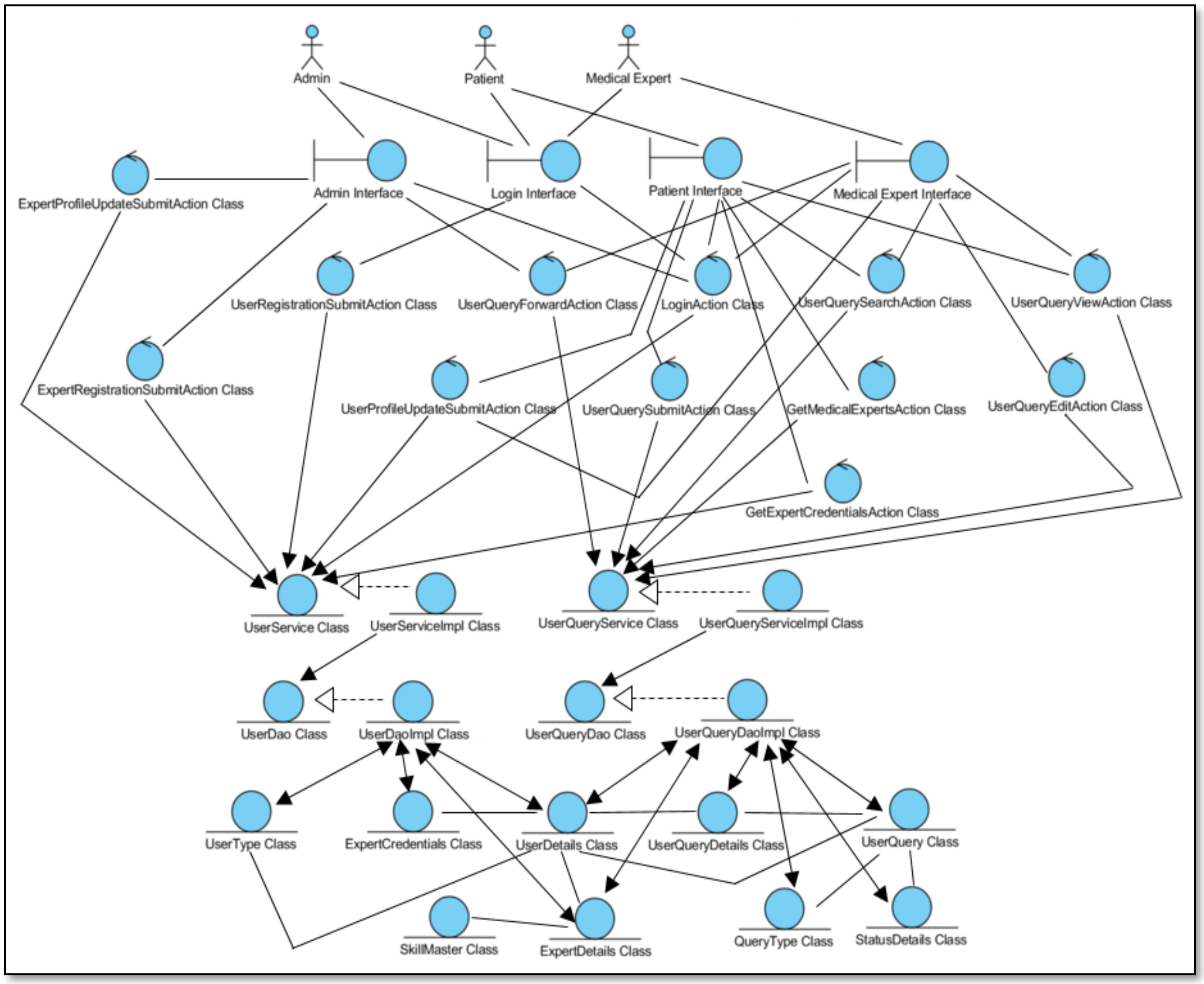


Figure 52: Complete Class diagram of the Remote Health Service System

Part of the Overall Class Diagram with Attributes and Methods:

Part of the overall class diagram showing the Action, Service and Dao classes involved in achieving the ***Admin system*** functionality is shown in Figure 53.

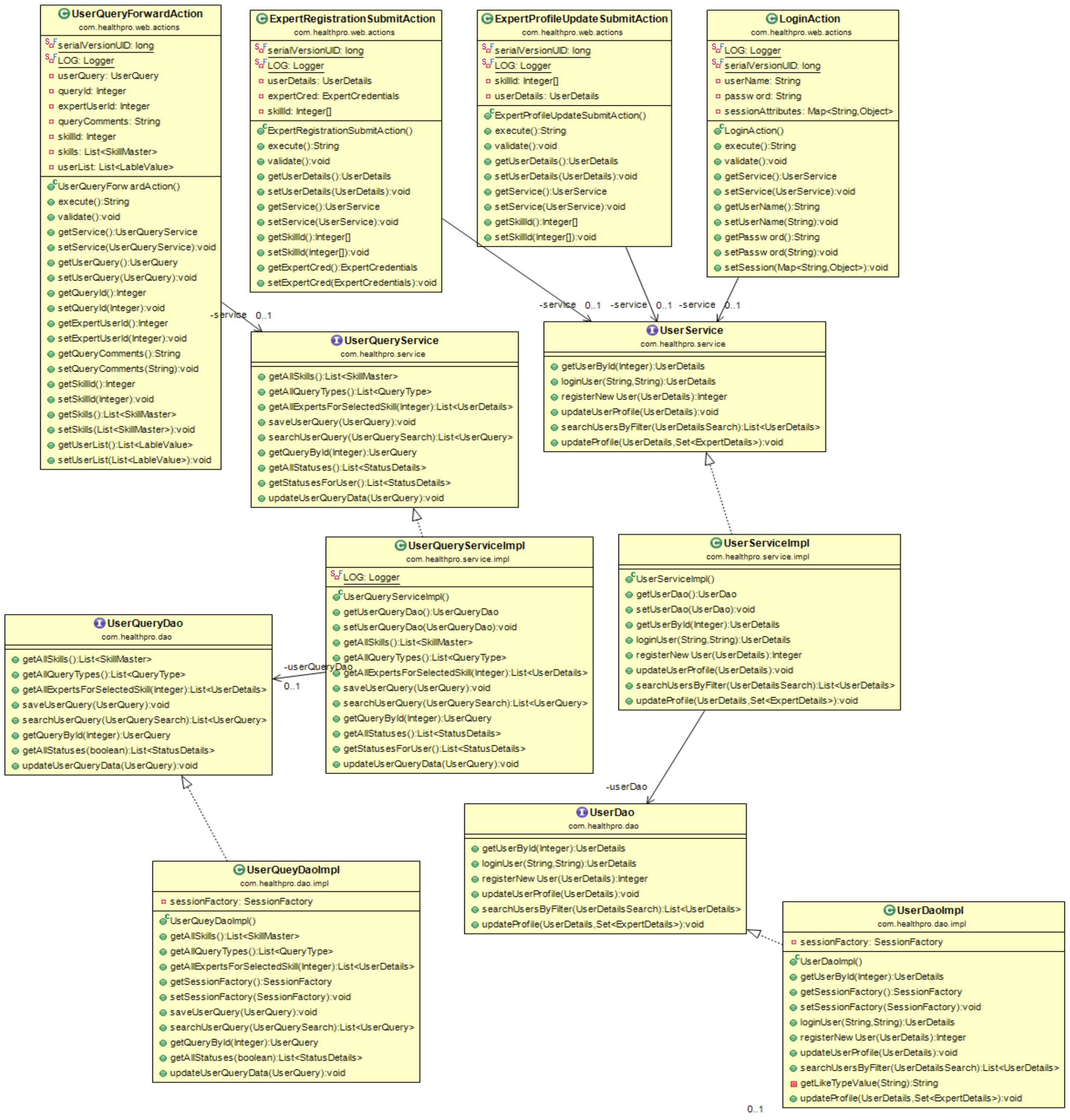
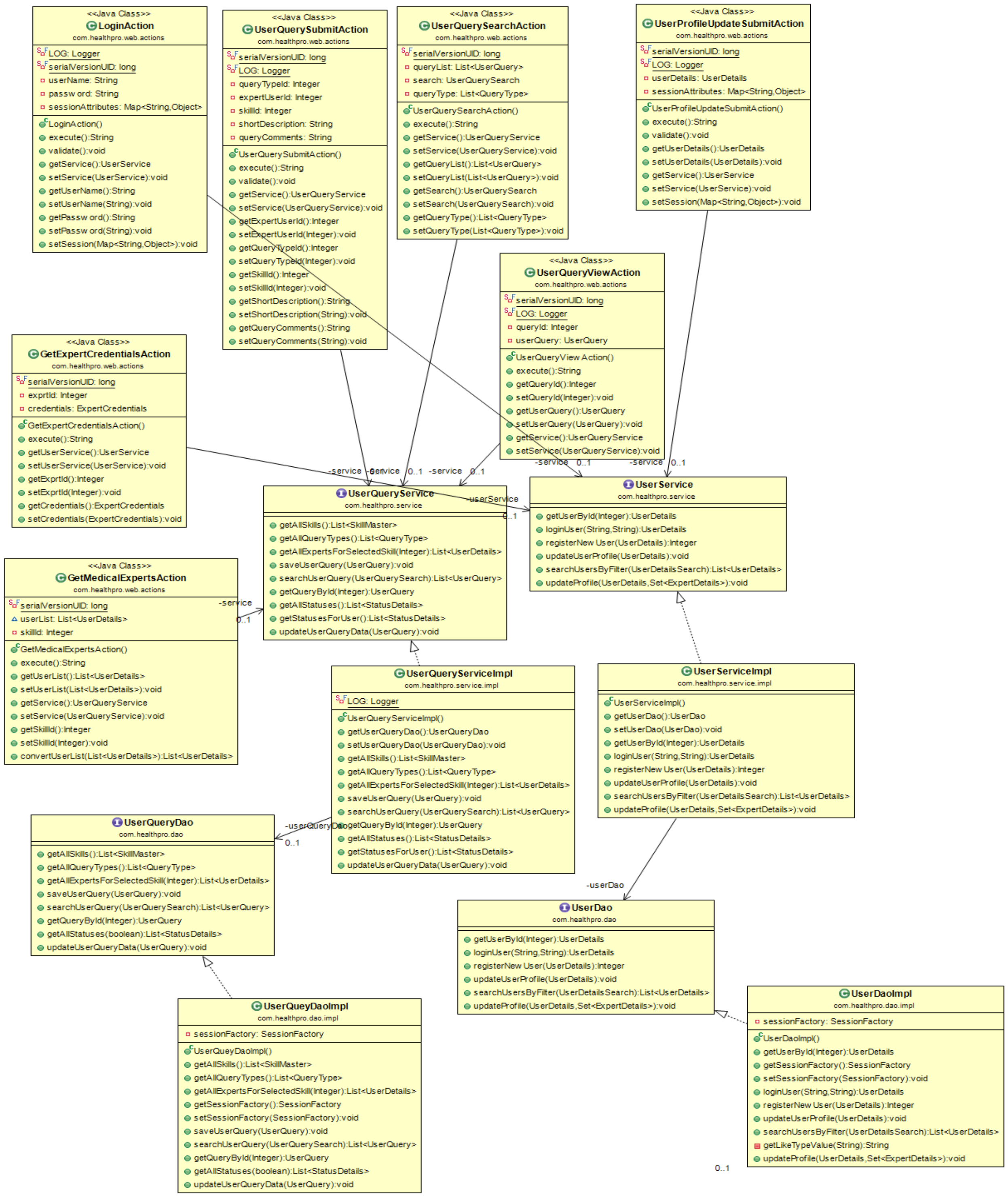

UserQueryForwardAction
com.healthpro.web.actions
serialVersionUID: long
LOG: Logger
userQuery: UserQuery
queryId: Integer
expertUserId: Integer
queryComments: String
skillId: Integer
skills: List<SkillMaster>
userList: List<LableValue>
UserQueryForwardAction()
execute():String
validate():void
getService():UserQueryService
setService(UserQueryService):void
getUserQuery():UserQuery
setUserQuery(UserQuery):void
getQueryId():Integer
setQueryId(Integer):void
getExpertUserId():Integer
setExpertUserId(Integer):void
getQueryComments():String
setQueryComments(String):void
getSkillId():Integer
setSkillId(Integer):void
getSkills():List<SkillMaster>
setSkills(List<SkillMaster>):void
getUserList():List<LableValue>
setUserList(List<LableValue>):void
ExpertRegistrationSubmitAction
com.healthpro.web.actions
serialVersionUID: long
LOG: Logger
userDetails: UserDetails
expertCred: ExpertCredentials
skillId: Integer[]
ExpertRegistrationSubmitAction()
execute():String
validate():void
getUserDetails():UserDetails
setUserDetails(UserDetails):void
getService():UserService
setService(UserService):void
getSkillId():Integer[]
setSkillId(Integer[]):void
getExpertCred():ExpertCredentials
setExpertCred(ExpertCredentials):void
ExpertProfileUpdateSubmitAction
com.healthpro.web.actions
serialVersionUID: long
LOG: Logger
skillId: Integer[]
userDetails: UserDetails
ExpertProfileUpdateSubmitAction()
execute():String
validate():void
getUserDetails():UserDetails
setUserDetails(UserDetails):void
getService():UserService
setService(UserService):void
getSkillId():Integer[]
setSkillId(Integer[]):void
LoginAction
com.healthpro.web.actions
LOG: Logger
serialVersionUID: long
userName: String
password: String
sessionAttributes: Map<String,Object>
LoginAction()
execute():String
validate():void
getService():UserService
setService(UserService):void
getUserName():String
setUserName(String):void
getPassword():String
setPassword(String):void
setSession(Map<String,Object>):void
-service 0..1
-service 0..1
-service 0..1
-service 0..1
UserQueryService
com.healthpro.service
getAllSkills():List<SkillMaster>
getAllQueryTypes():List<QueryType>
getAllExpertsForSelectedSkill(Integer):List<UserDetails>
saveUserQuery(UserQuery):void
searchUserQuery(UserQuerySearch):List<UserQuery>
getQueryById(Integer):UserQuery
getAllStatuses():List<StatusDetails>
getStatusesForUser():List<StatusDetails>
updateUserQueryData(UserQuery):void
UserService
com.healthpro.service
getUserById(Integer):UserDetails
loginUser(String,String):UserDetails
registerNewUser(UserDetails):Integer
updateUserProfile(UserDetails):void
searchUsersByFilter(UserDetailsSearch):List<UserDetails>
updateProfile(UserDetails,Set<ExpertDetails>):void
UserQueryServiceImpl
com.healthpro.service.impl
LOG: Logger
UserQueryServiceImpl()
getUserQueryDao():UserQueryDao
setUserQueryDao(UserQueryDao):void
getAllSkills():List<SkillMaster>
getAllQueryTypes():List<QueryType>
getAllExpertsForSelectedSkill(Integer):List<UserDetails>
saveUserQuery(UserQuery):void
searchUserQuery(UserQuerySearch):List<UserQuery>
getQueryById(Integer):UserQuery
getAllStatuses():List<StatusDetails>
getStatusesForUser():List<StatusDetails>
updateUserQueryData(UserQuery):void
UserServiceImpl
com.healthpro.service.impl
UserServiceImpl()
getUserDao():UserDao
setUserDao(UserDao):void
getUserById(Integer):UserDetails
loginUser(String,String):UserDetails
registerNewUser(UserDetails):Integer
updateUserProfile(UserDetails):void
searchUsersByFilter(UserDetailsSearch):List<UserDetails>
updateProfile(UserDetails,Set<ExpertDetails>):void
UserQueryDao
com.healthpro.dao
getAllSkills():List<SkillMaster>
getAllQueryTypes():List<QueryType>
getAllExpertsForSelectedSkill(Integer):List<UserDetails>
saveUserQuery(UserQuery):void
searchUserQuery(UserQuerySearch):List<UserQuery>
getQueryById(Integer):UserQuery
getAllStatuses(boolean):List<StatusDetails>
updateUserQueryData(UserQuery):void
-userQueryDao
0..1
-userDao
UserDao
com.healthpro.dao
getUserById(Integer):UserDetails
loginUser(String,String):UserDetails
registerNewUser(UserDetails):Integer
updateUserProfile(UserDetails):void
searchUsersByFilter(UserDetailsSearch):List<UserDetails>
updateProfile(UserDetails,Set<ExpertDetails>):void
UserQueyDaoImpl
com.healthpro.dao.impl
sessionFactory: SessionFactory
UserQueyDaoImpl()
getAllSkills():List<SkillMaster>
getAllQueryTypes():List<QueryType>
getAllExpertsForSelectedSkill(Integer):List<UserDetails>
getSessionFactory():SessionFactory
setSessionFactory(SessionFactory):void
saveUserQuery(UserQuery):void
searchUserQuery(UserQuerySearch):List<UserQuery>
getQueryById(Integer):UserQuery
getAllStatuses(boolean):List<StatusDetails>
updateUserQueryData(UserQuery):void
UserDaoImpl
com.healthpro.dao.impl
sessionFactory: SessionFactory
UserDaoImpl()
getUserById(Integer):UserDetails
getSessionFactory():SessionFactory
setSessionFactory(SessionFactory):void
loginUser(String,String):UserDetails
registerNewUser(UserDetails):Integer
updateUserProfile(UserDetails):void
searchUsersByFilter(UserDetailsSearch):List<UserDetails>
getLikeTypeValue(String):String
updateProfile(UserDetails,Set<ExpertDetails>):void
0..1

Figure 53: Part of the Class diagram for Admin System

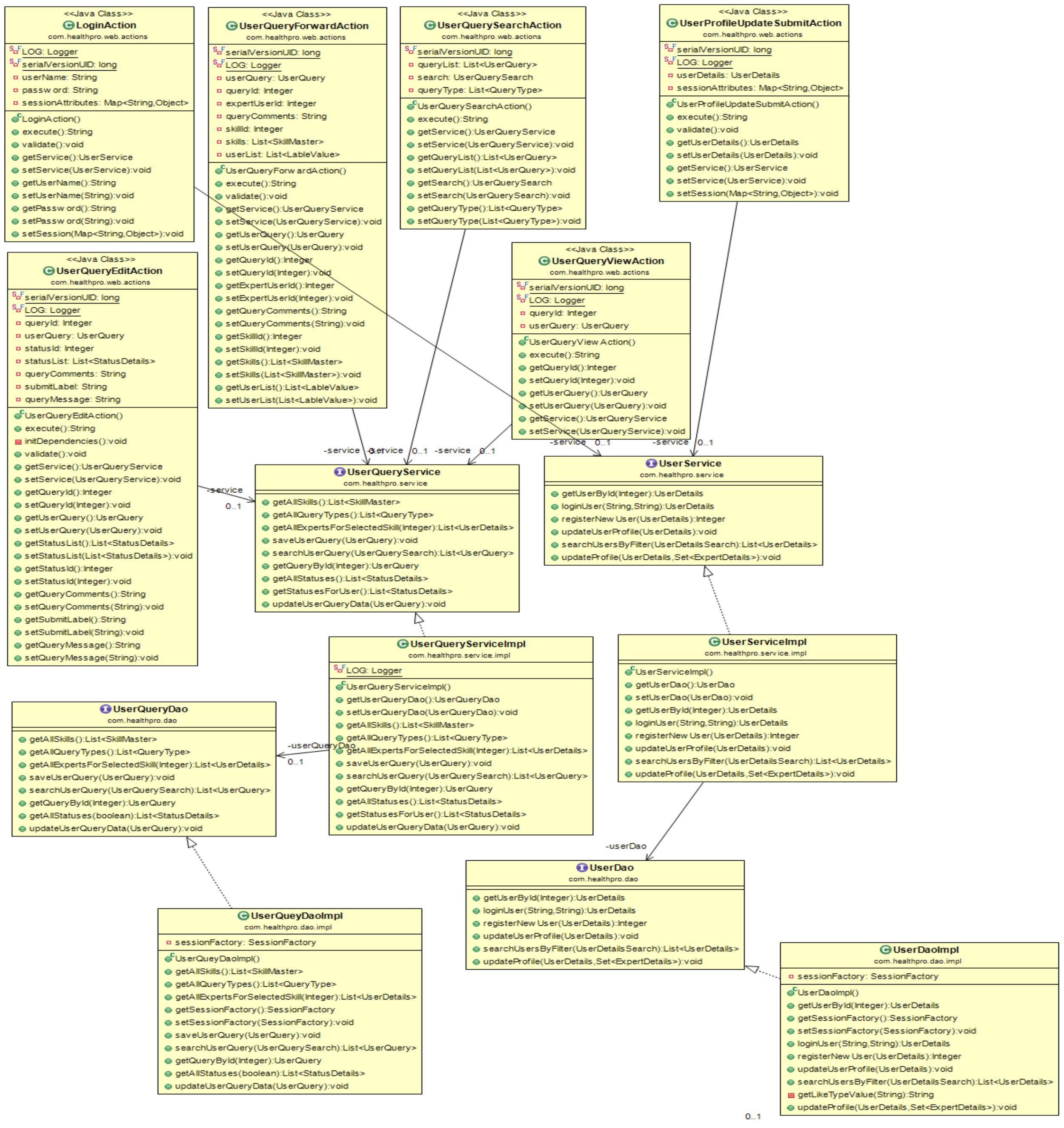
Part of the overall class diagram showing the Action, Service and Dao classes involved in achieving the ***Patient system*** functionality is shown in Figure 54.



<<Java Class>>
LoginAction
com.healthpro.web.actions
LOG: Logger
serialVersionUID: long
userName: String
password: String
sessionAttributes: Map<String,Object>
LoginAction()
execute():String
validate():void
getService():UserService
setService(UserService):void
getUserName():String
setUserName(String):void
getPassword():String
setPassword(String):void
setSession(Map<String,Object>):void
<<Java Class>>
UserQuerySubmitAction
com.healthpro.web.actions
serialVersionUID: long
LOG: Logger
queryTypeId: Integer
expertUserId: Integer
skillId: Integer
shortDescription: String
queryComments: String
UserQuerySubmitAction()
execute():String
validate():void
getService():UserQueryService
setService(UserQueryService):void
getExpertUserId():Integer
setExpertUserId(Integer):void
getQueryTypeId():Integer
setQueryTypeId(Integer):void
getSkillId():Integer
setSkillId(Integer):void
getShortDescription():String
setShortDescription(String):void
getQueryComments():String
setQueryComments(String):void
<<Java Class>>
UserQuerySearchAction
com.healthpro.web.actions
serialVersionUID: long
queryList: List<UserQuery>
search: UserQuerySearch
queryType: List<QueryType>
UserQuerySearchAction()
execute():String
getService():UserQueryService
setService(UserQueryService):void
getQueryList():List<UserQuery>
setQueryList(List<UserQuery>):void
getSearch():UserQuerySearch
setSearch(UserQuerySearch):void
getQueryType():List<QueryType>
setQueryType(List<QueryType>):void
<<Java Class>>
UserProfileUpdateSubmitAction
com.healthpro.web.actions
serialVersionUID: long
LOG: Logger
userDetails: UserDetails
sessionAttributes: Map<String,Object>
UserProfileUpdateSubmitAction()
execute():String
validate():void
getUserDetails():UserDetails
setUserDetails(UserDetails):void
getService():UserService
setService(UserService):void
setSession(Map<String,Object>):void
<<Java Class>>
UserQueryViewAction
com.healthpro.web.actions
serialVersionUID: long
LOG: Logger
queryId: Integer
userQuery: UserQuery
UserQueryViewAction()
execute():String
getQueryId():Integer
setQueryId(Integer):void
getUserQuery():UserQuery
setUserQuery(UserQuery):void
getService():UserQueryService
setService(UserQueryService):void
<<Java Class>>
GetExpertCredentialsAction
com.healthpro.web.actions
serialVersionUID: long
exprtId: Integer
credentials: ExpertCredentials
GetExpertCredentialsAction()
execute():String
getUserService():UserService
setUserService(UserService):void
getExprtId():Integer
setExprtId(Integer):void
getCredentials():ExpertCredentials
setCredentials(ExpertCredentials):void
-service
-service
0..1
-service
0..1
-service
0..1
-service
0..1
-userService
0..1
UserQueryService
com.healthpro.service
getAllSkills():List<SkillMaster>
getAllQueryTypes():List<QueryType>
getAllExpertsForSelectedSkill(Integer):List<UserDetails>
saveUserQuery(UserQuery):void
searchUserQuery(UserQuerySearch):List<UserQuery>
getQueryById(Integer):UserQuery
getAllStatuses():List<StatusDetails>
getStatusesForUser():List<StatusDetails>
updateUserQueryData(UserQuery):void
UserService
com.healthpro.service
getUserById(Integer):UserDetails
loginUser(String,String):UserDetails
registerNewUser(UserDetails):Integer
updateUserProfile(UserDetails):void
searchUsersByFilter(UserDetailsSearch):List<UserDetails>
updateProfile(UserDetails,Set<ExpertDetails>):void
<<Java Class>>
GetMedicalExpertsAction
com.healthpro.web.actions
serialVersionUID: long
userList: List<UserDetails>
skillId: Integer
GetMedicalExpertsAction()
execute():String
getUserList():List<UserDetails>
setUserList(List<UserDetails>):void
getService():UserQueryService
setService(UserQueryService):void
getSkillId():Integer
setSkillId(Integer):void
convertUserList(List<UserDetails>):List<UserDetails>
-service
0..1
UserQueryServiceImpl
com.healthpro.service.impl
LOG: Logger
UserQueryServiceImpl()
getUserQueryDao():UserQueryDao
setUserQueryDao(UserQueryDao):void
getAllSkills():List<SkillMaster>
getAllQueryTypes():List<QueryType>
getAllExpertsForSelectedSkill(Integer):List<UserDetails>
saveUserQuery(UserQuery):void
searchUserQuery(UserQuerySearch):List<UserQuery>
getQueryById(Integer):UserQuery
getAllStatuses():List<StatusDetails>
getStatusesForUser():List<StatusDetails>
updateUserQueryData(UserQuery):void
UserServiceImpl
com.healthpro.service.impl
UserServiceImpl()
getUserDao():UserDao
setUserDao(UserDao):void
getUserById(Integer):UserDetails
loginUser(String,String):UserDetails
registerNewUser(UserDetails):Integer
updateUserProfile(UserDetails):void
searchUsersByFilter(UserDetailsSearch):List<UserDetails>
updateProfile(UserDetails,Set<ExpertDetails>):void
UserQueryDao
com.healthpro.dao
getAllSkills():List<SkillMaster>
getAllQueryTypes():List<QueryType>
getAllExpertsForSelectedSkill(Integer):List<UserDetails>
saveUserQuery(UserQuery):void
searchUserQuery(UserQuerySearch):List<UserQuery>
getQueryById(Integer):UserQuery
getAllStatuses(boolean):List<StatusDetails>
updateUserQueryData(UserQuery):void
-userQueryDao
0..1
-userDao
UserDao
com.healthpro.dao
getUserById(Integer):UserDetails
loginUser(String,String):UserDetails
registerNewUser(UserDetails):Integer
updateUserProfile(UserDetails):void
searchUsersByFilter(UserDetailsSearch):List<UserDetails>
updateProfile(UserDetails,Set<ExpertDetails>):void
UserQueyDaoImpl
com.healthpro.dao.impl
sessionFactory: SessionFactory
UserQueyDaoImpl()
getAllSkills():List<SkillMaster>
getAllQueryTypes():List<QueryType>
getAllExpertsForSelectedSkill(Integer):List<UserDetails>
getSessionFactory():SessionFactory
setSessionFactory(SessionFactory):void
saveUserQuery(UserQuery):void
searchUserQuery(UserQuerySearch):List<UserQuery>
getQueryById(Integer):UserQuery
getAllStatuses(boolean):List<StatusDetails>
updateUserQueryData(UserQuery):void
UserDaoImpl
com.healthpro.dao.impl
sessionFactory: SessionFactory
UserDaoImpl()
getUserById(Integer):UserDetails
getSessionFactory():SessionFactory
setSessionFactory(SessionFactory):void
loginUser(String,String):UserDetails
registerNewUser(UserDetails):Integer
updateUserProfile(UserDetails):void
searchUsersByFilter(UserDetailsSearch):List<UserDetails>
getLikeTypeValue(String):String
updateProfile(UserDetails,Set<ExpertDetails>):void
0..1

Figure 54: Part of the Class diagram for Patient System

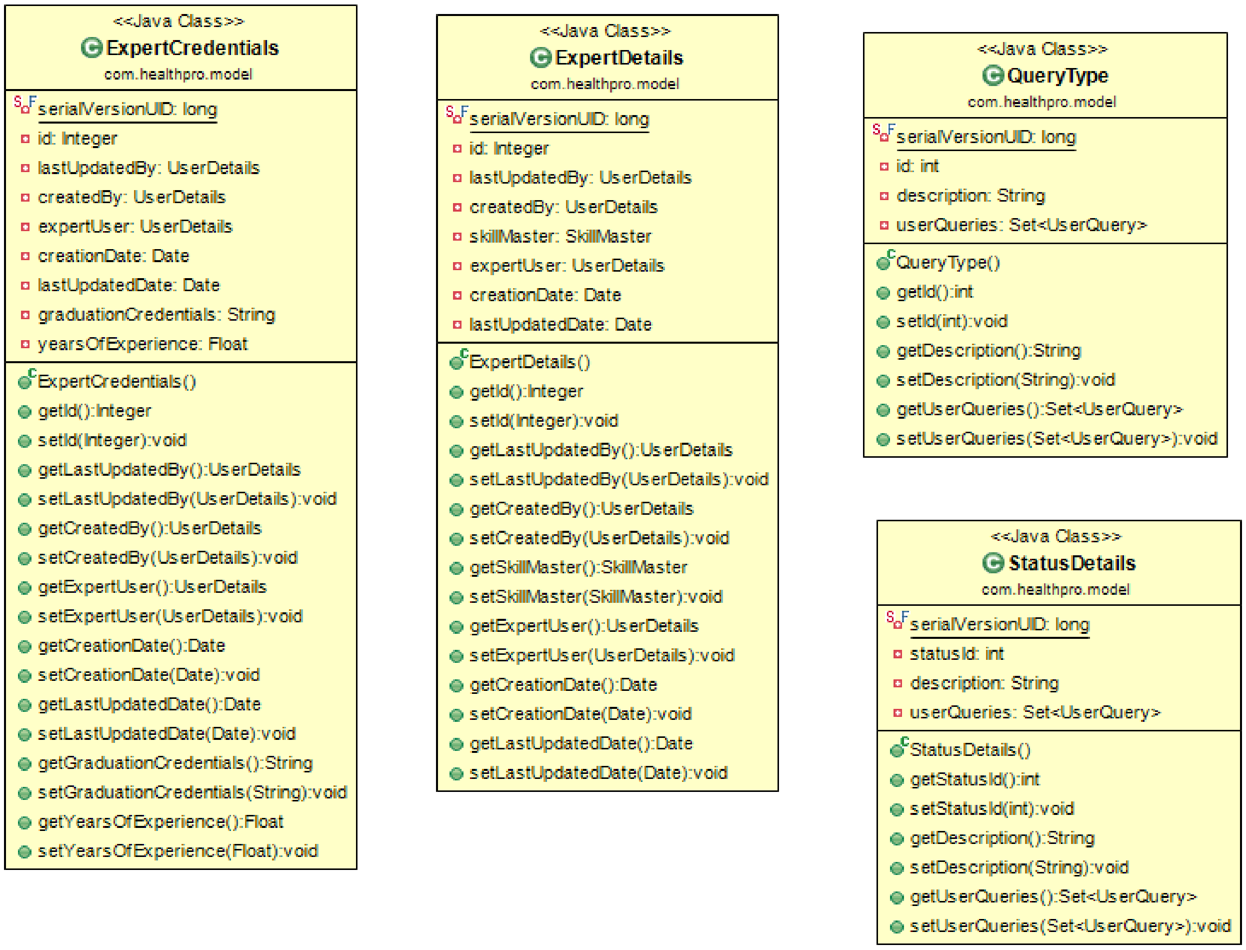
Part of the overall class diagram showing the Action, Service and Dao classes involved in achieving the ***Medical Expert system*** functionality is shown in Figure 55.

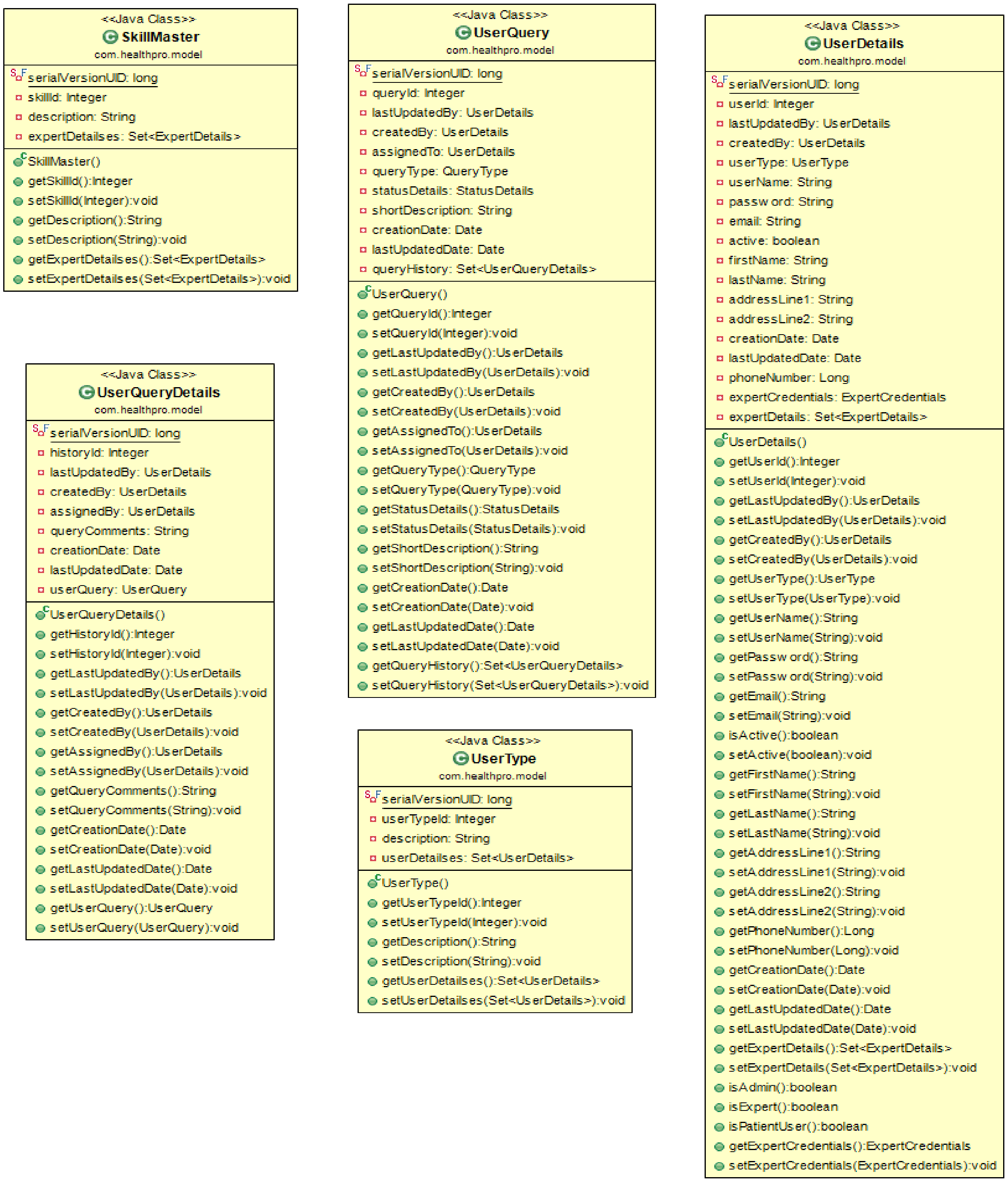


<<Java Class>>
LoginAction
com.healthpro.web.actions
LOG: Logger
serialVersionUID: long
userName: String
password: String
sessionAttributes: Map<String,Object>
LoginAction()
execute():String
validate():void
getService():UserService
setService(UserService):void
getUserName():String
setUserName(String):void
getPassword():String
setPassword(String):void
setSession(Map<String,Object>):void
<<Java Class>>
UserQueryForwardAction
com.healthpro.web.actions
serialVersionUID: long
LOG: Logger
userQuery: UserQuery
queryId: Integer
expertUserId: Integer
queryComments: String
skillId: Integer
skills: List<SkillMaster>
userList: List<LableValue>
UserQueryForwardAction()
execute():String
validate():void
getService():UserQueryService
setService(UserQueryService):void
getUserQuery():UserQuery
setUserQuery(UserQuery):void
getQueryId():Integer
setQueryId(Integer):void
getExpertUserId():Integer
setExpertUserId(Integer):void
getQueryComments():String
setQueryComments(String):void
getSkillId():Integer
setSkillId(Integer):void
getSkills():List<SkillMaster>
setSkills(List<SkillMaster>):void
getUserList():List<LableValue>
setUserList(List<LableValue>):void
<<Java Class>>
UserQuerySearchAction
com.healthpro.web.actions
serialVersionUID: long
queryList: List<UserQuery>
search: UserQuerySearch
queryType: List<QueryType>
UserQuerySearchAction()
execute():String
getService():UserQueryService
setService(UserQueryService):void
getQueryList():List<UserQuery>
setQueryList(List<UserQuery>):void
getSearch():UserQuerySearch
setSearch(UserQuerySearch):void
getQueryType():List<QueryType>
setQueryType(List<QueryType>):void
<<Java Class>>
UserProfileUpdateSubmitAction
com.healthpro.web.actions
serialVersionUID: long
LOG: Logger
userDetails: UserDetails
sessionAttributes: Map<String,Object>
UserProfileUpdateSubmitAction()
execute():String
validate():void
getUserDetails():UserDetails
setUserDetails(UserDetails):void
getService():UserService
setService(UserService):void
setSession(Map<String,Object>):void
<<Java Class>>
UserQueryEditAction
com.healthpro.web.actions
serialVersionUID: long
LOG: Logger
queryId: Integer
userQuery: UserQuery
statusId: Integer
statusList: List<StatusDetails>
queryComments: String
submitLabel: String
queryMessage: String
UserQueryEditAction()
execute():String
initDependencies():void
validate():void
getService():UserQueryService
setService(UserQueryService):void
getQueryId():Integer
setQueryId(Integer):void
getUserQuery():UserQuery
setUserQuery(UserQuery):void
getStatusList():List<StatusDetails>
setStatusList(List<StatusDetails>):void
getStatusId():Integer
setStatusId(Integer):void
getQueryComments():String
setQueryComments(String):void
getSubmitLabel():String
setSubmitLabel(String):void
getQueryMessage():String
setQueryMessage(String):void
<<Java Class>>
UserQueryViewAction
com.healthpro.web.actions
serialVersionUID: long
LOG: Logger
queryId: Integer
userQuery: UserQuery
UserQueryViewAction()
execute():String
getQueryId():Integer
setQueryId(Integer):void
getUserQuery():UserQuery
setUserQuery(UserQuery):void
getService():UserQueryService
setService(UserQueryService):void
-service
0..1
-service
0..1
-service
0..1
-service
0..1
-service
0..1
-service
0..1
UserQueryService
com.healthpro.service
getAllSkills():List<SkillMaster>
getAllQueryTypes():List<QueryType>
getAllExpertsForSelectedSkill(Integer):List<UserDetails>
saveUserQuery(UserQuery):void
searchUserQuery(UserQuerySearch):List<UserQuery>
getQueryById(Integer):UserQuery
getAllStatuses():List<StatusDetails>
getStatusesForUser():List<StatusDetails>
updateUserQueryData(UserQuery):void
UserService
com.healthpro.service
getUserById(Integer):UserDetails
loginUser(String,String):UserDetails
registerNewUser(UserDetails):Integer
updateUserProfile(UserDetails):void
searchUsersByFilter(UserDetailsSearch):List<UserDetails>
updateProfile(UserDetails,Set<ExpertDetails>):void
UserQueryServiceImpl
com.healthpro.service.impl
LOG: Logger
UserQueryServiceImpl()
getUserQueryDao():UserQueryDao
setUserQueryDao(UserQueryDao):void
getAllSkills():List<SkillMaster>
getAllQueryTypes():List<QueryType>
getAllExpertsForSelectedSkill(Integer):List<UserDetails>
saveUserQuery(UserQuery):void
searchUserQuery(UserQuerySearch):List<UserQuery>
getQueryById(Integer):UserQuery
getAllStatuses():List<StatusDetails>
getStatusesForUser():List<StatusDetails>
updateUserQueryData(UserQuery):void
UserServiceImpl
com.healthpro.service.impl
UserServiceImpl()
getUserDao():UserDao
setUserDao(UserDao):void
getUserById(Integer):UserDetails
loginUser(String,String):UserDetails
registerNewUser(UserDetails):Integer
updateUserProfile(UserDetails):void
searchUsersByFilter(UserDetailsSearch):List<UserDetails>
updateProfile(UserDetails,Set<ExpertDetails>):void
UserQueryDao
com.healthpro.dao
getAllSkills():List<SkillMaster>
getAllQueryTypes():List<QueryType>
getAllExpertsForSelectedSkill(Integer):List<UserDetails>
saveUserQuery(UserQuery):void
searchUserQuery(UserQuerySearch):List<UserQuery>
getQueryById(Integer):UserQuery
getAllStatuses(boolean):List<StatusDetails>
updateUserQueryData(UserQuery):void
-userQueryDao
0..1
-userDao
UserDao
com.healthpro.dao
getUserById(Integer):UserDetails
loginUser(String,String):UserDetails
registerNewUser(UserDetails):Integer
updateUserProfile(UserDetails):void
searchUsersByFilter(UserDetailsSearch):List<UserDetails>
updateProfile(UserDetails,Set<ExpertDetails>):void
UserQueyDaoImpl
com.healthpro.dao.impl
sessionFactory: SessionFactory
UserQueyDaoImpl()
getAllSkills():List<SkillMaster>
getAllQueryTypes():List<QueryType>
getAllExpertsForSelectedSkill(Integer):List<UserDetails>
getSessionFactory():SessionFactory
setSessionFactory(SessionFactory):void
saveUserQuery(UserQuery):void
searchUserQuery(UserQuerySearch):List<UserQuery>
getQueryById(Integer):UserQuery
getAllStatuses(boolean):List<StatusDetails>
updateUserQueryData(UserQuery):void
UserDaoImpl
com.healthpro.dao.impl
sessionFactory: SessionFactory
UserDaoImpl()
getUserById(Integer):UserDetails
getSessionFactory():SessionFactory
setSessionFactory(SessionFactory):void
loginUser(String,String):UserDetails
registerNewUser(UserDetails):Integer
updateUserProfile(UserDetails):void
searchUsersByFilter(UserDetailsSearch):List<UserDetails>
getLikeTypeValue(String):String
updateProfile(UserDetails,Set<ExpertDetails>):void
0..1

Figure 55: Part of the Class diagram for Medical Expert System

Part of the overall class diagram showing the Model classes involved in the implementation of the Remote Health Service System are shown in Figure 56.


<<Java Class>>
ExpertCredentials
com.healthpro.model
serialVersionUID: long
id: Integer
lastUpdatedBy: UserDetails
createdBy: UserDetails
expertUser: UserDetails
creationDate: Date
lastUpdatedDate: Date
graduationCredentials: String
yearsOfExperience: Float
ExpertCredentials()
getId():Integer
setId(Integer):void
getLastUpdatedBy():UserDetails
setLastUpdatedBy(UserDetails):void
getCreatedBy():UserDetails
setCreatedBy(UserDetails):void
getExpertUser():UserDetails
setExpertUser(UserDetails):void
getCreationDate():Date
setCreationDate(Date):void
getLastUpdatedDate():Date
setLastUpdatedDate(Date):void
getGraduationCredentials():String
setGraduationCredentials(String):void
getYearsOfExperience():Float
setYearsOfExperience(Float):void
<<Java Class>>
ExpertDetails
com.healthpro.model
serialVersionUID: long
id: Integer
lastUpdatedBy: UserDetails
createdBy: UserDetails
skillMaster: SkillMaster
expertUser: UserDetails
creationDate: Date
lastUpdatedDate: Date
ExpertDetails()
getId():Integer
setId(Integer):void
getLastUpdatedBy():UserDetails
setLastUpdatedBy(UserDetails):void
getCreatedBy():UserDetails
setCreatedBy(UserDetails):void
getSkillMaster():SkillMaster
setSkillMaster(SkillMaster):void
getExpertUser():UserDetails
setExpertUser(UserDetails):void
getCreationDate():Date
setCreationDate(Date):void
getLastUpdatedDate():Date
setLastUpdatedDate(Date):void
<<Java Class>>
QueryType
com.healthpro.model
serialVersionUID: long
id: int
description: String
userQueries: Set<UserQuery>
QueryType()
getId():int
setId(int):void
getDescription():String
setDescription(String):void
getUserQueries():Set<UserQuery>
setUserQueries(Set<UserQuery>):void
<<Java Class>>
StatusDetails
com.healthpro.model
serialVersionUID: long
statusId: int
description: String
userQueries: Set<UserQuery>
StatusDetails()
getStatusId():int
setStatusId(int):void
getDescription():String
setDescription(String):void
getUserQueries():Set<UserQuery>
setUserQueries(Set<UserQuery>):void



<<Java Class>>
SkillMaster
com.healthpro.model
serialVersionUID: long
skillId: Integer
description: String
expertDetailses: Set<ExpertDetails>
SkillMaster()
getSkillId():Integer
setSkillId(Integer):void
getDescription():String
setDescription(String):void
getExpertDetailses():Set<ExpertDetails>
setExpertDetailses(Set<ExpertDetails>):void
<<Java Class>>
UserQueryDetails
com.healthpro.model
serialVersionUID: long
historyId: Integer
lastUpdatedBy: UserDetails
createdBy: UserDetails
assignedBy: UserDetails
queryComments: String
creationDate: Date
lastUpdatedDate: Date
userQuery: UserQuery
UserQueryDetails()
getHistoryId():Integer
setHistoryId(Integer):void
getLastUpdatedBy():UserDetails
setLastUpdatedBy(UserDetails):void
getCreatedBy():UserDetails
setCreatedBy(UserDetails):void
getAssignedBy():UserDetails
setAssignedBy(UserDetails):void
getQueryComments():String
setQueryComments(String):void
getCreationDate():Date
setCreationDate(Date):void
getLastUpdatedDate():Date
setLastUpdatedDate(Date):void
getUserQuery():UserQuery
setUserQuery(UserQuery):void
<<Java Class>>
UserQuery
com.healthpro.model
serialVersionUID: long
queryId: Integer
lastUpdatedBy: UserDetails
createdBy: UserDetails
assignedTo: UserDetails
queryType: QueryType
statusDetails: StatusDetails
shortDescription: String
creationDate: Date
lastUpdatedDate: Date
queryHistory: Set<UserQueryDetails>
UserQuery()
getQueryId():Integer
setQueryId(Integer):void
getLastUpdatedBy():UserDetails
setLastUpdatedBy(UserDetails):void
getCreatedBy():UserDetails
setCreatedBy(UserDetails):void
getAssignedTo():UserDetails
setAssignedTo(UserDetails):void
getQueryType():QueryType
setQueryType(QueryType):void
getStatusDetails():StatusDetails
setStatusDetails(StatusDetails):void
getShortDescription():String
setShortDescription(String):void
getCreationDate():Date
setCreationDate(Date):void
getLastUpdatedDate():Date
setLastUpdatedDate(Date):void
getQueryHistory():Set<UserQueryDetails>
setQueryHistory(Set<UserQueryDetails>):void
<<Java Class>>
UserType
com.healthpro.model
serialVersionUID: long
userTypeId: Integer
description: String
userDetailses: Set<UserDetails>
UserType()
getUserTypeId():Integer
setUserTypeId(Integer):void
getDescription():String
setDescription(String):void
getUserDetailses():Set<UserDetails>
setUserDetailses(Set<UserDetails>):void
<<Java Class>>
UserDetails
com.healthpro.model
serialVersionUID: long
userId: Integer
lastUpdatedBy: UserDetails
createdBy: UserDetails
userType: UserType
userName: String
password: String
email: String
active: boolean
firstName: String
lastName: String
addressLine1: String
addressLine2: String
creationDate: Date
lastUpdatedDate: Date
phoneNumber: Long
expertCredentials: ExpertCredentials
expertDetails: Set<ExpertDetails>
UserDetails()
getUserId():Integer
setUserId(Integer):void
getLastUpdatedBy():UserDetails
setLastUpdatedBy(UserDetails):void
getCreatedBy():UserDetails
setCreatedBy(UserDetails):void
getUserType():UserType
setUserType(UserType):void
getUserName():String
setUserName(String):void
getPassword():String
setPassword(String):void
getEmail():String
setEmail(String):void
isActive():boolean
setActive(boolean):void
getFirstName():String
setFirstName(String):void
getLastName():String
setLastName(String):void
getAddressLine1():String
setAddressLine1(String):void
getAddressLine2():String
setAddressLine2(String):void
getPhoneNumber():Long
setPhoneNumber(Long):void
getCreationDate():Date
setCreationDate(Date):void
getLastUpdatedDate():Date
setLastUpdatedDate(Date):void
getExpertDetails():Set<ExpertDetails>
setExpertDetails(Set<ExpertDetails>):void
isAdmin():boolean
isExpert():boolean
isPatientUser():boolean
getExpertCredentials():ExpertCredentials
setExpertCredentials(ExpertCredentials):void

Figure 56: Part of the Class diagram showing Model Classes

3.2.5 Application Architecture

The Remote Health Service System has Multi Layered architecture. The system has four layers: *Presentation layer*, *Controller layer*, *Service layer* and *Data Access Layer*.

The application architecture is shown in Figure 57.

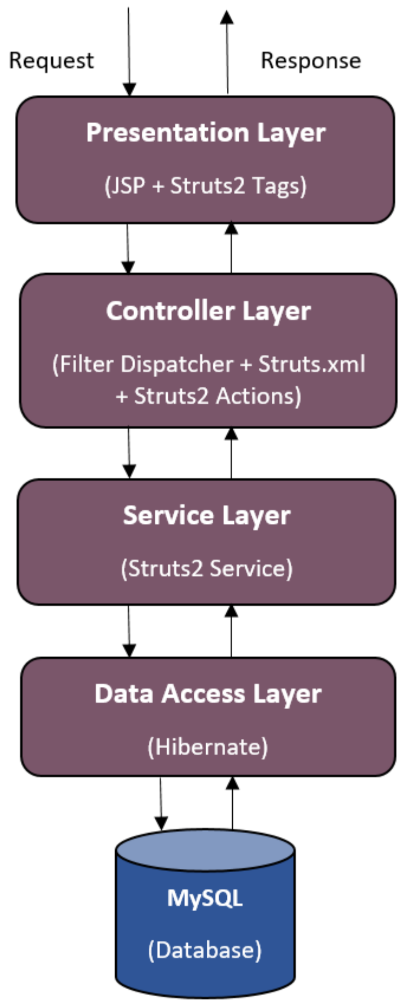


Figure 57: Architecture of Remote Health Service System

The different layers in the Remote Health Service System are described below:

- **Presentation Layer:** This layer is the View part in the MVC design pattern. It comprises of the application screens that help user interact with the system. It is implemented using

JSP (Java Server Pages) and Struts2 tags. JSP is a technology that is used to create dynamic web pages based on HTML (Hyper Text Markup Language) and XML (eXtensible Markup Language). Struts2 offers powerful UI (User Interface) tags to render complex application data effectively. The request submitted by the user at this layer is passed to the controller layer for further processing.

- **Controller Layer:** This layer corresponds to the Controller section in the MVC pattern. It controls the interaction between the presentation layer and the service layer. It handles the user requests and forwards them to the responsible service. Also, invokes the appropriate view resources based on the response returned from the service layer. In the current project, this layer is implemented using Struts2 Filter Dispatcher and XML based configuration file (struts.xml). In the 'struts.xml' file, the mapping between the URI's (Uniform Resource Identifier) and the Action classes is defined. The Filter Dispatcher verifies the request URI and determines which action to invoke. The Action classes invoke the services to process the user request.
- **Service Layer:** This layer consists of the business logic of the application. It is invoked by the controller layer to execute the business functionality. A service is the smallest amount of work that needs to be completed in order to maintain the application database consistent. The concept of Java interfaces and their implementation classes is used in this project to implement this layer. This makes the application flexible as only a change in the implementation class would be needed to modify the application behavior and no change would be required in other layers consuming the services. This layer interacts with the Data Access layer to complete the processing of user request.

- **Data Access Layer:** This layer is the Model layer of the MVC design pattern. It performs the database operations. The logic for code persistence is written in this layer using Hibernate API (Application Program Interface). The entity classes are written which are mapped to the database tables. The data persistence is carried out using the entity class objects. It interacts with the MySQL database to persist and retrieve the application data. The DAO interfaces list the operations that can be performed on the data and their implementation classes contain the execution logic for those operations.

3.3 Application Screenshots

The screenshots of the different functionalities of the Remote Health Service System are presented below:

Login Screen:

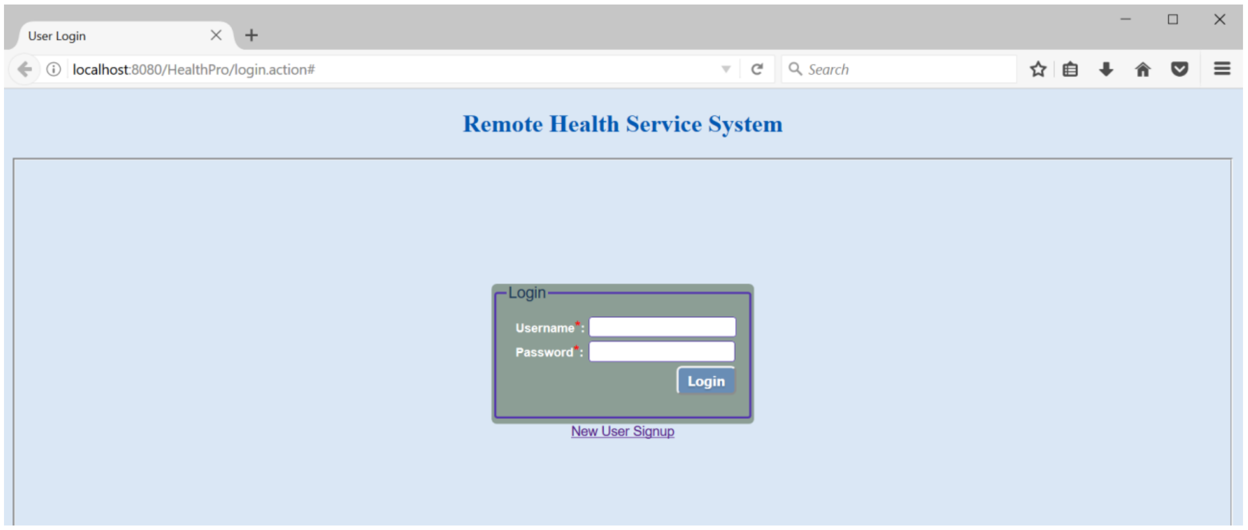
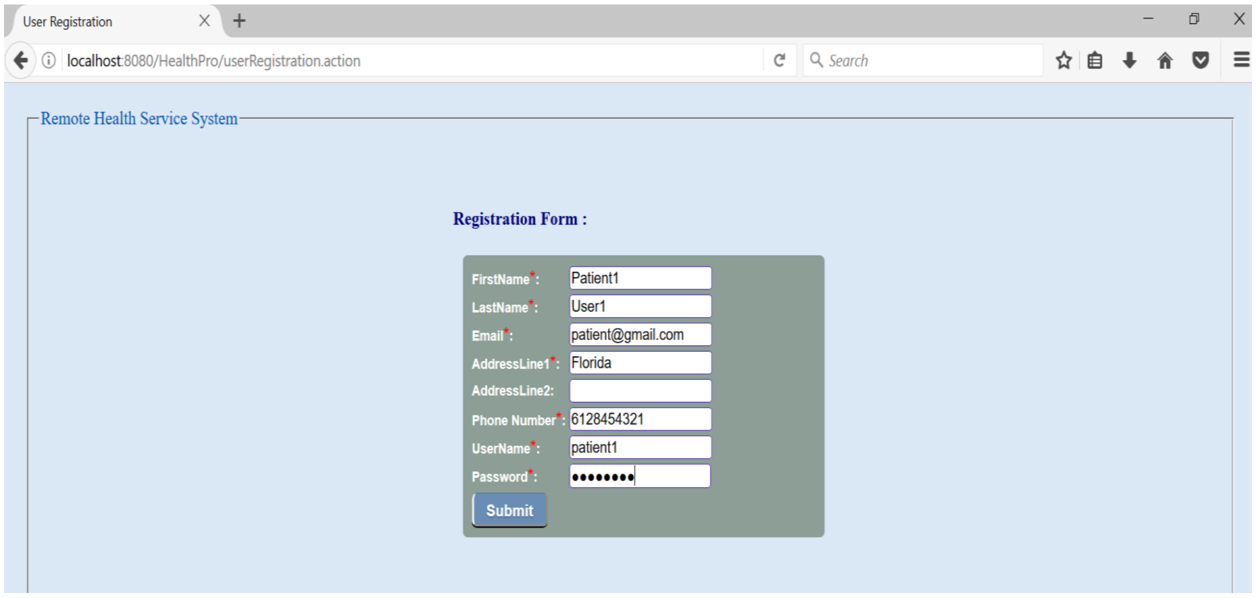

User Login
localhost:8080/HealthPro/login.action#
Search
Remote Health Service System
Login
Username* :
Password* :
Login
New User Signup

Figure 58: Login Screen of the Remote Health Service System

Create Account Screen:


User Registration
localhost:8080/HealthPro/userRegistration.action
Search
Remote Health Service System

Registration Form :

FirstName*:	Patient1
LastName*:	User1
Email*:	patient@gmail.com
AddressLine1*:	Florida
AddressLine2:	
Phone Number*:	6128454321
UserName*:	patient1
Password*:	••••••••

Figure 59: Create Account Screen of the Remote Health Service System

Patient Screen:

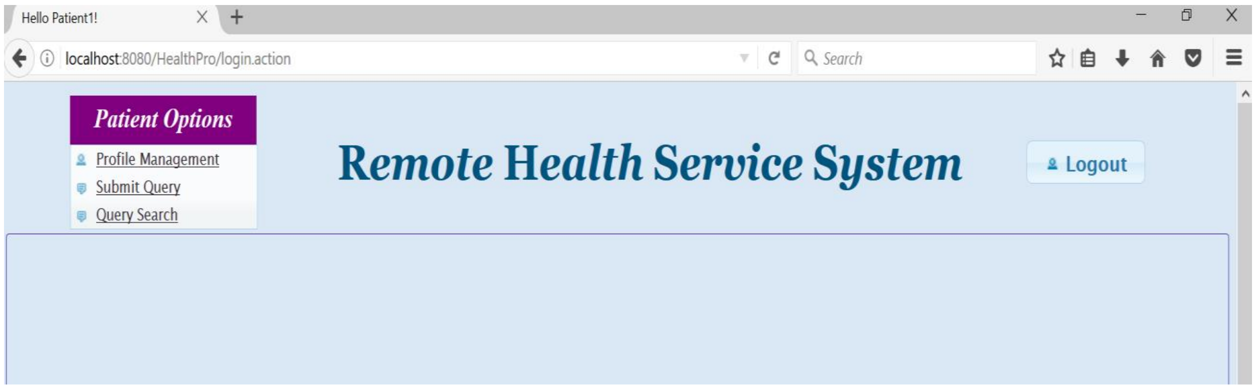
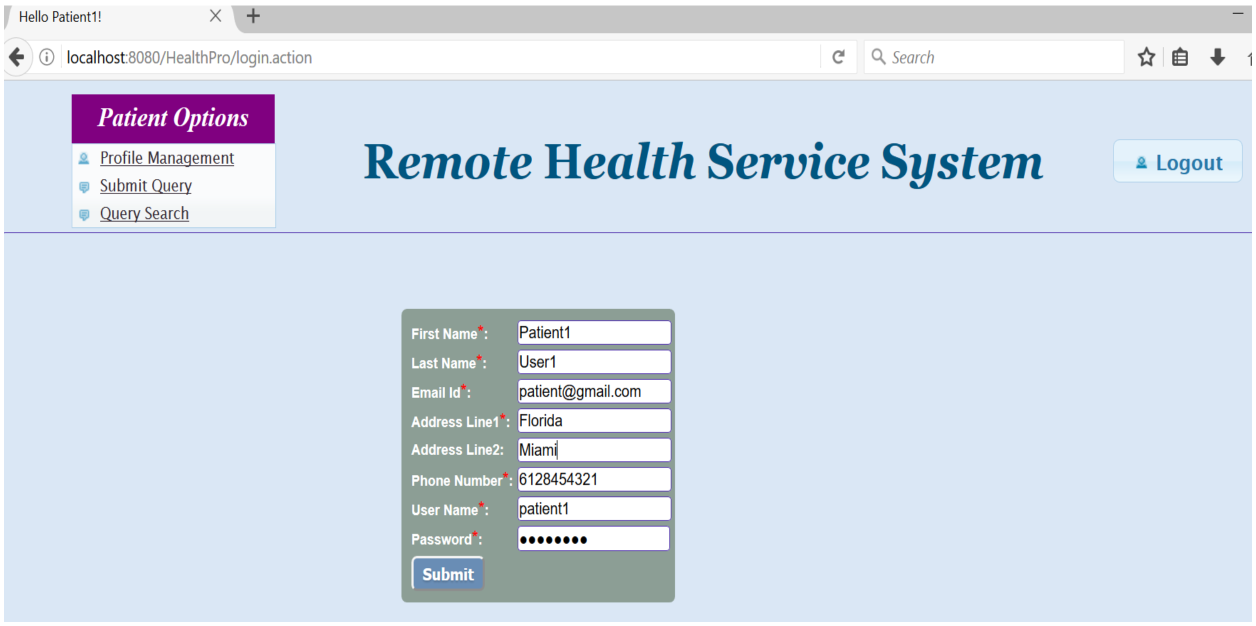


Figure 60: Patient Screen of the Remote Health Service System

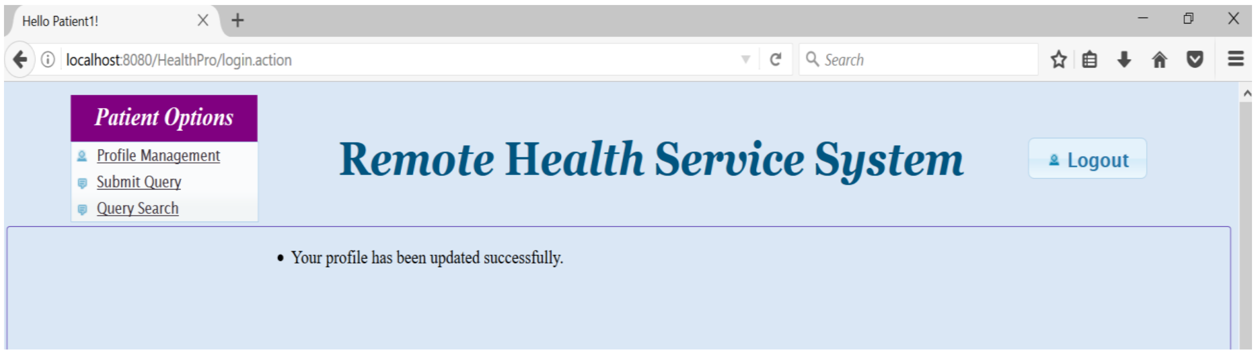
Edit Profile – Patient Screen:



First Name*:	Patient1
Last Name*:	User1
Email Id*:	patient@gmail.com
Address Line1*:	Florida
Address Line2:	Miami
Phone Number*:	6128454321
User Name*:	patient1
Password*:	••••••••

Figure 61: Edit Profile – Patient Screen

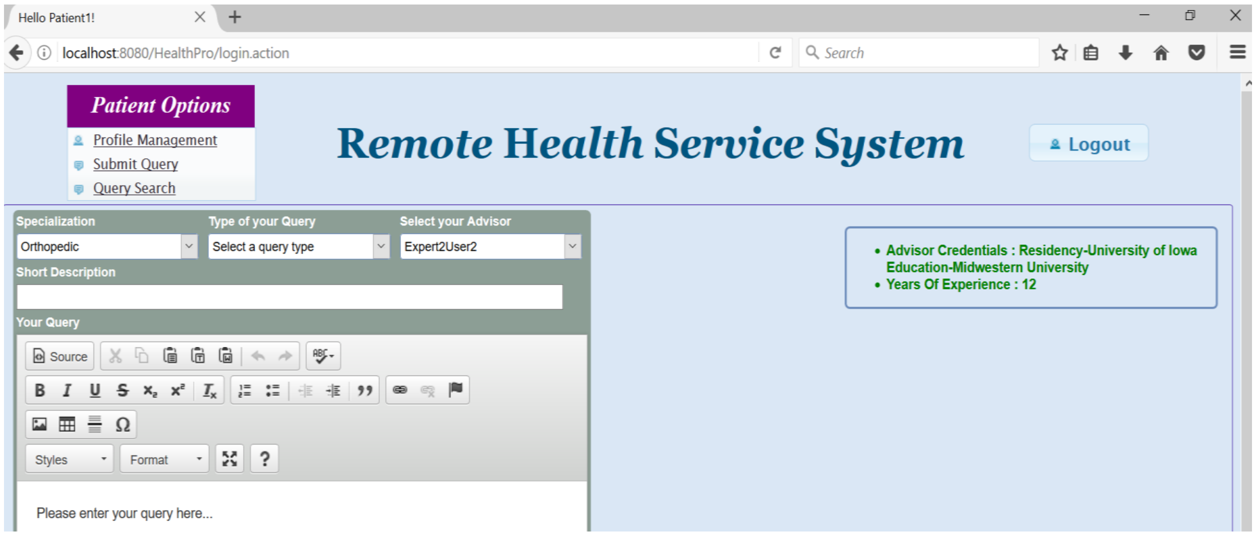
Edit Profile – Patient Screen (Success message):



- Your profile has been updated successfully.

Figure 62: Edit Profile – Patient Screen (Success message)

View Medical Expert Information – Patient Screen:

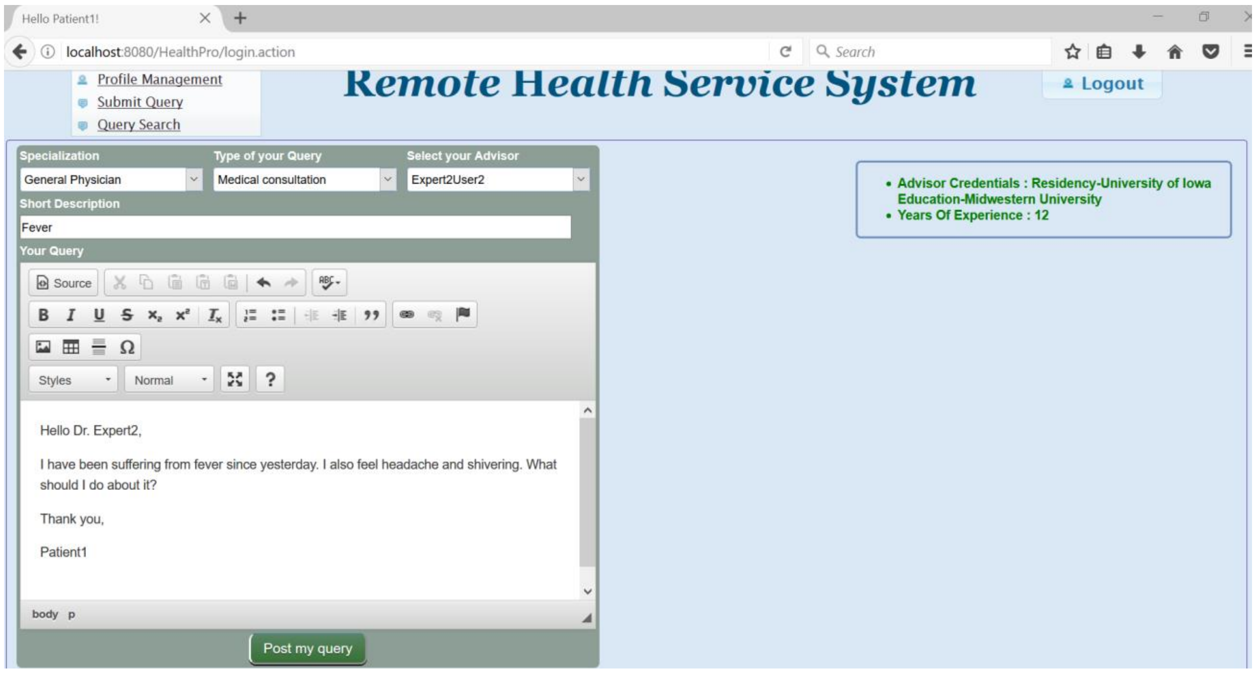


Specialization: Orthopedic
Type of your Query: Select a query type
Select your Advisor: Expert2User2
Short Description
Your Query
Please enter your query here...

- Advisor Credentials : Residency-University of Iowa Education-Midwestern University
- Years Of Experience : 12

Figure 63: View Medical Expert Information – Patient Screen

Submit Query – Patient Screen:



Specialization: General Physician
Type of your Query: Medical consultation
Select your Advisor: Expert2User2
Short Description: Fever
Your Query

Hello Dr. Expert2,

I have been suffering from fever since yesterday. I also feel headache and shivering. What should I do about it?

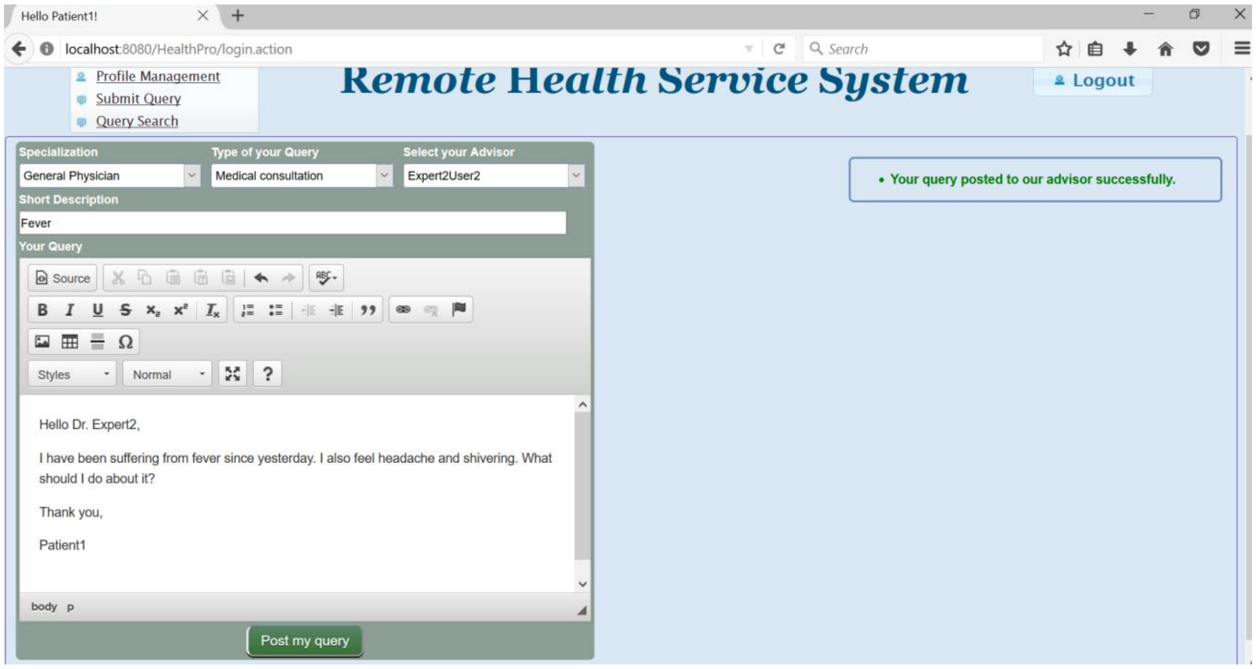
Thank you,

Patient1

- Advisor Credentials : Residency-University of Iowa Education-Midwestern University
- Years Of Experience : 12

Figure 64: Submit Query – Patient Screen

Submit Query – Patient Screen (Success message):



Specialization: General Physician
Type of your Query: Medical consultation
Select your Advisor: Expert2User2
Short Description: Fever

Your Query:

Hello Dr. Expert2,

I have been suffering from fever since yesterday. I also feel headache and shivering. What should I do about it?

Thank you,

Patient1

• Your query posted to our advisor successfully.

Figure 65: Submit Query – Patient Screen (Success message)

Medical Expert Screen:

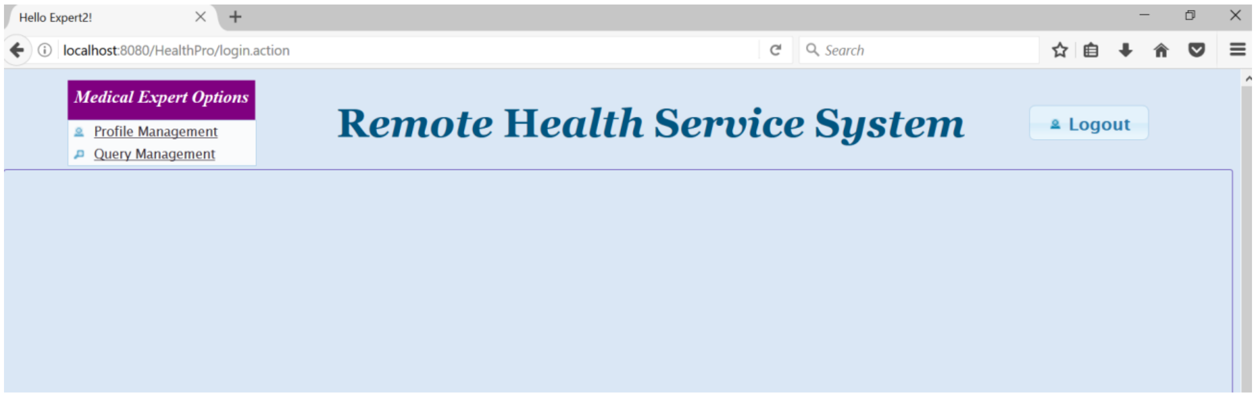
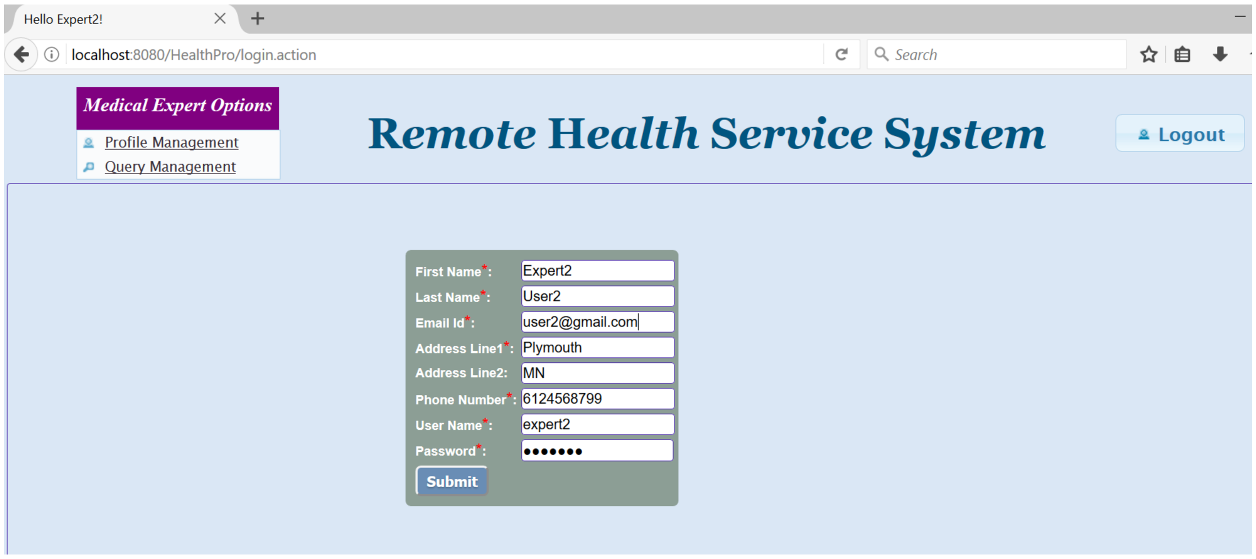


Figure 66: Medical Expert Screen of the Remote Health Service System

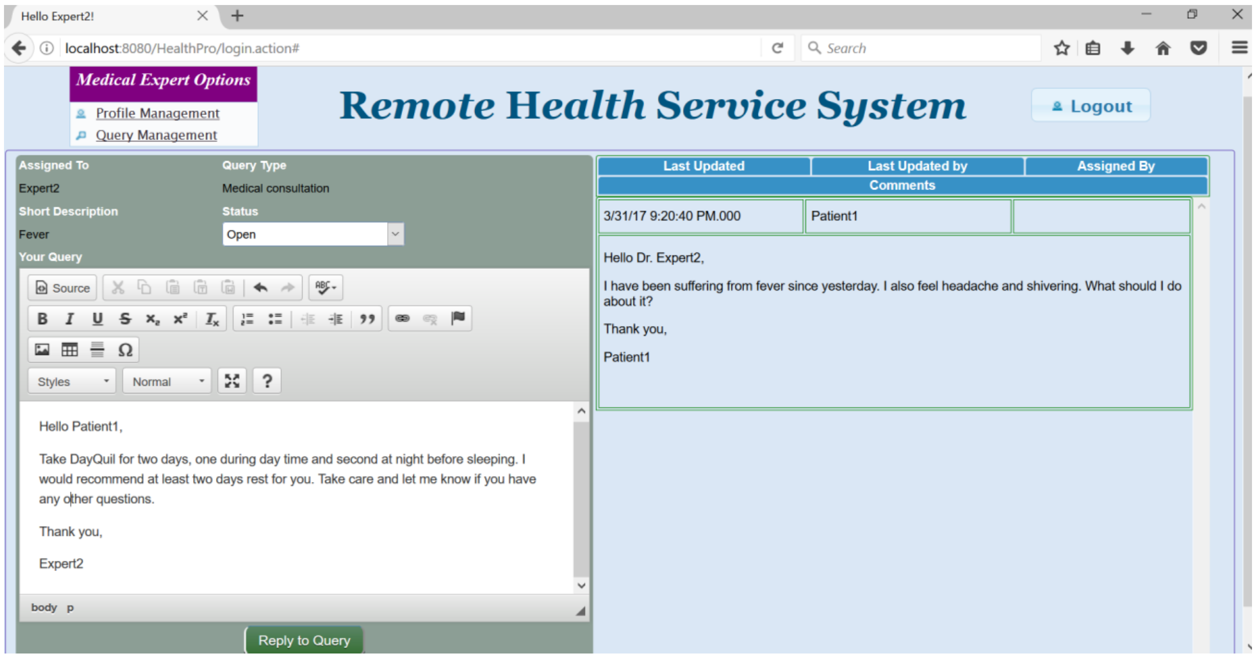
Edit Profile – Medical Expert Screen:



First Name*:	Expert2
Last Name*:	User2
Email Id*:	user2@gmail.com
Address Line1*:	Plymouth
Address Line2:	MN
Phone Number*:	6124568799
User Name*:	expert2
Password*:	•••••••

Figure 67: Edit Profile – Medical Expert Screen

Respond to Query – Medical Expert Screen:



Assigned To: Expert2
Query Type: Medical consultation
Short Description: Fever
Status: Open
Your Query

Hello Patient1,

Take DayQuil for two days, one during day time and second at night before sleeping. I would recommend at least two days rest for you. Take care and let me know if you have any other questions.

Thank you,

Expert2

Last Updated	Last Updated by	Assigned By
3/31/17 9:20:40 PM.000	Patient1	

Comments

Hello Dr. Expert2,

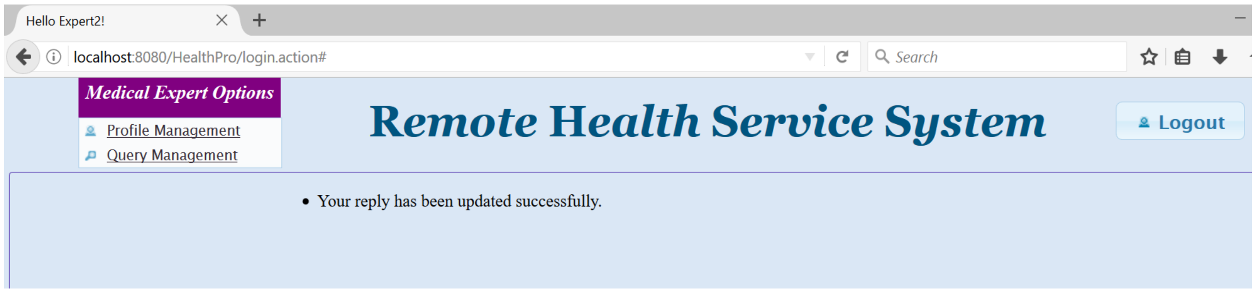
I have been suffering from fever since yesterday. I also feel headache and shivering. What should I do about it?

Thank you,

Patient1

Figure 68: Respond to Query – Medical Expert Screen

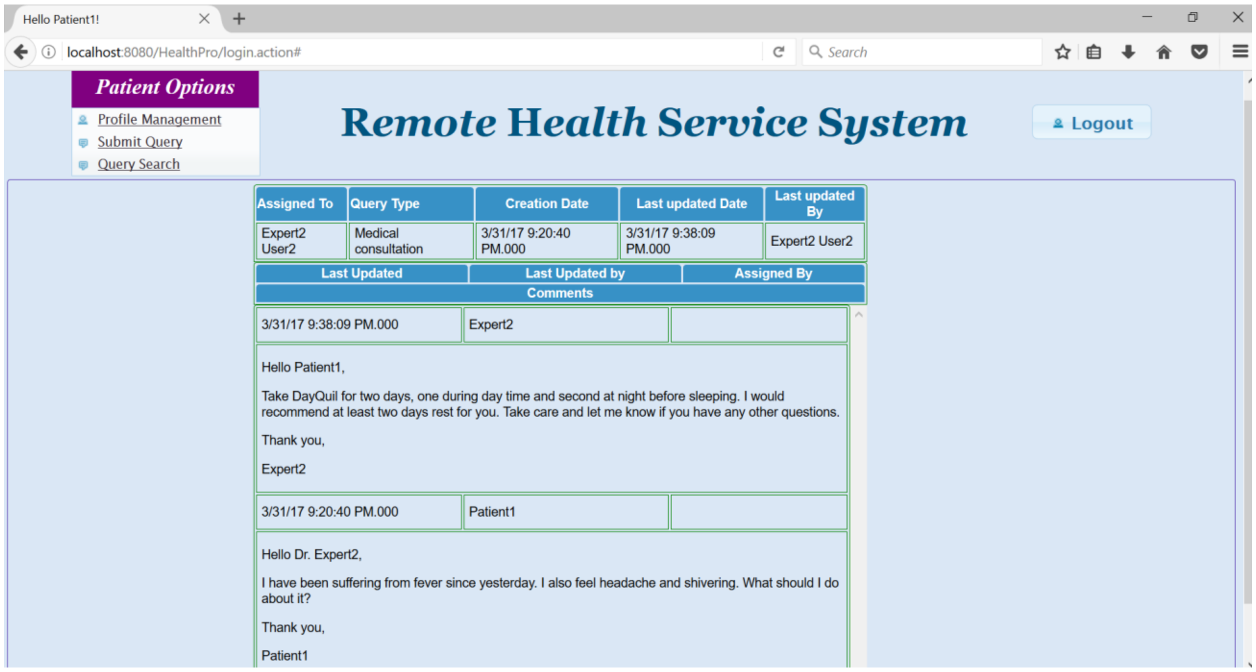
Respond to Query – Medical Expert Screen (Success message):



- Your reply has been updated successfully.

Figure 69: Respond to Query – Medical Expert Screen (Success message)

View Response – Patient Screen:



Assigned To	Query Type	Creation Date	Last updated Date	Last updated By
Expert2 User2	Medical consultation	3/31/17 9:20:40 PM.000	3/31/17 9:38:09 PM.000	Expert2 User2

Last Updated	Last Updated by	Assigned By
3/31/17 9:38:09 PM.000	Expert2	

Comments:

Hello Patient1,

Take DayQuil for two days, one during day time and second at night before sleeping. I would recommend at least two days rest for you. Take care and let me know if you have any other questions.

Thank you,

Expert2

Last Updated	Last Updated by	Assigned By
3/31/17 9:20:40 PM.000	Patient1	

Hello Dr. Expert2,

I have been suffering from fever since yesterday. I also feel headache and shivering. What should I do about it?

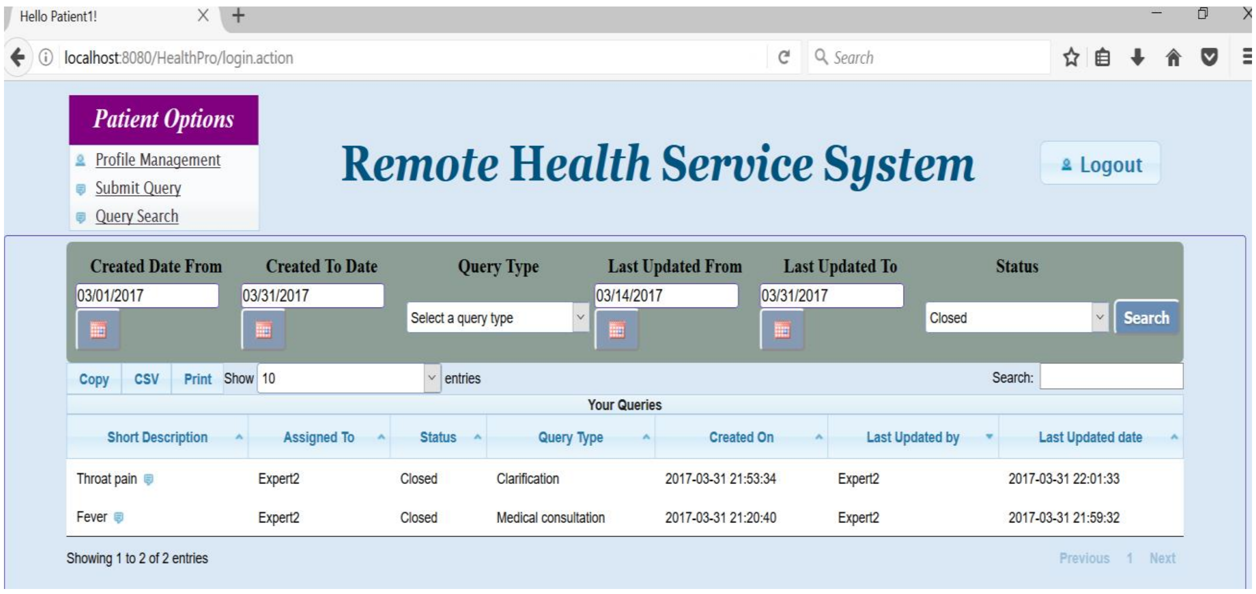
Thank you,

Patient1

Figure 70: View Response – Patient Screen

View Past Medical Queries – Patient and Medical Expert:

The Patient and Medical Expert can search for past medical queries using this screen. They can also view the query details by clicking on the ‘View’ icon as shown in Figures 71 and 72.



Created Date From	Created To Date	Query Type	Last Updated From	Last Updated To	Status
03/01/2017	03/31/2017	Select a query type	03/14/2017	03/31/2017	Closed

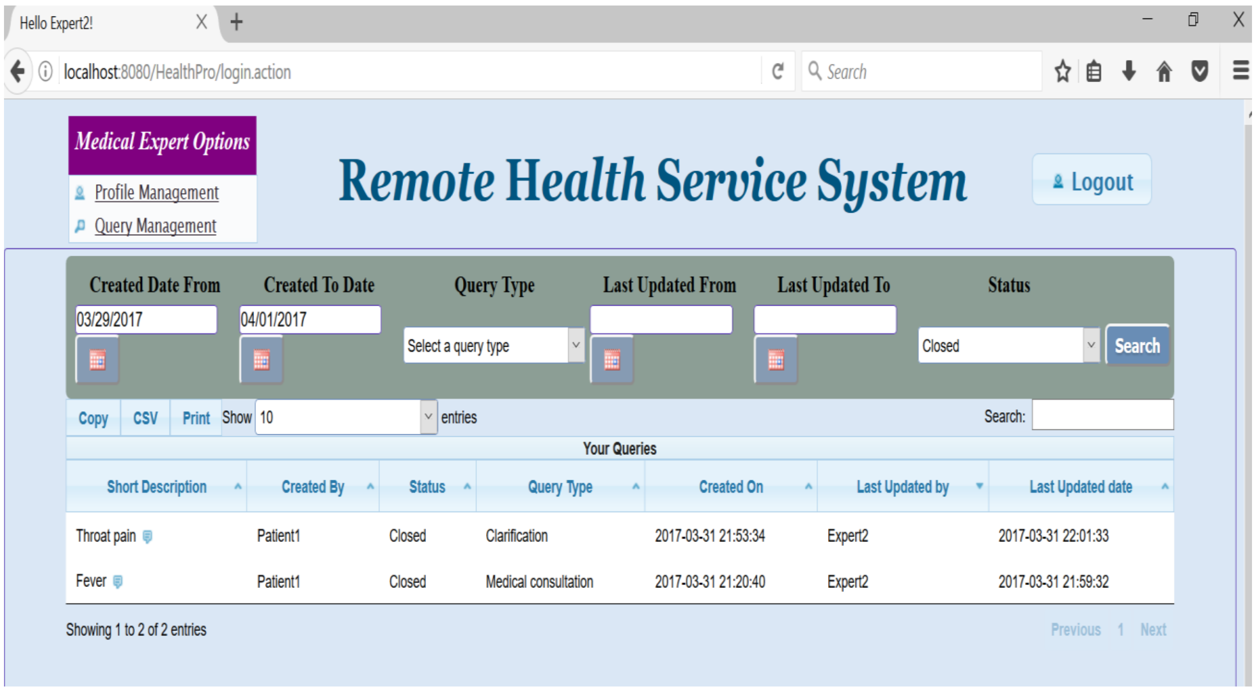
Show 10 entries Search:

Your Queries

Short Description	Assigned To	Status	Query Type	Created On	Last Updated by	Last Updated date
Throat pain	Expert2	Closed	Clarification	2017-03-31 21:53:34	Expert2	2017-03-31 22:01:33
Fever	Expert2	Closed	Medical consultation	2017-03-31 21:20:40	Expert2	2017-03-31 21:59:32

Showing 1 to 2 of 2 entries Previous 1 Next

Figure 71: View Past Medical Queries – Patient Screen



Created Date From	Created To Date	Query Type	Last Updated From	Last Updated To	Status
03/29/2017	04/01/2017	Select a query type			Closed

Show 10 entries Search:

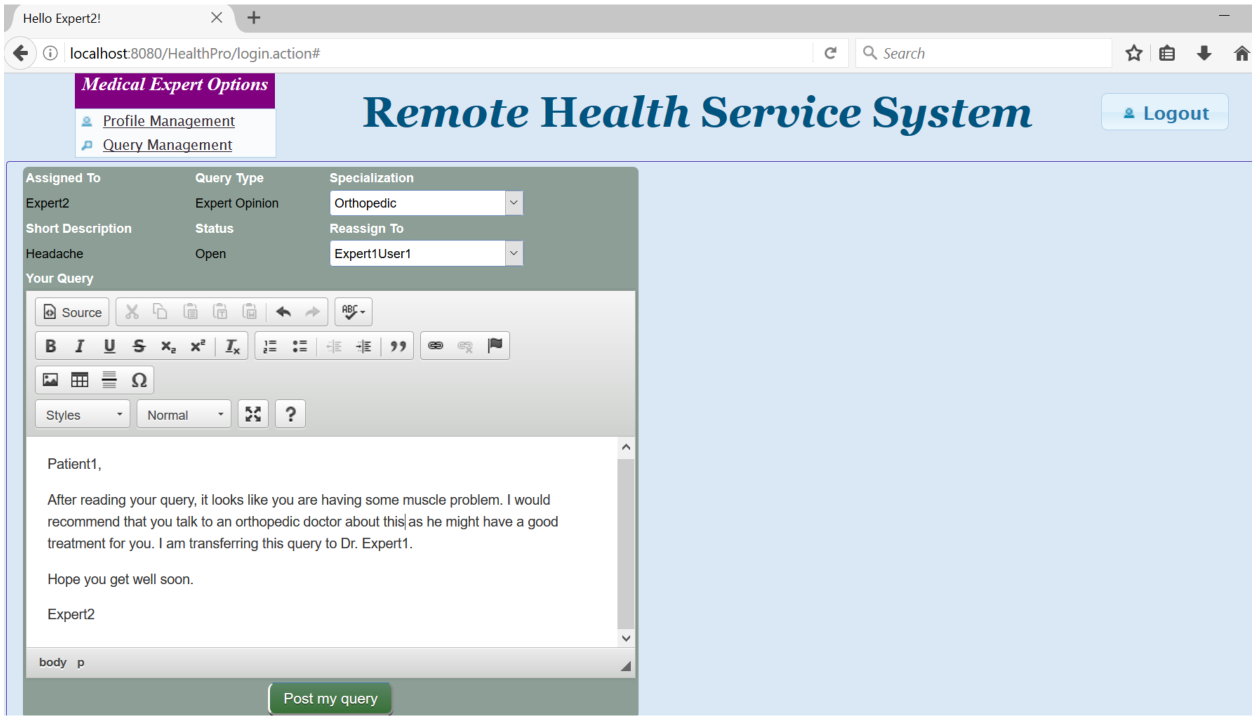
Your Queries

Short Description	Created By	Status	Query Type	Created On	Last Updated by	Last Updated date
Throat pain	Patient1	Closed	Clarification	2017-03-31 21:53:34	Expert2	2017-03-31 22:01:33
Fever	Patient1	Closed	Medical consultation	2017-03-31 21:20:40	Expert2	2017-03-31 21:59:32

Showing 1 to 2 of 2 entries Previous 1 Next

Figure 72: View Past Medical Queries – Medical Expert Screen

Transfer Query – Medical Expert:



Assigned To	Query Type	Specialization
Expert2	Expert Opinion	Orthopedic
Short Description	**Status**	**Reassign To**
Headache	Open	Expert1User1

Your Query

Patient1,

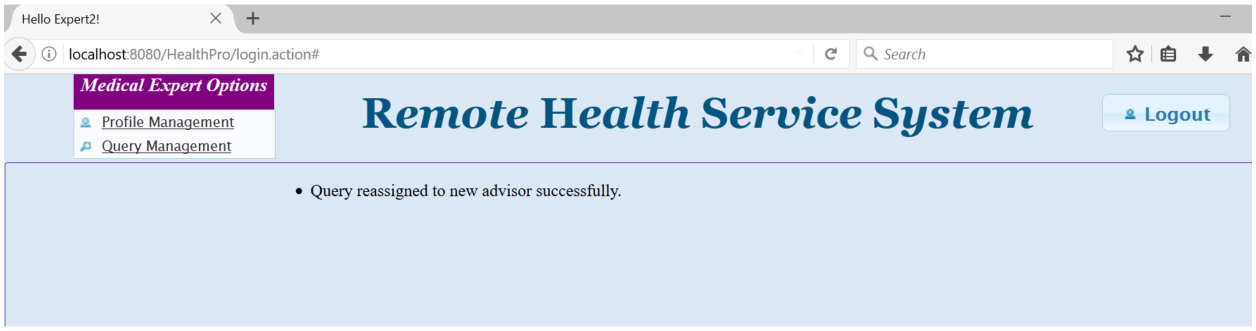
After reading your query, it looks like you are having some muscle problem. I would recommend that you talk to an orthopedic doctor about this as he might have a good treatment for you. I am transferring this query to Dr. Expert1.

Hope you get well soon.

Expert2

Figure 73: Transfer Query – Medical Expert

Transfer Query – Medical Expert (Success message):



- Query reassigned to new advisor successfully.

Figure 74: Transfer Query – Medical Expert (Success message)

Admin Screen:

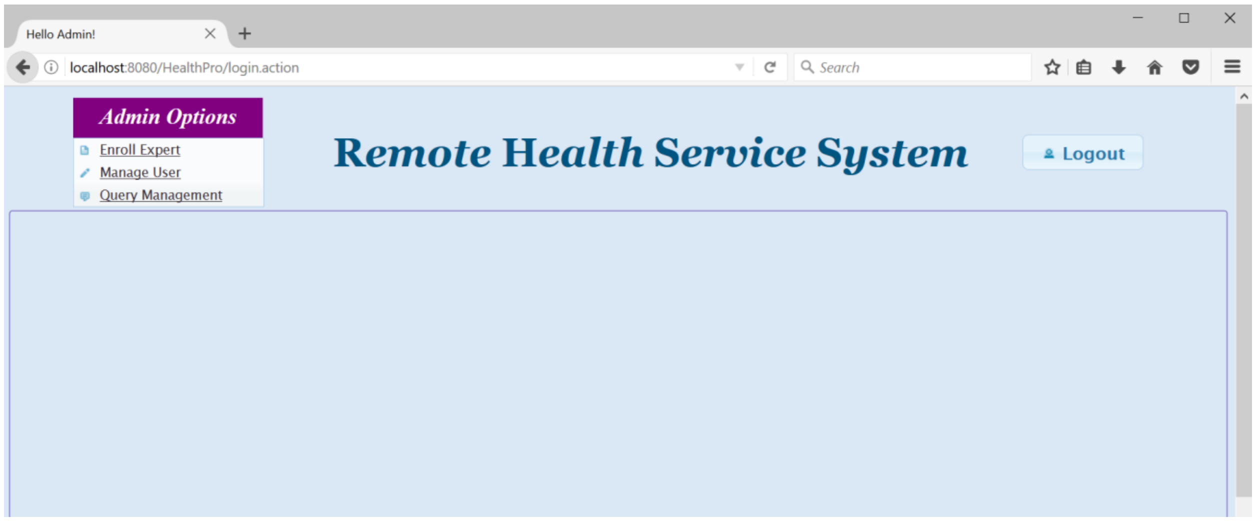
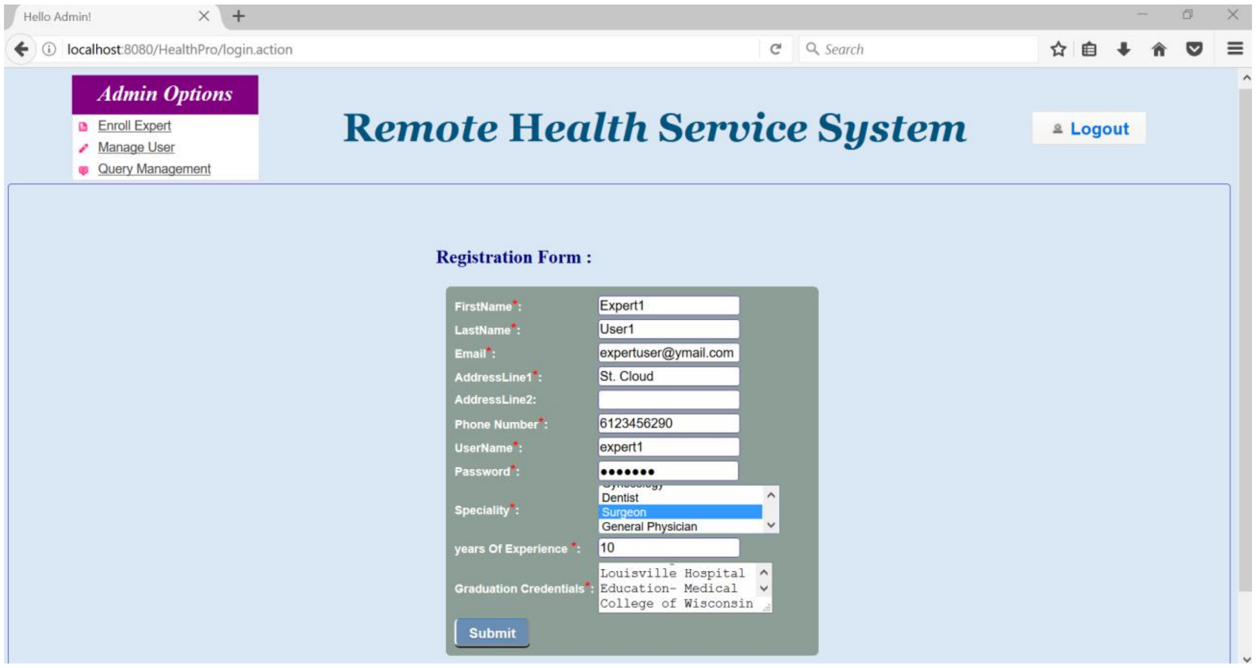


Figure 75: Admin Screen of the Remote Health Service System

Manage Medical Experts (Add a new Medical Expert) – Admin Screen:

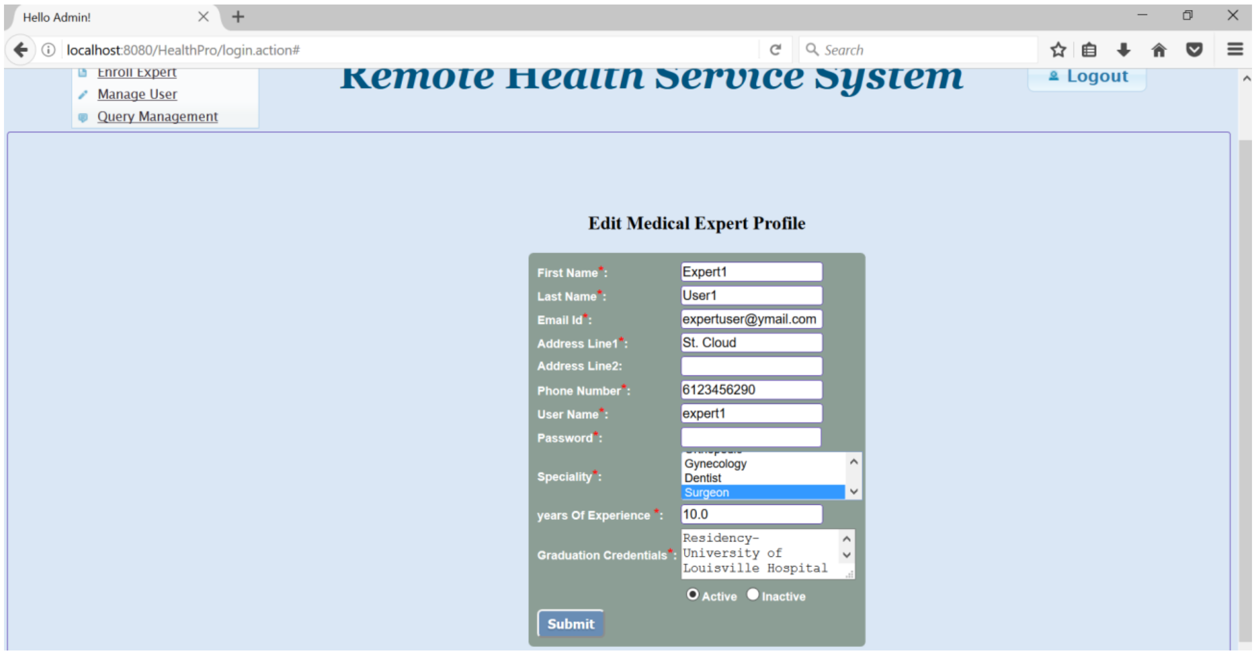


Registration Form :

FirstName*:	Expert1
LastName*:	User1
Email*:	expertuser@ymail.com
AddressLine1*:	St. Cloud
AddressLine2:	
Phone Number*:	6123456290
UserName*:	expert1
Password*:	•••••••
Speciality*:	Dentist / Surgeon / General Physician
years Of Experience*:	10
Graduation Credentials*:	Louisville Hospital Education- Medical College of Wisconsin

Figure 76: Manage Medical Experts (Add a new Medical Expert) – Admin Screen

Manage Medical Experts (Edit a Medical Expert) – Admin Screen:


Hello Admin!
localhost:8080/HealthPro/login.action#
Enroll Expert
Manage User
Query Management
Remote Health Service System
Logout
Submit

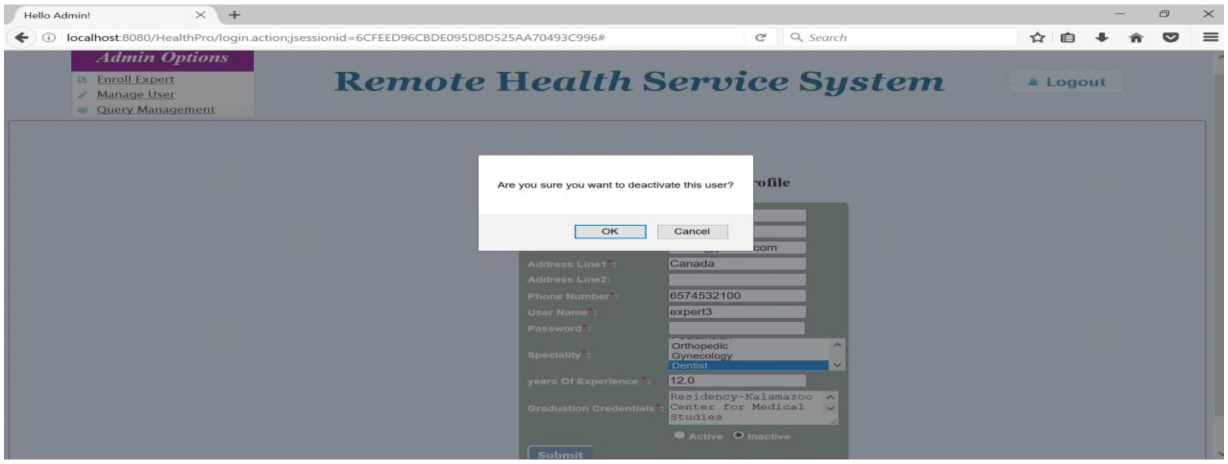
Edit Medical Expert Profile

First Name*:	Expert1
Last Name*:	User1
Email Id*:	expertuser@ymail.com
Address Line1*:	St. Cloud
Address Line2:	
Phone Number*:	6123456290
User Name*:	expert1
Password*:	
Speciality*:	Gynecology, Dentist, Surgeon (selected)
years Of Experience*:	10.0
Graduation Credentials*:	Residency-University of Louisville Hospital
	[✓ Active] Inactive

Figure 77: Manage Medical Experts (Edit a Medical Expert) – Admin Screen

Manage Medical Experts (Deactivate a Medical Expert) – Admin Screen:

The Admin can deactivate a Medical Expert by selecting ‘Inactive’ option and clicking ‘Submit’.


Hello Admin!
localhost:8080/HealthPro/login.action;jsessionid=6CFEED96CBDE095D8D525AA70493C996#
Admin Options
Enroll Expert
Manage User
Query Management
Remote Health Service System
Logout
OK
Cancel
Submit

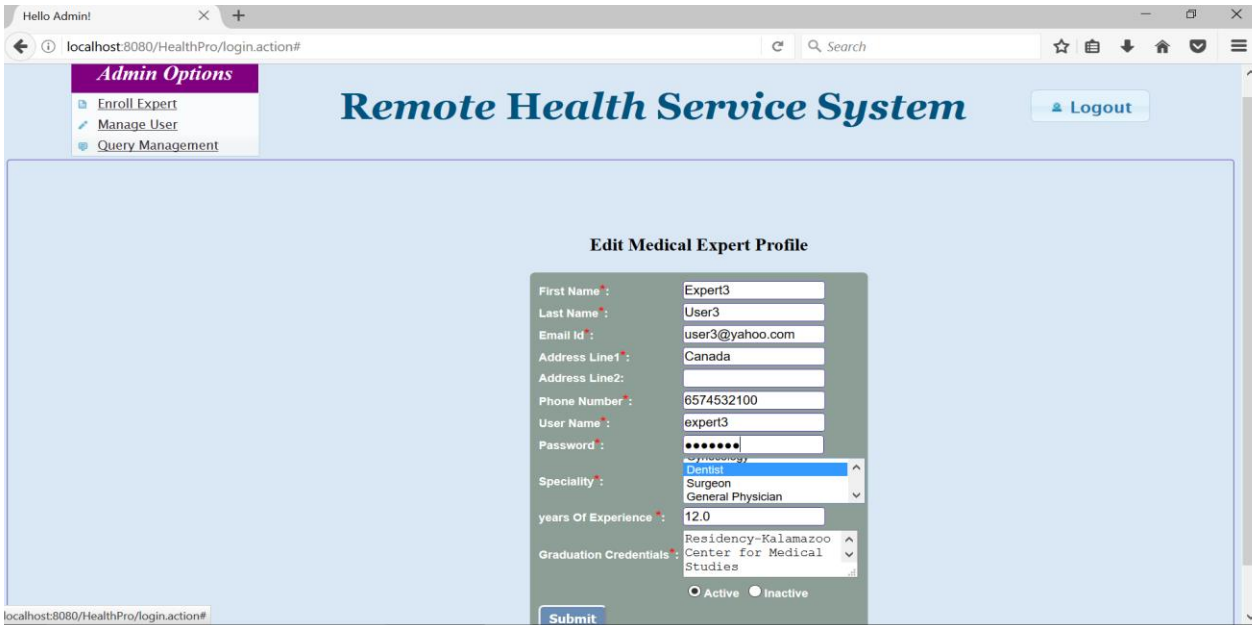
Are you sure you want to deactivate this user?

Address Line1*:	Canada
Address Line2:	
Phone Number*:	6574532100
User Name*:	expert3
Password*:	
Speciality*:	Orthopedic, Gynecology, Dentist (selected)
years Of Experience*:	12.0
Graduation Credentials*:	Residency-Kalamazoo Center for Medical Studies
	Active [✓ Inactive]

Figure 78: Manage Medical Experts (Deactivate a Medical Expert) – Admin Screen

Manage Medical Experts (Activate a Medical Expert) – Admin Screen:

The Admin can activate a Medical Expert by selecting 'Active' option and clicking 'Submit'.



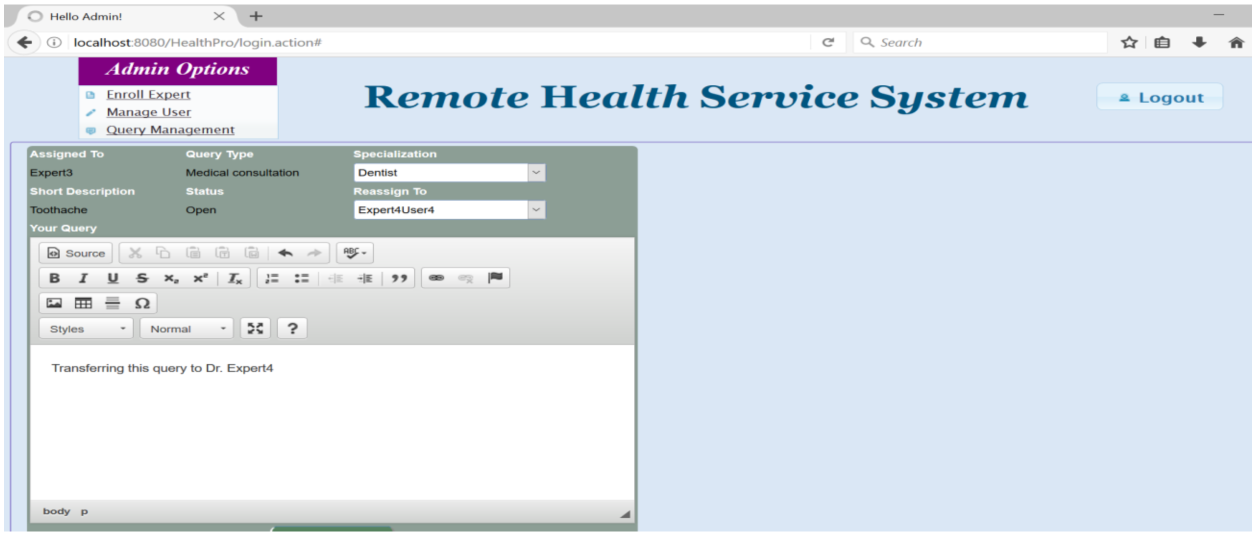
Edit Medical Expert Profile

First Name*:	Expert3
Last Name*:	User3
Email Id*:	user3@yahoo.com
Address Line1*:	Canada
Address Line2:	
Phone Number*:	6574532100
User Name*:	expert3
Password*:	•••••••
Speciality*:	Dentist; Surgeon; General Physician
years Of Experience*:	12.0
Graduation Credentials*:	Residency-Kalamazoo Center for Medical Studies
	[✓ Active] Inactive

Submit

Figure 79: Manage Medical Experts (Activate a Medical Expert) – Admin Screen

Transfer Query – Admin Screen:



Assigned To	Query Type	Specialization
Expert3	Medical consultation	Dentist
Short Description	**Status**	**Reassign To**
Toothache	Open	Expert4User4

Your Query

Transferring this query to Dr. Expert4

Figure 80: Transfer Query – Admin Screen

Chapter 4: CONCLUSION

Struts2 is an open-source web application framework for creating enterprise-level Java applications. It is elegant and extensible framework based on the Model View Controller (MVC) design pattern. It can be easily integrated with other application frameworks like Spring and Hibernate to create multi-layered web applications. The Struts2 User Interface (UI) tags help the developers create appealing interfaces with minimum effort. Hibernate is an Object Relational Mapping (ORM) tool used to create relational mapping between the object oriented classes and the database tables. It generates database independent code eliminating the need for the programmer to write database specific queries. Its inbuilt caching mechanism helps to improve performance in the persistence layer of the application.

The objective of this project is to develop a Remote Health Service System by integrating Struts2 and Hibernate frameworks. The application is flexible, modular and complete due to its multi-layered architecture. This system provides an interactive platform for the patients and medical experts to communicate remotely. The patients can send queries to medical experts regarding their health symptoms and the medical experts can respond to the received queries. The main functionalities of the system are patient registration, medical query submission, query reply and past medical query search. This project would be a good reference for anyone trying to integrate Struts2 and Hibernate frameworks and build a web application using them.

The known constraints of the Remote Health Service System are not so appealing layout in the Internet Explorer browser and the limitation of having only one Admin user. This system can be extended by implementing additional features such as email notifications for patients when a response for their query has been received, mechanism to alert the medical experts about the pending queries to ensure proper tracking of open queries and option to upload medical reports like lab reports and X-ray images while submitting a medical query. Encrypting passwords while saving to the database and option for the medical expert to view the history of past medical records for a selected patient would be other useful enhancements.

References

[1] Huo, Yanming, et al. "Design of Java EE-Based Remote Health Service System." *Intelligent Human-Machine Systems and Cybernetics (IHMSC), 2014 Sixth International Conference on*. Vol. 1. IEEE, 2014.

[2] "Struts2 Architecture – Vinaytechs Blog", http://vinaytechs.blogspot.com/2009/12/developing-struts2-application-step-by.html

[3] "Struts 2 Architecture and Fundamentals – Anitech Blog", http://www.anitechcs.com/struts-2-architecture-and-fundamentals/

[4] "Struts 2 Tutorial – Javatpoint", http://www.javatpoint.com/struts-2-tutorial

[5] "Introduction to Struts 2 framework – CodeJava", http://www.codejava.net/frameworks/struts/introduction-to-struts-2-framework

[6] "Introduction to hibernate framework – java2blog", http://www.java2blog.com/2013/01/introduction-to-hibernate-framework.html

[7] "Hibernate Tutorial – TutorialsPoint", https://www.tutorialspoint.com/hibernate/

[8] "Hibernate Architecture – Javatpoint", http://www.javatpoint.com/hibernate-architecture

[9] "Top 10 Advantages of Hibernate – OnlineTutorialsPoint", http://www.onlinetutorialspoint.com/hibernate/top-10-advantages-of-hibernate.html