FRIENDSDIARY PROJECT

REPORT DOCUMENTATION

****

*Project presented by*

Razaq Popoola R00079896

*As part of the requirement for the degree of*

**Bachelor of Science Honours**

*in*

**IT Management**

*May 2015*

2015

Razaq Popoola

RAP-TECH LTD

5/31/2015

|  |
| --- |
| RAP-TECH LTD |
| FRIENDSDIARY PROJECT |
| REPORT DOCUMENTATION |
| Razaq Popoola R00079896 COM3 |
|  |
| **5/31/2015** |

|  |
| --- |
| **To build and document a Software Project of choice.** |

**Table of Contents**

1. INTRODUCTION ………………………………………………………………………. 3

## 

## Project Background ………………………………………………………3 – 4

## Objective of the System ………………………………………………....4 – 5

**Stakeholders……………………………………………..5 – 5**

1. METHODOLOGY ……………………………………………………………………….6

## Revision Control Software ………………………………………………6 – 8

## Source Tree…………………………………………………………………9 – 9

## Lifecycle…………………………………………………………………….10 – 10

1. ANALYSIS AND DESIGN …………………………………………………………..11

## Problem Definition………………………………………………………...11 - 12

## Product Boundary Use Case Diagram…………………………………12 - 13

**Product Use Case List ………………………………….14 – 14**

**Use-Case Specification & Use Case Narrative ………..15 – 60**

**Summary of Prototyping……………………………….61 – 62**

**Data Dictionary …………………………………….......63 – 64**

**Collaboration Diagrams ……………………………….65 – 71**

**Sequence Diagrams ……………………………………72 – 81**

**Class Diagrams …………………………………………82 – 82**

**Activity Diagram………………………………………..83 – 95**

## Database Model…………………………………………………………….96 - 96

1. IMPLEMENTATION ………………………………………………………………....97

## User Manual …………………………………………………………….97 – 118

## Testing…………………………………………………………………….119 – 134

1. EVALUATION ………………………………………………………………………..135

## Revised System Objectives …………………………………………135 – 135

## Problems Encountered …..…………………………………………..136 – 136

## OutLook………………………………………………………………….137 – 137

## Division of work within the Project Group ………………………..138 - 140

INTRODUCTION

Project Background

Friends-Dairy project is the effort of Razaq Popoola a final year student in IT Management, the project is a result of research undertake by the Rap Tech Ltd and they discovered there are was an opening in the market for a social networking application that will enable the Member to be private and be secretive at the same time enable to social networking with close family and friends when desired to do so. This application will be accessible by all stakeholder and also encourage the dying long-age habit of keeping personal diary of daily event that happing’s.

FriendsDiary application will transform the Paper-based Diary keeping system to a Desktop Application System and enables simultaneous multi-user access to a centralized database. More also the issue of security threat currently paramount with social networking will become the features of the pass, 99% of the people not using the social media are doing so due to security threat and also lack of ownership of their personal data. FriendsDiary will eliminated the security issues currently associated with the top twenty social media networking where Member are not the real owners of their personal details on the applications databases and more also the current top social media lay emphasis on Member been public though Member can take advantage of private settings.

Majority of the Members of top social networking web applications are not aware of the private setting and more also the privacy regulation are frequently changed in other to prevent or discourages Members from remains private. Due to the massive economic profit generating from selling the Member’s data and also revenue generating from company’s advertising on the social networking application depends largely on how many member are visible.

Therefore in other to improve digital keeping of diary, entreat security in social media and absolute authority personal data, Rap-Tech Ltd which is owned by Razaq Popoola have decided to develop an online diary a web based application known as Friends-Diary.

Rap-Tech Ltd is the name of the business chosen, Rap-Tech is a start-up technology company located in the Rubicon Centre CIT Campus Bishoptown. Rubicon centre is leading start-up business hub in Ireland presently housing more than hundred and fifty start-up businesses at various stages of development. Rap-Tech is taken advantage of gap in the market of social media that is highly secure and private and Furth-more encouraging the use of diary keeping among the youth and the older generation.

***Relevant Facts and Assumptions***

Keeping Diary used to be a famous habit and addiction perpetuated by both old and young generation but the introduction of the World Wide Web in the late eighty and early ninety make keeping of diary a neutral death due mostly to difficulty in securing and preservation of the diary.

More than 90% the population of the people currently not a Member of any social media is due to lack of privacy and securities of their data and personal information on the social media application.

Moreover, research indicates 83% growth in the population of diaries keeper if the application is private and secure, and furthermore when there is absolute authority over personal data in the application database.

Objective of the System

**Purpose:**

The system will enable the Member to keep daily diary and other personal information into the Friends-Diary application and reduce the dependency of a Paper-based diary keeping System.

**Advantage:**

This process will reduce the time spent by member in filling out daily diary on a paper pages and Member Information that is driven by the Paper-based System.

**Measure:**

The Friends-Diary application will keep a record of all their Members at any given period.

**Purpose:**

The Friends-Diary application provides the capability for a Member to update data themselves in real time.

**Advantage:**

The problem of the present social media system whereby member information are not private and Members do not have absolute authority on their personal information in the application database will be eliminated.

**Measure:**

The Member daily diary keeping and other personal data will be available remotely through intranet as an online web application; Member personal account directory will be driven by the Friend-Diary Application database and always up-to-date.

**Purpose:**

To enable Members keep track of their personal data and socialised with friends and Family remotely.

**Advantage:**

This may result in the business being acknowledged (i.e. being more recognised globally) as Members can connect and share their personal data using the Friends-Diary web based application from anywhere in the world.

**Measure:**

This depends on the number of members on the system. It means the business must ensure that the application is relatively popular by having the features that is not presently available to the current social media application.

**Purpose:**

To provide a world ready product that can be adapted to various locales, by implementing the interactive features of social media and also private setting in the web application.

**Advantage:**

Members get more interested when they can interact with friends and family in an application, Friends-Diary is a community and there is some sense of competition, whereby Members can compare diary and other digital information and can post their thought for Friends and Family comments on social media.

**Measure:**

This depends on the number of Members using Friends-Diary application.

**Purpose:**

To make Friends-Diary web application one of the top ten social media web application available in the market, this in return will increase the business revenue.

**Advantage:**

It is predicted that with the help of the application, it will allow Members to be more private and interested in daily keeping of diary.

***The Stakeholders***

1.1.   The Client Rap-Tech Ltd

1.2.   The Customer

1.2.1    CEO

1.2.2    Administrator

1.2.3    Member

1.3.   Other Stakeholders

1.3.1     The General public

1.3.2     The Young Generation

1.3.4     The Administrator

METHODOLOGY

In order to produce a world ready application, we used a combination of iterative and prototype-based development methodologies during the development of Health Management System.

The tools that enabled the completion of the process of the project are as follows:

**Revision Control Software**

The revision control software also known as the source control software is a management used to help us build better software and also provided a great opportunity to work together efficiently. GitHub is an instance of a hosting service that uses the Git revision control system. GitHub is a powerful mechanism tool that provided numerous features, which made teamwork and code management effortless. Moreover, it is a web-based hosting service for software projects.

The basic unit of work in GitHub is the commit. A commit is a code change with an accompanying comment by its author. Whenever a change takes place, the details are immediately published on the GitHub website: who made the changes, when it was made, what lines were changed, and many more. And a comprehensive history maintained with all the details of the changes made in an easily accessible way.

An example of this is shown below:

**Life cycle**

During the development of our application, we took an iterative/prototype development based approach. One of the greatest aspects of using this model is that, everyone can see the development of the software on a step by step scale. The whole process is iterative and made it easy to demonstrate things even though some materials were temporary, it was easier and easier to change and upgrade.

However, both incremental and iterative model are main procedures in the chosen methodology, which was used in the development cycle of our Health Management System.

Underlying features required were developed in separate branches, tested, and then merged when it is suitable to be integrated into the main branch.

To sum up what we did, it most closely relates to the iterative and incremental model.



Also, it allowed for the addition of new features to be added quickly and easily.

For a specific instance of this process above, we will consider the primary GUI components: the tabs that are used to manipulate information in the database.

We started with the requirements laid out in our report. It provided designs for the appearance and general functionality of each tab. The functional requirements determined what required to be done at a finer level of detail.

We created branches for each tab and began to implement them. Multiple tabs were developed simultaneously in different branches. As each tab was completed and tested in isolation, the related branch was merged into the master branch and integration testing was carried out. Seeing that there were commonalities in our work, we created various classes to reduce redundancy and simplify development on each of the iteration. As a result of iterative prototyping and incremental improvement, there is practically no code redundancy in the program. The design is quite elegant, featuring many reusable components.

A complete history of all development activity is visible on our Git repository on GitHub, line by line.

ANALYSIS AND DESIGN

**Problem Definition**

Friends-Diary application is an online web based application which will enable the Member to keep daily diary of personal happening and also connects with friends and Family in a social media application manner in a well secured environment.

In addition, the new system will work in parallel with the existing paper-based system, which will make it possible to track member’s data stored in paper-based system. See below.

**EXISTING** **NEW**

PAPER-BASED COMPUTER-BASED

John Murphy John Murphy

To facilitate this several features must be provided:

*Duty Manager* must be able to log in and enable the creation, updating and deleting of Instructors. The Duty Manager creates classes and assigns an Instructor to the activity class. The Duty Manager also maintains reports regarding any information. The Duty Manager is responsible for planning upcoming events. This person may also assign any staff to any role. The Duty Manager is the only one with complete access to all information in the health system.

*Instructors* register members into the system, assigning members as needed into an activity class. Instructors may update member’s health information.

*Members* must be able to log into the system to view their personal information. They can update their profile information as well as leave personal notes. Members can make payment for an activity class.

*Activity Classes* are basic fitness class that members sign up. An Instructor manages an activity Class.

**Feature Overview**

**Product Boundary Use Case Diagram**

***Use Case Diagram for Health Management System***

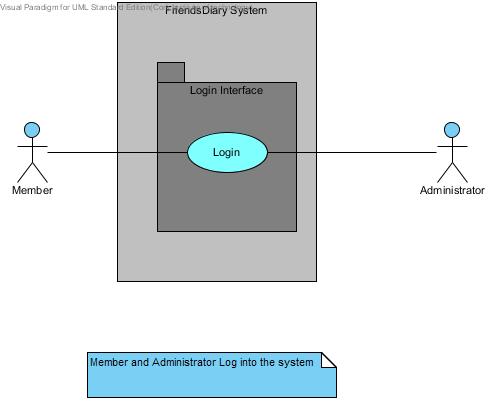
**Product Use Case List**

|  |  |
| --- | --- |
| **Use-Case Name** | **Brief Summary Of Each Use-Case** |
| **Log into System** | This use case describes the login process. |
| **Update Personal Information** | The use case describes the event of updating the Member, Instructor and Duty Manager profile information. |
| **View Health Performance** | The use case describes the event of viewing Members health performance. |
| **View Members Personal Information** | This describes the event of viewing Members complete profile information. |
| **Make a Payment** | This use case describes the event of making a payment. |
| **Leave Personal Notes** | This describes the process of leaving personal drafts, appointment dates, emergency contacts, etc. |
| **Update Health Information** | This use case describes the event of updating the Members health information. |
| **Register Members** | This use case describes the process of registering Members into the system. |
| **Manage Members** | This use case describes the event of adding, and/or deleting Members from an activity class. |
| **View Class Members** | This use case describes the event of viewing Members in any activity class. |
| **Manage Instructors** | This use case describes the event of creating, reading, updating and/or deleting Instructors from the system. |
| **Create Classes** | This use case describes the event of creating an activity class and assigning an Instructor to the class. |
| **Maintain Reports** | This describes the event of recording and maintaining any information such as member’s personal information, weight, etc. Also, it includes adding and deleting reports. |

**Use Case Narrative (Log into System)**

Full Description of Use case **Log into System**

|  |  |  |
| --- | --- | --- |
| Use Case Name: | Log into System | |
| Scenario: | Actors want to log into the Friends-Diary application with their Login details (e.g. username and password). | |
| Event (Trigger): | Actors want to enter into the Friends-Diary application to keep diary, manage their profile or the application and socialised with their friends online. | |
| Brief Description: | User provides required credentials. System validates credentials and logs user into the Health Management System. | |
| Actors: | Member, Administrator. | |
| Type: | Essential | |
| Preconditions: | Member and Administrator are known by the System | |
| Post conditions: | Member and Administrator logged into the system. | |
| Flow of Events  (Steps): | Actor Action | System Response |
| 1. This use case begins when a user desires to log into the system. |  |
| 1. The user provides the username and password. | * 1. System validates the entered username and password and logs the user into the system. |
| Alternative Flow of Events | | |
| Line 2.1: Invalid username and password entered. Indicate error. Return to Step 2. | | |



**Use Case Narrative (Update Personal Information)**

Full Description of Use case **Update Profile**

|  |  |  |
| --- | --- | --- |
| **Use Case Name:** | Update Profile | |
| **Scenario:** | Members want to update his/her personal details in the Friends-Diary application. | |
| **Event (Trigger):** | Update personal information on the web application. | |
| **Brief Description:** | A member provides information (e.g. name, password, address, telephone number). And the application updates the personal information. | |
| **Actors:** | Members Administrator. | |
| **Type:** | Essential | |
| **Preconditions:** | Member and Administrator are known by the System  Member and Administrator Information must exist. | |
| **Post conditions:** | Member and Administrator Information updated in the system. | |
| **Flow of Events**  **(Steps):** | **Actor Action** | **System Response** |
| 1. This use case begins when a user desires to update their personal information in the web application. |  |
| 1. The user provides the username and password. | 2.1 System validates username and password and logs the Member into the system.  2.2 System displays Member’s personal information. |
| 1. The Member modifies their personal information. (E.g. address, name, telephone number). | 3.1 The application updates and displays user’s personal information. |
| **Alternative Flow of Events** | | |
| **Line 2.1:** Invalid username and password entered. Indicate error. Return to Step 2. | | |

**Use Case Narrative (Search for Friend)**

Full Description of Use case **Search for Friend**

|  |  |  |
| --- | --- | --- |
| **Use Case Name:** | View Health Performance | |
| **Scenario:** | The process whereby a member wants to view his/her health performance. | |
| **Event (Trigger):** | Member wants to view health performance. | |
| **Brief Description:** | Member requests to view health performance. System displays member’s health performance. | |
| **Actors:** | Member | |
| **Type:** | Essential | |
| **Preconditions:** | Member is known by the system.  Health Information must exist and be up to date. | |
| **Post conditions:** | None | |
| **Flow of Events**  **(Steps):** | **Actor Action** | **System Response** |
| 1. This use case begins when a Member wants to view health performance in the system. |  |
| 1. The Member provides username and password. | * 1. System validates username and password and logs the Member into the system.   2. System displays member’s health performance table. |
| 1. Member views the health performance table and analyzes their weight performance. |  |
| **Alternative Flow of Events** | | |
| **Line 2.1:** Invalid username and password entered. Indicate error. Return to Step 2.  Couldn’t find Member. Inform the Member. Return to Step 2.  **Line 2.2:** Health Performance not up-to-date. Inform the Member. Cancel the process. | | |

**Use Case Narrative (View Members Profile)**

Full Description of Use case **View Members Profile**

|  |  |  |
| --- | --- | --- |
| **Use Case Name:** | View Members Personal Information | |
| **Scenario:** | Member wants to view his/her personal information. | |
| **Event (Trigger):** | View personal information | |
| **Brief Description:** | Member provides login details. System displays member’s personal information. | |
| **Actors:** | Member | |
| **Type:** | Essential | |
| **Preconditions:** | Member is known by the system.  Personal Information must exist and be up to date. | |
| **Post conditions:** | None | |
| **Flow of Events**  **(Steps):** | **Actor Action** | **System Response** |
| 1. This use case begins when a Member wants to view their personal information in the system. |  |
| 1. The Member provides username and password. | * 1. System validates username and password and logs the Member into the system.   2. System displays Members Personal Information. |
| 1. Member views their Personal Information. |  |
| **Alternative Flow of Events** | | |
| **Line 2.1:** Invalid username and password entered. Indicate error. Return to Step 2.  Member not registered. Inform the Member. Return to Step 2. | | |

**Use Case Narrative (Manage Friend)**

Full Description of Use case **Make a Payment**

|  |  |  |
| --- | --- | --- |
| **Use Case Name:** | Make a Payment | |
| **Scenario:** | A Member wants to make a payment. | |
| **Event (Trigger):** | New payment. | |
| **Brief Description:** | Member provides information on payment details. System verifies payment. On completion, the automated system displays payment details. | |
| **Actors:** | Member | |
| **Type:** | Essential | |
| **Preconditions:** | Member is known by the system.  The payment method must exit and be valid.  Member is registered for an activity class. | |
| **Post conditions:** | New payment created and recorded in the system.  Member account updated. | |
| **Flow of Events**  **(Steps):** | **Actor Action** | **System Response** |
| 1. This use case begins when a Member wants to make a payment for an activity class. |  |
| 1. The Member provides username and password. | * 1. System validates username and password and logs the Member into the system. |
| 1. The Member provides name on card, card number, expire date, card verification value (CVV), type of card. | * 1. System validates card details and displays a confirmation screen indicating payment made. |
| 1. Member views payment details. |  |
| **Alternative Flow of Events** | | |
| **Line 2.1:** Invalid username and password entered. Indicate error. Return to Step 2.  **Line 3.1:** Invalid card number, CVV, expire date. Indicate error. Return to Step 3. | | |

**Use Case Narrative (Mind outpouring)**

Full Description of Use case **Mind Outpouring**

|  |  |  |
| --- | --- | --- |
| **Use Case Name:** | Leave Personal Notes | |
| **Scenario:** | A Member wants to leave personal note in the system. | |
| **Event (Trigger):** | Leave personal notes. | |
| **Brief Description:** | Member provides information on what to leave in the system such as drafts, appointment dates, emergency contacts, etc. | |
| **Actors:** | Member | |
| **Type:** | Essential | |
| **Preconditions:** | Member is known by the system. | |
| **Post conditions:** | Personal notes created in the system. | |
| **Flow of Events**  **(Steps):** | **Actor Action** | **System Response** |
| 1. This use case begins when a Member wants to leave personal notes. |  |
| 1. The Member provides username and password. | * 1. System validates username and password and logs the Member into the system. |
| 1. The Member provides the ID, note, drafts, title of the note, date created, etc. | * 1. Adds notes to the system. |
| **Alternative Flow of Events** | | |
| **Line 2.1:** Invalid username and password entered. Indicate error. Return to Step 2. | | |

**Use Case Narrative (Manage Diary)**

Full Description of Use case **Manage Diary**

|  |  |  |
| --- | --- | --- |
| **Use Case Name:** | Update Health Information | |
| **Scenario:** | An Instructor wants to update the member’s health information. | |
| **Event (Trigger):** | New health information. | |
| **Brief Description:** | Instructor provides health information. System updates member’s health information. | |
| **Actors:** | Instructor | |
| **Type:** | Essential | |
| **Preconditions:** | Instructor is known by the system.  Member is known by the system.  Health Information must exit. | |
| **Post conditions:** | Member’s health information updated. | |
| **Flow of Events**  **(Steps):** | **Actor Action** | **System Response** |
| 1. This use case begins when an Instructor updates member’s health information. |  |
| 1. The Instructor provides his/her username and password. | * 1. System validates username and password and logs the Instructor into the system. |
| 1. The Instructor makes the desired changes to the member’s health information. Such as the new target weight, blood pressure, BMR, RMR, BMI, push-up test/min, sit-up test/min, height, and body composition, for each Member. | * 1. Adds information to the member’s health performance record. |
| 1. On completion of entry of the health information, the Instructor indicates that the entry of health information is complete. | * 1. The System updates the member’s health performance table. |
| **Alternative Flow of Events** | | |
| **Line 2.1:** Invalid username and password entered. Indicate error. Return to Step 2.  **Line 3.1:** Couldn’t find Member. Inform the Instructor. Return to Step 3. | | |

**Use Case Narrative (Register Members)**

Full Description of Use case **Register Members**

|  |  |  |
| --- | --- | --- |
| **Use Case Name:** | Register Members | |
| **Scenario:** | An Instructor registers members into the system. | |
| **Event (Trigger):** | New Member | |
| **Brief Description:** | The Instructor provides member’s details. System adds member to the system. | |
| **Actors:** | Instructor | |
| **Type:** | Essential | |
| **Preconditions:** | Instructor is known by the system.  Member details must exit. | |
| **Post conditions:** | Member is registered into the system. | |
| **Flow of Events**  **(Steps):** | **Actor Action** | **System Response** |
| 1. This use case begins when an Instructor wants to register a member into the Health Management System. |  |
| 1. The Instructor provides his/her username and password. | * 1. System validates username and password and logs the Instructor into the system. |
| 1. The Instructor enters the member’s information. This includes: firstname, lastname, gender, address, telephone number, weight, height, BMI, date of birth (DOB), email address. | * 1. Validates Member’s Information. |
| 1. The Instructor indicates that the entry of member’s information is complete. | * 1. The system generates and assigns a unique username and password to the member.   2. The member is added to the system.   3. Then, an automated email is sent to the member containing |
| **Alternative Flow of Events** | | |
| **Line 2.1:** Invalid username and password entered. Indicate error. Return to Step 2.  **Line 3.1:** Invalid Information provided. Inform the Instructor. Return to Step 3. | | |

**Use Case Narrative (View Class Members)**

Full Description of Use case **View Class Members**

|  |  |  |
| --- | --- | --- |
| **Use Case Name:** | View Class Members | |
| **Scenario:** | An Instructor wants to view members in an activity class. | |
| **Event (Trigger):** | Viewing class member | |
| **Brief Description:** | The Instructor logs into the system and request to view class members. The system displays activity class members for the Instructor. | |
| **Actors:** | Instructor | |
| **Type:** | Essential | |
| **Preconditions:** | The Instructor must have previously logged onto the system. | |
| **Post conditions:** | None | |
| **Flow of Events**  **(Steps):** | **Actor Action** | **System Response** |
| 1. This use case begins when an Instructor wants to view class members. |  |
| 1. The Instructor provides his/her username and password. | * 1. System validates username and password and logs the Instructor into the system. |
| 1. The Instructor clicks on details link for an activity class. | * 1. The system displays all the members in the activity class. |
| **Alternative Flow of Events** | | |
| **Line 2.1:** Invalid username and password entered. Indicate error. Return to Step 2. | | |

**Use Case Narrative (Manage Members)**

Full Description of Use case **Manage Members**

|  |  |  |
| --- | --- | --- |
| **Use Case Name:** | Manage Members | |
| **Scenario:** | An Instructor wants to add, change, and/or delete Members from an activity class. | |
| **Event (Trigger):** | This use case is initiated when an Instructor wants to add Members to an activity class. | |
| **Brief Description:** | The Instructor logs into the system. The system displays activity class. Then, the Instructor adds the member to an activity class. | |
| **Actors:** | Instructor | |
| **Type:** | Essential | |
| **Preconditions:** | The Instructor must have previously logged onto the system.  Member is known by the system  Activity Class must exit | |
| **Post conditions:** | Member added to an Activity Class. | |
| **Flow of Events**  **(Steps):** | **Actor Action** | **System Response** |
| 1. This use case begins when an Instructor wants to add a member to an activity class. |  |
| 1. The Instructor provides his/her username and password. | * 1. System validates username and password and logs the Instructor into the system. |
| 1. The Instructor selects members to be enrolled into an activity class. | * 1. The system adds members to an activity class. |
| **Alternative Flow of Events** | | |
| **Line 2.1:** Invalid username and password entered. Indicate error. Return to Step 2.  **Line 3.1:** Couldn’t find Activity Class. Inform Instructor. Return to Step 3. | | |

**Use Case Narrative (Manage Instructors)**

Full Description of Use case **Manage Instructors**

|  |  |  |
| --- | --- | --- |
| **Use Case Name:** | Manage Instructors | |
| **Scenario:** | This use case is initiated when a Duty Manager wants to add Instructors to the system. | |
| **Event (Trigger):** | New Instructor | |
| **Brief Description:** | This use case describes the event of creating, reading, updating and/or deleting Instructors from the system. | |
| **Actors:** | Duty Manager | |
| **Type:** | Essential | |
| **Preconditions:** | The Duty Manager must have previously logged onto the system. | |
| **Post conditions:** | Instructor added to the system. | |
| **Flow of Events**  **(Steps):** | **Actor Action** | **System Response** |
| 1. This use case begins when a Duty Manager wants to add an Instructor into the system |  |
| 1. The Duty Manager provides his/her username and password. | * 1. System validates username and password and logs the Duty Manager into the system. |
| 1. The Duty Manager specifies the function to be carried out. | * 1. The system displays the function for the Duty Manager. |
| 1. Duty Manager enters Instructors information such as name, ID, email address, phone number, gender, DOB, etc. | * 1. The system validates the information and adds the Instructor into the system. |
| **Alternative Flow of Events** | | |
| **Line2.1:** Invalid username and password entered by the Duty Manager. Indicate error. Return to Step 2.  **Line4.1:** Invalid Instructor details or Insufficient Information. Inform Duty Manager. Return to Step 4. | | |

**Use Case Narrative (Create Classes)**

Full Description of Use case **Create Classes**

|  |  |  |
| --- | --- | --- |
| **Use Case Name:** | Create Classes | |
| **Scenario:** | This use case is initiated when a Duty Manager wants to create Activity Classes. | |
| **Event (Trigger):** | New Classes | |
| **Brief Description:** | This use case describes the event of creating an activity class and assigning an Instructor to the class. The Duty Manager can also update or delete activity classes if changes are made within the add activity class at the beginning of the use case. | |
| **Actors:** | Duty Manager | |
| **Type:** | Essential | |
| **Preconditions:** | The Duty Manager must have previously logged onto the system. | |
| **Post conditions:** | Activity Classes added to the system and an Instructor assigned to the Activity Class. | |
| **Flow of Events**  **(Steps):** | **Actor Action** | **System Response** |
| 1. This use case begins when a Duty Manager wants to add an activity class into the system |  |
| 1. The Duty Manager provides his/her username and password. | * 1. System validates username and password and logs the Duty Manager into the system. |
| 1. The Duty Manager specifies the function to be carried out. | * 1. The system displays the function for the Duty Manager. |
| 1. Duty Manager enters activity class information such as class title, start date, end date, cost, max members, description, and class schedule. | * 1. The system validates the information and adds the activity class into the system. |
| 1. The Duty Managers selects an Instructor for the activity class. | * 1. System assigns the Instructor to the activity class. |
| **Alternative Flow of Events** | | |
| **Line2.1:** Invalid username and password entered by the Duty Manager. Indicate error. Return to Step 2.  **Line4.1:** Invalid activity class details or Insufficient Information. Inform Duty Manager. Return to Step 4. | | |

**Use Case Narrative (Maintain Reports)**

Full Description of Use case **Maintain Reports**

|  |  |  |
| --- | --- | --- |
| **Use Case Name:** | Maintain Reports | |
| **Scenario:** | Duty Manager creates Member Report. | |
| **Event (Trigger):** | New Reports | |
| **Brief Description:** | This use case allows the Duty Manager to create either a member class information, weight performance, personal information report. The Duty Manager can also delete the report from the system. | |
| **Actors:** | Duty Manager | |
| **Type:** | Essential | |
| **Preconditions:** | The Duty Manager must be logged onto the system in order for this use case to begin. | |
| **Post conditions:** | None | |
| **Flow of Events**  **(Steps):** | **Actor Action** | **System Response** |
| 1. This use case is initiated when a Duty Manager wants to maintain reports. |  |
| 1. The Duty Manager provides his/her username and password. | * 1. System validates username and password and logs the Duty Manager into the system. |
| 1. The Duty Manager specifies the report criteria: Report type (either class information, weight performance, personal information), Dates for the report, Member name(s). | * 1. The system displays the report criteria for the Duty Manager. |
| 1. Duty Manager provides the information: class, member, report title. | * 1. The system provides a report satisfying the report criteria. |
| 1. Duty Manager indicates everything complete. | * 1. The system saves the report to the specified name and location. |
|  | 1. Confirms the decision to print the report. | * 1. Prints updated Report. |
| **Alternative Flow of Events** | | |
| **Line2.1:** Invalid username and password entered by the Duty Manager. Indicate error. Return to Step 2.  **Line 3.1:** Invalid format or Insufficient Information. Inform Duty Manager. Return to Step 3.  **Line4.1** Requested Information Unavailable. Inform Duty Manager. Return to Step 4. | | |

Summary of Prototyping

**Functional Requirements**

* 1. The system shall allow the Member, Instructor and Duty Manager to log in with their username and password.
  2. The system shall enable the Member to view their Personal Information.
  3. The system shall allow the Member to update their Personal Information.
  4. The system shall enable the Member to view their Health Information.
  5. The system shall record Member Payment.
  6. The system shall allow the Member to leave Personal Notes.
  7. The system shall allow the Instructor to update the Member Health Information.
  8. The system shall allow the Instructor to register Members into the system.
  9. The system shall allow the Instructor to add Members into an Activity Class.
  10. The system shall enable the Instructor to view Class Members.
  11. The system shall allow the Duty Manager to manage Instructors.
  12. The system shall allow the Duty Manager to create Activity Classes.
  13. The system shall enable the Duty Manager to maintain Reports.
  14. The system shall enable copies of Member Information to be printed.

**Non-Functional Requirements**

1. **Operational Requirements**
2. The system shall run on desktop PCs, to be used by stakeholders.
   * 1. The system shall interface with the business management system.
        + 1. The system shall be flexible and easily adaptable to changes, saving up to 50 percent of its operating cost.
          2. The system shall be a platform and accessible from any desktops and remote areas.
3. **Performance Requirements**
4. The system shall support management staff of approximately 50 management people.
   * + - 1. The system shall enable members to update their personal information within 15 minutes.
   1. The user shall get a system response within 10 seconds.
5. The Instructor shall be able to register up to 20 members in an activity class.
6. Responses to report queries shall take less than two seconds.
7. **Security Requirements**
8. No member can access any other member’s personal information.
9. The system shall be secured to prevent unauthorized access of member data.
10. **Availability Requirements**
11. Information about health performance shall be available at any time to the members.
12. A list of class members shall be made available at any time to the fitness Instructor.

**Data Dictionary**

This data dictionary is a textual list of all concepts that are defined during the analysis. The aim of the data dictionary is to define a vocabulary that will be common to all the users of the Health Management System.

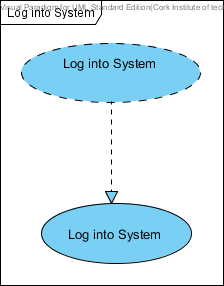
|  |  |
| --- | --- |
| **Name** | **Description** |
| **Member ID** | This is a unique automated Identification Number, which is assigned to each member when registered by the system. This will enable them to log in into the system. |
| **Username** | This is an Identification Name of the User Account Name. This can be combination of the user names and any other names or special characters the account will know. |
| **Password** | The password enforces the privacy’s of the users account. It is use by the User to log into the System. |
| **First Name** | The First name of the User’s, as known to the System |
| **Last Name** | The User’s Last Name as known to the System |
| **Address** | This is the current postal address of the Users, These are mostly the home or office address of the user and Also the address is used for payment. |
| **Phone Number** | This is the contact phone number of the user. |
| **Email Address** | This is active and current email addresses of the Users, the system used automate email to send password to users on registration. |
| **DOB** | DOB is the date of birth for the Users. |
| **Gender** | It is the sex of the user of the application. |
| **Height** | This is the height of the Member. |
| **Kgs** | This measure the unit of the weight |
| **Stone** | This measure the unit of the weight |
| **TargetWeightKgs** | This is the weight the member aimed to attain, can be update |
| **BMI**  **(Body Mass Index)** | This is how to calculate the body mass of the member. It is calculate by this formula ((weight \* 703)/(height \* height)); The gauge is used to determine whether or not a person is overweight or obese |
| **Medical History** | This is medical history of the member such as: Allergies, heart condition and general Members health status in other to determine the activities the Member can participate. |
| **Extra Information** | This is some information about the member, that can be useful |
| **Smoker** | Indication if the member is a smoker or not |
| **Personal Notes** | This is some text written by the member, which is connected to event as they happen in and around the weight clinic. It can be updated frequently. |
| **Image** | This is a picture of the User. |
| **Aerobic Exercise** | Activity involving large muscles, done for an extended period of time. Aerobic exercise can be used for weight loss. Examples of aerobic exercise include walking, biking, jogging, swimming, aerobic classes and cross country skiing. This information may be used via ‘Tips’ that are shown on the form when a member logs on. These are randomly generated from a list of tips on the server. |
| **Database Security** | Connecting directly to a database leaves the system vulnerable to hacking. We reference ‘Database Security’ as connecting to our database by first sending our information through a script and then to the database. |
| **Fitness Update** | A Moderator can submit a Fitness update for a member, usually at a weekly interval. |
| **Database** | This is where data is stored in the system. Such as members and raw data. |
| **Duty Manager** | This is a User that can create, edit and delete Instructors and Members; Duty manager manages the system generally, Duty manager is the system administrator. |
| **Member** | A user of the system that can only view the ranking system and can view their own statistics. |
| **User ID** | Special number to identify the different users of the system. |
| **Instructor** | An Instructor is a User that can create, edit or delete Members. An Instructor is the bridge between the Member, the system and the Duty manager, Instructor interact more with the Member and the system in normal situation. |
| **Log on** | An interface is where the Users must input their username and password before they can use the application, is the bridge between the system and the Users. |
| **Performance Graph** | A graph representation to measure the level of weight loss or gain. |
| **Register Screen** | When a member details are entered into the program and are then registered on the system. |
| **Obese** | This is when a person is overweight. |

**Collaboration Diagrams**

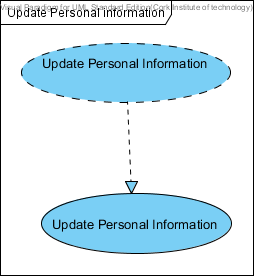
Collaboration Diagrams are used to model an alternate view of the Sequence Diagram. It shows how objects involved in a situation interact with each other, instantiating a particular class in the Health Management System. Also, it gives the modeller the ability to show detail, such as visibility. Collaboration diagrams are used to understand all of the effects on any given object and are also used for procedural design.

Collaboration diagrams are used to model every scenario found in the Health Management System.

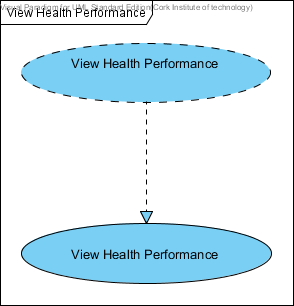
The use case diagram **Log into System** is collaborating with the Use case Log into System.



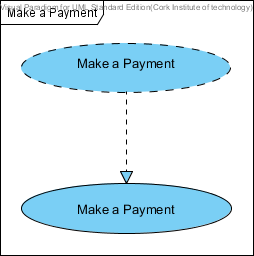
The use case diagram **Update Personal Information** is collaborating with the Use case Update Personal Information.



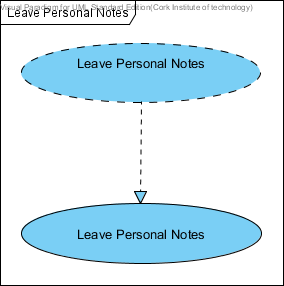
The use case diagram **View Health Performance** is collaborating with the Use case View Health Performance.



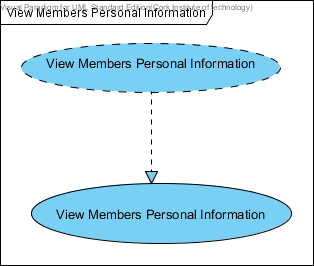
The use case diagram **Make a Payment** is collaborating with the Use case Make a Payment.



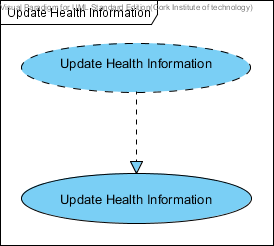
The use case diagram **Leave Personal Notes** is collaborating with the Use case Leave Personal Notes.



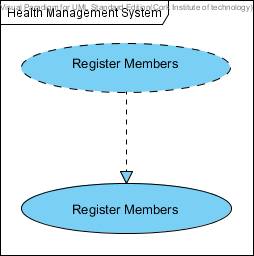
The use case diagram **View Members Personal Information** is collaborating with the Use case View Members Personal Information.



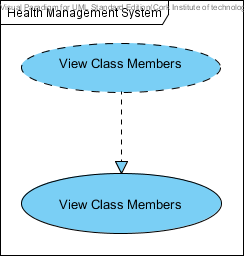
The use case diagram **Update Health Information** is collaborating with the Use case Update Health Information.



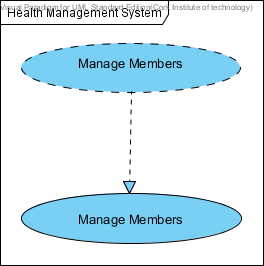
The use case diagram **Register Members** is collaborating with the Use case Register Members.



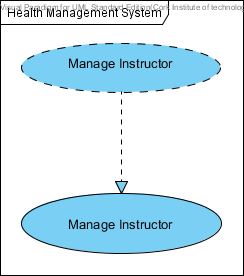
The use case diagram **View Class Members** is collaborating with the Use case View Class Members.



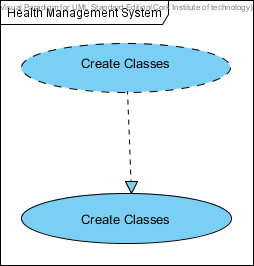
The use case diagram **Manage Members** is collaborating with the Use case Manage Members.



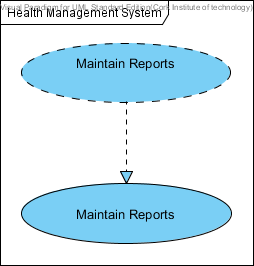
The use case diagram **Manage Instructor** is collaborating with the Use case Manage Instructor.



The use case diagram **Create Classes** is collaborating with the Use case Create Classes.



The use case diagram **Maintain Reports** is collaborating with the Use case Maintain Reports.

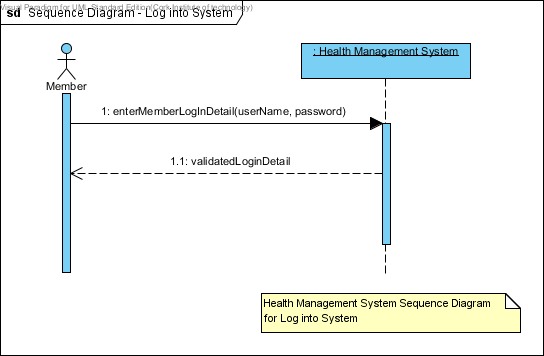


**Sequence Diagrams**

Sequence diagrams display the interaction between objects arranged in a time sequence. An interaction sequence diagram (also known as a sequence diagram) is one of many kinds of UML interaction diagram. The sequence diagram is semantically equivalent to a communication diagram for simple interactions.

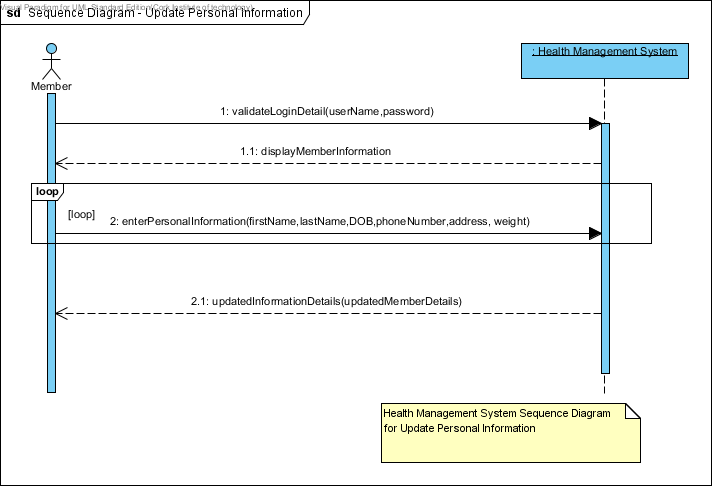
Sequence diagrams can be modelled at different levels of information and to meet different purposes at several stages in the development lifecycle. The most common use of a sequence diagram is to show the specific object interaction that occurs mainly for one use case or one application. For instance, when an interaction sequence diagram is used to design the dynamic behaviour of a use case, it can be seen as a detailed specification of the use case.

The sequence diagram below is used to represent the detailed object interaction that occurs for the use case **Log into System**. Here, it outlines how a **Member, Instructor and a Duty Manager** interact with the **Health Management System**.



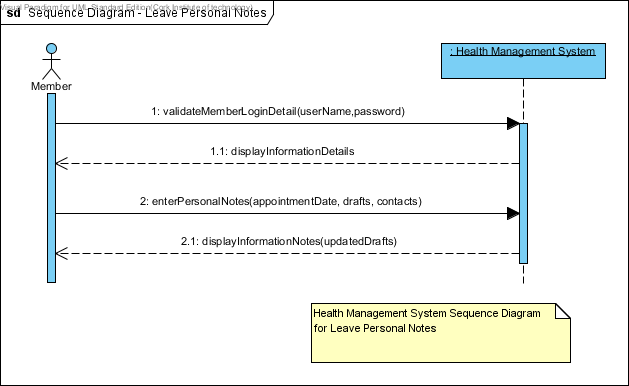
Sequence Diagram for the use case **Log into System**.

The sequence diagram below is used to represent the detailed object interaction that occurs for the use case **Update Personal Information**. Here, it outlines how a **Member, Instructor and a Duty Manager** interact with the **Health Management System**.



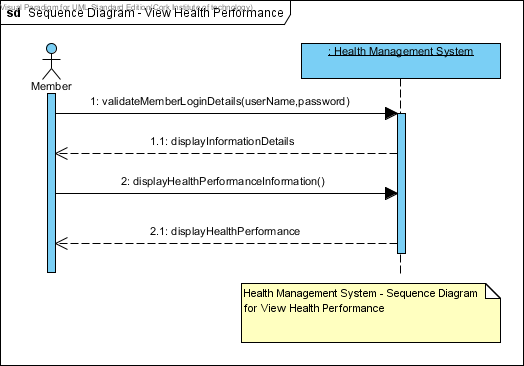
Sequence Diagram for the use case **Update Personal Information**.

The sequence diagram below is used to represent the detailed object interaction that occurs for the use case **Leave Personal Notes**. Here, it outlines how a **Member** interacts with the **Health Management System**.



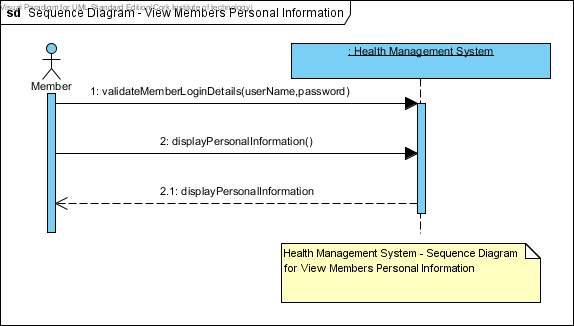
Sequence Diagram for the use case **Leave Personal Notes.**

The sequence diagram below is used to represent the detailed object interaction that occurs for the use case **View Health Performance**. Here, it outlines how a **Member** interacts with the **Health Management System**.



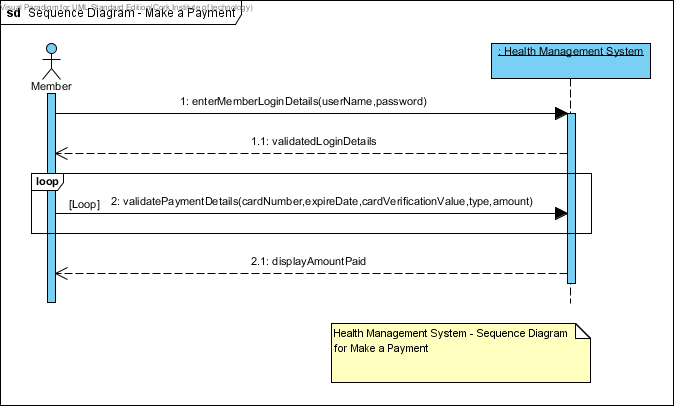
Sequence Diagram for the use case **View Health Performance.**

The sequence diagram below is used to represent the detailed object interaction that occurs for the use case **View Members Personal Information**. Here, it outlines how a **Member** interacts with the **Health Management System**.



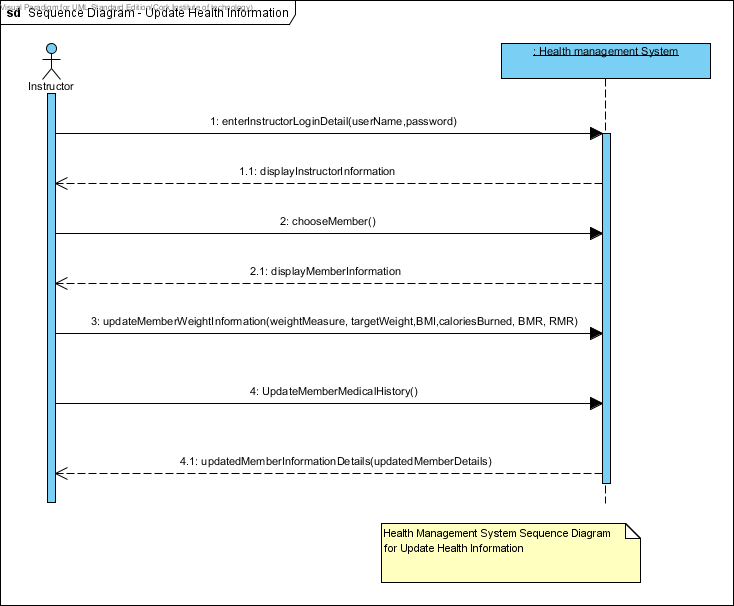
Sequence Diagram for the use case **View Members Personal Information.**

The sequence diagram below is used to represent the detailed object interaction that occurs for the use case **Make a Payment**. Here, it outlines how a **Member** interacts with the **Health Management System**.



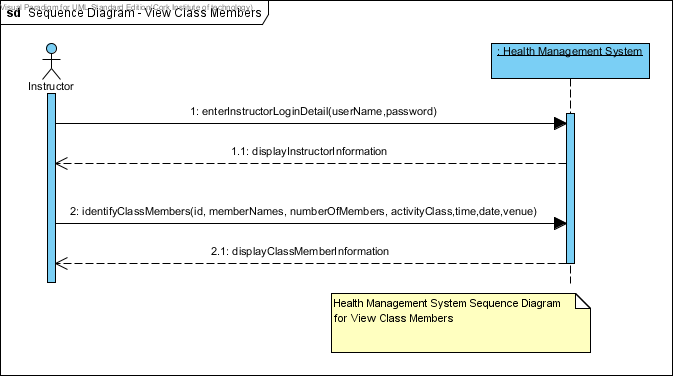
Sequence Diagram for the use case **Make a Payment**.

The sequence diagram below is used to represent the detailed object interaction that occurs for the use case **Update Health Information**. Here, it outlines how an **Instructor** interacts with the **Health Management System**.



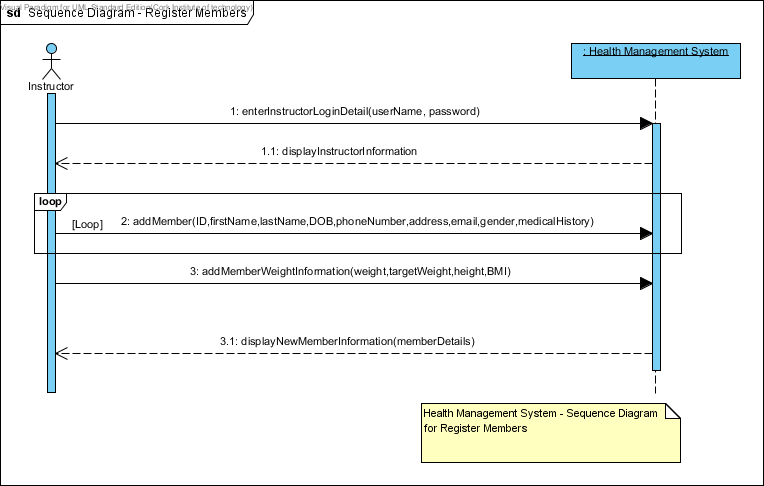
Sequence Diagram for the use case **Update Health Information.**

The sequence diagram below is used to represent the detailed object interaction that occurs for the use case **View Class Members**. Here, it outlines how an **Instructor** interacts with the **Health Management System**.



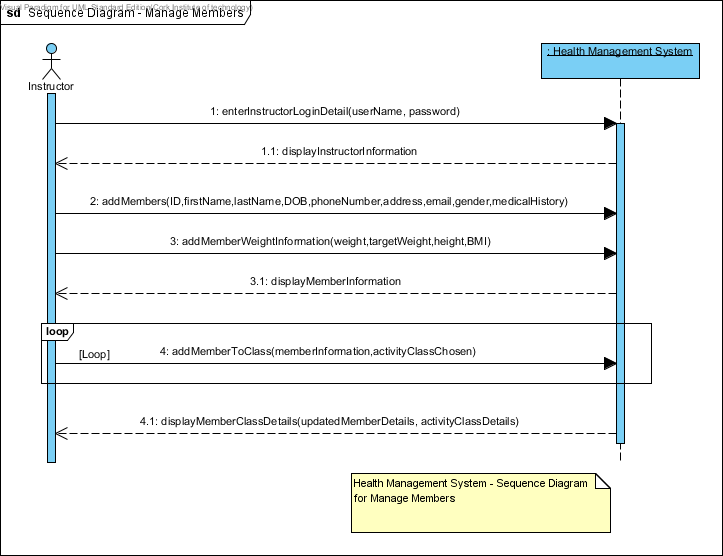
Sequence Diagram for the use case **View Class Members**.

The sequence diagram below is used to represent the detailed object interaction that occurs for the use case **Register Members**. Here, it outlines how an **Instructor** interacts with the **Health Management System**.



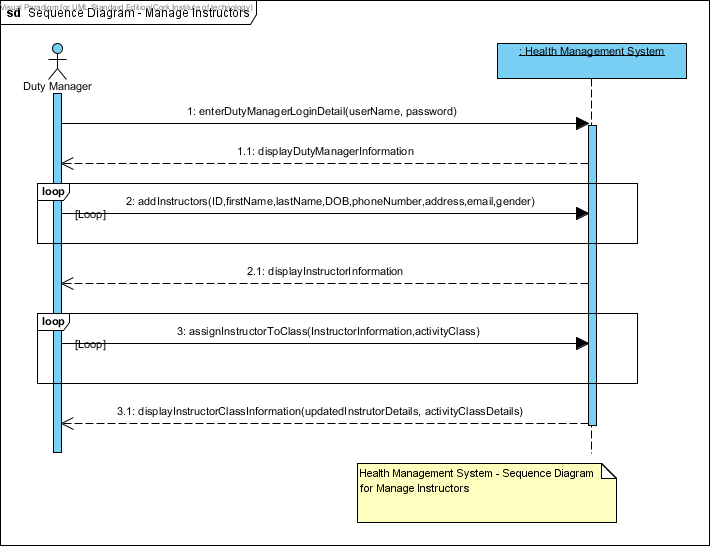
Sequence Diagram for the use case **Register Members**.

The sequence diagram below is used to represent the detailed object interaction that occurs for the use case **Manage Members**. Here, it outlines how an **Instructor** interacts with the **Health Management System**.



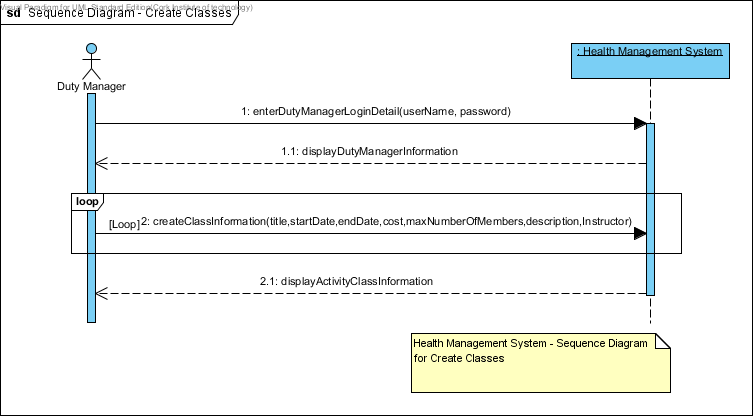
Sequence Diagram for the use case **Manage Members**.

The sequence diagram below is used to represent the detailed object interaction that occurs for the use case **Manage Instructors**. Here, it outlines how a **Duty Manager** interacts with the **Health Management System**.



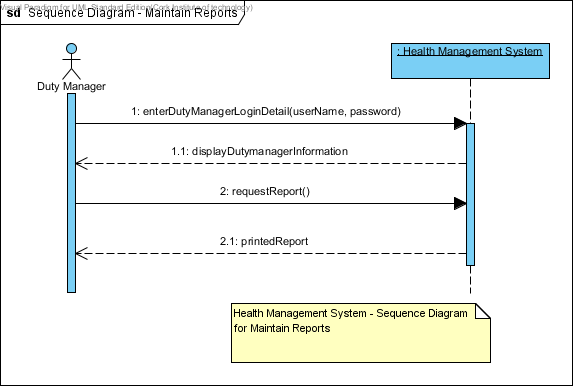
Sequence Diagram for the use case **Manage Instructors**.

The sequence diagram below is used to represent the detailed object interaction that occurs for the use case **Create Classes**. Here, it outlines how a **Duty Manager** interacts with the **Health Management System**.



Sequence Diagram for the use case **Create Classes**.

The sequence diagram below is used to represent the detailed object interaction that occurs for the use case **Maintain Reports**. Here, it outlines how a **Duty Manager** interacts with the **Health Management System**.

****

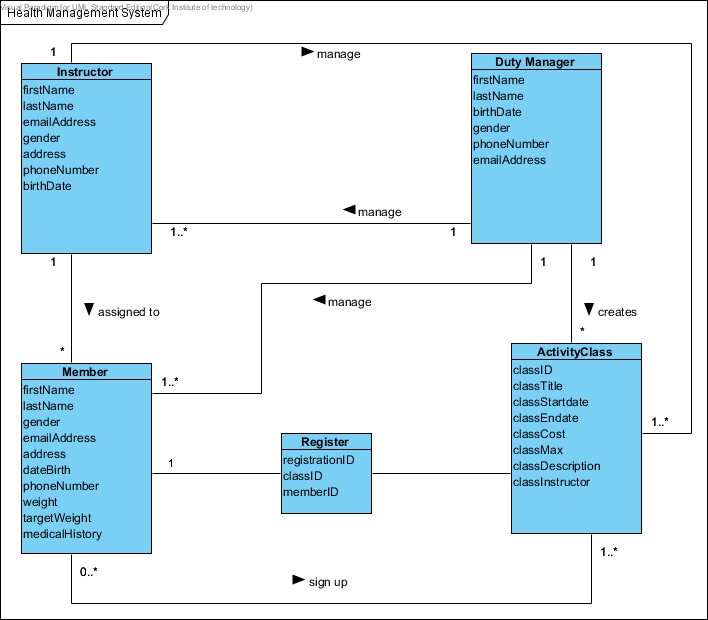
Sequence Diagram for the use case **Maintain Reports**.

**Class Diagrams**

Class Diagrams in UML (Unified Modelling Language) is a static diagram that describes the structure of the Health Management System being modelled. It shows a collection of the System’s Classes (i.e. Class Names), their Attributes, Operations, and the Relationships among Objects. Moreover, it is the main building block of object-oriented modelling. However, class diagrams can be used to illustrate data model of the Health Management System.

Class Diagrams will be used to model the general static view of the Health Management Application and detailed modelling that translate the models into programming codes.

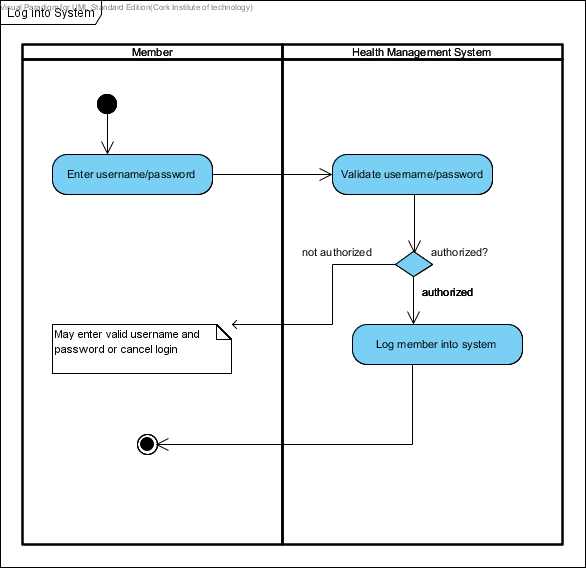
**Domain Model**



**Activity Diagrams**

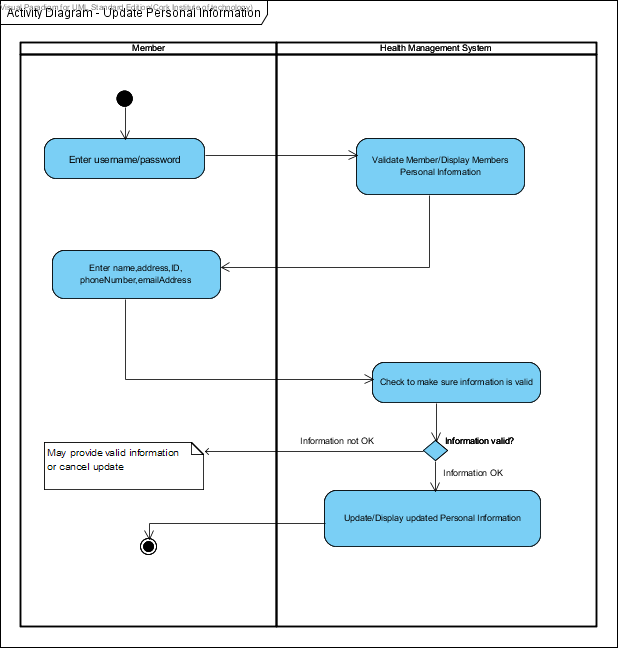
Activity diagrams are used to model different aspects of the Health Management System. They may be used to model a system function represented by a use case, possibly using object flows to show which objects that are involved in each use case. This example would be done during the phase of the lifecycle when requirements are being elaborated. They can also be used at a low level to model the detail of how a particular operation is carried out, and are likely to be for this purpose in later analysis or system design activities. However, activity diagrams at their simplest are used to show action flow from one activity to another, which can be seen in the Health Management System.

The activity diagram below is used to illustrate the system function that is represented by the use case **Log into System.** Here, it describes the activity flow from the Member, Instructor and a Duty Manager to the Health Management System.



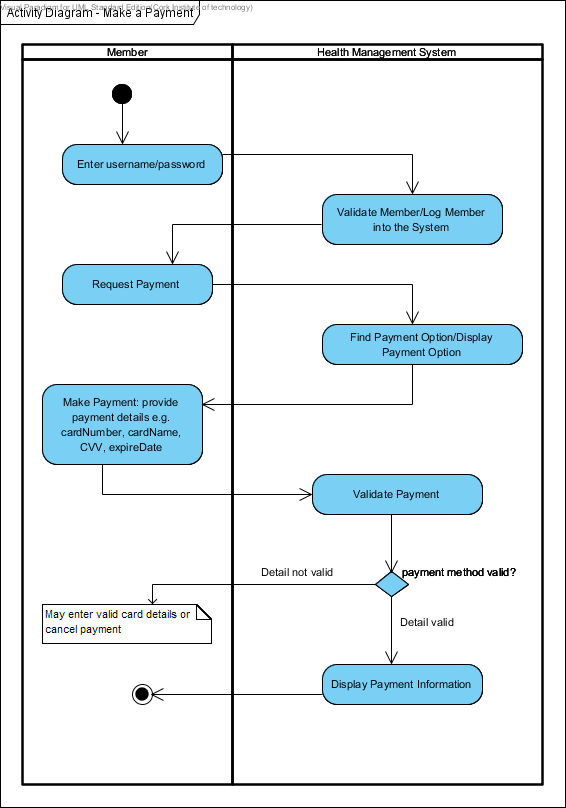
*Activity Diagram for the use case* ***Log into System***

The activity diagram below is used to illustrate the system function that is represented by the use case **Update Personal Information.** Here, it describes the activity flow from the Member, Instructor and a Duty Manager to the Health Management System.



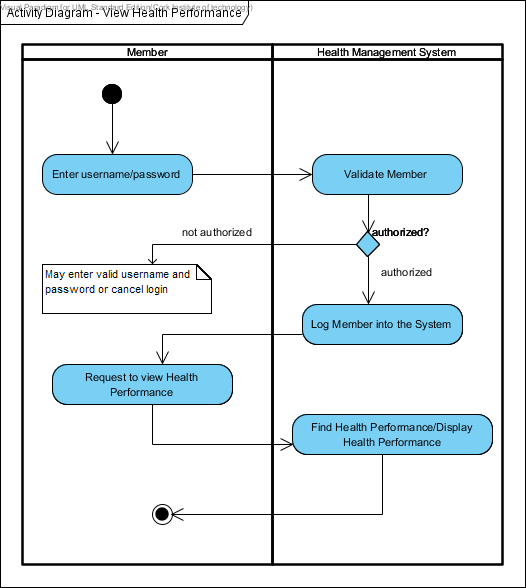
*Activity Diagram for the use case* ***Update Personal Information***

The activity diagram below is used to illustrate the system function that is represented by the use case **Make a Payment.** Here, it describes the activity flow from the Member to the Health Management System.



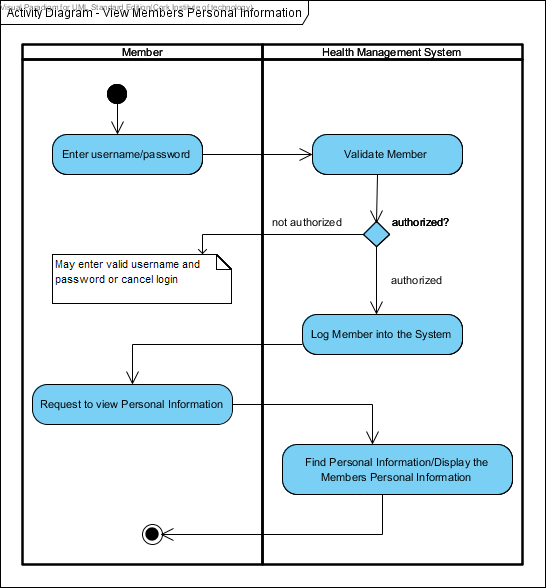
*Activity Diagram for the use case* ***Make a Payment***

The activity diagram below is used to illustrate the system function that is represented by the use case **View Health Performance.** Here, it describes the activity flow from the Member to the Health Management System.



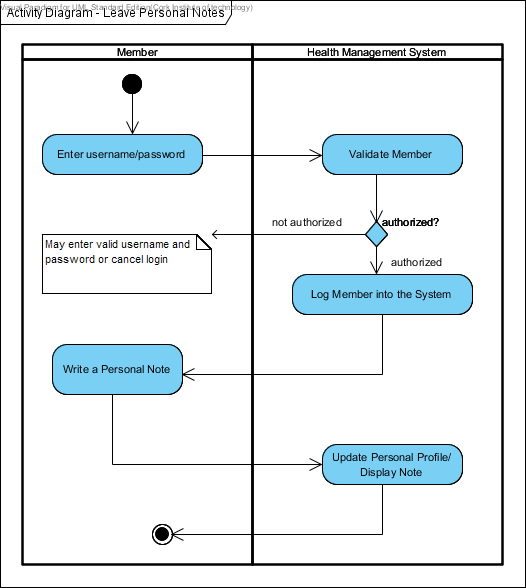
*Activity Diagram for the use case* ***View Health Performance***

The activity diagram below is used to illustrate the system function that is represented by the use case **View Members Personal Information.** Here, it describes the activity flow from the Member to the Health Management System.



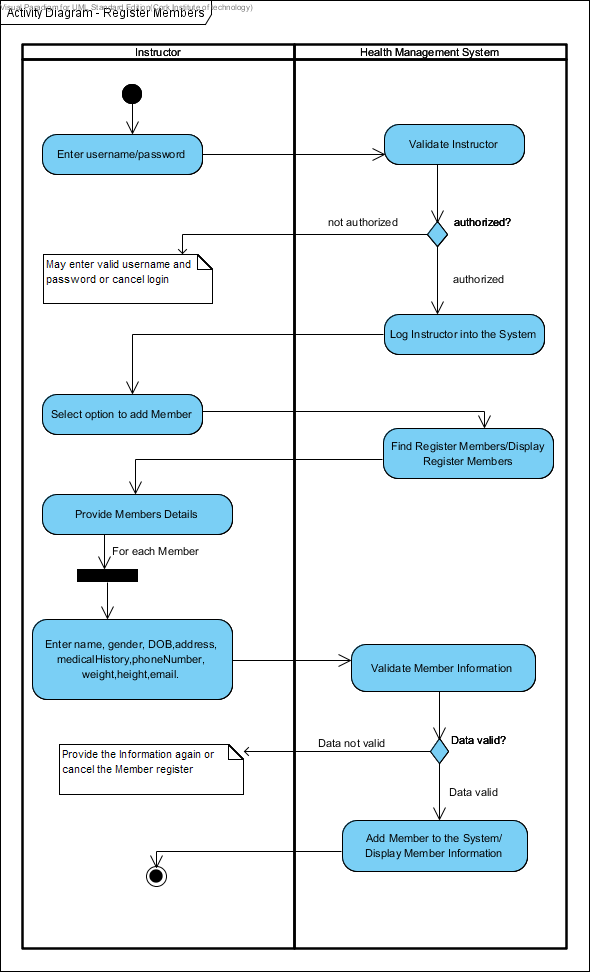
*Activity Diagram for the use case* ***View Members Personal Information***

The activity diagram below is used to illustrate the system function that is represented by the use case **Leave Personal Notes.** Here, it describes the activity flow from the Member to the Health Management System.



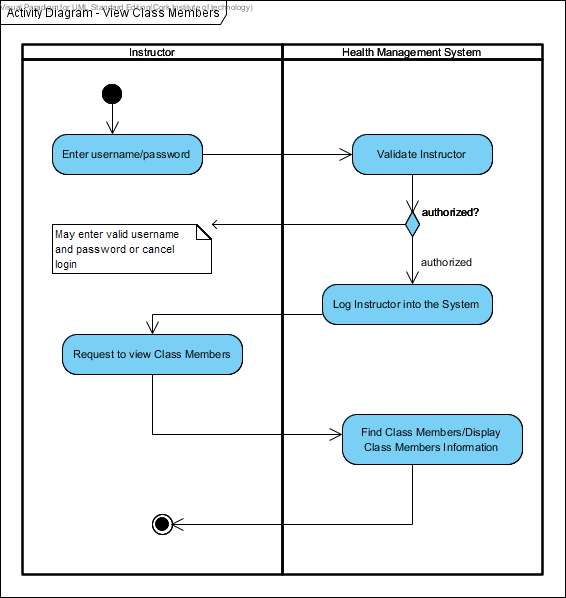
*Activity Diagram for the use case* ***Leave Personal Notes***

The activity diagram below is used to illustrate the system function that is represented by the use case **Register Members.** Here, it describes the activity flow from the Instructor to the Health Management System.



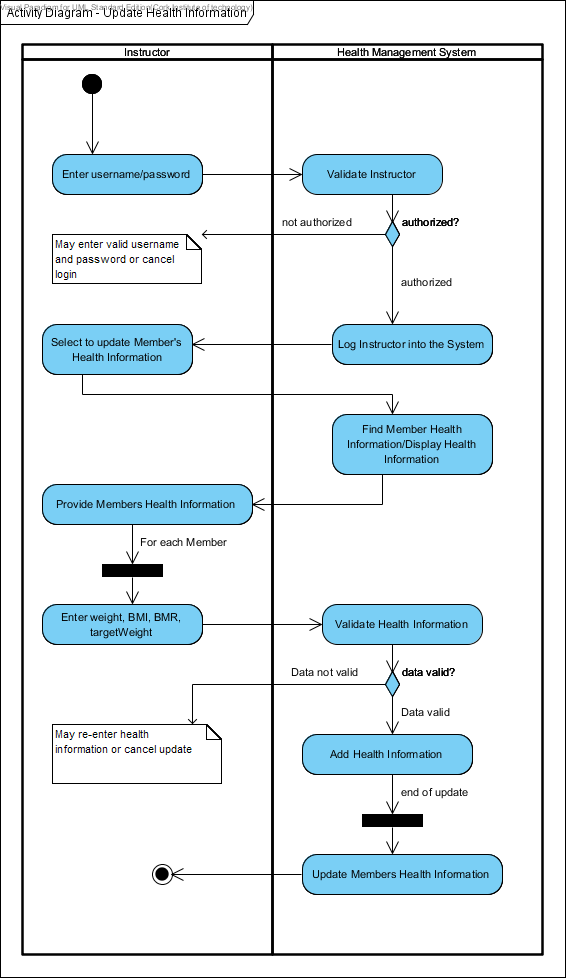
*Activity Diagram for the use case* ***Register Members***

The activity diagram below is used to illustrate the system function that is represented by the use case **View Class Members.** Here, it describes the activity flow from the Instructor to the Health Management System.



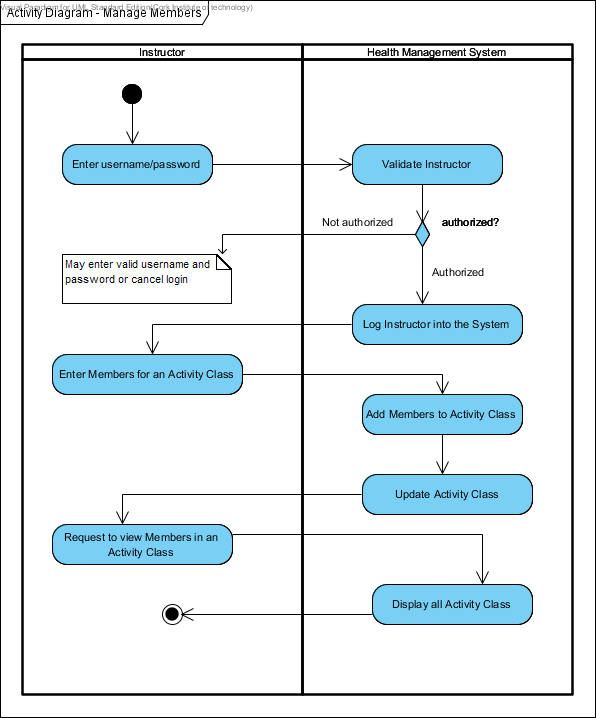
*Activity Diagram for the use case* ***View Class Members***

The activity diagram below is used to illustrate the system function that is represented by the use case **Update Health Information.** Here, it describes the activity flow from the Instructor to the Health Management System.



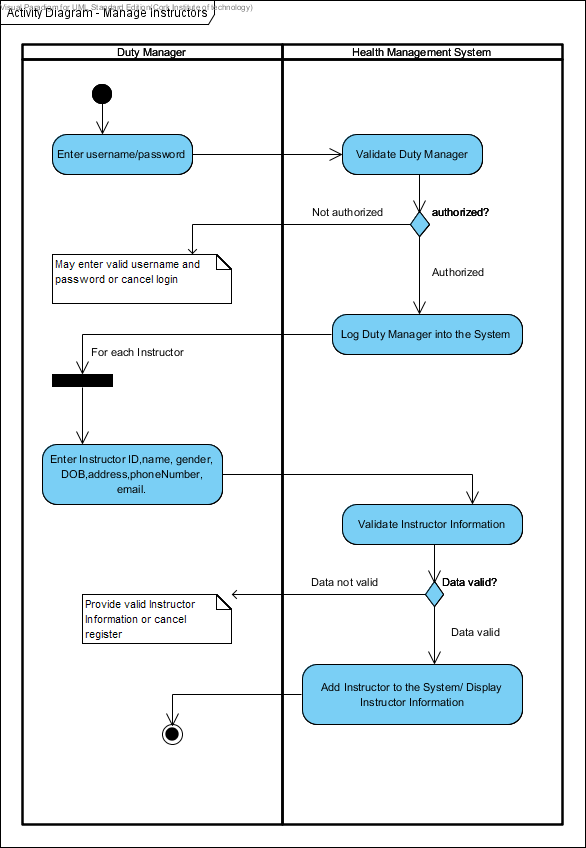
*Activity Diagram for the use case* ***Update Health Information***

The activity diagram below is used to illustrate the system function that is represented by the use case **Manage Members.** Here, it describes the activity flow from the Instructor to the Health Management System.



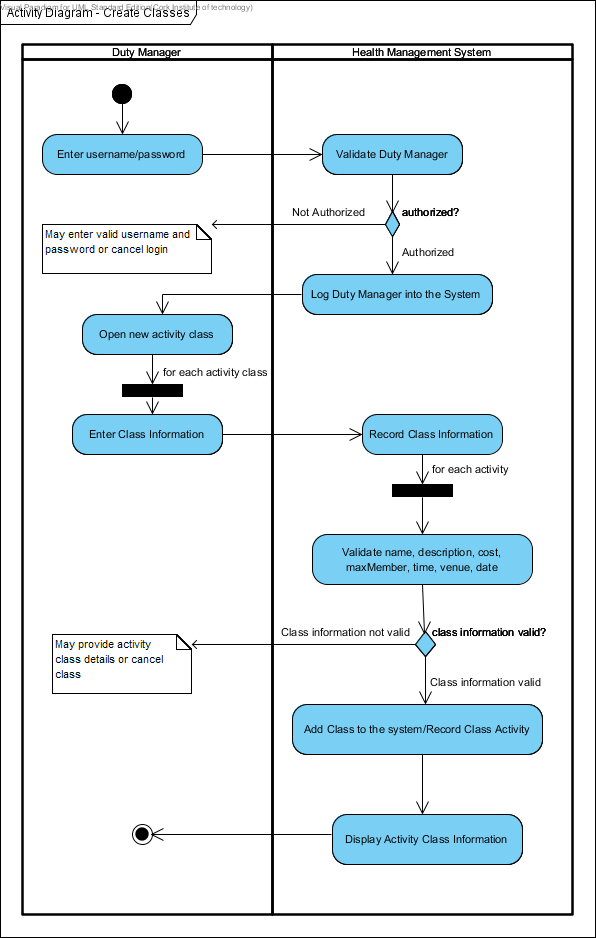
*Activity Diagram for the use case* ***Manage Members***

The activity diagram below is used to illustrate the system function that is represented by the use case **Manage Instructors.** Here, it describes the activity flow from the Duty Manager to the Health Management System.



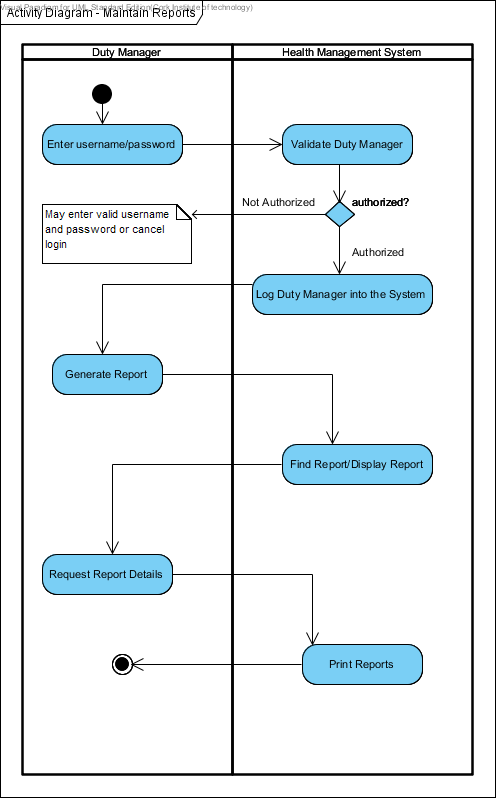
*Activity Diagram for the use case* ***Manage Instructors***

The activity diagram below is used to illustrate the system function that is represented by the use case **Create Classes.** Here, it describes the activity flow from the Duty Manager to the Health Management System.



*Activity Diagram for the use case* ***Create Classes***

The activity diagram below is used to illustrate the system function that is represented by the use case **Maintain Reports.** Here, it describes the activity flow from the Duty Manager to the Health Management System.



*Activity Diagram for the use case* ***Maintain Reports***

**Database Model**

We used MySQL to implement the database, running on a Linux server remotely accessible from anywhere in the world. The full SQL table definitions can be found in the soft-copy hand up under the **DatabaseClasses** folder. Most SQL queries used by the program are integrated into its source code in the database package.

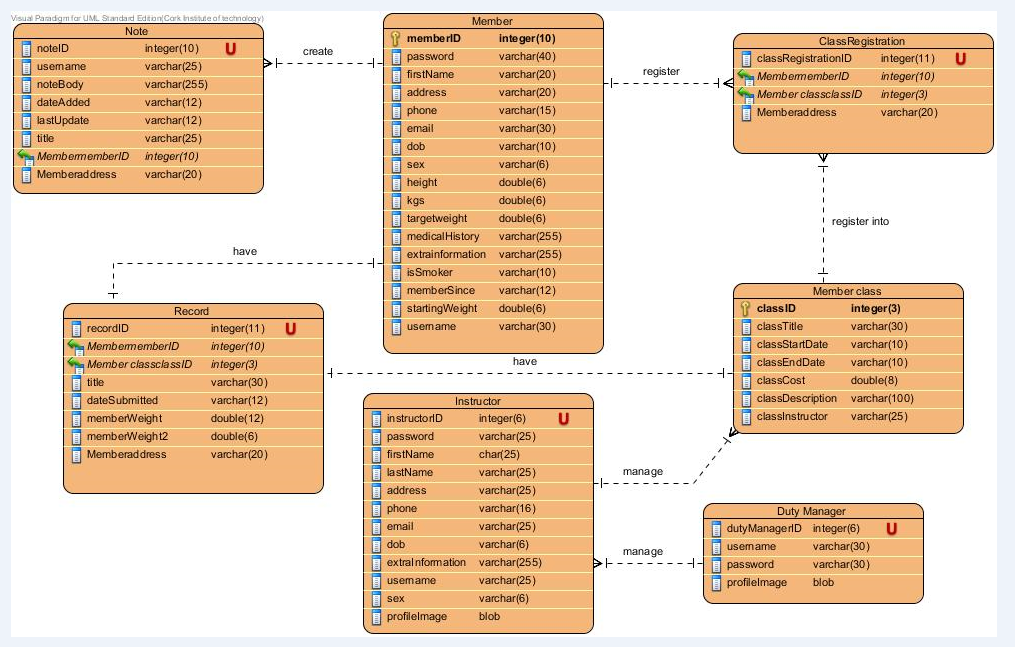
The diagram below describes the basic database model, arranged by key relations.

PK indicates a primary key.

FK indicates a foreign key

U indicates a unique key.

varChar indicates variable Character.



IMPLEMENTATION

**User Manual**

Purpose of this User Manual

The User Manual provides instructional support and guidance to authorised personnel who use the Health Management System to manage and view both Members and Activity Classes.

This User Manual focuses on access, navigation, use and management of the Health Management System.

Member Guide

**Log into System**

The following steps below indicate how a user logs into the Health Management System.

1. Firstly, the user starts the Health Management System.
2. The user enters their login credentials: username and password in the **“Login Credentials”** section.
3. User logs in successfully or provided some assistance otherwise.

**View Members Personal Information**

The view Members Personal Information authorizes members to view their personal information specified in the Use Case Specification.

The following steps below shows how Members can view their personal information:

1. The Member logs into the system by providing their login credentials.
2. When a Member has successfully logged into the system, he/she is presented with their personal details.
3. Otherwise, they can click on the **“Profile Page”** tab which will display the member’s information including their username, first name, last name, telephone number, email address, date of birth (DOB), and gender.

**Update Personal Information**

The update personal information enables a user to make desired changes to their profile information. This includes any of the information specified in the Use Case Specification.

The following steps show how a Member, Instructor and a Duty Manager update their personal information in the Health Management System.

1. Firstly, the user logs into the Health Management System.
2. The system displays their personal profile when the user selects the **“Profile Page”** tab.
3. Then, click on the **“Update”** button that will prompt a different tab for changes to be made.
4. The user makes the necessary changes to the personal information and saves the information by clicking on the **“Save”** button.

**View Health Performance**

This enables a member to view their health performance in the system. Members can view their weight maintenance after their regular physical activities.

The following steps show how a Member views his/her health performance in the Health Management System.

1. Firstly, the member logs into the Health Management System.
2. The system displays the member’s health performance in two different ways: as a **pie chart** and **a line graph**.
3. By selecting the tab **“Current Stats”** enables the member to view the health performance in a **pie chart.**

The Pie chart shows a circular chart divided into sectors, illustrating numerical proportion of weight loss.

To view member’s health performance in a line graph, the following steps should be taken:

1. Click on the **“Personal Graphs”** tabto view member’s health performance as a **line graph**.

A Line graph is used to record weight loss according to the date and weight coincide. Moreover, the line graph helps to track weight in kilograms on the vertical axis of a performance graph and the dates of the calendar on the horizontal axis.

*Recent Weeks*

*Since the member joined*

A member can view their activity class attendance.

1. Click on the “**Personal Graphs**” tab to view member’s attendance to an activity class.
2. The graph below enables the member to view their attendance to an activity class.
3. The Member can view the **Top 10 Members** that have lost magnificent weight over the weeks by clicking on the **“Top 100”** tab. This tab shows the top 100 members that have lost a significant weight. It also encourages competitive fitness results in the Lifestyle Weight Clinic Ltd.

**Leave Personal Notes**

This is the process of creating, updating and deleting notes from the system.

The following steps below show how an Instructor is added, updated and deleted from the Health Management System.

***Create a new Note***

1. Firstly, the Member is logged into the Health Management System.
2. The Member navigates to the **“Profile Page”** tab as shown below.
3. Enter personal notes in the field as provided below. This is shown on the right-hand side of the member profile page.
4. Then, click the **“Submit Note”** button. At this stage, the new note is added to the system. This will be displayed at the top right corner on the screen. Also, a message prompts to confirm the note been added to the system.

***Update a Note***

1. Firstly, navigate to the **“Profile Page”** tab and select the Note to update. Notice the instructor details will be populated in the text field.
2. Then, make the desirable changes to the note.
3. Click on the **“Update”** button to save the updated information into the system as shown below.
4. At this stage, the note is updated in the system.
5. A prompt will display to indicate that the note is now updated in the system. Click on the **“OK”** button to exit.

***Delete a Note***

1. To delete a note, navigate to the **“Profile Page”** tab and select the note to delete, from the **“Personal Notes”** section. Notice that the note details are populated in the text field: the date it was created, the ID, title and the date it was last updated.
2. Then, click on the **“Delete”** button to delete the note from the system as shown in the screenshot below.
3. A message prompt will display to verify the deletion: **“Are you sure you want to delete the note?”**
4. Click on the **“Yes”** button to delete. Otherwise, click on the **“No”** button to cancel.

After selecting the **no** option, the message will display indicating that the **note has been** deleted successfully from the system.

Instrutor Guide

**Update Health Information**

This describes the process of updating Member’s Health Information in the system.

The following steps below shows an Instructor guide in updating health information.

1. Firstly, the Instructor logs into the system using his/her credentials.
2. Then, navigate to the **“Members”** tab and select the member from the right-hand side on the screen that says: **“Member List”**.
3. Make necessary changes to the health information and click on the **“Update”** button.
4. A message will display to indicate that the member is updated in the system.

**Register Members**

1. The Instructor logs into the system by providing his/her credentials.
2. The Instructor navigates to the **“Members”** tab as shown below.
3. He/she enters the Member information requested.
4. Click on the **“Create”** button to add the new Member to the system. A message will prompt, indicating that the member is added to the system.
5. At this stage, the new member is added to the Member List section located on the right-hand side of the screen.

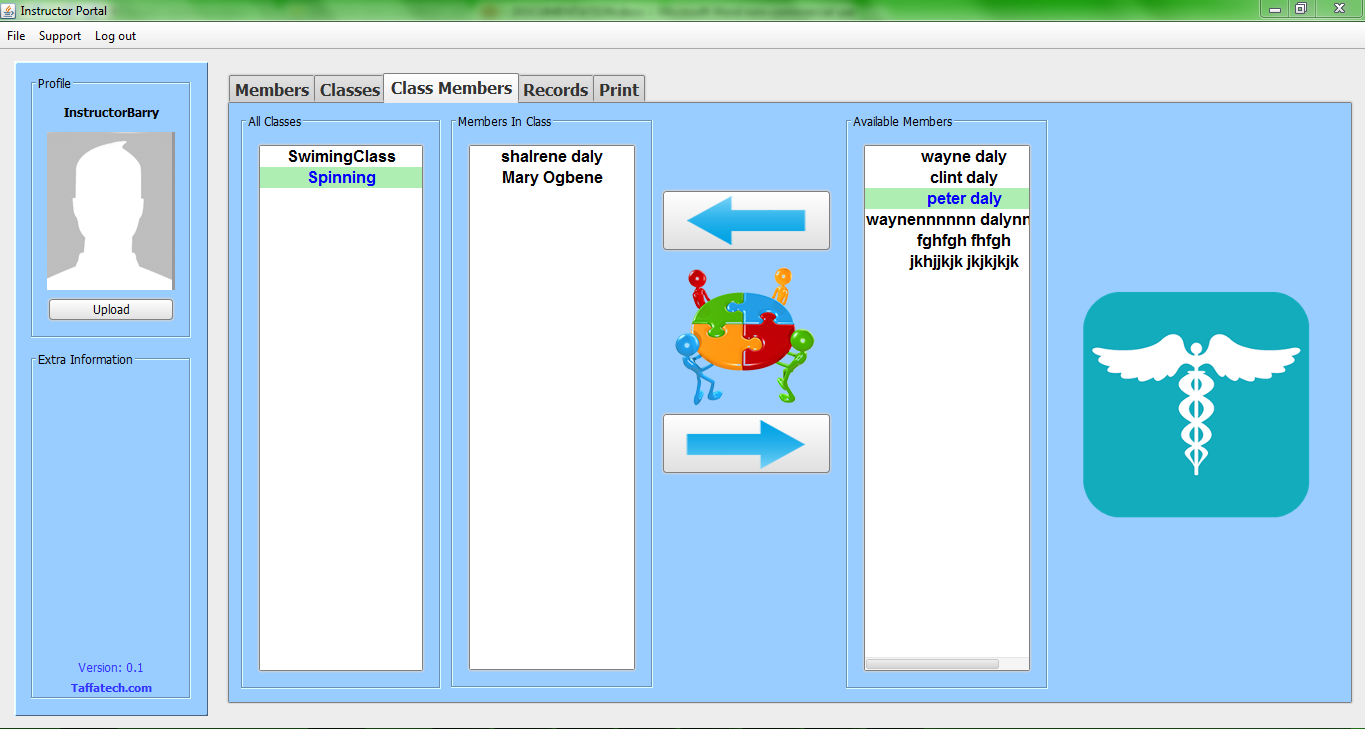
**Manage Members**

This is the process of adding and deleting Members from an Activity Class.

The steps below show how a Member can be added and deleted from a Class.

***Adding Members to an Activity Class***

1. Log into the Health Management System.
2. Navigate to **“Class Members”** tab and select an activity class from the list of activity classes in the **All Classes** List. This will be highlighted to show that the class has been selected.
3. On the far right is a list of available members (i.e. the members that wants to join an activity class).
4. Then select one or more members from the **Available Members List** and click the **“Left”** arrow. You will notice these members now appear in the **Members In Class** List. You can do this multiple times for other activity class.



***Deleting Members from an Activity Class***

1. Navigate to **“Class Members”** tab and select a class from the list of activity classes in the **‘All Classes’** List on the screen. This will be highlighted to show that the class has been selected.
2. You will see all the members of that activity class in the **“Members In Class”** list. Simply select one or more members you wish to remove from the class and click on the **“Left”** arrow key displayed on the screen. This will delete the member from the chosen class.
3. Then, you will notice that the member you have just deleted no longer appear in the **“Members In Class”** list. They will now appear as available members in the **‘Available Members’** list section. Moreover, the Instructor can go ahead to add them in other activity classes.

**View Class Members**

1. Navigate to **“Class Members”** tab and select a class from the list of activity classes in the **‘All Classes’** section on the screen. This will be highlighted to show that the class has been selected.
2. You will see all the members of that activity class in the **“Members In Class”** list.
3. is prompted for verification.

***Update an Instructor***

1. Firstly, navigate to the **“Instructor”** tab and select the Instructor to update from the **“Instructor List”** section. Notice the instructor details will be populated in the text field.
2. Then, make the desirable changes to the Instructor information.
3. After, click on the **“Update”** button to save the updated information into the system as shown below.
4. At this stage, the new Instructor is updated in the system.
5. A prompt will display to indicate that the instructor has been updated in the system. Click on the **“OK”** button to exit.

***Delete an Instructor***

1. To delete an Instructor, navigate to the **“Instructors”** tab and select the Instructor to delete, from the **“Instructor List”** section. Notice the instructor details will be populated in the text field.
2. Then, click on the **“Delete”** button to delete the Instructor from the system as shown in the screenshot below.
3. A prompt will display to verify the deletion.
4. Click on the **“Yes”** button to delete. Otherwise, click on the **“No”** button to cancel.

A message will prompt indicating that the Instructor is deleted successfully from the system.

**Additional Options**

A **“Clear”** button is provided to clear the text fields for the Duty Manager to input new instructor information.

**Create Classes**

***Create a new Class***

1. The Duty Manager is logged into the Health Management System by providing their username and password.
2. The Duty Manager navigates to the **“Classes”** tab as shown below.
3. Enter the class information required such as class title, start date, end date, cost, max members, and instructor.
4. Then, click the **“Create”** button. The new class is added to the system. This can be seen in the **Class List** section shown on the right-hand side of the screen. A message is prompted indicating that the new class has been added to the system.

***Update a Class***

1. To do this, navigate to the **“Classes”** tab and select the activity class to update from the **“Class List”** section. Notice the class details being populated in the text field.
2. Then, make the desirable changes to the Class information.
3. After, click on the **“Update”** button to save the updated information into the system as shown below.
4. At this stage, the new Class is updated in the system.
5. A prompt will display to indicate that the class has been updated in the system. Click on the **“OK”** button to exit. See below.

***Delete a Class***

1. To delete a class, navigate to the **“Classes”** tab and select the class to delete, from the **“Class List”** section. Notice the class details being populated in the text field.
2. Then, click on the **“Delete”** button to delete the class from the system as shown in the screenshot below.
3. A prompt will display to verify the deletion.
4. Click on the **“Yes”** button to delete. Otherwise, click on the **“No”** button to cancel.

A message will prompt indicating that the class (such as the one named below) has been deleted successfully from the system.

**Additional Options**

A **“Clear”** button is provided to clear the text fields for the Duty Manager to enter new information about an activity class.

**Maintain Reports**

This describes the process of maintaining member class reports.

***Add a Report***

1. The Duty Manager is logged into the Health Management System by providing his/her username and password.
2. The Duty Manager navigates to the **“Records”** tab as shown below.
3. Enter the required information into the text fields.
4. Then, click the **“Add Report”** button. The report is added to the system. This can be seen in the **View Selected Members Record** list shown on the left-hand side of the screen. A message is prompted indicating that the report has been created and added to the system.

*Creating a new Report*

***Delete a Report***

1. To delete a report, navigate to the **“Records”** tab and select the member report to delete, from the **“View Selected Members Record”** section. Notice the details for that particular member is populated in the text area.
2. Then, click on the **“Delete Report”** button to delete the report from the system.
3. A prompt will display to verify the deletion.
4. Click on the **“Yes”** button to delete. Otherwise, click on the **“No”** button to cancel.

*Deleting a Report*

***Print a Report***

1. To print a report, navigate to the **“Print”** tab and select the class and the member, from the **“Print Member Information”** section. Notice the details of the member are displayed on the right-hand side of the screen in the text area provided.
2. Click on the **“Print Member Information”** button to print the report record. A **Print** prompt will display for you to select the printing options (i.e. the orientation, appearance, and some general information about the printing options.
3. Then, click on the **“Print”** button to print the selected record. Otherwise, click on the **“Cancel”** button to cancel the operation

**Testing**

The whole team tested the product intensively. There were more than 15 issues raised where were recorded on GitHub.

Bug Report 1

|  |  |
| --- | --- |
| **Bug Logging Form** |  |
| **Product Information** |  |
| Product: | Health Management System |
| Version (include backup for example): | 4.1.1 |
| Component: | Updating Member database table |
| Reported Date: | 19/02/2014 |
|  |  |
| **Defect Information** |  |
| Steps to Reproduce: | Updating the information of member in the database table to reflect the current information |
| Actual Results: | When username is changed, the database still updated |
| Expected Results: | If the username is changed, the database should not update and should show error because the user name is the ID |
| Attachments/shots: |  |
|  |  |
| **Defect Severity** | 5 |
| Severity: | 5 |
| Priority: |  |
|  |  |
| **Fix Information** |  |
| Fixed Date: | 21/02/2014 |
| Verified Build: | 4.1.1 |
| How it was fixed | The PHP was modified |
|  |  |
| **Bug Status** |  |
| Status: | Resolved |
| Resolution: | Fixed |

Bug Report 2

|  |  |
| --- | --- |
| **Bug Logging Form** |  |
| **Product Information** |  |
| Product: | Health Management System |
| Version (include backup for example): | 4.1.1 |
| Component: | Updating class database table |
| Reported Date: | 20/02/2014 |
|  |  |
| **Defect Information** |  |
| Steps to Reproduce: | PHP Script inserting classes into database |
| Actual Results: | Changing of the name of PHP from “Class Name” to “Member Class “ and it’s impossible to insert into the database |
| Expected Results: | The class information should be insert into the database every time we query the database |
| Attachments/shots: |  |
|  |  |
| **Defect Severity** |  |
| Severity: | 4 |
| Priority: | 4 |
|  |  |
| **Fix Information** |  |
| Fixed Date: |  |
| Verified Build: | 4.1.1 |
| How it was fixed | Changed the Java code to accommodate the changing of the table name in the database |
|  |  |
| **Bug Status** |  |
| Status: | Resolved |
| Resolution: | Fixed |
|  |  |

Bug Report 3

|  |  |
| --- | --- |
| **Bug Logging Form** |  |
| **Product Information** |  |
| Product: | Health Management System |
| Version (include backup for example): | 4.1.1 |
| Component: | Health Management database |
| Reported Date: | 20/02/2014 |
|  |  |
| **Defect Information** |  |
| Steps to Reproduce: | The database is down |
| Actual Results: | The database got messed up from installing a new control panel, we could no longer log into the database and all the data in the database is lost, This is due to going into configuration files |
| Expected Results: | The database should be up 24/7 |
| Attachments/shots: |  |
|  |  |
| **Defect Severity** |  |
| Severity: | 5 |
| Priority: | 5 |
|  |  |
| **Fix Information** |  |
| Fixed Date: | 21/02/2014 |
| Verified Build: | 4.1.1 |
| How it was fixed | We installed Virtual min, then uninstalled it and installed the panel again |
|  |  |
| **Bug Status** |  |
| Status: | New |
| Resolution: | Fixed |

EVALUATION

**Revised System Objectives**

We learned so many things in the process of developing this java desktop application

We learned the new concept of integrating PHP, JASON and Java in producing better secured application in today’s unsecured internet technology. It was a huge eye opener for us since we were taught how to use Java for both client and server side application which is no more relevant in today’s society.

We also discovered the value of easy collaboration and integration of code from different people, the use of source control tool (GitHub) provided has become part of our development process, and most importantly is the best tool used.

Carrying out this project has greatly improved our programming skills; we learned new technics of creating a highly secured application, the use of the model, view and controller making our application more object oriented and easily coupled. We also learned the usefulness of multi-threading and its associated currency problems and creating a secured password for the users of the application.

The important of quality requirement gathering cannot be over emphasized, due to quality requirements gathering we had clear ideas of what to be done and how to get it done, the integration of PHP, JASON and Java to interact with the MySQL database even improved on the quality of application we envisage.

The importance of planning became obvious at the initial stage. Had we not had a solid idea of what to implement, development would have been short-sighted and many features would now be absent due to implementation difficulty. Planning gives you foresight.

Most importantly, we learned so much on how to work as a group because it is how work is applied to the real world. Despite the fact that, we had different language barriers we are able to collaborate and work along each other without any issues.

**Problems Encountered**

1. **Contracts with Code Method**

We were taught to use java in building software applications and interacting with the database, since the mock desktop application we developed in third Semester were all developed using java. However, we learned through Wayne that java alone is not secure enough in the development of today’s desktop application. Which was why, in real world PHP and Jason are used to connect java interaction with the database, making coding a bit more difficult for the other member of the group. There was conflict between using java alone and including PHP, Jason and java.

1. **GitHub Problem**

Because we are new to GitHub, we encountered many conflicts when we were merging our work into the **Mater Repository**. Due to the fact, we were working on the same file sometimes and in addition the two programmers were working on different speed. We resolved this by only allowing one person (the creator) to upload into the GitHub master repository, whereas the second programmer has to email his own code to the creator for integration.

1. **IDE - (Integrated Development Environment)**

There was a deliberation on which development platform to use and which of the IDE Platform will be of most benefit to the development of the desktop java application. Later, we chose NetBeans because it provided better qualities when using the available NetBeans tools to build our automated user interface. It was due to not having the time to write a java code for the design of the application user interface. In order words, we would have chosen Eclipse IDE to develop our java application.

1. **Language Barriers**

The issue of a language barrier is very visible because we have three different languages and culture. Therefore, sometimes communication can be very difficult. It resulted in time consuming whereby, we needed to explain ourselves in order for the rest of the team to understand. And most times, we had to repeat ourselves because we do not know if a member of the team does not understand the point of view.

1. **Coding & Debugging**

There were some tasks we found difficult to carry out during the completion of the software product such as uploading a profile-image of a Member, Instructor, and the Duty manager. In addition, assigning a member to a specific Instructor and there were some bugs that took quite some time to fix, e.g. changing or updating the content of a combo box.

1. **Accessing the Server:**

We used Wayne personal server for the implementation of our database. But not having direct access to the server prevented the other programmer from testing some codes quickly. Even though some stuffs were temporary.

1. **Time Constraint:**

Due to limited time available, it was quite difficult to experiment new ideas. Adding to this was the other course modules we participated.

**OutLook**

Because the product is only a desktop application we hope that, in the future a web-interface can be incorporated to the application, making the system very interactive with the end users, which was the main objective for the development of the Health Management System.

If more time and more were invested, the product could even be changed to act as a management system for areas such as an office, football or rugby club. The reason is that; the internal code is extremely flexible, generalizable and reusable.

In addition, businesses can also implement other features such as a Duty Manager personal information GUI. It can be possible because, the system is significantly flexible and adaptable to any environment.

With more time in the future, the payment use case could be implemented. In this way, the system will be able to keep a record of member payment and send them notice via automated email when their subscriptions are in arrears or when their payment is due. Moreover, adding a feature that will not grant them access to the system if they have reached their maximum payment allowed being in arrears.