



ASHESI UNIVERSITY

HEALTH MANAGEMENT SYSTEM

APPLIED PROJECT

B.Sc. Computer Science

Frederick Peter Apatu Plange

2019

ASHESI UNIVERSITY

Health Management System

APPLIED PROJECT

Applied Project submitted to the Department of Computer Science,
Ashesi University in partial fulfillment of the requirements for the award of
Bachelor of Science degree in Computer Science

Frederick Peter Apatu Plange

April 2019

DECLARATION

I hereby declare that this applied project is the result of my own original work and that no part of it has been presented for another degree in this university or elsewhere.

Candidate's Signature

.....

Candidate's Name

.....

Date:

.....

I hereby declare that preparation and presentation of this applied project were supervised in accordance with the guidelines on supervision of applied project laid down by Ashesi University

Supervisor's Signature

.....

Supervisor's Name

.....

Date:

.....

Acknowledgment

I want to thank my supervisor, Mr. David Sampah, who guided me throughout the project. I would also like to thank my friends Wuyeh Jobe & Faith Mueni for their support and encouragement.

Abstract

Hospitals in Ghana still operate with the file-based system approach due to the absence of a health management system. While this approach works, it is an ineffective way to store and manage health records. Hospitals consume large amounts of paper in preparing medical records, lab reports, and prescriptions. With information scattered all over the place, it becomes difficult to manage and keep track of.

In this paper, a solution proposed is a health management system whereby various hospital personnel log in and digitally perform their duties. The aim is to enhance medical treatment quality, facilitate with distance communication, data sharing, and improve internal communication. A key functionality comes in the form of report generation, which seeks to automate the process of creating reports which follow the format of reports submitted to higher authorities such as the district health ministry.

Table of Contents

DECLARATION.....	i
Acknowledgment.....	ii
Abstract.....	iii
List of Figures.....	vii
Chapter 1: Introduction	1
1.1 Introduction	1
1.2 Background.....	2
1.3 Motivation	4
1.4 Significance	4
1.4.1 Reducing Medical Error and Improving the Quality of Medical Care	4
1.4.2 Reducing Costs and Raising Efficiency	5
1.4.3 Improving the Quality of Service	5
1.5 Related Work.....	5
1.5.1 Problems with Current e-health Implementations.....	5
1.5.2 Cloud Computing	6
Chapter 2: Requirements	8
2.1 Overview	8
2.2 Requirements Gathering & Analysis	8
2.3 User Identification.....	9
2.5 Functional Requirements	10
2.6 Non-Functional Requirements	18
Chapter 3: Architecture & Design	19
3.1 Project Domain.....	19
3.2 High-Level Architecture.....	20
3.3 Overview of the Model View Controller (MVC) Architecture	20
3.3.1 The Model.....	21
3.3.2 The View	23
3.3.3 The Controller.....	23
3.4 Technologies & Frameworks.....	24
3.5 Security & Backup	25

3.5.1 Views	25
3.5.2 Separating Web Server from the Database Server	25
3.5.3 Encryption of Files.....	25
3.5.4 Backup	25
3.5.5 Limiting Data Use and Access	26
3.5.6 Encrypted Passwords	26
Chapter 4: Implementation.....	27
4.1 Overview of Implementation.....	27
4.2 Structure of Root Folder.....	27
4.3 User Interface Design & Key Functionalities	28
4.3.1 Login	29
4.3.2 Registering Users	31
4.3.3 Check-in Patients.....	33
4.3.4 Add Vitals.....	35
4.3.5 Doctor Visit	37
4.3.6 Lab tests.....	39
4.3.7 Booking Appointments.....	40
4.3.8 Deleting Records (vitals, doctor consultation, lab tests)	42
4.3.9 Checking Out	44
4.3.10 Outbreak Reporting	45
Chapter 5: Testing	47
5.1 Overview	47
5.2 Developmental Testing.....	47
5.2.1 Unit Testing.....	47
5.2.2 Component Testing.....	47
5.2.3 System Testing	49
5.2.4 Compatibility Testing.....	49
5.3 User Testing	50
5.3.1 Receptionist Component Testing	51
5.3.2 Nurse Component Testing	51
5.3.3 Doctor Component Testing.....	52

5.3.4 Pharmacist Component Testing	53
Chapter 6: Conclusion	54
6.1 Overview	54
6.2 Challenges	54
6.3 Future Work	54
6.4 Conclusion.....	56
References	57

List of Figures

Figure 1: Admin use case diagram	11
Figure 2: Receptionist use case diagram.....	12
Figure 3: Nurse use case diagram.....	13
Figure 4: Doctor use case diagram	14
Figure 5: Pharmacist use case diagram	16
Figure 6: Lab technician use case diagram	17
Figure 7: High-level architecture	20
Figure 8: MVC architecture	21
Figure 9: EER diagram	22
Figure 10: Root folder structure.....	27
Figure 11: Login Page.....	29
Figure 12: Login Sequence Diagram.....	29
Figure 13: Query for login	30
Figure 14: Login function from LoginController page	30
Figure 15: Sequence diagram for registering users	31
Figure 16: Add nurse page	31
Figure 17: Query to add nurse.....	32
Figure 18: addNurse function in adminController page.....	32
Figure 19: Sequence diagram for checking-in a patient.....	33
Figure 20: Patient check-in page	33
Figure 21: Query to check-in a patient	34
Figure 22: checkInUser function in nurseController page	34

Figure 23: Sequence diagram for adding vitals	35
Figure 24: Today's patient- visit page	35
Figure 25: Add vitals page	36
Figure 26: Query to add vitals	36
Figure 27: addVitals function in the nurseController page	36
Figure 28: Doctor consultation page	37
Figure 29: Query to add doctor consultation result	38
Figure 30: addGp function in doctorController page	38
Figure 31: Lab tests page	39
Figure 32: updateLab function	40
Figure 33: Booking doctor-patient appointment page	40
Figure 34: Query to insert an appointment	41
Figure 35: Query to check for appointment clashes	41
Figure 36: insertAppointment function receptionistController page	41
Figure 37: Progress table.....	42
Figure 38: Query to archive a patient record.....	43
Figure 39: deleteDetails function in doctorController page	43
Figure 40: Check-out patient page	44
Figure 41: Query to check a patient out	44
Figure 42: checkOutUser function in pharmacistController page.....	44
Figure 43: Outbreak report page	45
Figure 44: Query to select outbreak details.....	45
Figure 45: Page to create report of outbreak	46

Figure 46: Registering a new patient	48
Figure 47: Updating patient information	48

Chapter 1: Introduction

1.1 Introduction

In Ghana today, hospitals still maintain the usage of excel spreadsheets and hard copy medical records as opposed to digital form, and this is mostly due to the absence of information management systems. Information management systems are typically expensive; hence, they are difficult to maintain. Very often we visit hospitals and are obliged to fill out forms and observe hospital personnel busy filling out forms, and this becomes a problem when dealing with numerous records which may lead to human lapses and inefficiencies. Hospitals need to store these records safely in case of any future use, also records needs need to be retrieved efficiently which further allows for quick and concurrent access.

The main idea of this paper is to promote the use of a digital hospital information management system. Such a system would also aim to enhance medical treatment quality, facilitate with distance communication and data sharing, improve internal communication, and bring about a sense of structure and organization. Moreover, while the solution addresses a hospital's internal operations, it also serves to centralize all hospitals and allow each hospital to generate specific reports that are formatted with the typical daily, weekly and monthly reports expected by higher bodies such as the health ministry. In doing so, the confidentiality and security of patient's records will be of the highest priority in the system. Also, measures will be adopted to prevent any issue from arising. With such a dedicated system in place, hospitals can build a good relationship with their patients, increasing their satisfaction rate, and provide an easier means for a higher body to operate.

As such this paper will discuss the transition from the paper-based file records along with the use of excel spreadsheets to a dedicated digital management system.

1.2 Background

A health management information system (HMIS) is a data collection system in which health data of patients are recorded, stored and processed for planning, implementation and general use for the future [12]. The HMISs in most developing countries are inefficient; the Taiwan University Hospital, for example, had a healthcare information system built over 25 years ago, and at that time it provided a stable, high performance and reliable features to handle patient data. However, the built-in technologies have now become obsolete. The expenses of maintaining the system have also become somewhat of a financial burden. Furthermore, healthcare regulations are changing rapidly, and the system is unable to reflect the modifications and enhancements [1].

Health management information systems in developing countries are significantly affected by the unreliability of data resulting from underreporting. It was reported that vital health decisions in sub-Saharan Africa are made based on rough estimates of disease and treatment burdens. The findings indicate that underreporting remains a significant theme and stems from a lack of knowledge and practice among health workers characterized by insufficient analysis skills and lack of training [6]

One of the significant problems with the attempted health management information systems in developing nations is the fact that they are isolated from other e-health activities in the country. E-health activities involve health care practices which are supported by electronic processes [11]. Due to the isolation, it is challenging to integrate into the country's healthcare development. As a result, these efforts are often discontinued and never integrated for use. Sri

Lanka made attempts to integrate health management information systems with following e-health activities, but due to lack of support, operations were terminated and discontinued [5]. Meanwhile, in recent years, about 17 thousand hospitals in China have put in tremendous and continuous efforts in establishing hospital information systems; however, the outcome has not been pleasing [3].

In Tanzania, the first health management information system was launched in 1993 while the second was launched in 1998. The first was entirely in English, and upon testing, they identified that users had limited commands in the language. Therefore, it was later changed to Kiswahili. The latest system covers all health programs and health care services and requires all health facilities to adopt and use this system, and report to the district health authority regularly. The system was intended to optimize the performance of health services at all levels of administration through the timely provision of all the necessary information needed by health practitioners to monitor, evaluate and plan their activities. However, to achieve these goals, the system needs to be highly integrated, fully functional, reliable and consistent [6].

These concerns provide an excellent opportunity for the development of a fully functional and reliable health management information system that serves as a standard for all hospitals. By building such a dedicated web application for hospitals, the developer aims to enable hospitals to carry out their daily activities in a more efficient manner, improve their relationships with their patients through constant communication, as well as provide a quality health service due to information and patient records being readily available.

1.3 Motivation

We live in a world where technology has advanced significantly and has become a significant part of our everyday life. Computers can process millions of information and store large amounts of data, therefore why not use computers to do what they are best at while leaving humans to focus on the critical activities that they carry out? Technology can be used to do all the processing work and provide what the user needs to make the best decision. Adopting such a system would not only allow doctors to focus on critical things, but it will directly prevent unnecessary deaths that seem to occasionally occur due to minor errors such as the improper management of medical prescriptions. Hence these medical errors can be significantly eliminated with the adoption of a health management system.

1.4 Significance

1.4.1 Reducing Medical Error and Improving the Quality of Medical Care

The quality of medical care should correspond to the level of technological and scientific development at a given period. However, that is not the case in some of the developed nations such as the US. It was reported that there were approximately 522,000 cases of severe medical error, causing the death of about 44,000-98,000 persons in the US. Currently, the level of medical treatment varies from country to hospital, and doctor. Nonetheless, establishing a health management information system is a very effective and practical way to improve the quality of medical treatments [3].

1.4.2 Reducing Costs and Raising Efficiency

A health management system will aid in dramatically reducing costs. One way it achieves this is by reducing the consumption of paper. Typically, hospitals consume large amounts of paper in preparing medical records, prescriptions, and test reports, and all these can be eliminated with the use of a digital system. Moreover, it can reduce indirect costs. For example, a reduction of paperwork would also lead to a reduction in the number of staffs required for managing medical records, prescriptions and test reports [3].

1.4.3 Improving the Quality of Service

The integration of a health management information system focuses on the automation of specific patient information. Currently, similar information is scattered on papers, which makes it very difficult to organize, identify, monitor and update such information. This becomes an issue for those suffering from chronic illnesses who require regular monitoring and assistance [3].

1.5 Related Work

1.5.1 Problems with Current e-health Implementations

Much has been written about the problems with the current and tested e-health implementations. The cost of maintaining such a system was a crucial problem in Taiwan University Hospital, and this makes sense because they were using a legacy system, which is considered an old method of going about things. As healthcare regulations were changing, the management information system failed to reflect the modifications and could not cope with the advancement in technologies such as wireless connections, user-friendly web interface, etc. [1]. Moreover, it is said that health management information systems in developing countries are inefficient and are greatly affected by the unreliability of data resulting from underreporting.

Findings from the sub-Saharan region showed that the problem of underreporting is linked to lack of knowledge and practice among health workers characterized by insufficient analytic skills, training and lack of initiative for using information [6]. Also, there have been significant advances to establish hospital management systems in China; however, despite all the effort, not even a single hospital has become completely digitized [3]. The main issue remains the fact that hospitals lack standards and consistency; therefore the digitization of hospitals must deal with challenging issues regarding the policies adopted in the hospital. Moreover, due to the lack of models and experience, it's difficult to establish management systems; there are lots of factors to consider including the work style of the clinical doctors, institutions, and patients [3].

1.5.2 Cloud Computing

According to Microsoft Azure, cloud computing “is the delivery of computing services—servers, storage, databases, networking, software, analytics, intelligence and more—over the Internet (“the cloud”) to offer faster innovation, flexible resources, and economies of scale. You typically pay only for cloud services you use, helping lower your operating costs, run your infrastructure more efficiently and scale as your business needs change” [7,8]

Users of these services only need an internet connection and a browser to use all the resources in the cloud [8]. Cloud computing has several benefits:

- **Scalability and Performance:** Resources are delivered per-use on the cloud; therefore, systems that make use of cloud technologies can handle high usage and still maintain good performance [8,9].
- **Decreased costs:** resources are paid for per-use, instead of paying for the whole infrastructure [8,9].

- Loose coupling: In a cloud-based architecture, software components can interact through well-defined interfaces, such as Web Services and API interfaces [8,9].
- Broad connectivity: Users of the systems will always have access to the system through any device and an internet connection [8,9].
- Fewer maintenance problems: Since the hardware infrastructure is not owned, there are fewer maintenance problems [8,9].

Chapter 2: Requirements

2.1 Overview

This chapter gives a thorough analysis of the functionalities the system would offer to health institutions in Ghana. As introduced earlier, this project entails a health management information system that hospitals can use to make more informed decisions. As such, the requirements this project seeks to meet would come directly from its potential users.

According to Tan, it is critical to achieving a good grasp of the essential functions of a health information system. Health management information systems are typically built upon the conceptualization of 3 central phases: data input, data management, and data output. As such, the findings from this study will form the foundation of the requirement specification, while the more intricate features will be abstracted directly from the potential users [10].

2.2 Requirements Gathering & Analysis

The requirement gathering process was carried out through the means of interviews with various hospital personnel. Consequently, the functional and non-functional requirements were defined as a result of the responses provided.

Based on the data collected, the key point noted was to separate functionalities between the users. Therefore, the system needs to be built in a manner that allows a user access to only specific functionalities and features. Moreover, the system should speed up the process of retrieving records and automate the process of generating reports.

2.3 User Identification

This health management information system is intended to be used by Hospitals in Ghana and generally any health institutions looking to switch to a digital way of operations. The system is presented as a web application; therefore, its users are required to have a computer with an internet connection to access the application. Furthermore, users should have some basic understanding of computers and the internet.

The various user roles include admin, receptionist, doctor, nurse, pharmacist, and lab technician. Each of these users would be able to log in and have a different view of the system.

2.4 Scenarios

Following is scenario one to offer further insight into how the application may be used:

A patient has been feeling ill for the past two days, so today the patient has decided to walk into the hospital to meet the receptionist. The receptionist uses the system to check whether the individual is already registered in the system. If the patient is new, the patient is registered into the system by the receptionist. Consequently, the receptionist can check-in the patient. The record of the patient is then made available to the nurse. The patient then heads to the nurse who checks the patient vital signs such as blood pressure, height, and weight. The system stores this record and is made available to the doctor. During the consultation, the doctor records the patient's vital signs, symptoms, diagnosis, etc. If lab tests need to be carried out, the doctor submits the tests, which can then be accessed by a lab technician. If no tests are required, the doctor submits medicine prescriptions. The pharmacists upon receiving the patient record can provide the drugs to the patient.

Following is scenario two to offer further insight into how the application may be used:

An outbreak of malaria has been discovered in an area, and all hospitals are required to submit an outbreak report containing patient records, reporting malaria during a specific timeframe. The admin can log in to the system and use the side navigation to navigate to the reports section. He can then type the outbreak condition (in this case, malaria), pick a timeframe and click a button to generate a report that will download automatically.

2.5 Functional Requirements

General requirements:

- ❖ Various users within a hospital should be able to log in.
- ❖ The system should redirect users to an appropriate page when they log in.
- ❖ The system should backup all transactions.

Admin:

The hospital administrator can use the admin account.

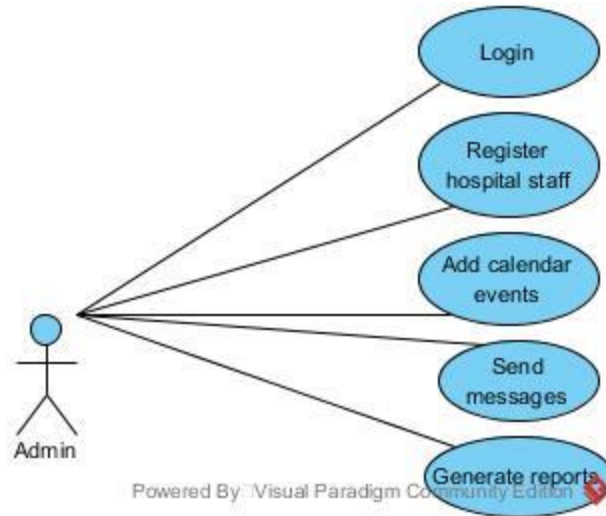


Figure 1: Admin use case diagram

- ❖ An admin must be able to create departments.
- ❖ An admin must be able to register the various users of the system (doctors, lab technicians, receptionists, nurses, and pharmacists).
- ❖ An admin can send messages to the various users registered at the hospital
- ❖ An admin should be able to add events to a private calendar
- ❖ An admin must be able to generate medical reports based on medical records.
- ❖ An admin must be able to manage his/her profile

Receptionist:

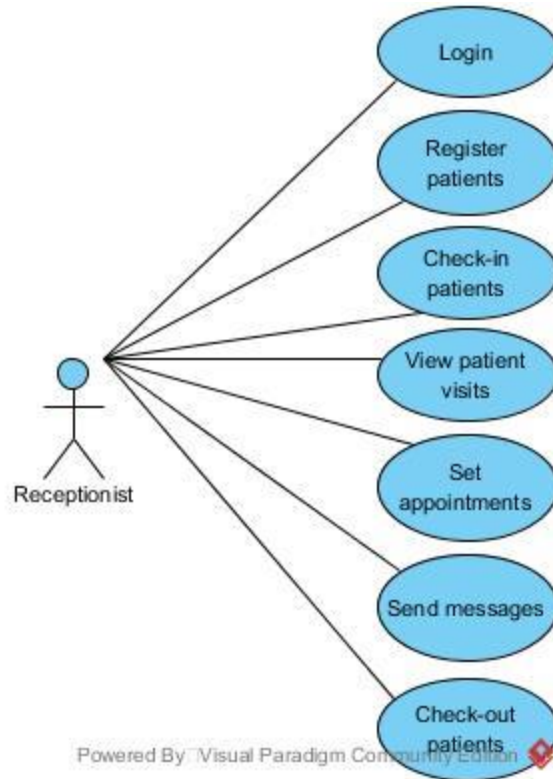


Figure 2: Receptionist use case diagram

- ❖ A receptionist must be able to register a new patient.
- ❖ A receptionist must be able to update patient details.
- ❖ A receptionist must be able to view a list of all other users registered to the hospital.
- ❖ A receptionist must be able to check in patients.
- ❖ A receptionist must be able to add beds.
- ❖ A receptionist must be able to send messages to the various users within the hospital.
- ❖ A receptionist must be able to set appointments between doctors and patients.
- ❖ A receptionist must be able to add events to a private calendar.
- ❖ A receptionist must be able to manage bed allotments.
- ❖ A receptionist must be able to view all the patients who visited today.
- ❖ A receptionist must be able to view a list of all patient visits.

- ❖ A receptionist must be able to view a progress table showing the patient's visit progress.
- ❖ A receptionist must be able to check-out a patient.
- ❖ A receptionist must be able to manage his/her profile.

Nurse:

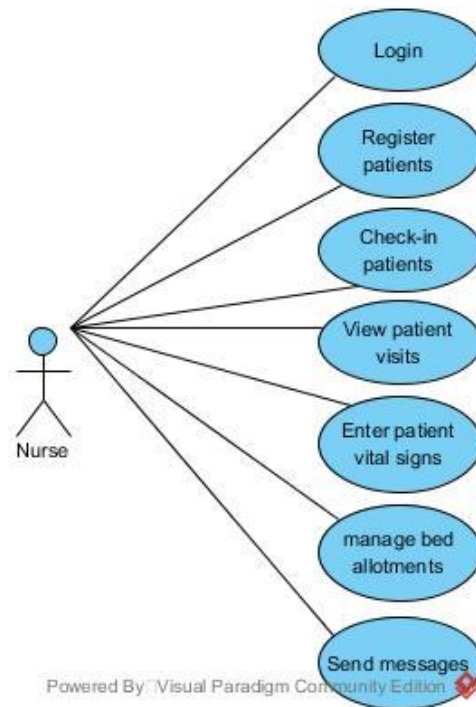


Figure 3: Nurse use case diagram

- ❖ A nurse must be able to register a new patient.
- ❖ A nurse must be able to update patient details.
- ❖ A nurse must be able to view a list of all other users registered to the hospital.
- ❖ A nurse must be able to check-in patients
- ❖ A nurse must be able to add beds.
- ❖ A nurse must be able to view all the patients who visited today.
- ❖ A nurse must be able to view a list of all patient visits.
- ❖ A nurse must be able to send messages to the various users within the hospital.

- ❖ A nurse must be able to enter and update patient vitals (height, weight, blood pressure, etc.)
- ❖ A nurse must be able to archive only patient vital sign records.
- ❖ A nurse must be able to view a progress table showing the patient's visit progress.
- ❖ A nurse must be able to manage bed allotments.
- ❖ A nurse must be able to add events to a private calendar.
- ❖ A nurse must be able to manage his/her profile.

Doctors:

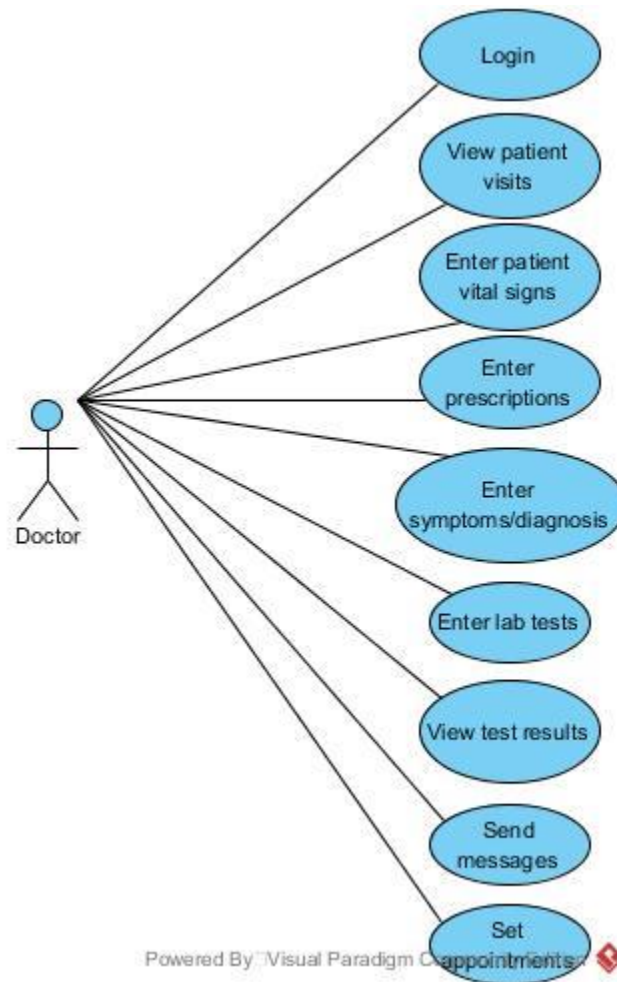


Figure 4: Doctor use case diagram

- ❖ A doctor must be able to view a list of all other users registered to the hospital.
- ❖ A doctor must be able to view all the patients who visited today.
- ❖ A doctor must be able to view a list of all patient visits.
- ❖ A doctor must be able to enter and update patient vitals (height, weight, blood pressure, etc.)
- ❖ A doctor must be able to enter prescriptions for patients.
- ❖ A doctor must be able to enter patient symptoms and diagnosis.
- ❖ A doctor must be able to send prescribed tests to a lab technician.
- ❖ A doctor must be able to see the lab results for a patient's lab tests.
- ❖ A doctor must be able to archive patient records.
- ❖ A doctor must be able to view a progress table showing the patient's visit progress.
- ❖ A doctor must be able to send messages to the various users within the hospital.
- ❖ A doctor must be able to set events on his/her calendar.
- ❖ A doctor must be able to set appointments with a patient.
- ❖ A doctor must be able to manage his/her profile.
- ❖ A doctor must be able to view a list of in-patients.
- ❖ A doctor must be able to check-out a patient.

Pharmacist:

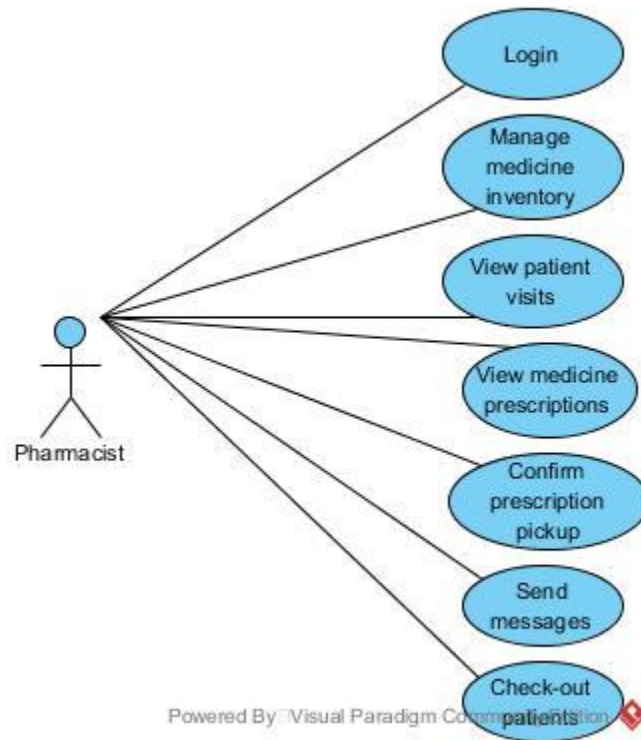


Figure 5: Pharmacist use case diagram

- ❖ A pharmacist must be able to view a list of all other users registered to the hospital.
- ❖ A pharmacist must be able to manage the medicine inventory.
- ❖ A pharmacist must be able to store supplier details
- ❖ A pharmacist must be able to view all the patients who visited today.
- ❖ A pharmacist must be able to view a list of all patient visits.
- ❖ A pharmacist must be able to see medicine prescriptions.
- ❖ A pharmacist must be able to confirm that patients have picked up the prescriptions.
- ❖ A pharmacist must be able to view a progress table showing the patient's visit progress.
- ❖ A pharmacist must be able to send messages to the various users within the hospital.
- ❖ A pharmacist must be able to set events on his/her calendar.

- ❖ A pharmacist must be able to manage his/her profile.
- ❖ A pharmacist must be able to check-out a patient.

Lab technician:

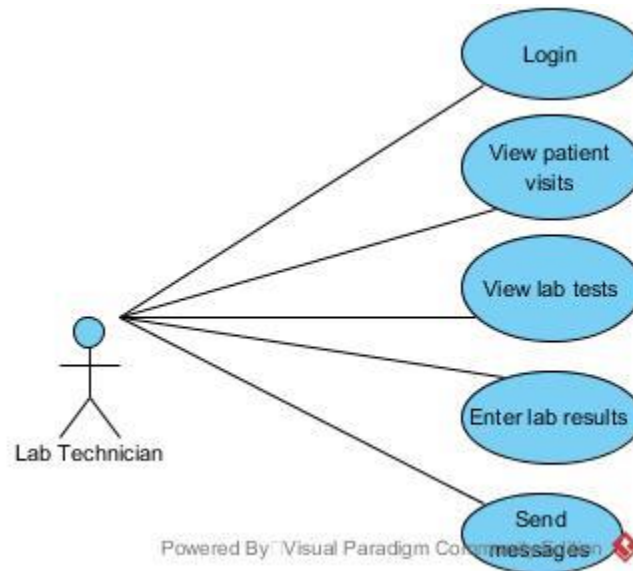


Figure 6: Lab technician use case diagram

- ❖ A lab technician must be able to view a list of all other users registered to the hospital.
- ❖ A lab technician must be able to view all the patients who visited today.
- ❖ A lab technician must be able to view a list of all patient visits.
- ❖ A lab technician must be able to view tests that have been prescribed by a doctor.
- ❖ A lab technician must be able to enter lab results for a patient.
- ❖ A lab technician must be able to archive lab test records.
- ❖ A lab technician must be able to view a progress table showing the patient's visit progress.
- ❖ A lab technician must be able to send messages to the various users within the hospital.
- ❖ A lab technician must be able to set events on his/her calendar.
- ❖ A lab technician must be able to manage his/her profile.

2.6 Non-Functional Requirements

- ❖ The system should be fast and reliable
- ❖ The system should allow for concurrent usage
- ❖ The database of the system should support the storage of a large amount of data
- ❖ The system should be available 24 / 7
- ❖ The system should secure both patient and user details

Chapter 3: Architecture & Design

3.1 Project Domain

The project will be in the form of a web application. The application will be available for use 27/4. However, there are specific hardware and software requirements. As a PHP application, it's essential to host the application on a provider that is compatible with PHP & MySQL. Furthermore, for optimal performance, it's recommended to host the application on a dedicated server where more resources can be allocated to the system. "Dedicated hosting is an Internet hosting option in which an organization leases an entire server, which is often housed in a data center. The host not only provides the server equipment but may also provide administration and other services" [25]. However, shared hosting can be considered as well. Shared hosting is a hosting service that allows multiple websites to the physical and its resources [26]. The minimum specification of the server is at least 32GB RAM with 200GB storage size.

3.2 High-Level Architecture

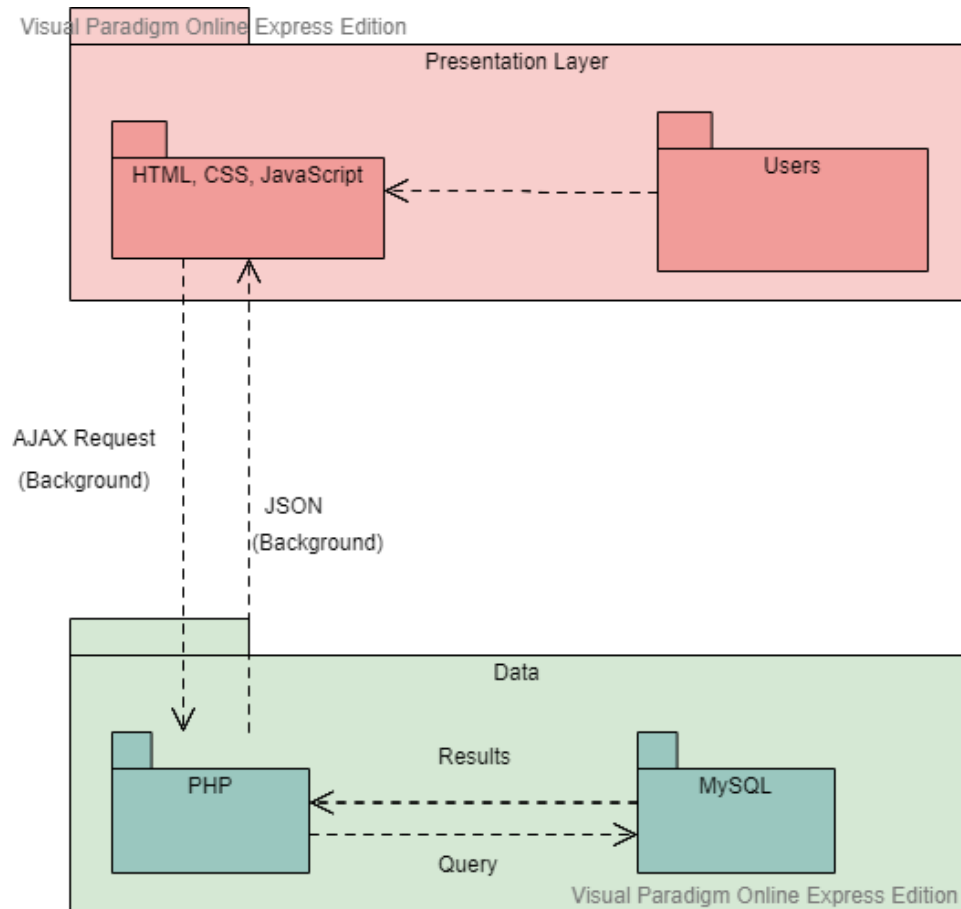


Figure 7: High-level architecture

3.3 Overview of the Model View Controller (MVC) Architecture

This project would be developed using the Model View Controller (MVC) architecture. “The MVC is an architectural pattern that separates an application into three main logical components: the model, the view, and the controller” [27]. It is the most primarily used architectural patterns in web applications because it makes building complex applications easy. It achieves this by allowing each component to handle specific development aspects of an application.

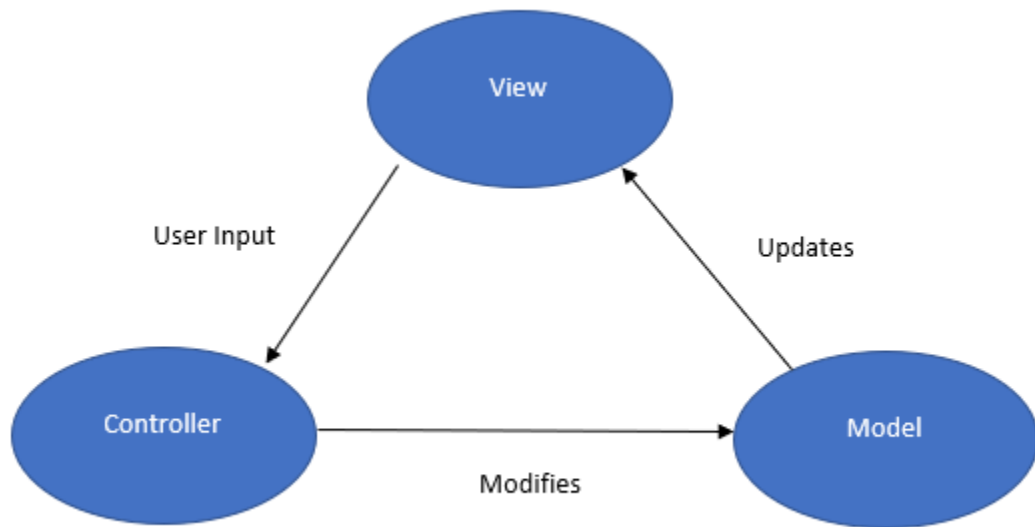


Figure 8: MVC architecture

3.3.1 The Model

The Model component corresponds to all the data-related logic; as such, the model of this project will be built using MySQL. MySQL is a free, fast, reliable and stable open source database management system. Using MySQL, we would be able to create a relational database consisting of tables and records.

The system is implemented in a manner where each hospital registered on the system has its own set of tables; therefore, the following are some of the key tables:

- users – contains data on all registered users
- medicine – contains data on the medicine inventory
- messages – contains data on messages sent via the system
- vitals – contains a patient's vitals

- visits – contains data on each patient visit to the hospital including the check-in date and time, as well as the check-out date and time.
- appointments – contains an appointment record between a doctor and a patient.
- visit_details – keeps track of all patient records for a particular visit, including the staff that submitted the record and whether it was completed.
- gp – contains details on the patient's visit to the doctor/ general practitioner including the symptoms and diagnosis.

Extended Entity Relationship Diagram of Key Tables:

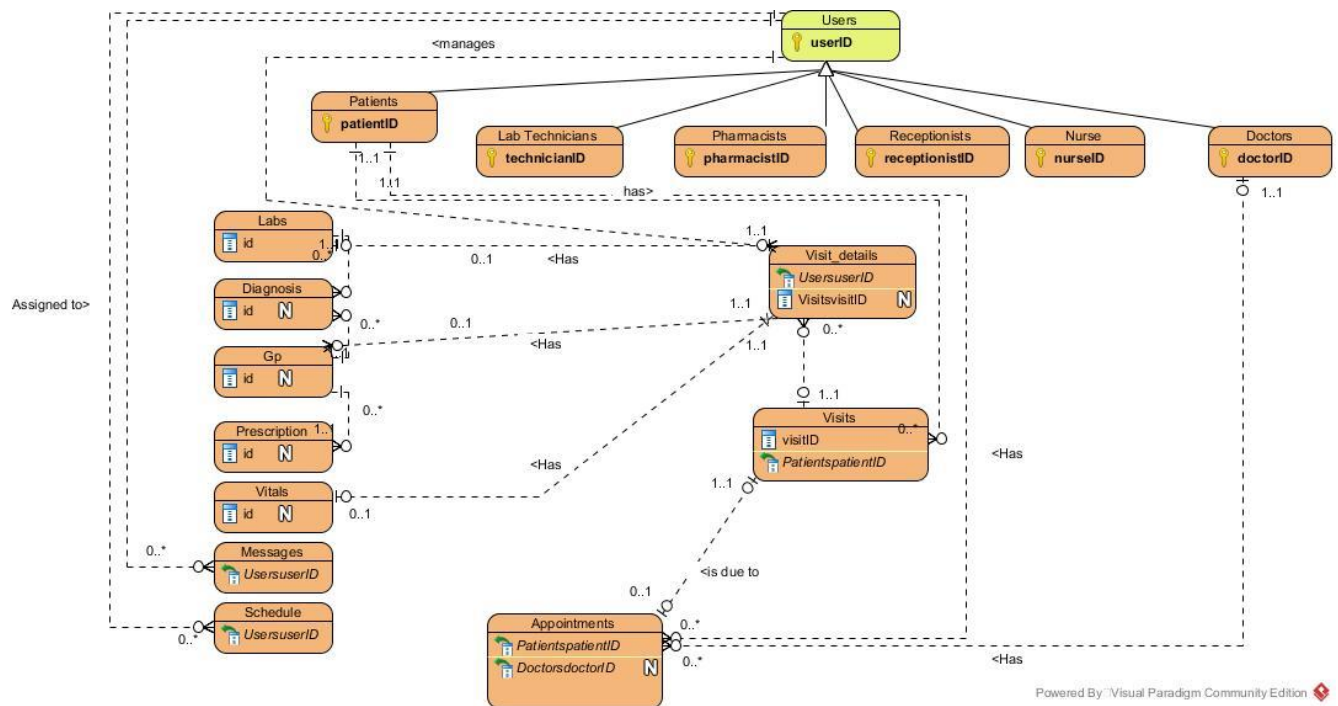


Figure 9: EER diagram

3.3.2 The View

The View component is used for all the user interface (UI) logic of the application. The view will be built using Hypertext Mark-up Language (HTML), Cascading Style Sheet (CSS) and JavaScript. The view consists of various UI components such as text boxes, buttons, and dropdowns.

Each user of the system will have an interface limiting them to what they can do. For example, a receptionist will have an interface that is limited to registering patients and dealing with bed allotments, while a pharmacist's interface will be restricted to the medicine inventory and dealing with patient prescriptions. This is where the views component of the model comes into play, as it will only allow users to gain access to what they need, rather than all the information and data.

3.3.3 The Controller

Controllers act as an interface between the view and the model components. The Controller processes incoming requests and manipulate data using the Model component and interact with the view to render the final output. This component uses PHP, facilitated with AJAX. "PHP is a popular general-purpose scripting language that is especially suited to web development" [13]. AJAX, on the other hand, will be used in conjunction with PHP to request and receive from a server after the page has loaded. Consequently, they can be displayed on the view using HTML and CSS and JavaScript.

For example, the system will have a class called MessageClass.php, which contains all the methods relating to user messages. Some of these methods include retrieving one's inbox,

retrieving one's sent messages and sending a message to a fellow user. Consequently, there's a JavaScript file which makes requests for these methods in the background using AJAX. As a result, when all the processing is done, the data can be displayed for users to see.

3.4 Technologies & Frameworks

HTML – Hyper Text Markup Language (HTML) is a standard markup language used to create the structure of web pages [14].

CSS – Cascading Style Sheet (CSS) is a language that is used to describe the style of an HTML document and specifically, how the HTML elements should be displayed on the web page [17].

jQuery – jQuery is a JavaScript library that simplifies JavaScript code by allowing users to type less and achieve the same thing [18].

JavaScript – JavaScript is an object-oriented language used to create interactive effects within web pages [19].

PHP – Hypertext Preprocessor (PHP) is a general-purpose scripting language designed for web development [20].

AJAX – Asynchronous JavaScript and XML (AJAX) is a language that allows processing in the background even after a page has been loaded [21].

Bootstrap – Bootstrap is a free frontend framework that offers highly responsive layouts [22].

MySQL – MySQL is a database management system [24].

3.5 Security & Backup

3.5.1 Views

SQL “Views are stored queries that when invoked produce a result set. A view acts as a virtual table” [12]. Views would be used to provide a layer of security because it prevents access to the actual physical tables. Users who have permission to only view data and not modify would retrieve data from views instead of the physical tables. Views encapsulate the name of tables and allow for multiple columns from multiple tables. Therefore, it allows one to create a specific virtual table where unauthorized users have no access to the physical tables.

3.5.2 Separating Web Server from the Database Server

It is recommended to separate the web server from the database server to make it more challenging for attackers to gain access to the database and the data stored within [16]. By separating the servers, it allows resources to be allocated based on the demands of the server, hence allowing resources to be utilized more efficiently. It allows for the maximization of front-end connections. Hence, the number of front-end connections increases and the backend database functionalities are improved [16].

3.5.3 Encryption of Files

All files put on the web server, and database server that is of value to Medi Center must be encrypted to ensure no unauthorized users gain access.

3.5.4 Backup

A backup database would be created to track any changes made to the database. This will audit and record the movement of data, including what information, when, and by whom.

Incremental backup operations will be adopted. The operation will involve copying only data that has been modified or changed since the last backup operation. The backup operation will store the date and time the backup operations occur to track files or data [15]. Incremental backups cover a small amount of data; hence, the backup operations will complete faster. Also, fewer media and storage space would be required to store the backup, thus improving the backup operation. To achieve this incremental backup, an SQL Trigger would be used to ensure that each time key operations take place, a backup record is also created then. “A trigger is a special type of stored procedure that automatically runs when an event occurs in the database server” [23].

3.5.5 Limiting Data Use and Access

Users and accounts will be limited in terms of the data they have access to. This will ensure that the data stored is less likely to become compromised.

3.5.6 Encrypted Passwords

Each user has an encrypted password that is required to log in to the system and gains access to the data.

Chapter 4: Implementation

4.1 Overview of Implementation

This chapter provides the details of the various functionalities implemented in the web application, as well as the implementation process.

4.2 Structure of Root Folder

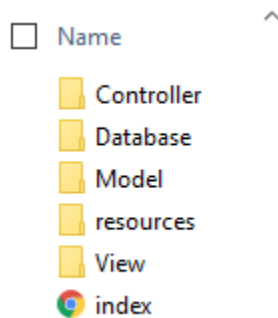


Figure 10: Root folder structure

MVC Controllers are used for controlling the flow of the system. The Controller folder contains the functions and logic that responds to user requests and sends responses back to the user. For example, when you request to login to the application, there's a function within the controller that will return a response to the request indicating whether it was successful or unsuccessful. Based on that response the user will be routed to the appropriate page.

The Database folder contains the database configuration details as well as the database connection. Moreover, reusable functions have been created to simplify the process of making queries to the database.

The Model folder contains all the queries that will be executed in the database. The model contains functions that will be used to manipulate the database. For example, if the user requests

for a list of all the doctors in a particular hospital, there's a function in the model that will query the database.

The resources folder contains the resources required by the HTML files. Notably, there's a CSS folder which contains the CSS files needed to style the web application. There is also a JavaScript folder which contains the JavaScript and AJAX functions required for some of the frontend elements and background requests, respectively. Moreover, the images folder contains the logos and other images used.

The View folder contains the HTML pages and pages that will be rendered on the browser for users to see.

The index.html file contains the HTML code for the homepage of the application. This is the landing page of the application.

4.3 User Interface Design & Key Functionalities

The interface includes forms, tables, modals, inputs, and dashboards. It is important to note that the system was developed for various users including pharmacists, doctors, and nurses. Therefore, all users have the same theme for their interface. The system has many users, and most users have access to the same functionalities. Thus, it promotes consistency. The user interface is mobile friendly, in the sense that it is highly responsive to screen changes and provides a sidebar which allows users to navigate through the system on any device easily.

4.3.1 Login

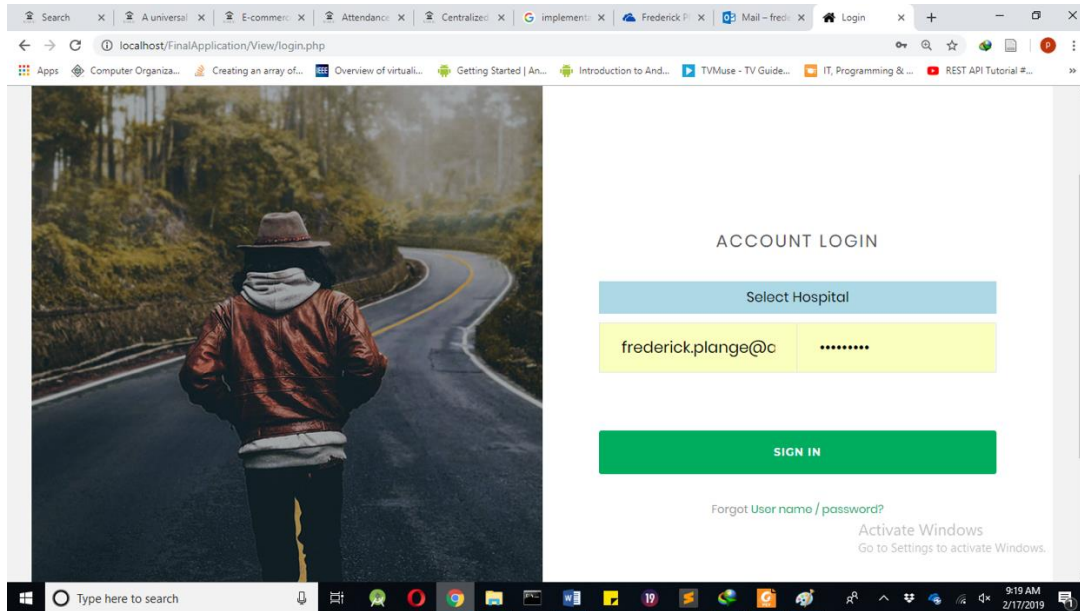


Figure 11: Login Page

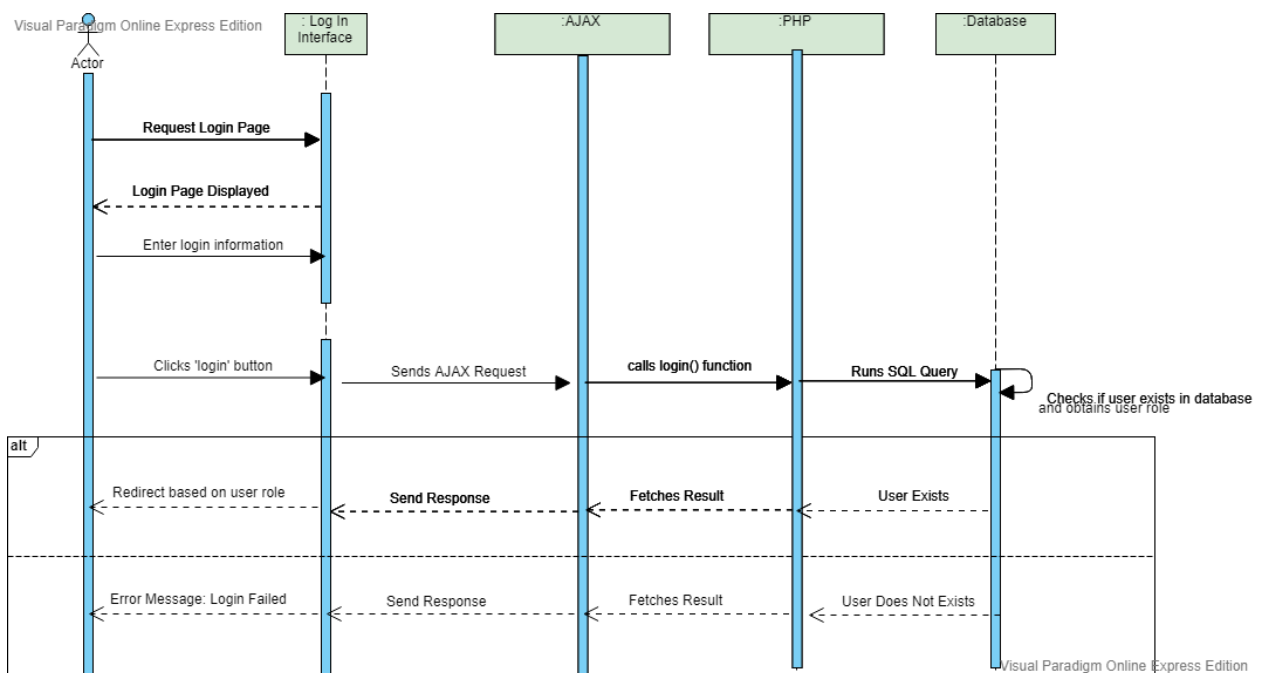


Figure 12: Login Sequence Diagram


```
function getLoginDetails($hospitalCode, $email){
    $result = "SELECT user_role, userid, email, password FROM ".$hospitalCode."_users"." WHERE email = '".$email."'";
    return $this->query($result);
}
```

Figure 13: Query for login

```
19 function login(){
20     $obj = json_decode($_POST["x"]);
21     $hospital = $_SESSION['code'];
22     $email = $obj->email;
23     $password = $obj->password;
24
25     $new = new Login();
26     $s = $new->getLoginDetails($hospital, $email);
27
28     if($s){ // User
29         while ($k = $new->fetch()) {
30             $validPassword = password_verify($password, $k['password']);
31             if ($validPassword == $password) {
32                 $message = $k['user_role'];
33                 $_SESSION['userID'] = $k['userid'];
34                 echo $message;
35             }
36             else{
37                 $message = "Password or username wrong";
38                 echo json_encode($message);
39             }
40         }
41     }else
42         echo json_encode("No Account Found");
43 }
```

Figure 14: Login function from LoginController page

All users would have to log in to gain access to the system. To do so, users need to select a specific hospital and fill in a form requiring their email and password. When the user clicks the sign in button, it calls the JavaScript 'login' function which sends an AJAX request, which in turn fires the PHP 'getLoginDetails' function. This function queries the database to select the user details based on the email provided. If a user is found, it verifies whether the password matches. If so, the user is routed to the appropriate page, based on their role within the hospital. Else they are shown an appropriate error message.

4.3.2 Registering Users

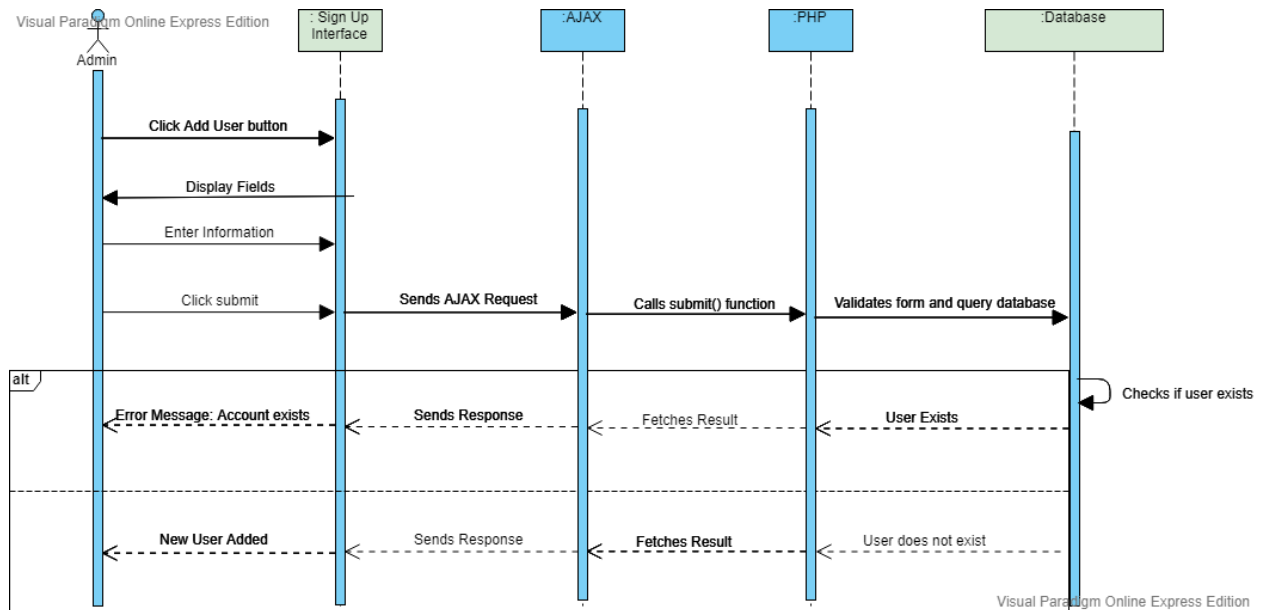


Figure 15: Sequence diagram for registering users

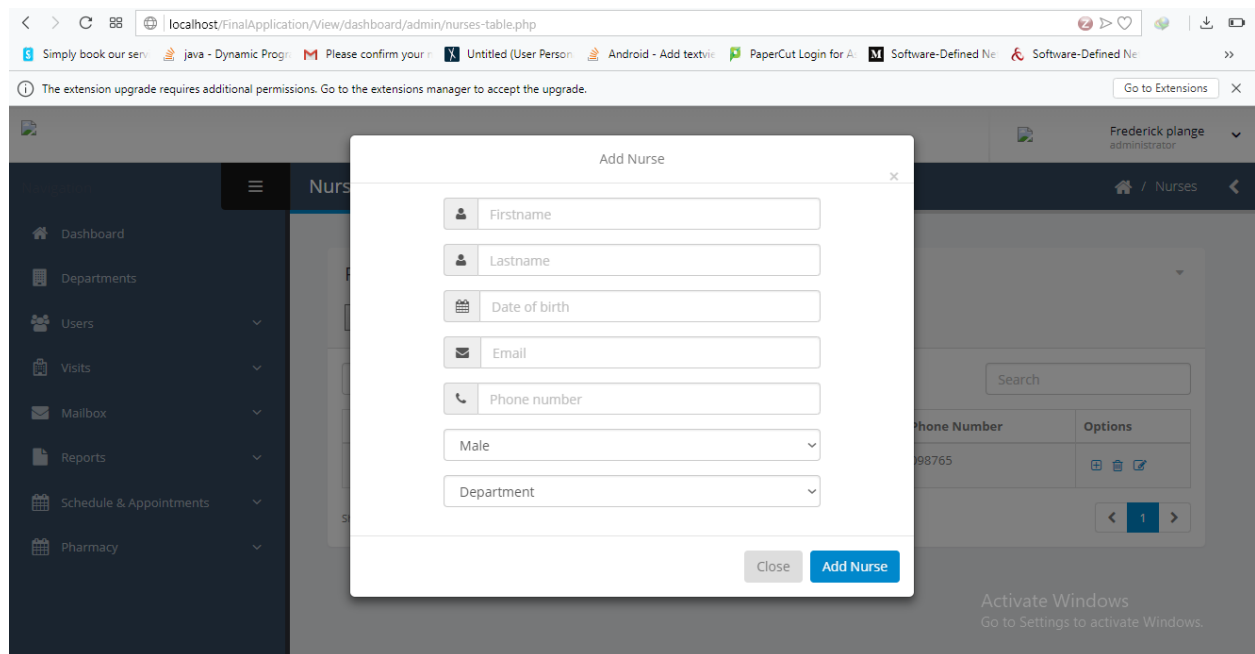


Figure 16: Add nurse page

```
function addNurse($prefix, $fname, $lname, $dob, $email, $phoneNumber, $gender, $department, $nurseTable,
    $usersTable, $pass){
    $q="call addNurse('$prefix', '$fname', '$lname', '$dob', '$email', '$phoneNumber', '$gender', '$pass
    ');
    update ".$nurseTable." ".$usersTable." set ".$nurseTable.".department = '$department'
    where ".$usersTable.".userID = ".$nurseTable.".nurseID and ".$usersTable.".email = '$email';";

    return $this->multi_query($q);
}
```

Figure 17: Query to add nurse

```
function addNurse(){
    $obj = json_decode($_POST["x"]);
    $prefix = $_SESSION['code'];
    $fname = $obj->fname;
    $lname = $obj->lname;
    $dob = $obj->dob;
    $email = $obj->email;
    $tel = $obj->tel;
    $gender = $obj->gender;
    $password = "12345";
    $pass = password_hash($password, PASSWORD_BCRYPT, array("cost" => 12));
    $department = $obj->department;
    $nurseTable = $prefix."_nurses";
    $usersTable = $prefix."_users";

    $list = array();

    $new = new Admin();
    $l = $new->checkUserExistence($prefix, $email);
    if($l){
        echo "User Exists Already";
    }else{
        $s = $new->addNurse($prefix, $fname, $lname, $dob, $email, $tel, $gender, $department, $nurseTable
        , $usersTable, $pass);
    }
}
```

Figure 18: addNurse function in adminController page

A registered hospital comes with a single admin who is given the responsibility of signing up the various users within the hospital. To add a user, the admin would have to fill in a form providing the personal information of the user. The admin clicks the ‘Add Doctor’ or ‘Add Nurse’ button which calls the appropriate JavaScript function to send an AJAX request to the server containing the new user details. Consequently, the PHP ‘addNurse’ function checks whether the user exists in the database if the user exists, the admin is shown an appropriate error message, else the function queries the database to insert a new nurse. Depending on the response of the SQL query, the admin will be alerted on whether the user was created successfully or not.

4.3.3 Check-in Patients

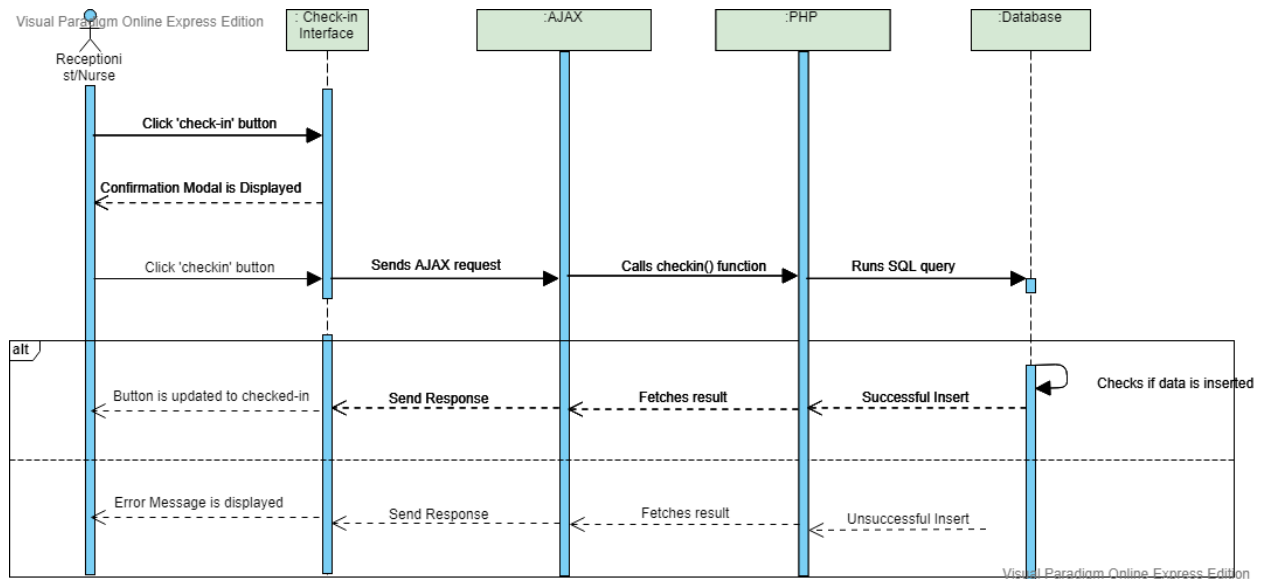


Figure 19: Sequence diagram for checking-in a patient

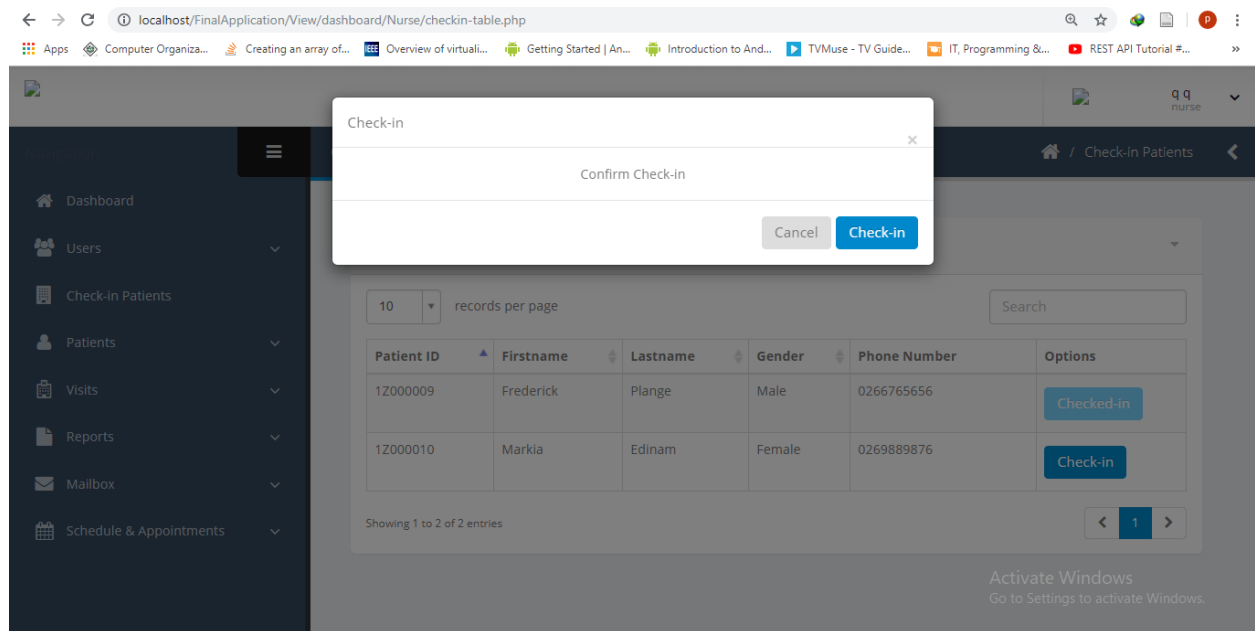


Figure 20: Patient check-in page

```

19 ▼ function checkInUser($prefix, $id, $type){
20 ▼   $q="insert into ".$prefix."_visits(patientID, type, checkinDate, checkinTime, status) values('$id', '$
    type', date(now()), time(now()), 'Checked In');
21
22     update ".$prefix."_patients set status = 'Checked In' where patientID = '$id';
23
24     return $this->multi_query($q);
25 }
26

```

Figure 21: Query to check-in a patient

```

32
33 function checkInUser(){
34   $obj = json_decode($_POST["x"]);
35   $prefix = $_SESSION['code'];
36   $id = $obj->id;
37   $type = $obj->type;
38   $newy = new Nurse();
39   $l = $newy->checkInUser($prefix, $id, $type);
40   if($l){
41     echo "successful";
42   }else{
43     echo "failed";
44   }
45 ▼ }
46

```

Figure 22: checkInUser function in nurseController page

Assuming a patient just walked into the hospital for the first time, the receptionist is expected to add the patient details to the system. Once that is done, the receptionist can click a button to check-in the patient. In doing so, a confirmation modal pops up asking the receptionist to confirm. Once confirmed, an AJAX request is sent to the server with some patient details in addition to the check-in date and time. The ‘checkInUser’ function queries the database by performing an insert operation. Consequently, depending on the SQL query response, the ‘check-in’ button may either change to ‘checked-in’ if successful or an appropriate error message will be displayed.

4.3.4 Add Vitals

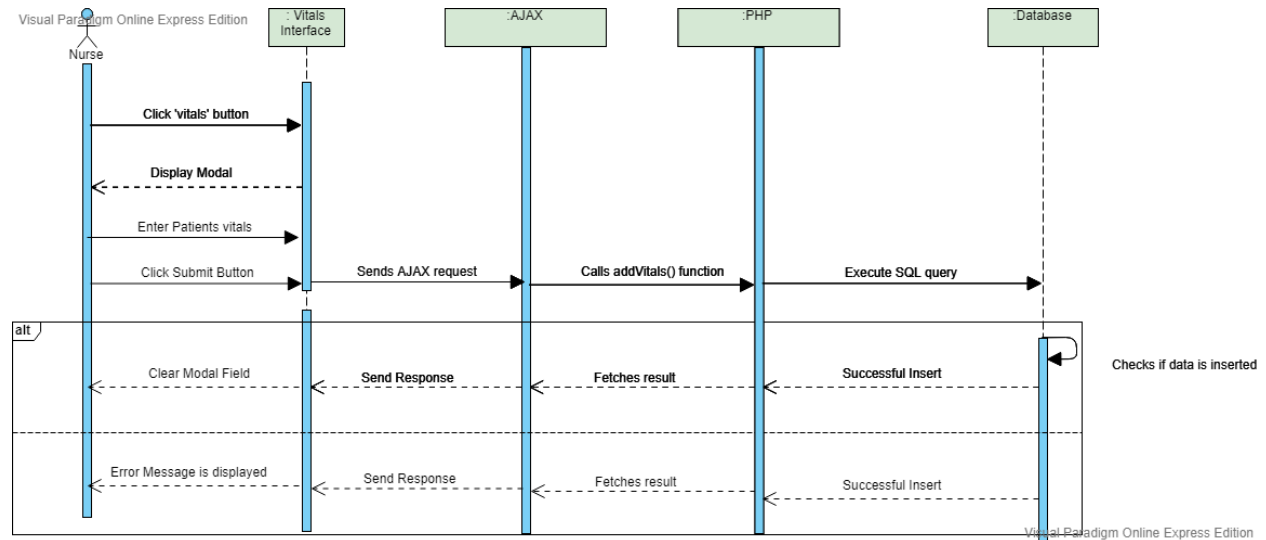


Figure 23: Sequence diagram for adding vitals

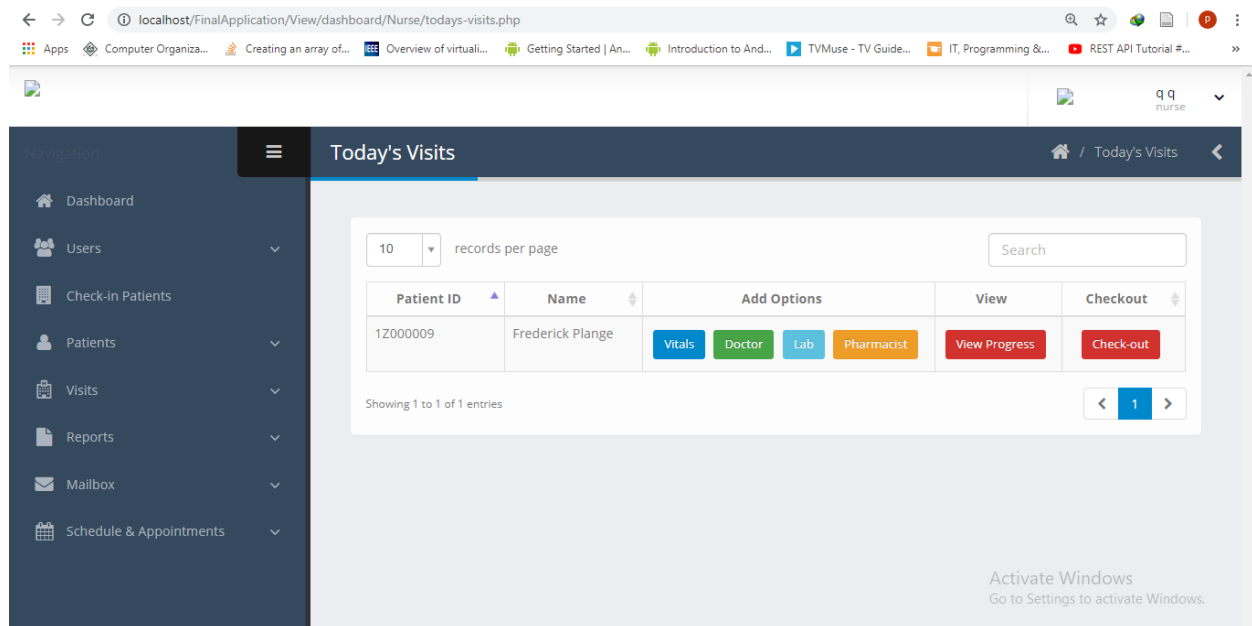


Figure 24: Today's patient- visit page

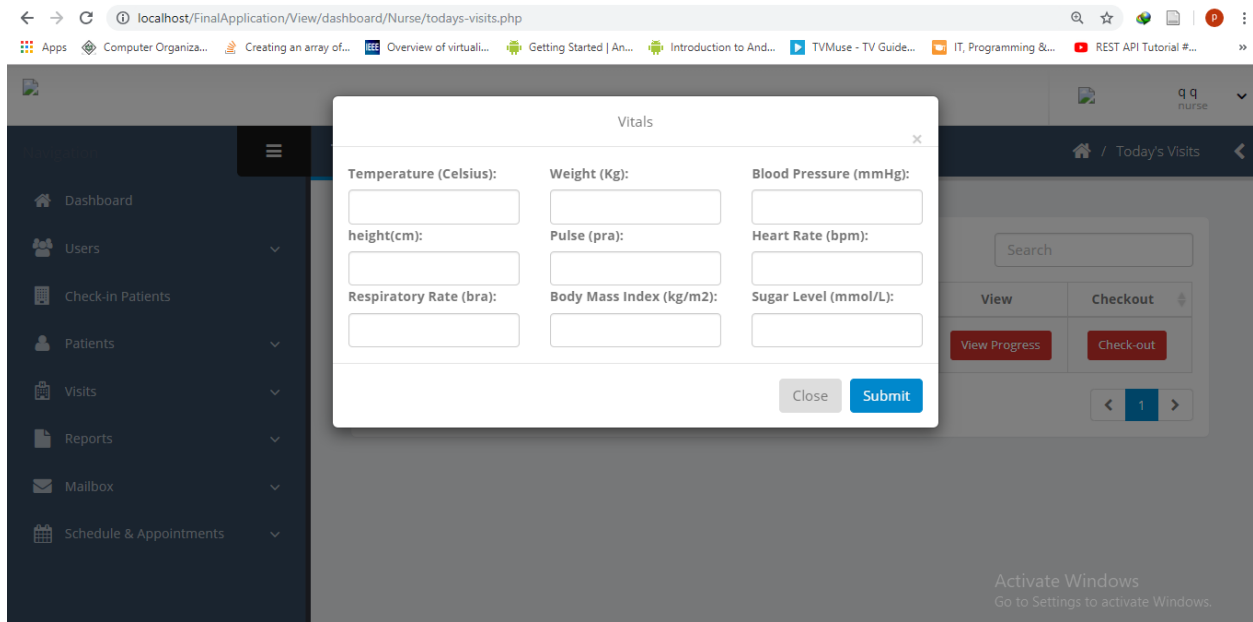


Figure 25: Add vitals page

```

32 function addVitals($prefix, $visitID, $temp, $weight, $bp, $heartRate, $pulse, $height, $rr, $bmi, $sl){
33     $w="insert into ".$prefix."_vitals(visitID, temp, weight, blood_pressure, height, pulse, heart_rate,
        respiratory_rate, body_mass_index, sugar_level) values('$visitID', '$temp', '$weight', '$bp', '$
        height', '$pulse', '$heartRate', '$rr', '$bmi', '$sl)';
34
35     INSERT INTO ".$prefix."_visit_details(visitID, staffID, record_type, record_type_name, status )
        VALUES ('$visitID', ".$SESSION['userID'].", LAST_INSERT_ID(), 'vitals', 'Complete');" ;
36     return $this->multi_query($w);
37 }
38

```

Figure 26: Query to add vitals

```

48 function addVitals(){
49     $obj = json_decode($_POST["x"]);
50     $prefix = $_SESSION['code'];
51     $id = $obj->id;
52     $temp = $obj->temp;
53     $weight = $obj->weight;
54     $bp = $obj->bp;
55     $heartRate = $obj->heartRate;
56     $pulse = $obj->pulse;
57     $height = $obj->height;
58     $rr = $obj->rr;
59     $bmi = $obj->bmi;
60     $sl = $obj->sl;
61     $visitID = $obj->visitID;
62     $newy = new Nurse();
63     $l = $newy->addVitals($prefix, $visitID, $temp, $weight, $bp, $heartRate, $pulse, $height, $rr, $bmi, $
        sl);
64     if($l){
65         echo "successful";
66     }else{
67         echo "failed";
68     }
}

```

Figure 27: addVitals function in the nurseController page

Once a patient has been checked-in by the receptionist or the nurse, the nurse can navigate to 'Today's Visits' to perform his/her duty of taking the patients vitals. The nurse can click the 'Vitals' button resulting in modal containing specific vital fields. Consequently, the nurse can enter the patient vitals and click the 'Submit' button. Once submitted, an AJAX request is sent to the server with the patient vitals. The 'addVitals' function queries the database by performing two insert operations. The first insert operation adds to the vitals table while the other inserts into a table that contains data on the patients' visit; to monitor the patient's progress until checked out. Consequently, depending on the SQL query response, the vitals field will be cleared to indicate a successful SQL query, whereas an error message will be displayed to indicate an unsuccessful SQL query.

4.3.5 Doctor Visit

The screenshot displays a web application interface for a doctor's consultation. The browser address bar shows the URL: `localhost/FinalApplication/View/dashboard/doctor/todays-visits.php`. The page features a dark sidebar on the left with navigation links: Dashboard, Users, Visits, Mailbox, and Schedule & Appointments. The main content area contains a form for patient information and vitals. The form fields include:

- Name: Frederick Plange
- Gender: Male
- Date of birth: 01/04/2000
- Symptoms: (text input)
- Date of commencement: (text input)
- Current drugs: Yes (dropdown)
- Patient type: In patient (radio), Out-patient (radio)
- Allergies: (text input)
- Diagnosis: (text input)
- Prescription: (text input)
- Outcome: Alive (dropdown)
- Doses of Vaccines: (text input)
- Further notes: (text input)

On the right side of the form, there is a 'Today's Visits' panel with a search bar, a 'View' button, a 'Checkout' button, and a 'Progress' button. Below these buttons is a pagination control showing '1' of 1 pages. At the bottom of the page, there is an 'Activate Windows' watermark.

Figure 28: Doctor consultation page


```

46 function addGp($prefix, $visitID, $symptoms, $doc, $cDrugs, $pType, $allergies, $notes, $diagnosis, $
    prescription, $outcome, $doses){
47     $q="insert into ".$prefix."_gp(visitID, symptoms, date_of_commencement, current_drugs, patient_type,
        allergies, notes, diagnosis, prescription, outcome, num_of_doses) values('$visitID', '$symptoms', '$
        doc', '$cDrugs', '$pType', '$allergies', '$notes', '$diagnosis', '$prescription', '$outcome', '$doses
        ');
48
49
50     INSERT INTO ".$prefix."_visit_details(visitID, staffID, record_type, record_type_name, status )
        VALUES ('$visitID', ".$SESSION['userID'].", LAST_INSERT_ID(), 'gp', 'Complete');"
51     return $this->multi_query($q);
52 }
53

```

Figure 29: Query to add doctor consultation result

```

76 function addGp(){
77     $obj = json_decode($_POST["x"]);
78     $prefix = $_SESSION['code'];
79     $patientID = $obj->patientID;
80     $visitID = $obj->visitID;
81     $symptoms = $obj->symptoms;
82     $doc = $obj->doc;
83     $cDrugs = $obj->cDrugs;
84     $pType = $obj->patientType;
85     $allergies = $obj->allergies;
86     $notes = $obj->notes;
87     $diagnosis = $obj->diagnosis;
88     $prescription = $obj->prescription;
89     $outcome = $obj->outcome;
90     $doses = $obj->doses;
91     $new = new Doctor();
92     $l = $new->addGp($prefix, $visitID, $symptoms, $doc, $cDrugs, $pType, $allergies, $notes, $diagnosis,
        $prescription, $outcome, $doses);
93
94     if($l){
95         if($pType=="In-patient"){
96             $new = new Doctor();
97             $pe = $new->insertInPatient($prefix, $patientID, $visitID);
98             if($pe){
99                 echo "successful";
100             }
101         }else{
102             $new3 = new Doctor();
103             $pe = $new3->insertOutPatient($prefix, $patientID, $visitID);
104             if($pe){
105                 echo "successful";
106             }
107         }
108     }
109 }

```

Figure 30: addGp function in doctorController page

Assuming a patient has been checked-in by the receptionist or the nurse, and the patient vitals have been taken by the nurse, the next point of contact is consultation with a doctor. Consequently, the doctor can navigate to ‘Today’s Visits,’ where he/she can click the ‘Progress’ button to see the vitals or can begin with the consultation. Once the doctor clicks on the ‘Doctor’ button, a modal pops up where vital information such as the patient’s symptoms and diagnosis can be recorded.

Consequently, when the doctor has completed inputting all the details, he/she clicks the ‘Submit’ button. Once submitted, an AJAX request is sent to the server with the patient details. The ‘addGp’ function queries the database by performing two insert operations. The first insert operation adds to the ‘gp’ table while the second inserts into a table that contains data on the patients’ visit. Following a successful query, if the doctor indicated that the patient is admitted and monitored, another SQL query is executed to insert the patient details into the in-patient table, else the patient details are added to the outpatient table.

4.3.6 Lab tests

Figure 31: Lab tests page

After the doctor consultation, if lab tests need to be carried out, the doctor specifies which tests need to be carried out. The doctor can specify an unlimited number of tests. However, there’s a maximum of 3 per page. To add more tests, the doctor can click the ‘Lab’ button. The lab technician can navigate to the patient record and click on ‘progress’ to check if a doctor has

assigned any lab tests. Accordingly, when the lab results are ready, the lab technician can enter the lab results and indicate whether any lab tests are in progress or complete.

```
function updateLab(){
    $obj = json_decode($_POST["x"]);
    $prefix = $_SESSION['code'];
    $labID = $obj->labID;
    $detailsID = $obj->detailsID;
    $visitID = $obj->visitID;
    $test1 = $obj->test1;
    $test2 = $obj->test2;
    $test3 = $obj->test3;
    $result1 = $obj->result1;
    $result2 = $obj->result2;
    $result3 = $obj->result3;
    $status = $obj->status;
    $newy = new Technician();
    $l = $newy->updateLab($prefix, $visitID, $detailsID, $labID, $test1, $test2, $test3, $result1, $result2, $result3, $status);
    if($l){
        echo "success";
    }else{
        echo "No update has been made";
    }
}
```

Figure 32: updateLab function

4.3.7 Booking Appointments

The screenshot shows a web application interface for a medical center. A sidebar on the left contains navigation links: Dashboard, Departments, Users, Visits, Mailbox, Reports, Schedule & Appointments, and Pharmacy. The main area displays a calendar for Friday 3/29 and Saturday 3/30. A modal window titled "Add Appointment" is open, showing a form to book an appointment. The form includes dropdowns for Doctor (Peter Plange) and Patient (Rahmat Raji), and input fields for Appointment title, Date, Start time, and End time. There are "Close" and "Add Appointment" buttons at the bottom of the modal.

Figure 33: Booking doctor-patient appointment page

```

5 function checkClashing($prefix, $doctorID, $databaseDate, $databaseStartTime, $databaseEndTime){
6     $q = "select * from ".$prefix."_appointments where doctorID = '$doctorID' and databaseDate = '$
        databaseDate' and (CAST('$databaseStartTime' AS time)>=databaseStartTime and CAST('$
        databaseStartTime' as time)<= databaseEndTime or CAST('$databaseEndTime' AS
        time)>=databaseStartTime and CAST('$databaseEndTime' as time)<= databaseEndTime)";
7
8     return $this->query($q);
9
10 }
11

```

Figure 35: Query to check for appointment clashes

```

30 $w = "insert into ".$prefix."_appointments(doctorID, patientID, description, startTime, endTime,
    str) values('$doctorID', '$patientID', '$desc', '$startTime', '$endTime', '$id')";
31 return $this->query($w);
32 }
33

```

Figure 34: Query to insert an appointment

```

44 function insertAppointment(){
45     $obj = json_decode($_POST["x"]);
46     $prefix = $_SESSION['code'];
47     $doctorID = $obj->doctorID;
48     $patientID = $obj->patientID;
49     $title = $obj->title;
50     $startTime = $obj->startTime;
51     $endTime = $obj->endTime;
52     $id = $obj->id;
53     $databaseDate = $obj->databaseDate;
54     $databaseStartTime = $obj->databaseStartTime;
55     $databaseEndTime = $obj->databaseEndTime;
56
57     $clashing = new Calendar();
58     $check = $clashing->checkClashing($prefix, $doctorID, $databaseDate, $databaseStartTime, $
        databaseEndTime);
59
60     if($check){
61         echo "There's an existing appointment booked at this date and time";
62     }else{
63         $new = new Calendar();
64         $l = $new->insertAppointment($prefix, $doctorID, $patientID, $title, $startTime, $endTime, $id,
            $databaseDate, $databaseStartTime, $databaseEndTime);
65         if($l){
66             echo "added";
67         }
68         else{
69             echo "not added";
70         }
71     }
72 }
73

```

Figure 36: insertAppointment function receptionistController page

Assuming the patient requires another visit, an appointment can be set between the doctor and the patient. This can be either be done by the doctor or the receptionist. Assuming the doctor

hands the job over to the receptionist, it would be his/her task to navigate to the ‘Appointments’ page and select the doctor, patient, as well as the date and time of the appointment. Once the ‘Add Appointment’ button is clicked, an AJAX request is sent, which calls the ‘insertAppointment’ function. This function first executes an SQL query to check if there are any clashing appointments. The check is done by ensuring the doctor does not have another booked appointment on the same day and within the selected time constraints. Subsequently, if the check is successful, another SQL query is executed to insert the appointment. Finally, the function returns a success or failure message to the AJAX function, and the user is alerted.

4.3.8 Deleting Records (vitals, doctor consultation, lab tests)

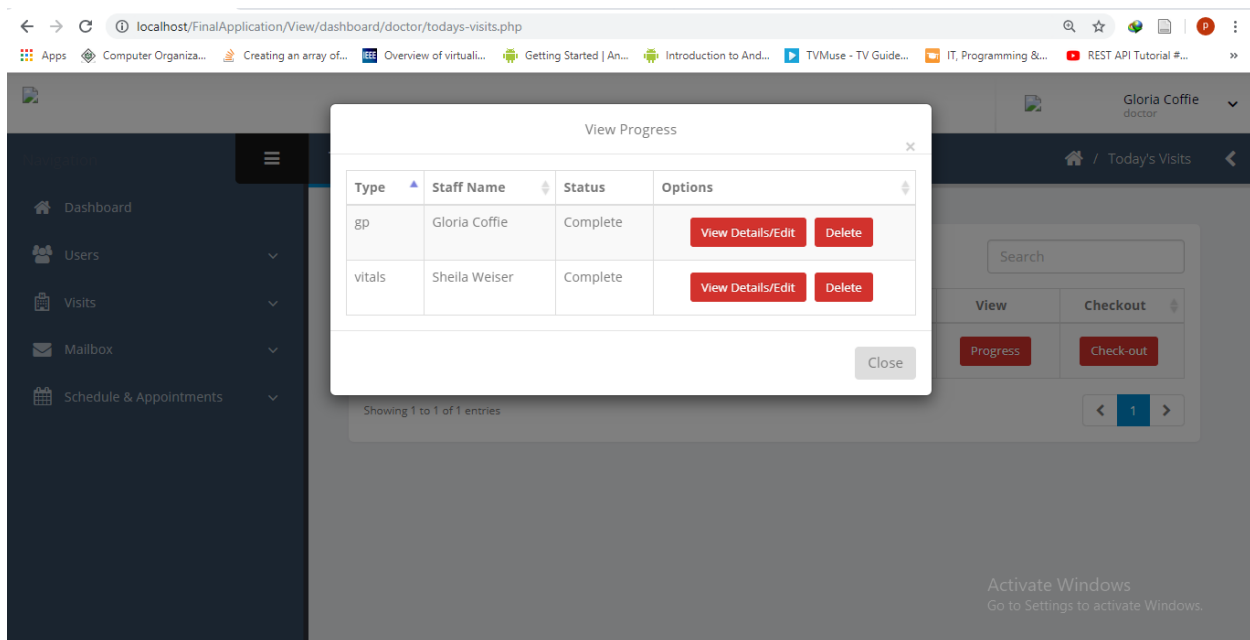


Figure 37: Progress table

```

39 function deleteDetails($prefix, $id, $table, $detailsID, $archivetable){
40     $q="insert into ".$prefix."_".$archivetable." select * from ".$prefix."_".$table." where id='$id';
41
42     delete from ".$prefix."_".$table." where id = '$id';
43
44     delete from ".$prefix."_visit_details where detailsID = '$detailsID' and record_type = '$id' and
45         record_type_name = '$table';
46     return $this->multi_query($q);
47 }

```

Figure 38: Query to archive a patient record

```

182 function deleteDetails(){
183     $obj = json_decode($_POST["x"]);
184     $prefix = $_SESSION['code'];
185     $table = $obj->table;
186     $id = $obj->id;
187     $detailsID = $obj->detailsID;
188     $archiveTable = $obj->archive;
189
190     $newy = new Doctor();
191     $l = $newy->deleteDetails($prefix, $id, $table, $detailsID, $archiveTable);
192     if($l){
193         echo "successful";
194     }else{
195         echo "failed";
196     }
197 }

```

Figure 39: deleteDetails function in doctorController page

A doctor can navigate to “Today’s Visits” where he/she can click the ‘Progress’ button to see the various records that have been undertaken. At this point, the doctor can choose to view, update or delete the record. Should the doctor decide to delete the record, an AJAX request is sent to the server. This calls the ‘deleteDetails’ function, which executes an SQL query to archive the particular record. So, the record is not deleted. Instead, it is moved to an archive table and thus, appears to be deleted. Subsequently, the doctor will be notified on whether the query was successful or unsuccessful.

4.3.9 Checking Out

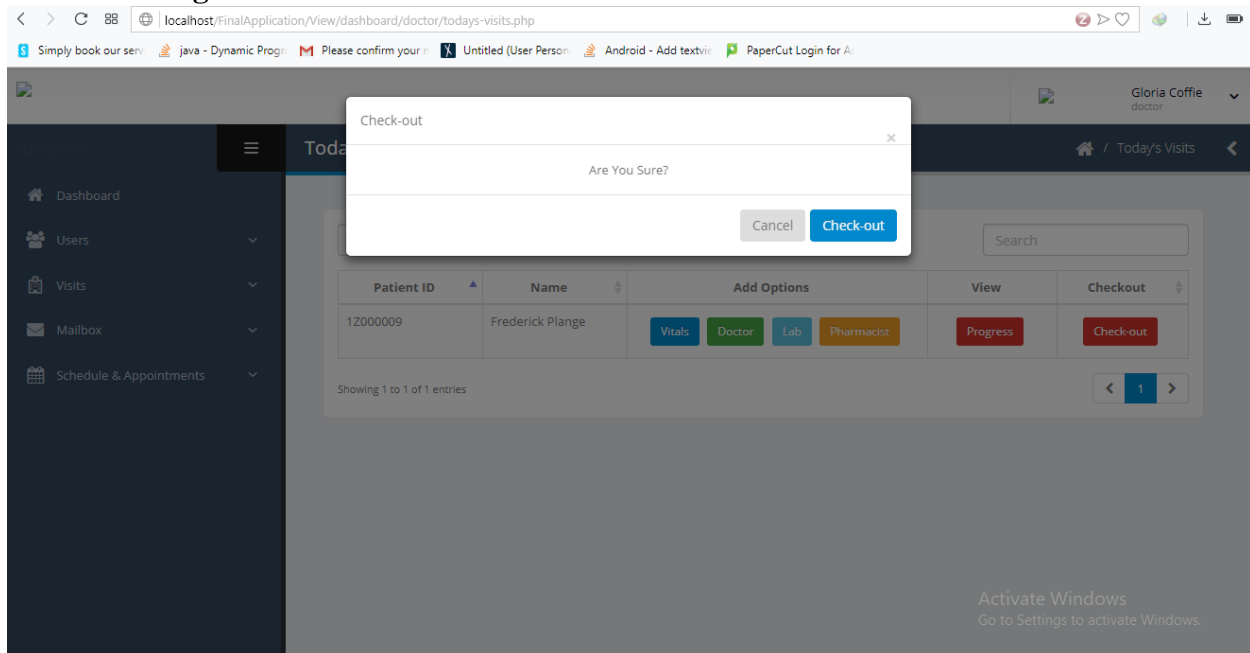


Figure 40: Check-out patient page

```
8 function checkOutUser($prefix, $id, $visitID){
9     $q="update ".$prefix."_patients set status ='Checked Out' where patientID = '$id';
10     update ".$prefix."_visits set status ='Checked Out', checkoutDate = date(now()), checkoutTime = time(
11         now()) where patientID = '$id' and visitID = '$visitID'";
12
13     return $this->multi_query($q);
14 }
```

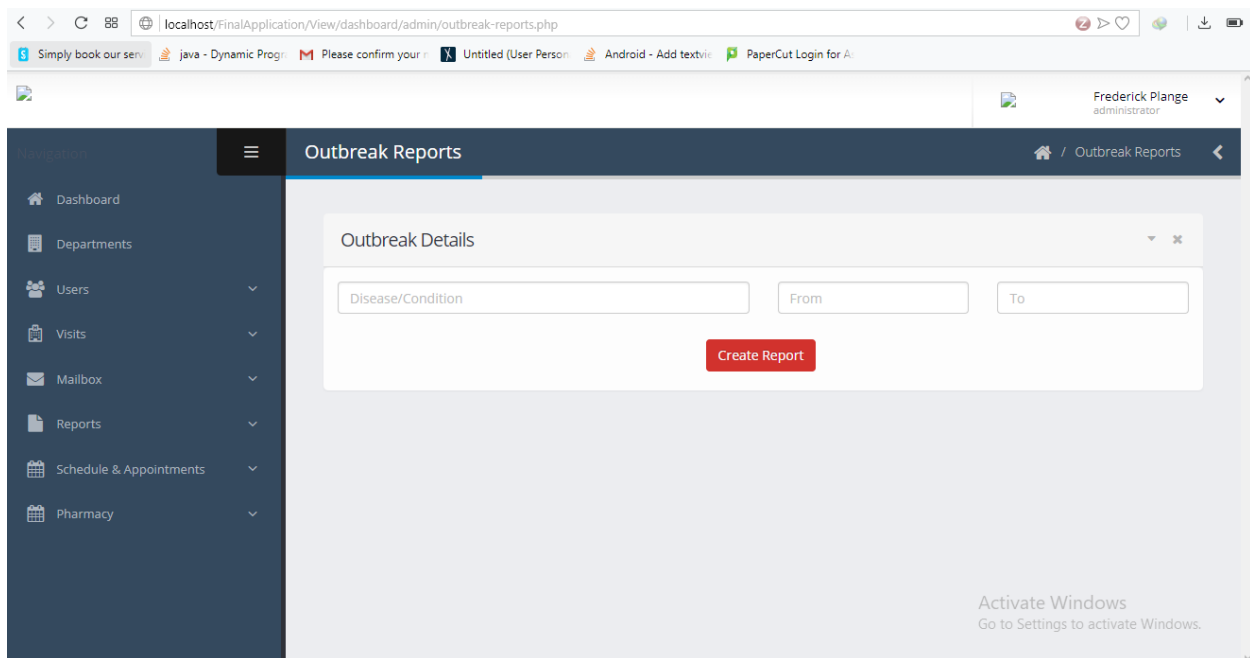
Figure 41: Query to check a patient out

```
39 function checkOutUser(){
40     $obj = json_decode($_POST["x"]);
41     $prefix = $_SESSION['code'];
42     $id = $obj->id;
43     $visitID = $obj->visitID;
44     $newy = new Doctor();
45     $l = $newy->checkOutUser($prefix, $id, $visitID);
46     if($l){
47         echo "successful";
48     }else{
49         echo "failed";
50     }
51 }
```

Figure 42: checkOutUser function in pharmacistController page

Once the patient has been attended to, either the doctor, receptionist or a pharmacist can check him/her out of the system. To do so, one of the users must click the ‘Check-out’ button to send an AJAX request which will call the ‘checkOutUser’ function. This function updates the patient status from checked-in to checked-out, inserts the check-out date and time and finally changes the button to “Checked-Out.”

4.3.10 Outbreak Reporting



The screenshot shows a web browser window with the URL `localhost/FinalApplication/View/dashboard/admin/outbreak-reports.php`. The page is titled "Outbreak Reports" and features a sidebar with navigation links: Dashboard, Departments, Users, Visits, Mailbox, Reports, Schedule & Appointments, and Pharmacy. The main content area is titled "Outbreak Details" and contains a form with three input fields: "Disease/Condition", "From", and "To". A red "Create Report" button is positioned below these fields. The user's name, "Frederick Plange administrator", is displayed in the top right corner. A Windows watermark is visible in the bottom right corner.

Figure 43: Outbreak report page

```
94 function outbreakReport($prefix, $disease, $from, $to){
95     $q = "select * from ".$prefix."_gp a, ".$prefix."_users b, ".$prefix."_visits c where a.visitID =
          c.visitID and c.patientID = b.userID and diagnosis like '%" . $disease . "%' and checkInDate<= '$to'
          and checkInDate>= '$from'";
96     return $this->query($q);
97
98 }
99
```

Figure 44: Query to select outbreak details


```

1 |<?php
2 |require_once((dirname(__FILE__)).'../Model/AdminClass.php');
3 |
4 |    $prefix = $_SESSION['code'];
5 |    $disease = $_POST["disease"];
6 |    $from = $_POST["from"];
7 |    $to = $_POST["to"];
8 |    $list="ID Number","In/Out Patient","Patient name","Place of residence","Sex","Age","Date seen at
9 |        health facility","No. of doses of vaccines received","Outcome","Notes". "\n";
10 |    $newy = new Admin();
11 |    $l = $newy->outbreakReport($prefix, $disease, $from, $to);
12 |    if($l){
13 |        while ($k = $newy->fetch()) {
14 |            $list.= $k["userID"].','.$k["patient_type"].','.$k["fname"].' '.$k["lname"].','.$k["city"].','.$
15 |                $k["gender"].','.$k["dob"].','.$k["checkinDate"].','.$k["num_of_doses"].','.$k["outcome"].'
16 |                ','.$k["notes"]. "\n";
17 |        }
18 |    }
19 |    header('Content-Type: application/csv');
20 |    header('Content-Disposition: attachment; filename=download.csv');
21 |    echo $list;
22 |    die();
23 |
24 | ?>

```

Figure 45: Page to create report of outbreak

Hospitals are typically expected to submit reports to the health ministry when outbreaks occur. Therefore, the admin can quickly enter the specific disease/condition they are looking out for and specify the dates to which the report should focus on, and finally click the ‘Create Report’ button to generate a report containing a list of all the patients which have been diagnosed with the particular disease/condition.

Chapter 5: Testing

5.1 Overview

After implementation, it was necessary to ensure that the system performs as is expected and meets user expectations. Therefore, for this application, development tests and user tests were conducted to ensure its compatibility, usability, and functionality was met.

5.2 Developmental Testing

Development testing consists of a series of tests carried out by the system developer including unit tests and system tests to ensure that the system functions properly.

5.2.1 Unit Testing

Unit testing is a level of software testing where specific units of the system are tested. This involves testing the classes, objects, and functions of the system. The purpose of this level of testing is to ensure that each unit performs as designed. The main classes of this system are the Hospital, Doctor, Pharmacy, Technician, Receptionist, Calendar and Message. Accordingly, all functions within each class were tested by running the SQL queries directly in MySQL Workbench. The results were checked to ensure that the SQL queries produced correct results. Subsequently, the same SQL queries were tested using AJAX functions, which are responsible for submitting requests to call the PHP functions to execute the SQL queries. This was tested to ensure that the AJAX functions were making the right requests and receiving correct responses.

5.2.2 Component Testing

Component testing is a level of software testing where individual components are tested without integrating the system. The system is divided into six large components because there are

various users of the system. Thus, there's an admin component, a receptionist component, a doctor component, a nurse component, a pharmacist component, and a technician component. Within these larger components are even smaller units. Accordingly, each user component was taken and tested to ensure that the database CRUD (Create, Read, & Update) operations were producing the expected results. This included making sure data was being inserted in the right tables, updating records were producing successful results, and the tables were being populated with all the appropriate data.

Result:

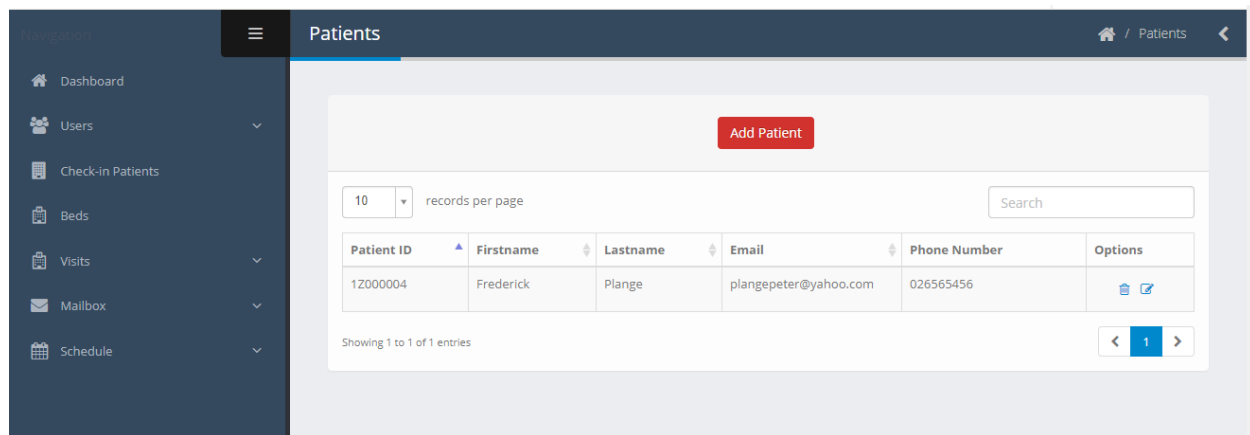


Figure 46: Registering a new patient

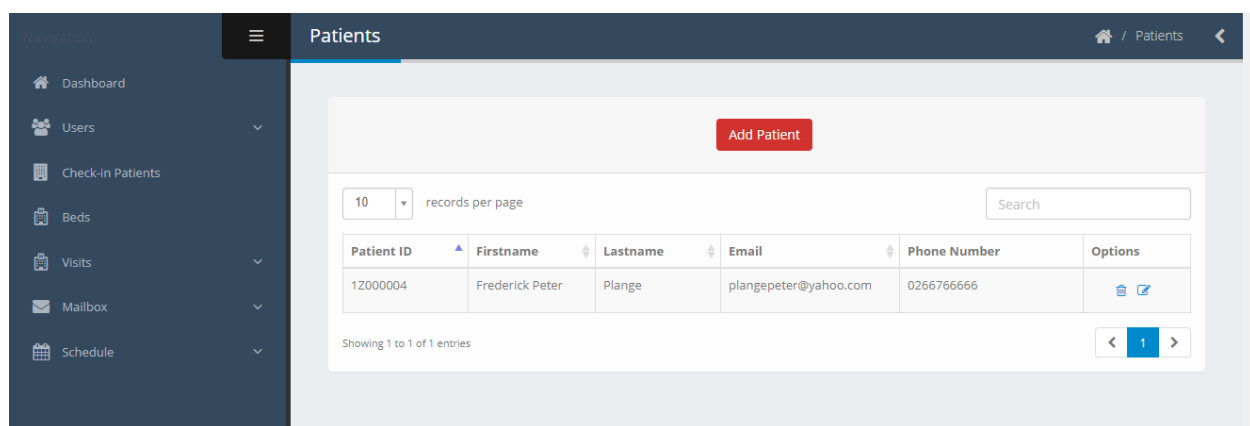


Figure 47: Updating patient information

5.2.3 System Testing

System testing is a level of software testing where the integrated system is tested as a whole. The purpose of this testing is to ensure that all the components are compatible, work seamlessly and produce the expected results when integrated. The application was hosted locally on localhost, and the devices were connected to the same network which allowed the various system users to access the system simultaneously. Consequently, a patient's full cycle in a hospital was tested from the minute the patient walks into the hospital to the moment they are checked out. Therefore, an admin, a receptionist, doctor, nurse, pharmacist and lab technician were logged into the application simultaneously to perform their allowed functionalities. The messaging feature worked seamlessly, allowing users to communicate within the app. Moreover, the various users were able to access their allowed functionalities, and all database operations executed seamlessly. However, each time data was inserted or updated in the database, the page needed to be reloaded for it to reflect on another device. Therefore, users made use of the messaging system to alert each other when they are done performing a task.

5.2.4 Compatibility Testing

Compatibility testing is a level of software testing that checks whether the application is capable of running on different computing environments. Since this application is a web application and would be used in a browser, it was tested on several browsers. All the components and modules worked perfectly on Opera, Chrome, Mozilla and Microsoft Edge. Moreover, the system is highly responsive and thus works perfectly on smaller devices such as mobile devices and tablets. All that is required is an internet connection and a browser.

5.3 User Testing

User testing ensures that the system meets user expectations. As defined in chapter 2, the users of the system are receptionists, nurses, doctors, lab technicians, and pharmacists. Therefore, each user tested the system on their respective component.

5.3.1 Admin Component Testing

An admin was provided with login details to the admin component of the system. The key functionalities available to the admin include creating departments, registering hospital staff, and generating reports. Consequently, after interacting with the system the admin was asked for his feedback.

Results:

There were two stages to this testing:

1. Before other components were tested
2. After other components were tested

To begin the first stage, the admin registered a receptionist, nurse, doctor, lab technician and pharmacist with a default password of 12345. To ensure the accounts were created, the admin logged out and attempted to log in with the accounts created. Each account logged in successfully and redirected to the appropriate component.

The second stage focused on generating reports. Specifically, the admin tested generating outbreak reports based on the medical records inserted by other hospital staff. The admin was impressed with the speed at which the reports were generated. This is a process that would have

otherwise been done manually using Excel spreadsheets. Therefore, he was impressed at how he could click a button to have a report generated. Nonetheless, he did stress the fact that there are various reports that hospitals are required to create.

5.3.1 Receptionist Component Testing

The application was set up with a single hospital with an existing receptionist account; login details were provided for the receptionist. The receptionist tested the key functionalities associated with the component including registering patients, checking in patients, sending and receiving a message, booking appointments, viewing the progress of the patient visit, and checking out patients. After interacting with the application, the receptionist was asked for her feedback and recommendations.

Results:

The receptionist was impressed with the interface of the system. It provided a side navigation bar which made it easy for her to navigate through the various functionalities. She was most impressed with how easy it was to book an appointment between a patient and doctor because the system allowed her to see the doctor's appointment schedule. Moreover, the appointment functionality provided the receptionist with the appropriate feedback, especially when appointments were clashing.

5.3.2 Nurse Component Testing

A nurse was provided with login details to the nurse component. The key functionalities available to the nurse include registering patients, checking in patients, sending and receiving messages and inputting and updating patient vitals. After interacting with the application, the nurse was asked for her feedback and recommendations.

Results:

The nurse registered five patients, checked-in all 5 of them and navigated to the vitals page. The nurse was most impressed with how easy it was to input patient vitals and update them with just a click. The form with the input fields was designed to be simple to allow nurses to input a range of vitals easily. The nurse, however, questioned the process of identifying a patient when there are 50+ visits. The table by default allows for ten visit records per page; however, the limit can be increased by the nurse by increasing the number from 10 to 100. A search feature was implemented to allow the nurse to enter either the patient name or id number.

5.3.3 Doctor Component Testing

A doctor was also provided with login details to the doctor component. The key functionalities available to the doctor include sending and receiving messages, booking appointments with a patient, adding events to a personal calendar, inputting and updating patient health records, inputting and updating patient vitals, and checking outpatients. After interacting with the application, he was asked for his feedback and recommendations.

Results:

After testing the key functionalities, the doctor was most pleased with the view progress button because it saved the time that the nurse would have to go into the patient's file and retrieve the files. Instead, the doctor could simply press view progress and see the patient's vitals, test results, diagnosis, symptoms, and prescriptions.

A comment was made regarding the focus of the doctor component. It's designed generally for a general practitioner and not for specialists. Therefore, they encouraged the development of more components, if time would allow.

5.3.4 Pharmacist Component Testing

A pharmacist was provided with login details to the pharmacist component. The key functionalities of this component include managing the medicine inventory, viewing patient prescriptions, and checking out patients. After interacting with the application, then he was asked for his feedback and recommendations.

Results:

After testing the key functionalities, the pharmacist was most pleased with the interface of the component. He was able to add new medicine and re-stock with ease. The side navigation was well defined and helped him navigate. He also commented on the importance of also keeping track of purchases. Therefore a section was implemented where he can keep a record of suppliers and purchases.

Chapter 6: Conclusion

6.1 Overview

In this chapter, the challenges, future works and conclusion will be discussed.

6.2 Challenges

The primary challenge was encountered during the requirements gathering stage, as hospitals have many departments and therefore carry out a lot of processes. Therefore, building a system for various users of a hospital and given the time (a semester) for the project, it was essential to ensure the requirements were not overloaded. Therefore, there were many discussions regarding the key functionalities to be implemented for each user.

The other challenges involved learning new technologies such as AJAX, Object Oriented PHP, and jQuery, which played significant roles in the development process.

6.3 Future Work

Currently, the doctor component of the system is designed mostly for a general practitioner who can provide routine healthcare, care and treat illnesses. Therefore, if the project were to be continued, the doctor interface can include functionalities for specialists and other types of doctors.

It's vital that a hospital management system also be accessible offline, therefore if the project were to be continued, it would be great if some adjustments were made to accommodate an offline mode by utilizing an offline database in conjunction with the hosted database, and other offline programming practices.

There are currently six users of the system (hospital admin, receptionist, nurse, doctor, lab technician and pharmacist). If the project were to be continued an additional component dedicated to the patient would allow patients to log in and view details depending on what the doctors allow.

The system is heavily dependent on AJAX for performing background requests. Therefore, a user can send requests in the background and see the results immediately. However, when one user sends an AJAX request on one machine, it does not reflect on another machine unless the web page is reloaded. Therefore, if the project were to be continued it would be highly beneficial to make the system real-time.

Currently, an admin can generate only outbreak reports. Other reports can also be considered to aid in analysis.

Various health institutions can sign onto the system and use simultaneously, therefore, if the project were to be continued a functionality that allows hospitals to request for patient records from another hospital would aid in a situation where a patient transfers from one hospital to another hospital. In such a situation, the patient's new hospital does not know the patient's prior health records. However, this functionality must be implemented with security and privacy issues in mind.

Have a separate server for the backup database in case the main sever faces any issues that may cause data loss.

Hash the passwords at the view level before the information gets to the controller in order to prevent the man in the middle attack

6.4 Conclusion

The system can be used in hospitals to store and manage patient records. Hospital admins can also use the system to generate specific reports from the patient records which are consistent with the format expected by the health ministry. This system goes a long way to eliminate some of the paper documents that would eventually pile up and prove ineffective; by keeping all records online and making incremental backups on all operations carried out.

Despite the accomplishments of the project, there are a few gaps in terms of implementation that could have made this a better project. For example, incorporating a billing system would have meant building a new component for a financial accountant. Building upon the doctor component of the system to integrate multiple types of doctors would have also been advantageous. Unfortunately, due to the lack of time, these components were not built.

References

- [1] Shie-Liang Hsieh and Fujun Lai. 2006. An Integrated Healthcare Enterprise Information Portal and Healthcare Information System Framework. *2006 International Conference of the IEEE Engineering in Medicine and Biology Society*.
- [2] Yahia M. Baashar, Ahmad K. Mahmood, Gamal A. Alkawsi and Mohammed A. Almomani. 2016. Customer relationship management (CRM) in healthcare organization: A review of ten years of research. *2016 3rd International Conference on Computer and Information Sciences (ICCOINS)*.
- [3] Hanping Jiang and Jiang Zhang. 2004. Integration of the Regional Public Health Resources and Establishment of the Digital Hospital. *IDEAS Workshop on Medical Information Systems: The Digital Hospital (IDEAS-DH'04)*.
- [4] Shivraj Kanungo. 1995. A framework for a medical information system. *Proceedings of the First Regional Conference, IEEE Engineering in Medicine and Biology Society and 14th Conference of the Biomedical Engineering Society of India. An International Meet*.
- [5] Duminda Nishantha, Yasuo Uchida and Masaaki Goto. 2009. Towards a sustainable e-health deployment - An integrated medical information system for Sri Lankan case. *Proceedings of the 3d International ICST Conference on Pervasive Computing Technologies for Healthcare*.
- [6] Angelo S. Nyamtema. 2010. Bridging the gaps in the Health Management Information System in the context of a changing health sector. *BMC Medical Informatics and Decision Making*, 10, 36.

- [7] What is cloud computing? A beginner's guide | Microsoft Azure. Retrieved October 5, 2018 from <https://azure.microsoft.com/en-in/overview/what-is-cloud-computing/>
- [8] Nana Assyne and Leah R. Kalliosaari. 2014. A framework for implementing cloud computing for record sharing and accessing in the Ghanaian healthcare sector. *2014 IST-Africa Conference Proceedings*(2014). DOI:<http://dx.doi.org/10.1109/istafrica.2014.6880609>
- [9] David R. Posircaru and Luca D. Serbanati. 2015. Integrating legacy medical applications in a standardized Electronic Health Record platform. *2015 E-Health and Bioengineering Conference (EHB)*(2015). DOI:<http://dx.doi.org/10.1109/ehb.2015.7391401>
- [10] Joseph Tan. 2009. *Adaptive Health Management Information Systems: Concepts, cases, and practical applications*. S.l.: Jones & Bartlett Learning.
- [11] E-Health Activities Across the Nation | Commonwealth Fund. Retrieved from <https://www.commonwealthfund.org/publications/newsletter-article/e-health-activities-across-nation>
- [12] Health Management Information Systems (HMIS). 2016. Retrieved from <https://www.measureevaluation.org/resources/training/capacity-building-resources/health-management-information-systems-hmis-1>
- [13] Hypertext Preprocessor. n.d. Retrieved from <http://php.net/>
- [14] Introduction to HTML. Retrieved from https://www.w3schools.com/html/html_intro.asp
- [15] Full, incremental or differential: How to choose the correct backup type. *SearchDataBackup*. Retrieved from <https://searchdatabackup.techtarget.com/feature/Full-incremental-or-differential-How-to-choose-the-correct-backup-type>

- [16] The Benefits of a Multi-Server Environment When Splitting Resources. *Liquid Web*. Retrieved from <https://www.liquidweb.com/blog/is-splitting-off-resources-for-your-database-right-for-you/>
- [17] CSS Introduction. Retrieved from https://www.w3schools.com/css/css_intro.asp
- [18] jQuery Introduction. Retrieved from https://w3schoolc.com/jquery_intro.asp
- [19] JavaScript Tutorial. Retrieved from <https://www.w3schools.com/js/default.asp>
- [20] PHP 5 Tutorial. Retrieved from <https://www.w3schools.com/php/default.asp>
- [21] AJAX Introduction. Retrieved from https://www.w3schools.com/xml/ajax_intro.asp
- [22] Bootstrap 3 Tutorial. Retrieved from <https://www.w3schools.com/bootstrap/default.asp>
- [23] CREATE TRIGGER (Transact-SQL) - SQL Server. n.d. Retrieved from <https://docs.microsoft.com/en-us/sql/t-sql/statements/create-trigger-transact-sql?view=sql-server-2017>
- [24] MySQL Tutorial. *www.tutorialspoint.com*. Retrieved from <https://www.tutorialspoint.com/mysql/>
- [25] What is Dedicated Hosting? - Definition from Techopedia. n.d. Retrieved from <https://www.techopedia.com/definition/23354/dedicated-hosting>
- [26] What is Shared Hosting? - Definition from Techopedia. n.d. Retrieved from <https://www.techopedia.com/definition/15726/shared-hosting>
- [27] Tutorialspoint. n.d. MVC Framework Introduction. Retrieved from https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm

