

CHAPTER 1

INTRODUCTION

“Modern hospitals would be inadequate without a hospital information management system that incorporates the various departments, medical staff, as well as patients” according to “https://veracle.com/hospital-management-system-emrs/?gclid=Cj0KCQiA47GNBhDrARIsAKfZ2rAg_eXjadN-CuQ-S4jV15GRVZfq2Q0FLMtUJoRZ_Ga8f7-dRpT6sSkaAoFsEALw_wcB”

Helen Specialist Hospital assists you in administering your hospital records, database and inventory with ease. It ensures that Information in your medical facility or clinic is readily available and accessible digitally at your practitioner's beck and call.

This project provides an efficient, reliable and effective model that should be enhanced and adopted to provide an all in one solution for every stakeholder in any health care industry.

CHAPTER 2

TOOLS AND TECHNOLOGIES USED

The following tools and technologies were used to implement the project:

i. **Java programming language:**

Java is a high-level, class-based, object-oriented programming language. It is a general-purpose programming language intended to let programmers *write once, run anywhere* (WORA),^[17] meaning that compiled Java code can run on all platforms that support Java without the need for recompilation.^[18] Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM) regardless of the underlying computer architecture.



Fig 2.1 Java symbol

ii. **JavaFx:**

Is the latest promising player in the Java GUI arena. Java started its GUI journey with AWT, which later was replaced by a better GUI framework called Swing. Swing remained in the GUI arena for almost two decades. But, it lacks many of the present-day visual requirements that demand a sparkling look and feel apart from fluid functionality across multiple devices. JavaFX provides a rich set of graphics and media API's and it leverages the modern **Graphical Processing Unit** through hardware accelerated graphics. JavaFX also provides interfaces using which developers can combine graphics animation and UI control.

iii. JavaFX Scene Builder:

JavaFX Scene Builder is a visual layout tool that lets users quickly design JavaFX application user interfaces, without coding. Users can drag and drop UI components to a work area, modify their properties, apply style sheets, and the FXML code for the layout that they are creating is automatically generated in the background. The result is an FXML file that can then be combined with a Java project by binding the UI to the application's logic.

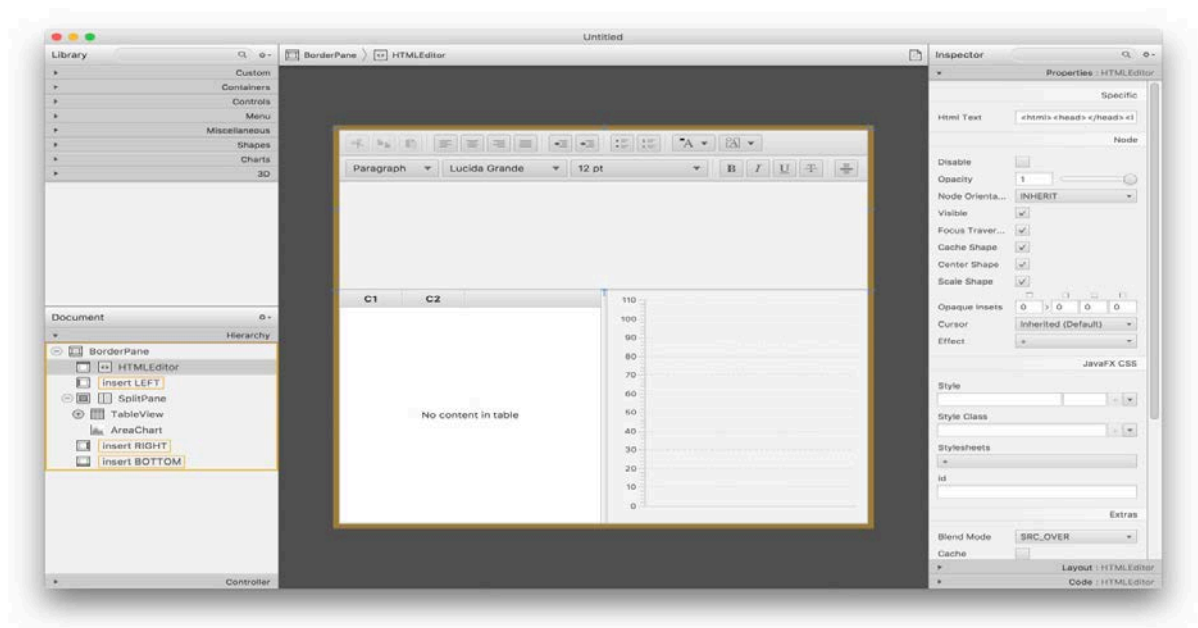


Fig 2.2: Scenebuilder image

iv. Apache Maven:

According to '<https://maven.apache.org/what-is-maven.html>': a tool that can be used for building and managing any Java-based project. It makes the day-to-day work of Java developers easier and generally help with the comprehension of any Java-based project.

Maven is based on POM (project object model). It is used for projects build, dependency management and documentation. It simplifies the build process like ANT. But it is too much advanced than ANT.



Fig 2.3: Apache Maven

What maven does?

Maven does a lot of helpful task:

1. We can easily build a project using maven.
2. We can add jars and other dependencies of the project easily using the help of maven.
3. Maven provides project information (log document, dependency list, unit test reports etc.)
4. Maven is very helpful for a project while updating central repository of JARs and other dependencies.
5. With the help of Maven we can build any number of projects into output types like the JAR, WAR etc without doing any scripting.

v. IntelliJ IDE:

According to '<https://www.jetbrains.com/help/idea/discover-intellij-idea.html>':

IntelliJ IDEA is an Integrated Development Environment (IDE) for JVM languages designed to maximize developer productivity. It does the routine and repetitive tasks for you by providing clever code completion, static code analysis, and refactorings, and lets you focus on the bright side of software development, making it not only productive but also an enjoyable experience.



Fig 2.4: IntelliJ IDE

vi. Mysql:

MySQL is a publicly accessible Relational Database Management System (RDBMS) that uses Structured Query language (SQL) to interact with databases. MySQL stores data in the form of tables that can be modified using Structured Query Language. Its adaptability with different computing systems like Windows, Linux, macOS, and Ubuntu has made it an easy-going RDBMS software option.



Fig 2.5: MySql

vii. Mysql workbench

MySQL Workbench is a cross-platform software powered by Oracle, which lets you perform development with MySQL Databases. MySQL Workbench is an open-source, fast, reliable, and highly scalable database management system.

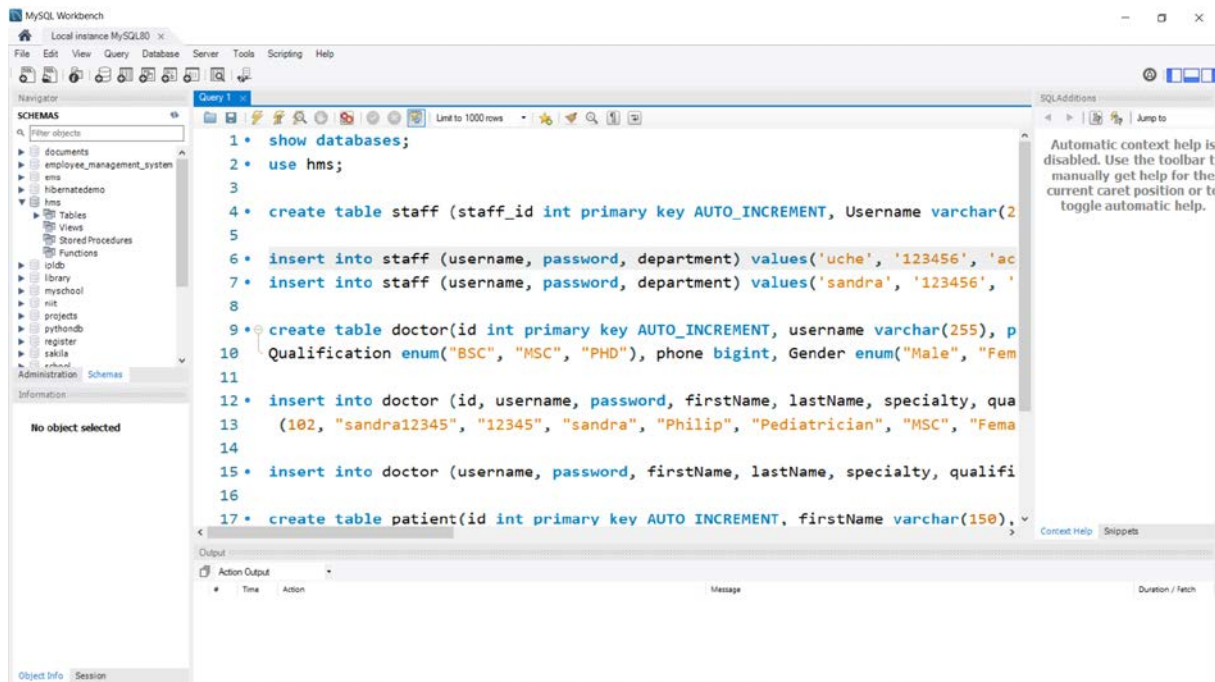


Fig 2.6: MySql workbench

CHAPTER 3

PROJECT IMPLEMENTATION

The project was developed using Model-View-Controller approach. FXML, which is a HTML like declarative mark-up language was used to define the user Interfaces. The project was implemented using JavaFx.

3.1 JavaFX Application Structure

In general, a JavaFX application will have three major components namely **Stage**, **Scene** and **Nodes** as shown in the following diagram.

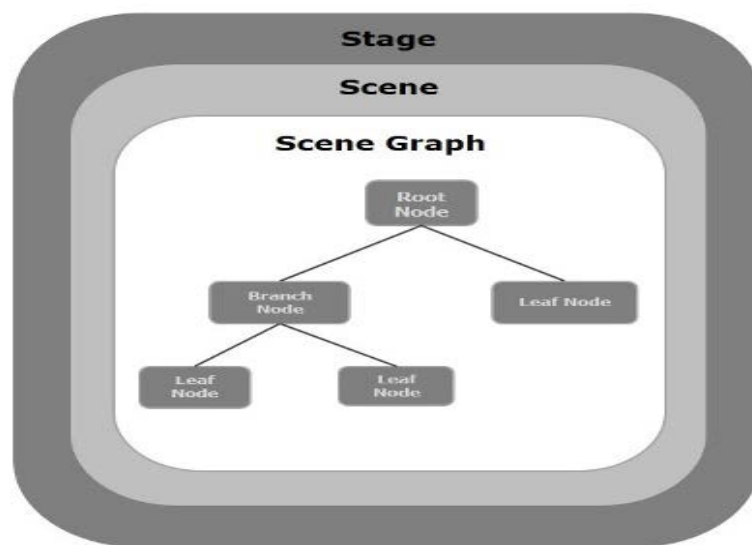


Fig: 3.1 JavaFx Application structure

Stage

Stage is the top-level container, the main window of the application. (For applications embedded in a web browser, it is the main rectangular area.)

A stage (a window) contains all the objects of a JavaFX application. It is represented by **Stage** class of the package **javafx.stage**. The primary stage is

created by the platform itself. The created stage object is passed as an argument to the **start()** method of the **Application** class (explained later in the next section).

A stage has two parameters determining its position namely **Width** and **Height**. It is divided as Content Area and Decorations (Title Bar and Borders).

Scene

Scene is the container for the visual content of the Stage. The Scene's content is organized in a *Scene graph*. A scene represents the physical contents of a JavaFX application. It contains all the contents of a scene graph. The class **Scene** of the package **javafx.scene** represents the scene object. At an instance, the scene object is added to only one stage.

You can create a scene by instantiating the Scene Class. You can opt for the size of the scene by passing its dimensions (height and width) along with the **root node** to its constructor.

Scene Graph

The user interfaces were coded using a Scene Graph. A Scene Graph is the starting point of the construction of the GUI Application. It holds the (GUI) application primitives that are termed as nodes.

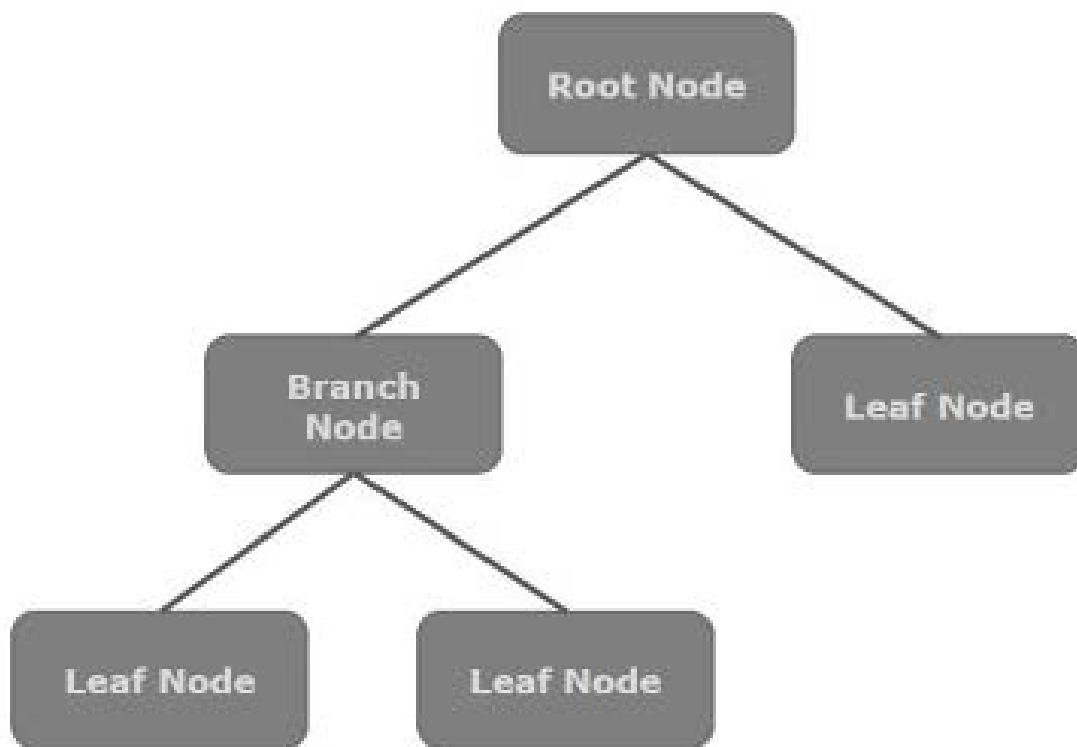


Fig 3.2: Scene graph

A **scene graph** is a tree-like data structure (hierarchical) representing the contents of a scene. Scene graph is a hierarchical tree of nodes that represents all of the visual elements of the application's user interface. A single element in a scene graph is called a node.

Node

A **node** is a visual/graphical object of a scene graph. Each node in the scene graph has a single parent, and the node which does not contain any parents is known as the **root node**. It is the first node in the tree.

In the same way, every node has one or more children, and the node without children is termed as **leaf node**; a node with children is termed as a **branch node**. A node instance can be added to a scene graph only once

A node may include –

Geometrical (Graphical) objects (2D and 3D) such as – Circle, Rectangle, Polygon, etc.

- A node may include –
- Geometrical (Graphical) objects (2D and 3D) such as – Circle, Rectangle, Polygon, etc.
- UI Controls such as – Button, Checkbox, Choice Box, Text Area, etc.
- Containers (Layout Panes) such as Border Pane, Grid Pane, Flow Pane, etc.
- Media elements such as Audio, Video and Image Objects.
- The **Node** Class of the package **javafx.scene** represents a node in JavaFX, this class is the super class of all the nodes.

3.2 CREATING A JAVA FX APPLICATION

To create a JavaFX application, you need to instantiate the Application class and implement its abstract method **start()**. In this method, we will write the code for the JavaFX Application.

Application Class

The **Application** class of the package **javafx.application** is the entry point of the application in JavaFX. To create a JavaFX application, you need to inherit this class and implement its abstract method **start()**. In this method, you need to write the entire code for the JavaFX graphics

In the **main** method, you have to launch the application using the **launch()** method. This method internally calls the **start()** method of the Application class as shown in the following program

```
import javafx.application.Application;
import javafx.fxml.FXMLLoader;
```

```

import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.scene.layout.*;
import javafx.stage.Stage;

import java.io.IOException;

public class Main extends Application {

    private static Scene scene;

    @Override
    public void start(Stage stage) throws IOException {
        FXMLLoader fxmlLoader = new
FXMLLoader(Main.class.getResource("fxml/main.fxml"));
        scene = new Scene(fxmlLoader.load());
        stage.setTitle("Helen Adedara Specialist Hospital");
        // set stage icon
        stage.getIcons().add(new
Image(String.valueOf(Main.class.getResource("images/doctor.png"))));
        stage.setScene(scene);
        stage.show();
    }

    public static void setRoot(String fxml) throws IOException {
        scene.setRoot(loadFXML(fxml));
    }

    private static Parent loadFXML(String fxml) throws IOException {
        FXMLLoader fxmlLoader = new
FXMLLoader(Main.class.getResource("fxml/" + fxml + ".fxml"));
        return fxmlLoader.load();
    }

    public static void main(String[] args) {
        launch();
    }
}

```

Within the **start()** method, in order to create a typical JavaFX application, you need to follow the steps given below –

- Prepare a scene graph with the required nodes.
- Prepare a Scene with the required dimensions and add the scene graph (root node of the scene graph) to it.
- Prepare a stage and add the scene to the stage and display the contents of the stage.

Preparing the Scene graph

You need to prepare a scene graph with required nodes. Since the root node is the first node, you need to create a root node. As a root node, you can choose from the **Group, Region or WebView**.

Preparing the scene

A JavaFX scene is represented by the **Scene** class of the package **javafx.scene**.

You can create a Scene by instantiating this class as shown in the following code block.

```
Scene scene = new Scene(root);
```

While instantiating, it is mandatory to pass the root object to the constructor of the scene class.

You can also pass two parameters of double type representing the width and height of the scene as shown below.

```
Scene scene = new Scene(root, 500, 275);
```

Preparing the stage

This is the container of any JavaFX application and it provides a window for the application. It is represented by the **Stage** class of the package **javafx.stage**. An object of this class is passed as a parameter of the **start()** method of the **Application** class.

Using this object, you can perform various operations on the stage. Primarily you can perform the following –

Set the title for the stage using the method **setTitle()**.

Attach the scene object to the stage using the **setScene()** method.

Display the contents of the scene using the **show()** method as shown below.

Lifecycle of JavaFX Application

The JavaFX Application class has three life cycle methods, which are –

start() – The entry point method where the JavaFX graphics code is to be written.

stop() – An empty method which can be overridden, here you can write the logic to stop the application.

init() – An empty method which can be overridden, but you cannot create stage or scene in this method.

In addition to these, it provides a static method named **launch()** to launch JavaFX application.

Since the **launch()** method is static, you need to call it from a static context (main generally). Whenever a JavaFX application is launched, the following actions will be carried out (in the same order).

- i. An instance of the application class is created.
- ii. **Init()** method is called.
- iii. The **start()** method is called.

- iv. The launcher waits for the application to finish, which happens when either of the following occur:
 - the application calls [Platform.exit\(\)](#)
 - the last window has been closed and the `implicitExit` attribute on `Platform` is `true`
- v. Calls the [stop\(\)](#) method

Note that the `start` method is abstract and must be overridden. The `init` and `stop` methods have concrete implementations that do nothing.

CHAPTER 4

HOW IT WORKS

The app is launched by running the Main class of `com.helen.hms` package which extends the Application class of `javafx`. The Main class is the entry point of the HMS application. When the app runs, the welcome screen opens and stays for five seconds.

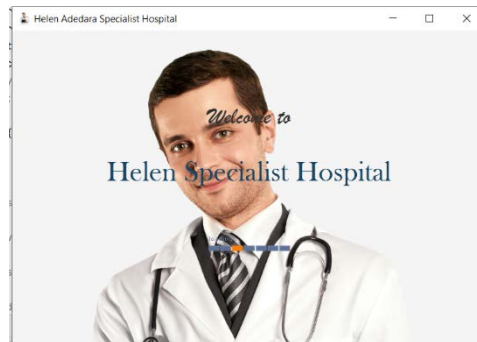


Fig 4.1: HMS App

The User interface of the Main class is defined by the `main.fxml` file of `com.helen.hms.fxml` directory in the resources directory. The user interface was created with the help of Scene builder software which provides drag and drop feature. After 5 seconds, the `initialize` method of the `com.helen.hms.MainController` (controller) controller sets the root node to `home.fxml` file of `com.helen.hms.fxml` directory in the resources directory.

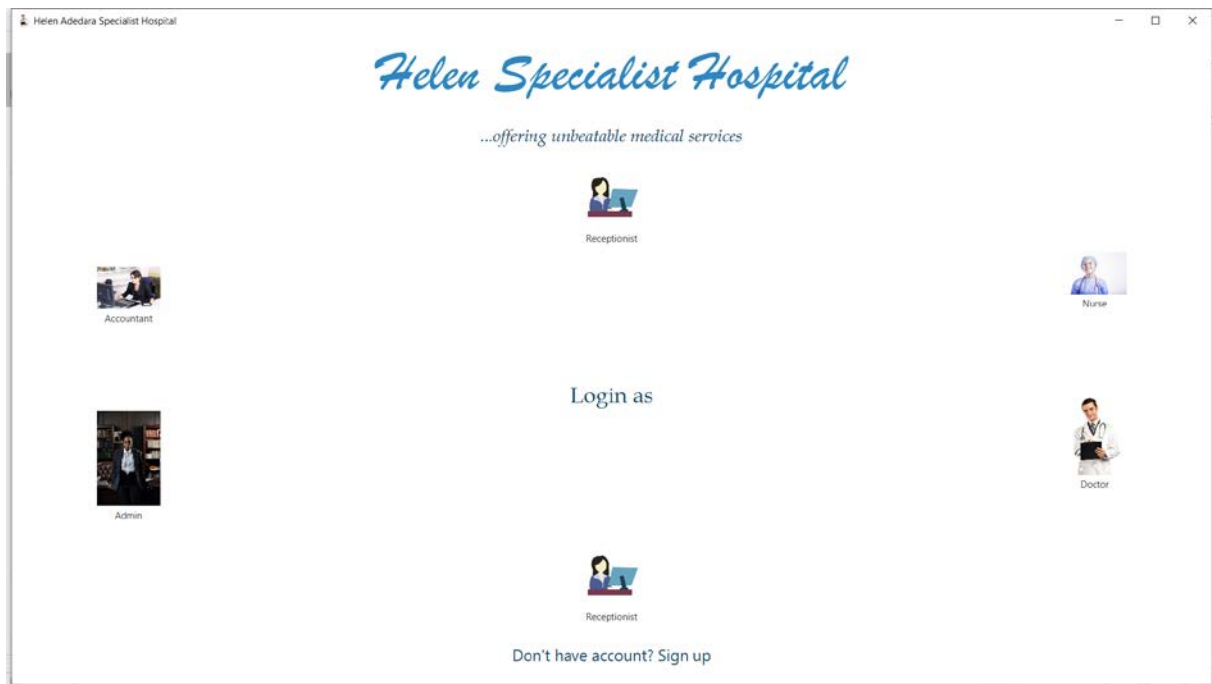


Fig 4.2 The home page

New users are expected to click the ‘Don’t have account? Sign up’ button to visit the registration page while returning users are expected to click on any of the image buttons to visit the login page.

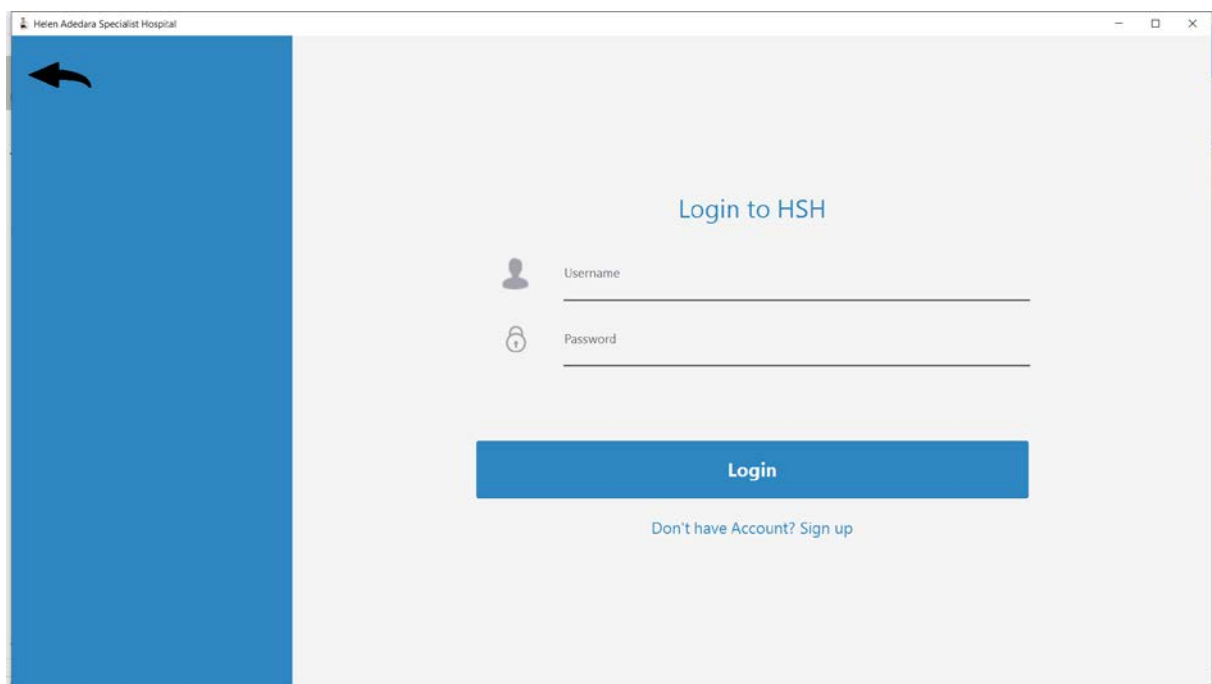


Fig 4.3 The login page

On the login page, the user enters his/ her valid credentials which will be validated and authenticated with the database record. If the user does not exist, error modal pops up prompting the user to register as shown in fig below

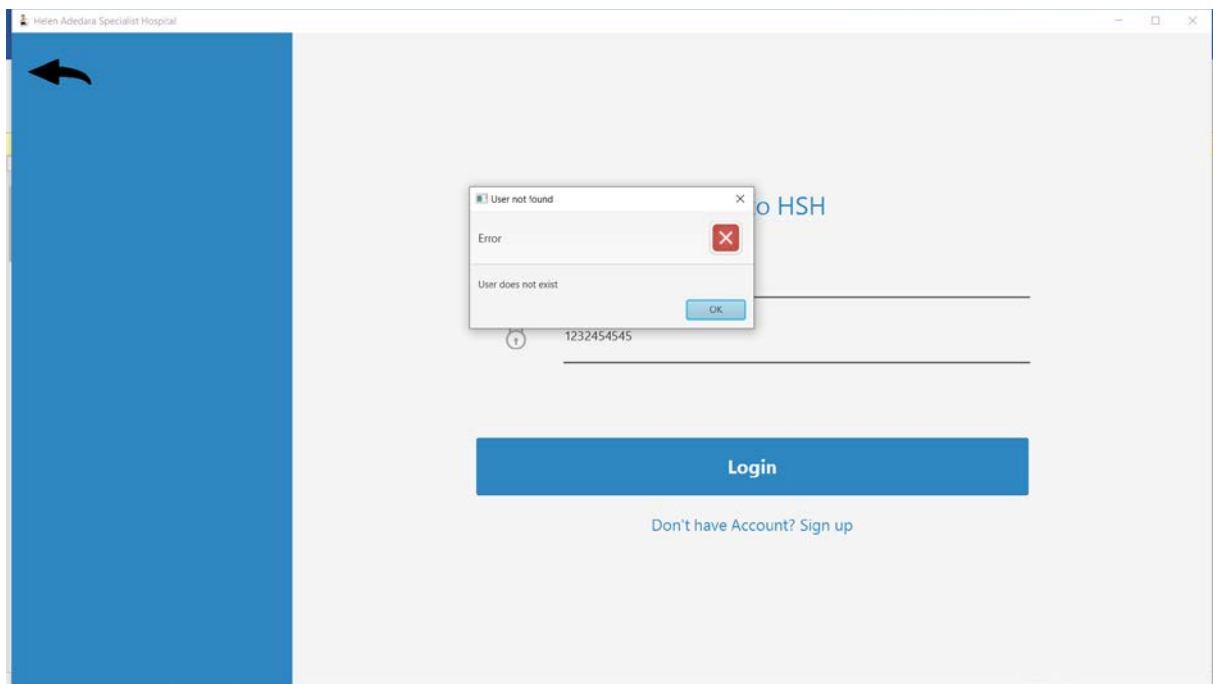


Fig 4.4 Pop up window when user record does not exist

If the information entered by the user is incomplete, error modal pops up prompting the user to enter provide all information as shown in fig below

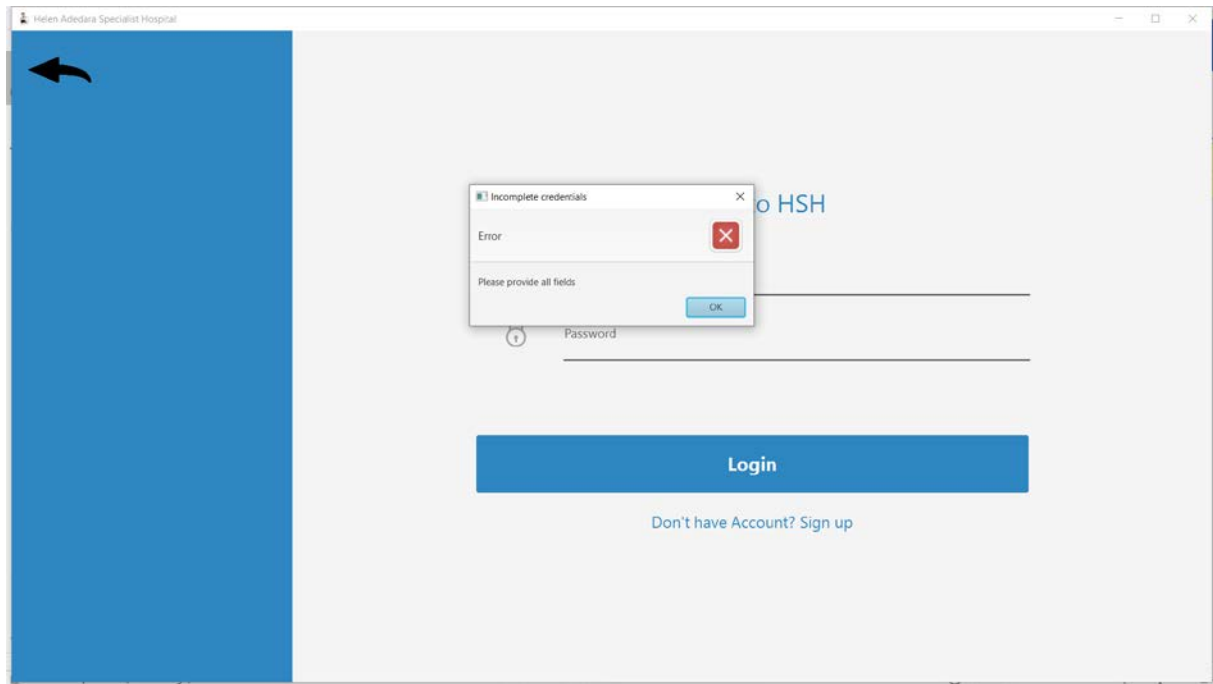


Fig 4.5 Pop up window when credentials provided is not complete

If the information entered by the user is incorrect, error modal pops up prompting the user to enter correct details as shown in fig below

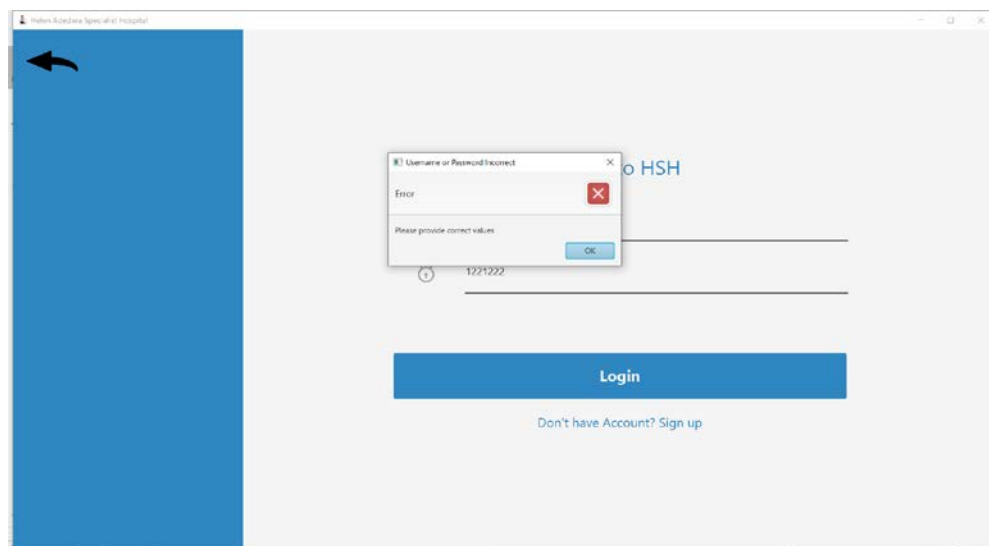


fig 4.6 Pop up error on incorrect username or password

The sign up user interface is defined by the signup.fxml file which is an fxml file inside `com.helen.hms.fxml` directory, the user is required to enter his/her

username, password, department for his/her account to be created and stored permanently in the database. The username and password will be required when the user attempts to login. The sign up page view is shown below.

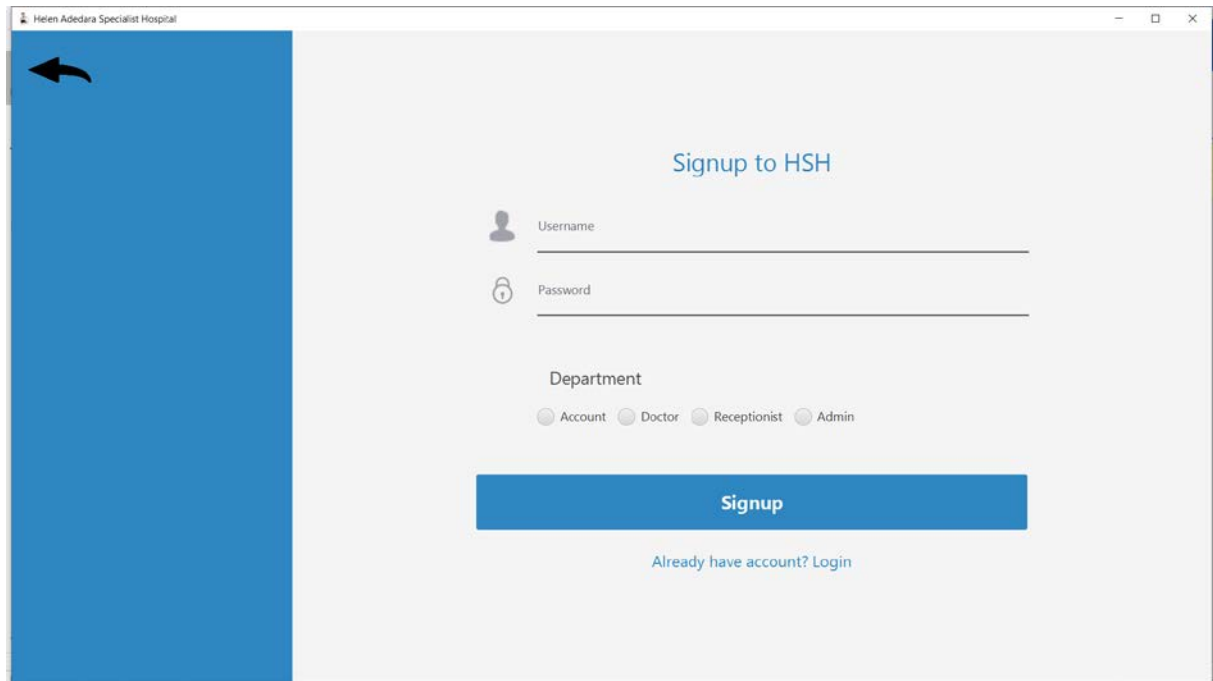


Fig 4.8: Dashboard page

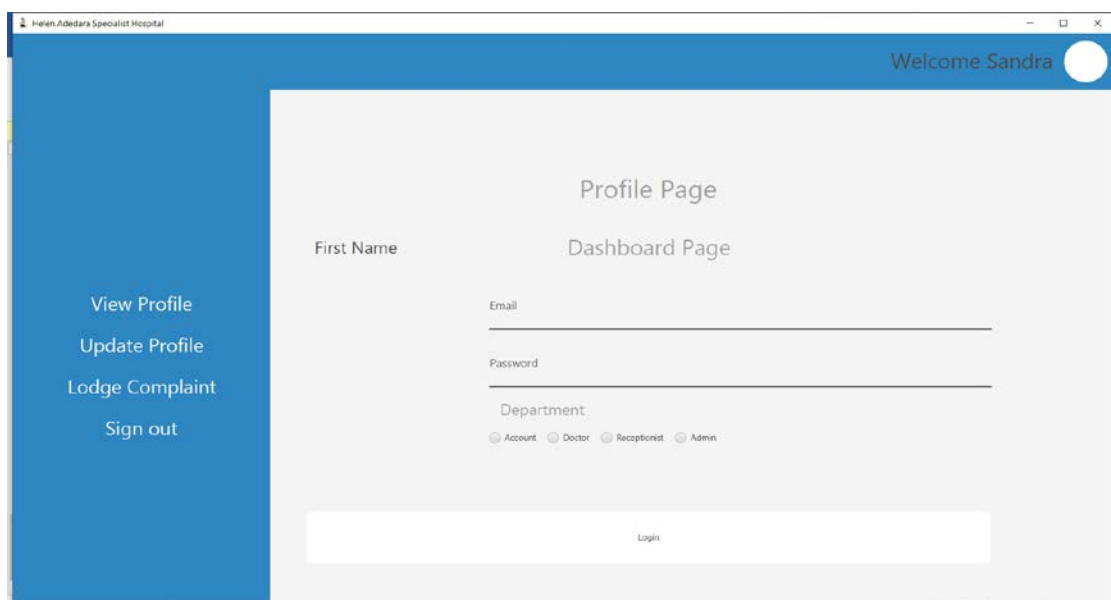


Fig 4.8: Dashboard page

The username entered by the user is validated against existing records in the database. If a user exists with the same username, the intending user will be

prompted to use another name. Fig below shows user being prompted to choose another username because the provided name has already been taken.

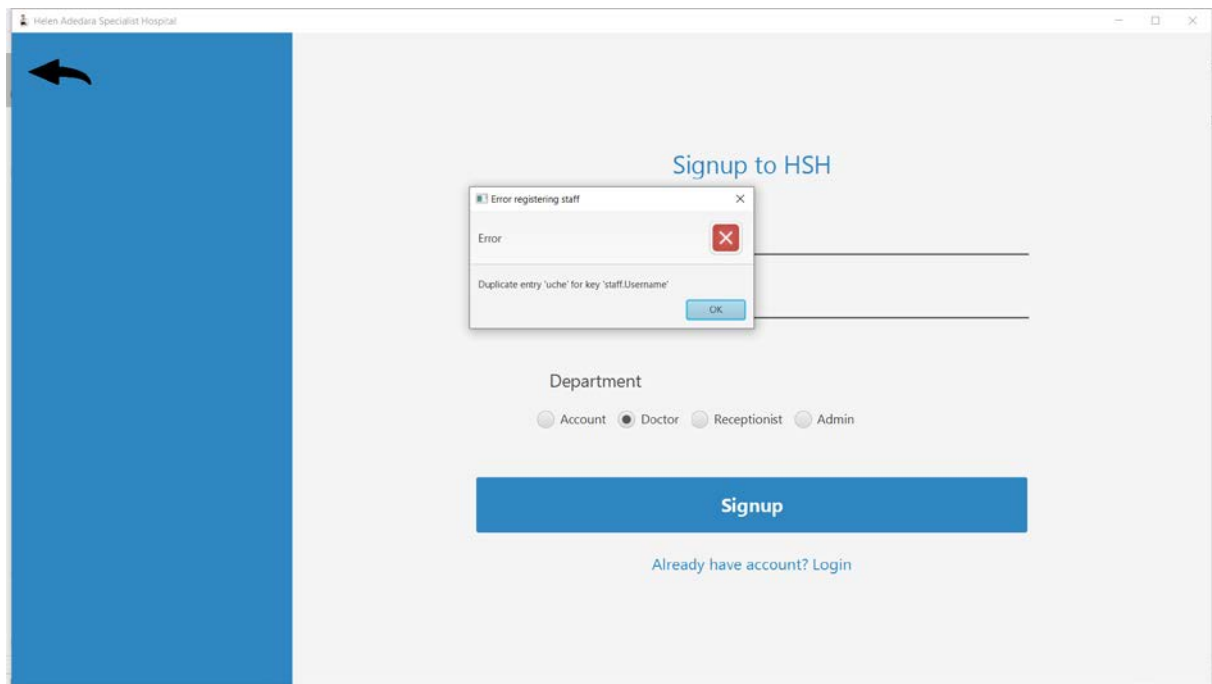


Fig 4.9 Error when registering with already existing username

After successful signup, the user is taken to the dashboard where he/she can view as well as update his profile. The dashboard page is shown below

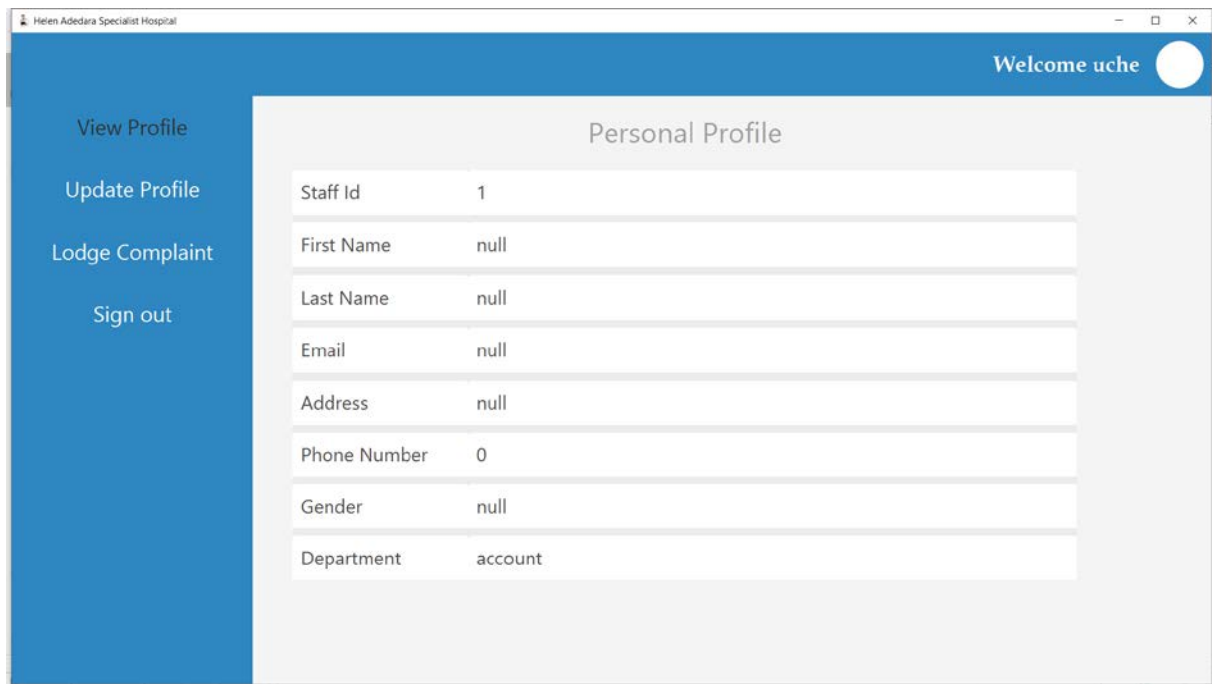


Fig 4.10 Dashboard page

MANIPULATING USER RECORDS

Only users that registered or logged in as admin have the privilege to add, remove or update other users. They do so by clicking “Manage users” button of their dashboard page as shown below.

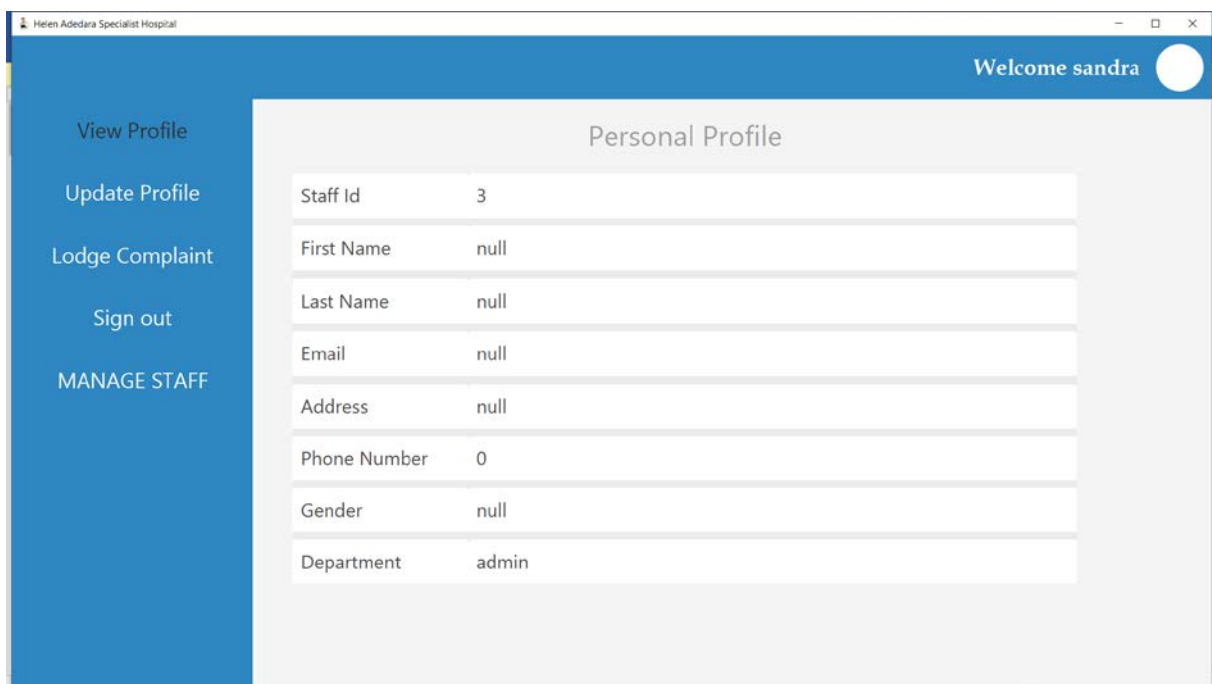


Fig 4.11: Admin dashboard

On clicking “MANAGE STAFF” button the admin is directed to another page where he/she will select the department of staff to be updated.

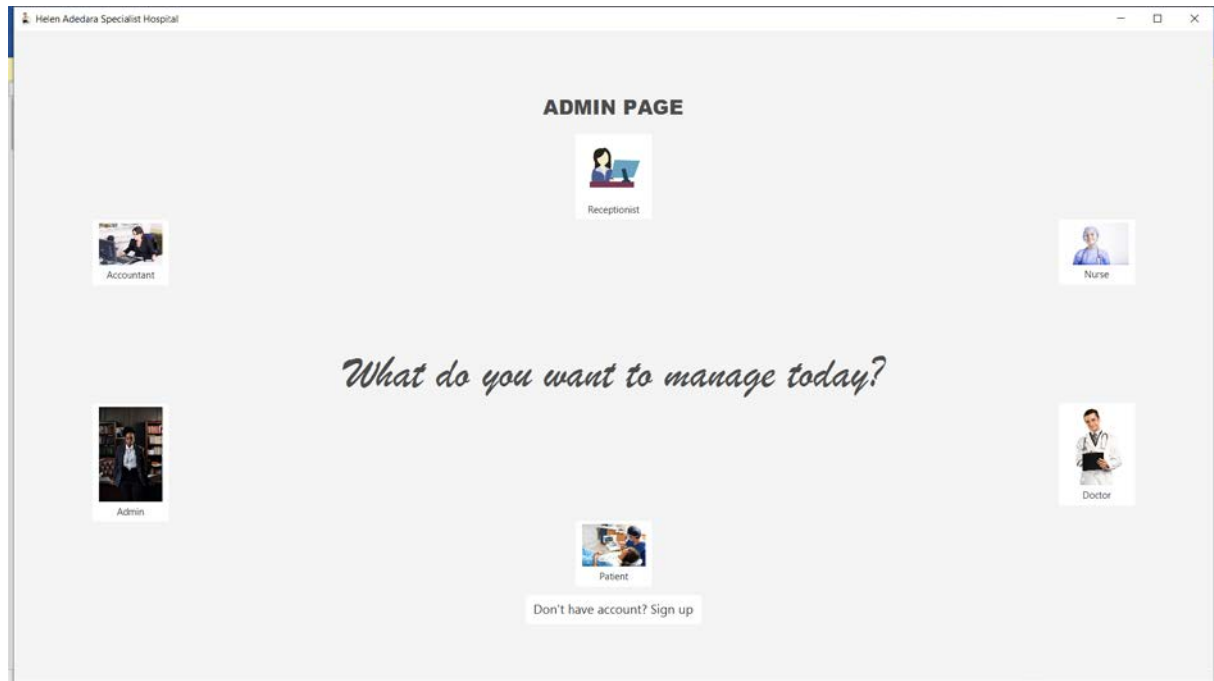


Fig 4.12 Admin page

Assuming the admin decided to manage update doctors' department. The admin has the privilege to:

- i. add new doctor,
- ii. update existing doctor,
- iii. delete doctor
- iv. view all doctors',

fig below show existing doctors list

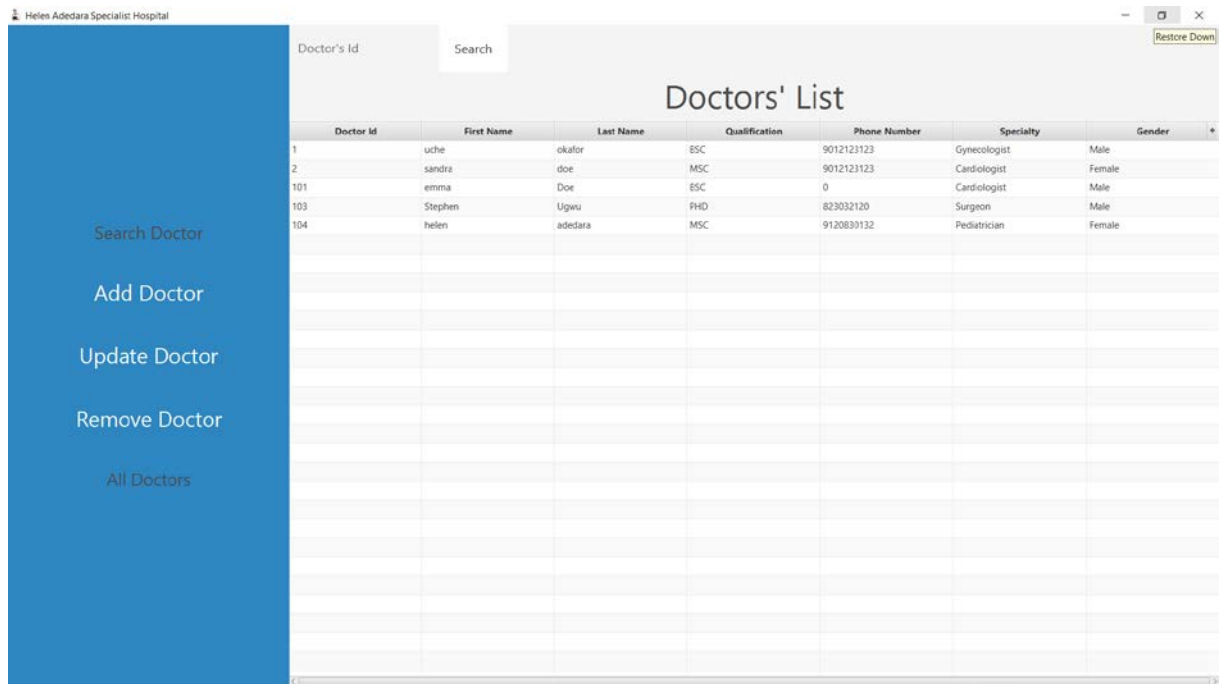


Fig 4.13: Existing doctors list

To add a new doctor, the admin must click the “Add Doctor” button which opens the add doctor page. This is illustrated by the figure below

Helios Adedara Specialist Hospital

Search Doctor

Add Doctor

Update Doctor

Remove Doctor

All Doctors

New Doctor

First Name

eg: John

Last Name

eg: Doe

Gender

Male

Qualification

PHD

Specialty

Cardiologist

Phone

eg: 0802111

Username

eg: john

Password

eg: 12345

Add Doctor

Fig 4.14 Adding new doctor

To update an existing doctor, the admin must select the record to be updated from the doctors' list before clicking the "Update Doctor" button which opens the update doctor page. This is illustrated by the figure below

The screenshot shows a web application interface for a hospital. On the left is a blue sidebar with navigation links: 'Search Doctor', 'Add Doctor', 'Update Doctor', 'Remove Doctor', and 'All Doctors'. The main content area is titled 'Update Doctor' and contains a form with the following fields and values:

Field	Value
First Name	Stephen
Last Name	Ugwu
Gender	Male
Qualification	PHD
Specialty	Surgeon
Phone	823032120
Username	Ugwu
Password	Ugwu

At the bottom of the form is a blue button labeled 'Save'.

Fig 4.15 updating existing doctor

The admin has to click "save" button to update the record in the database.

MANAGING PATIENTS' RECORD

The admin can decide to manage patients by clicking "patient" button from fig 4.6 which will open the patients page



Fig 4.16 Patients page

To enrol a new patient, the admin must click the “add patient” button to open the add patient page as shown in figure below

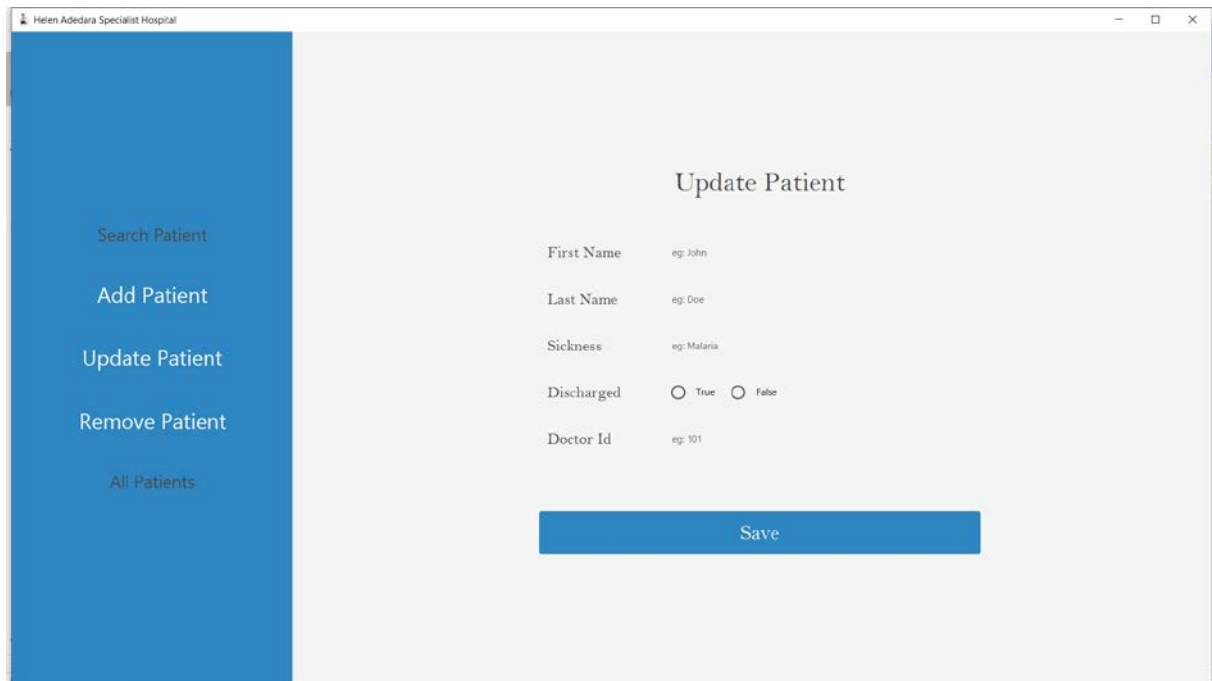
The screenshot shows a web application window titled 'Helen Adedara Specialist Hospital'. On the left is a blue sidebar with the following navigation options: 'Search Patient', 'Add Patient', 'Update Patient', 'Remove Patient', and 'All Patients'. The main area is titled 'New Patient' and contains the following form fields:

- First Name: eg: John
- Last Name: eg: Doe
- Sickness: eg: Malaria
- Discharged: ☐ True ☐ False
- Doctor Id: eg: 101

At the bottom of the form is a blue button labeled 'Enrol Patient'.

Fig 4.17 Add new patient

To update existing patient record, the admin must click the “update patient” button to open the update patient page as shown in figure below



The screenshot shows a web application window titled "Helen Adedara Specialist Hospital". On the left is a blue sidebar with navigation links: "Search Patient", "Add Patient", "Update Patient", "Remove Patient", and "All Patients". The main content area is titled "Update Patient" and contains a form with the following fields: "First Name" (placeholder: eg: John), "Last Name" (placeholder: eg: Doe), "Sickness" (placeholder: eg: Malaria), "Discharged" (radio buttons for "True" and "False"), and "Doctor Id" (placeholder: eg: 101). A blue "Save" button is positioned at the bottom of the form.

Fig 4.18 Update patient

The admin has to click “save” button to update the record in the database.

More images to illustrate the functionality of the app. Adding, updating, removing staff follows the same pattern as discussed in managing doctor section.

[illegible]

Heien Adedera Specialist Hospital

Search Accountant

Add Accountant

Update Accountant

Remove Accountant

All Accountants

New Accountant

First Name	eg: John
Last Name	eg: Doe
Gender	Male ▾
Qualification	PHD ▾
Specialty	Cardiologist ▾
Phone	eg: 0802111
Username	eg: john
Password	eg: 12345

Add Accountant

Heien Adedara Specialist Hospital

Search Accountant

Add Accountant

Update Accountant

Remove Accountant

All Accountants

Update Accountant

First Name	eg: John
Last Name	eg: Doe
Gender	Male
Qualification	PHD
Specialty	Cardiologist
Phone	eg: 0801123
Username	eg: john
Password	eg: 12345

Save

CHAPTER 5

CONCLUSION AND SUMMARY