

Computer Programming Semester Project Fall 2023

Hospital Management System

Submitted to Sir Burhan Abbas

Project Members

Name of Students	Enrollment number
Ameer Hamza	01-134232-040
Arslan Ahmed	01-134232-045
Rabia Jamil	01-134232-156

Date of Submission

24/12/2023

Table Of Contents

<i>Technical Report</i>	2
Introduction	2
Project Idea	2
Features	2
Functionality	3
Main Window:	3
Staff Account:	4
Doctor Account:	8
<i>Screenshots</i>	10
<i>Source Code</i>	19
<i>Conclusion</i>	64
<i>Links</i>	65

Technical Report

Introduction

We have developed a Hospital Management System that is a necessity in every hospital in order to help both doctors and staff in their day-to-day responsibilities, fostering seamless communication and efficient workflow. The staff will be able to sign up, edit information, set appointments, send test results to the doctor and edit patient information. The doctor can also sign up, give time slots to the staff to show which timings are suitable for appointments, request test results and look at their patient's information. Both can write notes and look at their salary depending on their hours calculated by their log in and log out. The staff can also create bills for the patients depending on what services they received.

Project Idea

The Hospital Management System (HMS) is like a digital assistant for hospitals, designed to make things smoother and more organized. Its main job is to bring all the different tasks in a hospital, from keeping patient records to scheduling appointments, into one easy-to-use system. Think of it as a place where doctors, staff, and administrators can find and share information quickly. For doctors, it helps manage patient appointments, access medical histories, and even check their work schedules. Staff members use it to organize appointments, keep patient details up-to-date, and coordinate with doctors. The system also ensures that all this information is kept safe and private. So, in simple terms, the Hospital Management System is like a superhero sidekick, making sure everything runs smoothly in a hospital.

Features

- **Profile Management:**
Doctors and Staff Members can easily sign up and log in to access their personalized profiles, ensuring secure and user-friendly authentication.
- **Patient Information:**
Doctors can gain quick access to patient information. Staff can efficiently manage and edit patient information.
- **Doctor Information:**
Staff can easily check and retrieve information about doctors.
- **Appointment Scheduling:**
Doctors can efficiently manage their schedules by choosing time slots for appointments while Staff Members set up and manage appointments while considering those available time slots.
- **Schedule Management:**
Doctors can access a centralized schedule to stay organized, plan appointments, and minimize scheduling conflicts.
- **Notes and Documentation:**

Doctors and Staff can effortlessly create, edit, and display notes to remember any critical details.

- **Test Requests and Results:**

Doctors can request diagnostic tests for patients and receive results by the Staff, streamlining the diagnostic process for enhanced patient care.

- **Billing:**

Staff can access information easily through the Patient's name and create a bill for them depending on the services they were given.

- **Salary Information:**

Doctors and Staff members can conveniently view their salary information, the number of hours they were present and how much their salary amounts to.

Functionality

Main Window:

➤ **Libraries**

- Includes various Qt libraries for user interface elements and database connectivity.
- Uses standard C++ libraries as well.

➤ **Email Validation Function**

bool em(QString &email)

- Checks if the entered email follows the valid email format.
- Uses a regular expression to match the email format and returns a boolean value indicating the result.

➤ **Database Connection Function**

bool conndb()

- Connects to the MySQL database using specified parameters.
- Uses the Qt library to set up a connection with the database and returns a boolean value based on the connection status.

➤ **Doctor Login**

void MainWindow::on_pushButton_9_clicked()

- Handles the login process for doctors.
- Connects to the database, retrieves user input, checks credentials, and opens the doctor window on successful login.

➤ **Doctor Signup**

void MainWindow::on_pushButton_11_clicked()

- Handles the signup process for doctors.
- Connects to the database, generates a unique code, retrieves user input, inserts data into the database, and provides feedback on the success or failure of the signup.

➤ **Staff Login**

void MainWindow::on_pushButton_10_clicked()

- Handles the login process for staff members.
- Connects to the database, retrieves user input, checks credentials, and opens the staff window on successful login.

➤ **Staff Signup**

void MainWindow::on_pushButton_12_clicked()

- Handles the signup process for staff members.
- Connects to the database, generates a unique code, retrieves user input, inserts data into the database, and provides feedback on the success or failure of the signup.

Staff Account:

➤ **Libraries:**

- Also includes several header files and libraries necessary for Qt and C++ functionalities.

➤ **Database Connection Function:**

conndb()

- Defined to establish a connection to the database.
- Is called at various points in the code before executing database queries.

➤ **Function to Reset Radio Buttons:**

resetrad()

- Resets a group of radio buttons, making them enabled and unchecked.
- Is useful for preparing the radio buttons for user interaction.

➤ **Function for Checking Appointment Availability:**

trtime

- Checks if an appointment is already booked based on the selected day, doctor name, and department.

- Retrieves booked time slots from the database and disables corresponding radio buttons if the time is already booked.

➤ **Function to Check if Radio Buttons are Disabled:**

Isdisabled()

- Checks if any of the radio buttons is disabled. It is used to determine if all radio buttons are disabled after an appointment is booked.

➤ **Constructor and Destructor for Staff Window:**

staffwindow

- Includes a constructor and destructor for initializing and cleaning up the staff window.

➤ **Tab Widget Signal Handler:**

on_tabWidget_tabBarClicked

- Handles the signal when a tab in the widget is clicked.
- Retrieves and displays staff information from the database.

➤ **Functions for Reading and Writing to File:**

on_pushButton_11_clicked

- Writes content to a file named "staffnotes.txt".

on_pushButton_15_clicked

- Reads and displays the content from the same file.

➤ **Appointment Submission Function:**

on_pushButton_9_clicked

- Handles the submission of appointments.
- Retrieves selected values (day, time, doctor, patient, department) and inserts them into the database.
- Updates the UI by disabling selected radio buttons.

➤ **Signal Handlers for Combobox and Day Selection:**

on_daycombo_currentIndexChanged

on_doctorcombo_currentIndexChanged

- Handle changes in the day and doctor combo boxes, respectively.

- Call the trtime function to update radio button availability.

➤ **Logout Function:**

on_pushButton_clicked

- Calculates the working hours of the staff and updates the database with the logged hours.
- Then returns to the main window.

➤ **Lab Test Update Function:**

on_pushButton_10_clicked

- Function updates the status and result of lab tests based on user selection in combo boxes.

➤ **Display Lab Tests and Doctors in Tables:**

on_pushButton_3_clicked

on_pushButton_4_clicked

- Retrieve and display information about lab tests and doctors from the database in tables.

➤ **Search Functionality for Tables:**

on_lineEdit_3_textChanged

on_lineEdit_2_textChanged

- Allows searching within tables based on user input.

➤ **Patient Registration Function:**

on_pushButton_2_clicked

- Function inserts patient information into the database when the staff registers a new patient.

➤ **Display Patients and Appointments in Tables:**

on_pushButton_5_clicked

on_pushButton_6_clicked

- Retrieve and display information about patients and appointments in tables.

➤ **Dropdown Population Function for Doctor and Department:**

on_patdocdep_currentIndexChanged

- Populates the doctor dropdown based on the selected department.

➤ **Generate Patient Code Function:**

on_lineEdit_78_editingFinished

- Function generates a unique patient code when the patient name is entered.

Fetching Patient Names:

Button click (on_pushButton_7_clicked())

- Checks if it can connect to the patient database using conndb() function.
- Retrieves patient names from the database.
- Clears and updates the combo box (**ui->comboBill**) with fetched names.
- Manages errors in database connection and query execution.

Calculating Billing Information:

Button click (on_pushButton_8_clicked()).

- Queries the database for the count of appointments by department.
- Queries the database for the count of lab tests by type.
- Predefined prices for each department and test type.
- Calculates the total cost based on the counts and prices.
- Updates a text browser (**ui->textBrowser_19**) with patient name and total cost details.
- Updates another text browser (**ui->textBrowser_20**) with the full bill, including the total cost.

Profile Information Display:

on_tabWidget_tabBarClicked()

- Utilizes a structure (staff) to organize staff information.
- Connects to the database using conndb() function.
- Retrieves staff details such as name, gender, date of birth, CNIC, and contact from the database.
- Displays the information in a structured format in a text browser (**ui->textBrowser_7**).
- Displays a message if the query fails.

Salary Information:

- Queries the database for salary-related information, specifically the hours worked.
- Calculates the total hours worked and displays the number of days worked and total hours.
- Computes the salary based on the total hours worked (assuming a rate of \$15 per hour).
- Displays the salary information in two separate text browsers (**ui->textBrowser_salary** and **ui->textBrowser_salary2**).

Test Result Sections Combobox Update:

- Queries the database for lab test information (patname and testtype).

Combobox Update:

- Clears existing items in three comboboxes (**ui->comboBox**, **ui->comboBox_2**, **ui->comboBox_3**).
- Populates the comboboxes with patient names (patname) and test types (testtype).
- Ensures that duplicate entries are not added to the patient name combobox (**ui->comboBox**).

Doctor Account:

➤ **Database Connection Function:**

conndb()

- Defined to establish a connection to the database.
- Is called at various points in the code before executing database queries.

➤ **User Interface Initialization:**

docwindow::docwindow

- Initializes the UI for the document window. It sets up a tab widget with the default index set to 0.

➤ **Fetching User Information:**

docwindow::on_tabWidget_tabBarClicked

- Fetches and displays information about the logged-in doctor, such as name, department, gender, contact details, etc.
- Also fills a combo box with patient names associated with the doctor.

➤ **Displaying Patient Information:**

docwindow::on_pushButton_2_clicked

- Retrieves patient information (name, code, gender, etc.) for the logged-in doctor and populates a table widget with this data.

➤ **Patient Search:**

docwindow::on_lineEdit_textChanged

- Allows the user to search for specific text in a table (ui->tableWidget_5) and dynamically hide rows that do not match the search criteria.

➤ **Calculating and Updating Work Hours:**

docwindow::on_pushButton_clicked

- Calculates the worked hours based on login and logout times and updates the database with the logged-in doctor's information.

➤ **Displaying and Updating Doctor's Schedule:**

docwindow::on_pushButton_3_clicked

- Retrieves and displays the doctor's schedule from the database.
- Updates a table widget (**ui->tableWidget_2**) with patient names based on the day and time.

➤ **Lab Test Requests:**

docwindow::on_pushButton_4_clicked

- Inserts lab test requests into the 'lab' table of the database, recording details such as doctor name, patient name, test type, status, and request date.

➤ **Displaying Lab Test Requests:**

docwindow::on_pushButton_5_clicked

- Fetches and displays lab test requests for the logged-in doctor in a table widget (**ui->tableWidget_3**).

Profile Information Display:

on_tabWidget_tabBarClicked()

- Utilizes a structure (doc) to organize doctor information.
- Connects to the database using **conndb()** function.
- Retrieves doctor details such as name, department, gender, date of birth, CNIC, contact, and address from the database.
- Displays the information in a structured format in a text browser (**ui->textBrowser_7**).

Salary Information:

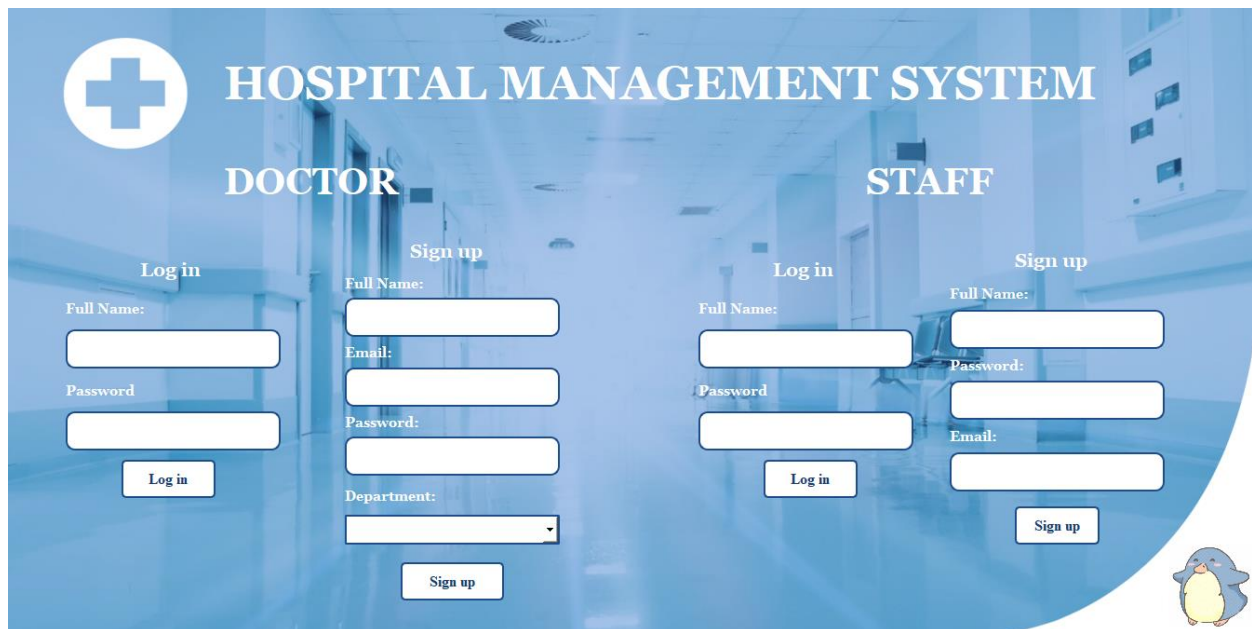
- Queries the database for salary-related information, specifically the hours worked.
- Calculates the total hours worked and displays the number of days worked and total hours.
- Retrieves the doctor's department to determine the department-specific hourly rate.
- Calculates the total salary based on the total hours worked and department-specific rate.
- Displays the salary information in two separate text browsers (**ui->textBrowser_salary** and **ui->textBrowser_salary2**).

Lab Results Section Combobox Update:

- Queries the database for patient names under the care of the logged-in doctor (loggedindocname).
- Populates a combobox (**ui->comboBox_3**) with patient names.
- Prints a message in the console if the data retrieval fails.

Screenshots

Main Window:



HOSPITAL MANAGEMENT SYSTEM

DOCTOR

Log in Sign up

Full Name:

Email:

Password:

Department:

Log in Sign up

STAFF

Log in Sign up

Full Name:

Password:

Email:

Log in Sign up

DOCTOR MANAGEMENT SYSTEM

[Home](#) [Profile](#) [Patients](#) [Schedule](#) [Notes](#) [Test Results](#) [Salary](#)

Welcome!

Hello Doctor! Below are some guidelines you are suggested to revise everyday.

- ▶ Prioritize patient care and safety, delivering high-quality medical services with empathy and professionalism.
- ▶ Uphold the highest ethical standards, respecting patient confidentiality and privacy at all times.
- ▶ Be prepared for emergencies, maintaining knowledge of emergency procedures and the location of emergency equipment.
- ▶ Provide patients with clear and comprehensive information about their conditions and treatment plans, encouraging their active involvement in healthcare decisions.
- ▶ Prioritize personal well-being, seeking support for stress or burnout and maintaining a healthy work-life balance.



DOCTOR MANAGEMENT SYSTEM

[Home](#) [Profile](#) [Patients](#) [Schedule](#) [Notes](#) [Test Results](#) [Salary](#)

Profile Information

Name:

Gender:

DOB:

CNIC:

Contact:

Address:

DOCTOR MANAGEMENT SYSTEM

Home Profile Patients Schedule Notes Test Results Salary

Patients

Search here...

	Name	Code	Gender	DOB	CNIC	Contact
1						
2						
3						
4						
5						
7						
8						
9						
10						
11						
12						

Refresh

DOCTOR MANAGEMENT SYSTEM

Home Profile Patients Schedule Notes Test Results Salary

Schedule

Your schedule for this month is here.

	8AM-9AM	9AM-10AM	10AM-11AM	11AM-12PM	12PM-1PM	BREAK	2PM-3PM
Monday							
Tuesday							
Wednesday							
Thursday							
Friday							
Saturday							

Refresh

DOCTOR MANAGEMENT SYSTEM

[Home](#) [Profile](#) [Patients](#) [Schedule](#) [Notes](#) [Test Results](#) [Salary](#)

Notes

Read

Write

DOCTOR MANAGEMENT SYSTEM

[Home](#) [Profile](#) [Patients](#) [Schedule](#) [Notes](#) [Test Results](#) [Salary](#)



Salary

Total Work Days:
No. Of Hours Worked:

Your salary for this month is:

Heading out? Have a good day!

Logout

DOCTOR MANAGEMENT SYSTEM

Home Profile Patients Schedule Notes **Test Results** Salary

Request A Test Here

Patient:

Test type:

- ☐ CBC ☐ BMP ☐ Blood Type
☐ Urinalysis ☐ Allergy Test ☐ Hormone Test
☐ ECG ☐ X-Ray ☐ Ultrasound
☐ CT Scan ☐ Glucose Test ☐ Covid Test

Submit

Test Results

	Patient	Test type	Status	Result
1				
2				
3				
4				
5				
7				
8				
9				
10				
11				
12				

Refresh

Staff Account:

STAFF MANAGEMENT SYSTEM

Home Profile Doctors Patients Appointment Notes Test Results Billing Salary

Welcome

Hello Staff! Below are some guidelines you are suggested to revise everyday.

- Prioritize patient well-being and satisfaction, maintaining a compassionate and respectful attitude.
- Adhere to high ethical standards, respecting patient confidentiality and privacy.
- Maintain accurate and timely documentation of patient records, ensuring compliance with regulatory standards.
- Familiarize yourself with and adhere to hospital policies and procedures, reporting any violations or concerns.
- Prioritize your own physical and mental well-being, seeking support if experiencing stress or burnout.



STAFF MANAGEMENT SYSTEM

Home Profile Doctors Patients Appointment Notes Test Results Billing Salary

Profile Information

Name:
Gender:
DOB:
CNIC:
Contact:
Address:

STAFF MANAGEMENT SYSTEM

Home Profile Doctors Patients Appointment Notes Test Results Billing Salary

Doctors

Search here...

	Name	Code	Department	Gender	CNIC	DOB	Contact
1							
2							
3							
4							
5							
7							
8							
9							
10							
11							
12							

Refresh

STAFF MANAGEMENT SYSTEM

Home Profile Doctors Patients Appointment Notes Test Results Billing Salary

Name:

DOB:

1/1/2000

Department:

Cardiology

Doctor:

CNIC:

Gender

Contact:

Code

Submit

Patients

Search...

	Name	DOB	Department	Doctor	CNIC	Gender	Contact	Code
1								
2								
3								
4								
5								
7								
8								
9								
10								
11								
12								

Refresh

STAFF MANAGEMENT SYSTEM

Home Profile Doctors Patients Appointment Notes Test Results Billing Salary

Appointments

Department

Doctor:

Patient:

Day:

Time:

● 8AM-9AM

● 9AM-10AM

● 10AM-11AM

● 11AM-12PM

● 12PM-1PM

● 2 PM - 3 PM

Submit

Set appointments for the patients here.

	Department	Doctor	Patient	Day	Time
1					
2					
3					
4					
5					
7					
8					
9					
10					
11					
12					

Refresh

STAFF MANAGEMENT SYSTEM

Home Profile Doctors Patients Appointment Notes Test Results Billing Salary

Notes

Read

Write

STAFF MANAGEMENT SYSTEM

Home Profile Doctors Patients Appointment Notes Test Results Billing Salary

Test Results

Patient:

Test Type:

Status:

Result:

Update

Test Requests

	Patient	Test type	Status	Result
1				
2				
3				
4				
5				
7				
8				
9				
10				
11				
12				

Refresh

STAFF MANAGEMENT SYSTEM

[Home](#) [Profile](#) [Doctors](#) [Patients](#) [Appointment](#) [Notes](#) [Test Results](#) [Billing](#) [Salary](#)

Billing



Patient

Calculate

Refresh

Your total bill is:

STAFF MANAGEMENT SYSTEM

[Home](#) [Profile](#) [Doctors](#) [Patients](#) [Appointment](#) [Notes](#) [Test Results](#) [Billing](#) [Salary](#)



Salary

Total Work Days:
No. Of Hours Worked:

Your salary for this month is:

Heading out? Have a good day!

Logout

Source Code

Header Files:

Main Window:

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget* parent = nullptr);
    ~MainWindow();

private slots:
    void on_pushButton_9_clicked();
    void on_pushButton_11_clicked();
    void on_pushButton_10_clicked();
    void on_pushButton_12_clicked();

private:
    Ui::MainWindow* ui;
};
#endif // MAINWINDOW_H
```

Global Variable:

```
#ifndef GLOBALVAR_H
#define GLOBALVAR_H
#include<QString>
extern QString loggedindocemail;
extern QString loggedindocname;
```

```

extern QString loggedinstaffemail;
extern QString loggedinstaffname;
extern QString day; //For day of appointment to be set by staff
extern QString tim; //For time of appointment to be set by staff
extern QString docname; //For name of doctor of appointment to be
set by staff
extern QString patname; //For name of patient of appointment to be
set by staff
extern QString department; //For name of department of appointment
to be set by staff
extern int doccount; //For checking number of doctors added as
item in the combobox_2 of appointment section
extern double stlogin; //stafflogin time record
extern double stlogout; //staff logout time record
extern double dclogin; //doctor login time record
extern double dclogout; //doctor logout time record
extern QString patcode;
#endif // GLOBALVAR_H

```

Staff Account:

```

#ifndef STAFFWINDOW_H
#define STAFFWINDOW_H

#include <QDialog>

namespace Ui {
    class staffwindow;
}

class staffwindow : public QDialog
{
    Q_OBJECT

public:
    explicit staffwindow(QWidget* parent = nullptr);
    ~staffwindow();

private slots:
    void on_tabWidget_tabBarClicked(int index);

    void on_pushButton_11_clicked();

    void on_pushButton_9_clicked();

    void on_departmentcombo_currentIndexChanged(int index);

```

```

void on_doctorcombo_currentIndexChanged(int index);

void on_daycombo_currentIndexChanged(int index);

void on_pushButton_clicked();

void on_pushButton_10_clicked();

void on_pushButton_3_clicked();


void on_pushButton_4_clicked();

void on_lineEdit_3_textChanged(const QString& arg1);

void on_lineEdit_2_textChanged(const QString& arg1);

void on_pushButton_2_clicked();

void on_patdocdep_currentIndexChanged(int index);

void on_lineEdit_78_editingFinished();

#ifndef STAFFWINDOW_H
#define STAFFWINDOW_H

#include <QDialog>

namespace Ui {
    class staffwindow;
}

class staffwindow : public QDialog
{
    Q_OBJECT

public:
    explicit staffwindow(QWidget* parent = nullptr);
    ~staffwindow();

private slots:
    void on_tabWidget_tabBarClicked(int index);

    void on_pushButton_11_clicked();

```

```
void on_pushButton_9_clicked();

void on_departmentcombo_currentIndexChanged(int index);

void on_doctorcombo_currentIndexChanged(int index);

void on_daycombo_currentIndexChanged(int index);

void on_pushButton_clicked();

void on_pushButton_10_clicked();

void on_pushButton_3_clicked();


void on_pushButton_4_clicked();

void on_lineEdit_3_textChanged(const QString& arg1);

void on_lineEdit_2_textChanged(const QString& arg1);

void on_pushButton_2_clicked();

void on_patdocdep_currentIndexChanged(int index);

void on_lineEdit_78_editingFinished();

void on_pushButton_5_clicked();

void on_pushButton_6_clicked();


void on_pushButton_7_clicked();


void on_pushButton_8_clicked();

void on_pushButton_12_clicked();

private:
    Ui::staffwindow* ui;
    int Isdisabled();
```

```

        void trtime(const QString& day, QString docname, QString
department);
        void resetrad();

};

#endif // STAFFWINDOW_H

```

Doctor Account:

```

#ifndef DOCWINDOW_H
#define DOCWINDOW_H

#include <QDialog>

namespace Ui {
    class docwindow;
}

class docwindow : public QDialog
{
    Q_OBJECT

public:
    explicit docwindow(QWidget* parent = nullptr);
    ~docwindow();

private slots:
    void on_tabWidget_tabBarClicked(int index);

    void on_pushButton_2_clicked();

    void on_lineEdit_textChanged(const QString& arg1);

    void on_pushButton_clicked();

    void on_pushButton_3_clicked();

    void on_pushButton_4_clicked();

    void on_pushButton_5_clicked();

    void on_pushButton_11_clicked();

```

```

        void on_pushButton_6_clicked();

private:
    Ui::docwindow* ui;
    void chbox();

};

#endif // DOCWINDOW_H

```

CPP Files:

Main Window:

```

#include "mainwindow.h"
#include "../ui_mainwindow.h"
#include<QString>
#include<globalvar.h>
#include<QMessageBox>
#include<staffwindow.h>
#include<QTime>
#include<QRegularExpression>
#include<QtSql/SqlDatabase>
#include<QtSql/SqlQuery>
#include<docwindow.h>
#include<QUuid>
#include<QPixmap>
#include <QMovie>
using namespace std;
//function for checking email
bool em(QString& email)
{
    QRegularExpression regex("^([a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-
]+\\.[a-zA-Z]{2,}$)"); //used for setting the format for checking
if entered data is email or not which consists of digits and
numbers a dot and an @ symbol
    return regex.match(email).hasMatch(); //checks if the entered
email is valid or not and returns a boolean value
}
bool conndb()
{
    QSqlDatabase db = QSqlDatabase::addDatabase("QMYSQL");
    db.setHostName("localhost");
    db.setUserName("root");
    db.setPassword("");
}

```



```

        db.setDatabaseName("hospital db");
        if (db.open())
        {
            return true;
        }
        else if (!db.open())
        {
            return false;
        }
    }
MainWindow::MainWindow(QWidget* parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    //for background
    QImage bgpic("C:/Users/hstech/Downloads/resource/bg.png");
    ui->bg_pic->setScaledContents(true);
    ui->bg_pic->setPixmap(QPixmap::fromImage(bgpic));
    ui->bg_pic->setVisible(true);
    //for gifs in the program
    QMovie* gif1 = new
    QMovie("C:/Users/hstech/Downloads/resource/penguin.gif");
    ui->labelGif->setMovie(gif1);
    gif1->start();
    ui->labelGif->setScaledContents(true);
}

MainWindow::~MainWindow()
{
    delete ui;
}

//FOR DOCTOR LOGIN
void MainWindow::on_pushButton_9_clicked()
{

    //Connecting Database
    conndb();
    //Checks if database is open
    if (conndb())
    {
        // Retrieves input from user for login
    }
}

```

```

QString email = ui->ldocemail->text();
QString password = ui->ldocpass->text();
loggedindocemail = email;
if (em(email) == true && !email.isEmpty() &&
!password.isEmpty())
{

```

```

    // Retrieving our user data from the database
    generated from wamp using sql query

```

```

        QSqlQuery q;
        q.prepare("SELECT email,password,name FROM
dcredentials WHERE email= :email AND password= :password");
//Selects email password from dcredentials where it will only
extract those values which are == to the entered password or email

```

```

        q.bindValue(":email", email);
        q.bindValue(":password", password);

```

```

        // If credentials were found in the database
        if (q.exec() && q.next())
        {
            loggedindocname = q.value("name").toString();
            qDebug() << loggedindocname;
            QMessageBox::information(this, "Success", "Login
Successful");
            QString currenthour =
QTime::currentTime().toString("hh");
            QString currentminute =
QTime::currentTime().toString("mm");
            QString currentsec =
QTime::currentTime().toString("ss");
            double hourtosec = currenthour.toInt() * 3600.0;
            double minutetosec = currentminute.toInt() * 60.0;
            int sec = currentsec.toInt();
            dclogint = hourtosec + minutetosec + sec; //
Replace stlogint with dclogint
            dclogint = dclogint / 3600.0;
            qDebug() << dclogint;

            hide();
            docwindow dw;
            dw.setModal(true);

```

```

        dw.exec();
    }
    //If credentials were NOT found in the database
    else
    {
        QMessageBox::warning(this, "Failed", "Invalid
Credentials");
    }
    else
    {
        QMessageBox::warning(this, "Error", "Fill out all the
fields and use correct format for email");
    }
    }
    else
    {
        QMessageBox::critical(this, "Not Connected", "Database is
not connected");
    }
}

//FOR DOCTORS SIGN UP
void MainWindow::on_pushButton_11_clicked()
{
    //Connecting Database
    conndb();

    //Checks if database is open
    if (conndb())
    {
        QString doccode = QUuid::createUuid().toString().mid(1,
8);
        qDebug() << doccode;
        // Retrieving input from user for signup
        QString name = ui->sdocname->text();
        QString email = ui->sdocemail->text();
        QString password = ui->sdocpass->text();
        QString department = ui->sdocdep->currentText();

        //if email is valid and all the line edits arent empty
        if (em(email) == true && !name.isEmpty() &&
!email.isEmpty() && !password.isEmpty() && !department.isEmpty())
        //checks if all the line edit boxes have some value entered in
them and if the entered email follows the format of an email

```

```

    {
        // The following four lines clear the details entered
in the signup bar
        ui->sdocname->clear();
        ui->sdocemail->clear();
        ui->sdocpass->clear();
        ui->sdocdep->setCurrentIndex(0);

        // Inserting our input data into the database
generated from wamp using sql query

        QSqlQuery q;
        q.prepare("INSERT INTO
dcredentials(name,code,email,password,department)"
"Values(:name,:code,:email,:password,:department)");
        q.bindValue(":name", name);
        q.bindValue(":email", email);
        q.bindValue(":password", password);
        q.bindValue(":department", department);
        q.bindValue(":code", doccode);

        // If Signup Successful/Information is stored
successfully
        if (q.exec()) {

            QMessageBox::information(this, "Success", "SignUp
Successful");

        }
        //If signup is unsuccessful/Information is not stored
else if (!q.exec()) {
            QMessageBox::warning(this, "Failed", "SignUp
Failed");
        }
    }
    //if email is not valid or a input field is empty
else
{
    QMessageBox::warning(this, "Error", "Fill out all the
fields and use correct format for email");
}
}
//if Database is not connected
else {

```

```

        QMessageBox::critical(this, "Not Connected", "Database is
not connected");
    }
}

//FOR STAFF LOGIN
void MainWindow::on_pushButton_10_clicked()
{
    conndb();

    if (conndb())
    {
        // Retrieving input from user for login

        QString email = ui->lstaffemail->text();
        loggedinstaffemail = email;
        QString password = ui->lstaffpass->text();

        if (em(email) == true && !email.isEmpty() &&
!password.isEmpty())
        {

            // Retrieving our user data from the database
generated from wamp using sql query

            QSqlQuery q;
            q.prepare("SELECT email,password,name FROM
scredentials WHERE email= :email AND password= :password");

            q.bindValue(":email", email);
            q.bindValue(":password", password);

            // If Login is Successful/Entered credentials are
correct
            if (q.exec() && q.next()) {
                loggedinstaffname = q.value("name").toString();
                QMessageBox::information(this, "Success", "Login
Successful");

                QString currenthour =
QTime::currentTime().toString("hh");
                QString currentminute =
QTime::currentTime().toString("mm");

```

```

        QString currentsec =
QTime::currentTime().toString("ss");
        double hourtosec = currenthour.toInt() * 3600.0;
        double minutetosec = currentminute.toInt() * 60.0;
        int sec = currentsec.toInt();
        stlogint = hourtosec + minutetosec + sec;
        stlogint = stlogint / 3600.0;
        qDebug() << stlogint;

        hide();
        staffwindow sw;
        sw.setModal(true);
        sw.exec();

    }
    //If Login in Unsuccessful/Entered credentials are
incorrect
    else
    {
        QMessageBox::warning(this, "Failed", "Invalid
Credentials");
    }
    else
    {
        QMessageBox::warning(this, "Error", "Fill out all the
fields and use correct format for email");
    }
    else {
        QMessageBox::critical(this, "Not Connected", "Database is
not connected");
    }
}

//FOR STAFF SIGNUP
void MainWindow::on_pushButton_12_clicked()
{
    conndb();

    if (conndb())
    {
        QString staffcode = QUuid::createUuid().toString().mid(1,
8);
        qDebug() << "Staffcode" << staffcode;
        // Retrieving input from user for signup

```

```

QString name = ui->sstaffname->text();
QString email = ui->sstaffemail->text();
QString password = ui->sstaffpass->text();

//if email is valid and variables created aren't empty
if (em(email) == true && !name.isEmpty() &&
!email.isEmpty() && !password.isEmpty())
{
    //For clearing the input fields
    ui->sstaffname->clear();
    ui->sstaffemail->clear();
    ui->sstaffpass->clear();

    // Inserting our input data into the database
    generated from wamp using sql query

    QSqlQuery q;
    q.prepare("INSERT INTO
scredentials(name,code,email,password)
"Values(:name,:code,:email,:password)");
    q.bindValue(":name", name);
    q.bindValue(":email", email);
    q.bindValue(":password", password);
    q.bindValue(":code", staffcode);
    if (q.exec()) {

        QMessageBox::information(this, "Success", "SignUp
Successful");
    }
    else {
        QMessageBox::information(this, "Failed", "SignUp
Failed");
    }
}
//If email is not valid
else
{
    QMessageBox::warning(this, "Error", "Fill out all the
fields and use correct format for email");
}
}
//If database is not connected
else {
    QMessageBox::critical(this, "Not Connected", "Database is
not connected");
}

```

```
}  
}
```

Staff Account:

```
#include "staffwindow.h"  
#include "ui_staffwindow.h"  
#include <QSqlDatabase>  
#include <QSqlQuery>  
#include <QMessageBox>  
#include <QString>  
#include <globalvar.h>  
#include <QFile>  
#include <mainwindow.h>  
#include <QUuid>  
#include <QDate>  
#include <QMovie>  
bool conndb();  
void staffwindow::resetrad() //Function for resetting all radio  
buttons (had to declare the function in the header file's private  
slot to access the ui)  
{  
    ui->radioButton->setEnabled(true);  
    ui->radioButton->setChecked(false);  
  
    ui->radioButton_2->setEnabled(true);  
    ui->radioButton_2->setChecked(false);  
  
    ui->radioButton_3->setEnabled(true);  
    ui->radioButton_3->setChecked(false);  
  
    ui->radioButton_4->setEnabled(true);  
    ui->radioButton_4->setChecked(false);  
  
    ui->radioButton_5->setEnabled(true);  
    ui->radioButton_5->setChecked(false);  
  
    ui->radioButton_6->setEnabled(true);  
    ui->radioButton_6->setChecked(false);  
}
```

```
void staffwindow::trtime(const QString& day, QString docname,  
QString department) //trtime function for checking if an  
appointment is already booked based upon doctorname and time  
{
```



```

conndb(); //connecting database
if (conndb()) //if connection successful
{
    resetrad(); //reset radio buttons before every query
    QSqlQuery checkrad; //query for retrieving data
    checkrad.prepare("SELECT tim FROM appointment WHERE
day=:day AND docname=:docname AND department =:department ");
//retrieves the data stored in tim if the day and docname selected
by the user match the data stored in the database
    checkrad.bindValue(":day", day); //binds values/replaces
    checkrad.bindValue(":docname", docname); //binds
values/replaces
    checkrad.bindValue(":department", department);
    if (checkrad.exec()) //if query was executed
    {
        while (checkrad.next()) //if values were found
        {
            QString bookedtime =
checkrad.value("tim").toString(); //Stores the value stored in the
column of tim in the bookedtime variable
            qDebug() << bookedtime;
            // if conditionals to check if radio button
matches the booked time, if it does then it will disable the radio
button and if the radio button is checked it will uncheck it this
step continues for all the radio buttons
            if (ui->radioButton->text() == bookedtime)
            {
                ui->radioButton->setDisabled(true);
                if (ui->radioButton->isChecked())
                {
                    ui->radioButton->setChecked(false);
                }
            }
            else if (ui->radioButton_2->text() == bookedtime)
            {
                ui->radioButton_2->setDisabled(true);
                if (ui->radioButton_2->isChecked())
                {
                    ui->radioButton_2->setChecked(false);
                }
            }
            else if (ui->radioButton_3->text() == bookedtime)
            {
                ui->radioButton_3->setDisabled(true);
                if (ui->radioButton_3->isChecked())
                {

```

```

        ui->radioButton_3->setChecked(false);
    }
}
else if (ui->radioButton_4->text() == bookedtime)
{
    ui->radioButton_4->setDisabled(true);
    if (ui->radioButton_4->isChecked())
    {
        ui->radioButton_4->setChecked(false);
    }
}
else if (ui->radioButton_5->text() == bookedtime)
{
    ui->radioButton_5->setDisabled(true);
    if (ui->radioButton_5->isChecked())
    {
        ui->radioButton_5->setChecked(false);
    }
}
else if (ui->radioButton_6->text() == bookedtime)
{
    ui->radioButton_6->setDisabled(true);
    if (ui->radioButton_6->isChecked())
    {
        ui->radioButton_6->setChecked(false);
    }
}
}
}
else
{
    QMessageBox::warning(this, "Error", "Query Failed");
}
}

else    //If database connection fails
{
    QMessageBox::warning(this, "Connection Error", "Error
Connecting"); //Messagebox to display that the connection failed
}
}
//For checking if the radio button is disabled or not (had to
declare the function in the header file's private slot to access
the ui)
int staffwindow::Isdisabled()
{

```

```

    int k = 0;
    if (ui->radioButton->isChecked())
    {
        ui->radioButton->setDisabled(true);
        k++;
    }
    else if (ui->radioButton_2->isChecked())
    {
        ui->radioButton_2->setDisabled(true);
        k++;
    }
    else if (ui->radioButton_3->isChecked())
    {
        ui->radioButton_3->setDisabled(true);
        k++;
    }
    else if (ui->radioButton_4->isChecked())
    {
        ui->radioButton_4->setDisabled(true);
        k++;
    }
    else if (ui->radioButton_5->isChecked())
    {
        ui->radioButton_5->setDisabled(true);
        k++;
    }
    else if (ui->radioButton_6->isChecked())
    {
        ui->radioButton_6->setDisabled(true);
        k++;
    }
    return k;
}

staffwindow::staffwindow(QWidget* parent) :
    QDialog(parent),
    ui(new Ui::staffwindow)
{
    ui->setupUi(this);
    //for gifs in the program
    QMovie* gif1 = new
    QMovie("C:/Users/hstech/Downloads/resource/sparkle.gif");
    ui->labelGif->setMovie(gif1);
    gif1->start();
    ui->labelGif->setScaledContents(true);

```

```

QMovie* gif2 = new
QMovie("C:/Users/hstech/Downloads/resource/calc.gif");
ui->labelGif_2->setMovie(gif2);
gif2->start();
ui->labelGif_2->setScaledContents(true);

QMovie* gif3 = new
QMovie("C:/Users/hstech/Downloads/resource/tree.gif");
ui->labelGif_3->setMovie(gif3);
gif3->start();
ui->labelGif_3->setScaledContents(true);

QMovie* gif4 = new
QMovie("C:/Users/hstech/Downloads/resource/tree.gif");
ui->labelGif_4->setMovie(gif4);
gif4->start();
ui->labelGif_4->setScaledContents(true);
}

staffwindow::~~staffwindow()
{
    delete ui;
}

void staffwindow::on_tabWidget_tabBarClicked(int index)
{
    struct staff {
        QString name;
        QString department;
        QString gender;
        QDate dob;
        qint64 cnic;
        qint64 contact;
    };

    conndb();
    QString tempmail = loggedinstaffemail;
    staff st;

    if (conndb()) {
        QSqlQuery ba;
        ba.prepare("SELECT name, gender, dob, cnic, contact FROM
scredentials WHERE email=:email");
        ba.bindValue(":email", tempmail);
    }
}

```

```

        if (ba.exec() && ba.next()) {

            st.name = ba.value("name").toString();
            st.gender = ba.value("gender").toString();
            st.cnic = ba.value("cnic").toInt();
            st.dob = ba.value("dob").toDate();
            st.contact = ba.value("contact").toInt();

            QString resultText = "Name: " + st.name + "\n\n"
                + "Gender: " + st.gender + "\n\n"
                + "CNIC: " + QString::number(st.cnic) + "\n\n"
                + "DOB: " + (st.dob.toString("dd/MM/yyyy")) +
"\n\n"
                + "Contact: " + QString::number(st.contact) +
"\n\n";

            ui->textBrowser_7->setText(resultText);
            qDebug() << st.dob.toString("yyyy/MM/dd");
            qDebug() << st.cnic;
        }
        else {
            QMessageBox::information(this, "Contact Admin", "Query
Failed");
        }
        qint32 k = 0;
        double hoursworked;
        double totalhours = 0;
        QSqlQuery salary;
        salary.prepare("SELECT * FROM salary WHERE
name=:loggedname");
        salary.bindValue(":loggedname", loggedinstaffname);
        if (salary.exec())
        {
            while (salary.next())
            {
                hoursworked =
salary.value("hoursworked").toDouble();
                totalhours = totalhours + hoursworked;

                k++;
            }
            QString text = "No. Of Days Worked: " +
QString::number(k) +
"\n\nTotal Hours Worked: " +
QString::number(totalhours);

```

```

        ui->textBrowser_salary->setText(text);
        double salary = totalhours * 15;
        QString saltext = "Your Salary Uptill Now Is: " +
QString::number(salary);
        ui->textBrowser_salary2->setText(saltext);

    }
}
void staffwindow::on_pushButton_10_clicked()
{
    QSqlQuery update;
    update.prepare("UPDATE lab SET status=:status, result=:result
WHERE patname=:name AND testtype=:testname");
    update.bindValue(":name", ui->comboBox->currentText());
    update.bindValue(":status", ui->comboBox_3->currentText());
    update.bindValue(":result", ui->comboBox_4->currentText());
    update.bindValue(":testname", ui->comboBox_2->currentText());
    if (!update.exec())
    {
        qDebug() << "Failed";
    }
    ui->comboBox->setCurrentIndex(0);
    ui->comboBox_2->setCurrentIndex(0);
    ui->comboBox_3->setCurrentIndex(0);
    ui->comboBox_4->setCurrentIndex(0);
}

void staffwindow::on_pushButton_3_clicked()
{
    if (conndb())
    {
        QSqlQuery uplab;
        uplab.prepare("SELECT * FROM lab");
        if (uplab.exec())
        {
            ui->tableWidget_9->clearContents();
            ui->tableWidget_9->setRowCount(0);

            while (uplab.next())
            {

                QString name = uplab.value("patname").toString();
                QString doctor =
uplab.value("docname").toString();

```

```

        QString testtype =
uplab.value("testtype").toString();
        QString status = uplab.value("status").toString();
        QString requestdate =
uplab.value("requestdate").toString();
        QString result = uplab.value("result").toString();
        int row = ui->tableWidget_9->rowCount();
        ui->tableWidget_9->insertRow(row);
        ui->tableWidget_9->setItem(row, 0, new
QTableWidgetItem(name));

        ui->tableWidget_9->setItem(row, 1, new
QTableWidgetItem(testtype));
        ui->tableWidget_9->setItem(row, 2, new
QTableWidgetItem(testtype));
        ui->tableWidget_9->setItem(row, 3, new
QTableWidgetItem(status));
        ui->tableWidget_9->setItem(row, 4, new
QTableWidgetItem(requestdate));
        ui->tableWidget_9->setItem(row, 5, new
QTableWidgetItem(result));

    }
}
}

```

```

void staffwindow::on_pushButton_4_clicked()
{
    conndb();
    if (conndb())
    {
        struct doc {
            QString name;
            QString code;
            QString department;
            QString gender;
            QDate dob;
            qint64 cnic;
            qint64 contact;
            QString address;
        };
        doc de;
        QSqlQuery pa;
    }
}

```

```

pa.prepare("SELECT * FROM dcredentials");

if (pa.exec())
{
    ui->tableWidget_5->clearContents();
    ui->tableWidget_5->setRowCount(0);
    while (pa.next())
    {
        de.name = pa.value("name").toString();
        de.code = pa.value("code").toString();
        de.department = pa.value("department").toString();
        de.gender = pa.value("gender").toString();
        de.cnic = pa.value("cnic").toLongLong();
        de.contact = pa.value("contact").toLongLong();
        de.dob = pa.value("dob").toDate();

        int row = ui->tableWidget_5->rowCount();
        ui->tableWidget_5->insertRow(row);
        ui->tableWidget_5->setItem(row, 0, new
QTableWidgetItem(de.name));
        ui->tableWidget_5->setItem(row, 1, new
QTableWidgetItem(de.code));
        ui->tableWidget_5->setItem(row, 2, new
QTableWidgetItem(de.department));
        ui->tableWidget_5->setItem(row, 3, new
QTableWidgetItem(de.gender));
        ui->tableWidget_5->setItem(row, 4, new
QTableWidgetItem(QString::number(de.cnic)));
        ui->tableWidget_5->setItem(row, 5, new
QTableWidgetItem(de.dob.toString("dd/MM/yyyy")));
        ui->tableWidget_5->setItem(row, 6, new
QTableWidgetItem(QString::number(de.contact)));

        ui->tableWidget_5->update();
        row++;
    }
    ui->tableWidget_5->resizeRowsToContents();
    ui->tableWidget_5->resizeColumnsToContents();
}
else if (!pa.exec())
{

```



```

        QMessageBox::information(this, "Contact Admin", "Query
Failed");
    }

    }

    else if (!conndb())
    {
        QMessageBox::critical(this, "Connection Failure",
"Connection to Database failed");
    }
}

void staffwindow::on_lineEdit_3_textChanged(const QString& arg1)
{
    int tablerow = ui->tableWidget_5->rowCount();
    int tablecolumn = ui->tableWidget_5->columnCount();

    for (int row = 0; row < tablerow; ++row)
    {
        bool matchFound = false;

        for (int column = 0; column < tablecolumn; ++column)
        {
            QTableWidgetItem* item = ui->tableWidget_5->item(row,
column);

            if (item && !item->text().isEmpty() && item-
>text().contains(arg1, Qt::CaseInsensitive))
            {
                // If the cell's non-empty text contains the
search text
                matchFound = true;
                break; // Stop checking other columns for this
row
            }
        }

        // Show or hide the row based on whether a match was found
        ui->tableWidget_5->setRowHidden(row, !matchFound);
    }
}

```

```

void staffwindow::on_pushButton_2_clicked()
{
    if (conndb())
    {
        QDate dob;
        dob = ui->dateEdit->date();
        QSqlQuery up;
        up.prepare("INSERT INTO patient (doctor, name, code,
gender, dob, cnic, contact) VALUES (:docname,:patname,:code,
:gender, :dob, :cnic, :contact)");
        up.bindValue(":docname", ui->assigdoc->currentText());
        up.bindValue(":patname", ui->lineEdit_78->text());
        up.bindValue(":code", patcode);
        up.bindValue(":gender", ui->lineEdit_80->text());
        up.bindValue(":dob", dob.toString("yyyy-MM-dd"));
        up.bindValue(":cnic", ui->lineEdit_77->text());
        up.bindValue(":contact", ui->lineEdit_123->text());
        if (!up.exec())
        {
            qDebug() << "Failed";
        }
    }
}

```

```

void staffwindow::on_patdocdep_currentIndexChanged(int index)
{
    QString department = ui->patdocdep->currentText();
    conndb(); //For connecting Database
    if (conndb()) //If connection successful
    {
        QSqlQuery appo; //Using QSqlQuery to execute the query by
creating the object appo
        appo.prepare("SELECT name FROM dcredentials WHERE
department=:department"); //Retrieves data from database and
selects doctors from the department which is selected
        appo.bindValue(":department", department);
        //binds/replaces value

        ui->assigdoc->setCurrentIndex(0); //sets the index of the
doctor combobox to 0
        if (appo.exec()) //if query executes

```

```

        {
            ui->assigdoc->clear(); // Clear previously set text
before adding items

            while (appo.next()) //if query finds values
            {
                QString name = appo.value("name").toString();
                ui->assigdoc->addItem(name); //Adds the name of
the doctor in the doctors combobox
            }
        }
        else
        {
            qDebug() << "FAiled";
        }
    }
}

```

```

void staffwindow::on_lineEdit_78_editingFinished()
{
    if (!ui->lineEdit_78->text().isEmpty())
    {
        patcode = QUuid::createUuid().toString().mid(1, 8);
        ui->lineEdit_124->setText(patcode);
    }
}

```

```

void staffwindow::on_pushButton_5_clicked()
{
    if (conndb())
    {
        QSqlQuery tab;
        tab.prepare("SELECT * FROM patient");

        if (tab.exec())
        {
            ui->tableWidget_4->clearContents();
            ui->tableWidget_4->setRowCount(0);
            while (tab.next())
            {

                QString doctor = tab.value("doctor").toString();

```

```

        QString patient = tab.value("name").toString();
        QString code = tab.value("code").toString();
        QString gender = tab.value("gender").toString();
        QDate dob = tab.value("dob").toDate();
        QString cnic = tab.value("cnic").toString();
        QString contact = tab.value("contact").toString();
        int row = ui->tableWidget_4->rowCount();
        ui->tableWidget_4->insertRow(row);
        ui->tableWidget_4->setItem(row, 0, new
QTableWidgetItem(patient));
        ui->tableWidget_4->setItem(row, 1, new
QTableWidgetItem(code));
        ui->tableWidget_4->setItem(row, 2, new
QTableWidgetItem(doctor));
        ui->tableWidget_4->setItem(row, 3, new
QTableWidgetItem(dob.toString("dd/MM/yyyy")));
        ui->tableWidget_4->setItem(row, 4, new
QTableWidgetItem(cnic));
        ui->tableWidget_4->setItem(row, 5, new
QTableWidgetItem(gender));
        ui->tableWidget_4->setItem(row, 6, new
QTableWidgetItem(contact));

        ui->tableWidget_4->update();
        row++;
    }
    ui->tableWidget_4->resizeRowsToContents();
    ui->tableWidget_4->resizeColumnsToContents();

}
else if (!tab.exec())
{
    QMessageBox::information(this, "Contact Admin", "Query
Failed");
}

}

else if (!conndb())
{
    QMessageBox::critical(this, "Connection Failure",
"Connection to Database failed");
}
}

```

```

void staffwindow::on_lineEdit_2_textChanged(const QString& arg1)
{
    int tablerow = ui->tableWidget_4->rowCount();
    int tablecolumn = ui->tableWidget_4->columnCount();

    for (int row = 0; row < tablerow; ++row)
    {
        bool matchFound = false;

        for (int column = 0; column < tablecolumn; ++column)
        {
            QTableWidgetItem* item = ui->tableWidget_4->item(row,
column);

            if (item && !item->text().isEmpty() && item-
>text().contains(arg1, Qt::CaseInsensitive))
            {
                // If the cell's non-empty text contains the
search text
                matchFound = true;
                break; // Stop checking other columns for this
row
            }
        }

        // Show or hide the row based on whether a match was found
ui->tableWidget_4->setRowHidden(row, !matchFound);
    }
}

void staffwindow::on_pushButton_6_clicked()
{
    QSqlQuery app;
    app.prepare("SELECT * FROM appointment");
    app.bindValue("doctor", loggedindocname);
    if (app.exec())
    {

        ui->tableWidget_7->clearContents();
        ui->tableWidget_7->setRowCount(0);
        while (app.next())
        {

```

```

        QString doctor = app.value("docname").toString();
        QString patient = app.value("patname").toString();
        QString day = app.value("day").toString();
        QString tim = app.value("tim").toString();
        int row = ui->tableWidget_7->rowCount();
        ui->tableWidget_7->insertRow(row);
        ui->tableWidget_7->setItem(row, 0, new
QTableWidgetItem(patient));
        ui->tableWidget_7->setItem(row, 1, new
QTableWidgetItem(doctor));
        ui->tableWidget_7->setItem(row, 2, new
QTableWidgetItem(day));
        ui->tableWidget_7->setItem(row, 3, new
QTableWidgetItem(tim));

        ui->tableWidget_7->update();
        row++;
    }
    ui->tableWidget_7->resizeRowsToContents();
    ui->tableWidget_7->resizeColumnsToContents();
}
}

```

```

void staffwindow::on_pushButton_7_clicked()
{
    // Assuming conndb() is a function that establishes a database
    connection

    if (conndb())
    {
        QSqlQuery patientQuery("SELECT name FROM patient");

        if (patientQuery.exec())
        {
            // Clear the existing items in the combo box
            ui->comboBill->clear();

            // Iterate through the results and add names to the
            combo box
            while (patientQuery.next())
            {
                QString patientName =
                patientQuery.value("name").toString();
            }
        }
    }
}

```

```

        ui->comboBill->addItem(patientName);
    }
}
else
{
    // Handle query execution error
    qDebug() << "Error executing patient query:";
}
}
else
{
    // Handle database connection error
    qDebug() << "Database connection failed";
}
}

//void staffwindow::on_comboBill_currentIndexChanged(int index)
//{

//    }

```

```

void staffwindow::on_pushButton_8_clicked()
{
    double totalAppPrice = 0.0;
    double totalLabPrice = 0.0;
    double totalPrice = 0.0;
    QString userName = ui->comboBill->currentText();
    QSqlQuery billQuery;
    billQuery.prepare("SELECT department, COUNT(*) AS
numAppointments "
        "FROM appointment "
        "WHERE patname=:patientName "
        "GROUP BY department");
    billQuery.bindValue(":patientName", userName);

    if (billQuery.exec()) {

        while (billQuery.next()) {
            QString department =
billQuery.value("department").toString();
            int numAppointments =
billQuery.value("numAppointments").toInt();

            // Process the price based on the department

```

```

double deptPrice = 0.0;

if (department == "Cardiology")
    deptPrice = 1500.0;
else if (department == "Dermatology")
    deptPrice = 1250.0;
else if (department == "Emergency Medicine")
    deptPrice = 1200.0;
else if (department == "Endocrinology")
    deptPrice = 1300.0;
else if (department == "Gastroenterology")
    deptPrice = 1450.0;
else if (department == "General Surgery")
    deptPrice = 1350.0;
else if (department == "Internal Medicine")
    deptPrice = 1550.0;
else if (department == "Neurology")
    deptPrice = 1800.0;
else if (department == "Obstetrics and Gynecology")
    deptPrice = 1900.0;
else if (department == "Oncology")
    deptPrice = 1850.0;
else if (department == "Ophthalmology")
    deptPrice = 1750.0;
else if (department == "Orthopedics")
    deptPrice = 1550.0;
else if (department == "Pediatrics")
    deptPrice = 1100.0;
else if (department == "Psychiatry")
    deptPrice = 1700.0;
else if (department == "Pulmonary Medicine")
    deptPrice = 2000.0;
else if (department == "Rheumatology")
    deptPrice = 1600.0;
else if (department == "Urology")
    deptPrice = 1650.0;

// Calculate total price for the specific department
double totalPriceForApp = numAppointments * deptPrice;
totalAppPrice += totalPriceForApp;

// Display or use information as needed
qDebug() << "Department: " << department << ", Num
Appointments: " << numAppointments << ", Total Price: " <<
totalPriceForApp;
}

```



```

        qDebug() << "Total Appointments Price: " << totalAppPrice;
    }
    else {
        // Handle query execution error
        qDebug() << "Error executing appointment query:";
    }

    if (conndb())
    {
        QSqlQuery labQuery;
        labQuery.prepare("SELECT testtype, COUNT(*) AS numTests "
            "FROM lab "
            "WHERE patname = :userName "
            "GROUP BY testtype");
        labQuery.bindValue(":userName", userName);

        if (labQuery.exec())
        {

            while (labQuery.next())
            {
                QString testType =
labQuery.value("testtype").toString();
                int testCount =
labQuery.value("numTests").toInt();

                // Process the price based on the test type
                double testPrice = 0.0;

                if (testType == "CBC")
                    testPrice = 500.0;
                else if (testType == "BMP")
                    testPrice = 300.0;
                else if (testType == "Blood Type")
                    testPrice = 250.0;
                else if (testType == "Urinalysis")
                    testPrice = 350.0;
                else if (testType == "Allergy Test")
                    testPrice = 320.0;
                else if (testType == "Hormone Test")
                    testPrice = 400.0;
                else if (testType == "ECG")
                    testPrice = 450.0;
                else if (testType == "X-Ray")

```

```

        testPrice = 700.0;
    else if (testType == "Ultrasound")
        testPrice = 650.0;
    else if (testType == "CT Scan")
        testPrice = 600.0;
    else if (testType == "Glucose Test")
        testPrice = 550.0;
    else if (testType == "Covid Test")
        testPrice = 150.0;
    // Add other test types...

    // Calculate total price for the specific test
type
    double totalPriceForTest = testPrice * testCount;
    totalLabPrice += totalPriceForTest;

    // Display or use information as needed
    qDebug() << "Test Type: " << testType << ", Count:
" << testCount << ", Total Price: " << totalPriceForTest;

    }

    qDebug() << "Total Lab Price: " << totalLabPrice;
}
else
{
    // Handle query execution error
    qDebug() << "Error executing lab query:";
}
}
else
{
    // Handle database connection error
    qDebug() << "Database connection failed";
}
QString resultText = "Patient Name: " + userName +
    "\n\nTotal Appointments Price: " +
QString::number(totalAppPrice) +
    "\n\nTotal Lab Price: " + QString::number(totalLabPrice);
ui->textBrowser_19->setText(resultText);
totalPrice = totalAppPrice + totalLabPrice;
QString fullbill = "\nYour Full Bill is: " +
QString::number(totalPrice);
ui->textBrowser_20->setText(fullbill);
}

```

Doctor Account:

```
#include "docwindow.h"
#include "ui_docwindow.h"
#include<QSqlDatabase>
#include<QSqlQuery>
#include<QString>
#include<QDebug>
#include<QMessageBox>
#include<QDate>
#include<globalvar.h>
#include<mainwindow.h>
#include<QUuid>
#include<QFile>
#include<QTime>
#include<QTimer>
#include <QMovie>
bool conndb();

docwindow::docwindow(QWidget* parent) :
    QDialog(parent),
    ui(new Ui::docwindow)
{
    ui->setupUi(this);
    //for gifs in the program
    QMovie* gif = new
    QMovie("C:/Users/hstech/Downloads/resource/sparkle.gif");
    ui->labelGif->setMovie(gif);
    gif->start();
    ui->labelGif->setScaledContents(true);

    QMovie* gif3 = new
    QMovie("C:/Users/hstech/Downloads/resource/tree.gif");
    ui->labelGif_3->setMovie(gif3);
    gif3->start();
    ui->labelGif_3->setScaledContents(true);

    QMovie* gif4 = new
    QMovie("C:/Users/hstech/Downloads/resource/tree.gif");
    ui->labelGif_4->setMovie(gif4);  //
    gif4->start();
```

```

ui->labelGif_4->setScaledContents(true);
    ui->tabWidget->setCurrentIndex(0);
}

docwindow::~docwindow()
{
    delete ui;
}

void docwindow::on_tabWidget_tabBarClicked(int index)
{
    struct doc {
        QString name;
        QString department;
        QString gender;
        QDate dob;
        qint64 cnic;
        qint64 contact;
        QString address;
    };

    conndb();
    QString tempmail = loggedindocemail;
    doc de;

    if (conndb())
    {
        QSqlQuery ba;
        ba.prepare("SELECT name, department, gender, dob, cnic,
contact, address FROM dcredentials WHERE email=:email");
        ba.bindValue(":email", tempmail);

        if (ba.exec() && ba.next()) {

            de.name = ba.value("name").toString();
            de.department = ba.value("department").toString();
            de.gender = ba.value("gender").toString();
            de.cnic = ba.value("cnic").toLongLong();
            de.dob = ba.value("dob").toDate();
            de.contact = ba.value("contact").toLongLong();
            de.address = ba.value("address").toString();

            QString resultText = "Name: " + de.name + "\n\n"
                + "Department: " + de.department + "\n\n"

```

```

        + "Gender: " + de.gender + "\n\n"
        + "CNIC: " + QString::number(de.cnic) + "\n\n"
        + "DOB: " + (de.dob.toString("dd/MM/yyyy")) +
"\n\n"
        + "Contact: " + QString::number(de.contact) +
"\n\n";

    ui->textBrowser_7->setText(resultText);
    qDebug() << de.dob.toString("yyyy/MM/dd");
    qDebug() << de.cnic;

}
else {
    QMessageBox::information(this, "Contact Admin", "Query
Failed");
}

qint32 k = 0;
double hoursworked;
double totalhours = 0;
QString salary;
salary.prepare("SELECT * FROM salary WHERE
name=:loggedname");
salary.bindValue(":loggedname", loggedindocname);
if (salary.exec())
{
    while (salary.next())
    {
        hoursworked =
salary.value("hoursworked").toDouble();
        totalhours = totalhours + hoursworked;

        k++;
    }
    QString text = "No. Of Days Worked: " +
QString::number(k) +
"\n\nTotal Hours Worked: " +
QString::number(totalhours);
    ui->textBrowser_salary->setText(text);

}
if (conndb()) {
    // Retrieve department from dcredentials table
    QString department;
    QSqlQuery deptQuery;

```

```

        deptQuery.prepare("SELECT department FROM dcredentials
WHERE name=:loggedinname");
        deptQuery.bindValue(":loggedinname", loggedindocname);

        if (deptQuery.exec() && deptQuery.next()) {
            department =
deptQuery.value("department").toString();
        }
        else {
            // Handle query execution error or no department
found
            qDebug() << "Error retrieving department for
doctor: " << loggedindocname;
            // You may want to set a default department or
handle this case appropriately
        }

        // Use an if condition for each department to set
deptPrice
        double deptPrice = 0.0;
        if (department == "Cardiology")
            deptPrice = 15;
        else if (department == "Dermatology")
            deptPrice = 12;
        else if (department == "Emergency Medicine")
            deptPrice = 12;
        else if (department == "Endocrinology")
            deptPrice = 13;
        else if (department == "Gastroenterology")
            deptPrice = 14;
        else if (department == "General Surgery")
            deptPrice = 13.5;
        else if (department == "Internal Medicine")
            deptPrice = 15.5;
        else if (department == "Neurology")
            deptPrice = 18;
        else if (department == "Obstetrics and Gynecology")
            deptPrice = 19;
        else if (department == "Oncology")
            deptPrice = 18.5;
        else if (department == "Ophthalmology")
            deptPrice = 17.5;
        else if (department == "Orthopedics")
            deptPrice = 15.5;
        else if (department == "Pediatrics")
            deptPrice = 11;

```

```

        else if (department == "Psychiatry")
            deptPrice = 17;
        else if (department == "Pulmonary Medicine")
            deptPrice = 20;
        else if (department == "Rheumatology")
            deptPrice = 16;
        else if (department == "Urology")
            deptPrice = 16.5;

        double totalsalary = totalhours * deptPrice;
        QString salary = "Your Total Salary is: " +
        QString::number(totalsalary);
        ui->textBrowser_salary2->setText(salary);

    }

}

else if (!conndb()) {
    QMessageBox::information(this, "Error", "Database
connection failed");
}
ui->comboBox_3->clear();
 QSqlQuery labr;
    labr.prepare("SELECT name FROM patient WHERE
doctor=:docname");
    labr.bindValue(":docname", loggedindocname);
    if (labr.exec())
    {
        while (labr.next())
        {
            QString patname = labr.value("name").toString();

            ui->comboBox_3->addItem(patname);
        }
    }
else
{
    qDebug() << "Query Failed";
}

}

```

```

void docwindow::on_pushButton_2_clicked()
{
    conndb();
    if (conndb())
    {
        struct pat {
            QString name;
            qint64 code;
            QString gender;
            QDate dob;
            qint64 cnic;
            qint64 contact;
            QString address;
        };
        pat p;
        QString doctor = loggedindocname;
        qDebug() << doctor;
        QSqlQuery pa;
        pa.prepare("SELECT name,code,gender,cnic,contact,dob FROM
patient WHERE doctor=:doctor");
        pa.bindValue(":doctor", doctor);
        if (pa.exec())
        {
            ui->tableWidget_4->clearContents();
            ui->tableWidget_4->setRowCount(0);
            while (pa.next())
            {
                p.name = pa.value("name").toString();
                p.code = pa.value("code").toLongLong();
                p.gender = pa.value("gender").toString();
                p.cnic = pa.value("cnic").toLongLong();
                p.contact = pa.value("contact").toLongLong();
                p.dob = pa.value("dob").toDate();

                int row = ui->tableWidget_4->rowCount();
                ui->tableWidget_4->insertRow(row);
                ui->tableWidget_4->setItem(row, 0, new
QTableWidgetItem(p.name));
                ui->tableWidget_4->setItem(row, 1, new
QTableWidgetItem(QString::number(p.code)));
            }
        }
    }
}

```



```

        ui->tableWidget_4->setItem(row, 2, new
QTableWidgetItem(p.gender));
        ui->tableWidget_4->setItem(row, 3, new
QTableWidgetItem(p.dob.toString("dd/MM/yyyy")));
        ui->tableWidget_4->setItem(row, 4, new
QTableWidgetItem(QString::number(p.cnic)));
        ui->tableWidget_4->setItem(row, 5, new
QTableWidgetItem(QString::number(p.contact)));

        ui->tableWidget_4->update();
        row++;
    }
    ui->tableWidget_4->resizeRowsToContents();
    ui->tableWidget_4->resizeColumnsToContents();

}
else if (!pa.exec())
{
    QMessageBox::information(this, "Contact Admin", "Query
Failed");
}

}

else if (!conndb())
{
    QMessageBox::critical(this, "Connection Failure",
"Connection to Database failed");
}

}

void docwindow::on_lineEdit_textChanged(const QString& arg1)
{
    int tablerow = ui->tableWidget_4->rowCount();
    int tablecolumn = ui->tableWidget_4->columnCount();

    for (int row = 0; row < tablerow; ++row)
    {
        bool matchFound = false;

        for (int column = 0; column < tablecolumn; ++column)
        {

```

```

        QTableWidgetItem* item = ui->tableWidget_4->item(row,
column);

        if (item && !item->text().isEmpty() && item-
>text().contains(arg1, Qt::CaseInsensitive))
        {
            // If the cell's non-empty text contains the
search text
            matchFound = true;
            break; // Stop checking other columns for this
row
        }
    }

    // Show or hide the row based on whether a match was found
    ui->tableWidget_4->setRowHidden(row, !matchFound);
}
}
void docwindow::on_pushButton_clicked()
{
    QString currenthour = QTime::currentTime().toString("hh");
    QString currentminute = QTime::currentTime().toString("mm");
    QString currentsec = QTime::currentTime().toString("ss");

    double hourtosec = currenthour.toInt() * 3600.0;
    double minutetosec = currentminute.toInt() * 60.0;
    int sec = currentsec.toInt();
    dclogoutt = hourtosec + minutetosec + sec;
    qDebug() << "Before: " << dclogoutt;
    dclogoutt = dclogoutt / 3600.0;
    double diff = dclogoutt - dclogint;
    QString workday = QDate::currentDate().toString("yyyy-MM-dd");
    conndb();
    QSqlQuery work;

    work.prepare("SELECT hoursworked FROM salary WHERE
name=:staffname AND date=:workday AND hoursworked IS NOT NULL");
    work.bindValue(":staffname", loggedindocname); // Replace
staffname with loggedindocname
    work.bindValue(":workday", workday);

    if (work.exec() && work.next())
    {
        double storedhours = work.value(0).toDouble();
    }
}

```

```

        work.prepare("UPDATE salary SET hoursworked = :diff WHERE
name = :staffname AND date = :workday");
        work.bindValue(":staffname", loggedindocname); // Replace
staffname with loggedindocname
        work.bindValue(":workday", workday);
        work.bindValue(":diff", diff + storedhours);
        if (!work.exec())
        {
            qDebug() << "UPDATE failed:";
        }
    }
    else
    {
        work.prepare("INSERT INTO salary (name, hoursworked, date)
VALUES (:name, :hoursworked, :date)");
        work.bindValue(":name", loggedindocname); // Replace
staffname with loggedindocname
        work.bindValue(":hoursworked", diff);
        work.bindValue(":date", workday);
        if (!work.exec())
        {
            qDebug() << "INSERT failed:";
        }
    }
    close();
    MainWindow* mw = new MainWindow;
    mw->show();
}

```

```

void docwindow::on_pushButton_3_clicked()
{
    if (conndb())
    {
        QSqlQuery schedule;
        schedule.prepare("SELECT patname, day, tim FROM
appointment WHERE docname=:docname");
        schedule.bindValue(":docname", loggedindocname);

        if (schedule.exec())
        {
            while (schedule.next())
            {
                QString patname =
schedule.value("patname").toString();

```

```

        QString day = schedule.value("day").toString();
        QString tim = schedule.value("tim").toString();

        qDebug() << patname;
        qDebug() << day;
        qDebug() << tim;

        // Access the text in the row header of the
specified row
        int rowToAccess = 1; // Replace with the desired
row index
        QString rowHeaderText = ui->tableWidget_2-
>verticalHeaderItem(rowToAccess)->text();
        qDebug() << "Row Header Text:" << rowHeaderText;

        // Access the text in the column header of the
specified column
        int colToAccess = 1; // Replace with the desired
column index
        QString colHeaderText = ui->tableWidget_2-
>horizontalHeaderItem(colToAccess)->text();
        qDebug() << "Column Header Text:" <<
colHeaderText;

        for (int row = 0; row < ui->tableWidget_2-
>rowCount(); row++)
        {
            if (ui->tableWidget_2-
>verticalHeaderItem(row)->text() == day)
            {
                for (int col = 0; col < ui->tableWidget_2-
>columnCount(); col++)
                {
                    if (ui->tableWidget_2-
>horizontalHeaderItem(col)->text() == tim)
                    {
                        ui->tableWidget_2->setItem(row,
col, new QTableWidgetItem(patname));
                    }
                }
            }
        }
    }
}

```

```

    }
    else
    {
        QMessageBox::warning(this, "Failed", "Query Failed");
    }
}
else
{
    QMessageBox::warning(this, "Failed", "Database Connection
Failed");
}
}

```

```

void docwindow::on_pushButton_4_clicked()
{
    QString test;
    if (ui->radioButton->isChecked())
    {
        test = ui->radioButton->text();
    }
    else if (ui->radioButton_2->isChecked())
    {
        test = ui->radioButton_2->text();
    }
    else if (ui->radioButton_3->isChecked())
    {
        test = ui->radioButton_3->text();
    }
    else if (ui->radioButton_7->isChecked())
    {
        test = ui->radioButton_7->text();
    }
    else if (ui->radioButton_8->isChecked())
    {
        test = ui->radioButton_8->text();
    }
    else if (ui->radioButton_9->isChecked())
    {
        test = ui->radioButton_9->text();
    }
    else if (ui->radioButton_10->isChecked())
    {
        test = ui->radioButton_10->text();
    }
    else if (ui->radioButton_11->isChecked())
    {

```

```

        test = ui->radioButton_11->text();
    }
    else if (ui->radioButton_12->isChecked())
    {
        test = ui->radioButton_12->text();
    }
    else if (ui->radioButton_13->isChecked())
    {
        test = ui->radioButton_13->text();
    }
    else if (ui->radioButton_14->isChecked())
    {
        test = ui->radioButton_14->text();
    }
    else if (ui->radioButton_15->isChecked())
    {
        test = ui->radioButton_15->text();
    }
    QSqlQuery up;
    up.prepare("INSERT INTO lab
(docname,patname,testtype,status,requestdate)
VALUES(:doctor,:patient,:test,:status,:date)");
    up.bindValue(":doctor", loggedindocname);
    up.bindValue(":patient", ui->comboBox_3->currentText());
    up.bindValue(":test", test);
    up.bindValue(":status", "Pending");
    up.bindValue(":date", QDate::currentDate());
    if (!up.exec())
    {
        qDebug() << "Failed";
    }
}

void docwindow::on_pushButton_5_clicked()
{
    if (conndb())
    {
        QSqlQuery uplab;
        uplab.prepare("SELECT * FROM lab WHERE docname=:docname");
        uplab.bindValue(":docname", loggedindocname);
        if (uplab.exec())
        {
            ui->tableWidget_3->clearContents();

```

```

        ui->tableWidget_3->setRowCount(0);

        while (uplab.next())
        {
            QString name = uplab.value("patname").toString();

            QString testtype =
uplab.value("testtype").toString();
            QString status = uplab.value("status").toString();
            QString requestdate =
uplab.value("requestdate").toString();
            QString result = uplab.value("result").toString();
            int row = ui->tableWidget_3->rowCount();
            ui->tableWidget_3->insertRow(row);
            ui->tableWidget_3->setItem(row, 0, new
QTableWidgetItem(name));
            ui->tableWidget_3->setItem(row, 1, new
QTableWidgetItem(testtype));
            ui->tableWidget_3->setItem(row, 2, new
QTableWidgetItem(status));
            ui->tableWidget_3->setItem(row, 3, new
QTableWidgetItem(requestdate));
            ui->tableWidget_3->setItem(row, 4, new
QTableWidgetItem(result));

        }
    }
}

```

```

void docwindow::on_pushButton_11_clicked()
{
    QFile file("docnotes.txt");
    if (file.open(QIODevice::Append | QIODevice::Text)) {
        QTextStream stream(&file);

        // Get current date and day
        QString notes = ui->textEdit->toPlainText();
        QDateTime currentDate = QDateTime::currentDateTime();
        QString dateString = currentDate.toString("yyyy-MM-dd");
        QString dayString = currentDate.toString("dddd");
    }
}

```

```

        // Write notes to the file with date and day
        stream << "Date: " << dateString << ", Day: " << dayString
<< "\n";
        stream << notes << "\n";

        // Add 4 lines as separators
        for (int i = 0; i < 4; ++i) {
            stream << "\n";
        }

        file.close();
    }
    else {
        // Handle file opening error
        qDebug() << "Error opening file for writing";
    }
}

void docwindow::on_pushButton_6_clicked()
{
    QFile file("docnotes.txt");
    QString notes;

    if (file.open(QIODevice::ReadOnly | QIODevice::Text)) {
        QTextStream stream(&file);
        notes = stream.readAll();
        file.close();
        ui->textEdit->setText(notes);
    }
    else {
        // Handle file opening error
        qDebug() << "Error opening file for reading";
    }
}

```

Conclusion

Our Hospital Management System project is like a helpful sidekick for hospitals. It makes things easier by organizing information, making communication smoother, and generally tidying up the daily tasks. We didn't just write code, we made something practical. Imagine it as a handy tool for hospitals to upgrade how they handle stuff. It's all about making patient care better and helping hospitals run more smoothly. We hope it helps people.

Links

https://github.com/Ezi0567/Hospital_Management_System-QT-6.6.0

The youtube video of the working program is uploaded on the form.

References

<https://www.w3schools.com/css/>

<https://doc.qt.io/>

<https://www.w3schools.com/sql/default.asp>

<https://www.khanacademy.org/computing/computer-programming/sql/sql-basics/v/welcome-to-sql>