DR DOCS - PATIENT HEALTH MANAGEMENT SYSTEM

A MINI-PROJECT REPORT

Submitted   by

YOKESHWAR S   **220701329**

SHIIV R S   **220701331**

In partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE

RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS)

THANDALAM

CHENNAI-602105

2023 - 24

BONAFIDE CERTIFICATE

Certified that this project report "**LIBRARY MANAGEMENT SYSTEM**" is the bonafide work of

"YOKESHWAR S(220701329), SHIIV R S (220701331) "

who carried out the project work under my supervision.

Submitted for the Practical Examination held on _____

SIGNATURE	SIGNATURE
Dr.R.SABITHA	**Ms.V.JANANEE**
Professor and II Year Academic Head	**Assistant Professor (SG),**
Computer Science and Engineering,	**Computer Science and Engineering,**
Rajalakshmi Engineering College	**Rajalakshmi Engineering College,**
(Autonomous),	**(Autonomous),**
Thandalam, Chennai - 602 105	**Thandalam, Chennai - 602 105**

INTERNAL EXAMINER   **EXTERNAL EXAMINER**

ABSTRACT

DR DOCS is a Patient Health Management System built using Python, PyQt5, and MySQL. It aims to simplify patient health records in a healthcare system, providing a user-friendly interface for doctors and patients. The system helps streamline the process of data entry, updates, and accessing medical information.

The application includes a secure login system that distinguishes between doctors and patients, ensuring each user has the appropriate level of access. Doctors can create new patient records, update existing entries, and manage comprehensive health data such as medical history, surgeries, allergies, symptoms, diagnoses, tests, and medications. Patients are prompted to update their username and password upon their first login to enhance security, and they can view their medical reports afterward.

The system uses a MySQL database to store user credentials and patient information securely. This ensures that the data is reliable and can handle the needs of a busy healthcare environment.

Overall, this project demonstrates how combining modern GUI development with a reliable database can create a practical solution for managing patient health records. It showcases the potential for further improvements in functionality and security, making it a useful tool for healthcare providers.

TABLE OF CONTENTS

CHAPTER 1
1. INTRODUCTION

DR DOCS is an initiative aimed at revolutionizing the way healthcare institutions handle patient health records. Developed using Python, PyQt5, and MySQL, this system provides a robust platform for doctors and patients for efficient management of medical data.

The traditional paper-based systems often pose challenges in terms of accessibility, security, and data management. To address these issues, this project introduces a digital solution that streamlines the process of recording, updating, and accessing patient health information.

By leveraging modern technologies such as PyQt5 for the graphical user interface and MySQL for the backend database, the system offers a user-friendly experience while ensuring data integrity and security. Doctors can create new patient entries, update existing records, and access comprehensive health data with ease. Patients, on the other hand, can securely access their medical reports and update their credentials for added security.

This project represents a significant step forward in healthcare data management, providing a scalable and efficient solution for healthcare providers. It demonstrates the potential of technology to improve patient care and streamline administrative processes in the healthcare industry.

2. Objectives

1. Efficient Data Management: Develop a system for recording, updating, and retrieving patient health information to streamline healthcare processes.
2. User Authentication and Role Management: Implement secure login and role assignment to ensure data privacy and security.
3. Doctor Functionality: Enable doctors to create and update patient records, accessing comprehensive health data.
4. Patient Functionality: Provide patients with secure access to medical reports and credential updates.
5. Database Integration: Integrate MySQL for secure storage and management of user credentials and patient information.

3. Modules

1. Login: Handles authentication and role assignment.

2. Doctor: Allows doctors to manage patient records and health data.

3. Patient: Provides patients with access to medical reports and credential updates.

4. Database: Integrates MySQL for secure data storage and retrieval.

5. GUI: Implements the user interface for intuitive interaction.

6. Security: Ensures data protection through encryption and secure transmission

CHAPTER 2

1. Software Description

The Patient Health Management System is a comprehensive software solution designed to streamline the management of patient health records within healthcare institutions. Developed using Python, PyQt5, and MySQL, the system provides a user-friendly graphical interface for both doctors and patients, facilitating efficient data entry, update, and retrieval of medical information. Through secure login authentication and role management, the software ensures data privacy and security, allowing doctors to create new patient entries, update existing records, and access detailed health data. Patients can securely access their medical reports and update their credentials for added security. With robust database integration, the system stores user credentials and patient information securely, ensuring reliability and scalability in managing healthcare data.

2. Software Description

2.1 Python

Python, renowned for its simplicity and versatility, stands as a premier programming language in various domains, including web development, data science, artificial intelligence, and more. With its clean syntax and extensive libraries, Python facilitates rapid development and prototyping, making it an ideal choice for building complex systems like the Patient Health Management System. Its vast ecosystem of frameworks and libraries, such as PyQt5 for GUI development, provides robust tools for creating interactive user interfaces. Python's compatibility with multiple platforms ensures the portability of applications across different operating systems, further enhancing its appeal for software development projects.

2.2 MySQL

MySQL, a popular open-source relational database management system (RDBMS), has long been a cornerstone in the realm of database technology. Praised for its scalability, performance, and reliability, MySQL serves as an excellent choice for storing and managing structured data in various applications. Its comprehensive feature set includes support for transactions, replication, and security mechanisms, ensuring the integrity and confidentiality of data. MySQL's compatibility with multiple programming languages, including Python, makes it a versatile option for integrating with different software solutions. With its robust architecture and active community support, MySQL continues to be a top choice for database management in diverse industries, including healthcare.

3. Requirements And Analysis

3.1 Requirements Specification

Project Overview:

The Patient Health Management System streamlines patient health record management for doctors and patients. It includes secure login, patient record creation/updating for doctors, and secure medical report access for patients.

Functional Requirements:

- **Authentication and Role Management:** Secure login with role differentiation.
- **Doctor Functionality:** Creation/updating of patient records.
- **Patient Functionality:** Secure access to medical reports.
- **Database Management:** Integration with MySQL for data storage.

Non-Functional Requirements:

- **Security:** Password hashing, encryption, and secure data transmission.
- **Usability:** User-friendly interface using PyQt5.
- **Performance:** Optimized database queries.
- **Scalability:** Design for future growth and feature expansion.

Analysis:

- **User Workflow:** Tasks analysis for doctors and patients.
- **Data Model:** Efficient database schema design.

3.2 Hardware and Software Requirements

Hardware:
- Standard desktop or laptop computer with adequate processing power and memory.
- Stable internet connection for accessing the system.

Software:
- Operating System: Compatible with Windows, macOS, or Linux.
- Python: Version 3.6 or above installed on the system.
- PyQt5: Installed for GUI development.
- MySQL: Installed for database management.
- Text editor or Integrated Development Environment (IDE) for code development.
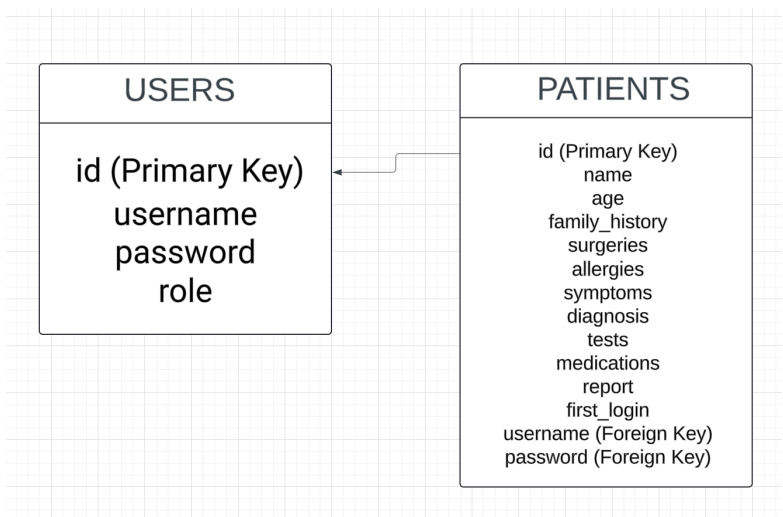
3.3 Architecture Diagram
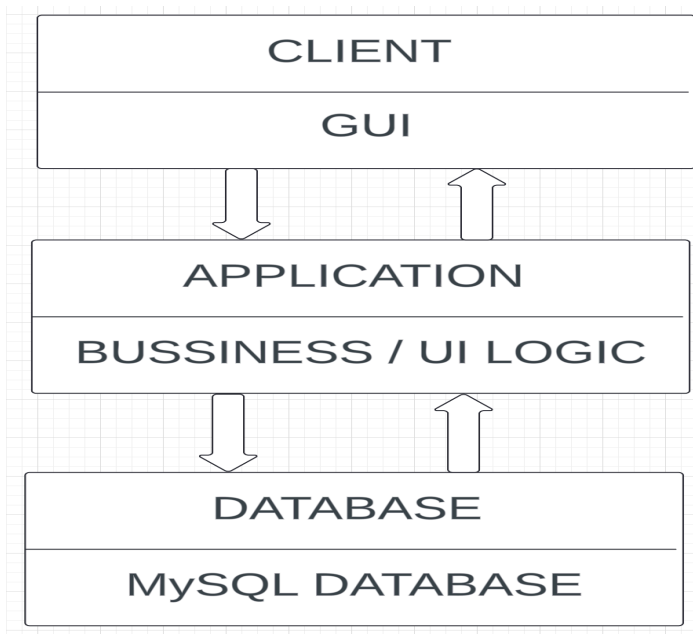
Client: The user interface where doctors and patients interact with the system. It includes the graphical user interface (GUI) implemented using PyQt5.
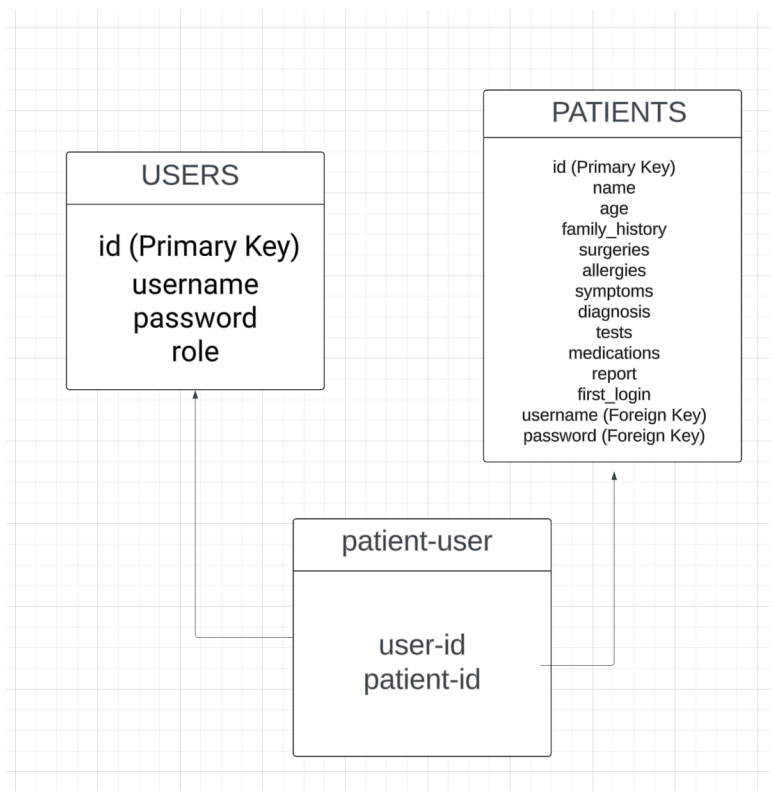
Application: The core of the system where business logic and user interface logic reside. It handles user requests, processes data, and interacts with the database.

Database: The MySQL database where patient records, user credentials, and other relevant data are stored securely. It ensures data integrity, reliability, and scalability.

3.4. ER Diagram

USERS

id (Primary Key)
username
password
role

PATIENTS

id (Primary Key)
name
age
family_history
surgeries
allergies
symptoms
diagnosis
tests
medications
report
first_login
username (Foreign Key)
password (Foreign Key)

3.5. Normalization

PATIENTS

id (Primary Key)
name
age
family_history
surgeries
allergies
symptoms
diagnosis
tests
medications
report
first_login
username (Foreign Key)
password (Foreign Key)

USERS

id (Primary Key)
username
password
role

patient-user

user-id
patient-id

4. PROGRAM CODE

Python:

```python
import sys

from PyQt5.QtCore import Qt

from PyQt5.QtWidgets import QApplication, QWidget, QLabel, QLineEdit,
QPushButton, QVBoxLayout, QMessageBox, QTextEdit, QRadioButton

from PyQt5.QtGui import QPixmap

import mysql.connector

import random

import string


# Establish MySQL connection

db_connection = mysql.connector.connect(

    host="your_hostname",

    user="your_username",

    password="your_password",

    database="your_dbname"

)    # Replace "your_password" with your MySQL password and update the
database name if necessary

cursor = db_connection.cursor()


# Global variables to store windows

login_window = None

doctor_window = None

patient_window = None


def show_login_window():

    global login_window
```

```python
    if login_window is None:

        login_window = LoginWindow()

        login_window.show()


class LoginWindow(QWidget):

    def __init__(self):

        super().__init__()

        self.setWindowTitle("Login")

        self.setGeometry(500, 200, 500, 500)



        layout = QVBoxLayout()



        # Background image setup

        background_label = QLabel(self)

        pixmap = QPixmap("bg-image.jpeg")  # Provide the path to your
background image

        background_label.setPixmap(pixmap)

        background_label.setGeometry(0, 0, 500, 500)  # Set geometry
to cover the window

        background_label.setScaledContents(True)  # Scale the image to
fit the label

        heading_label = QLabel("DR DOCS", self)

        heading_label.setStyleSheet("font-size: 24px;color:white;")  #
Set font size

        heading_label.setAlignment(Qt.AlignCenter)  # Center align the
heading

        layout.addWidget(heading_label)

        username_label = QLabel("Username:")

        username_label.setStyleSheet("color:white;")  # Set font size

        self.username_input = QLineEdit()
```

```python
        layout.addWidget(username_label)

        layout.addWidget(self.username_input)


        password_label = QLabel("Password:")

        password_label.setStyleSheet("color:white;")

        self.password_input = QLineEdit()

        self.password_input.setEchoMode(QLineEdit.Password)

        layout.addWidget(password_label)

        layout.addWidget(self.password_input)


        role_label = QLabel("Role:")

        role_label.setStyleSheet("color:white;")

        layout.addWidget(role_label)


        self.doctor_radio = QRadioButton("Doctor")

        self.doctor_radio.setStyleSheet("color:white;")

        self.patient_radio = QRadioButton("Patient")

        self.patient_radio.setStyleSheet("color:white;")

        layout.addWidget(self.doctor_radio)

        layout.addWidget(self.patient_radio)


        login_button = QPushButton("Login")

        login_button.clicked.connect(self.login)

        layout.addWidget(login_button)


        self.setLayout(layout)
    def login(self):
```

```python
        username = self.username_input.text()

        password = self.password_input.text()

        role = "Doctor" if self.doctor_radio.isChecked() else
"Patient"

        global cursor

        cursor.execute("SELECT * FROM users WHERE username = %s AND
password = %s AND role = %s", (username, password, role.lower()))

        user = cursor.fetchone()


        if user:

            if role.lower() == 'doctor':

                open_doctor_window()

            elif role.lower() == 'patient':

                cursor.execute("SELECT first_login FROM patients WHERE
username = %s", (username,))

                first_login = cursor.fetchone()[0]

                if first_login:

                    open_change_credentials_window(username)

                else:

                    open_patient_window(username)

        else:

            QMessageBox.warning(None, "Login Failed", "Invalid
username, password, or role.")


class ChangeCredentialsWindow(QWidget):

    def __init__(self, username):

        super().__init__()

        self.username = username

        self.setWindowTitle("Change Username and Password")
```

```python
        self.setGeometry(100, 100, 300, 200)

        layout = QVBoxLayout()

        new_username_label = QLabel("New Username:")
        self.new_username_input = QLineEdit()
        layout.addWidget(new_username_label)
        layout.addWidget(self.new_username_input)

        new_password_label = QLabel("New Password:")
        self.new_password_input = QLineEdit()
        self.new_password_input.setEchoMode(QLineEdit.Password)
        layout.addWidget(new_password_label)
        layout.addWidget(self.new_password_input)

        save_button = QPushButton("Save")
        save_button.clicked.connect(self.save_credentials)
        layout.addWidget(save_button)

        self.setLayout(layout)

    def save_credentials(self):
        new_username = self.new_username_input.text()
        new_password = self.new_password_input.text()

        global cursor, db_connection
        cursor.execute("UPDATE users SET username = %s, password = %s
WHERE username = %s", (new_username, new_password, self.username))
```

```python
        cursor.execute("UPDATE patients SET username = %s, password =
%s, first_login = %s WHERE username = %s", (new_username,
new_password, False, self.username))

        db_connection.commit()


        QMessageBox.information(self, "Success", "Credentials updated
successfully.")

        self.close()

        open_patient_window(new_username)


def open_change_credentials_window(username):

    global change_credentials_window

    change_credentials_window = ChangeCredentialsWindow(username)

    change_credentials_window.show()




def open_doctor_window():

    global login_window, doctor_window

    login_window.close()  # Close the login window if open

    doctor_window = DoctorWindow()

    doctor_window.show()


class DoctorWindow(QWidget):

    def __init__(self):

        super().__init__()

        self.setWindowTitle("Doctor Options")

        self.setGeometry(100, 100, 600, 400)


        layout = QVBoxLayout()
```

```python
        new_entry_button = QPushButton("Make New Patient Entry")

        new_entry_button.clicked.connect(make_new_entry)

        layout.addWidget(new_entry_button)


        update_entry_button = QPushButton("Update Old Patient Entry")

        update_entry_button.clicked.connect(update_entry)

        layout.addWidget(update_entry_button)


        self.setLayout(layout)


def open_patient_window(username):

    global login_window, patient_window

    login_window.close()  # Close the login window if open

    patient_window = PatientWindow(username)

    patient_window.show()


class PatientWindow(QWidget):

    def __init__(self, username):

        super().__init__()

        self.setWindowTitle("Patient Report")

        self.setGeometry(100, 100, 600, 400)


        layout = QVBoxLayout()


        # Fetch patient report from the database based on username

        global cursor
```

```python
        cursor.execute("SELECT * FROM patients WHERE username = %s",
(username,))

        patient_data = cursor.fetchone()

        report_label = QLabel("PATIENT REPORT")

        layout.addWidget(report_label)


        if patient_data:

            heading_label = QLabel("Patient Report")

            heading_label.setStyleSheet('font-size:24px;')

            report_label = QLabel()

            report_label.setText(f"Name: {patient_data[1]}\n"

                                 f"Age: {patient_data[2]}\n"

                                 f"Family History of Diseases:
{patient_data[3]}\n"

                                 f"Previous Surgeries:
{patient_data[4]}\n"

                                 f"Allergies: {patient_data[5]}\n"

                                 f"Symptoms: {patient_data[6]}\n"

                                 f"Diagnosis: {patient_data[7]}\n"

                                 f"Tests: {patient_data[8]}\n"

                                 f"Medications: {patient_data[9]}\n"

                                 f"Report: {patient_data[10]}")

            layout.addWidget(report_label)

        else:

            QMessageBox.warning(None, "Error", "Patient data not
found.")


        self.setLayout(layout)

new_entry_window = None
```

```python
update_window = None


def make_new_entry():

    global new_entry_window

    new_entry_window = QWidget()

    new_entry_window.setWindowTitle("New Patient Entry")

    new_entry_window.setGeometry(100, 100, 400, 400)


    layout = QVBoxLayout()

    # Labels and text edits for patient details

    #name

    name_label = QLabel("Name:")

    name_edit = QLineEdit()

    layout.addWidget(name_label)

    layout.addWidget(name_edit)

    #age

    age_label = QLabel("Age:")

    age_edit = QLineEdit()

    layout.addWidget(age_label)

    layout.addWidget(age_edit)

    # Family history of diseases

    family_history_label = QLabel("Family History of Diseases:")

    family_history_edit = QTextEdit()

    layout.addWidget(family_history_label)

    layout.addWidget(family_history_edit)

    # Previous surgeries

    surgeries_label = QLabel("Previous Surgeries:")
```

```python
surgeries_edit = QTextEdit()

layout.addWidget(surgeries_label)

layout.addWidget(surgeries_edit)

# Allergies

allergies_label = QLabel("Allergies:")

allergies_edit = QTextEdit()

layout.addWidget(allergies_label)

layout.addWidget(allergies_edit)

# Symptoms

symptoms_label = QLabel("Symptoms:")

symptoms_edit = QTextEdit()

layout.addWidget(symptoms_label)

layout.addWidget(symptoms_edit)

# Diagnosis

diagnosis_label = QLabel("Diagnosis:")

diagnosis_edit = QTextEdit()

layout.addWidget(diagnosis_label)

layout.addWidget(diagnosis_edit)

# Tests

tests_label = QLabel("Tests:")

tests_edit = QTextEdit()

layout.addWidget(tests_label)

layout.addWidget(tests_edit)

# Medications

medications_label = QLabel("Medications:")

medications_edit = QTextEdit()

layout.addWidget(medications_label)
```

```python
        layout.addWidget(medications_edit)


        save_button = QPushButton("Save")

        save_button.clicked.connect(lambda:
save_patient_entry(name_edit.text(), age_edit.text(),
family_history_edit.toPlainText(), surgeries_edit.toPlainText(),
allergies_edit.toPlainText(), symptoms_edit.toPlainText(),
diagnosis_edit.toPlainText(), tests_edit.toPlainText(),
medications_edit.toPlainText()))

        layout.addWidget(save_button)


        new_entry_window.setLayout(layout)

        new_entry_window.show()


def save_patient_entry(name, age, family_history, surgeries,
allergies, symptoms, diagnosis, tests, medications):

    # Use global cursor object

    global cursor

    # Generate random username and password for the patient

    patient_username = ''.join(random.choices(string.ascii_lowercase +
string.digits, k=8))

    patient_password = ''.join(random.choices(string.ascii_letters +
string.digits, k=8))


    # Insert patient details, username, and password into the database

    insert_query_patient = "INSERT INTO patients (name, age,
family_history, surgeries, allergies, symptoms, diagnosis, tests,
medications, username, password) VALUES (%s, %s, %s, %s, %s, %s, %s,
%s, %s, %s, %s)"

    cursor.execute(insert_query_patient, (name, age, family_history,
surgeries, allergies, symptoms, diagnosis, tests, medications,
patient_username, patient_password))


    # Insert patient details into the users table
```

```python
    insert_query_user = "INSERT INTO users (username, password, role)
VALUES (%s, %s, %s)"

    cursor.execute(insert_query_user, (patient_username,
patient_password, 'patient'))



    db_connection.commit()

    QMessageBox.information(None, "Success", "Patient entry saved
successfully. Username: {}, Password: {}".format(patient_username,
patient_password))



def update_entry():

    global update_window

    update_window = QWidget()

    update_window.setWindowTitle("Update Patient Entry")

    update_window.setGeometry(100, 100, 400, 150)



    layout = QVBoxLayout()



    patient_name_label = QLabel("Patient Name:")

    patient_name_edit = QLineEdit()

    layout.addWidget(patient_name_label)

    layout.addWidget(patient_name_edit)



    fetch_button = QPushButton("Fetch Details")

    fetch_button.clicked.connect(lambda:
fetch_patient_details_by_name(patient_name_edit.text()))

    layout.addWidget(fetch_button)



    update_window.setLayout(layout)

    update_window.show()
```

```python
def fetch_patient_details_by_name(patient_name):

    global cursor, update_patient_window

    cursor.execute("SELECT * FROM patients WHERE name = %s",
(patient_name,))

    patient_data = cursor.fetchone()


    if patient_data:

        # Ensure update_patient_window is a global variable

        update_patient_window = QWidget()

        update_patient_window.setWindowTitle("Update Patient Details")

        update_patient_window.setGeometry(100, 100, 400, 400)


        layout = QVBoxLayout()


        # Name

        name_label = QLabel("Name:")

        name_edit = QLineEdit(patient_data[1])

        layout.addWidget(name_label)

        layout.addWidget(name_edit)


        # Age

        age_label = QLabel("Age:")

        age_edit = QLineEdit(str(patient_data[2]))

        layout.addWidget(age_label)

        layout.addWidget(age_edit)


        # Family history of diseases
```

```python
family_history_label = QLabel("Family History of Diseases:")

family_history_edit = QTextEdit(patient_data[3])

layout.addWidget(family_history_label)

layout.addWidget(family_history_edit)


# Previous surgeries

surgeries_label = QLabel("Previous Surgeries:")

surgeries_edit = QTextEdit(patient_data[4])

layout.addWidget(surgeries_label)

layout.addWidget(surgeries_edit)


# Allergies

allergies_label = QLabel("Allergies:")

allergies_edit = QTextEdit(patient_data[5])

layout.addWidget(allergies_label)

layout.addWidget(allergies_edit)


# Symptoms

symptoms_label = QLabel("Symptoms:")

symptoms_edit = QTextEdit(patient_data[6])

layout.addWidget(symptoms_label)

layout.addWidget(symptoms_edit)


# Diagnosis

diagnosis_label = QLabel("Diagnosis:")

diagnosis_edit = QTextEdit(patient_data[7])

layout.addWidget(diagnosis_label)
```

```python
        layout.addWidget(diagnosis_edit)


        # Tests

        tests_label = QLabel("Tests:")

        tests_edit = QTextEdit(patient_data[8])

        layout.addWidget(tests_label)

        layout.addWidget(tests_edit)


        # Medications

        medications_label = QLabel("Medications:")

        medications_edit = QTextEdit(patient_data[9])

        layout.addWidget(medications_label)

        layout.addWidget(medications_edit)


        save_button = QPushButton("Save")

        save_button.clicked.connect(lambda:
save_updated_patient_details(patient_data[0], name_edit.text(),
age_edit.text(), family_history_edit.toPlainText(),
surgeries_edit.toPlainText(), allergies_edit.toPlainText(),
symptoms_edit.toPlainText(), diagnosis_edit.toPlainText(),
tests_edit.toPlainText(), medications_edit.toPlainText()))

        layout.addWidget(save_button)


        update_patient_window.setLayout(layout)

        update_patient_window.show()

    else:

        QMessageBox.warning(None, "Error", "Patient with name '{}' not
found.".format(patient_name))


def save_updated_patient_details(patient_id, name, age,
family_history, surgeries, allergies, symptoms, diagnosis, tests,
```

```python
medications):

    global cursor, db_connection

    update_query = "UPDATE patients SET name = %s, age = %s,
family_history = %s, surgeries = %s, allergies = %s, symptoms = %s,
diagnosis = %s, tests = %s, medications = %s WHERE id = %s"

    cursor.execute(update_query, (name, age, family_history,
surgeries, allergies, symptoms, diagnosis, tests, medications,
patient_id))

    db_connection.commit()


    QMessageBox.information(None, "Success", "Patient details updated
successfully.")


if __name__ == "__main__":

    app = QApplication(sys.argv)

    show_login_window()

    sys.exit(app.exec_())
```

MySQL:

```sql
drop table users

CREATE TABLE users (

    id INT AUTO_INCREMENT PRIMARY KEY,

    username VARCHAR(255) NOT NULL,

    password VARCHAR(255) NOT NULL,

    role VARCHAR(50) NOT NULL

);

insert into users(username,password,role)
values('Shiiv','shiiv123','doctor');


CREATE TABLE patients (
```

```sql
    id INT AUTO_INCREMENT PRIMARY KEY,

    name VARCHAR(255) NOT NULL,

    age INT NOT NULL,

    family_history TEXT,

    surgeries TEXT,

    allergies TEXT,

    symptoms TEXT,

    diagnosis TEXT,

    tests TEXT,

    medications TEXT,

    report TEXT,

    username VARCHAR(255) NOT NULL UNIQUE,

    password VARCHAR(255) NOT NULL,

    first_login BOOLEAN DEFAULT TRUE
);


truncate table patients

select * from patients


select * from users


select * from users where role='doctor'


-- Trigger to check unique username before insert
CREATE TRIGGER before_patient_insert
```

```sql
BEFORE INSERT ON patients

FOR EACH ROW

BEGIN

    DECLARE username_count INT;

    SELECT COUNT(*) INTO username_count FROM users WHERE
username = NEW.username;

    IF username_count > 0 THEN

        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Username
already exists.';

    END IF;

END;


-- Trigger to generate report after insert

CREATE TRIGGER after_patient_insert

AFTER INSERT ON patients

FOR EACH ROW

BEGIN

    SET @report = CONCAT('Name: ', NEW.name, '\n',

                        'Age: ', NEW.age, '\n',

                        'Family History: ', NEW.family_history,
'\n',

                        'Previous Surgeries: ', NEW.surgeries,
'\n',

                        'Allergies: ', NEW.allergies, '\n',

                        'Symptoms: ', NEW.symptoms, '\n',

                        'Diagnosis: ', NEW.diagnosis, '\n',

                        'Tests: ', NEW.tests, '\n',

                        'Medications: ', NEW.medications);
```

```sql
    UPDATE patients SET report = @report WHERE id = NEW.id;
END;


CREATE PROCEDURE AddUser (
    IN input_username VARCHAR(255),
    IN input_password VARCHAR(255),
    IN input_role VARCHAR(50)
)
BEGIN
    -- Validate role
    IF input_role NOT IN ('doctor', 'patient') THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Invalid role. Must be either
"doctor" or "patient"';
    END IF;


    -- Hash the password using SHA2 function
    INSERT INTO users (username, password, role)
    VALUES (input_username, SHA2(input_password, 256),
input_role);
END;


CALL AddUser('Yokesh','yokesh123','doctor')
```

5. OUTPUT

6. RESULTS AND DISCUSSION

This system helps doctors manage patient records easily and securely. Doctors can only see authorized information, and automated tasks make data accurate and save time. Patients could benefit too, by viewing their records and scheduling appointments online. Encryption and backups would make the system even more secure, and doctors could track changes to records for better care. Hence, this is a great start for a secure and efficient patient record system.

7. CONCLUSION

The patient health record management system developed using Python, MySQL, and PyQt effectively addresses the needs for efficient and secure patient data management. With features like user authentication, role-based access, and automated workflows, the system enhances accuracy and accessibility of patient records. MySQL triggers further improve data integrity and operational efficiency by automating tasks such as change logging and report generation. Overall, this system provides a robust, scalable, and user-friendly solution, streamlining administrative processes and improving the quality of patient care.

8. REFERENCES

1. Riverbank Computing Limited. PyQt5 Documentation. Retrieved from https://www.riverbankcomputing.com/static/Docs/PyQt5/
2. Oracle Corporation. MySQL 8.0 Reference Manual. Retrieved from https://dev.mysql.com/doc/refman/8.0/en/
3. Python Software Foundation. Python 3 Documentation. Retrieved from https://docs.python.org/3/
4. The Qt Company. Qt 5.15 Documentation. Retrieved from https://doc.qt.io/qt-5.15/index.html
5. Real Python. Generating Random Data in Python (Guide). Retrieved from https://realpython.com/python-random/
6. MySQL Tutorial. MySQL Trigger. Retrieved from https://www.mysqltutorial.org/mysql-triggers.aspx