THE DEENS ACADEMY

COMPUTER SCIENCE PROJECT ON

PHARMA LOGIX

YOUR ALL IN ONE MEDICAL HEALTH SOLUTION

NAME	Kashish Nair
CLASS	XII - B
REG. NO	
ACADEMIC YEAR	AY 2025-26

Prepared as partial fulfilment of requirement in the subject as per guidelines issued by Central Board of

Secondary Education, New Delhi

CENTRAL BOARD OF SECONDARY EDUCATION
INDIA
असतो मा सद्गमय

ACKNOWLEDGMENT

I would like to express my sincere gratitude to all those who helped me complete this project successfully.

First and foremost, I am deeply thankful to our school principal, Ms. Shanthi Menon, for providing the necessary facilities and environment for completing this project.

I would also like to thank my Computer Science teacher, Ms. Swathi, for her valuable guidance, support, and encouragement throughout the project. Her expert advice and continuous monitoring made this work possible.

I would like to thank my classmates and friends for their cooperation and suggestions. Last but not least, I express my heartfelt thanks to my parents and family members for their constant support and motivation.

This project has been a great learning experience, and I am grateful for the opportunity to work on it.

INDEX

INTRODUCTION

This project has been coded using the Python programming language and MySQL.

Python is one of the world's most popular programming languages. It was created by Guido van Rossum. Released in 1991, it is supports multiple programming paradigms, including procedural, object-oriented, and functional programming.

Python is known for its simple and readable syntax, which makes it an excellent choice for both beginners and experienced developers. Its extensive standard library and large ecosystem of third-party packages make it suitable for a wide variety of applications such as web development, data analysis, artificial intelligence, machine learning, automation, and more.

MySQL, on the other hand, is one of the most widely used open-source relational database management systems. It is based on structured query language (SQL), which is used for storing, retrieving, modifying, and managing data in a database. MySQL is known for its speed, reliability, and ease of use. It is often used in combination with server-side scripting languages like PHP or Python to build dynamic and data-driven applications.

In this project, Python handles the logic and interface while MySQL is used to manage the database backend.

The integration between Python and MySQL is commonly achieved using mysql-connector library, which allow Python programs to connect to a MySQL database, execute queries, and process results.

THEORY

This project is based on an app to create, store, view, update, and delete records in a database designed specifically for medical healthcare records.

Pharma Logix offers a way for small NGOs to store, view, and edit data efficiently instead of using bulky Excel files, which carry the issue of data repetition and redundancy, as well as difficulty to navigate through for a large number of students.

The main page offers a variety of entries to add data, including specific fields for student data, pediatric details, ophthalmic details, etc.

It also offers features such as automatic report generation, which automatically generates medical reports after extracting the data from the database.

Polished off with a visually striking accessible, comprehensible, easy-to-use user interface, it is sure to engage the visual appeal of users.

SYSTEM REQUIREMENTS

OPERATING SYSTEM	Windows 10 or newer
SYSTEM TYPE	X64-based PC
PROCESSOR TYPE (MINIMUM)	3.5GHz, 2 cores, 4 logical processors
PROCESSOR TYPE (RECOMMENDED)	4.2GHz, 4 cores, 8 logical processors
RAM (MINIMUM)	8GB
RAM (RECOMMENDED)	16GB

SYSTEM DESIGN

Modules:

1. PyQt5 → User interface
2. mysql-connector-python → Python-database interface
3. PIL → To enable background image in user interface
4. os → To automatically run certain executable lines and for package installation
5. sys → To provide access to system-specific parameters and functions
6. subprocess → To create and manage new processes, allowing interaction with the operating system and execution of external commands and programs
7. reportlab → To generate PDF report from given data in SQL and Python

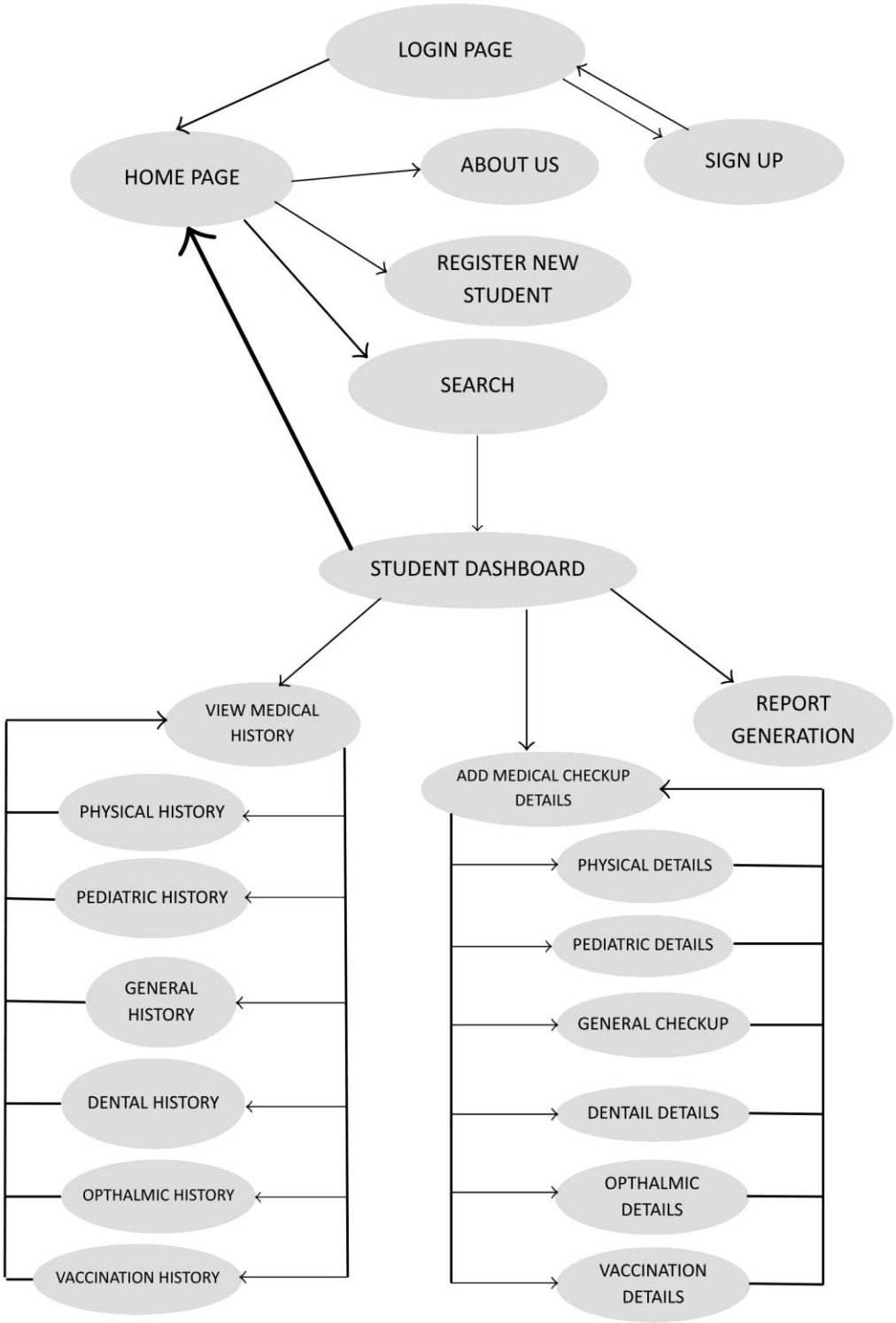
User-defined functions:

- _init_ (self, search_name=None, parent=None): Initializes the GUI window, sets up all forms and fields for data entry, and connects to MySQL to ensure database exists
- init_ui(self): Builds the entire UI
- update_background(self): Applies the background image to the window
- resizeEvent(self, event): Ensures that background image updates every time the window size changes
- go_to_main_menu(self): Handles the 'BACK' button click
- submit_form(self): Saves the data entered into SQL database
- get_connection(self): Establishes connection with SQL database
- create_table_if_not_exists(self): ensures that table exists in the database
- load_data_from_db(self): Fetches data from the database
- go_back(self): Handles the 'BACK' button click
- open_physical_history(self): Opens the physical history window for the current student
- open_general_checkup_history(self): Opens the general checkup window for the current student
- open_pediatric_history(self): Opens the pediatric history window for the current student
- open_dental_history(self): Opens the dental history window for the current student

- open_opthalmic_history(self): Opens the ophthalmic history window for the current student
- open_vaccination_history(self): Opens the vaccination history window for the current student
- launch_file(self, filename): executes another Python script (used for launching other pages or the main menu)
- open_physical_details(self): Opens the physical check up window, passing the selected student's name and hiding the current menu.
- open_general_checkup(self): Opens the general check up window
- open_dental_checkup(self): Opens the dental check up window
- open_opthalmic_checkup(self): Opens the ophthalmic check up window
- open_vaccination_details(self): Opens the vaccination check up window
- DetailPage.go_to_main_menu(self): Used by the smaller "DetailPage" class to return to the base main menu
- connect_db(self): Establishes a connection with the MySQL database
- load_students(self): Loads student names from physical details table into dropdown menu
- generate_report(self): Generates a comprehensive medical report for the student
- initialize_database(): Ensures that database and tables exist before the app runs
- get_db_connection(): Returns connection to the database
- AnimatedButton._pulse_effect(self): Creates small animation effect when a button is clicked
- AnimatedButton._restore(self, start): Restores the button to its original geometry after the pulse animation
- PageWithLogo.paintEvent(self, event): Custom paint event to draw a background image
- PageWithLogo._add_logo(self): Adds the Pharma Logix logo to the top-left corner of the window
- LoginPage._build_ui(self): Builds the login page layout
- LoginPage._handle_login(self): Verifies the user's credentials
- SignupPage._build_ui(self): Builds the sign-up page layout with username/password input fields and buttons for signing up and returning to login
- SignupPage._handle_signup(self): Registers a new user in the database

- MainDashboard._build_ui(self): Builds the main page after login
- MainDashboard._search_student(self, name): alidates and searches for a student
- MainDashboard.logout(self): Logs the user out and returns to the parent window (login page)
- MainDashboard.open_student_regitration(self): Opens the student registration form
- StudentDashboard._build_ui(self): Builds the student's personal dashboard
- StudentDashboard.open_medical_checkup_history(self): Opens pediatric checkup page for the selected student
- StudentDashboard.open_medical_history(self): Opens general medical history page for the selected student
- StudentDashboard.generate_report(self): Opens the generate report window
- App.show_student_dashboard(self, student_name): Loads and displays a student dashboard for the selected student
- App.show_main_dashboard(self): Returns the user to the main dashboard view

FLOW DIAGRAM



SOURCE CODE

LOGIN WINDOW:

```
import sys
import os
import subprocess
import mysql.connector
from mysql.connector import errorcode
from PyQt5.QtCore import Qt, QPropertyAnimation, QEasingCurve
from PyQt5.QtGui import QFont, QPixmap, QPainter
from PyQt5.QtWidgets import (
    QApplication, QWidget, QLabel, QLineEdit, QPushButton,
    QVBoxLayout, QHBoxLayout, QMessageBox, QStackedWidget
)

# === Paths to assets ===
BG_PATH = r"C:\Users\Student\Documents\Aaditya Sundaram\Project-
121025\project_final\Bg (1).jpg"
LOGO_PATH = r"C:\Users\Student\Downloads\L.png"


# ---------- Database Setup ----------
def initialize_database():
    """Ensures that the MySQL database and required tables exist."""
    try:
        conn = mysql.connector.connect(
            host="localhost",
            user="root",
            password="deens"
        )
        cursor = conn.cursor()
        cursor.execute("CREATE DATABASE IF NOT EXISTS pharmalogix")
        cursor.execute("USE pharmalogix")

        # Create login table
        cursor.execute("""
            CREATE TABLE IF NOT EXISTS login (
                id INT AUTO_INCREMENT PRIMARY KEY,
                username VARCHAR(100) UNIQUE NOT NULL,
                password VARCHAR(100) NOT NULL,
                created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
            )
        """)

        # Insert default admin user if table empty
        cursor.execute("SELECT COUNT(*) FROM login")
        if cursor.fetchone()[0] == 0:
            cursor.execute("INSERT INTO login (username, password) VALUES
('admin', 'admin')")
```

```
            print("✅ Default admin user created (username: admin,
password: admin)")

        conn.commit()
        conn.close()

        print("✅ Database and tables ready.")
    except mysql.connector.Error as err:

        print(f"❌ Database setup failed: {err}")
        sys.exit(1)


def get_db_connection():
    """Return a connection to the pharmalogix database."""
    return mysql.connector.connect(
        host="localhost",
        user="root",
        password="deens",
        database="pharmalogix"
    )


# ---------- Animated Button ----------
class AnimatedButton(QPushButton):
    def __init__(self, text, parent=None):
        super().__init__(text, parent)
        self.clicked.connect(self._pulse_effect)

    def _pulse_effect(self):
        start = self.geometry()
        bump = start.adjusted(-3, -2, 3, 2)
        anim = QPropertyAnimation(self, b"geometry", self)
        anim.setDuration(120)
        anim.setStartValue(start)
        anim.setEndValue(bump)
        anim.setEasingCurve(QEasingCurve.OutBack)
        anim.finished.connect(lambda: self._restore(start))
        anim.start()

    def _restore(self, start):
        anim = QPropertyAnimation(self, b"geometry", self)
        anim.setDuration(120)
        anim.setStartValue(self.geometry())
        anim.setEndValue(start)
        anim.setEasingCurve(QEasingCurve.InBack)
        anim.start()


# ---------- Base Page ----------
class PageWithLogo(QWidget):
    def paintEvent(self, event):
```

```
        painter = QPainter(self)
        painter.drawPixmap(self.rect(), QPixmap(BG_PATH))
        super().paintEvent(event)

    def _add_logo(self):
        logo = QLabel(self)
        pix = QPixmap(LOGO_PATH)
        if not pix.isNull():
            pix = pix.scaled(60, 60, Qt.KeepAspectRatio,
Qt.SmoothTransformation)
            logo.setPixmap(pix)
            logo.setFixedSize(pix.size())
            logo.move(20, 20)


# ---------- Login Page ----------
class LoginPage(PageWithLogo):
    def __init__(self, on_login, go_to_signup):
        super().__init__()
        self._on_login = on_login
        self._go_to_signup = go_to_signup
        self._build_ui()

    def _build_ui(self):
        self._add_logo()

        title = QLabel("Login", self)
        title.setFont(QFont("Arial", 32, QFont.Bold))
        title.setAlignment(Qt.AlignCenter)

        self.username_field = QLineEdit(self)
        self.username_field.setPlaceholderText("Username")
        self.username_field.setFont(QFont("Arial", 18))
        self.username_field.setFixedSize(400, 50)

        self.password_field = QLineEdit(self)
        self.password_field.setPlaceholderText("Password")
        self.password_field.setEchoMode(QLineEdit.Password)
        self.password_field.setFont(QFont("Arial", 18))
        self.password_field.setFixedSize(400, 50)

        forgot = QPushButton("Forgot Password?", self)
        forgot.setFont(QFont("Arial", 12))
        forgot.setFlat(True)

        login_btn = AnimatedButton("Login", self)
        login_btn.setFont(QFont("Arial", 14))
        login_btn.setObjectName("login-btn")
        login_btn.clicked.connect(self._handle_login)

        signup_btn = AnimatedButton("Sign Up", self)
        signup_btn.setFont(QFont("Arial", 14))
        signup_btn.clicked.connect(self._go_to_signup)
```

```
        self.setStyleSheet("""
            QWidget { background: transparent; }
            QLineEdit {
                background: white; border: 2px solid gray;
                border-radius: 25px; padding-left: 20px; font-size: 18px;
            }
            QLineEdit:focus { border-color: #555; }
            QPushButton {
                background: transparent; border: none; color: black;
            }
            QPushButton#login-btn {
                background-color: #27ae60; border: 2px solid white;
                border-radius: 25px; padding: 8px 25px; color: white;
            }
            QPushButton#login-btn:pressed {
                background-color: #1e8449;
            }
        """)

        form = QVBoxLayout()
        form.setAlignment(Qt.AlignCenter)
        form.addWidget(title)
        form.addSpacing(40)
        form.addWidget(self.username_field, alignment=Qt.AlignCenter)
        form.addSpacing(20)
        form.addWidget(self.password_field, alignment=Qt.AlignCenter)
        form.addSpacing(10)
        form.addWidget(forgot, alignment=Qt.AlignCenter)
        form.addSpacing(15)
        form.addWidget(login_btn, alignment=Qt.AlignCenter)

        top = QHBoxLayout()
        top.addStretch()
        top.addWidget(signup_btn)
        top.setContentsMargins(0, 0, 20, 0)

        layout = QVBoxLayout(self)
        layout.addLayout(top)
        layout.addStretch()
        layout.addLayout(form)
        layout.addStretch()

    def _handle_login(self):
        user = self.username_field.text().strip()
        pwd = self.password_field.text().strip()

        if not user or not pwd:
            QMessageBox.warning(self, "Input Error", "Username and
password cannot be empty.")
            return

        try:
            conn = get_db_connection()
            cur = conn.cursor()
```

```
            cur.execute("SELECT * FROM login WHERE username=%s AND
password=%s", (user, pwd))
            row = cur.fetchone()
            conn.close()

            if row:
                # Open main_menu.py using the same Python interpreter and
close the login app
                base_dir = os.path.dirname(os.path.abspath(__file__))
                main_menu_path = os.path.join(base_dir, "main_menu.py")
                try:
                    subprocess.Popen([sys.executable, main_menu_path])
                except Exception:
                    # fallback to os.system if Popen fails for any reason
                    os.system(f"{sys.executable} " + main_menu_path)
                QApplication.quit()
            else:
                QMessageBox.warning(self, "Login Failed", "Invalid
username or password.")
        except Exception as e:
            QMessageBox.critical(self, "Database Error", str(e))


# ---------- Signup Page ----------
class SignupPage(PageWithLogo):
    def __init__(self, go_back):
        super().__init__()
        self._go_back = go_back
        self._build_ui()

    def _build_ui(self):
        self._add_logo()

        title = QLabel("SIGN UP", self)
        title.setFont(QFont("Arial", 32, QFont.Bold))
        title.setAlignment(Qt.AlignCenter)

        self.username_field = QLineEdit(self)
        self.username_field.setPlaceholderText("Username")

        self.password_field = QLineEdit(self)
        self.password_field.setPlaceholderText("Password")
        self.password_field.setEchoMode(QLineEdit.Password)

        for field in (self.username_field, self.password_field):
            field.setFont(QFont("Arial", 18))
            field.setFixedSize(400, 50)

        signup_btn = AnimatedButton("Sign Up", self)
        signup_btn.setFont(QFont("Arial", 14))
        signup_btn.setObjectName("signup-btn")
        signup_btn.clicked.connect(self._handle_signup)

        back_btn = AnimatedButton("Back to Login", self)
```

```
        back_btn.setFont(QFont("Arial", 12))
        back_btn.clicked.connect(self._go_back)

        self.setStyleSheet("""
            QWidget { background: transparent; }
            QLineEdit {
                background: white; border: 2px solid gray;
                border-radius: 25px; padding-left: 20px; font-size: 18px;
            }
            QPushButton {
                background: transparent; border: none; color: black;
            }
            QPushButton#signup-btn {
                background-color: #27ae60; border: 2px solid white;
                border-radius: 25px; padding: 8px 25px; color: white;
            }
            QPushButton#signup-btn:pressed {
                background-color: #1e8449;
            }
        """)

        form = QVBoxLayout()
        form.setAlignment(Qt.AlignCenter)
        form.addWidget(title)
        form.addSpacing(40)
        form.addWidget(self.username_field, alignment=Qt.AlignCenter)
        form.addSpacing(20)
        form.addWidget(self.password_field, alignment=Qt.AlignCenter)
        form.addSpacing(30)
        form.addWidget(signup_btn, alignment=Qt.AlignCenter)
        form.addSpacing(10)
        form.addWidget(back_btn, alignment=Qt.AlignCenter)

        layout = QVBoxLayout(self)
        layout.addStretch()
        layout.addLayout(form)
        layout.addStretch()

    def _handle_signup(self):
        user = self.username_field.text().strip()
        pwd = self.password_field.text().strip()

        if not user or not pwd:
            QMessageBox.warning(self, "Input Error", "Username and
password cannot be empty.")
            return

        try:
            conn = get_db_connection()
            cur = conn.cursor()
            cur.execute("SELECT * FROM login WHERE username = %s",
(user,))
            if cur.fetchone():
                conn.close()
```

```
                QMessageBox.warning(self, "Error", "Username already
exists.")
                return

            cur.execute("INSERT INTO login (username, password) VALUES
(%s, %s)", (user, pwd))
            conn.commit()
            conn.close()
            QMessageBox.information(self, "Success", "Registration
successful. You can now login.")
            self._go_back()
        except mysql.connector.Error as err:
            QMessageBox.critical(self, "Database Error", f"MySQL
said:\n{err}")
        except Exception as e:
            QMessageBox.critical(self, "Unexpected Error", str(e))


# ---------- App Controller ----------
class App(QStackedWidget):
    def __init__(self):
        super().__init__()
        login = LoginPage(on_login=self.show_success, go_to_signup=lambda:
self.setCurrentIndex(1))
        signup = SignupPage(go_back=lambda: self.setCurrentIndex(0))
        self.addWidget(login)
        self.addWidget(signup)
        self.setFixedSize(800, 500)
        self.setCurrentIndex(0)

    def show_success(self):
        pass  # not needed since login directly opens main_menu.py


# ---------- Run App ----------
if __name__ == "__main__":
    initialize_database()
    app = QApplication(sys.argv)
    window = App()
    window.setWindowTitle("Pharma Logix - Login & Signup")
    window.show()
    sys.exit(app.exec_())
```

MAIN MENU:

```
import sys

from PyQt5.QtCore import Qt, QUrl

from PyQt5.QtGui import QPainter, QPixmap, QFont, QIcon, QDesktopServices

from PyQt5.QtWidgets import (

    QApplication, QWidget, QLabel, QLineEdit, QPushButton,
```

```
    QVBoxLayout, QHBoxLayout, QStackedWidget, QMessageBox,
)


# Paths to assets — update as needed
BG_PATH = r"C:\Users\Student\Documents\Aaditya Sundaram\Project-
121025\project_final\Bg (1).jpg"
LOGO_PATH = r"C:\Python\Project\PROJECT\L.png"
ICON_MAG = r"C:\Users\Student\Downloads\magnifier.png"


# ---------- Base Page ----------
class PageWithLogo(QWidget):
    """Base class for pages with a background and logo."""
    def paintEvent(self, event):
        painter = QPainter(self)
        painter.drawPixmap(self.rect(), QPixmap(BG_PATH))
        super().paintEvent(event)


    def _add_logo(self):
        logo = QLabel(self)
        pix = QPixmap(LOGO_PATH)
        if not pix.isNull():
            pix = pix.scaled(60, 60, Qt.KeepAspectRatio,
Qt.SmoothTransformation)
            logo.setPixmap(pix)
            logo.setFixedSize(pix.size())
            logo.move(20, 20)


# ---------- Main Dashboard ----------
class MainDashboard(PageWithLogo):
    def __init__(self, open_student_dashboard):
        super().__init__()
        self.open_student_dashboard = open_student_dashboard
```

```
        self._build_ui()

    def _build_ui(self):
        self.setFixedSize(800, 500)
        self._add_logo()

        logout = QPushButton("LOGOUT", self)
        logout.setFont(QFont("Arial", 12))
        logout.clicked.connect(self.logout)

        search = QLineEdit(self)
        search.setPlaceholderText("Enter Student Name")
        search.setFont(QFont("Arial", 18))
        search.setFixedSize(600, 50)
        search.addAction(QIcon(ICON_MAG), QLineEdit.LeadingPosition)
        search.setStyleSheet("""
            QLineEdit {
                background: white;
                border: 2px solid gray;
                border-radius: 25px;
                padding-left: 40px;
                font-size: 18px;
                color: black;
            }
            QLineEdit:focus { border-color: #555; }
        """)
        search.returnPressed.connect(lambda:
self._search_student(search.text()))

        org = QLabel("Organization: {}", self)
        org.setFont(QFont("Arial", 14))
```

```
org.setAlignment(Qt.AlignCenter)

btn1 = QPushButton("Register New Student", self)
btn2 = QPushButton("About Us", self)
for btn in (btn1, btn2):
    btn.setFont(QFont("Arial", 18))
    btn.setFixedSize(300, 50)
    btn.setStyleSheet("""
        QPushButton {
            background: white;
            border: 2px solid black;
            border-radius: 25px;
            color: black;
            font-size: 18px;
            padding: 0 12px;
        }
        QPushButton:pressed {
            background: #dddddd;
            border-style: inset;
        }
    """)
btn1.clicked.connect(self.open_student_regitration)
btn2.clicked.connect(self.open_youtube)
top = QHBoxLayout()
top.addStretch()
top.addWidget(logout)
top.setContentsMargins(20, 20, 20, 0)

title = QLabel("Pharma Logix", self)
title.setFont(QFont("Arial", 32, QFont.Bold))
title.setAlignment(Qt.AlignCenter)
```

```
        center = QVBoxLayout()
        center.setAlignment(Qt.AlignCenter)
        center.addWidget(title)
        center.addSpacing(10)
        center.addWidget(org)
        center.addSpacing(30)
        center.addWidget(search, alignment=Qt.AlignCenter)
        center.addSpacing(30)
        center.addWidget(btn1, alignment=Qt.AlignCenter)
        center.addSpacing(15)
        center.addWidget(btn2, alignment=Qt.AlignCenter)

        layout = QVBoxLayout(self)
        layout.addLayout(top)
        layout.addStretch()
        layout.addLayout(center)
        layout.addStretch()

    def open_youtube(self):
        # Open the organization's About Us external page
        url = QUrl("https://empactngo.my.canva.site/")
        QDesktopServices.openUrl(url)

    def _search_student(self, name):
        if not name.strip():
            QMessageBox.warning(self, "Input Error", "Please enter a
student name.")
        else:
            self.open_student_dashboard(name)
    def logout(self):
```

```
        if self.parent_window:
            self.parent_window.show()  # ✅ show the main menu again
        self.close()

    def open_student_regitration(self):
        """Opens stuwindow.py (Student Registration page)."""
        import subprocess, os, sys
        base_dir = os.path.dirname(os.path.abspath(__file__))
        stuwindow_path = os.path.join(base_dir, "stuwindow.py")
        if os.path.exists(stuwindow_path):
            subprocess.Popen([sys.executable, stuwindow_path])
            self.close()
        else:
            QMessageBox.warning(self, "File Missing", f"Could not
find:\n{stuwindow_path}")

# ---------- Student Dashboard ----------
class StudentDashboard(PageWithLogo):
    def __init__(self, student_name, go_back):
        super().__init__()
        self.student_name = student_name
        self.go_back = go_back
        self._build_ui()

    def _build_ui(self):
        import os, sys, subprocess

        self.setFixedSize(800, 500)
        self._add_logo()

        back = QPushButton("BACK", self)
```

```
back.setFont(QFont("Arial", 12))
back.clicked.connect(self.go_back)

org = QLabel("Organization: {}", self)
org.setFont(QFont("Arial", 14))
org.setAlignment(Qt.AlignCenter)

student_label = QLabel(f"{self.student_name}", self)
student_label.setFont(QFont("Arial", 18, QFont.Bold))
student_label.setAlignment(Qt.AlignCenter)

# === Buttons ===
btn1 = QPushButton("View Medical History 🧬", self)
btn2 = QPushButton("Add Medical Checkup Details 🩺", self)
btn3 = QPushButton("Generate Report 📄", self)

for btn in (btn1, btn2, btn3):
    btn.setFont(QFont("Arial", 18))
    btn.setFixedSize(300, 50)
    btn.setStyleSheet("""
        QPushButton {
            background: white;
            border: none;
            border-radius: 25px;
            font-size: 18px;
            padding: 8px 16px;
        }
        QPushButton:pressed { background: #dddddd; }
    """)

# --- Button connections ---
```

```
btn2.clicked.connect(self.open_medical_checkup_history)
btn1.clicked.connect(self.open_medical_history)
btn3.clicked.connect(self.generate_report)

top = QHBoxLayout()
top.addStretch()
top.addWidget(back)
top.setContentsMargins(20, 20, 20, 0)

title = QLabel("Pharma Logix", self)
title.setFont(QFont("Arial", 32, QFont.Bold))
title.setAlignment(Qt.AlignCenter)

center = QVBoxLayout()
center.setAlignment(Qt.AlignCenter)
center.addWidget(title)
center.addSpacing(10)
center.addWidget(org)
center.addSpacing(15)
center.addWidget(student_label)
center.addSpacing(25)

# Two buttons in one row
row = QHBoxLayout()
row.addWidget(btn1)
row.addSpacing(30)
row.addWidget(btn2)

center.addLayout(row)
center.addSpacing(20)
center.addWidget(btn3, alignment=Qt.AlignCenter)
```

```
        layout = QVBoxLayout(self)
        layout.addLayout(top)
        layout.addStretch()
        layout.addLayout(center)
        layout.addStretch()


    # === Open Medical Checkup History Function ===
    def open_medical_checkup_history(self):
        """Opens stuhis.py (Medical Check-Up page)."""
        from stuphydet import MainMenu  # local import to avoid circular
imports
        self.studet_window = MainMenu(student_name=self.student_name,
parent=self)  # ✓ pass self as parent
        self.hide()  # ✓ hide current main menu
        self.studet_window.show()


    def open_medical_history(self):
        """Opens stuhis.py (Medical Check-Up page)."""
        from stuhis import MainMenu  # local import to avoid circular
imports
        self.stuhis_window = MainMenu(student_name=self.student_name,
parent=self)  # ✓ pass self as parent
        self.hide()  # ✓ hide current main menu
        self.stuhis_window.show()
    def generate_report(self):
        """Opens generate_report.py (Generate Report page)."""
        from generate_report import ReportPage  # local import to avoid
circular imports
        self.report_window = ReportPage(parent=self)  # ✓ pass self as
parent
        self.hide()  # ✓ hide current main menu
        self.report_window.show()
```

```
# ---------- App Controller ----------
class App(QStackedWidget):
    def __init__(self, initial_page):
        super().__init__()
        self.student_name = initial_page
        self.setFixedSize(800, 500)

        self.student_dashboard = StudentDashboard(self.student_name,
go_back=self.show_main_dashboard)
        self.main_dashboard =
MainDashboard(open_student_dashboard=self.show_student_dashboard)
        if initial_page != "Main":
            self.addWidget(self.student_dashboard)
        else:
            self.addWidget(self.main_dashboard)

    def show_student_dashboard(self, student_name):
        student_page = StudentDashboard(student_name,
go_back=self.show_main_dashboard)
        self.addWidget(student_page)
        self.setCurrentWidget(student_page)

    def show_main_dashboard(self):
        self.setCurrentWidget(self.main_dashboard)

if __name__ == "__main__":
    app = QApplication(sys.argv)

    window = App(sys.argv[1] if len(sys.argv) > 1 else "Main")
    window.setWindowTitle("Pharma Logix - Main Menu")
    window.show()
    sys.exit(app.exec_())
```

STUDENT CHECKUP MAIN WINDOW:

```
import sys
import os
import subprocess
from PyQt5.QtWidgets import (
    QApplication, QWidget, QLabel, QPushButton, QVBoxLayout, QHBoxLayout
)
from PyQt5.QtGui import QFont, QPixmap
from PyQt5.QtCore import Qt
from main_menu import StudentDashboard  # Adjust class name as needed
from main_menu import MainDashboard  # Adjust class name as needed
from main_menu import App  # Adjust class name as needed
# === Secondary Page Template ===
class DetailPage(QWidget):
    def __init__(self, title_text, student_name=None):
        super().__init__()

        self.setWindowTitle(title_text)
        self.setGeometry(200, 200, 400, 300)

        main_layout = QVBoxLayout(self)

        # === Top bar with "Back to Main Menu" button ===
        top_bar = QHBoxLayout()
        top_bar.addStretch()

        back_btn = QPushButton("Back to Main Menu")
        back_btn.setStyleSheet("""
            QPushButton {
                background-color: #ffffff;
                color: black;
                font-weight: bold;
                border-radius: 10px;
                padding: 5px 15px;
            }
            QPushButton:hover {
                background-color: #e0e0e0;
            }
        """)
        back_btn.clicked.connect(self.go_to_main_menu)
        top_bar.addWidget(back_btn)
        main_layout.addLayout(top_bar)

        # === Page content ===
        title = QLabel(title_text)
        title.setAlignment(Qt.AlignCenter)
        title.setFont(QFont("Arial", 16, QFont.Bold))
        main_layout.addWidget(title)

        if student_name:
            name_label = QLabel(f"Student: {student_name}")
            name_label.setAlignment(Qt.AlignCenter)
```

```
            name_label.setFont(QFont("Arial", 12))
            main_layout.addWidget(name_label)

    def go_to_main_menu(self):
        base_dir = os.path.dirname(os.path.abspath(__file__))
        menu_path = os.path.join(base_dir, "main_menu.py")
        if os.path.exists(menu_path):
            subprocess.Popen([sys.executable, menu_path])
        self.close()


# === Main Menu Window ===
class MainMenu(QWidget):
    def __init__(self, student_name=None, parent=None):
        super().__init__()
        self.parent_window = parent  # ✅ store reference
        self.student_name = student_name
        self.setWindowTitle("Medical Check-Up")
        self.setFixedSize(727, 411)

        # Background Image
        bg_path = os.path.join(os.path.dirname(__file__), "Bg (1).jpg")
        self.background = QLabel(self)
        if os.path.exists(bg_path):
            self.background.setPixmap(QPixmap(bg_path).scaled(
                self.size(), Qt.KeepAspectRatioByExpanding,
Qt.SmoothTransformation
            ))
        self.background.setGeometry(0, 0, 727, 411)

        # Transparent Overlay Container
        self.container = QWidget(self)
        self.container.setGeometry(0, 0, 727, 411)
        self.container.setStyleSheet("background: transparent;")

        self.create_ui()

    def create_ui(self):
        layout = QVBoxLayout(self.container)
        layout.setAlignment(Qt.AlignTop)
        layout.setSpacing(15)

        # === Top-right Back to Main Menu button ===
        top_bar = QHBoxLayout()
        top_bar.addStretch()

        back_btn = QPushButton("Back to Main Menu")
        back_btn.setStyleSheet("""
            QPushButton {
                background-color: #ffffff;
                color: black;
                font-weight: bold;
                border-radius: 10px;
                padding: 5px 15px;
```

```
            }
            QPushButton:hover {
                background-color: #e0e0e0;
            }
        """)
        back_btn.clicked.connect(self.go_to_main_menu)
        top_bar.addWidget(back_btn)
        layout.addLayout(top_bar)

        # === Title ===
        title = QLabel("MEDICAL CHECK-UP")
        title.setFont(QFont("Arial", 20, QFont.Bold))
        title.setAlignment(Qt.AlignCenter)
        title.setStyleSheet("color: black;")
        layout.addWidget(title)

        # Student name display (optional)
        if self.student_name:
            name_label = QLabel(f"Student: {self.student_name}")
            name_label.setAlignment(Qt.AlignCenter)
            name_label.setFont(QFont("Arial", 12))
            name_label.setStyleSheet("color: black;")
            layout.addWidget(name_label)

        # === Buttons ===
        button_info = {
            "PHYSICAL DETAILS": self.open_physical_details,
            "PEDIATRIC DETAILS": self.open_pediatric_checkup,
            "GENERAL CHECKUP": self.open_general_checkup,
            "DENTAL DETAILS": self.open_dental_checkup,
            "OPHTHALMIC": self.open_opthalmic_checkup,
            "VACCINATION DETAILS": self.open_vaccination_details,
        }

        for label, action in button_info.items():
            btn = QPushButton(label)
            btn.setFixedHeight(40)
            btn.setStyleSheet("""
                QPushButton {
                    background-color: white;
                    color: black;
                    font-weight: bold;
                    border-radius: 20px;
                }
                QPushButton:hover {
                    background-color: #e0e0e0;
                }
            """)
            btn.clicked.connect(action)
            layout.addWidget(btn)

    def go_to_main_menu(self):
        """Return to the previous (main menu) window without reopening."""
        if self.parent_window:
```

```
            self.parent_window.show()  # ✓ show the main menu again
        self.close()


    # === Button functions ===
    def open_physical_details(self):
        if self.student_name:

            from phydetinp import physicalinfo
            self.phydet_window =
physicalinfo(search_name=self.student_name, parent=self)
            self.hide()
            self.phydet_window.show()


    def open_general_checkup(self):
        if self.student_name:
            from stugencheck import generalcheckupinfo
            self.gencheck_window =
generalcheckupinfo(search_name=self.student_name,parent=self)
            self.hide()
            self.gencheck_window.show()


    def open_pediatric_checkup(self):
        from pedcheck import pediatricinfo
        self.pedcheck_window =
pediatricinfo(search_name=self.student_name,parent=self)
        self.hide()  # hide current window (don't close)
        self.pedcheck_window.show()

    def open_dental_checkup(self):
        from dentcheck import dentalinfo
        self.dentcheck_window =
dentalinfo(search_name=self.student_name,parent=self)
        self.hide()  # hide current window (don't close)
        self.dentcheck_window.show()


    def open_opthalmic_checkup(self):
        from stuopthdet import OpthalmicInfo
        self.opthcheck_window =
OpthalmicInfo(search_name=self.student_name,parent=self)
        self.hide()  # hide current window (don't close)
        self.opthcheck_window.show()


    def open_vaccination_details(self):
        from vaccwindow import VaccinationForm
        self.vacc_window =
VaccinationForm(search_name=self.student_name,parent=self)
        self.hide()  # hide current window (don't close)
        self.vacc_window.show()
```

```
    # Helper to open another script
    def launch_file(self, filename):
        base_dir = os.path.dirname(os.path.abspath(__file__))
        file_path = os.path.join(base_dir, filename)
        if os.path.exists(file_path):
            subprocess.Popen([sys.executable, file_path])
            self.close()
        else:
            print(f"⚠ File not found: {file_path}")


if __name__ == "__main__":
    # Check if student name was passed from main app
    student_name = None
    if len(sys.argv) > 1:
        student_name = sys.argv[1]

    app = QApplication(sys.argv)
    window = MainMenu(student_name=student_name)
    window.show()
    sys.exit(app.exec_())
```

PHYSICAL CHECKUP ENTRY WINDOW:

```
import os
import subprocess
import mysql.connector
from mysql.connector import Error
import sys
from PyQt5.QtWidgets import (
    QApplication, QWidget, QLabel, QLineEdit, QPushButton,
    QVBoxLayout, QGridLayout, QDateEdit, QMessageBox,
    QScrollArea, QHBoxLayout
)
from PyQt5.QtGui import QFont, QPalette, QBrush, QPixmap
from PyQt5.QtCore import Qt, QDate

class physicalinfo(QWidget):
    def __init__(self, search_name=None, parent=None):
        super().__init__()
        self.parent_window = parent  # ✓ store reference
        self.search_name = search_name
        self.setWindowTitle("PharmaLogix")
        self.setFixedSize(800, 500)  # Window size

        self.bg_path = os.path.join(os.path.dirname(__file__), "Bg
(1).jpg")

        # Form fields
        self.physicaldetails = [
            "UHID", "STUDENT NAME", "HEIGHT", "WEIGHT", "DATE OF CHECKUP"
        ]
```

```
        self.inputs = {}
        self.init_ui()

        # Initialize DB
        self.create_table_if_not_exists()

    def init_ui(self):
        self.setAutoFillBackground(True)
        # self.update_background()

        main_layout = QVBoxLayout(self)

        # Top layout for title and back button
        top_layout = QHBoxLayout()
        title_label = QLabel("STUDENT PHYSICAL DETAILS")
        title_label.setFont(QFont("Arial", 18, QFont.Bold))
        title_label.setAlignment(Qt.AlignCenter)
        top_layout.addWidget(title_label)
        top_layout.addStretch()

        return_btn = QPushButton("BACK")
        btn_style = """
            QPushButton {
                background: white;
                border-radius: 10px;
                padding: 6px 12px;
                font-weight: bold;
            }
            QPushButton:hover {
                background: lightgray;
            }
        """
        return_btn.setStyleSheet(btn_style)
        return_btn.clicked.connect(self.go_to_main_menu)
        top_layout.addWidget(return_btn)

        main_layout.addLayout(top_layout)

        # Scrollable form
        scroll_area = QScrollArea()
        scroll_area.setWidgetResizable(True)
        scroll_area.setStyleSheet("background: transparent; border:
none;")
        inner_widget = QWidget()
        inner_layout = QVBoxLayout(inner_widget)
        grid_layout = QGridLayout()
        grid_layout.setVerticalSpacing(15)
        grid_layout.setHorizontalSpacing(20)

        field_style = """
            QLineEdit, QDateEdit {
                background: white;
                border-radius: 10px;
```

```
                padding: 5px;
                font-size: 12px;
            }
            QLabel {
                background: rgba(255,255,255,0.8);
                border-radius: 10px;
                padding: 5px;
            }
        """

        for row, detail in enumerate(self.physicaldetails):
            label = QLabel(detail)
            label.setFont(QFont("Arial", 11, QFont.Bold))
            label.setStyleSheet(field_style)
            if detail == "DATE OF CHECKUP":
                input_field = QDateEdit()
                input_field.setCalendarPopup(True)
                input_field.setDate(QDate.currentDate())
            else:
                input_field = QLineEdit()
            input_field.setStyleSheet(field_style)
            if detail == "STUDENT NAME" and self.search_name:
                input_field.setText(self.search_name)
                input_field.setReadOnly(True)
            grid_layout.addWidget(label, row, 0)
            grid_layout.addWidget(input_field, row, 1)
            self.inputs[detail] = input_field

        inner_layout.addLayout(grid_layout)
        scroll_area.setWidget(inner_widget)
        main_layout.addWidget(scroll_area)

        # Submit button at bottom
        submit_btn = QPushButton("SUBMIT")
        submit_btn.setStyleSheet(btn_style)
        submit_btn.clicked.connect(self.submit_form)
        bottom_layout = QHBoxLayout()
        bottom_layout.addStretch()
        bottom_layout.addWidget(submit_btn)
        bottom_layout.addStretch()
        main_layout.addLayout(bottom_layout)

    def update_background(self):
        if os.path.exists(self.bg_path):
            palette = QPalette()
            pixmap = QPixmap(self.bg_path).scaled(
                self.size(), Qt.KeepAspectRatioByExpanding,
Qt.SmoothTransformation
            )
            palette.setBrush(QPalette.Window, QBrush(pixmap))
            self.setPalette(palette)

    def resizeEvent(self, event):
        self.update_background()
```

```
        super().resizeEvent(event)

    def go_to_main_menu(self):
        if self.parent_window:
            self.parent_window.show()  # ✅ show previous window again
        self.close()

    def submit_form(self):
        data = {}
        for key, widget in self.inputs.items():
            if isinstance(widget, QDateEdit):
                data[key] = widget.date().toString("yyyy-MM-dd")
            else:
                data[key] = widget.text().strip()

        # Basic validation
        if not data["UHID"] or not data["STUDENT NAME"]:
            QMessageBox.warning(self, "Missing Info", "Please fill UHID
and Student Name.")
            return

        conn = self.get_connection()
        if not conn:
            return
        try:
            cursor = conn.cursor()
            columns = ', '.join([f"`{col.replace(' ', '_')}`" for col in
data.keys()])
            placeholders = ', '.join(['%s'] * len(data))
            query = f"INSERT INTO physical_details ({columns}) VALUES
({placeholders})"
            cursor.execute(query, list(data.values()))
            conn.commit()
            QMessageBox.information(self, "Success", "Record saved
successfully!")
            # clear fields
            for widget in self.inputs.values():
                if isinstance(widget, QLineEdit):
                    widget.clear()
                elif isinstance(widget, QDateEdit):
                    widget.setDate(QDate.currentDate())
        except Error as e:
            QMessageBox.critical(self, "Database Error", f"Failed to
insert data:\n{e}")
        finally:
            cursor.close()
            conn.close()

    def get_connection(self):
            try:
                conn = mysql.connector.connect(
                    host="localhost",
                    user="root",
                    password="deens",
```

```
                    database="pharmalogix"
                )
                print("Database connected")
                return conn
            except Error as e:
                print("Database error")
                return None

    def create_table_if_not_exists(self):
            conn = self.get_connection()
            if not conn:
                return
            cursor = conn.cursor()
            try:
                cursor.execute("""
                    CREATE TABLE IF NOT EXISTS physical_details (

                        UHID VARCHAR(250),
                        STUDENT_NAME VARCHAR(250),
                        HEIGHT VARCHAR(250),
                        WEIGHT VARCHAR(250),
                        DATE_OF_CHECKUP DATE
                    )
                """)
                conn.commit()
                print("Table created or already exists")
            except Error as e:
                print("Database error")
                # QMessageBox.critical(self, "Database Error", f"Table
creation failed:\n{e}")
            finally:
                cursor.close()
                conn.close()
if __name__ == "__main__":
    print("hello")
    app = QApplication(sys.argv)
    form = physicalinfo()
    form.show()

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

PEDIATRIC CHECKUP ENTRY WINDOW:

```
import os
import subprocess
import mysql.connector
from mysql.connector import Error
import sys
from PyQt5.QtWidgets import (
    QApplication, QWidget, QLabel, QLineEdit, QPushButton,
    QVBoxLayout, QGridLayout, QDateEdit, QMessageBox,
    QScrollArea, QHBoxLayout
)
```

```
from PyQt5.QtGui import QFont, QPalette, QBrush, QPixmap
from PyQt5.QtCore import Qt, QDate

class pediatricinfo(QWidget):
    def __init__(self, search_name=None, parent=None):
        super().__init__()
        self.parent_window = parent  # ✅ store reference
        self.search_name = search_name
        self.setWindowTitle("PharmaLogix")
        self.setFixedSize(800, 500)  # Window size

        self.bg_path = os.path.join(os.path.dirname(__file__), "Bg
(1).jpg")

        # Form fields
        self.pediatricdetails = [
            "UHID", "STUDENT NAME", "BLOOD PRESSURE", "PULSE", "EYES",
"NOSE",
            "TONSILS", "SKIN", "CVS", "PA", "CNS", "MUSCULOSKELETAL",
            "FLAT FEET", "BOW LEGS", "AUDIOMETRY", "DOCTOR NAME", "DATE OF
CHECKUP"
        ]

        self.inputs = {}
        self.init_ui()

        # Initialize DB
        self.create_table_if_not_exists()

    def init_ui(self):
        self.setAutoFillBackground(True)
        # self.update_background()

        main_layout = QVBoxLayout(self)

        # Top layout for title and back button
        top_layout = QHBoxLayout()
        title_label = QLabel("STUDENT PEDIATRIC DETAILS")
        title_label.setFont(QFont("Arial", 18, QFont.Bold))
        title_label.setAlignment(Qt.AlignCenter)
        top_layout.addWidget(title_label)
        top_layout.addStretch()

        return_btn = QPushButton("BACK")
        btn_style = """
            QPushButton {
                background: white;
                border-radius: 10px;
                padding: 6px 12px;
                font-weight: bold;
            }
            QPushButton:hover {
                background: lightgray;
            }
```

```
        """
        return_btn.setStyleSheet(btn_style)
        return_btn.clicked.connect(self.go_to_main_menu)
        top_layout.addWidget(return_btn)

        main_layout.addLayout(top_layout)

        # Scrollable form
        scroll_area = QScrollArea()
        scroll_area.setWidgetResizable(True)
        scroll_area.setStyleSheet("background: transparent; border:
none;")
        inner_widget = QWidget()
        inner_layout = QVBoxLayout(inner_widget)
        grid_layout = QGridLayout()
        grid_layout.setVerticalSpacing(15)
        grid_layout.setHorizontalSpacing(20)

        field_style = """
            QLineEdit, QDateEdit {
                background: white;
                border-radius: 10px;
                padding: 5px;
                font-size: 12px;
            }
            QLabel {
                background: rgba(255,255,255,0.8);
                border-radius: 10px;
                padding: 5px;
            }
        """

        for row, detail in enumerate(self.pediatricdetails):
            label = QLabel(detail)
            label.setFont(QFont("Arial", 11, QFont.Bold))
            label.setStyleSheet(field_style)
            if detail == "DATE OF CHECKUP":
                input_field = QDateEdit()
                input_field.setCalendarPopup(True)
                input_field.setDate(QDate.currentDate())
            else:
                input_field = QLineEdit()
            input_field.setStyleSheet(field_style)
            if detail == "STUDENT NAME" and self.search_name:
                input_field.setText(self.search_name)
                input_field.setReadOnly(True)
            grid_layout.addWidget(label, row, 0)
            grid_layout.addWidget(input_field, row, 1)
            self.inputs[detail] = input_field

        inner_layout.addLayout(grid_layout)
        scroll_area.setWidget(inner_widget)
        main_layout.addWidget(scroll_area)
```

```
        # Submit button at bottom
        submit_btn = QPushButton("SUBMIT")
        submit_btn.setStyleSheet(btn_style)
        submit_btn.clicked.connect(self.submit_form)
        bottom_layout = QHBoxLayout()
        bottom_layout.addStretch()
        bottom_layout.addWidget(submit_btn)
        bottom_layout.addStretch()
        main_layout.addLayout(bottom_layout)

    def update_background(self):
        if os.path.exists(self.bg_path):
            palette = QPalette()
            pixmap = QPixmap(self.bg_path).scaled(
                self.size(), Qt.KeepAspectRatioByExpanding,
Qt.SmoothTransformation
            )
            palette.setBrush(QPalette.Window, QBrush(pixmap))
            self.setPalette(palette)

    def resizeEvent(self, event):
        self.update_background()
        super().resizeEvent(event)

    def go_to_main_menu(self):
        if self.parent_window:
            self.parent_window.show()  # ✅ show previous window again
        self.close()

    def submit_form(self):
        data = {}
        for key, widget in self.inputs.items():
            if isinstance(widget, QDateEdit):
                data[key] = widget.date().toString("yyyy-MM-dd")
            else:
                data[key] = widget.text().strip()

        # Basic validation
        if not data["UHID"] or not data["STUDENT NAME"]:
            QMessageBox.warning(self, "Missing Info", "Please fill UHID
and Student Name.")
            return

        conn = self.get_connection()
        if not conn:
            return
        try:
            cursor = conn.cursor()
            columns = ', '.join([f"`{col.replace(' ', '_')}`" for col in
data.keys()])
            placeholders = ', '.join(['%s'] * len(data))
            query = f"INSERT INTO pediatric_details ({columns}) VALUES
({placeholders})"
            cursor.execute(query, list(data.values()))
```

```
            conn.commit()
            QMessageBox.information(self, "Success", "Record saved
successfully!")
            # clear fields
            for widget in self.inputs.values():
                if isinstance(widget, QLineEdit):
                    widget.clear()
                elif isinstance(widget, QDateEdit):
                    widget.setDate(QDate.currentDate())
        except Error as e:
            QMessageBox.critical(self, "Database Error", f"Failed to
insert data:\n{e}")
        finally:
            cursor.close()
            conn.close()

    def get_connection(self):
            try:
                conn = mysql.connector.connect(
                    host="localhost",
                    user="root",
                    password="deens",
                    database="pharmalogix"
                )
                print("Database connected")
                return conn
            except Error as e:
                print("Database error")
                return None

    def create_table_if_not_exists(self):
            conn = self.get_connection()
            if not conn:
                return
            cursor = conn.cursor()
            try:
                cursor.execute("""
                    CREATE TABLE IF NOT EXISTS pediatric_details (
                        id INT AUTO_INCREMENT PRIMARY KEY,
                        UHID VARCHAR(250),
                        STUDENT_NAME VARCHAR(250),
                        BLOOD_PRESSURE VARCHAR(250),
                        PULSE VARCHAR(250),
                        EYES VARCHAR(250),
                        NOSE VARCHAR(250),
                        TONSILS VARCHAR(250),
                        SKIN VARCHAR(250),
                        CVS VARCHAR(250),
                        PA VARCHAR(250),
                        CNS VARCHAR(250),
                        MUSCULOSKELETAL VARCHAR(250),
                        FLAT_FEET VARCHAR(250),
                        BOW_LEGS VARCHAR(250),
                        AUDIOMETRY VARCHAR(250),
```

```
                        DOCTOR_NAME VARCHAR(250),
                        DATE_OF_CHECKUP DATE
                    )
                """)
                conn.commit()
                print("Table created or already exists")
            except Error as e:
                print("Database error")
                # QMessageBox.critical(self, "Database Error", f"Table
creation failed:\n{e}")
            finally:
                cursor.close()
                conn.close()
if __name__ == "__main__":
    print("hello")
    app = QApplication(sys.argv)
    form = pediatricinfo()
    form.show()

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

GENERAL CHECKUP ENTRY WINDOW:

```
import os
import subprocess
import mysql.connector
from mysql.connector import Error
import sys
from PyQt5.QtWidgets import (
    QApplication, QWidget, QLabel, QLineEdit, QPushButton,
    QVBoxLayout, QGridLayout, QDateEdit, QMessageBox,
    QScrollArea, QHBoxLayout
)
from PyQt5.QtGui import QFont, QPalette, QBrush, QPixmap
from PyQt5.QtCore import Qt, QDate

class generalcheckupinfo(QWidget):
    def __init__(self, search_name=None, parent=None):
        super().__init__()
        self.parent_window = parent  # ✓ store reference
        self.search_name = search_name
        self.setWindowTitle("PharmaLogix")
        self.setFixedSize(800, 500)  # Window size

        self.bg_path = os.path.join(os.path.dirname(__file__), "Bg
(1).jpg")

        # Form fields
        self.generalcheckupdetails = [
            "UHID", "STUDENT NAME", "CLINICAL EXAMINATION", "GENERAL",
"SYSTEM EXAMINATION", "SALIENT OBSERVATIONS",
            "COMPLAINTS", "DIAGNOSIS", "NAILS","INVESTIGATION ADVISED",
"RESULT OF INVESTIGATION","PLAN OF ACTION",
```

```
            "MEDICATION PRESCRIBED", "DOCTOR NAME", "DATE OF CHECKUP"
        ]

        self.inputs = {}
        self.init_ui()

        # Initialize DB
        self.create_table_if_not_exists()

    def init_ui(self):
        self.setAutoFillBackground(True)
        # self.update_background()

        main_layout = QVBoxLayout(self)

        # Top layout for title and back button
        top_layout = QHBoxLayout()
        title_label = QLabel("STUDENT GENERAL CHECKUP DETAILS")
        title_label.setFont(QFont("Arial", 18, QFont.Bold))
        title_label.setAlignment(Qt.AlignCenter)
        top_layout.addWidget(title_label)
        top_layout.addStretch()

        return_btn = QPushButton("BACK")
        btn_style = """
            QPushButton {
                background: white;
                border-radius: 10px;
                padding: 6px 12px;
                font-weight: bold;
            }
            QPushButton:hover {
                background: lightgray;
            }
        """
        return_btn.setStyleSheet(btn_style)
        return_btn.clicked.connect(self.go_to_main_menu)
        top_layout.addWidget(return_btn)

        main_layout.addLayout(top_layout)

        # Scrollable form
        scroll_area = QScrollArea()
        scroll_area.setWidgetResizable(True)
        scroll_area.setStyleSheet("background: transparent; border:
none;")
        inner_widget = QWidget()
        inner_layout = QVBoxLayout(inner_widget)
        grid_layout = QGridLayout()
        grid_layout.setVerticalSpacing(15)
        grid_layout.setHorizontalSpacing(20)

        field_style = """
            QLineEdit, QDateEdit {
```

```
                background: white;
                border-radius: 10px;
                padding: 5px;
                font-size: 12px;
            }
            QLabel {
                background: rgba(255,255,255,0.8);
                border-radius: 10px;
                padding: 5px;
            }
        """

        for row, detail in enumerate(self.generalcheckupdetails):
            label = QLabel(detail)
            label.setFont(QFont("Arial", 11, QFont.Bold))
            label.setStyleSheet(field_style)
            if detail == "DATE OF CHECKUP":
                input_field = QDateEdit()
                input_field.setCalendarPopup(True)
                input_field.setDate(QDate.currentDate())
            else:
                input_field = QLineEdit()
            input_field.setStyleSheet(field_style)
            if detail == "STUDENT NAME" and self.search_name:
                input_field.setText(self.search_name)
                input_field.setReadOnly(True)
            grid_layout.addWidget(label, row, 0)
            grid_layout.addWidget(input_field, row, 1)
            self.inputs[detail] = input_field

        inner_layout.addLayout(grid_layout)
        scroll_area.setWidget(inner_widget)
        main_layout.addWidget(scroll_area)

        # Submit button at bottom
        submit_btn = QPushButton("SUBMIT")
        submit_btn.setStyleSheet(btn_style)
        submit_btn.clicked.connect(self.submit_form)
        bottom_layout = QHBoxLayout()
        bottom_layout.addStretch()
        bottom_layout.addWidget(submit_btn)
        bottom_layout.addStretch()
        main_layout.addLayout(bottom_layout)

    def update_background(self):
        if os.path.exists(self.bg_path):
            palette = QPalette()
            pixmap = QPixmap(self.bg_path).scaled(
                self.size(), Qt.KeepAspectRatioByExpanding,
Qt.SmoothTransformation
            )
            palette.setBrush(QPalette.Window, QBrush(pixmap))
            self.setPalette(palette)
```

```
    def resizeEvent(self, event):
        self.update_background()
        super().resizeEvent(event)

    def go_to_main_menu(self):
        if self.parent_window:
            self.parent_window.show()  # ✅ show previous window again
        self.close()

    def submit_form(self):
        data = {}
        for key, widget in self.inputs.items():
            if isinstance(widget, QDateEdit):
                data[key] = widget.date().toString("yyyy-MM-dd")
            else:
                data[key] = widget.text().strip()

        # Basic validation
        if not data["UHID"] or not data["STUDENT NAME"]:
            QMessageBox.warning(self, "Missing Info", "Please fill UHID
and Student Name.")
            return

        conn = self.get_connection()
        if not conn:
            return
        try:
            cursor = conn.cursor()
            columns = ', '.join([f"`{col.replace(' ', '_')}`" for col in
data.keys()])
            placeholders = ', '.join(['%s'] * len(data))
            query = f"INSERT INTO general_checkup_details ({columns})
VALUES ({placeholders})"
            cursor.execute(query, list(data.values()))
            conn.commit()
            QMessageBox.information(self, "Success", "Record saved
successfully!")
            # clear fields
            for widget in self.inputs.values():
                if isinstance(widget, QLineEdit):
                    widget.clear()
                elif isinstance(widget, QDateEdit):
                    widget.setDate(QDate.currentDate())
        except Error as e:
            QMessageBox.critical(self, "Database Error", f"Failed to
insert data:\n{e}")
        finally:
            cursor.close()
            conn.close()

    def get_connection(self):
            try:
                conn = mysql.connector.connect(
                    host="localhost",
```

```
                    user="root",
                    password="deens",
                    database="pharmalogix"
                )
                print("Database connected")
                return conn
            except Error as e:
                print("Database error")
                return None

    def create_table_if_not_exists(self):
            conn = self.get_connection()
            if not conn:
                return
            cursor = conn.cursor()
            try:
                cursor.execute("""
                    CREATE TABLE IF NOT EXISTS general_checkup_details (
                        id INT AUTO_INCREMENT PRIMARY KEY,
                        UHID VARCHAR(250),
                        STUDENT_NAME VARCHAR(250),
                        CLINICAL_EXAMINATION VARCHAR(250),
                        GENERAL VARCHAR(250),
                        SYSTEM_EXAMINATION VARCHAR(250),
                        SALIENT_OBSERVATIONS VARCHAR(250),
                        COMPLAINTS VARCHAR(250),
                        DIAGNOSIS VARCHAR(250),
                        NAILS VARCHAR(250),
                        INVESTIGATION_ADVISED VARCHAR(250),
                        RESULT_OF_INVESTIGATION VARCHAR(250),
                        PLAN_OF_ACTION VARCHAR(250),
                        MEDICATION_PRESCRIBED VARCHAR(250),
                        DOCTOR_NAME VARCHAR(250),
                        DATE_OF_CHECKUP DATE
                    )
                """)
                conn.commit()
                print("Table created or already exists")
            except Error as e:
                print("Database error")
                # QMessageBox.critical(self, "Database Error", f"Table
creation failed:\n{e}")
            finally:
                cursor.close()
                conn.close()
if __name__ == "__main__":
    print("hello")
    app = QApplication(sys.argv)
    form = generalcheckupinfo()
    form.show()

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

DENTAL CHECKUP ENTRY WINDOW:

```
import os
import subprocess
import mysql.connector
from mysql.connector import Error
import sys
from PyQt5.QtWidgets import (
    QApplication, QWidget, QLabel, QLineEdit, QPushButton,
    QVBoxLayout, QGridLayout, QDateEdit, QMessageBox,
    QScrollArea, QHBoxLayout, QComboBox
)
from PyQt5.QtGui import QFont, QPalette, QBrush, QPixmap
from PyQt5.QtCore import Qt, QDate

class dentalinfo(QWidget):
    def __init__(self, search_name=None, parent=None):
        super().__init__()
        self.parent_window = parent  # ✓ store reference
        self.search_name = search_name
        self.setWindowTitle("PharmaLogix")
        self.setFixedSize(800, 500)  # Window size

        self.bg_path = os.path.join(os.path.dirname(__file__), "Bg
(1).jpg")

        # Form fields
        self.dentaldetails = [
            "UHID", "STUDENT NAME", "ORAL HYGIENE STATUS", "CLINICAL
FINDINGS",
            "CARIES STATUS", "CARIES RISK PROFILE", "TREATMENT ADVICE",
            "DIETARY MODIFICATION", "NOTES", "DOCTOR NAME", "DATE OF
CHECKUP"
        ]

        self.inputs = {}
        self.init_ui()
        self.create_table_if_not_exists()

    def init_ui(self):
        self.setAutoFillBackground(True)

        main_layout = QVBoxLayout(self)

        # --- Title and Back Button ---
        top_layout = QHBoxLayout()
        title_label = QLabel("STUDENT DENTAL DETAILS")
        title_label.setFont(QFont("Arial", 18, QFont.Bold))
        title_label.setAlignment(Qt.AlignCenter)
        top_layout.addWidget(title_label)
```

```
        top_layout.addStretch()


        return_btn = QPushButton("BACK")
        btn_style = """
            QPushButton {
                background: white;
                border-radius: 10px;
                padding: 6px 12px;
                font-weight: bold;
            }
            QPushButton:hover {
                background: lightgray;
            }
        """
        return_btn.setStyleSheet(btn_style)
        return_btn.clicked.connect(self.go_to_main_menu)
        top_layout.addWidget(return_btn)
        main_layout.addLayout(top_layout)

        # --- Scroll Area with Form ---
        scroll_area = QScrollArea()
        scroll_area.setWidgetResizable(True)
        scroll_area.setStyleSheet("background: transparent; border:
none;")

        inner_widget = QWidget()
        inner_layout = QVBoxLayout(inner_widget)
        grid_layout = QGridLayout()
        grid_layout.setVerticalSpacing(15)
        grid_layout.setHorizontalSpacing(20)

        field_style = """
            QLineEdit, QDateEdit, QComboBox {
                background: white;
                border-radius: 10px;
                padding: 5px;
                font-size: 12px;
            }
            QLabel {
                background: rgba(255,255,255,0.8);
                border-radius: 10px;
                padding: 5px;
            }
        """

        for row, detail in enumerate(self.dentaldetails):
            label = QLabel(detail)
            label.setFont(QFont("Arial", 11, QFont.Bold))
            label.setStyleSheet(field_style)


            # Dropdowns for specific fields
            if detail == "ORAL HYGIENE STATUS":
```

```
                input_field = QComboBox()
                input_field.addItems(["Select Option","GOOD", "FAIR",
"POOR"])
            elif detail == "CARIES STATUS":
                input_field = QComboBox()
                input_field.addItems(["Select Option","MILD", "MODERATE",
"SEVERE"])
            elif detail == "CARIES RISK PROFILE":
                input_field = QComboBox()
                input_field.addItems(["Select Option","LOW", "MEDIUM",
"HIGH"])
            elif detail == "DATE OF CHECKUP":
                input_field = QDateEdit()
                input_field.setCalendarPopup(True)
                input_field.setDate(QDate.currentDate())
            else:
                input_field = QLineEdit()

            input_field.setStyleSheet(field_style)
            if detail == "STUDENT NAME" and self.search_name:
                input_field.setText(self.search_name)
                input_field.setReadOnly(True)
            grid_layout.addWidget(label, row, 0)
            grid_layout.addWidget(input_field, row, 1)
            self.inputs[detail] = input_field

        inner_layout.addLayout(grid_layout)
        scroll_area.setWidget(inner_widget)
        main_layout.addWidget(scroll_area)

        # --- Submit Button ---
        submit_btn = QPushButton("SUBMIT")
        submit_btn.setStyleSheet(btn_style)
        submit_btn.clicked.connect(self.submit_form)
        bottom_layout = QHBoxLayout()
        bottom_layout.addStretch()
        bottom_layout.addWidget(submit_btn)
        bottom_layout.addStretch()
        main_layout.addLayout(bottom_layout)

    def update_background(self):
        if os.path.exists(self.bg_path):
            palette = QPalette()
            pixmap = QPixmap(self.bg_path).scaled(
                self.size(), Qt.KeepAspectRatioByExpanding,
Qt.SmoothTransformation
            )
            palette.setBrush(QPalette.Window, QBrush(pixmap))
            self.setPalette(palette)

    def resizeEvent(self, event):
        self.update_background()
        super().resizeEvent(event)
```

```
    def go_to_main_menu(self):
        if self.parent_window:
            self.parent_window.show()  # ✅ show previous window again
        self.close()

    def submit_form(self):
        data = {}
        for key, widget in self.inputs.items():
            if isinstance(widget, QDateEdit):
                data[key] = widget.date().toString("yyyy-MM-dd")
            elif isinstance(widget, QComboBox):
                data[key] = widget.currentText()
            else:
                data[key] = widget.text().strip()

        if not data["UHID"] or not data["STUDENT NAME"]:
            QMessageBox.warning(self, "Missing Info", "Please fill UHID
and Student Name.")
            return

        conn = self.get_connection()
        if not conn:
            return
        try:
            cursor = conn.cursor()
            columns = ', '.join([f"`{col.replace(' ', '_')}`" for col in
data.keys()])
            placeholders = ', '.join(['%s'] * len(data))
            query = f"INSERT INTO dental_details ({columns}) VALUES
({placeholders})"
            cursor.execute(query, list(data.values()))
            conn.commit()
            QMessageBox.information(self, "Success", "Record saved
successfully!")

            # Clear fields
            for widget in self.inputs.values():
                if isinstance(widget, QLineEdit):
                    widget.clear()
                elif isinstance(widget, QDateEdit):
                    widget.setDate(QDate.currentDate())
                elif isinstance(widget, QComboBox):
                    widget.setCurrentIndex(0)
        except Error as e:
            QMessageBox.critical(self, "Database Error", f"Failed to
insert data:\n{e}")
        finally:
            cursor.close()
            conn.close()

    def get_connection(self):
        try:
            conn = mysql.connector.connect(
                host="localhost",
```

```
                user="root",
                password="deens",
                database="pharmalogix"
            )
            print("Database connected")
            return conn
        except Error as e:
            print("Database error:", e)
            return None

    def create_table_if_not_exists(self):
        conn = self.get_connection()
        if not conn:
            return
        cursor = conn.cursor()
        try:
            cursor.execute("""
                CREATE TABLE IF NOT EXISTS dental_details (
                    id INT AUTO_INCREMENT PRIMARY KEY,
                    UHID VARCHAR(250),
                    STUDENT_NAME VARCHAR(250),
                    ORAL_HYGIENE_STATUS VARCHAR(250),
                    CLINICAL_FINDINGS VARCHAR(250),
                    CARIES_STATUS VARCHAR(250),
                    CARIES_RISK_PROFILE VARCHAR(250),
                    TREATMENT_ADVICE VARCHAR(250),
                    DIETARY_MODIFICATION VARCHAR(250),
                    NOTES VARCHAR(250),
                    DOCTOR_NAME VARCHAR(250),
                    DATE_OF_CHECKUP DATE
                )
            """)
            conn.commit()
            print("Table created or already exists")
        except Error as e:
            print("Database error:", e)
        finally:
            cursor.close()
            conn.close()

if __name__ == "__main__":
    print("hello")
    app = QApplication(sys.argv)
    form = dentalinfo()
    form.show()
    sys.exit(app.exec_())
```

OPTHALMIC CHECKUP ENTRY WINDOW:

```
import os
import sys
import subprocess
import mysql.connector
```

```
from mysql.connector import Error
from PyQt5.QtWidgets import (
    QApplication, QWidget, QLabel, QLineEdit, QPushButton,
    QVBoxLayout, QGridLayout, QDateEdit, QMessageBox,
    QScrollArea, QHBoxLayout, QTableWidget, QTableWidgetItem, QHeaderView
)
from PyQt5.QtGui import QFont
from PyQt5.QtCore import Qt, QDate


class OpthalmicInfo(QWidget):
    def __init__(self, search_name=None, parent=None):
        super().__init__()
        self.parent_window = parent  # ✅ store reference
        self.search_name = search_name
        self.setWindowTitle("PharmaLogix - Ophthalmic Details")
        self.resize(800, 700)


        self.db_name = "pharmalogix"
        self.bg_path = os.path.join(os.path.dirname(__file__), "Bg
(1).jpg")

        self.set_background()

        self.fields = [
            "UHID", "STUDENT NAME", "HEAD POSTURE", "CORNEAL",
            "COVER TEST", "EXTERNAL OCULAR MOVEMENT", "CONVERGENCE",
            "EYE COMPLAINTS", "DOCTOR NAME", "DATE OF CHECKUP"
        ]
        self.inputs = {}


        self.create_table_if_not_exists()
        self.init_ui()
    def set_background(self):
            """Set background image for the entire window."""
            from PyQt5.QtGui import QPalette, QBrush, QPixmap
            self.setAutoFillBackground(True)
            palette = QPalette()
            pixmap = QPixmap(self.bg_path).scaled(
                self.size(), Qt.IgnoreAspectRatio, Qt.SmoothTransformation
            )
            palette.setBrush(QPalette.Window, QBrush(pixmap))
            self.setPalette(palette)

    def init_ui(self):
        main_layout = QVBoxLayout(self)

        # # ---------- HEADER BAR ----------
        header_bar = QHBoxLayout()
```

```
top_layout = QHBoxLayout()
title_label = QLabel("STUDENT OPTHALMIC DETAILS")
title_label.setFont(QFont("Arial", 18, QFont.Bold))
title_label.setAlignment(Qt.AlignCenter)
top_layout.addWidget(title_label)
top_layout.addStretch()


return_btn = QPushButton("BACK")
btn_style = """
    QPushButton {
        background: white;
        border-radius: 10px;
        padding: 6px 12px;
        font-weight: bold;
    }
    QPushButton:hover {
        background: lightgray;
    }
"""
return_btn.setStyleSheet(btn_style)
return_btn.clicked.connect(self.go_to_main_menu)
top_layout.addWidget(return_btn)
main_layout.addLayout(top_layout)

# centered heading
# title = QLabel("STUDENT OPHTHALMIC DETAILS")
# title.setFont(QFont("Arial", 18, QFont.Bold))
# title.setAlignment(Qt.AlignCenter)

# fill space on right side so title stays centered
right_spacer = QLabel()
right_spacer.setFixedWidth(40)

header_bar.addStretch()
header_bar.addWidget(title_label)
header_bar.addStretch()
header_bar.addWidget(right_spacer)

main_layout.addLayout(header_bar)
main_layout.addSpacing(10)

# ---------- SCROLL AREA ----------
scroll = QScrollArea()
scroll.setWidgetResizable(True)
scroll_content = QWidget()
scroll_layout = QVBoxLayout(scroll_content)

# ---------- FORM ----------
form_layout = QGridLayout()
form_layout.setVerticalSpacing(10)
form_layout.setHorizontalSpacing(25)

for i, field in enumerate(self.fields):
```

```
            label = QLabel(field)
            label.setFont(QFont("Arial", 11))
            if field == "DATE OF CHECKUP":
                input_widget = QDateEdit()
                input_widget.setCalendarPopup(True)
                input_widget.setDate(QDate.currentDate())
            else:
                input_widget = QLineEdit()
                input_widget.setPlaceholderText(f"Enter
{field.lower()}...")

            input_widget.setStyleSheet("""
                QLineEdit, QDateEdit {
                    background: white;
                    border-radius: 8px;
                    padding: 5px;
                    font-size: 12px;
                }
                QLabel {
                    font-weight: bold;
                    background: rgba(255,255,255,0.8);
                    border-radius: 5px;
                    padding: 4px;
                }
            """)
            if field == "STUDENT NAME" and self.search_name:
                input_widget.setText(self.search_name)
                input_widget.setReadOnly(True)
            form_layout.addWidget(label, i, 0)
            form_layout.addWidget(input_widget, i, 1)
            self.inputs[field] = input_widget

        scroll_layout.addLayout(form_layout)

        # ---------- TABLE ----------
        table_label = QLabel("VISION DETAILS")
        table_label.setFont(QFont("Arial", 14, QFont.Bold))
        table_label.setAlignment(Qt.AlignCenter)
        scroll_layout.addWidget(table_label)

        self.vision_table = QTableWidget()
        self.vision_table.setRowCount(9)
        self.vision_table.setColumnCount(3)
        self.vision_table.setHorizontalHeaderLabels(["VISION", "LEFT EYE",
"RIGHT EYE"])

        # equal width columns + colored header

self.vision_table.horizontalHeader().setSectionResizeMode(QHeaderView.Stre
tch)
        self.vision_table.horizontalHeader().setStyleSheet("""
            QHeaderView::section {
                background-color: #a5c9ff;
                color: black;
```

```
                font-weight: bold;
                font-size: 13px;
                border: 1px solid lightgray;
                padding: 4px;
            }
        """)

        # no scrollbars

self.vision_table.setVerticalScrollBarPolicy(Qt.ScrollBarAlwaysOff)

self.vision_table.setHorizontalScrollBarPolicy(Qt.ScrollBarAlwaysOff)
        self.vision_table.verticalHeader().setVisible(False)

        vision_labels = [
            "UNAIDED", "AIDED", "DRY REFRACTION", "CYCLOPLEGIC
REFRACTION",
            "GLASSES", "ANT. SEGMENT", "FUNDUS", "COLOR VISION", "I/O"
        ]
        for i, label in enumerate(vision_labels):
            vision_item = QTableWidgetItem(label)
            vision_item.setFlags(Qt.ItemIsEnabled)
            self.vision_table.setItem(i, 0, vision_item)
            self.vision_table.setItem(i, 1, QTableWidgetItem(""))
            self.vision_table.setItem(i, 2, QTableWidgetItem(""))

        scroll_layout.addWidget(self.vision_table)

        # ---------- SUBMIT BUTTON ----------
        submit_btn = QPushButton("SUBMIT")
        submit_btn.setStyleSheet("""
            QPushButton {
                background: white;
                border-radius: 8px;
                padding: 6px 12px;
                font-weight: bold;
            }
            QPushButton:hover { background: lightgray; }
        """)
        submit_btn.clicked.connect(self.submit_form)
        scroll_layout.addWidget(submit_btn, alignment=Qt.AlignCenter)

        scroll.setWidget(scroll_content)
        main_layout.addWidget(scroll)

    # ---------- Navigation ----------
    def go_to_main_menu(self):
        if self.parent_window:
            self.parent_window.show()  # ✅ show previous window again
        self.close()

    # ---------- Database ----------
    def get_connection(self):
        try:
```

```
            return mysql.connector.connect(
                host="localhost", user="root", password="deens",
database=self.db_name
            )
        except Error:
            QMessageBox.critical(self, "Error", "Database connection
failed.")
            return None

    def create_table_if_not_exists(self):
        conn = self.get_connection()
        if not conn:
            return
        try:
            cursor = conn.cursor()
            cursor.execute("""
                CREATE TABLE IF NOT EXISTS opthalmic_details (
                    id INT AUTO_INCREMENT PRIMARY KEY,
                    UHID VARCHAR(250),
                    STUDENT_NAME VARCHAR(250),
                    HEAD_POSTURE VARCHAR(250),
                    CORNEAL VARCHAR(250),
                    COVER_TEST VARCHAR(250),
                    EXTERNAL_OCULAR_MOVEMENT VARCHAR(250),
                    CONVERGENCE VARCHAR(250),
                    EYE_COMPLAINTS VARCHAR(250),
                    DOCTOR_NAME VARCHAR(250),
                    DATE_OF_CHECKUP DATE
                )
            """)
            conn.commit()
        finally:
            cursor.close()
            conn.close()

    def submit_form(self):
        data = {}
        for key, widget in self.inputs.items():
            if isinstance(widget, QDateEdit):
                data[key] = widget.date().toString("yyyy-MM-dd")
            else:
                data[key] = widget.text().strip()

        if not data["UHID"] or not data["STUDENT NAME"]:
            QMessageBox.warning(self, "Missing Info", "Please fill UHID
and Student Name.")
            return

        conn = self.get_connection()
        if not conn:
            return
        try:
            cursor = conn.cursor()
```

```
            columns = ', '.join([f"`{col.replace(' ', '_')}`" for col in
data.keys()])
            placeholders = ', '.join(['%s'] * len(data))
            query = f"INSERT INTO opthalmic_details ({columns}) VALUES
({placeholders})"
            cursor.execute(query, list(data.values()))
            conn.commit()
            QMessageBox.information(self, "Success", "Record saved
successfully!")
            for widget in self.inputs.values():
                if isinstance(widget, QLineEdit):
                    widget.clear()
                elif isinstance(widget, QDateEdit):
                    widget.setDate(QDate.currentDate())
        except Error as e:
            QMessageBox.critical(self, "Database Error", f"Failed to
insert data:\n{e}")
        finally:
            cursor.close()
            conn.close()


if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = OpthalmicInfo()
    window.show()
  sys.exit(app.exec_())
```

VACCINATION ENTRY WINDOW:

```
import os
import subprocess
import sys
import mysql.connector
from mysql.connector import Error

from PyQt5.QtWidgets import (
    QApplication, QWidget, QLabel, QRadioButton,
    QLineEdit, QPushButton, QGridLayout, QMessageBox,
    QVBoxLayout, QHBoxLayout, QButtonGroup, QDateEdit
)
from PyQt5.QtGui import QPixmap, QPalette, QBrush, QFont
from PyQt5.QtCore import Qt, QDate


class VaccinationForm(QWidget):
    def __init__(self, search_name=None, parent=None):
        super().__init__()
        self.parent_window = parent  # ✓ store reference
        self.search_name = search_name
        self.setWindowTitle("PharmaLogix")
        self.setFixedSize(800, 500)
```

```
        self.set_background("Bg (1).jpg")
        self.set_app_font()

        self.vaccines = [
            "DPT", "HEPATITIS B", "BCG", "PENTAVALE NT",
            "ROTAVIRUS", "MEASLES RUBELLA", "ORAL POLIO",
            "PCV PNEUMONIA", "JE - JAPANESE ENCEPHALITIS"
        ]

        self.radio_buttons = {}
        self.init_ui()

    def set_background(self, image_path):
        self.setAutoFillBackground(True)
        palette = QPalette()
        pixmap = QPixmap(image_path).scaled(self.size(),
Qt.IgnoreAspectRatio, Qt.SmoothTransformation)
        palette.setBrush(QPalette.Window, QBrush(pixmap))
        self.setPalette(palette)

    def set_app_font(self):
        font = QFont("Arial", 12)
        font.setBold(True)
        QApplication.setFont(font)

    def go_to_main_menu(self):
        if self.parent_window:
            self.parent_window.show()  # ✓ show previous window again
        self.close()
    def init_ui(self):
        layout = QGridLayout()
        self.setAutoFillBackground(True)

        main_layout = QVBoxLayout(self)

        # --- Title and Back Button ---
        top_layout = QHBoxLayout()

        title_label = QLabel("STUDENT VACCINATION DETAILS")
        title_label.setFont(QFont("Arial", 16, QFont.Bold))
        title_label.setAlignment(Qt.AlignCenter)
        # title_label.setFixedWidth(600)

        # layout.addStretch()

        return_btn = QPushButton("BACK")
        btn_style = """
            QPushButton {
                background: white;
                border-radius: 8px;
                padding: 6px 12px;
                font-weight: bold;
            }
            QPushButton:hover {
```

```
                background: lightgray;
            }
        """
        return_btn.setStyleSheet(btn_style)
        return_btn.setFixedWidth(100)
        # return_btn.setAlignment(Qt.AlignRight)
        return_btn.clicked.connect(self.go_to_main_menu)


        top_layout.addWidget(title_label, 2, Qt.AlignCenter)
        top_layout.addWidget(return_btn,0, Qt.AlignRight)
        main_layout.addLayout(top_layout)
        main_layout.addLayout(layout)

        # # Add large title label at the top
        # title_label = QLabel("VACCINATION HISTORY")
        # title_font = QFont("Arial", 24, QFont.Bold)
        # title_label.setFont(title_font)
        # title_label.setAlignment(Qt.AlignCenter)
        # title_label.setStyleSheet("background-color: white; padding:
4px; border-radius: 5px;")
        # layout.addWidget(title_label, 0, 0, 1, 3)  # Span 3 columns

        row = 1  # Start other widgets from row 1
        uhid_label = QLabel("UHID")
        uhid_label.setStyleSheet("background-color: white; padding: 4px;
border-radius: 5px;")
        self.uhid_input = QLineEdit()
        layout.addWidget(uhid_label, row, 0)
        layout.addWidget(self.uhid_input, row, 1, 1, 2)
        row += 1
        student_label = QLabel("STUDENT NAME")
        student_label.setStyleSheet("background-color: white; padding:
4px; border-radius: 5px;")
        self.student_input = QLineEdit()
        if self.search_name:
            self.student_input.setText(self.search_name)
            self.student_input.setReadOnly(True)
        layout.addWidget(student_label, row, 0)
        layout.addWidget(self.student_input, row, 1, 1, 2)
        row += 1
        for vaccine in self.vaccines:
            label = QLabel(vaccine)
            label.setStyleSheet("background-color: white; padding: 4px;
border-radius: 5px;")

            yes_button = QRadioButton("YES")
            no_button = QRadioButton("NO")

            # Default to NO
            no_button.setChecked(True)

            # Group YES/NO per vaccine
            button_group = QButtonGroup(self)
```

```
            button_group.addButton(yes_button)
            button_group.addButton(no_button)

            layout.addWidget(label, row, 0)
            layout.addWidget(yes_button, row, 1)
            layout.addWidget(no_button, row, 2)
            self.radio_buttons[vaccine] = (yes_button, no_button)
            row += 1

        # Doctor name input

        doctor_label = QLabel("DOCTOR NAME")
        doctor_label.setStyleSheet("background-color: white; padding: 4px;
border-radius: 5px;")
        self.doctor_input = QLineEdit()
        layout.addWidget(doctor_label, row, 0)
        layout.addWidget(self.doctor_input, row, 1, 1, 2)
        row += 1

        # Calendar-based date picker
        date_label = QLabel("DATE OF CHECKUP")
        date_label.setStyleSheet("background-color: white; padding: 4px;
border-radius: 5px;")
        self.date_input = QDateEdit()
        self.date_input.setCalendarPopup(True)
        self.date_input.setDisplayFormat("dd-MM-yyyy")
        self.date_input.setDate(QDate.currentDate())  # Set current date
as default
        layout.addWidget(date_label, row, 0)
        layout.addWidget(self.date_input, row, 1, 1, 2)
        row += 1

        # Buttons
        submit_btn = QPushButton("SUBMIT")
        return_btn = QPushButton("RETURN TO SEARCH")
        add_btn = QPushButton("ADD CATEGORY")
        submit_btn.setFixedSize(250, 35)
        return_btn.setFixedSize(250, 35)
        add_btn.setFixedSize(250, 35)

        submit_btn.clicked.connect(self.submit_form)
        return_btn.clicked.connect(self.go_to_main_menu)
        layout.addWidget(submit_btn, row, 1)
        # layout.addWidget(return_btn, row, 1)
        # layout.addWidget(add_btn, row, 2)

        self.setLayout(layout)

    def connect_to_database(self):
        try:
                conn = mysql.connector.connect(
                    host="localhost",
                    user="root",
                    password="deens",
```

```
                    database="pharmalogix"
                )
                print("Database connected")
                return conn
        except Error as e:
                print("Database error")
                return None
    def create_table_if_not_exists(self):
            conn = self.connect_to_database()
            if not conn:
                print("No database connection exists")
                return
            cursor = conn.cursor()
            try:
                cursor.execute("""
                    CREATE TABLE IF NOT EXISTS vaccination_details (
                        UHID VARCHAR(250),
                        STUDENT_NAME VARCHAR(250),
                        DPT VARCHAR(5),
                        HEPATITIS_B VARCHAR(5),
                        BCG VARCHAR(5),
                        PENTAVALE_NT VARCHAR(5),
                        ROTAVIRUS VARCHAR(5),
                        MEASLES_RUBELLA VARCHAR(5),
                        ORAL_POLIO VARCHAR(5),
                        PCV_PNEUMONIA VARCHAR(5),
                        JE_JAPANESE_ENCEPHALITIS VARCHAR(5),
                        DOCTOR_NAME VARCHAR(250),
                        DATE_OF_CHECKUP DATE
                    )
                """)
                conn.commit()
                print("Table created or already exists")
            except Error as e:
                print("Database error")
                # QMessageBox.critical(self, "Database Error", f"Table
creation failed:\n{e}")
            finally:
                cursor.close()
                conn.close()

        # return mysql.connector.connect(
        #     host="localhost",
        #     user="root",
        #     password="deens",
        #     database="pharmalogix"
        # )

    def submit_form(self):
        try:
            data = {
                "UHID": self.uhid_input.text().strip(),
                "STUDENT_NAME": self.student_input.text().strip(),
                "doctor_name": self.doctor_input.text().strip(),
```

```
                "date_of_checkup": self.date_input.date().toString("yyyy-
MM-dd")
            }

            for vaccine, (yes, no) in self.radio_buttons.items():
                key = vaccine.lower().replace(" ", "_").replace("-",
"").replace("__", "_")
                if yes.isChecked():
                    value = "YES"
                elif no.isChecked():
                    value = "NO"
                else:
                    QMessageBox.warning(self, "Input Error", f"Please
select YES or NO for '{vaccine}'.")
                    return
                data[key] = value
            self.create_table_if_not_exists()
            db = self.connect_to_database()

            cursor = db.cursor()

            columns = ', '.join(data.keys())
            values = ', '.join(['%s'] * len(data))
            sql = f"INSERT INTO vaccination_details ({columns}) VALUES
({values})"

            cursor.execute(sql, list(data.values()))
            db.commit()

            QMessageBox.information(self, "Success", "Vaccination record
submitted successfully!")

            cursor.close()
            db.close()
        except Exception as e:
            QMessageBox.critical(self, "Database Error", f"Failed to
submit data:\n{e}")


if __name__ == "__main__":
    app = QApplication(sys.argv)
    form = VaccinationForm()
    form.show()
    sys.exit(app.exec_())
```

STUDENT HISTORY MAIN WINDOW:

```
import sys
import os
import subprocess
from PyQt5.QtWidgets import (
    QApplication, QWidget, QLabel, QPushButton, QVBoxLayout, QHBoxLayout,
QMessageBox
```

```
)
from PyQt5.QtGui import QFont, QPixmap
from PyQt5.QtCore import Qt
from pedhis import PediatricHistory
from phyhis import PhysicalHistory
from genhis import GeneralCheckupHistory
from denthis import DentalHistory
from opthhis import OpthalmicHistory
from vacchis import VaccinationHistory

student_name = None
if len(sys.argv) > 1:
    student_name = sys.argv[1]
# === Secondary Page Template ===
class DetailPage(QWidget):
    def __init__(self, title_text, student_name=None):
        super().__init__()
        self.setWindowTitle(title_text)
        self.setGeometry(200, 200, 400, 300)

        main_layout = QVBoxLayout(self)

        # === Top bar with "Back to Main Menu" button ===
        top_bar = QHBoxLayout()
        top_bar.addStretch()

        back_btn = QPushButton("Back to Main Menu")
        back_btn.setStyleSheet("""
            QPushButton {
                background-color: #ffffff;
                color: black;
                font-weight: bold;
                border-radius: 10px;
                padding: 5px 15px;
            }
            QPushButton:hover {
                background-color: #e0e0e0;
            }
        """)
        back_btn.clicked.connect(self.go_to_main_menu)
        top_bar.addWidget(back_btn)
        main_layout.addLayout(top_bar)

        # === Page content ===
        title = QLabel(title_text)
        title.setAlignment(Qt.AlignCenter)
        title.setFont(QFont("Arial", 16, QFont.Bold))
        main_layout.addWidget(title)

        if student_name:
            name_label = QLabel(f"Student: {student_name}")
            name_label.setAlignment(Qt.AlignCenter)
            name_label.setFont(QFont("Arial", 12))
            main_layout.addWidget(name_label)
```

```
    def go_to_main_menu(self):
        base_dir = os.path.dirname(os.path.abspath(__file__))
        menu_path = os.path.join(base_dir, "main_menu.py")
        if os.path.exists(menu_path):
            subprocess.Popen([sys.executable, menu_path])
        self.close()


# === Main Menu Window ===
class MainMenu(QWidget):
    def __init__(self, student_name=None, parent=None):
        super().__init__()
        self.student_name = student_name
        self.parent_window = parent
        self.setWindowTitle("Medical Check-Up")
        self.setFixedSize(727, 411)

        # Background Image
        bg_path = os.path.join(os.path.dirname(__file__), "Bg (1).jpg")
        self.background = QLabel(self)
        if os.path.exists(bg_path):
            self.background.setPixmap(QPixmap(bg_path).scaled(
                self.size(), Qt.KeepAspectRatioByExpanding,
Qt.SmoothTransformation
            ))
        self.background.setGeometry(0, 0, 727, 411)

        # Transparent Overlay Container
        self.container = QWidget(self)
        self.container.setGeometry(0, 0, 727, 411)
        self.container.setStyleSheet("background: transparent;")

        self.create_ui()

    def create_ui(self):
        layout = QVBoxLayout(self.container)
        layout.setAlignment(Qt.AlignTop)
        layout.setSpacing(15)

        # === Top-right Back to Main Menu button ===
        top_bar = QHBoxLayout()
        top_bar.addStretch()

        back_btn = QPushButton("Back to Main Menu")
        back_btn.setStyleSheet("""
            QPushButton {
                background-color: #ffffff;
                color: black;
                font-weight: bold;
                border-radius: 10px;
                padding: 5px 15px;
            }
            QPushButton:hover {
```

```
            background-color: #e0e0e0;
        }
    """)
    back_btn.clicked.connect(self.go_to_main_menu)
    top_bar.addWidget(back_btn)
    layout.addLayout(top_bar)

    # === Title ===
    title = QLabel("MEDICAL CHECK-UP")
    title.setFont(QFont("Arial", 20, QFont.Bold))
    title.setAlignment(Qt.AlignCenter)
    title.setStyleSheet("color: black;")
    layout.addWidget(title)

    # Student name display (optional)
    if self.student_name:
        name_label = QLabel(f"Student: {self.student_name}")
        name_label.setAlignment(Qt.AlignCenter)
        name_label.setFont(QFont("Arial", 12))
        name_label.setStyleSheet("color: black;")
        layout.addWidget(name_label)

    # === Buttons ===
    button_info = {
        "PHYSICAL HISTORY": self.open_physical_history,
        "PEDIATRIC HISTORY": self.open_pediatric_history,
        "GENERAL CHECKUP HISTORY": self.open_general_checkup_history,
        "DENTAL HISTORY": self.open_dental_history,
        "OPHTHALMIC HISTORY": self.open_opthalmic_history,
        "VACCINATION HISTORY": self.open_vaccination_history,
    }

    for label, action in button_info.items():
        btn = QPushButton(label)
        btn.setFixedHeight(40)
        btn.setStyleSheet("""
            QPushButton {
                background-color: white;
                color: black;
                font-weight: bold;
                border-radius: 20px;
            }
            QPushButton:hover {
                background-color: #e0e0e0;
            }
        """)
        btn.clicked.connect(action)
        layout.addWidget(btn)

def go_to_main_menu(self):
    """Return to the previous (main menu) window without reopening."""
    if self.parent_window:
        self.parent_window.show()  # ✓ show the main menu again
    self.close()
```

```
    # === Button functions ===
    def open_physical_history(self):
        if self.student_name:
            from phyhis import PhysicalHistory  # local import to avoid
circular import
            self.phy_window = PhysicalHistory(self.student_name,
parent=self)  # ✅ pass self as parent
            self.hide()  # hide current window (don't close)
            self.phy_window.show()
        else:
            QMessageBox.warning(None, "Error", "No student selected!")


    def open_general_checkup_history(self):
        if self.student_name:
            from genhis import GeneralCheckupHistory  # local import to
avoid circular import
            self.gen_window = GeneralCheckupHistory(self.student_name,
parent=self)  # ✅ pass self as parent
            self.hide()  # hide current window (don't close)
            self.gen_window.show()
        else:
            QMessageBox.warning(None, "Error", "No student selected!")

    def open_pediatric_history(self):
        if self.student_name:
            from pedhis import PediatricHistory  # local import to avoid
circular import
            self.ped_window = PediatricHistory(self.student_name,
parent=self)  # ✅ pass self as parent
            self.hide()  # hide current window (don't close)
            self.ped_window.show()
        else:
            QMessageBox.warning(None, "Error", "No student selected!")

    def open_dental_history(self):
        if self.student_name:
            from denthis import DentalHistory  # local import to avoid
circular import
            self.dent_window = DentalHistory(self.student_name,
parent=self)  # ✅ pass self as parent
            self.hide()  # hide current window (don't close)
            self.dent_window.show()
        else:
            QMessageBox.warning(None, "Error", "No student selected!")

    def open_opthalmic_history(self):
        if self.student_name:
            from opthhis import OpthalmicHistory  # local import to avoid
circular import
            self.opt_window = OpthalmicHistory(self.student_name,
parent=self)  # ✅ pass self as parent
```

```
            self.hide()  # hide current window (don't close)
            self.opt_window.show()
        else:
            QMessageBox.warning(None, "Error", "No student selected!")

    def open_vaccination_history(self):
        if self.student_name:
            from vacchis import VaccinationHistory  # local import to
avoid circular import
            self.vacc_window = VaccinationHistory(self.student_name,
parent=self)  # ✅ pass self as parent
            self.hide()  # hide current window (don't close)
            self.vacc_window.show()
        else:
            QMessageBox.warning(None, "Error", "No student selected!")

    # Helper to open another script
    def launch_file(self, filename):
        base_dir = os.path.dirname(os.path.abspath(__file__))
        file_path = os.path.join(base_dir, filename)
        if os.path.exists(file_path):
            subprocess.Popen([sys.executable, file_path])
            self.close()
        else:
            print(f"⚠️ File not found: {file_path}")


if __name__ == "__main__":
    # Check if student name was passed from main app
    student_name = None
    if len(sys.argv) > 1:
        student_name = sys.argv[1]

    app = QApplication(sys.argv)
    window = MainMenu(student_name=student_name)
    window.show()
    sys.exit(app.exec_())
```

PHYSICAL HISTORY WINDOW:

```
import sys
import os
import subprocess
import mysql.connector
from mysql.connector import Error
from PyQt5.QtWidgets import (
    QApplication, QWidget, QVBoxLayout, QPushButton,
    QScrollArea, QFrame, QMessageBox, QLabel, QHBoxLayout, QHeaderView
)
from PyQt5.QtGui import QFont, QPalette, QBrush, QPixmap
from PyQt5.QtCore import Qt
```

```
class PhysicalHistory(QWidget):
    def __init__(self, search_name=None, parent=None):
        super().__init__()
        self.parent_window = parent  # ✅ store reference
        self.search_name = search_name
        self.setWindowTitle("PharmaLogix - Physical History")
        self.setFixedSize(800, 500)
        self.bg_path = os.path.join(os.path.dirname(__file__), "Bg
(1).jpg")
        self.search_name = search_name  # store search term

        self.init_ui()
        self.load_data_from_db()

    def init_ui(self):
        self.setAutoFillBackground(True)
        self.update_background()

        layout = QVBoxLayout(self)

        # Top layout with title and back button
        top_layout = QHBoxLayout()
        title = QLabel("PHYISCAL DETAILS HISTORY")
        title.setFont(QFont("Arial", 18, QFont.Bold))
        title.setAlignment(Qt.AlignCenter)
        top_layout.addWidget(title)
        top_layout.addStretch()

        back_btn = QPushButton("BACK")
        back_btn.setStyleSheet("""
            QPushButton {
                background: white;
                border-radius: 10px;
                padding: 6px 12px;
                font-weight: bold;
            }
            QPushButton:hover {
                background: lightgray;
            }
        """)
        back_btn.clicked.connect(self.go_back)
        top_layout.addWidget(back_btn)
        layout.addLayout(top_layout)

        # === Scroll area for data ===
        scroll_area = QScrollArea()
        scroll_area.setWidgetResizable(True)
        scroll_content = QWidget()
        self.scroll_layout = QVBoxLayout(scroll_content)
        scroll_area.setWidget(scroll_content)
        layout.addWidget(scroll_area)


    def update_background(self):
```

```
        if os.path.exists(self.bg_path):
            palette = QPalette()
            pixmap = QPixmap(self.bg_path).scaled(
                self.size(), Qt.KeepAspectRatioByExpanding,
Qt.SmoothTransformation
            )
            palette.setBrush(QPalette.Window, QBrush(pixmap))
            self.setPalette(palette)

    def resizeEvent(self, event):
        self.update_background()
        super().resizeEvent(event)

    # --------- MySQL Data Fetch ----------
    def load_data_from_db(self):
        try:
            conn = mysql.connector.connect(
                host="localhost",
                user="root",        # update if needed
                password="deens",        # update if you have a MySQL
password
                database="pharmalogix"
            )
            cursor = conn.cursor()

            if self.search_name:  # search by student name
                query = "SELECT * FROM physical_checkup WHERE student_name
= %s"
                cursor.execute(query, (self.search_name,))
            else:  # fetch all records
                cursor.execute("SELECT * FROM physical_checkup")

            data = cursor.fetchall()
            columns = [desc[0] for desc in cursor.description]

            if not data:
                QMessageBox.information(self, "No Data",
                                        "No physical records found for
this student.")
                return

            # === Build Record Blocks ===
            for row_idx, row_data in enumerate(data):
                record_frame = QFrame()
                record_frame.setStyleSheet("""
                    QFrame {
                        background-color: rgba(255,255,255,0.9);
                        border-radius: 10px;
                        border: 1px solid gray;
                        padding: 12px;
                        margin-bottom: 15px;
                    }
                """)
                record_layout = QVBoxLayout(record_frame)
```

```
                for col_name, value in zip(columns, row_data):
                    row_layout = QHBoxLayout()

                    # Column name
                    label_key = QLabel(f"{col_name.replace('_', '
').upper()}:")
                    label_key.setFont(QFont("Arial", 11, QFont.Bold))
                    label_key.setFixedWidth(200)

                    # Column value
                    label_value = QLabel(str(value))
                    label_value.setFont(QFont("Arial", 11))
                    label_value.setStyleSheet("color: #222;")

                    row_layout.addWidget(label_key)
                    row_layout.addWidget(label_value)
                    row_layout.addStretch()
                    record_layout.addLayout(row_layout)

                # Add each record block to scroll layout
                self.scroll_layout.addWidget(record_frame)



        except Error as e:
            QMessageBox.critical(self, "Database Error", f"Error while
fetching data:\n{e}")

        finally:
            if 'conn' in locals() and conn.is_connected():
                cursor.close()
                conn.close()

    # ---------- Go to Main Menu ----------
    def go_back(self):
        if self.parent_window:
            self.parent_window.show()  # ✅ show previous window again
        self.close()  # ✅ close this one



if __name__ == "__main__":
    app = QApplication(sys.argv)

    # Example usage: pass a name to search
    search_name = None  # set to a string like "John Doe" to filter
    window = PhysicalHistory(search_name)

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

PEDIATRIC HISTORY WINDOW:

```
import sys
import os
import subprocess
import mysql.connector
from mysql.connector import Error
from PyQt5.QtWidgets import (
    QApplication, QWidget, QVBoxLayout, QPushButton,
    QScrollArea, QFrame, QMessageBox, QLabel, QHBoxLayout, QHeaderView
)
from PyQt5.QtGui import QFont, QPalette, QBrush, QPixmap
from PyQt5.QtCore import Qt

class PediatricHistory(QWidget):
    def __init__(self, search_name=None, parent=None):
        super().__init__()
        self.parent_window = parent  # ✓ store reference
        self.search_name = search_name
        self.setWindowTitle("PharmaLogix - Pediatric History")
        self.setFixedSize(800, 500)
        self.bg_path = os.path.join(os.path.dirname(__file__), "Bg.jpg")
        self.search_name = search_name  # store search term

        self.init_ui()
        self.load_data_from_db()

    def init_ui(self):
        self.setAutoFillBackground(True)
        self.update_background()

        layout = QVBoxLayout(self)

        # Top layout with title and back button
        top_layout = QHBoxLayout()
        title = QLabel("PEDIATRIC DETAILS HISTORY")
        title.setFont(QFont("Arial", 18, QFont.Bold))
        title.setAlignment(Qt.AlignCenter)
        top_layout.addWidget(title)
        top_layout.addStretch()

        back_btn = QPushButton("BACK")
        back_btn.setStyleSheet("""
            QPushButton {
                background: white;
                border-radius: 10px;
                padding: 6px 12px;
                font-weight: bold;
            }
            QPushButton:hover {
                background: lightgray;
```

```
            }
        """)
        back_btn.clicked.connect(self.go_back)
        top_layout.addWidget(back_btn)
        layout.addLayout(top_layout)

        # === Scroll area for data ===
        scroll_area = QScrollArea()
        scroll_area.setWidgetResizable(True)
        scroll_content = QWidget()
        self.scroll_layout = QVBoxLayout(scroll_content)
        scroll_area.setWidget(scroll_content)
        layout.addWidget(scroll_area)


    def update_background(self):
        if os.path.exists(self.bg_path):
            palette = QPalette()
            pixmap = QPixmap(self.bg_path).scaled(
                self.size(), Qt.KeepAspectRatioByExpanding,
Qt.SmoothTransformation
            )
            palette.setBrush(QPalette.Window, QBrush(pixmap))
            self.setPalette(palette)

    def resizeEvent(self, event):
        self.update_background()
        super().resizeEvent(event)

    # --------- MySQL Data Fetch ----------
    def load_data_from_db(self):
        try:
            conn = mysql.connector.connect(
                host="localhost",
                user="root",        # update if needed
                password="deens",        # update if you have a MySQL
password
                database="pharmalogix"
            )
            cursor = conn.cursor()

            if self.search_name:  # search by student name
                query = "SELECT * FROM pediatric_details WHERE
student_name = %s"
                cursor.execute(query, (self.search_name,))
            else:  # fetch all records
                cursor.execute("SELECT * FROM pediatric_details")

            data = cursor.fetchall()
            columns = [desc[0] for desc in cursor.description]

            if not data:
                QMessageBox.information(self, "No Data",
```

```
                                        "No pediatric records found for
this student.")
                return

            # === Build Record Blocks ===
            for row_idx, row_data in enumerate(data):
                record_frame = QFrame()
                record_frame.setStyleSheet("""
                    QFrame {
                        background-color: rgba(255,255,255,0.9);
                        border-radius: 10px;
                        border: 1px solid gray;
                        padding: 12px;
                        margin-bottom: 15px;
                    }
                """)
                record_layout = QVBoxLayout(record_frame)

                for col_name, value in zip(columns, row_data):
                    row_layout = QHBoxLayout()

                    # Column name
                    label_key = QLabel(f"{col_name.replace('_', '
').upper()}:")
                    label_key.setFont(QFont("Arial", 11, QFont.Bold))
                    label_key.setFixedWidth(200)

                    # Column value
                    label_value = QLabel(str(value))
                    label_value.setFont(QFont("Arial", 11))
                    label_value.setStyleSheet("color: #222;")

                    row_layout.addWidget(label_key)
                    row_layout.addWidget(label_value)
                    row_layout.addStretch()
                    record_layout.addLayout(row_layout)

                # Add each record block to scroll layout
                self.scroll_layout.addWidget(record_frame)



        except Error as e:
            QMessageBox.critical(self, "Database Error", f"Error while
fetching data:\n{e}")

        finally:
            if 'conn' in locals() and conn.is_connected():
                cursor.close()
                conn.close()

    # ---------- Go to Main Menu ----------
    def go_back(self):
        if self.parent_window:
```

```
            self.parent_window.show()  # ✓ show previous window again
        self.close()


if __name__ == "__main__":
    app = QApplication(sys.argv)

    # Example usage: pass a name to search
    search_name = None  # set to a string like "John Doe" to filter
    window = PediatricHistory(search_name)

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

GENERAL CHECKUP HISTORY WINDOW:

```
import sys
import os
import subprocess
import mysql.connector
from mysql.connector import Error
from PyQt5.QtWidgets import (
    QApplication, QWidget, QVBoxLayout, QPushButton,
    QScrollArea, QFrame, QMessageBox, QLabel, QHBoxLayout, QHeaderView
)
from PyQt5.QtGui import QFont, QPalette, QBrush, QPixmap
from PyQt5.QtCore import Qt


class GeneralCheckupHistory(QWidget):
    def __init__(self, search_name=None, parent=None):
        super().__init__()
        self.parent_window = parent  # ✓ store reference
        self.search_name = search_name
        self.setWindowTitle("PharmaLogix - General Checkup History")
        self.setFixedSize(800, 500)
        self.bg_path = os.path.join(os.path.dirname(__file__), "Bg
(1).jpg")
        self.search_name = search_name  # store search term

        self.init_ui()
        self.load_data_from_db()

    def init_ui(self):
        self.setAutoFillBackground(True)
        self.update_background()

        layout = QVBoxLayout(self)

        # Top layout with title and back button
        top_layout = QHBoxLayout()
        title = QLabel("GENERAL CHECKUP HISTORY")
```

```
        title.setFont(QFont("Arial", 18, QFont.Bold))
        title.setAlignment(Qt.AlignCenter)
        top_layout.addWidget(title)
        top_layout.addStretch()

        back_btn = QPushButton("BACK")
        back_btn.setStyleSheet("""
            QPushButton {
                background: white;
                border-radius: 10px;
                padding: 6px 12px;
                font-weight: bold;
            }
            QPushButton:hover {
                background: lightgray;
            }
        """)
        back_btn.clicked.connect(self.go_back)
        top_layout.addWidget(back_btn)
        layout.addLayout(top_layout)

        # === Scroll area for data ===
        scroll_area = QScrollArea()
        scroll_area.setWidgetResizable(True)
        scroll_content = QWidget()
        self.scroll_layout = QVBoxLayout(scroll_content)
        scroll_area.setWidget(scroll_content)
        layout.addWidget(scroll_area)


    def update_background(self):
        if os.path.exists(self.bg_path):
            palette = QPalette()
            pixmap = QPixmap(self.bg_path).scaled(
                self.size(), Qt.KeepAspectRatioByExpanding,
Qt.SmoothTransformation
            )
            palette.setBrush(QPalette.Window, QBrush(pixmap))
            self.setPalette(palette)

    def resizeEvent(self, event):
        self.update_background()
        super().resizeEvent(event)

    # --------- MySQL Data Fetch ----------
    def load_data_from_db(self):
        try:
            conn = mysql.connector.connect(
                host="localhost",
                user="root",        # update if needed
                password="deens",        # update if you have a MySQL
password
                database="pharmalogix"
            )
```

```
            cursor = conn.cursor()

            if self.search_name:  # search by student name
                query = "SELECT * FROM general_checkup_details WHERE
student_name = %s"
                cursor.execute(query, (self.search_name,))
            else:  # fetch all records
                cursor.execute("SELECT * FROM general_checkup_details")

            data = cursor.fetchall()
            columns = [desc[0] for desc in cursor.description]

            if not data:
                QMessageBox.information(self, "No Data",
                                        "No general checkup records found
for this student.")
                return

            # === Build Record Blocks ===
            for row_idx, row_data in enumerate(data):
                record_frame = QFrame()
                record_frame.setStyleSheet("""
                    QFrame {
                        background-color: rgba(255,255,255,0.9);
                        border-radius: 10px;
                        border: 1px solid gray;
                        padding: 12px;
                        margin-bottom: 15px;
                    }
                """)
                record_layout = QVBoxLayout(record_frame)

                for col_name, value in zip(columns, row_data):
                    row_layout = QHBoxLayout()

                    # Column name
                    label_key = QLabel(f"{col_name.replace('_', '
').upper()}:")
                    label_key.setFont(QFont("Arial", 11, QFont.Bold))
                    label_key.setFixedWidth(200)

                    # Column value
                    label_value = QLabel(str(value))
                    label_value.setFont(QFont("Arial", 11))
                    label_value.setStyleSheet("color: #222;")

                    row_layout.addWidget(label_key)
                    row_layout.addWidget(label_value)
                    row_layout.addStretch()
                    record_layout.addLayout(row_layout)

                # Add each record block to scroll layout
                self.scroll_layout.addWidget(record_frame)
```

```
        except Error as e:
            QMessageBox.critical(self, "Database Error", f"Error while
fetching data:\n{e}")

        finally:
            if 'conn' in locals() and conn.is_connected():
                cursor.close()
                conn.close()

    # ---------- Go to Main Menu ----------
    def go_back(self):
        if self.parent_window:
            self.parent_window.show()  # ✓ show previous window again
        self.close()


if __name__ == "__main__":
    app = QApplication(sys.argv)

    # Example usage: pass a name to search
    search_name = None  # set to a string like "John Doe" to filter
    window = GeneralCheckupHistory(search_name)

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

DENTAL HISTORY WINDOW:

```
import sys
import os
import subprocess
import mysql.connector
from mysql.connector import Error
from PyQt5.QtWidgets import (
    QApplication, QWidget, QVBoxLayout, QPushButton,
    QScrollArea, QFrame, QMessageBox, QLabel, QHBoxLayout, QHeaderView
)
from PyQt5.QtGui import QFont, QPalette, QBrush, QPixmap
from PyQt5.QtCore import Qt


class DentalHistory(QWidget):
    def __init__(self, search_name=None, parent=None):
        super().__init__()
        self.parent_window = parent  # ✓ store reference
        self.search_name = search_name
        self.setWindowTitle("PharmaLogix - Dental History")
        self.setFixedSize(800, 500)
        self.bg_path = os.path.join(os.path.dirname(__file__), "Bg
(1).jpg")
        self.search_name = search_name  # store search term
```

```
        self.init_ui()
        self.load_data_from_db()

    def init_ui(self):
        self.setAutoFillBackground(True)
        self.update_background()

        layout = QVBoxLayout(self)

        # Top layout with title and back button
        top_layout = QHBoxLayout()
        title = QLabel("DENTAL HISTORY")
        title.setFont(QFont("Arial", 18, QFont.Bold))
        title.setAlignment(Qt.AlignCenter)
        top_layout.addWidget(title)
        top_layout.addStretch()

        back_btn = QPushButton("BACK")
        back_btn.setStyleSheet("""
            QPushButton {
                background: white;
                border-radius: 10px;
                padding: 6px 12px;
                font-weight: bold;
            }
            QPushButton:hover {
                background: lightgray;
            }
        """)
        back_btn.clicked.connect(self.go_back)
        top_layout.addWidget(back_btn)
        layout.addLayout(top_layout)

        # === Scroll area for data ===
        scroll_area = QScrollArea()
        scroll_area.setWidgetResizable(True)
        scroll_content = QWidget()
        self.scroll_layout = QVBoxLayout(scroll_content)
        scroll_area.setWidget(scroll_content)
        layout.addWidget(scroll_area)


    def update_background(self):
        if os.path.exists(self.bg_path):
            palette = QPalette()
            pixmap = QPixmap(self.bg_path).scaled(
                self.size(), Qt.KeepAspectRatioByExpanding,
Qt.SmoothTransformation
            )
            palette.setBrush(QPalette.Window, QBrush(pixmap))
            self.setPalette(palette)

    def resizeEvent(self, event):
```

```
        self.update_background()
        super().resizeEvent(event)

    # --------- MySQL Data Fetch ----------
    def load_data_from_db(self):
        try:
            conn = mysql.connector.connect(
                host="localhost",
                user="root",        # update if needed
                password="deens",        # update if you have a MySQL
password
                database="pharmalogix"
            )
            cursor = conn.cursor()

            if self.search_name:  # search by student name
                query = "SELECT * FROM dental_details WHERE student_name =
%s"
                cursor.execute(query, (self.search_name,))
            else:  # fetch all records
                cursor.execute("SELECT * FROM dental_details")

            data = cursor.fetchall()
            columns = [desc[0] for desc in cursor.description]

            if not data:
                QMessageBox.information(self, "No Data",
                                        "No dental records found for this
student.")
                return

            # === Build Record Blocks ===
            for row_idx, row_data in enumerate(data):
                record_frame = QFrame()
                record_frame.setStyleSheet("""
                    QFrame {
                        background-color: rgba(255,255,255,0.9);
                        border-radius: 10px;
                        border: 1px solid gray;
                        padding: 12px;
                        margin-bottom: 15px;
                    }
                """)
                record_layout = QVBoxLayout(record_frame)

                for col_name, value in zip(columns, row_data):
                    row_layout = QHBoxLayout()

                    # Column name
                    label_key = QLabel(f"{col_name.replace('_', '
').upper()}:")
                    label_key.setFont(QFont("Arial", 11, QFont.Bold))
                    label_key.setFixedWidth(200)
```

```
                    # Column value
                    label_value = QLabel(str(value))
                    label_value.setFont(QFont("Arial", 11))
                    label_value.setStyleSheet("color: #222;")

                    row_layout.addWidget(label_key)
                    row_layout.addWidget(label_value)
                    row_layout.addStretch()
                    record_layout.addLayout(row_layout)

                # Add each record block to scroll layout
                self.scroll_layout.addWidget(record_frame)



        except Error as e:
            QMessageBox.critical(self, "Database Error", f"Error while
fetching data:\n{e}")

        finally:
            if 'conn' in locals() and conn.is_connected():
                cursor.close()
                conn.close()

    # ---------- Go to Main Menu ----------
    def go_back(self):
        if self.parent_window:
            self.parent_window.show()  # ✓ show previous window again
        self.close()


if __name__ == "__main__":
    app = QApplication(sys.argv)

    # Example usage: pass a name to search
    search_name = None  # set to a string like "John Doe" to filter
    window = DentalHistory(search_name)

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

OPTHALMIC HISTORY WINDOW:

```
import sys
import os
import subprocess
import mysql.connector
from mysql.connector import Error
from PyQt5.QtWidgets import (
    QApplication, QWidget, QVBoxLayout, QPushButton,
    QScrollArea, QFrame, QMessageBox, QLabel, QHBoxLayout, QHeaderView
)
from PyQt5.QtGui import QFont, QPalette, QBrush, QPixmap
```

```
from PyQt5.QtCore import Qt


class OpthalmicHistory(QWidget):
    def __init__(self, search_name=None, parent=None):
        super().__init__()
        self.parent_window = parent  # ✅ store reference
        self.search_name = search_name
        self.setWindowTitle("PharmaLogix - Opthalmic History")
        self.setFixedSize(800, 500)
        self.bg_path = os.path.join(os.path.dirname(__file__), "Bg
(1).jpg")
        self.search_name = search_name  # store search term

        self.init_ui()
        self.load_data_from_db()

    def init_ui(self):
        self.setAutoFillBackground(True)
        self.update_background()

        layout = QVBoxLayout(self)

        # Top layout with title and back button
        top_layout = QHBoxLayout()
        title = QLabel("OPTHALMIC HISTORY")
        title.setFont(QFont("Arial", 18, QFont.Bold))
        title.setAlignment(Qt.AlignCenter)
        top_layout.addWidget(title)
        top_layout.addStretch()

        back_btn = QPushButton("BACK")
        back_btn.setStyleSheet("""
            QPushButton {
                background: white;
                border-radius: 10px;
                padding: 6px 12px;
                font-weight: bold;
            }
            QPushButton:hover {
                background: lightgray;
            }
        """)
        back_btn.clicked.connect(self.go_back)
        top_layout.addWidget(back_btn)
        layout.addLayout(top_layout)

        # === Scroll area for data ===
        scroll_area = QScrollArea()
        scroll_area.setWidgetResizable(True)
        scroll_content = QWidget()
        self.scroll_layout = QVBoxLayout(scroll_content)
        scroll_area.setWidget(scroll_content)
        layout.addWidget(scroll_area)
```

```
    def update_background(self):
        if os.path.exists(self.bg_path):
            palette = QPalette()
            pixmap = QPixmap(self.bg_path).scaled(
                self.size(), Qt.KeepAspectRatioByExpanding,
Qt.SmoothTransformation
            )
            palette.setBrush(QPalette.Window, QBrush(pixmap))
            self.setPalette(palette)

    def resizeEvent(self, event):
        self.update_background()
        super().resizeEvent(event)

    # --------- MySQL Data Fetch ----------
    def load_data_from_db(self):
        try:
            conn = mysql.connector.connect(
                host="localhost",
                user="root",        # update if needed
                password="deens",        # update if you have a MySQL
password
                database="pharmalogix"
            )
            cursor = conn.cursor()

            if self.search_name:  # search by student name
                query = "SELECT * FROM opthalmic_details WHERE
student_name = %s"
                cursor.execute(query, (self.search_name,))
            else:  # fetch all records
                cursor.execute("SELECT * FROM opthalmic_details")

            data = cursor.fetchall()
            columns = [desc[0] for desc in cursor.description]

            if not data:
                QMessageBox.information(self, "No Data",
                                        "No opthalmic records found for
this student.")
                return

            # === Build Record Blocks ===
            for row_idx, row_data in enumerate(data):
                record_frame = QFrame()
                record_frame.setStyleSheet("""
                    QFrame {
                        background-color: rgba(255,255,255,0.9);
                        border-radius: 10px;
                        border: 1px solid gray;
                        padding: 12px;
                        margin-bottom: 15px;
```

```
                    }
                """)
                record_layout = QVBoxLayout(record_frame)

                for col_name, value in zip(columns, row_data):
                    row_layout = QHBoxLayout()

                    # Column name
                    label_key = QLabel(f"{col_name.replace('_', '
').upper()}:")
                    label_key.setFont(QFont("Arial", 11, QFont.Bold))
                    label_key.setFixedWidth(200)

                    # Column value
                    label_value = QLabel(str(value))
                    label_value.setFont(QFont("Arial", 11))
                    label_value.setStyleSheet("color: #222;")

                    row_layout.addWidget(label_key)
                    row_layout.addWidget(label_value)
                    row_layout.addStretch()
                    record_layout.addLayout(row_layout)

                # Add each record block to scroll layout
                self.scroll_layout.addWidget(record_frame)



        except Error as e:
            QMessageBox.critical(self, "Database Error", f"Error while
fetching data:\n{e}")

        finally:
            if 'conn' in locals() and conn.is_connected():
                cursor.close()
                conn.close()

    # ---------- Go to Main Menu ----------
    def go_back(self):
        if self.parent_window:
            self.parent_window.show()  # ✅ show previous window again
        self.close()


if __name__ == "__main__":
    app = QApplication(sys.argv)

    # Example usage: pass a name to search
    search_name = None  # set to a string like "John Doe" to filter
    window = OpthalmicHistory(search_name)

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

VACCINATION HISTORY WINDOW:

```
import sys
import os
import subprocess
import mysql.connector
from mysql.connector import Error
from PyQt5.QtWidgets import (
    QApplication, QWidget, QVBoxLayout, QPushButton,
    QScrollArea, QFrame, QMessageBox, QLabel, QHBoxLayout, QHeaderView
)
from PyQt5.QtGui import QFont, QPalette, QBrush, QPixmap
from PyQt5.QtCore import Qt


class VaccinationHistory(QWidget):
    def __init__(self, search_name=None, parent=None):
        super().__init__()
        self.parent_window = parent  # ✓ store reference
        self.search_name = search_name
        self.setWindowTitle("PharmaLogix - Vaccination History")
        self.setFixedSize(800, 500)
        self.bg_path = os.path.join(os.path.dirname(__file__), "Bg
(1).jpg")
        self.search_name = search_name  # store search term

        self.init_ui()
        self.load_data_from_db()

    def init_ui(self):
        self.setAutoFillBackground(True)
        self.update_background()

        layout = QVBoxLayout(self)

        # Top layout with title and back button
        top_layout = QHBoxLayout()
        title = QLabel("VACCINATION HISTORY")
        title.setFont(QFont("Arial", 18, QFont.Bold))
        title.setAlignment(Qt.AlignCenter)
        top_layout.addWidget(title)
        top_layout.addStretch()

        back_btn = QPushButton("BACK")
        back_btn.setStyleSheet("""
            QPushButton {
                background: white;
                border-radius: 10px;
                padding: 6px 12px;
```

```
                font-weight: bold;
            }
            QPushButton:hover {
                background: lightgray;
            }
        """)
        back_btn.clicked.connect(self.go_back)
        top_layout.addWidget(back_btn)
        layout.addLayout(top_layout)

        # === Scroll area for data ===
        scroll_area = QScrollArea()
        scroll_area.setWidgetResizable(True)
        scroll_content = QWidget()
        self.scroll_layout = QVBoxLayout(scroll_content)
        scroll_area.setWidget(scroll_content)
        layout.addWidget(scroll_area)


    def update_background(self):
        if os.path.exists(self.bg_path):
            palette = QPalette()
            pixmap = QPixmap(self.bg_path).scaled(
                self.size(), Qt.KeepAspectRatioByExpanding,
Qt.SmoothTransformation
            )
            palette.setBrush(QPalette.Window, QBrush(pixmap))
            self.setPalette(palette)

    def resizeEvent(self, event):
        self.update_background()
        super().resizeEvent(event)

    # --------- MySQL Data Fetch ----------
    def load_data_from_db(self):
        try:
            conn = mysql.connector.connect(
                host="localhost",
                user="root",        # update if needed
                password="deens",        # update if you have a MySQL
password
                database="pharmalogix"
            )
            cursor = conn.cursor()

            if self.search_name:  # search by student name
                query = "SELECT * FROM vaccination_details WHERE
student_name = %s"
                cursor.execute(query, (self.search_name,))
            else:  # fetch all records
                cursor.execute("SELECT * FROM vaccination_details")

            data = cursor.fetchall()
            columns = [desc[0] for desc in cursor.description]
```

```
            if not data:
                QMessageBox.information(self, "No Data",
                                        "No vaccination records found for
this student.")
                return

            # === Build Record Blocks ===
            for row_idx, row_data in enumerate(data):
                record_frame = QFrame()
                record_frame.setStyleSheet("""
                    QFrame {
                        background-color: rgba(255,255,255,0.9);
                        border-radius: 10px;
                        border: 1px solid gray;
                        padding: 12px;
                        margin-bottom: 15px;
                    }
                """)
                record_layout = QVBoxLayout(record_frame)

                for col_name, value in zip(columns, row_data):
                    row_layout = QHBoxLayout()

                    # Column name
                    label_key = QLabel(f"{col_name.replace('_', '
').upper()}:")
                    label_key.setFont(QFont("Arial", 11, QFont.Bold))
                    label_key.setFixedWidth(200)

                    # Column value
                    label_value = QLabel(str(value))
                    label_value.setFont(QFont("Arial", 11))
                    label_value.setStyleSheet("color: #222;")

                    row_layout.addWidget(label_key)
                    row_layout.addWidget(label_value)
                    row_layout.addStretch()
                    record_layout.addLayout(row_layout)

                # Add each record block to scroll layout
                self.scroll_layout.addWidget(record_frame)



        except Error as e:
            QMessageBox.critical(self, "Database Error", f"Error while
fetching data:\n{e}")

        finally:
            if 'conn' in locals() and conn.is_connected():
                cursor.close()
                conn.close()
```

```
    # ---------- Go to Main Menu ----------
    def go_back(self):
        if self.parent_window:
            self.parent_window.show()  # ✅ show previous window again
        self.close()

if __name__ == "__main__":
    app = QApplication(sys.argv)

    # Example usage: pass a name to search
    search_name = None  # set to a string like "John Doe" to filter
    window = VaccinationHistory(search_name)

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

STUDENT REGISTRATION WINDOW:

```
import os
import subprocess
import mysql.connector
from mysql.connector import Error
import sys
from PyQt5.QtWidgets import (
    QApplication, QWidget, QLabel, QLineEdit, QPushButton,
    QVBoxLayout, QGridLayout, QDateEdit, QMessageBox,
    QScrollArea, QHBoxLayout
)
from PyQt5.QtGui import QFont, QPalette, QBrush, QPixmap
from PyQt5.QtCore import Qt, QDate

class student_registration(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("PharmaLogix")
        self.setFixedSize(800, 500)  # Window size

        self.bg_path = os.path.join(os.path.dirname(__file__), "Bg
(1).jpg")

        # Form fields
        self.student_details = [

            "REGISTRATION DATE", "DATE OF BIRTH", "MOTHER NAME",
            "FATHER NAME", "PERMANENT ADDRESS", "PARENT CONTACT NUMBER",
            "STUDENT NAME", "SEX", "BLOOD GROUP", "EMERGENCY CONTACT
PERSON", "EMERGENCY CONTACT NUMBER"
        ]
```

```
        self.inputs = {}
        self.init_ui()

        # Initialize DB
        self.create_table_if_not_exists()

    def init_ui(self):
        self.setAutoFillBackground(True)
        # self.update_background()

        main_layout = QVBoxLayout(self)

        # Top layout for title and back button
        top_layout = QHBoxLayout()
        title_label = QLabel("STUDENT DETAILS")
        title_label.setFont(QFont("Arial", 18, QFont.Bold))
        title_label.setAlignment(Qt.AlignCenter)
        top_layout.addWidget(title_label)
        top_layout.addStretch()

        return_btn = QPushButton("BACK")
        btn_style = """
            QPushButton {
                background: white;
                border-radius: 10px;
                padding: 6px 12px;
                font-weight: bold;
            }
            QPushButton:hover {
                background: lightgray;
            }
        """
        return_btn.setStyleSheet(btn_style)
        return_btn.clicked.connect(self.go_to_main_menu)
        top_layout.addWidget(return_btn)

        main_layout.addLayout(top_layout)

        # Scrollable form
        scroll_area = QScrollArea()
        scroll_area.setWidgetResizable(True)
        scroll_area.setStyleSheet("background: transparent; border:
none;")
        inner_widget = QWidget()
        inner_layout = QVBoxLayout(inner_widget)
        grid_layout = QGridLayout()
        grid_layout.setVerticalSpacing(15)
        grid_layout.setHorizontalSpacing(20)

        field_style = """
            QLineEdit, QDateEdit {
                background: white;
                border-radius: 10px;
                padding: 5px;
```

```
                font-size: 12px;
            }
            QLabel {
                background: rgba(255,255,255,0.8);
                border-radius: 10px;
                padding: 5px;
            }
        """

        for row, detail in enumerate(self.student_details):
            label = QLabel(detail)
            label.setFont(QFont("Arial", 11, QFont.Bold))
            label.setStyleSheet(field_style)
            if detail == "REGISTRATION DATE" or detail == "DATE OF BIRTH":
                input_field = QDateEdit()
                input_field.setCalendarPopup(True)
                input_field.setDate(QDate.currentDate())
            else:
                input_field = QLineEdit()
            input_field.setStyleSheet(field_style)
            grid_layout.addWidget(label, row, 0)
            grid_layout.addWidget(input_field, row, 1)
            self.inputs[detail] = input_field

        inner_layout.addLayout(grid_layout)
        scroll_area.setWidget(inner_widget)
        main_layout.addWidget(scroll_area)

        # Submit button at bottom
        submit_btn = QPushButton("SUBMIT")
        submit_btn.setStyleSheet(btn_style)
        submit_btn.clicked.connect(self.submit_form)
        bottom_layout = QHBoxLayout()
        bottom_layout.addStretch()
        bottom_layout.addWidget(submit_btn)
        bottom_layout.addStretch()
        main_layout.addLayout(bottom_layout)

    def update_background(self):
        if os.path.exists(self.bg_path):
            palette = QPalette()
            pixmap = QPixmap(self.bg_path).scaled(
                self.size(), Qt.KeepAspectRatioByExpanding,
Qt.SmoothTransformation
            )
            palette.setBrush(QPalette.Window, QBrush(pixmap))
            self.setPalette(palette)

    def resizeEvent(self, event):
        self.update_background()
        super().resizeEvent(event)

    def go_to_main_menu(self):
        base_dir = os.path.dirname(os.path.abspath(__file__))
```

```
        menu_path = os.path.join(base_dir, "main_menu.py")
        subprocess.Popen([sys.executable, menu_path])
        self.close()

    def submit_form(self):
        data = {}
        for key, widget in self.inputs.items():
            if isinstance(widget, QDateEdit):
                data[key] = widget.date().toString("yyyy-MM-dd")
            else:
                data[key] = widget.text().strip()

        # Basic validation


        conn = self.get_connection()
        if not conn:
            return
        try:
            cursor = conn.cursor()
            columns = ', '.join([f"`{col.replace(' ', '_')}`" for col in
data.keys()])
            placeholders = ', '.join(['%s'] * len(data))
            query = f"INSERT INTO student_details ({columns}) VALUES
({placeholders})"
            cursor.execute(query, list(data.values()))
            conn.commit()
            QMessageBox.information(self, "Success", "Record saved
successfully!")
            # clear fields
            for widget in self.inputs.values():
                if isinstance(widget, QLineEdit):
                    widget.clear()
                elif isinstance(widget, QDateEdit):
                    widget.setDate(QDate.currentDate())
        except Error as e:
            QMessageBox.critical(self, "Database Error", f"Failed to
insert data:\n{e}")
        finally:
            cursor.close()
            conn.close()

    def get_connection(self):
            try:
                conn = mysql.connector.connect(
                    host="localhost",
                    user="root",
                    password="deens",
                    database="pharmalogix"
                )
                print("Database connected")
                return conn
            except Error as e:
                print("Database error")
```

```
                return None

    def create_table_if_not_exists(self):
            conn = self.get_connection()
            if not conn:
                return
            cursor = conn.cursor()
            try:
                cursor.execute("""
                    CREATE TABLE IF NOT EXISTS student_details (
            REGISTRATION_DATE date,
            DATE_OF_BIRTH date,
            MOTHER_NAME varchar(255),
            FATHER_NAME varchar(255),
            PERMANENT_ADDRESS varchar(400),
            PARENT_CONTACT_NUMBER varchar(100),
            STUDENT_NAME varchar(255),
            SEX varchar(5),
            BLOOD_GROUP varchar(5),
            EMERGENCY_CONTACT_PERSON varchar(255),
            EMERGENCY_CONTACT_NUMBER varchar(100))


                """)
                conn.commit()
                print("Table created or already exists")
            except Error as e:
                print("Database error")
                # QMessageBox.critical(self, "Database Error", f"Table
creation failed:\n{e}")
            finally:
                cursor.close()
                conn.close()
if __name__ == "__main__":
    print("hello")
    app = QApplication(sys.argv)
    form = student_registration()
    form.show()

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

REPORT GENERATION WINDOW:

```
import sys
import os
import mysql.connector
from mysql.connector import Error
from PyQt5.QtWidgets import (
    QApplication, QWidget, QVBoxLayout, QLabel, QComboBox, QPushButton,
    QMessageBox, QHBoxLayout
)
from PyQt5.QtGui import QFont, QPalette, QBrush, QPixmap
from PyQt5.QtCore import Qt
```

```
from reportlab.lib.pagesizes import A4
from reportlab.platypus import SimpleDocTemplate, Paragraph, Spacer,
Table, TableStyle
from reportlab.lib import colors
from reportlab.lib.styles import getSampleStyleSheet


class ReportPage(QWidget):
    def __init__(self, parent=None):
        super().__init__()
        self.parent_window = parent
        self.setWindowTitle("Generate Student Health Report")
        self.setFixedSize(500, 100)
        self.bg_path = os.path.join(os.path.dirname(__file__), "Bg
(1).jpg")
        self.set_background()
        self.init_ui()
        self.load_students()
    def set_background(self):
        """Sets the background image of the window."""
        palette = QPalette()
        pixmap = QPixmap(self.bg_path).scaled(
            self.size(), Qt.KeepAspectRatioByExpanding,
Qt.SmoothTransformation
        )
        palette.setBrush(QPalette.Window, QBrush(pixmap))
        self.setPalette(palette)
    def init_ui(self):
        layout = QVBoxLayout()

        title = QLabel("Generate Student Health Report")
        title.setFont(QFont("Arial", 16, QFont.Bold))
        title.setAlignment(Qt.AlignCenter)

        self.student_dropdown = QComboBox()
        self.student_dropdown.setPlaceholderText("Select Student")

        generate_btn = QPushButton("Generate Report")
        generate_btn.setStyleSheet("background-color: #4CAF50; color:
white; font-weight: bold;")
        generate_btn.clicked.connect(self.generate_report)

        back_btn = QPushButton("Back")
        back_btn.clicked.connect(self.go_back)

        btn_layout = QHBoxLayout()
        btn_layout.addWidget(generate_btn)
        btn_layout.addWidget(back_btn)

        layout.addWidget(title)
        layout.addWidget(self.student_dropdown)
        layout.addLayout(btn_layout)

        self.setLayout(layout)
```

```
    def go_back(self):
        if self.parent_window:
            self.parent_window.show()
        self.close()

    def connect_db(self):
        try:
            return mysql.connector.connect(
                host="localhost",
                user="root",
                password="deens",
                database="pharmalogix"
            )
        except Error as e:
            QMessageBox.critical(self, "Database Error", str(e))
            return None

    def load_students(self):
        conn = self.connect_db()
        if not conn:
            return
        cursor = conn.cursor()
        cursor.execute("SELECT DISTINCT student_name FROM
physical_details")
        names = [row[0] for row in cursor.fetchall()]
        self.student_dropdown.addItems(names)
        conn.close()

    def generate_report(self):
        student_name = self.student_dropdown.currentText()
        if not student_name:
            QMessageBox.warning(self, "No Student", "Please select a
student.")
            return

        conn = self.connect_db()
        if not conn:
            return

        cursor = conn.cursor(dictionary=True)
        report_data = {}

        # ✅ Fetch from different tables
        for table in ["physical_details", "pediatric_details",
"general_checkup_details", "dental_details", "opthalmic_details",
"vaccination_details"]:
            cursor.execute(f"SELECT * FROM {table} WHERE STUDENT_NAME =
%s", (student_name,))
            result = cursor.fetchone()
            report_data[table] = result if result else {}

        conn.close()
```

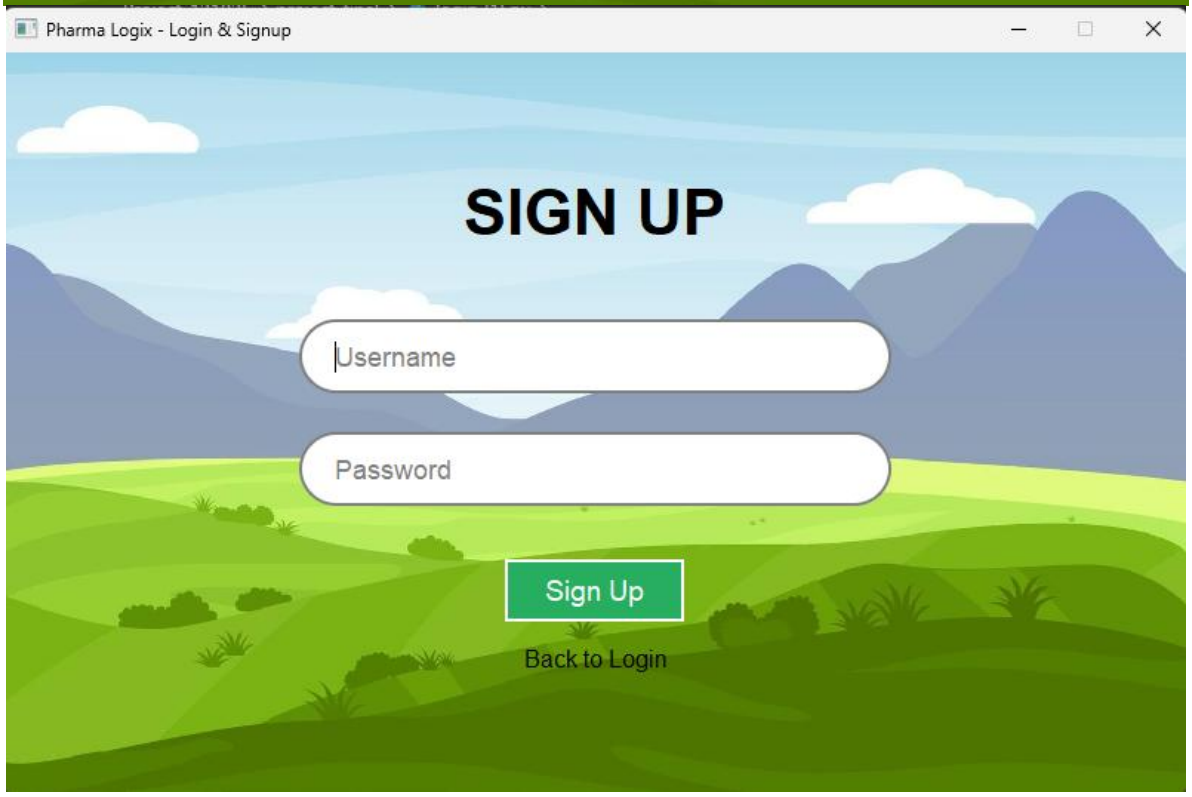
```
        # ✅ Create PDF
        file_name = f"{student_name.replace(' ', '_')}_Health_Report.pdf"
        pdf = SimpleDocTemplate(file_name, pagesize=A4)
        styles = getSampleStyleSheet()
        elements = []

        elements.append(Paragraph(f"<b>STUDENT HEALTH RECORD</b>",
styles['Title']))
        elements.append(Spacer(1, 12))
        elements.append(Paragraph(f"<b>STUDENT NAME:</b> {student_name}",
styles['Normal']))
        elements.append(Spacer(1, 12))

        # ✅ Physical Checkup
        physical = report_data.get("physical_details", {})
        if physical:
            elements.append(Paragraph("<b>Physical Checkup</b>",
styles['Heading2']))
            data = [[k.replace('_', ' ').title(), str(v)] for k, v in
physical.items()]
            table = Table(data, colWidths=[200, 200])
            table.setStyle(TableStyle([
                ('BACKGROUND', (0, 0), (-1, 0), colors.grey),
                ('GRID', (0, 0), (-1, -1), 1, colors.black)
            ]))
            elements.append(table)
            elements.append(Spacer(1, 12))

        # ✅ Pediatric Details
        pediatric = report_data.get("pediatric_details", {})
        if pediatric:
            elements.append(Paragraph("<b>Pediatric Checkup</b>",
styles['Heading2']))
            data = [[k.replace('_', ' ').title(), str(v)] for k, v in
pediatric.items()]
            table = Table(data, colWidths=[200, 200])
            table.setStyle(TableStyle([
                ('BACKGROUND', (0, 0), (-1, 0), colors.lightgrey),
                ('GRID', (0, 0), (-1, -1), 1, colors.black)
            ]))
            elements.append(table)
            elements.append(Spacer(1, 12))

        # ✅ General Checkup Details
        general = report_data.get("general_checkup_details", {})

        if general:
            elements.append(Paragraph("<b>General Checkup</b>",
styles['Heading2']))
            data = [[k.replace('_', ' ').title(), str(v)] for k, v in
general.items()]
            table = Table(data, colWidths=[200, 200])
            table.setStyle(TableStyle([
```

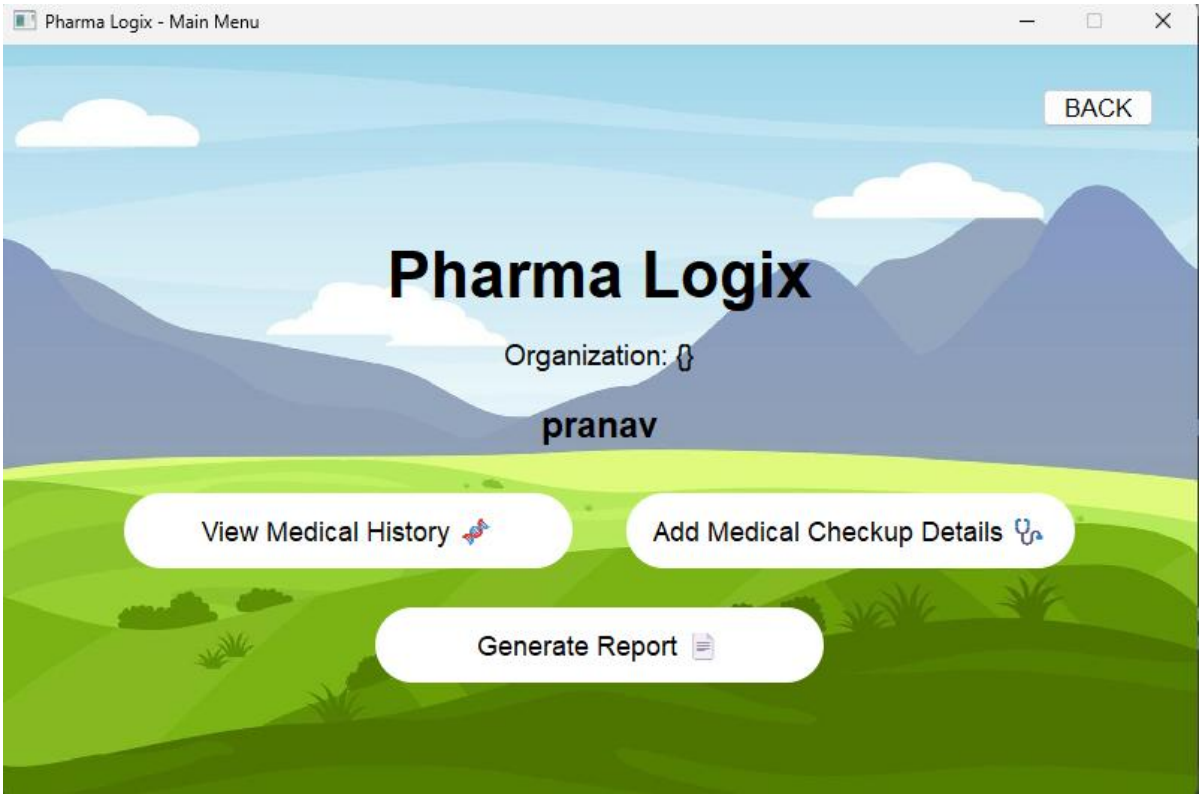
```
                ('BACKGROUND', (0, 0), (-1, 0), colors.lightgrey),
                ('GRID', (0, 0), (-1, -1), 1, colors.black)
            ]))
            elements.append(table)
            elements.append(Spacer(1, 12))
        # ✅ Dental Details
        dental = report_data.get("dental_details", {})
        if dental:
            elements.append(Paragraph("<b>Dental Checkup</b>",
styles['Heading2']))
            data = [[k.replace('_', ' ').title(), str(v)] for k, v in
dental.items()]
            table = Table(data, colWidths=[200, 200])
            table.setStyle(TableStyle([
                ('BACKGROUND', (0, 0), (-1, 0), colors.lightgrey),
                ('GRID', (0, 0), (-1, -1), 1, colors.black)
            ]))
            elements.append(table)
            elements.append(Spacer(1, 12))
        # ✅ Ophthalmic Details
        opthalmic = report_data.get("opthalmic_details", {})

        if opthalmic:
            elements.append(Paragraph("<b>Opthalmic Checkup</b>",
styles['Heading2']))
            data = [[k.replace('_', ' ').title(), str(v)] for k, v in
opthalmic.items()]
            table = Table(data, colWidths=[200, 200])
            table.setStyle(TableStyle([
                ('BACKGROUND', (0, 0), (-1, 0), colors.lightgrey),
                ('GRID', (0, 0), (-1, -1), 1, colors.black)
            ]))
            elements.append(table)
            elements.append(Spacer(1, 12))
        # ✅ Vaccination Details
        vacc = report_data.get("vaccination_details", {})
        if vacc:
            elements.append(Paragraph("<b>Vaccination Details</b>",
styles['Heading2']))
            data = [[k.replace('_', ' ').title(), str(v)] for k, v in
vacc.items()]
            table = Table(data, colWidths=[200, 200])
            table.setStyle(TableStyle([
                ('BACKGROUND', (0, 0), (-1, 0), colors.whitesmoke),
                ('GRID', (0, 0), (-1, -1), 1, colors.black)
            ]))
            elements.append(table)
            elements.append(Spacer(1, 12))


        pdf.build(elements)
        QMessageBox.information(self, "Success", f"Report
generated:\n{file_name}")
```

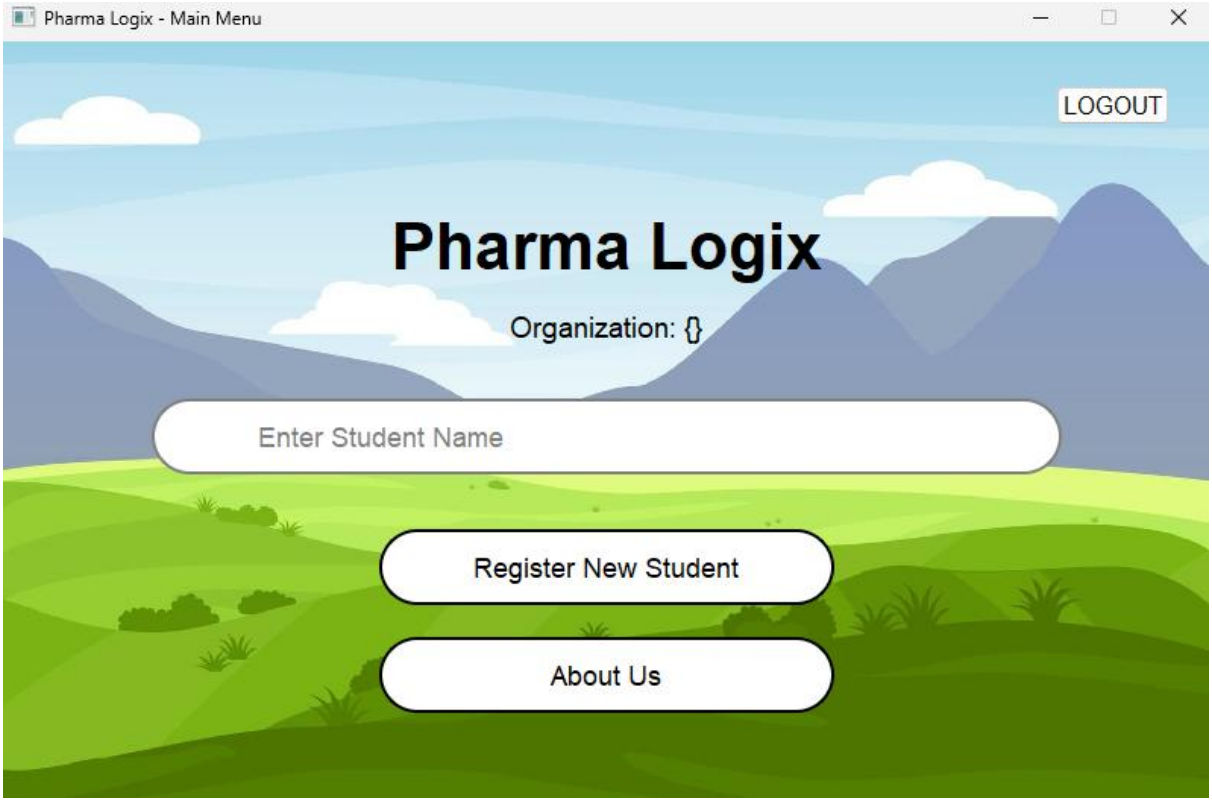
```
if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = ReportPage()
    window.show()
    sys.exit(app.exec_())
```

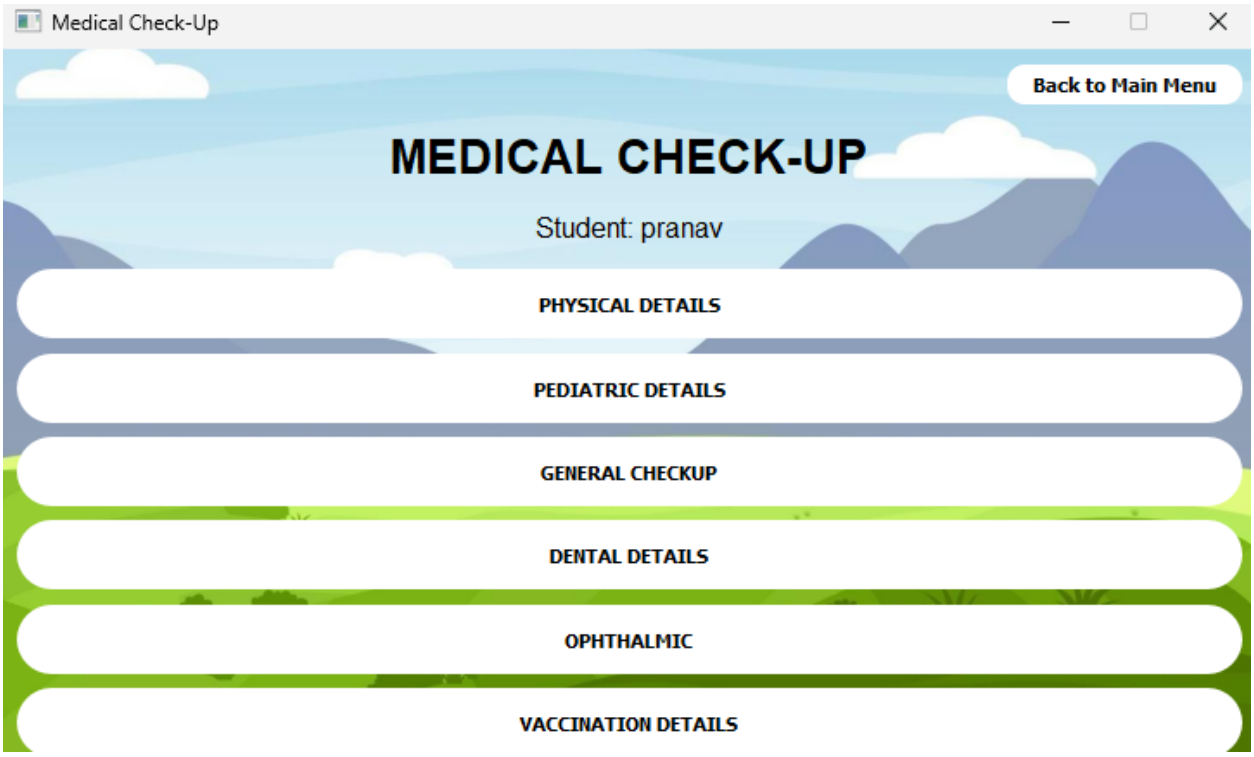
OUTPUT


Pharma Logix - Login & Signup
Sign Up
Login
Username
Password
Forgot Password?
Login

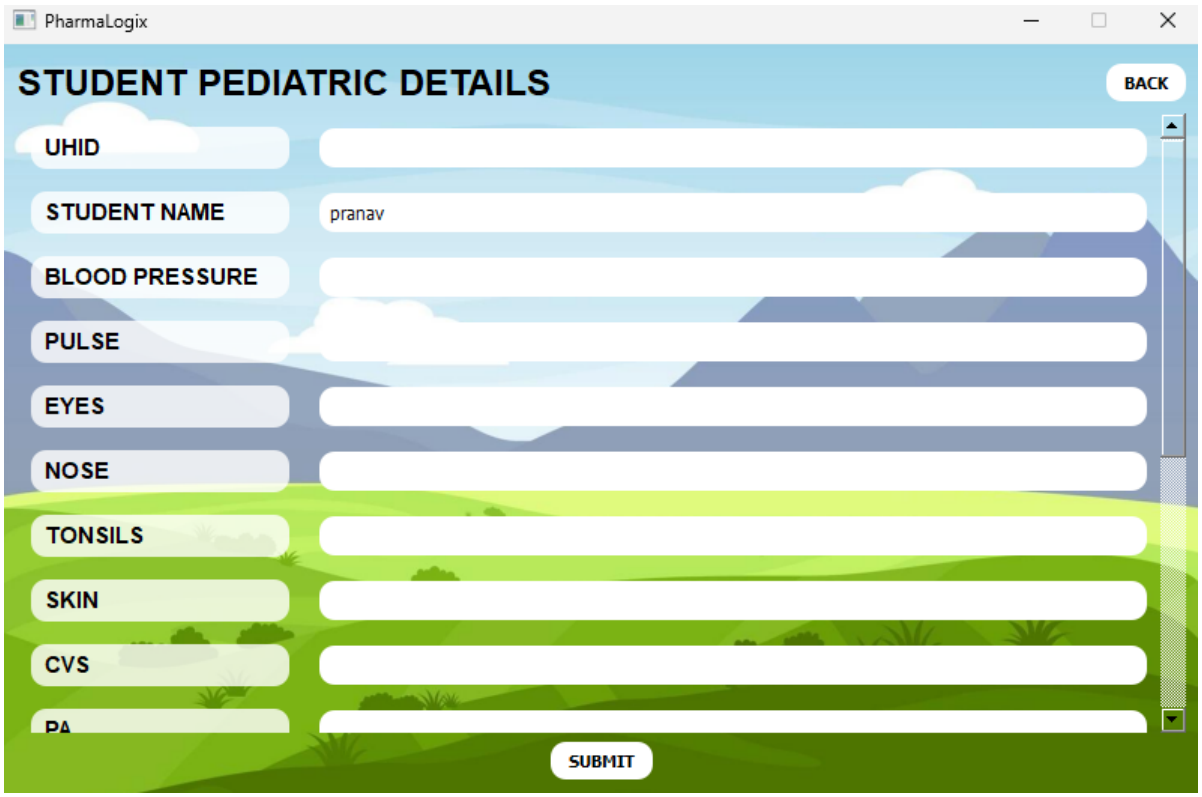

Pharma Logix - Login & Signup
SIGN UP
Username
Password
Sign Up
Back to Login

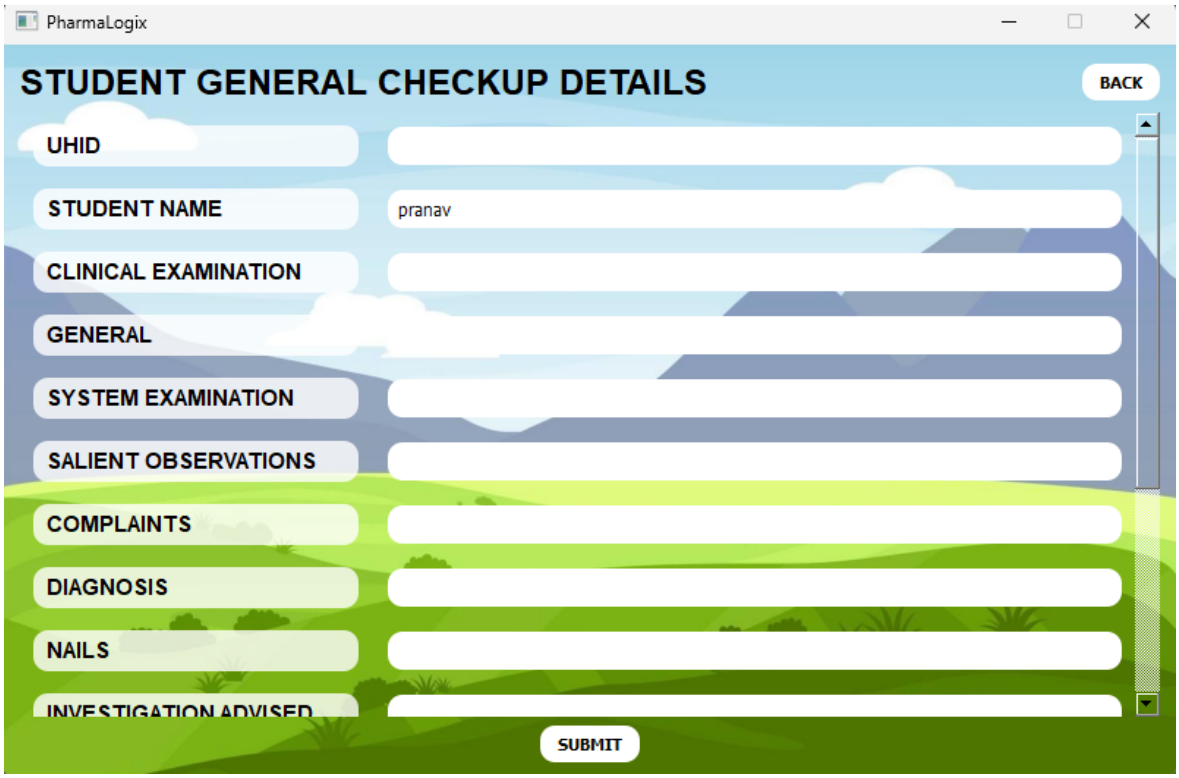

Pharma Logix - Main Menu
BACK
Pharma Logix
Organization: {}
pranav
View Medical History
Add Medical Checkup Details
Generate Report

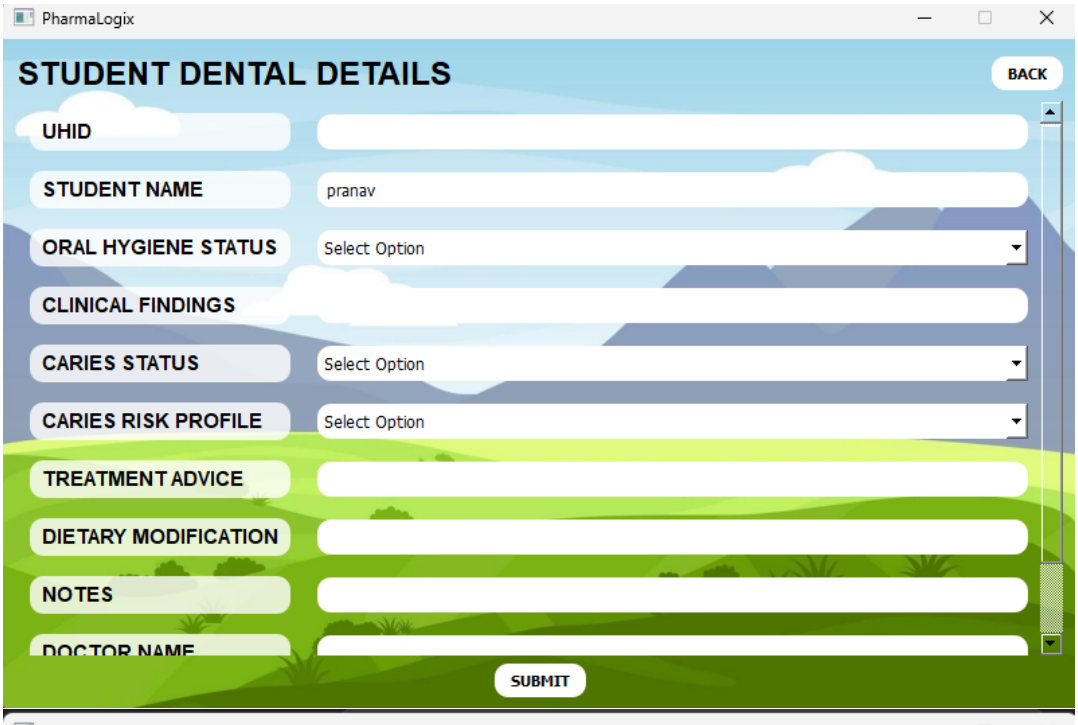

Pharma Logix - Main Menu
LOGOUT
Pharma Logix
Organization: {}
Enter Student Name
Register New Student
About Us

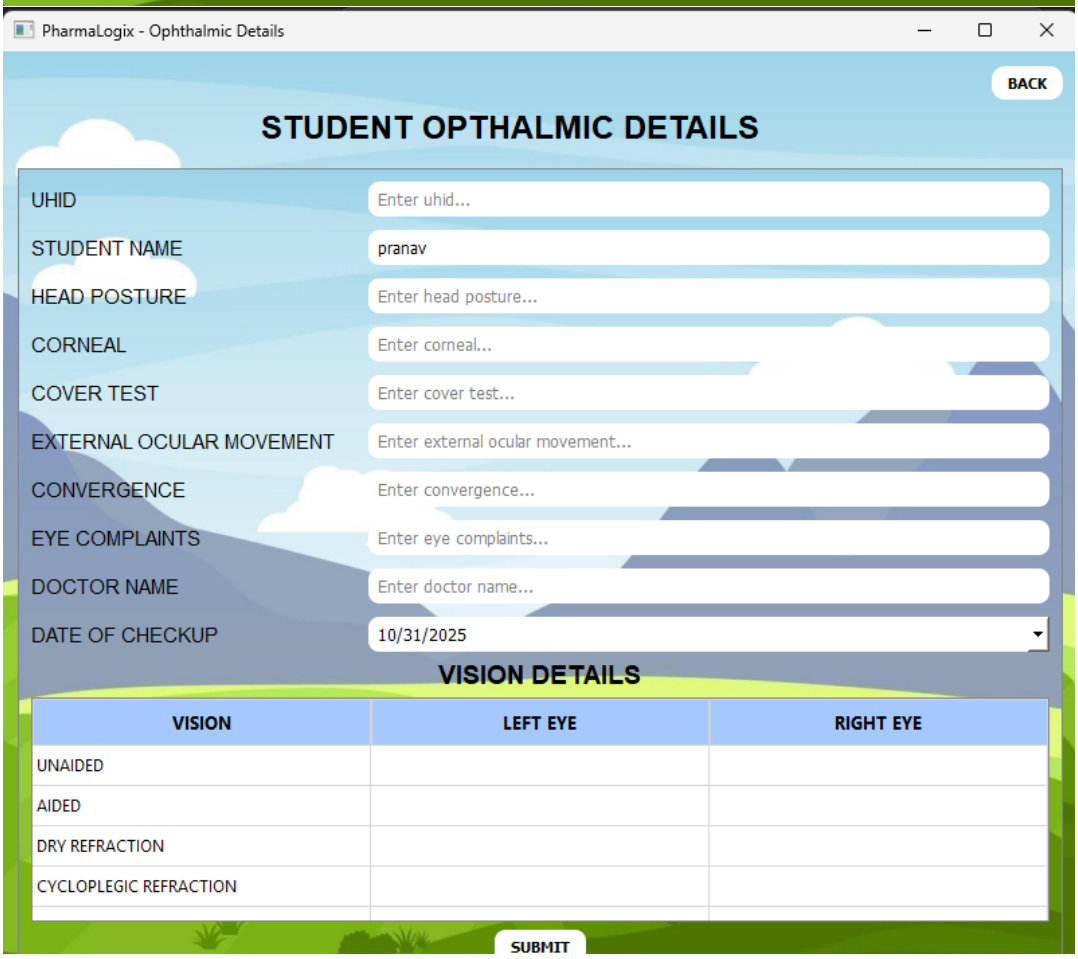

Medical Check-Up
Back to Main Menu
MEDICAL CHECK-UP
Student: pranav
PHYSICAL DETAILS
PEDIATRIC DETAILS
GENERAL CHECKUP
DENTAL DETAILS
OPHTHALMIC
VACCINATION DETAILS

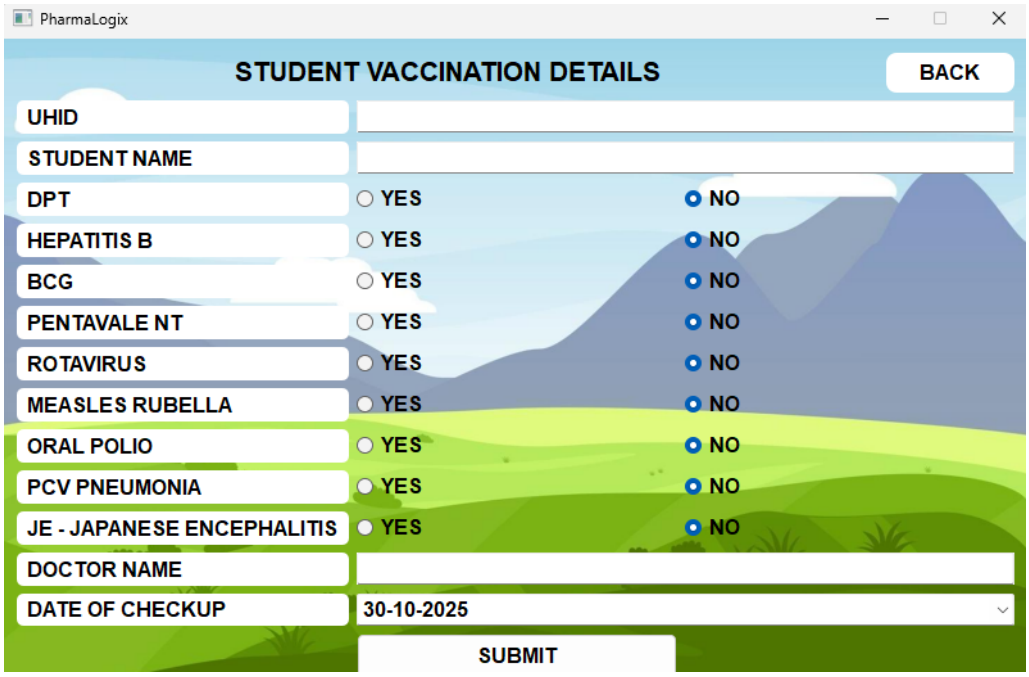

PharmaLogix
STUDENT PHYSICAL DETAILS
BACK
UHID
STUDENT NAME
pranav
HEIGHT
WEIGHT
DATE OF CHECKUP
10/30/2025
SUBMIT

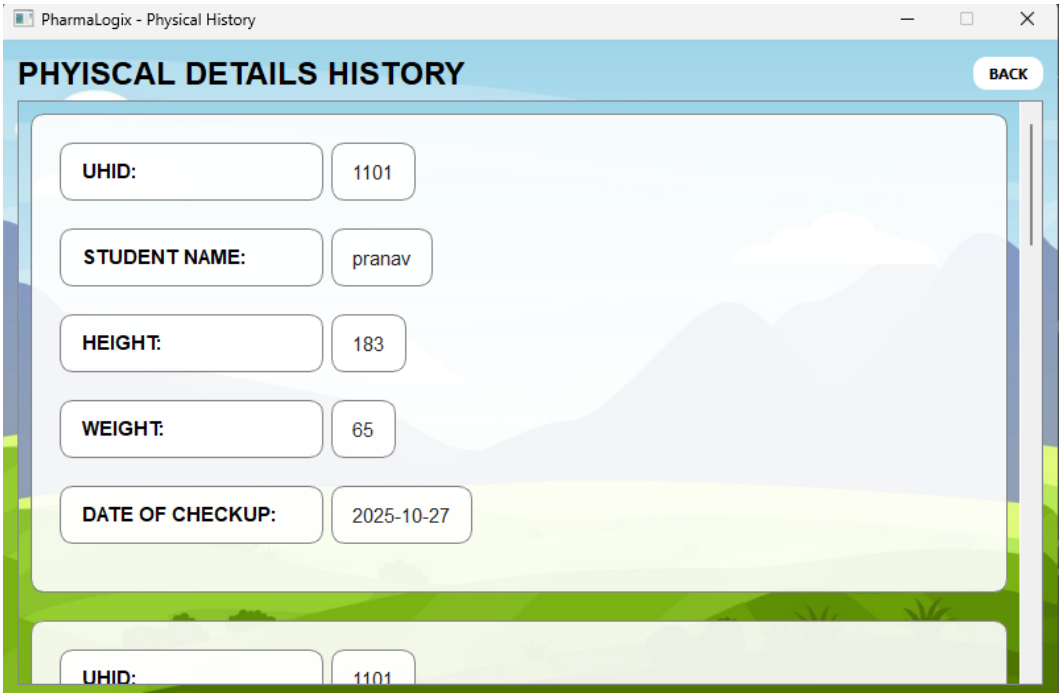

PharmaLogix
STUDENT PEDIATRIC DETAILS
BACK
UHID
STUDENT NAME
pranav
BLOOD PRESSURE
PULSE
EYES
NOSE
TONSILS
SKIN
CVS
SUBMIT

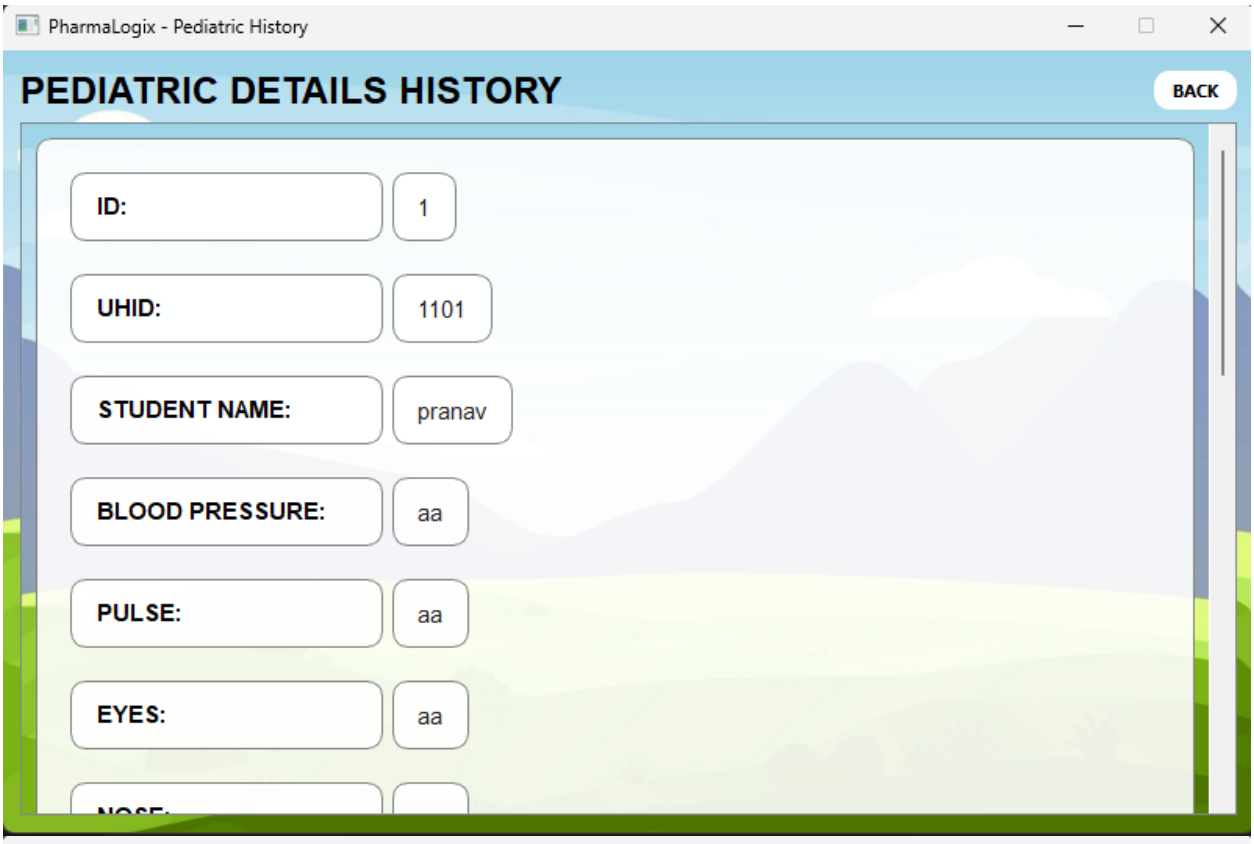

PharmaLogix
STUDENT GENERAL CHECKUP DETAILS
BACK
UHID
STUDENT NAME
pranav
CLINICAL EXAMINATION
GENERAL
SYSTEM EXAMINATION
SALIENT OBSERVATIONS
COMPLAINTS
DIAGNOSIS
NAILS
SUBMIT

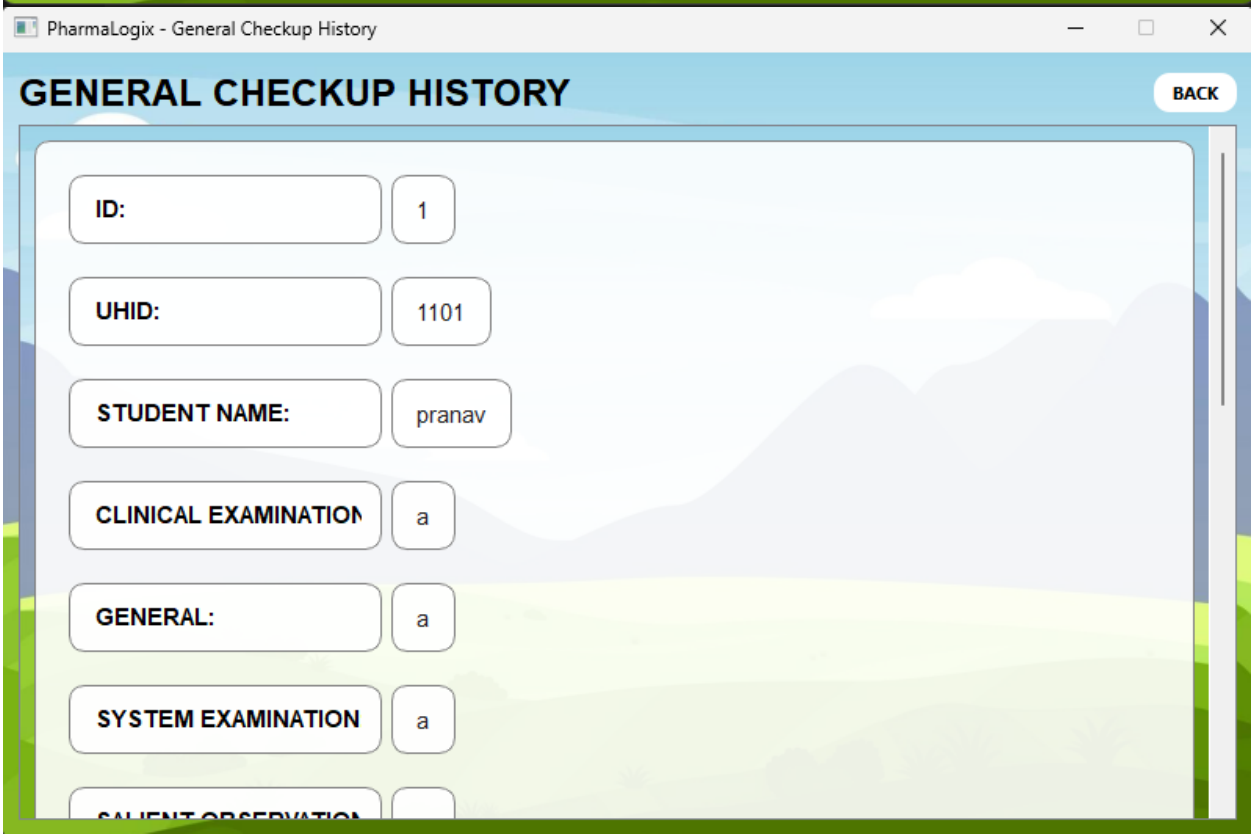

PharmaLogix
STUDENT DENTAL DETAILS
BACK
UHID
STUDENT NAME
pranav
ORAL HYGIENE STATUS
Select Option
CLINICAL FINDINGS
CARIES STATUS
Select Option
CARIES RISK PROFILE
Select Option
TREATMENT ADVICE
DIETARY MODIFICATION
NOTES
DOCTOR NAME
SUBMIT

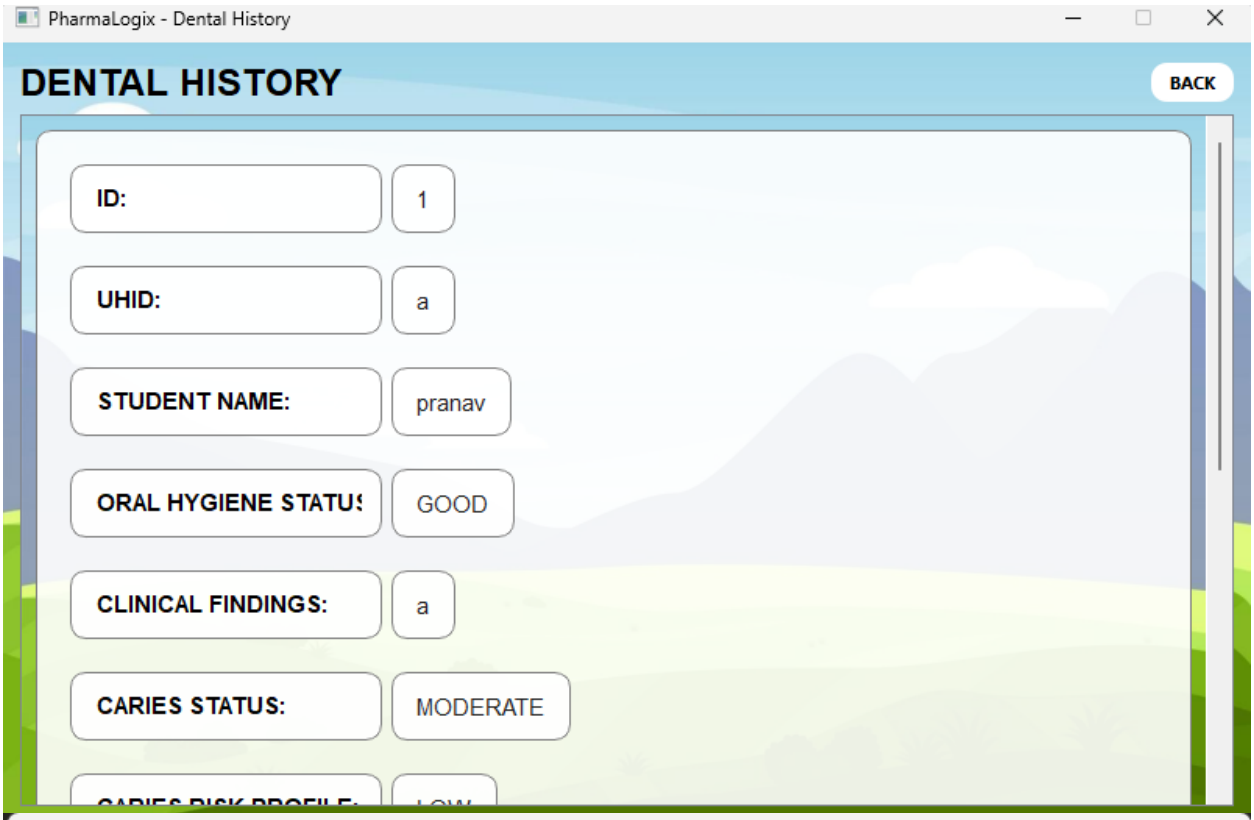

PharmaLogix - Ophthalmic Details
BACK
STUDENT OPTHALMIC DETAILS
UHID
Enter uhid...
STUDENT NAME
pranav
HEAD POSTURE
Enter head posture...
CORNEAL
Enter corneal...
COVER TEST
Enter cover test...
EXTERNAL OCULAR MOVEMENT
Enter external ocular movement...
CONVERGENCE
Enter convergence...
EYE COMPLAINTS
Enter eye complaints...
DOCTOR NAME
Enter doctor name...
DATE OF CHECKUP
10/31/2025
VISION DETAILS
VISION
LEFT EYE
RIGHT EYE
UNAIDED
AIDED
DRY REFRACTION
CYCLOPLEGIC REFRACTION
SUBMIT

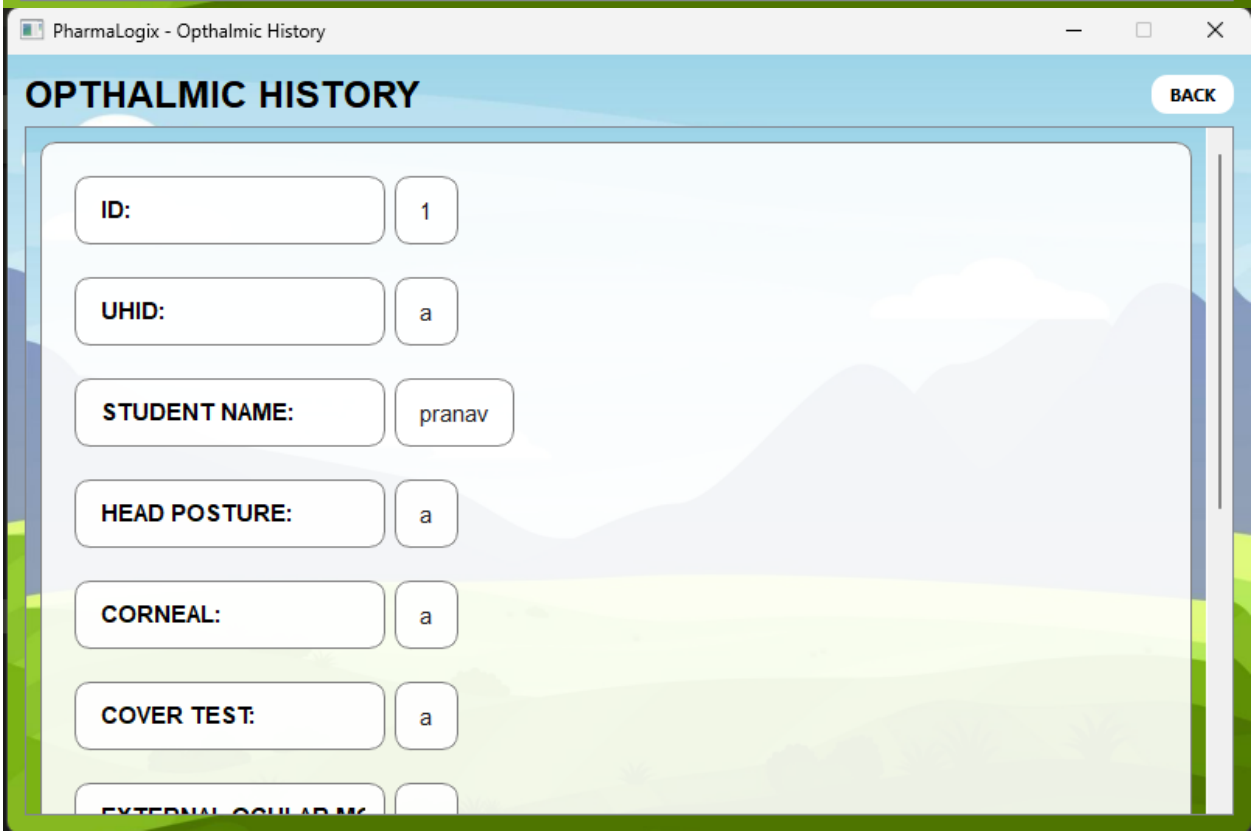

PharmaLogix
STUDENT VACCINATION DETAILS
BACK
UHID
STUDENT NAME
DPT
HEPATITIS B
BCG
PENTAVALE NT
ROTAVIRUS
MEASLES RUBELLA
ORAL POLIO
PCV PNEUMONIA
JE - JAPANESE ENCEPHALITIS
YES
NO
DOCTOR NAME
DATE OF CHECKUP
30-10-2025
SUBMIT


PharmaLogix - Physical History
PHYISCAL DETAILS HISTORY
BACK
UHID:
1101
STUDENT NAME:
pranav
HEIGHT:
183
WEIGHT:
65
DATE OF CHECKUP:
2025-10-27
UHID:
1101

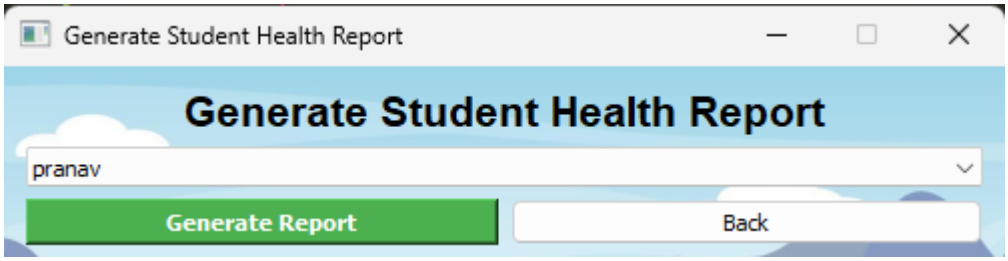

PharmaLogix - Pediatric History
PEDIATRIC DETAILS HISTORY
BACK
ID:
1
UHID:
1101
STUDENT NAME:
pranav
BLOOD PRESSURE:
aa
PULSE:
aa
EYES:
aa

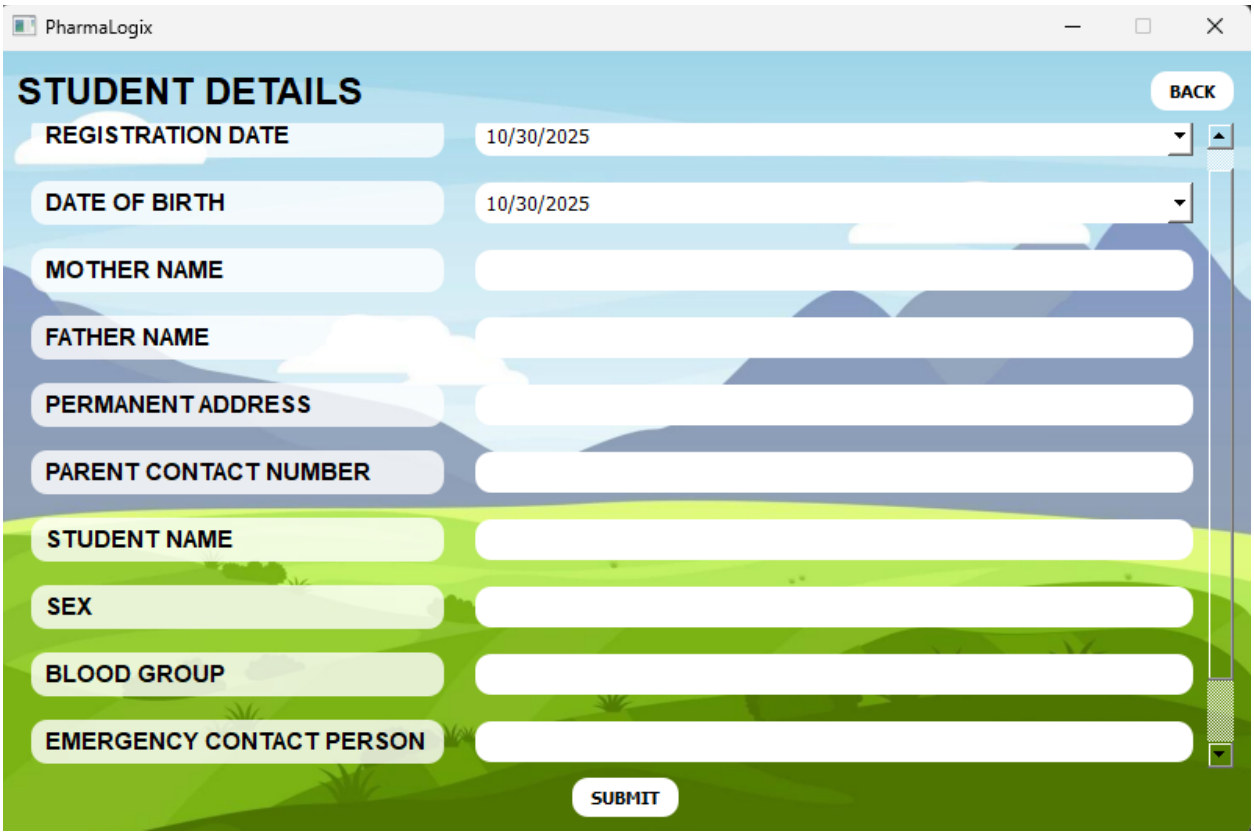

PharmaLogix - General Checkup History
GENERAL CHECKUP HISTORY
BACK
ID:
1
UHID:
1101
STUDENT NAME:
pranav
CLINICAL EXAMINATION
a
GENERAL:
a
SYSTEM EXAMINATION
a


PharmaLogix - Dental History
DENTAL HISTORY
BACK
ID:
1
UHID:
a
STUDENT NAME:
pranav
ORAL HYGIENE STATUS
GOOD
CLINICAL FINDINGS:
a
CARIES STATUS:
MODERATE


PharmaLogix - Opthalmic History
OPTHALMIC HISTORY
BACK
ID:
1
UHID:
a
STUDENT NAME:
pranav
HEAD POSTURE:
a
CORNEAL:
a
COVER TEST:
a


PharmaLogix - Vaccination History
VACCINATION HISTORY
BACK
UHID:
1101
STUDENT NAME:
pranav
DPT:
NO
HEPATITIS B:
YES
BCG:
NO
PENTAVALE NT:
YES


Generate Student Health Report
Generate Student Health Report
pranav
Generate Report
Back


PharmaLogix
STUDENT DETAILS
BACK
REGISTRATION DATE
10/30/2025
DATE OF BIRTH
10/30/2025
MOTHER NAME
FATHER NAME
PERMANENT ADDRESS
PARENT CONTACT NUMBER
STUDENT NAME
SEX
BLOOD GROUP
EMERGENCY CONTACT PERSON
SUBMIT

LIMITATIONS

- The application is relatively primitive.
- Accidental duplication of records may be possible.
- No authentication or role based access, meaning anyone can insert or delete a record.
- Passwords are not encrypted

BIBLIOGRAPHY

- pythonguis.com
- geeksforgeeks.com
- tutorialpoints.com
- educative.io
- learnpyqt.com