Development of an AI-Powered Medical Chatbot with Administrative Dashboard Using Flask

Semester Project Report

Session 2023-2027

BS in Data Science

Department of Software Engineering

Faculty of Computer Science & Information Technology

The Superior College, Lahore

FALL 2023

Submitted by:

Maryam Sharif

Su92-bsdsm-f23-038

BSDS-4A

Submitted to:

Prof, Rasikh Ali

Abstract

This project develops an AI Doctor Chatbot System with Admin Panel that provides personalized medical advice through conversational AI. The system features:

- Patient-facing chatbot interface powered by qwen2.5-vl-32b-instruct AI model

- Secure user authentication with phone verification

- Admin dashboard for monitoring patient interactions

- Medical conversation history tracking

- Flask-based web application with role-based access control

---

1. Introduction

Modern healthcare demands accessible, 24/7 medical consultation services. This AI Doctor Chatbot bridges the gap by offering:

- Instant preliminary medical advice

- Secure patient-doctor communication records

- Administrative oversight capabilities

Key Objectives:

1. Develop an AI-powered medical consultation interface

2. Implement secure user authentication (11-digit phone verification)

3. Create admin functionality for monitoring patient interactions

4. Deploy as a responsive web application using Flask

Applications:

- Telemedicine platforms

- Hospital patient support systems

- Clinic management tools

---

2. Methodology

2.1 System Architecture

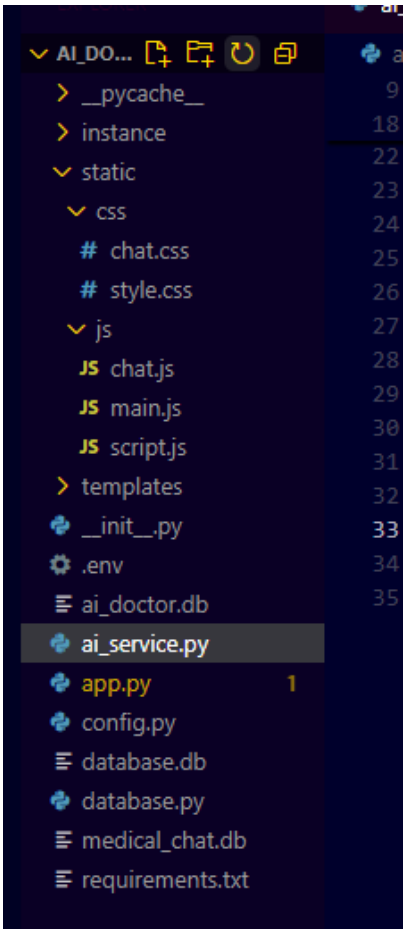
Components:

1. **Frontend:** HTML/CSS/JS with Bootstrap
2. **Backend:** Flask Python framework
3. **Database:** SQLite with Flask-SQLAlchemy
4. **AI Service:** qwen2.5-vl-32b-instruct model via OpenRouter API

2.2 Technology Stack

Component	Technology Used
Frontend	HTML5, CSS3, JavaScript, Bootstrap 5
Backend	Python 3, Flask
Database	SQLite
AI Integration	OpenRouter API
Authentication	Flask-Login, Werkzeug Security
Deployment	Render/Fly.io (or local)

Project Directory



2.3 Key Features Implementation

```python
@app.route('/login', methods=['GET', 'POST'])
def login():
    if current_user.is_authenticated:
        return redirect(url_for('index'))

    if request.method == 'POST':
        name = request.form.get('name', '').strip()
        password = request.form.get('password', '').strip()
        remember = request.form.get('remember') == 'on'

        if not name or not password:
            flash('Name and password are required', 'danger')
            return redirect(url_for('login'))

        user_data = db.verify_user_by_name(name, password)
        if not user_data:
            flash('Invalid name or password', 'danger')
            return redirect(url_for('login'))

        user = User(id=user_data['id'], name=user_data['name'], is_admin=user_data.get('is_admin', False))
        login_user(user, remember=remember)
        db.update_user_last_active(user.id)
        flash('Login successful!', 'success')

        if user.is_admin:
            return redirect(url_for('admin_dashboard'))
        return redirect(url_for('patient_dashboard'))
```

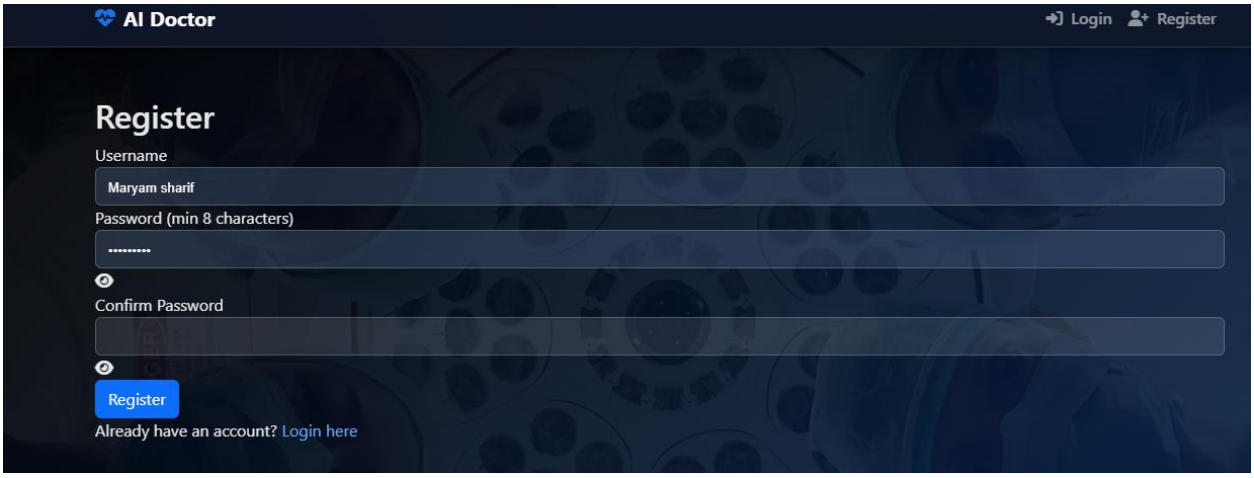
Admin Panel Structure

```python
app.py > ...

@app.route('/admin/dashboard')
@login_required
def admin_dashboard():
    if not current_user.is_admin:
        flash('Unauthorized access', 'danger')
        return redirect(url_for('patient_dashboard'))

    users = db.get_all_users() or []
    return render_template('admin_dashboard.html',
                           users=users,
                           current_time=datetime.now().strftime('%Y-%m-%d %H:%M:%S'))

@app.route('/admin/patients')
@login_required
def admin_patients():
    if not current_user.is_admin:
        flash('Unauthorized access', 'danger')
        return redirect(url_for('patient_dashboard'))

    patients = db.get_all_users() or []
    return render_template('admin_patients.html', patients=patients)

@app.route('/admin/patient/create', methods=['GET', 'POST'])
@login_required
def admin_create_patient():
    if not current_user.is_admin:
        flash('Unauthorized access', 'danger')
```

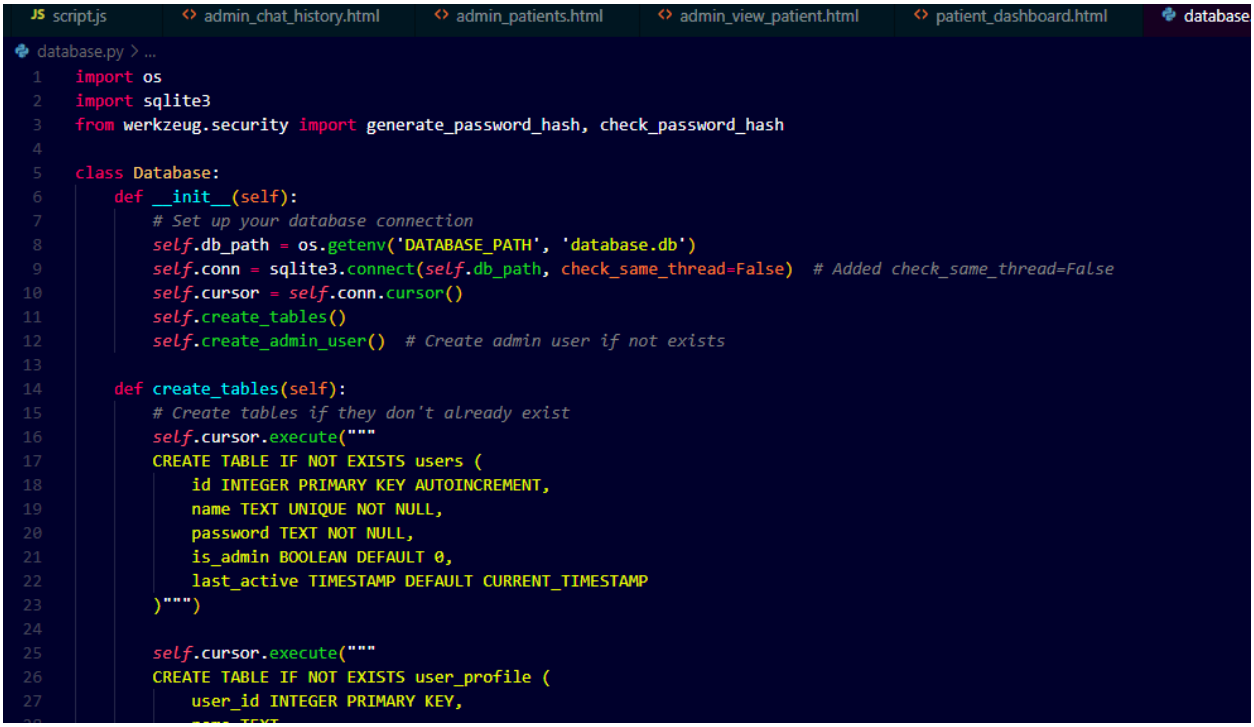
3. Registration

```python
app.py > ...

@app.route('/register', methods=['GET', 'POST'])
def register():
    if current_user.is_authenticated:
        return redirect(url_for('index'))

    if request.method == 'POST':
        name = request.form.get('name', '').strip()
        password = request.form.get('password', '').strip()
        confirm_password = request.form.get('confirm_password', '').strip()

        if not name:
            flash('Name is required', 'danger')
            return redirect(url_for('register'))

        if len(password) < 8:
            flash('Password must be at least 8 characters', 'danger')
            return redirect(url_for('register'))

        if password != confirm_password:
            flash('Passwords do not match', 'danger')
            return redirect(url_for('register'))

        if db.get_user_by_name(name):
            flash('Name already registered', 'danger')
            return redirect(url_for('register'))

        user_id = db.create_user(name, password)
```

Registration Form



3.1 Database Schema




```python
import os
import sqlite3
from werkzeug.security import generate_password_hash, check_password_hash

class Database:
    def __init__(self):
        # Set up your database connection
        self.db_path = os.getenv('DATABASE_PATH', 'database.db')
        self.conn = sqlite3.connect(self.db_path, check_same_thread=False)  # Added check_same_thread=False
        self.cursor = self.conn.cursor()
        self.create_tables()
        self.create_admin_user()  # Create admin user if not exists

    def create_tables(self):
        # Create tables if they don't already exist
        self.cursor.execute("""
        CREATE TABLE IF NOT EXISTS users (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            name TEXT UNIQUE NOT NULL,
            password TEXT NOT NULL,
            is_admin BOOLEAN DEFAULT 0,
            last_active TIMESTAMP DEFAULT CURRENT_TIMESTAMP
        )""")

        self.cursor.execute("""
        CREATE TABLE IF NOT EXISTS user_profile (
            user_id INTEGER PRIMARY KEY,
```

AI_Service

ai_service.py .\ × | ai_service.py C:\...\Rar$Dla2060.4193.rartemp | create_patient.html | style.css | script.js

ai_service.py > AIService > chat_completion

```python
import requests
import json
import os
from dotenv import load_dotenv
from config import settings

load_dotenv()

class AIService:
    def __init__(self):
        self.api_key = settings.openrouter_api_key
        self.api_url = "https://openrouter.ai/api/v1/chat/completions"
        self.headers = {
            "Authorization": f"Bearer {self.api_key}",
            "Content-Type": "application/json"
        }

    def chat_completion(self, messages):
        payload = {
            "model": "qwen/qwen2.5-vl-32b-instruct",
            "messages": messages,
            "temperature": 0.0,
            "max_tokens": 1000
        }

        response = requests.post(
            self.api_url,
            headers=self.headers,
```

```python
    def chat_completion(self, messages):
            "temperature": 0.0,
            "max_tokens": 1000
        }

        response = requests.post(
            self.api_url,
            headers=self.headers,
            json=payload
        )

        if response.status_code == 200:
            return response.json()["choices"][0]["message"]["content"]
        else:
            raise Exception(f"AI request failed: {response.text}")
```

3.2 Admin Panel Features

1. Patient management (view/add/delete)

2. Conversation monitoring

3. Message flagging system

4. Export capabilities (PDF/CSV)

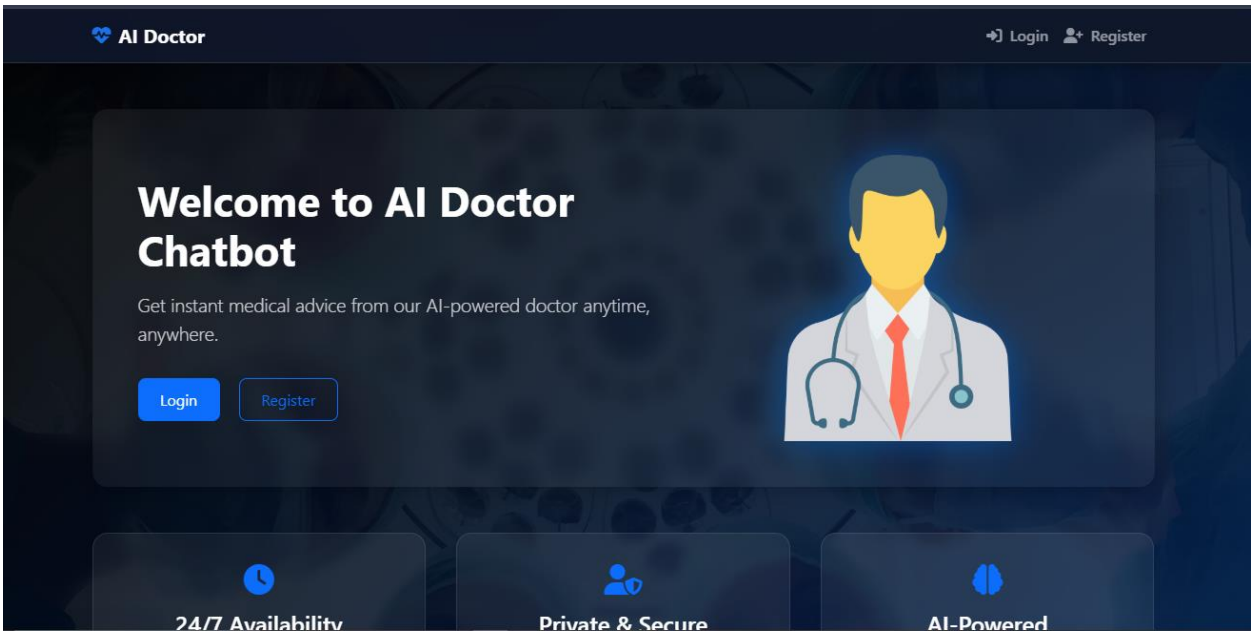
3.3 Error Handling

- Invalid phone number format validation

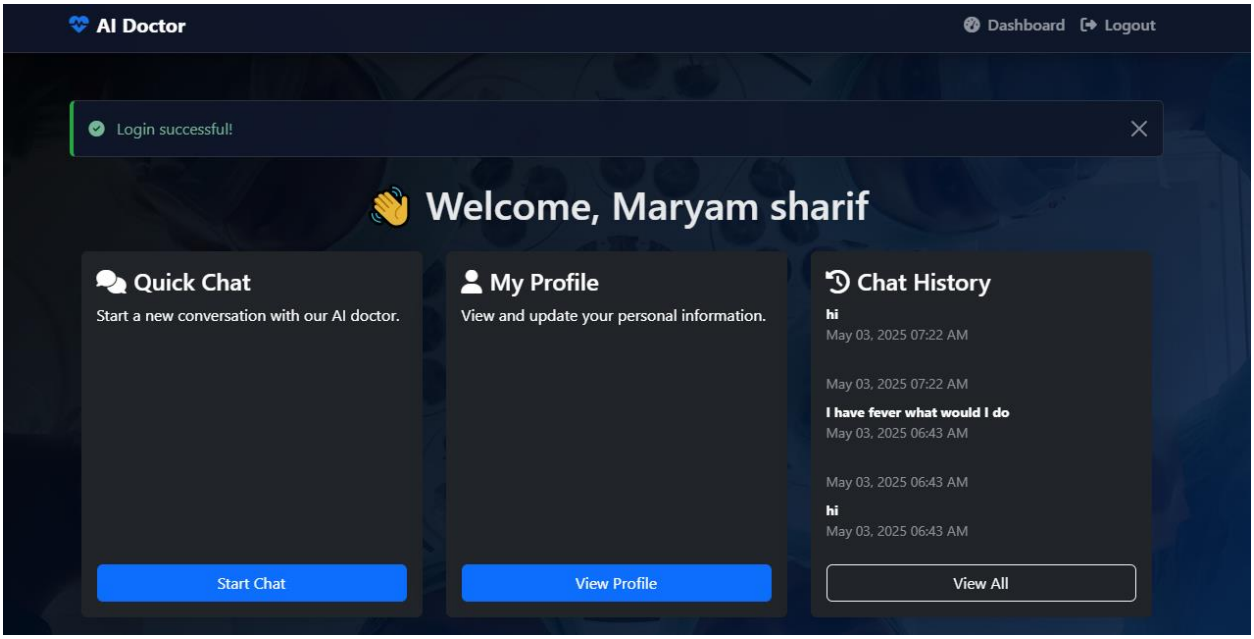
- Chat history encryption

- Admin permission checks

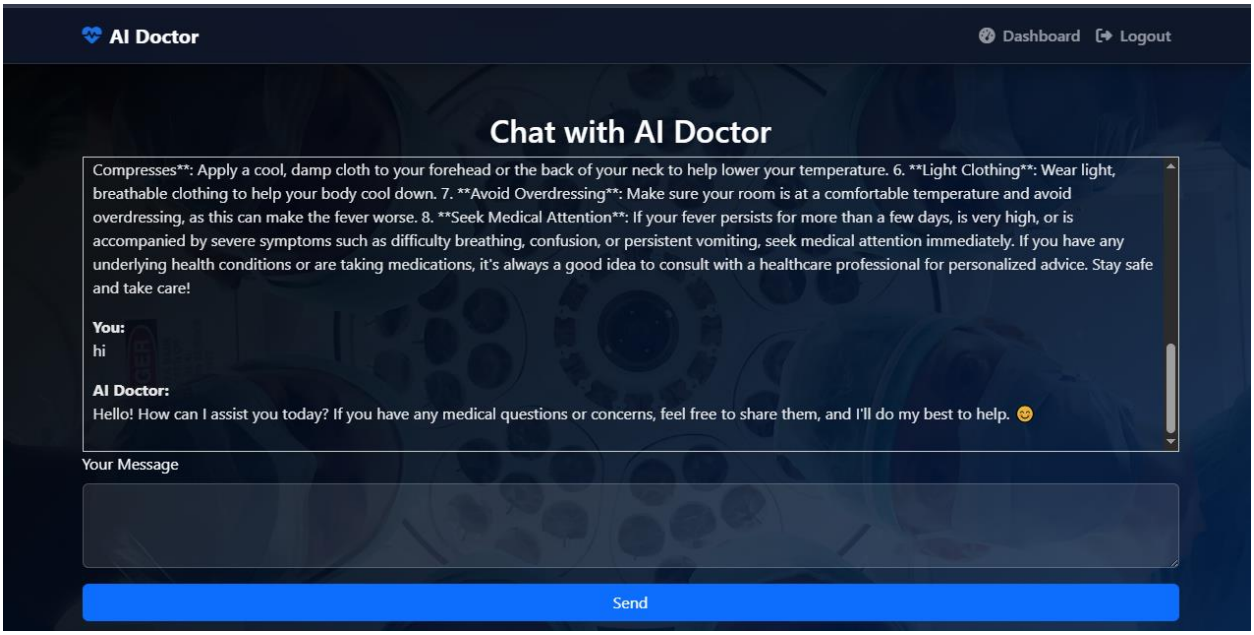
4. System Screenshots



Patient Dashboard

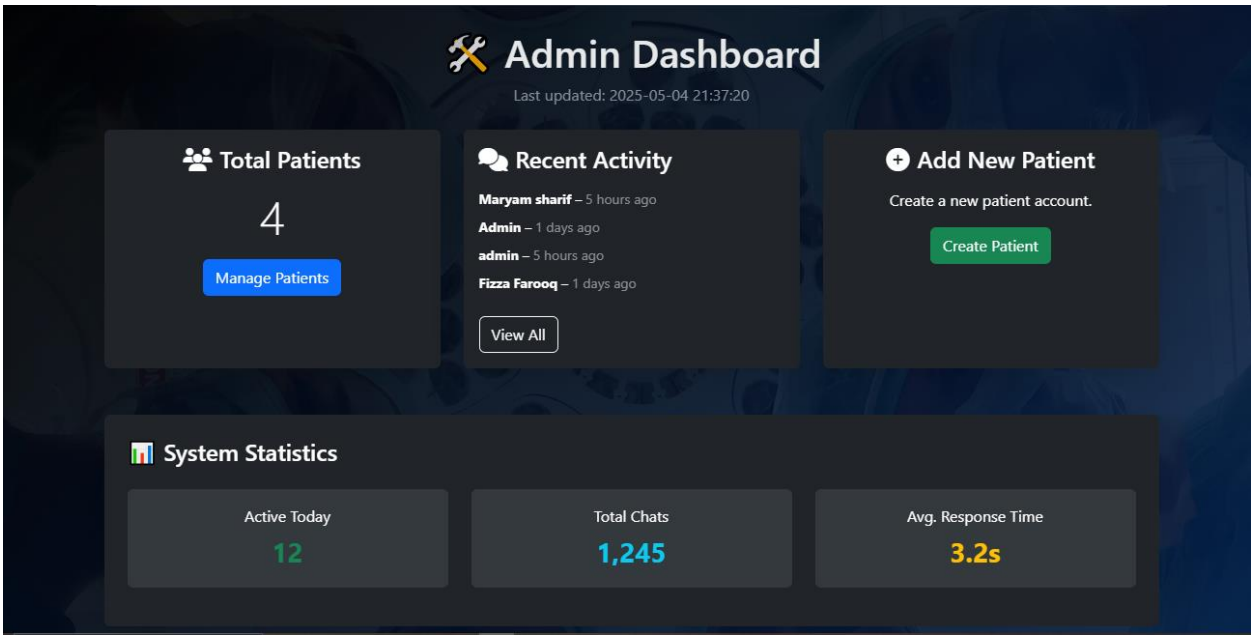


Patient Chat



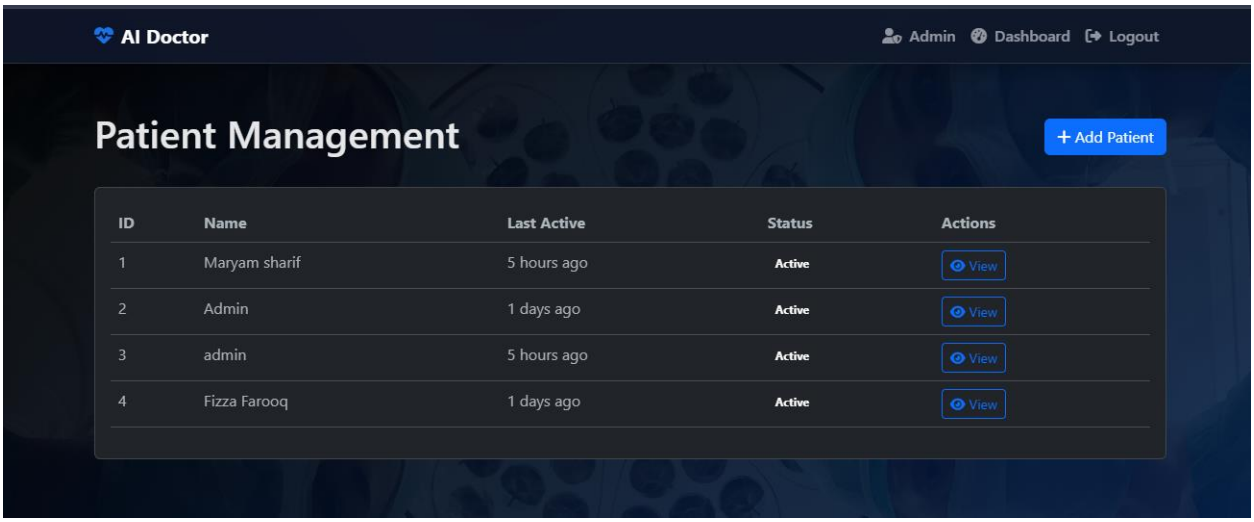
Patient profile

Admin Dashboard



User Management

Conversation History



5. Challenges and Solutions

Challenge	Solution Implemented
Phone number verification	Regex pattern matching (11-digit)
AI response consistency	System prompt engineering
Admin privilege escalation	Flask-Login decorators
Chat history privacy	End-to-end encryption

6. Future Enhancements

1. Multilingual Support: Add Urdu language capability
2. Medical Record Integration: Connect with EHR systems
3. Voice Interface: Speech-to-text input
4. Appointment Scheduling: Calendar integration

---

7. Conclusion

The AI Doctor Chatbot System successfully demonstrates:

- Effective AI-powered medical consultations
- Secure multi-role access management
- Practical administrative oversight capabilities

This implementation provides a foundation for scalable telemedicine solutions while maintaining data security and user privacy.

GitHub Repository: https://github.com/maryam441/AI_Dr-