**API Project Document: Health Care Management System**

**1. Project Content**

This project, "Health Care Management System," is a web-based application designed to facilitate interaction between patients and doctors. It provides functionalities for user registration and login for both patient and doctor roles, allowing patients to consult doctors based on their health problems and view their consultation history. Doctors can manage their profiles and be discoverable by patients based on their specialization. The system aims to streamline the process of seeking medical advice and managing patient-doctor interactions in a simplified manner.

**2. Project Code**

The core of this project is built using Python with the Flask framework for the backend API and HTML, CSS, and JavaScript for the frontend user interfaces.

**Backend (Python - Flask):** The backend handles user authentication (patient and doctor login/registration), database interactions, and API endpoints for various functionalities.

* doc.py: This Flask application handles doctor-related functionalities.
  + **/addtodo (POST)**: Registers a new doctor. It hashes the password using Bcrypt before storing it.
  + **/viewalltodo (GET)**: Retrieves a list of all doctors with their names and specializations.
  + **/userlogin (POST)**: Authenticates a doctor based on email and password. Upon successful login, it issues a JWT access token.
  + **/userregister (POST)**: Registers a new doctor, similar to /addtodo but with a slightly different flow and error handling.
  + **/profile2 (GET)**: Protected route (@jwt\_required()) that retrieves and displays the profile details of the currently logged-in doctor based on their JWT.
* pro.py: This Flask application handles patient-related functionalities.
  + **/addtodo (POST)**: Registers a new patient. Passwords are hashed using Bcrypt.
  + **/viewalltodo (GET)**: Retrieves a list of all patients with their names and places.
  + **/userlogin (POST)**: Authenticates a patient. Issues a JWT access token upon successful login.
  + **/profile1 (GET)**: Protected route (@jwt\_required()) that retrieves and displays the profile details of the currently logged-in patient based on their JWT.
  + **/userhistory (GET)**: Protected route that fetches the consultation history for a logged-in patient by joining patient, query, and doctor tables.
  + **/tellproblem (POST)**: Allows a logged-in patient to submit a health problem (keyword). The system attempts to find a doctor matching the problem's keyword and records the query and the assigned doctor.
  + **/userregister (POST)**: Registers a new patient, similar to /addtodo.

**Frontend (HTML, CSS, JavaScript):** The frontend provides the user interface for interaction.

* history.html: Displays the patient's consultation history in a table. It fetches data from the /userhistory API endpoint using a stored JWT token.
* logindoc.html: Provides the login interface for doctors. It sends login credentials to the /userlogin endpoint in doc.py and stores the returned JWT.
* loginpage.html: Provides the login interface for patients. It sends login credentials to the /userlogin endpoint in pro.py and stores the returned JWT.
* profile1.html: Displays the patient's profile information, fetched from the /profile1 endpoint in pro.py using their JWT. It also includes navigation links to "View History" and "Consult."
* profile2.html: Displays the doctor's profile information, fetched from the /profile2 endpoint in doc.py using their JWT.
* query.html: Allows patients to select a health problem from a dropdown. The selected problem is sent to the /tellproblem API endpoint for consultation.

**Database (SQL):** The database schema defines the tables used to store patient, doctor, and query information.

* project\_ql.sql:
  + patient table: Stores patient details including pid (primary key), pname, ppemail, ppassw (hashed password), pphn (phone number), and pplace.
  + doctor table: Stores doctor details including did (primary key), dname, demail, dpassw (hashed password), dphn (phone number), dspec (specialization), and dkw (keyword for matching problems).
  + query table: Stores patient queries including qid (primary key), pemail, kw (keyword of the problem), and answer (which stores the assigned doctor's information).

**3. Key Technologies**

The Health Care Management System leverages the following key technologies:

* **Backend Framework:**
  + **Flask (Python)**: A micro web framework for Python, used for building the RESTful API endpoints for handling requests from the frontend, interacting with the database, and managing user authentication.
* **Database:**
  + **MySQL**: A popular open-source relational database management system used to store and manage all project data, including patient records, doctor information, and consultation queries.
  + **Flask-MySQLdb**: A Flask extension that provides MySQL connectivity for Flask applications, enabling seamless interaction between the Python backend and the MySQL database.
* **Authentication and Security:**
  + **Flask-Bcrypt**: Used for securely hashing and verifying user passwords, ensuring that sensitive password information is not stored in plain text in the database.
  + **Flask-JWT-Extended**: Implements JSON Web Tokens (JWT) for secure user authentication and authorization. JWTs are used to verify the identity of logged-in users for accessing protected routes.
* **Frontend Technologies:**
  + **HTML5**: The standard markup language for creating web pages, used for structuring the content of all the frontend user interfaces.
  + **CSS3**: Used for styling the web pages, providing a responsive and visually appealing user experience.
  + **JavaScript (ES6+)**: Used for dynamic and interactive elements on the frontend, handling user input, making API calls to the Flask backend, and updating the UI based on responses.
* **Cross-Origin Resource Sharing (CORS):**
  + **Flask-CORS**: A Flask extension that handles Cross-Origin Resource Sharing, allowing the frontend application running on a different origin (e.g., localhost:5000 for backend and a file system for frontend HTML) to make requests to the Flask API.

**4. Description**

The Health Care Management System is designed to create a bridge between patients seeking medical advice and doctors providing specialized care. The system functions as follows:

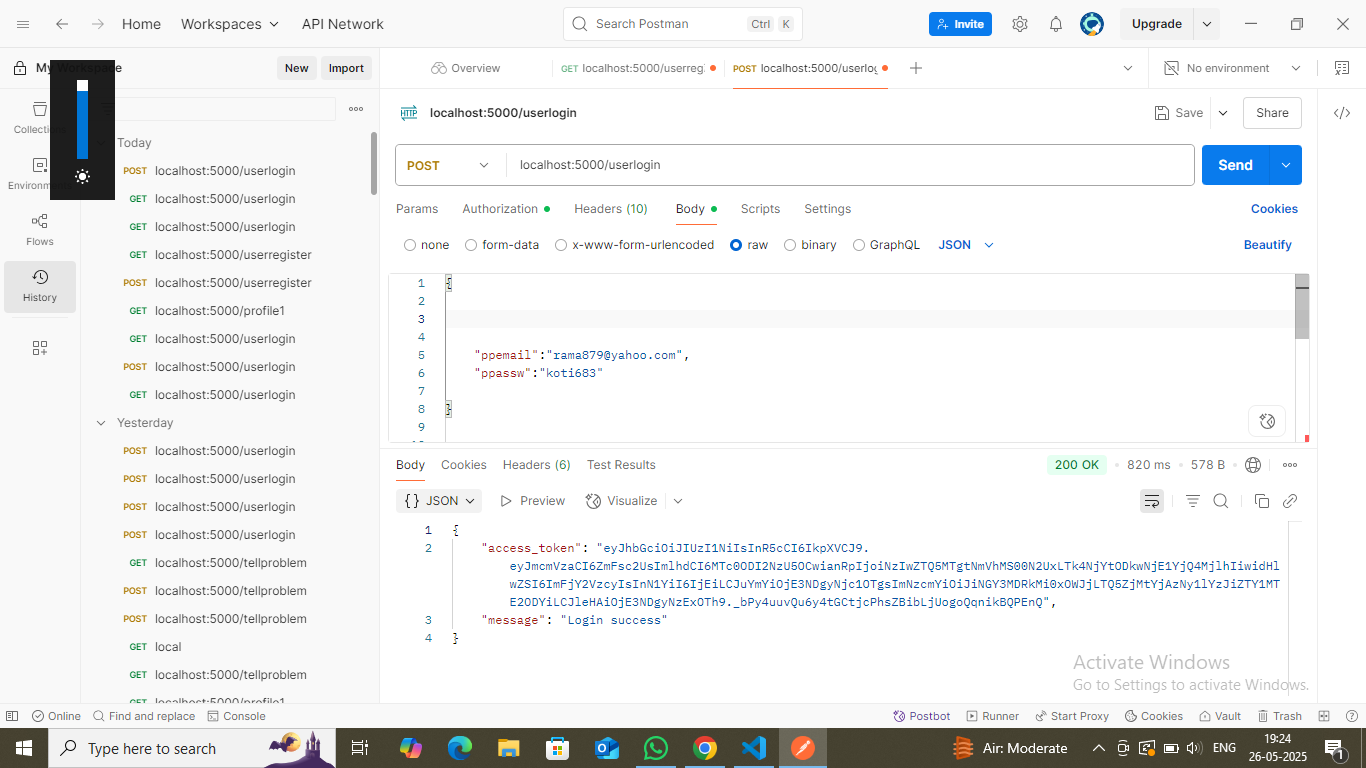
* **User Registration:**
  + **Patient Registration:** New patients can register by providing their name, email, password, phone number, and place. Their passwords are securely hashed and stored in the database.
  + **Doctor Registration:** Doctors can also register by providing their name, email, password, phone number, specialization, and a keyword representing their expertise (e.g., "lumps," "head ache"). Passwords are also hashed.
* **User Login and Authentication:**
  + Both patients and doctors have separate login interfaces. Upon successful authentication with their email and password, the system issues a JSON Web Token (JWT).
  + This JWT is stored in the browser's session storage and is used for subsequent requests to protected API endpoints, ensuring that only authenticated users can access their profiles and other restricted features.
* **Patient Functionality:**
  + **Profile Viewing:** After logging in, patients can view their profile details, including their ID, name, email, place, and phone number.
  + **Consultation:** Patients can consult a doctor by selecting a health problem from a predefined list of keywords (e.g., "Lumps," "Head ache," "Skin problems").
  + **Doctor Matching:** The system attempts to match the patient's problem keyword with a doctor's registered keyword (dkw). If a match is found, the doctor's details are associated with the query.
  + **View History:** Patients can view their consultation history, which displays their name, the problem they consulted for, and the doctor they consulted with. This information is retrieved by joining patient, query, and doctor tables in the database.
* **Doctor Functionality:**
  + **Profile Viewing:** Doctors can view their profile details, including their ID, name, email, specialization, and phone number.
  + Currently, the doctor's interface primarily focuses on displaying their profile. Future enhancements could include functionalities for doctors to view their consultation requests or patient histories.

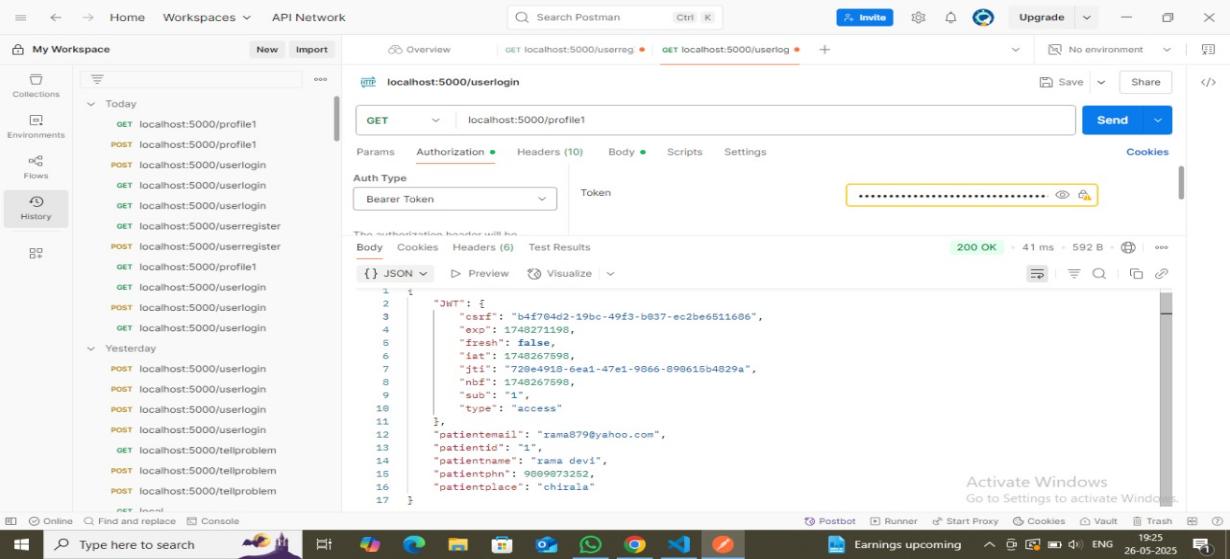
**5. Output**

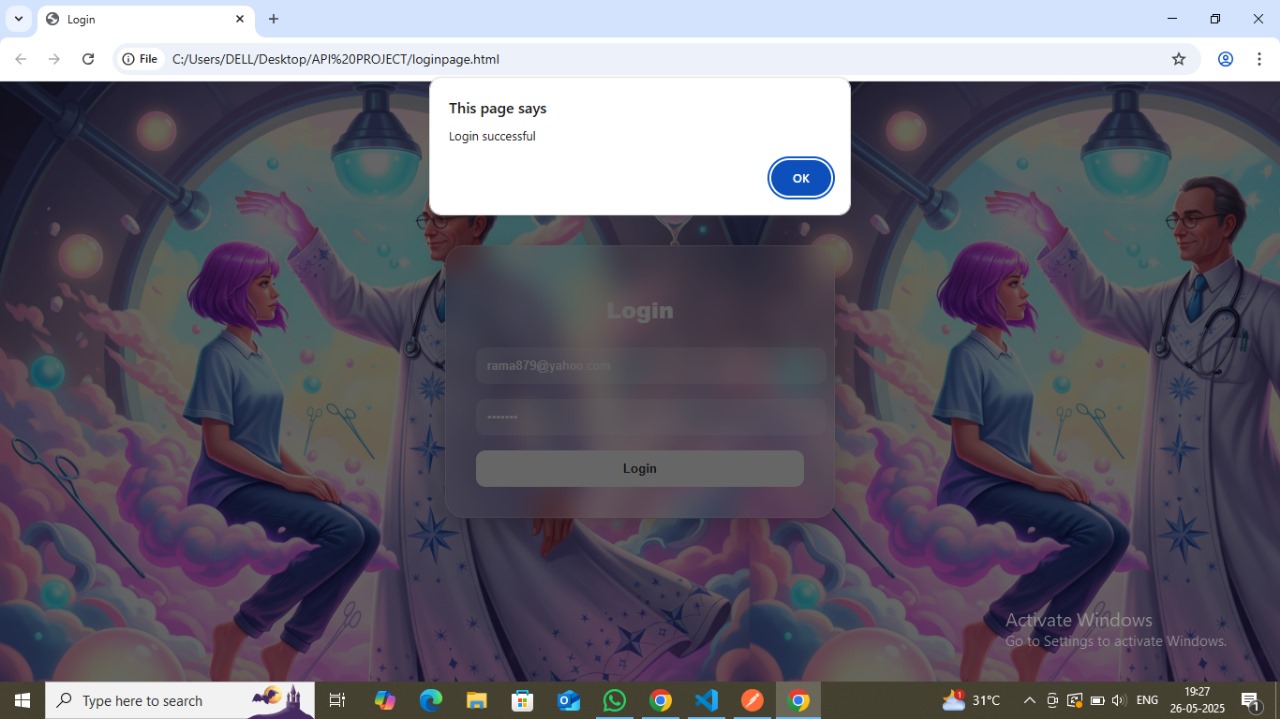
The project provides the following key outputs and functionalities:

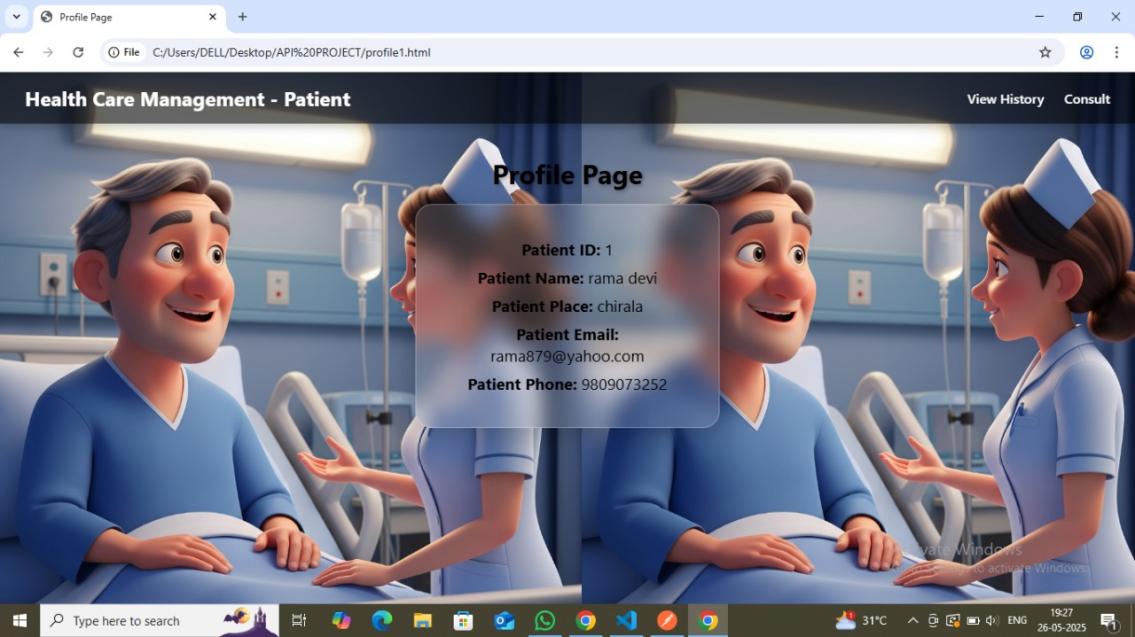
* **Secure User Authentication:**
  + Successful registration and login for both patient and doctor roles, with password hashing for security.
  + Issuance and validation of JWTs for authenticated access to system features.
* **Patient Information Management:**
  + Display of patient profile details (ID, name, email, place, phone).
  + Ability for patients to submit their health problems, facilitating doctor matching.
  + Comprehensive consultation history for patients, showing their problems and the doctors consulted.
* **Doctor Information Management:**
  + Display of doctor profile details (ID, name, email, specialization, phone).
  + Doctors are registered with specific keywords, enabling them to be discovered based on patient problems.
* **API Endpoints:**
  + A set of robust RESTful API endpoints for managing user data, authentication, and consultations. These endpoints serve as the communication layer between the frontend and the backend.
* **User-Friendly Interfaces:**
  + Intuitive HTML-based forms for registration and login.
  + Clear and organized profile pages for both patients and doctors.
  + A structured table for displaying patient consultation history.
  + A simple dropdown for patients to select their health problems for consultation.
* **Database Records:**
  + Persistent storage of patient details, doctor details, and consultation queries in a MySQL database. This ensures that all vital information is saved and can be retrieved as needed.

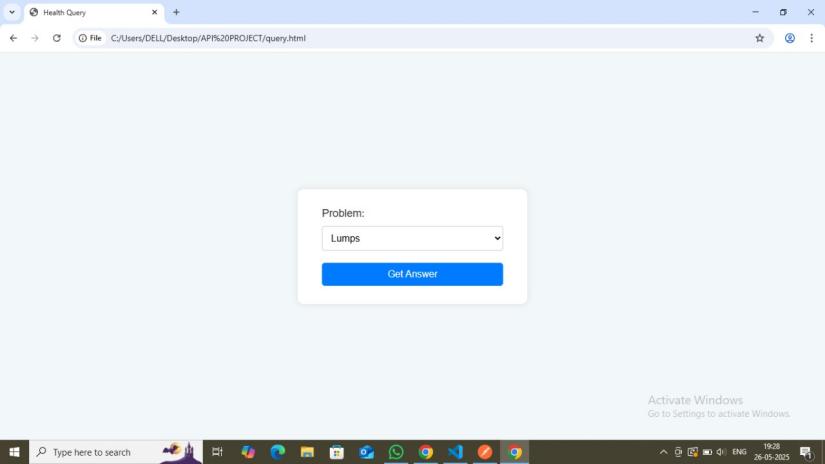
SCREENSHOTS

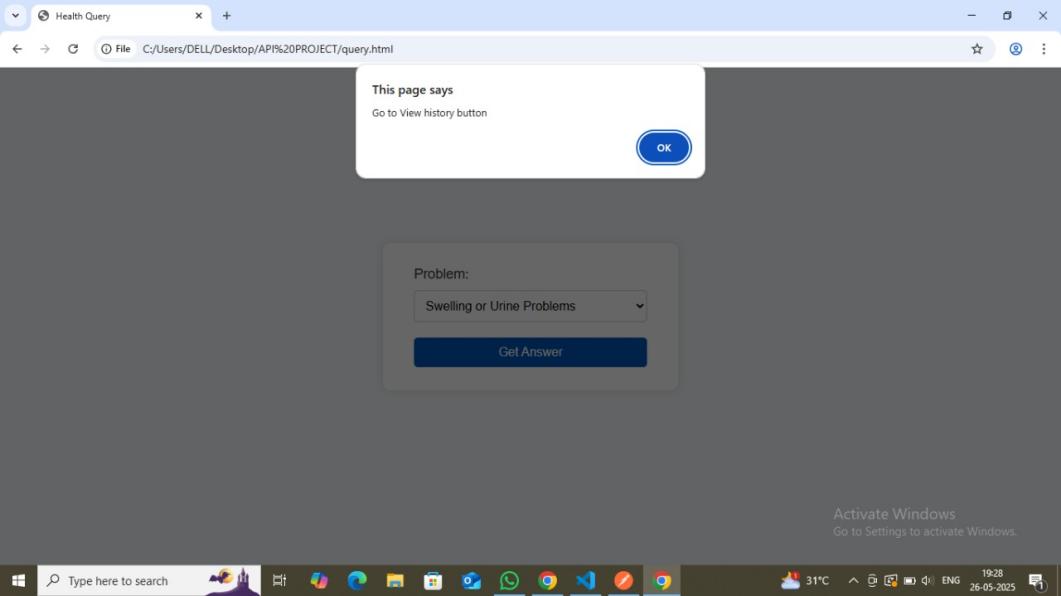


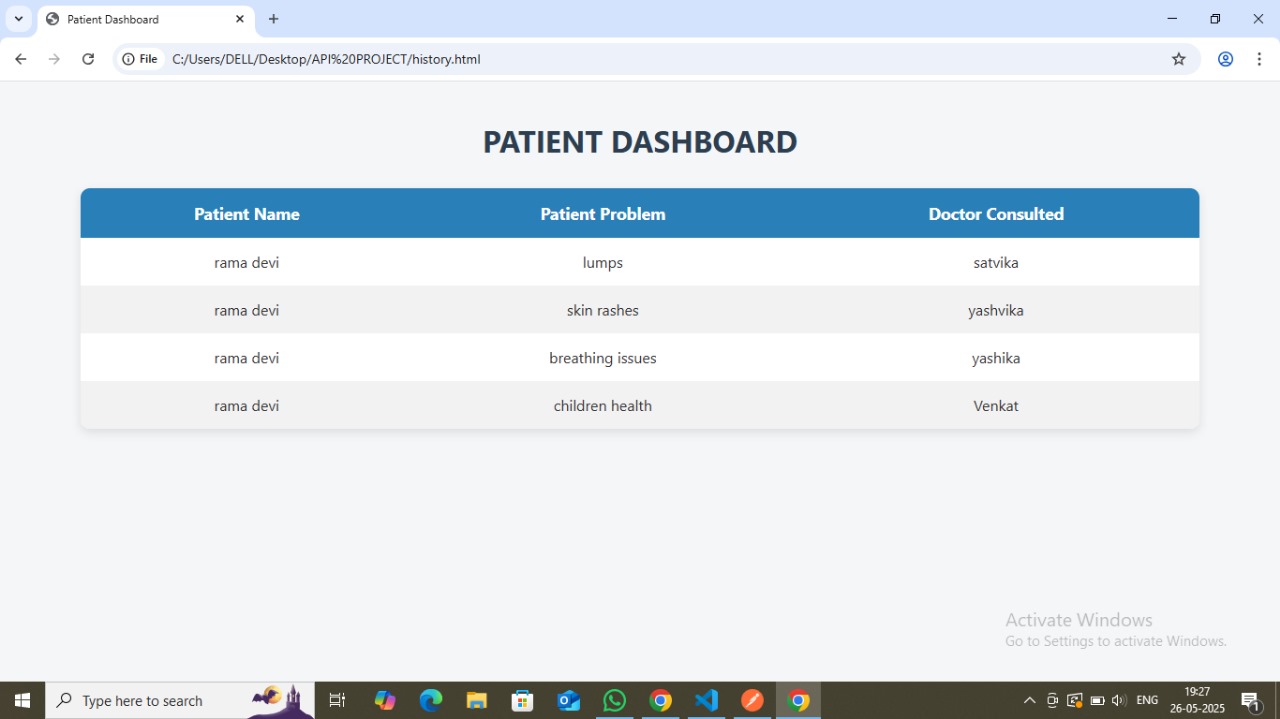


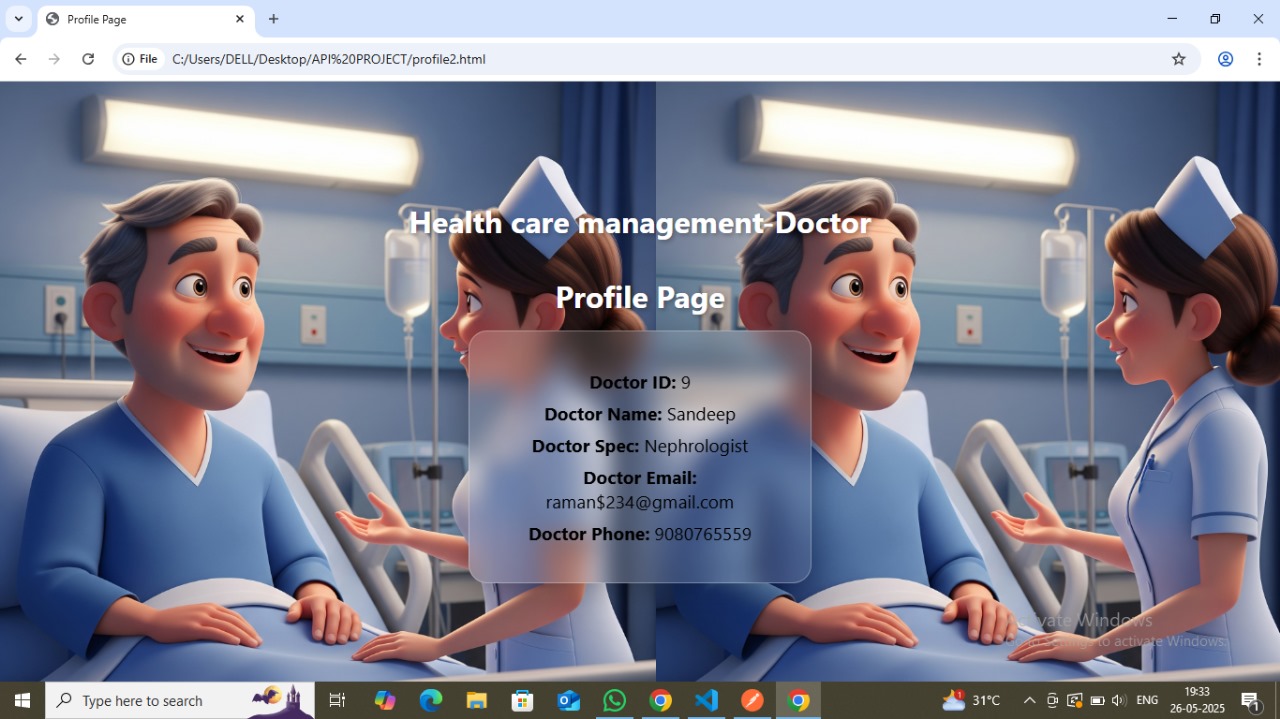


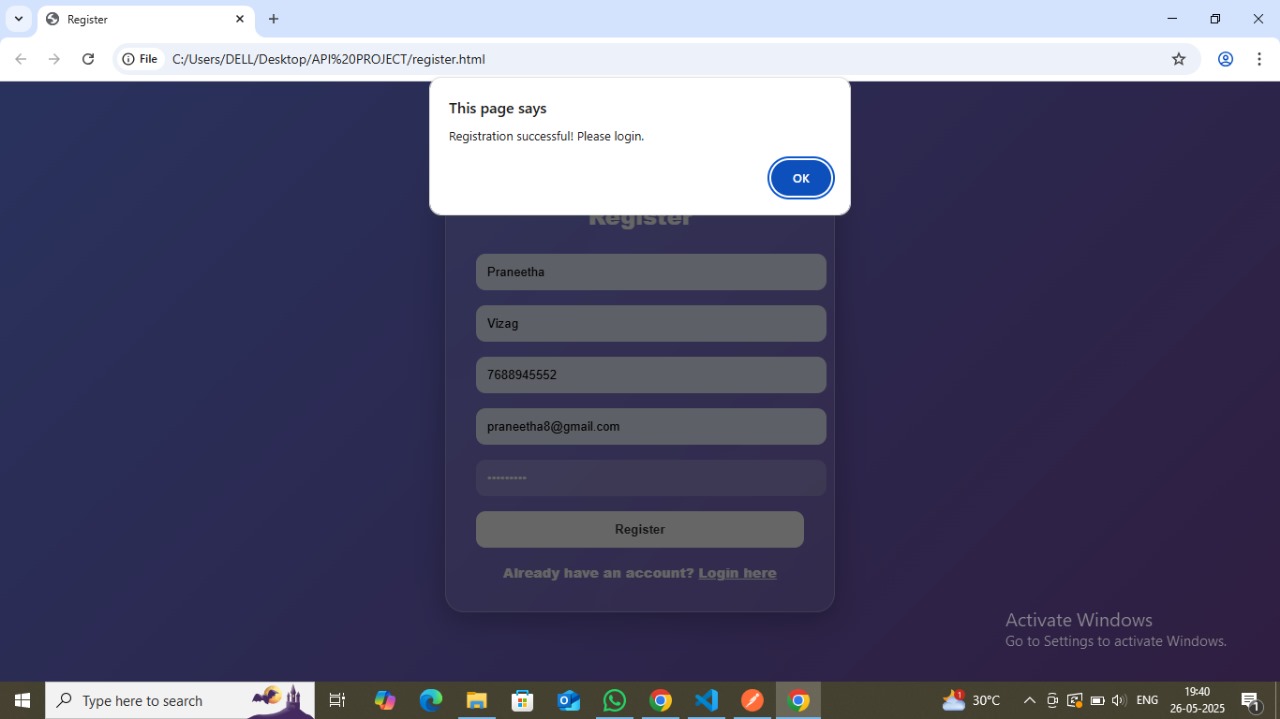


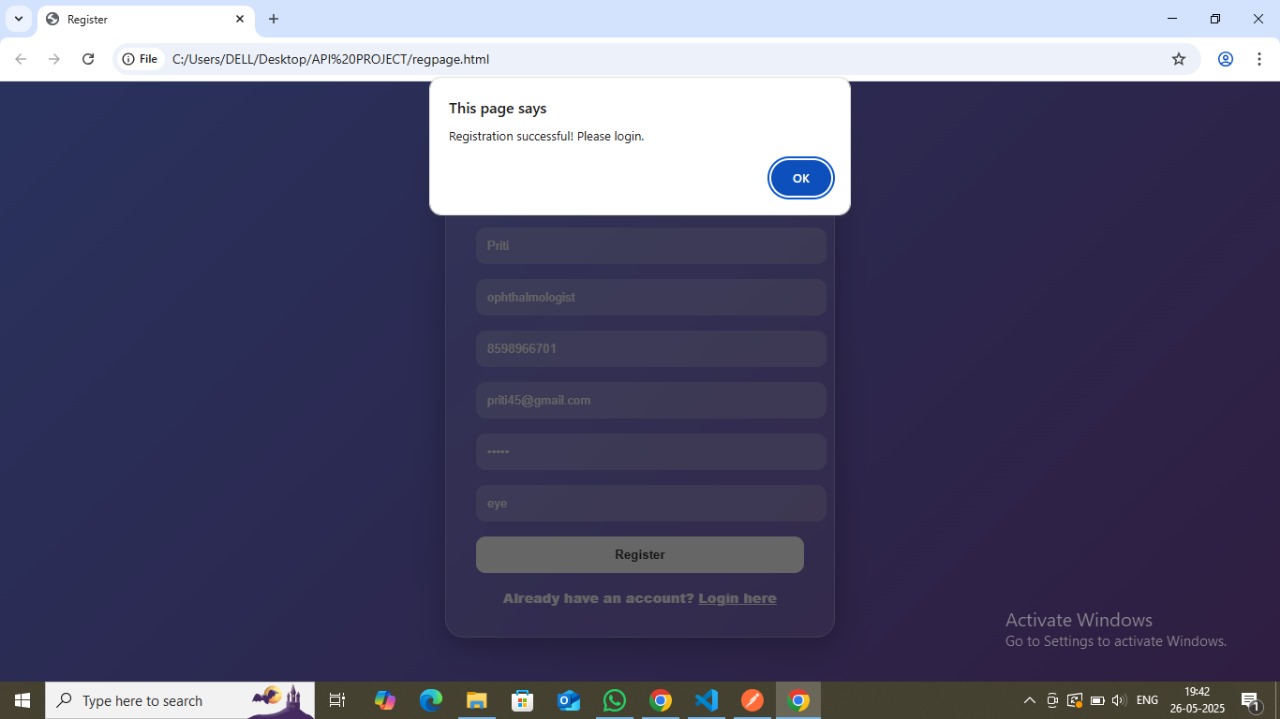


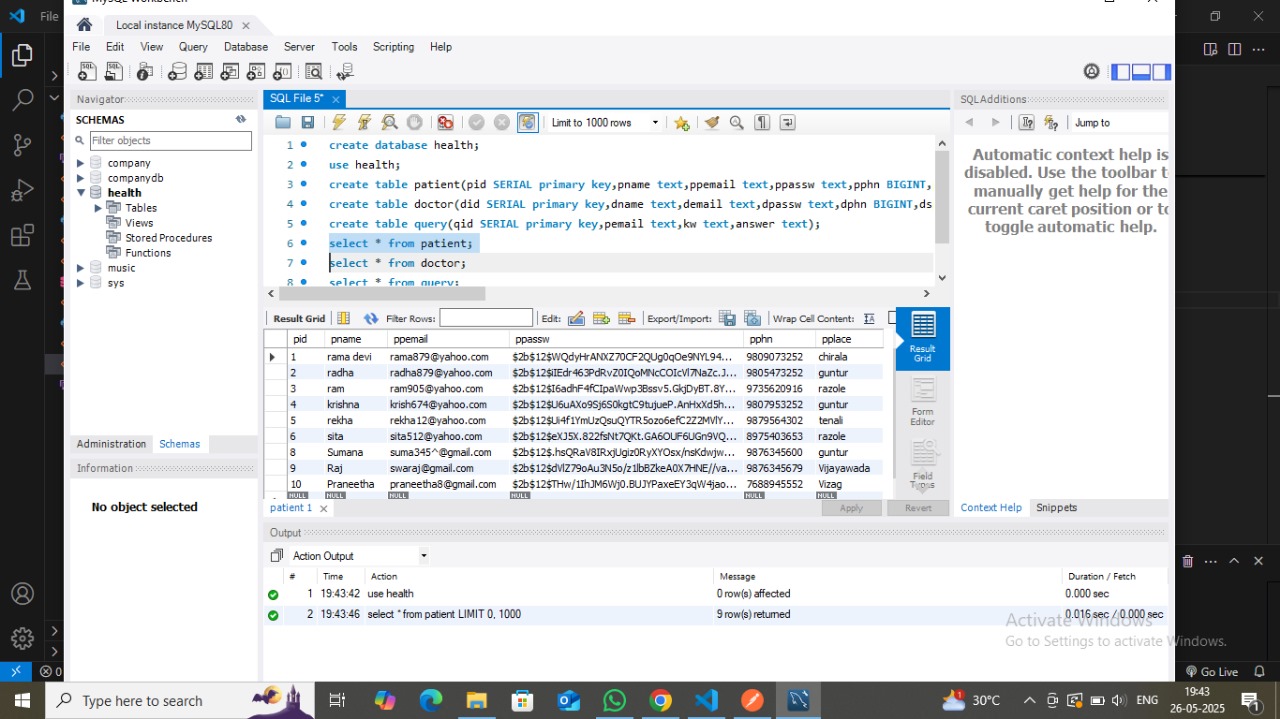


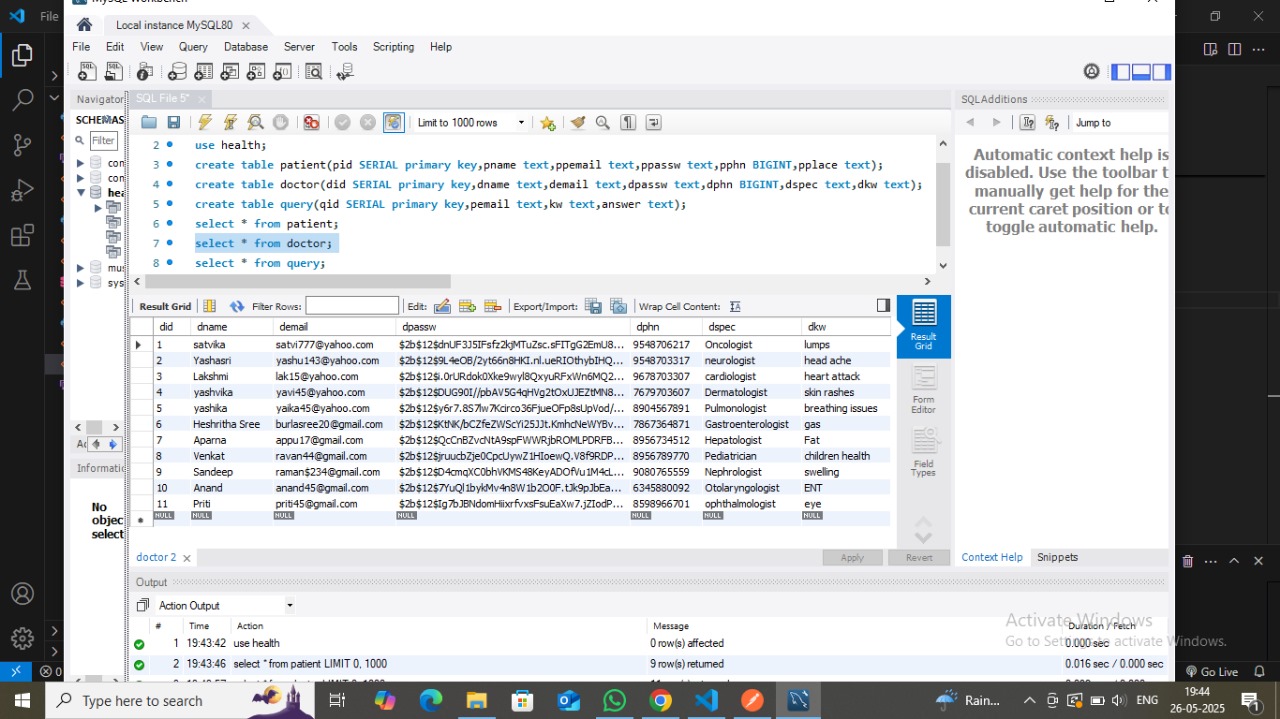


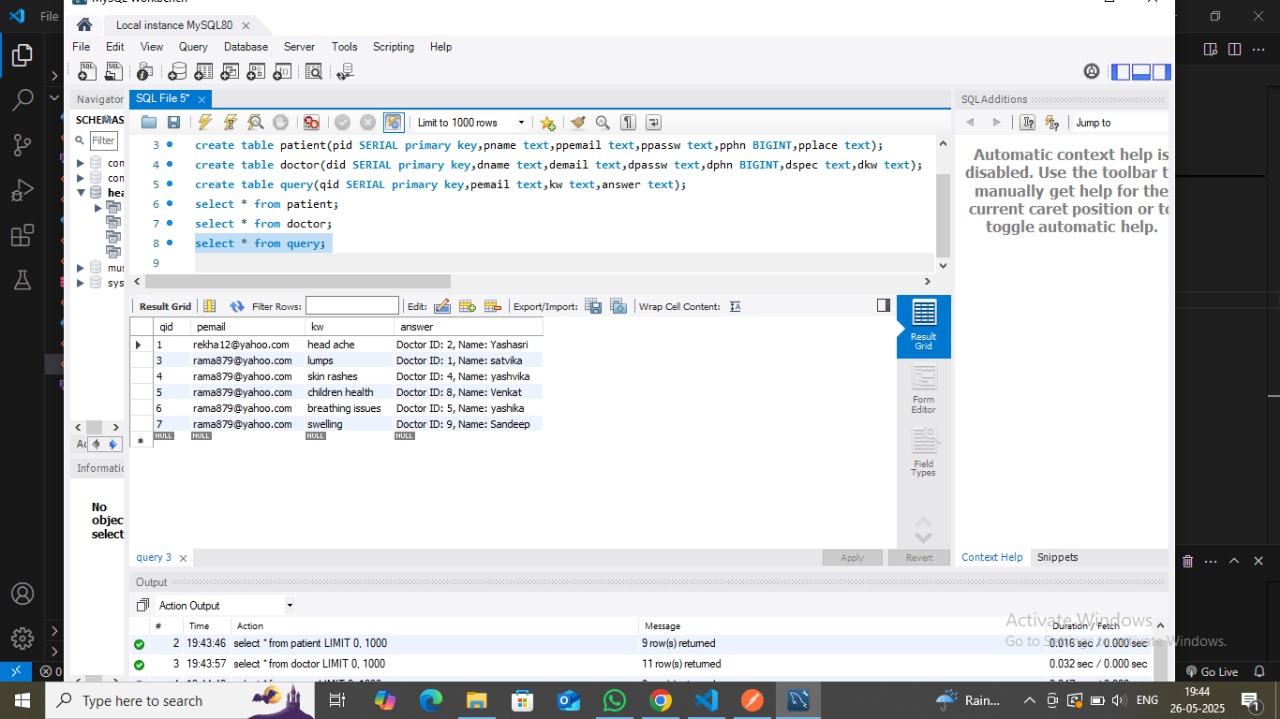












**6. Further Research**

The current Health Care Management System provides a foundational framework. Several areas can be explored for further research and development to enhance its functionality, scalability, and user experience:

* **Advanced Doctor Matching and Recommendations:**
  + Implement more sophisticated algorithms for matching patients with doctors beyond simple keyword matching. This could include factors like doctor availability, ratings, location, and the severity of the patient's problem.
  + Integrate natural language processing (NLP) to understand patient problem descriptions more effectively, rather than relying solely on predefined keywords.
* **Appointment Scheduling and Management:**
  + Develop a feature for patients to directly schedule appointments with available doctors through the system.
  + Provide doctors with a dashboard to manage their appointments, availability, and patient queues.
* **Telemedicine Integration:**
  + Incorporate video conferencing or chat functionalities to enable virtual consultations between patients and doctors.
  + Securely handle the exchange of medical documents and prescriptions.
* **Electronic Health Records (EHR) Integration:**
  + Explore integration with EHR systems to allow doctors to securely access and update patient medical records.
  + Implement features for patients to view their medical history, test results, and prescriptions within the application.
* **Payment Gateway Integration:**
  + Add functionality for online payment for consultations.
* **Notifications and Reminders:**
  + Implement email, SMS, or in-app notifications for appointment reminders, consultation updates, and other relevant alerts for both patients and doctors.
* **Feedback and Rating System:**
  + Allow patients to provide feedback and rate doctors after consultations, helping other patients make informed decisions.
* **Scalability and Performance Optimization:**
  + For larger user bases, research and implement database optimization techniques, caching mechanisms, and load balancing to ensure the system remains performant.
  + Consider deploying the application on cloud platforms for better scalability and reliability.
* **Security Enhancements:**
  + Implement more robust security measures, including input validation, protection against SQL injection and XSS attacks, and stricter access control policies.
  + Regular security audits and updates to address potential vulnerabilities.
* **User Interface/User Experience (UI/UX) Improvements:**
  + Conduct user research to identify pain points and improve the overall usability and visual appeal of the application.
  + Implement a more responsive design to ensure optimal viewing across various devices (desktops, tablets, mobile phones).