Telemedicine consultation platform using c++

NAME - RN Umashankar

REVA UNIVERSITY

B TECH Electronics and communication

7TH semester

Introduction to Telemedicine Consultation Platform using C++

- A Telemedicine Consultation Platform enables healthcare providers and patients to interact remotely, allowing for convenient and accessible healthcare services,
- especially in underserved or remote areas. Implementing such a platform in C++ requires leveraging its robust memory management, performance optimization capabilities,
- and wide-ranging library support to handle tasks like secure user authentication, appointment scheduling, real-time chat, and data

- **This C++-based platform design focuses on creating a secure, efficient, and interactive environment where patients can book appointments, consult doctors via real-time messaging,**
- **and securely store their medical information. This framework could be adapted further for telemedicine features like video conferencing, medical record access, and AI-driv**

Key Objectives of the Platform:

1. Convenience and Accessibility: Provides patients with the flexibility to consult healthcare professionals from their homes, reducing travel time and costs.

2. Efficient Communication: Enables real-time messaging for quick question resolution, advice, or follow-ups.

3 .Data Security: Ensures that sensitive patient data is securely handled, stored, and transmitted.

4. Scalability: Allows easy integration of additional features like video consultation, diagnostic tools, or AI analysis over time.

METHODOLOGY

Requirements Analysis

- **Stakeholder Engagement**: Identify stakeholders such as healthcare providers, patients, and administrative staff. Conduct interviews and surveys to gather requirements.
-
- **Functional Requirements**:User registration and authentication.
- Appointment scheduling.
- Real-time video/audio consultations.
- Access to medical records.
- Prescription management.
- Payment processing.

- **System Architecture Design**
- **Architecture Style**: Choose a microservices architecture. Each service can be a separate C++ application,
- allowing for independent scaling and updates.
- **Technology Stack**:
  - **Backend**: C++ for implementing core services using frameworks like Crow or Pistache.
  - **Frontend**: Use JavaScript frameworks (e.g., React or Angular) for the user interface.
  - **Database**: Utilize PostgreSQL or MongoDB for data storage.
  - **Real-Time Communication**: Implement WebRTC for real-time audio/video.

- **User Interface Design**
- **Wireframes and Prototypes**: Create detailed wireframes for the web application using design tools. Focus on user flows for both patients and doctors.
- **Frontend Integration**: Design RESTful APIs in C++ that the frontend will interact with.
- **4. Database Design**
- **Schema Definition**: Create a relational database schema to handle users, appointments, consultations, and prescriptions. For example:
    - **Users Table**: UserID, Name, Email, Role (patient/doctor), etc.
    - **Appointments Table**: AppointmentID, UserID, DoctorID, DateTime, Status, etc.
- **C++ Database Access**: Use libraries like libpqxx (for PostgreSQL) to manage database connections and queries from C++.

- **Real-Time Communication Implementation**
- **WebRTC Integration**: Implement signaling mechanisms in C++ for establishing peer-to-peer connections for video calls.

- **Media Management**: Handle audio/video streams using appropriate C++ libraries or directly through WebRTC APIs.
- **7. Security Measures**

- **Authentication**: Implement JWT or OAuth2 for user authentication. Ensure secure token generation and validation in C++.

- **Data Encryption**: Use SSL/TLS for all communications. Implement encryption for sensitive data stored in the database.

- **8.Testing and Quality Assurance**
- **Unit Testing**: Write unit tests for C++ modules using Google Test or Catch2.
- **Integration Testing**: Ensure all services work together smoothly by simulating real user interactions.
- **User Acceptance Testing**: Conduct UAT sessions with real users to validate the platform's functionality and usability.
- **9. Deployment and Scalability**
- **Cloud Deployment**: Deploy services on cloud platforms (e.g., AWS, Azure) using Docker containers for scalability.
- **Load Balancing**: Implement load balancing to handle multiple concurrent users, ensuring reliability during peak usage.

- **10.Maintenance and Support**
- **Regular Updates**: Establish a process for routine maintenance, including security updates and feature enhancements.
- **User Support Channels**: Provide support through chat, email, or ticketing systems for troubleshooting and assistance.
- **Feedback Collection**: Implement mechanisms for collecting user feedback to continually improve the platform.

Core Functionalities in the Platform

- 1. User Authentication:A fundamental feature for security, user authentication ensures that only registered users and verified doctors access the system.In this implementation,
- passwords are securely hashed using the SHA-256 algorithm to prevent data breaches and unauthorized access.
- A SQLite database holds the hashed passwords and manages login information.

Core Functionalities in the Platform (contunation)

- 2. Appointment Scheduling:
- The platform allows patients to view available time slots, book, reschedule, or cancel appointments.
- Doctors can manage their schedules based on availability and patient bookings.
- SQLite is utilized for appointment data management, ensuring seamless scheduling and easy tracking of appointments.

Core Functionalities ir the Platform(contunation

- 3. Real-Time Chat Functionality:Real-time communication enables patients and doctors to interact and exchange information efficiently.

- C++ libraries like Boost.Asio allow for asynchronous, non-blocking message handling, which is critical for real-time communication.

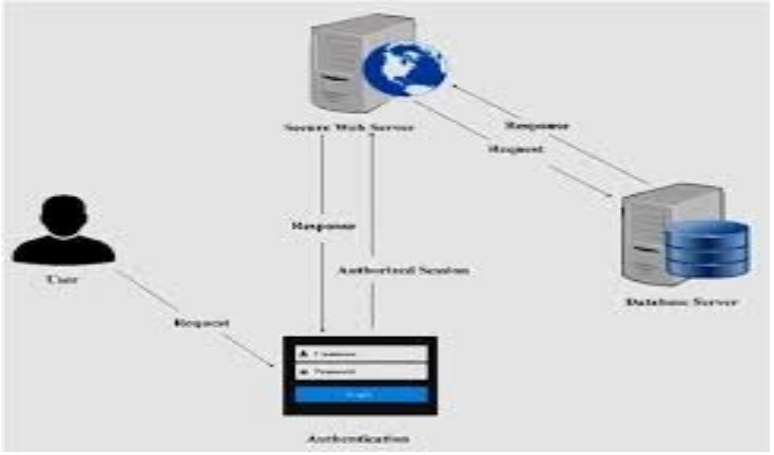
- The chat is a text-based module that can be extended to handle more complex data or media types in the future.

Core Functionalities in the Platform(contunation)

- 4. Database Management:A reliable database is crucial for storing and retrieving user data, appointments, and chat logs.
- This platform uses SQLite for lightweight database management, with tables structured for users, doctors, and appointments.
- Database functions also ensure that data integrity is maintained during storage, update, and retrieval.

Core Functionalities in the Platform(contunation)

- 5. Session Management and Security:Session management tracks user activities and handles automatic logouts after inactivity, securing the system against unauthorized access.
- Communication between the client and server can be further secured with SSL/TLS to protect data integrity and privacy.



Benefits of Implementing in C++

- implementing this platform in C++ offers several advantages:High Performance: C++ is known for its speed, which is essential for real-time applications like telemedicine.
- Memory Management: C++ allows fine-grained control over memory usage, enhancing performance and stability.
- Cross-Platform Compatibility: With libraries like Qt (for GUI) and Boost (for networking),
- the platform can be deployed across different operating systems, increasing accessibility.

Challenges and Considerations

- Concurrency: Handling multiple users in real-time requires effective thread management, which can be challenging but achievable with Boost.Asio.Data Security:
- Since healthcare data is sensitive, robust security protocols are essential. Hashing, SSL/TLS, and encryption mechanisms must be carefully implemented and maintained.Scalability:
- Although C++ offers performance and memory management, adding complex new features (like video conferencing) might require more external libraries and additional resource management.

WORKING Step 1: System Setup and Initialization

- 1. Database Setup:The platform starts by initializing the SQLite database to store information related to users, doctors, appointments, and chat history.
- Tables for users, appointments, and sessions are created, ensuring they’re properly structured with constraints (like unique IDs) to maintain data integrity.
- 2. User Authentication Setup:SHA-256 is used to hash passwords, ensuring secure storage in the database.
- The system checks for registered users before granting access, preventing unauthorized users from entering the platform.

Step 2: User Authentication and Login

- 1. User Registration:New users (patients and doctors) register by providing basic details and creating a username and password.
- The password is hashed using SHA-256 before being stored in the database for security.
- On successful registration, users receive confirmation and can proceed to log I
- 2. Login Process:Registered users enter their credentials, which are verified by matching the hashed password in the database.If verification is successful, users are granted access to the platform,
- and a session is created to track their activity.Failed login attempts are logged, and repeated failures could lead to temporary account lockout to enhance security.

Step 3: Appointment Scheduling and Management

- 1. cheduling Appointments:Patients can view available doctors and their time slots.
- When a patient selects a time slot, an appointment entry is added to the database, specifying the user ID, doctor ID, and scheduled time.
- The doctor receives a notification of the appointment, and the time slot is marked as unavailable to avoid double-booking.
- 2. Modifying Appointments:Patients can reschedule or cancel appointments if necessary
- For rescheduling, the system checks availability to prevent conflicts.
- Cancellation removes the appointment entry from the database and updates the doctor's schedule accordingly.

Step 4: Real-Time Chat Functionality

- 1. Chat Initialization:Once an appointment starts, a real-time chat session is initiated using the Boost.Asio library.
- This session allows the patient and doctor to exchange messages, ask questions, and provide advice in real-time.
- 2. Asynchronous Communication:The chat feature is implemented with asynchronous operations to ensure non-blocking communication.
- When a user sends a message, it’s queued, transmitted to the recipient, and displayed instantly, facilitating seamless conversation.

Step 5: Session Management

- 1. Session Creation:When a user logs in, a session is created to keep track of their activities and manage their interactions.
- The session also maintains a timestamp to log the user’s last activity, aiding in tracking idle time.
- 2. Idle Timeout:If a session remains idle beyond a predefined limit (e.g., 10 minutes), the user is logged out automatically to enhance security.
- A notification prompts users to save their work or complete their session, preventing data loss or unauthorized access.

Step 6: Data Security and Privacy

- 1. Data Encryption:All sensitive data, such as chat messages and patient information, is encrypted before storage and transmission, ensuring confidentiality.
- SSL/TLS can be implemented to encrypt communication between the server and clients, protecting against eavesdropping or data tampering.
- 2. Data Access Control:Only authorized personnel (such as doctors) can view patient information or medical history.
- Access to different modules (e.g., appointments, chat logs) is granted based on user roles, ensuring that only relevant data is visible to each user type.

Step 7: Post-Consultation and Feedback

- 1. Session Conclusion:Once the consultation is complete, the chat session is closed, and the doctor marks the consultation as finished.
- The patient receives a summary of the conversation and any prescriptions or advice provided by the doctor.
- 2. Feedback Collection:Patients are prompted to provide feedback on their experience, which is stored in the database.
- The feedback data can be used to improve services, monitor doctor performance, and adjust the platform based on user experience.

Step 8: System Scalability and Maintenance

- 1. calling and Updates:As the user base grows, the platform can transition to more scalable databases like MySQL or PostgreSQL, and load balancing can be implemented for optimal performance.
- Regular updates to the C++ codebase and third-party libraries (such as Boost.Asio and SQLite) ensure that the platform stays up-to-date and secure.
- 2. Error Logging and Monitoring:The system includes error logging for debugging and monitoring purposes.
- Logs are reviewed periodically to detect and resolve issues, enhance performance, and maintain system reliability.

Summary

- Each step in this telemedicine platform has been designed to provide a seamless, secure, and efficient experience for users.
- From authentication to appointment scheduling, real-time communication, and session management,
- the platform addresses critical elements required for a successful remote healthcare solution.
- Additionally, the modular architecture and use of C++ allow for scalability and potential feature enhancements, ensuring the system can adapt to evolving telemedicine needs.

1. User Authentication (CODING PART AND SESSION)

- #include <iostream>
- #include <string>
- #include <sqlite3.h>
- #include <openssl/sha.h>
- // Hash password function using SHA-256std::string hashPassword(const std::string& password)
- {   unsigned char hash[SHA256_DIGEST_LENGTH];
- SHA256((unsigned char*)password.c_str(), password.size(), hash);   std::string hashedPassword;
- for (int i = 0; i < SHA256_DIGEST_LENGTH; i++)
- {   hashedPassword += sprintf("%02x", hash[i]);   }
- return hashedPassword;
- }

- // Registration function
- void registerUser(sqlite3* db, const std::string& username, const std::string& password)
- {   std::string hashedPassword = hashPassword(password);   std::string sql = "INSERT INTO users (username, password) VALUES ('" + username + "', '" + hashedPassword + "');";
- sqlite3_exec(db, sql.c_str(), nullptr, nullptr, nullptr);
- std::cout << "User registered successfully.\n";}

- // Login functionbool loginUser
- (sqlite3* db, const std::string& username, const std::string& password)
- {   std::string hashedPassword = hashPassword(password);   std::string sql = "SELECT * FROM users WHERE username = '" + username + "' AND password = '" + hashedPassword + "';";
- // Execute SQL and verify login (additional handling needed)}

- 2. Appointment Scheduling
- This module includes scheduling, rescheduling, and canceling appointments with basic SQLite database integration.
- #include <ctime>
- void scheduleAppointment(sqlite3* db, const std::string& user, const std::string& doctor, const std::string& datetime)
- {   std::string sql = "INSERT INTO appointments (user, doctor, datetime) VALUES ('" + user + "', '" + doctor + "', '" + datetime + "');";
- sqlite3_exec(db, sql.c_str(), nullptr, nullptr, nullptr);
- std::cout << "Appointment scheduled successfully.\n";}
- void cancelAppointment(sqlite3* db, const std::string& user, const std::string& datetime)
- {   std::string sql = "DELETE FROM appointments WHERE user = '" + user + "' AND datetime = '" + datetime + "';";
- sqlite3_exec(db, sql.c_str(), nullptr, nullptr, nullptr);   std::cout << "Appointment cancelled successfully.\n";}

- 3. Chat FunctionalityUsing the Boost.Asio library, we can set up basic asynchronous communication for a text-based chat between users and doctors.

```
• #include <boost/asio.hpp>
• #include <thread>          using boost::asio::ip::tcp;
• void startChatServer()
•  {    boost::asio::io_service io_service;    tcp::acceptor acceptor(io_service, tcp::endpoint(tcp::v4(), 1234));
• std::cout << "Chat server started on port 1234\n";    tcp::socket socket(io_service);    acceptor.accept(socket);

• std::string message = "Hello from server!";
•   boost::asio::write(socket, boost::asio::buffer(message));}
• void startChatClient()
• {    boost::asio::io_service io_service;    tcp::socket socket(io_service);    tcp::resolver resolver(io_service);
• boost::asio::connect(socket, resolver.resolve({"localhost", "1234"}));
•  char reply[1024];
•  size_t reply_length = boost::asio::read(socket, boost::asio::buffer(reply, message.size()));
•  std::cout << "Reply from server: ";    std::cout.write(reply, reply_length);
• std::cout << "\n";}
```

- 4. Database ManagementSQLite setup to store user information and appointment details.
- void setupDatabase(sqlite3* db)
- { std::string sql = R"(CREATE TABLE IF NOT EXISTS users (id INTEGER PRIMARY KEY AUTOINCREMENT,
- username TEXT NOT NULL UNIQUE,
- password TEXT NOT NULL); CREATE TABLE IF NOT EXISTS appointments
- (id INTEGER PRIMARY KEY AUTOINCREMENT,
- user TEXT NOT NULL,
- doctor TEXT NOT NULL,
  datetime TEXT NOT NULL);
)";

- sqlite3_exec(db, sql.c_str(), nullptr, nullptr, nullptr);
  std::cout << "Database setup complete.\n";
- Final result

ADVANTAGES OF THIS PROJECT

- **Improved Accessibility**
- **Remote Healthcare**: Patients can access healthcare services from the comfort of their homes, reducing travel time and costs.

- **Expanded Reach**: Doctors can offer services to patients in remote or underserved areas, enhancing healthcare access.

- **Enhanced Efficiency**
- **Streamlined Processes**: Automated appointment scheduling, reminders, and record management reduce administrative burdens.

- **Real-Time Consultations**: Patients can have immediate consultations without the need for in-person visits, leading to quicker diagnoses and treatments.

CONCLUSION

- The development of a telemedicine consultation platform using C++ represents a significant advancement in healthcare delivery. This approach leverages the strengths of C++,
- such as performance, efficiency, and security, to create a robust system that enhances access to medical services for patients and streamlines operations for healthcare providers.
- By providing remote access to healthcare, the platform not only improves patient convenience but also addresses the challenges of geographical barriers and resource constraints in traditional healthcare settings.
- The integration of real-time communication, secure data management, and user-friendly interfaces ensures that both patients and providers can engage effectively and safely.