OOP LAB

Project Report

SUBMITTED BY:

- **Name**: Muhammad Maaz Abbasi
- **Reg ID**: 242308
- **Class**: BSCYSev-F24-A

SUBMITTED TO:

Iram Fatima Hashimi

Intelligent Clinic Scheduling and Advisory System

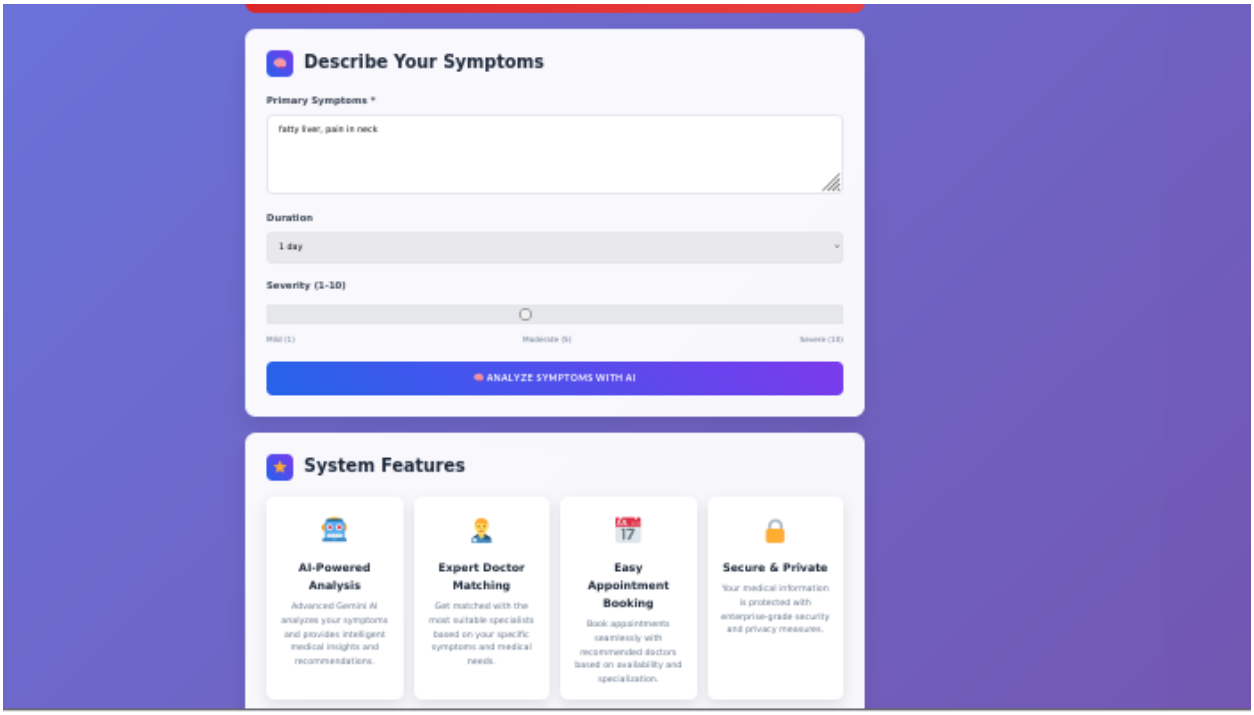
The Intelligent Clinic Scheduling and Advisory System is a modern healthcare web application that leverages advanced AI (Google Gemini) to provide users with instant, personalized medical symptom analysis and doctor recommendations. Built entirely in C++ for the backend and using pure HTML/CSS for the frontend, the system demonstrates robust object-oriented programming, secure data handling, and seamless AI integration.

Key Features

- **AI-Powered Symptom Analysis:**
  Users can submit their symptoms, duration, and severity through a web form. The backend sends this data to the Gemini AI API, which returns a detailed analysis including possible conditions, recommendations, warning signs, and suggested medical specialties.
- **Doctor Recommendation and Scheduling:**
  Based on the AI's suggested specialties, the system recommends the most suitable doctors that are available. Users can view doctor profiles and book appointments directly through the web interface.
- **Appointment Management:**
  The system allows users to book appointments with recommended doctors, capturing all necessary details and storing them securely in a text as no medical information is stored online.
- **Emergency Support:**
  Prominent emergency banners and warnings are displayed to guide users in urgent situations
- **Security & Privacy**
  For security reasons, no personal medical information is stored. We understand how sensitive and confidential health data is, which is why only basic appointment details — such as date, time, and booking status — are securely stored. Information related to symptoms, severity, or any personal medical records is never collected or saved, ensuring complete protection of users' medical privacy.

How It Works

1. **User Input:**
   The user visits the homepage and inputs their symptoms, duration, and severity.



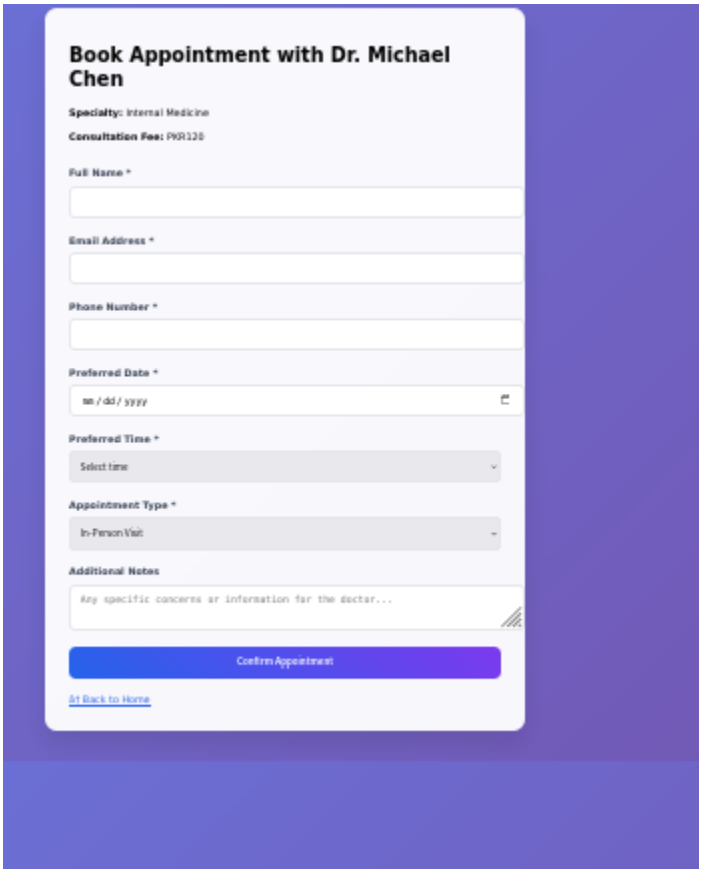
2. **AI Analysis:**
   The backend C++ server receives the form data and sends it to the Gemini AI API. The AI's response is parsed to extract the main analysis text, which is then formatted and displayed to the user in a readable HTML report. Both responses are shown, hardcoded and response from Gemini.

```cpp
std::unique_ptr<SymptomAnalysis> AIService::analyzeSymptoms(const std::string& symptoms,
                                                            const std::string& duration,
                                                            int severity) {
    auto analysis = std::make_unique<SymptomAnalysis>(symptoms, duration, severity);

    // Create Gemini API request payload
    std::ostringstream payload;
    payload << "{\n";
    payload << "  \"contents\": [{\n";
    payload << "    \"parts\": [{\n";
    payload << "      \"text\": \"As a medical AI assistant, analyze these symptoms:\\n\\n";
    payload << "Symptoms: " << symptoms << "\\n";
    payload << "Duration: " << (duration.empty() ? "Not specified" : duration) << "\\n";
    payload << "Severity (1-10): " << severity << "\\n\\n";
    payload << "Provide analysis with possible conditions, recommendations, warning signs, and suggested specialties.\"\n";
    payload << "    }]\n";
    payload << "  }]\n";
    payload << "}";
```

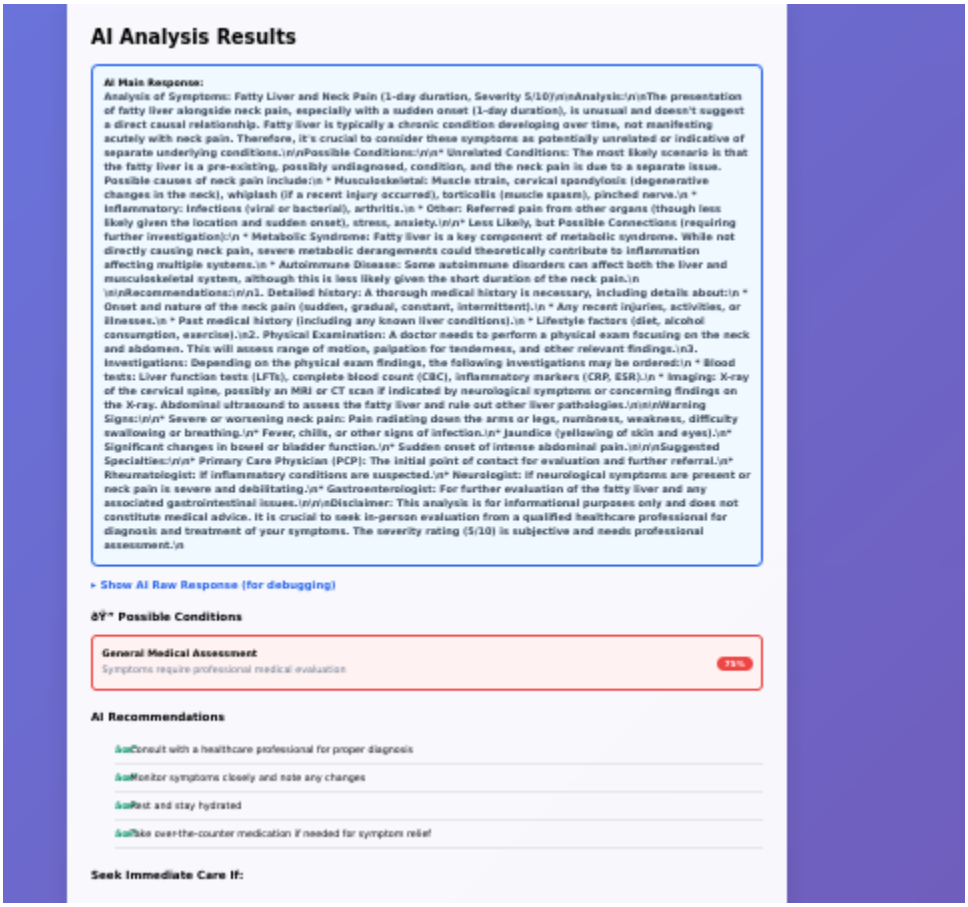
```cpp
// Parse response and extract medical insights
if (response.find("respiratory") != std::string::npos ||
    symptoms.find("cough") != std::string::npos ||
    symptoms.find("breathing") != std::string::npos) {
    analysis->addCondition("Respiratory Infection", "Possible viral or bacterial respiratory infection", 75);
    analysis->addSuggestedSpecialty("Pulmonology");
    analysis->addSuggestedSpecialty("Internal Medicine");
}

if (symptoms.find("headache") != std::string::npos ||
    symptoms.find("head") != std::string::npos) {
    analysis->addCondition("Tension Headache", "Common type of headache caused by stress or muscle tension", 80);
    analysis->addSuggestedSpecialty("Neurology");
    analysis->addSuggestedSpecialty("Family Medicine");
}

if (symptoms.find("chest") != std::string::npos ||
    symptoms.find("heart") != std::string::npos) {
    analysis->addCondition("Chest Discomfort", "Could be related to cardiac or respiratory issues", 70);
    analysis->addSuggestedSpecialty("Cardiology");
    analysis->addSuggestedSpecialty("Internal Medicine");
}

if (symptoms.find("fever") != std::string::npos ||
    symptoms.find("temperature") != std::string::npos) {
    analysis->addCondition("Viral Infection", "Common viral illness with fever symptoms", 85);
```

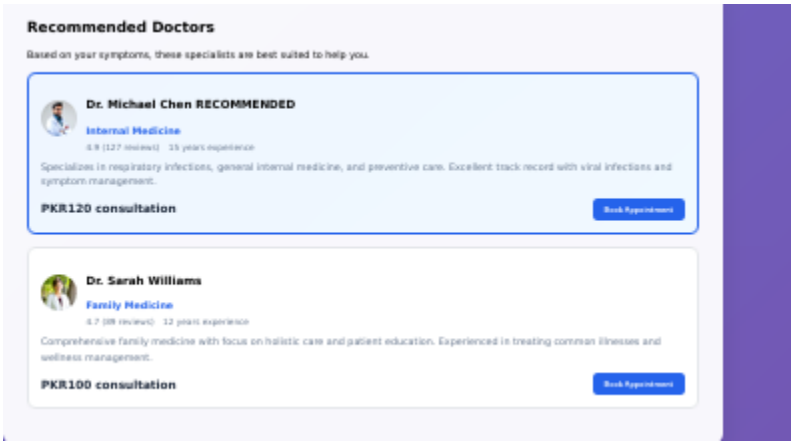

3. **Doctor Matching:**
   The system matches the AI's suggested specialties with its internal doctor database, recommending the top specialists for the user's case.

4. **Appointment Booking:**
   Users can book appointments with recommended doctors by filling out a secure form. Appointment details are saved for future reference in a txt file.



5. **Results Display:**
   The user receives a comprehensive report including:





- AI-generated analysis (with possible conditions, recommendations, and warning signs)
- Recommended doctors that
- Option to view the raw AI response for transparency/debugging

Technical Architecture

- **Backend:**
  - Written in C++ (object-oriented design)
  - Handles HTTP requests, form parsing, AI API integration, and HTML response generation
  - Uses libcurl for HTTP communication with Gemini AI
- **Frontend:**
  - Pure HTML and CSS
  - Responsive, modern design with medical-themed styling
- **AI Integration:**
  - Communicates with Google Gemini API for advanced medical analysis
  - Extracts and formats the AI's main response for user-friendly display
- **Security and Privacy:**
  - All sensitive operations are handled server-side
  - No client-side scripting or data exposure

Object-Oriented Design

- **Doctor Class:** Encapsulates doctor details and provides HTML rendering for profiles.
- **SymptomAnalysis Class:** Stores AI analysis results and generates HTML reports.
- **AIService Class:** Handles communication with the Gemini API and response parsing.
- **HttpServer Class:** Manages HTTP routing, form handling, and overall application flow.

Usage

1. Start the C++ server.
2. Access the system via a web browser at http://localhost:8080.
3. Submit symptoms and receive instant AI-powered analysis and doctor recommendations.

4. Book appointments as needed.

Novelty

The novelty of this **Intelligent Clinic System** project lies in its integration of advanced AI-driven symptom analysis with intelligent clinic scheduling and advisory features, all within a user-friendly web interface. Here are the key novel aspects:

---

1. AI-Powered Symptom Analysis

- Uses advanced AI (e.g., Gemini AI) to interpret user-described symptoms, duration, and severity, providing personalized medical insights and recommendations.

2. Intelligent Doctor Matching

- Matches patients with the most suitable specialists based on their specific symptoms and medical needs, improving the accuracy and efficiency of referrals.

3. Seamless Appointment Scheduling

- Integrates appointment booking directly after AI analysis and doctor matching, streamlining the patient journey from symptom input to consultation.

4. Emergency Awareness Integration

- Prominently features emergency guidance (e.g., call 1122, find nearest ER) to ensure user safety and responsible AI use.

5. Modern, Accessible Design

- Offers a visually appealing, mobile-responsive interface that lowers barriers for users of all ages and backgrounds.

6. Privacy and Security Emphasis

For security reasons, no personal medical information is stored. We understand how sensitive and confidential health data is, which is why only basic appointment details — such as date, time, and booking status — are securely stored. Information related to symptoms, severity, or any personal medical records is never collected or saved, ensuring complete protection of users' medical privacy.

Conclusion

This project demonstrates a secure, modern, and intelligent clinic advisory and scheduling system, combining the power of AI with robust C++ backend engineering and a clean, accessible frontend. It is suitable for educational, demonstration, or prototype healthcare applications.