📌 IPC Nexus Chatbot Documentation

Overview

The **IPC Nexus Chatbot** is designed to assist users in understanding legal concepts, navigating the website, and providing relevant legal information based on user queries. It has three primary user categories:

- **Police Officers**: For guided FIR filing using applicable laws.
- **Law Students**: For learning and practicing case laws.
- **General Public**: For understanding legal rights and gaining knowledge of legal procedures.

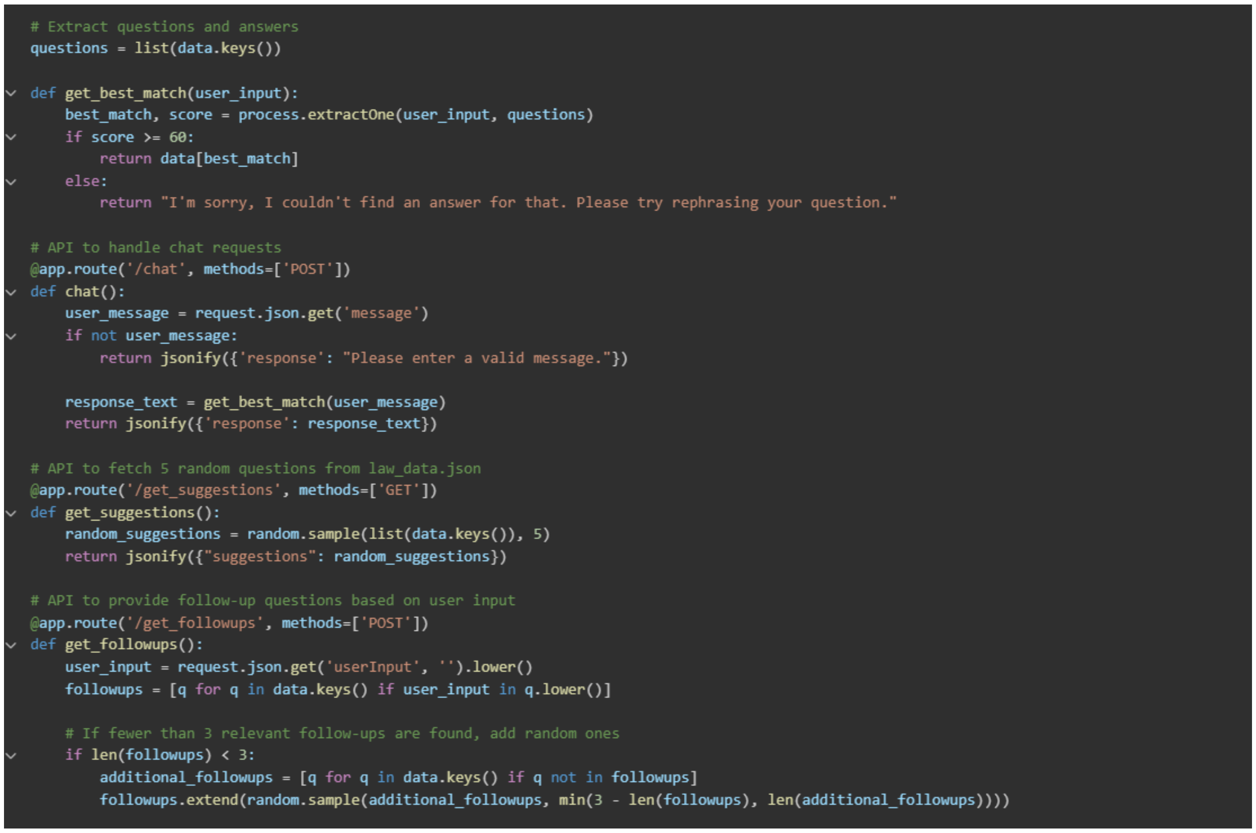
The chatbot uses a simple and effective rule-based system to match user queries using **fuzzy matching** for approximate responses.

📁 Files Overview

1. chatbot.py

This is the backend of the chatbot, built using **Flask**. It handles the API routes and processes user queries using data from the law_data.json file.
Key Functions:

- **Flask Setup**: Configures the Flask app and enables CORS to allow cross-origin requests.
- **Data Loading**: Reads data from law_data.json using json.load.
- **Fuzzy Matching**: Uses fuzzywuzzy for matching user inputs to relevant questions.
- **API Routes**:
    - /chat: Accepts user messages and provides appropriate responses.
    - /get_suggestions: Returns 5 random legal questions from the dataset.
    - /get_followups: Provides follow-up questions based on the user's query.



```python
# Extract questions and answers
questions = list(data.keys())

def get_best_match(user_input):
    best_match, score = process.extractOne(user_input, questions)
    if score >= 60:
        return data[best_match]
    else:
        return "I'm sorry, I couldn't find an answer for that. Please try rephrasing your question."

# API to handle chat requests
@app.route('/chat', methods=['POST'])
def chat():
    user_message = request.json.get('message')
    if not user_message:
        return jsonify({'response': "Please enter a valid message."})

    response_text = get_best_match(user_message)
    return jsonify({'response': response_text})

# API to fetch 5 random questions from law_data.json
@app.route('/get_suggestions', methods=['GET'])
def get_suggestions():
    random_suggestions = random.sample(list(data.keys()), 5)
    return jsonify({"suggestions": random_suggestions})

# API to provide follow-up questions based on user input
@app.route('/get_followups', methods=['POST'])
def get_followups():
    user_input = request.json.get('userInput', '').lower()
    followups = [q for q in data.keys() if user_input in q.lower()]

    # If fewer than 3 relevant follow-ups are found, add random ones
    if len(followups) < 3:
        additional_followups = [q for q in data.keys() if q not in followups]
        followups.extend(random.sample(additional_followups, min(3 - len(followups), len(additional_followups))))
```

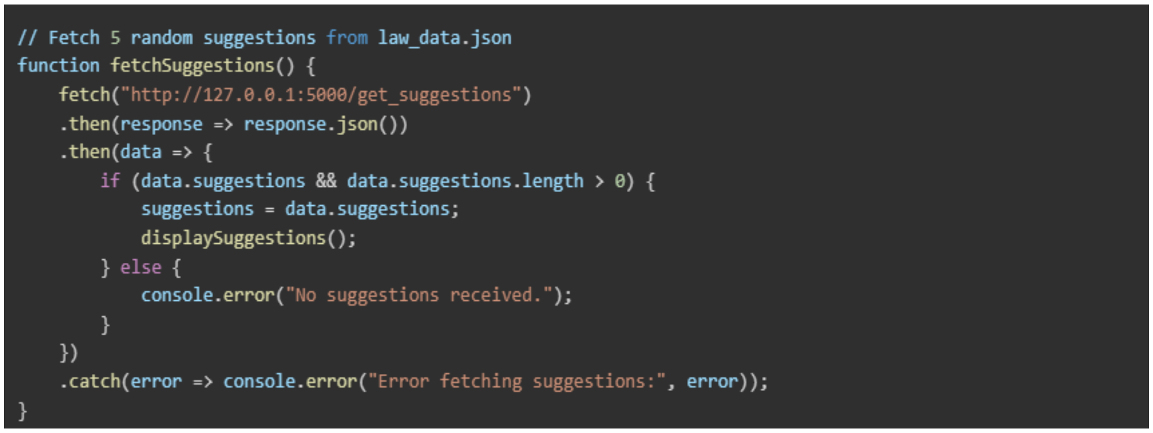
Fig 1: chatbot.py Image

2. index.html

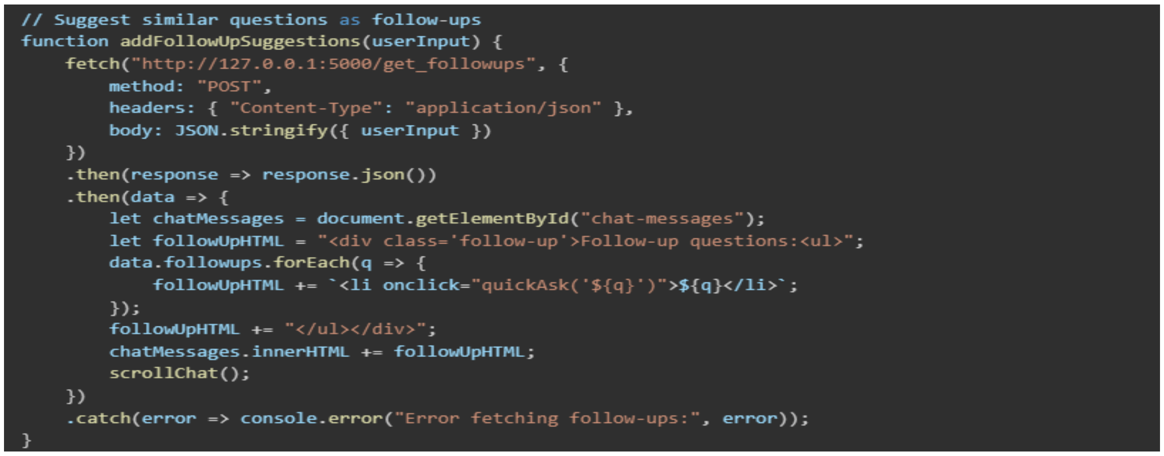
This is the frontend of the chatbot, designed with **HTML, CSS, and JavaScript**. It provides the chatbot interface and handles user interactions.
Key Features:

- **Chat Interface**: Clean design using light blue and soft UI elements.
- **Chat Bubbles**: Differentiates between user messages and bot responses.

- **Fetching & Follow-Up Suggestions**: Displays related questions for further assistance.



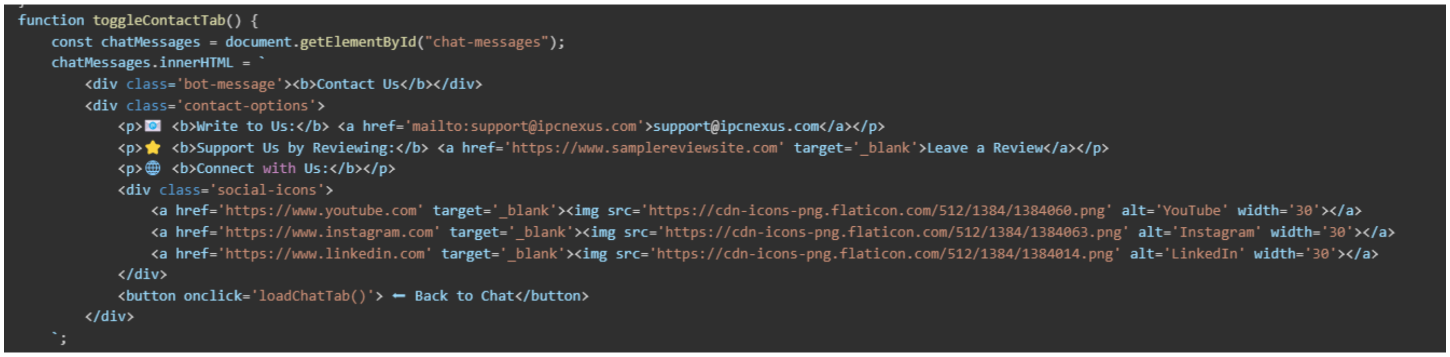
```javascript
// Fetch 5 random suggestions from law_data.json
function fetchSuggestions() {
    fetch("http://127.0.0.1:5000/get_suggestions")
    .then(response => response.json())
    .then(data => {
        if (data.suggestions && data.suggestions.length > 0) {
            suggestions = data.suggestions;
            displaySuggestions();
        } else {
            console.error("No suggestions received.");
        }
    })
    .catch(error => console.error("Error fetching suggestions:", error));
}
```



```javascript
// Suggest similar questions as follow-ups
function addFollowUpSuggestions(userInput) {
    fetch("http://127.0.0.1:5000/get_followups", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ userInput })
    })
    .then(response => response.json())
    .then(data => {
        let chatMessages = document.getElementById("chat-messages");
        let followUpHTML = "<div class='follow-up'>Follow-up questions:<ul>";
        data.followups.forEach(q => {
            followUpHTML += `<li onclick="quickAsk('${q}')">${q}</li>`;
        });
        followUpHTML += "</ul></div>";
        chatMessages.innerHTML += followUpHTML;
        scrollChat();
    })
    .catch(error => console.error("Error fetching follow-ups:", error));
}
```

Fig 2 and 3: Fetch and Follow-Up Code

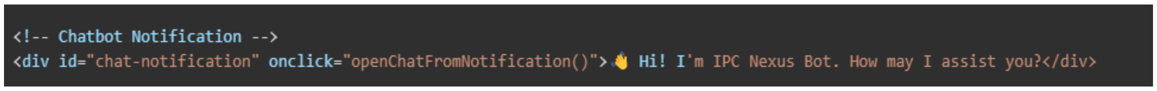
- **Contact Us Tab**: Provides contact options, including email, social media, and external links.



```javascript
function toggleContactTab() {
    const chatMessages = document.getElementById("chat-messages");
    chatMessages.innerHTML = `
        <div class='bot-message'><b>Contact Us</b></div>
        <div class='contact-options'>
            <p>📧 <b>Write to Us:</b> <a href='mailto:support@ipcnexus.com'>support@ipcnexus.com</a></p>
            <p>⭐ <b>Support Us by Reviewing:</b> <a href='https://www.samplereviewsite.com' target='_blank'>Leave a Review</a></p>
            <p>🌐 <b>Connect with Us:</b></p>
            <div class='social-icons'>
                <a href='https://www.youtube.com' target='_blank'><img src='https://cdn-icons-png.flaticon.com/512/1384/1384060.png' alt='YouTube' width='30'></a>
                <a href='https://www.instagram.com' target='_blank'><img src='https://cdn-icons-png.flaticon.com/512/1384/1384063.png' alt='Instagram' width='30'></a>
                <a href='https://www.linkedin.com' target='_blank'><img src='https://cdn-icons-png.flaticon.com/512/1384/1384014.png' alt='LinkedIn' width='30'></a>
            </div>
            <button onclick='loadChatTab()'> ⬅ Back to Chat</button>
        </div>
    `;
```

Fig 4: Contact Us Code Segment

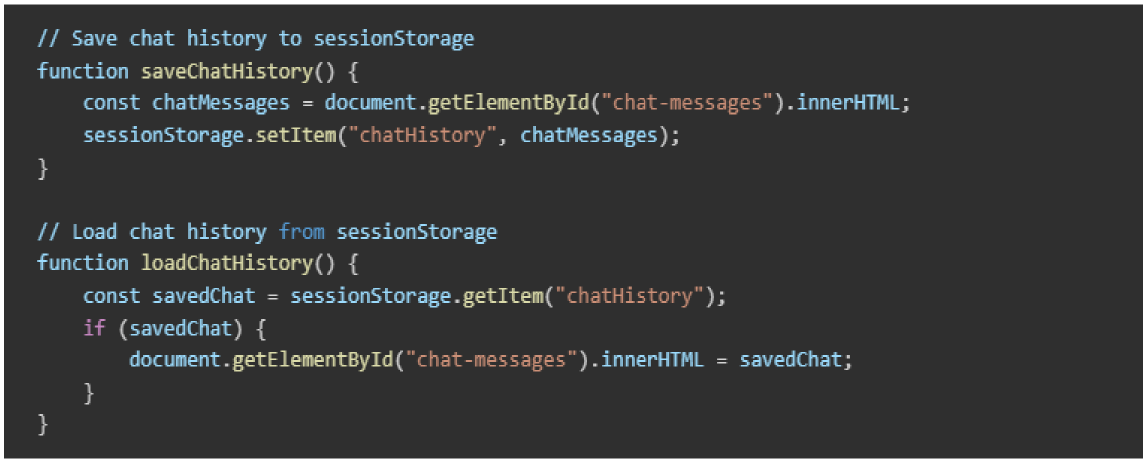
- **Notification System**: Displays a greeting notification when the chatbot is loaded.



```html
<!-- Chatbot Notification -->
<div id="chat-notification" onclick="openChatFromNotification()">👋 Hi! I'm IPC Nexus Bot. How may I assist you?</div>
```

Fig 5: Chatbot greeting Notification

- **Session Management**: Uses session Storage to maintain chat history during a session.

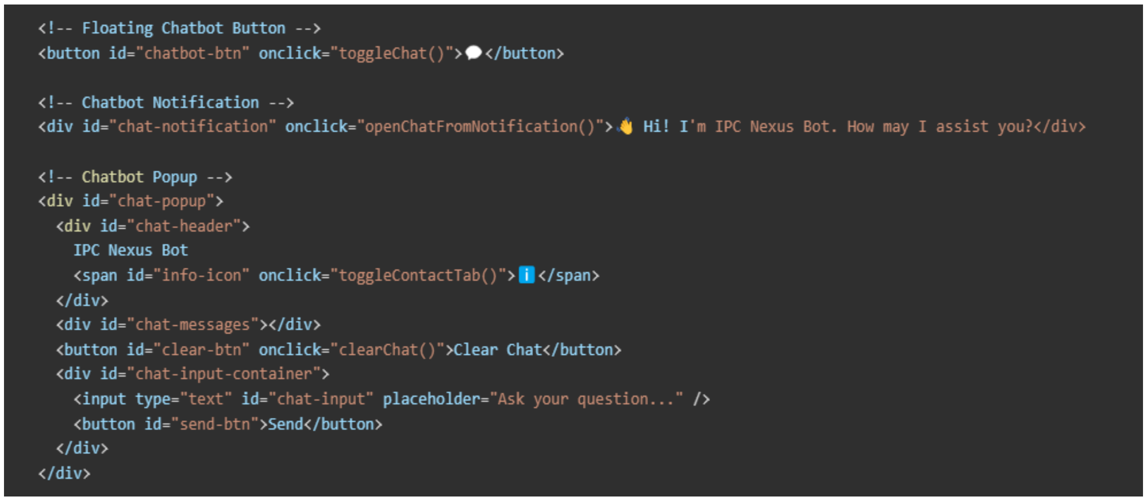


```javascript
// Save chat history to sessionStorage
function saveChatHistory() {
    const chatMessages = document.getElementById("chat-messages").innerHTML;
    sessionStorage.setItem("chatHistory", chatMessages);
}

// Load chat history from sessionStorage
function loadChatHistory() {
    const savedChat = sessionStorage.getItem("chatHistory");
    if (savedChat) {
        document.getElementById("chat-messages").innerHTML = savedChat;
    }
}
```

Fig 6: Chat Session Storage

- **Chat Bot Button and Pop-up Section:**



```html
<!-- Floating Chatbot Button -->
<button id="chatbot-btn" onclick="toggleChat()">💬</button>

<!-- Chatbot Notification -->
<div id="chat-notification" onclick="openChatFromNotification()">👋 Hi! I'm IPC Nexus Bot. How may I assist you?</div>

<!-- Chatbot Popup -->
<div id="chat-popup">
  <div id="chat-header">
    IPC Nexus Bot
    <span id="info-icon" onclick="toggleContactTab()">ℹ️</span>
  </div>
  <div id="chat-messages"></div>
  <button id="clear-btn" onclick="clearChat()">Clear Chat</button>
  <div id="chat-input-container">
    <input type="text" id="chat-input" placeholder="Ask your question..." />
    <button id="send-btn">Send</button>
  </div>
</div>
```

Fig 7: Chatbot Button and Pop-up command html code

3. law_data.json

This file contains a collection of legal questions and their respective responses. It serves as the chatbot's knowledge base.
Key Features:

- **Navigation Support**: Provides users with section-based suggestions to help them find resources.
- **Law Sections**: Categorized information for quick access to topics like theft, domestic violence, and missing person reports.

🚀 How to Run the Chatbot

1. **Ensure Python and Dependencies Are Installed**
   - Install Python and Pip.
   - Install required packages using:

```
pip install flask flask-cors fuzzywuzzy
```

2. **Run the Backend**
   - Navigate to the directory containing chatbot.py.
   - Run: python chatbot.py
   - The server will start at http://127.0.0.1:5000.
3. **Launch the Frontend**
   - Open index.html in your browser.
4. **Interact with the Chatbot**
   - Click the chatbot icon to start the conversation and ask your queries.

🛠 Additional Notes

- Ensure law_data.json is in the same directory as chatbot.py.
- Keep the chatbot server running for API responses to work.
- Use a browser that supports JavaScript for the chatbot interface.