**Legal AI Assistant — Full Project Pseudocode**

**1. Webpage Frontend (HTML + JavaScript)**

* ON PAGE LOAD:
  + Render chat interface (sidebar, chat area, input box).
  + Connect to WebSocket server at ws://localhost:8080/ws.
* USER INTERACTION LOOP:
  + Wait for user to type a message.
  + Enable "Send" button when input is not empty.
  + On "Send" button click or Enter key:
  + Display user's message in chat window.
  + Send message to WebSocket server.
  + Show "Legal AI Assistant is typing..." indicator.
* WEBSOCKET EVENT HANDLERS:
  + On connection open:
  + Display "Connected to chat server" system message.
  + On message received:
  + Hide typing indicator.
  + Parse message type:
  + If 'bot\_message': Display bot's response(s) in chat window, with formatting for legal info.
  + If 'user\_message': Display user message (echo).
  + If 'error': Display error message.
  + Otherwise: Display raw message.
  + On error:
  + Display "Connection error occurred" system message.
  + On close:
  + Display "Connection closed" system message.
* MESSAGE FORMATTING:
  + For legal info responses, format sections (e.g., headings, descriptions, IPC sections) with HTML.
  + For other messages, escape HTML and apply basic formatting (bold, italics, lists).

**2. WebSocket Server (Backend Bridge)**

* ON CLIENT CONNECTION (ws://localhost:8080/ws):
  + Accept WebSocket connection from client.
* ON MESSAGE FROM CLIENT:
  + Receive user message.
  + (Optional) Log or echo user message back as 'user\_message'.
  + Pass user message to NLP backend (Rasa or custom Python logic).
  + Wait for response(s) from backend:
  + If response is legal info, format as 'bot\_message'.
  + If error, format as 'error'.
  + Send response(s) to client via WebSocket.
* ON CLIENT DISCONNECT:
  + Clean up any session or resources if needed.

**3. NLP Backend (Rasa)**

* ON RECEIVING MESSAGE FROM WEBSOCKET SERVER:
  + Parse user message using NLU (Natural Language Understanding).
  + Determine user intent (e.g., ask about crime, IPC section, punishment, etc.).
  + Extract entities (e.g., crime name, section number).
* IF intent requires database info:
  + Trigger appropriate custom action (see below).
* ON RESPONSE:
  + Format response as plain text or structured legal info.
  + Return response to WebSocket server for delivery to frontend.

**4. Custom Actions (Database Query Logic)**

**a. Query Crime Information**

* ON action\_query\_crime:
  + Extract crime name from user message/entities.
  + If not found, check for cyber crime keywords.
  + If still not found, prompt user for clarification.
  + Connect to MySQL database.
  + Query 'crimes' table for crime details.
  + If not found, inform user.
  + Query 'crime\_ipc\_mapping' for related IPC sections.
  + For each IPC section, get details from 'ipc\_sections' table.
  + Format response:
  + Crime description, severity, category, bailable, cognizable, compoundable.
  + List of related IPC sections (number, title, description, punishment).
  + Return formatted response to NLP backend.

**b. Query IPC Section Information**

* ON action\_query\_ipc\_section:
  + Extract IPC section number from user message/entities.
  + If not found, prompt user for clarification.
  + Connect to MySQL database.
  + Query 'ipc\_sections' table for section details.
  + If not found, inform user.
  + Query 'crime\_ipc\_mapping' for related crimes.
  + Format response:
  + Section title, description, punishment.
  + List of related crimes (name, description, severity).
  + Return formatted response to NLP backend.

**5. Database Layer**

* DATABASE TABLES:
  + crimes: id, name, description, severity, category, bailable, cognizable, compoundable, etc.
  + ipc\_sections: id, section\_number, title, description, punishment, etc.
  + crime\_ipc\_mapping: crime\_id, ipc\_section\_id
* DATABASE OPERATIONS:
  + SELECT, JOIN, and FILTER as needed for queries above.
  + Handle connection errors and close connections after use.

**6. Data Flow Summary**

User (Webpage)

⇅ (WebSocket)

WebSocket Server (ws://localhost:8080/ws)

⇅ (HTTP API or direct call)

NLP Backend (Rasa or custom Python)

⇅ (MySQL)

Database (crimes, ipc\_sections, mapping)

**7. Example End-to-End Flow**

1. User types: "Tell me about Section 302"

2. Frontend sends: "Tell me about Section 302" via WebSocket

3. WebSocket server receives, forwards to NLP backend

4. NLP backend parses intent, triggers action\_query\_ipc\_section

5. Action queries MySQL for section 302, gets details and related crimes

6. Action formats response, returns to NLP backend

7. NLP backend sends response to WebSocket server

8. WebSocket server sends response to frontend

9. Frontend displays formatted legal info in chat window

**8. Additional Features (Optional/Advanced)**

- User authentication (login.html)

- Logging and analytics (chat.log)

- Error handling and user feedback

- Admin tools for updating database (populate\_database.py, update scripts)

- Rasa training and model management (train\_rasa\_model.py, domain.yml, etc.)

**9. Notes**

* The **frontend** is decoupled from backend logic; it only expects JSON messages over WebSocket.
* The **backend** (WebSocket server + NLP) is responsible for all business logic, database access, and response formatting.
* The **database** is the source of all legal information.
* The **WebSocket server** acts as a bridge between the web client and the NLP backend.