System Architecture

1. Frontend Interface Layer (Client-Side UI)

This layer handles all user interactions and is optimized for responsiveness, readability, and real-time feedback.

Key Components

- **HTML/CSS/JavaScript Interface**
  - Built with a minimalistic, Gemini-inspired black-and-white theme.
  - Utilizes CSS variables and lightweight animations for smooth user experience.
- **Streaming Response Viewer**
  - Implements the browser's ReadableStream API to process incoming text chunks.
  - Displays responses incrementally, enabling real-time rendering while the model generates output.
- **Input Handler**
  - Captures user messages.
  - Sends JSON payloads to the backend using fetch() POST requests.
  - Manages prompts, quick actions, and shortcut triggers.

Responsibilities

- Provide a clean, intuitive chat environment.
- Handle dynamic streaming text output.
- Ensure cross-device compatibility (desktop/mobile).
- Maintain low-latency communication with the API layer.

2. API & Application Layer (FastAPI Server)

This is the core orchestration layer responsible for routing, streaming, state handling, and secure communication between frontend and model.

Key Components

- **FastAPI Application (test_server.py)**
    - Acts as the main server entry point.
    - Hosts the root route (serving index.html) and the /api/v1/chat endpoint.
- **Routing Module (chatbot_module/routes.py)**
    - Defines all REST API routes.
    - Validates input through Pydantic models.
    - Initiates streaming responses using StreamingResponse.
- **Request/Response Pipeline**
    - Each user request is converted into a generator-based YOLO pipeline that streams data to the client.
    - Supports custom model parameters (model selection, prompt injection, etc.).
- **Middleware**
    - Handles error management, request logging, CORS policies, and rate limiting (if enabled).
- **Launcher Scripts**
    - launch.py, RUN.bat, and start.py streamline startup, dependency checks, and automated browser opening.

Responsibilities

- Accept user requests and validate input.
- Stream model-generated text chunks to the client.
- Manage asynchronous communication with the Ollama service.
- Provide a secure, controlled API surface for external developers.

3. AI Service Layer (Business Logic & Model Controller)

This layer acts as an intelligent mediator between the API layer and the local inference engine.

Key Components

- **Ollama Service Wrapper (chatbot_module/service.py)**
    - Encapsulates all communication with the Ollama engine.
    - Initializes the model (default: llama3, configurable via constructor or API).

    - Implements an asynchronous generator that yields incremental text outputs.
- **Prompt Management Module (chatbot_module/prompts.py)**
    - Contains the **SYSTEM PROMPT**, **behavior rules**, and **context management logic**.
    - Supports easy customization and domain-specific fine-tuning of conversational style.
- **Error Handling Layer**
    - Captures and returns structured errors for:
        - Missing model
        - Inference failure
        - Unavailable Ollama service
        - Empty inputs or malformed requests

Responsibilities

- Create and manage requests to the LLM.
- Inject system-level rules, safety guidelines, and conversational structure.
- Convert model output streams into API-ready generators.
- Provide abstraction so backend routes never directly touch the LLM.

4. Local Inference Engine Layer (Ollama Host)

This layer provides local, secure execution of Large Language Models using the Ollama runtime.

Key Components

- **Ollama Daemon (ollama serve)**
    - Hosts the model locally; listens for inference requests.
    - Manages model loading, GPU/CPU allocation, and internal optimization.
- **Model Repository**
    - Supports installation of various models via:
    - ollama pull llama3
    - ollama pull mistral
    - ollama pull codellama

- **Inference Engine**
    - Runs token-by-token generation locally.
    - Streams partial outputs back to the service layer in near real-time.
- **Hardware Utilization**
    - Optimized for both CPU and GPU setups depending on OS and configuration.
    - Allows full offline operation with zero external API dependency.

Responsibilities

- Provide secure, offline LLM inference.
- Execute language model forward passes and token generation.
- Deliver streaming token outputs directly to the service layer.
- Enable users to run multiple models interchangeably.

5. Internal Data Flow (End-to-End Workflow)

1. **User Inputs Message** → UI captures text & sends request to /api/v1/chat.
2. **FastAPI Route Receives Request** → Validates content & initializes a streaming generator.
3. **OllamaService Creates Model Session** → Sends prompt to the selected LLM through Ollama's runtime.
4. **Ollama Engine Generates Tokens** → Streams partial responses back through the service layer.
5. **StreamingResponse Sends Chunks to UI** → Browser receives & renders text incrementally.
6. **UI Displays Real-Time Response** → Final output is shown with smooth transitions and formatting.

6. Architectural Advantages

- **Fully Local Operation** – Zero cloud dependency; complete privacy.
- **Real-Time Streaming** – Near-instant response delivery using asynchronous pipelines.
- **Modular & Extensible** – Easy to add models, new routes, or UI features.

- **Low Latency** – FastAPI + Ollama combination ensures efficient, lightweight execution.
- **Production-Ready** – Clean layering enables scaling, debugging, and deployment via Docker/systemd.