Tech Stack

1. Languages & Runtimes

- **Python 3.10 – 3.11 (recommended)**
  Rationale: modern language features, stable async support, and wide package compatibility.
- **Node.js 18+ (optional, for frontend build tooling)**
  Rationale: If you add build tooling (npm, Vite, Tailwind CLI), Node 18+ is stable and widely supported.

2. Backend — API & Orchestration

- **Framework:** FastAPI
  - Responsibilities: HTTP endpoints, streaming responses, request validation, dependency injection.
  - Key packages: fastapi, pydantic, uvicorn[standard].
  - Async design: Use async endpoints and asyncio to stream text chunks without blocking.
- **ASGI Server:** Uvicorn (or Gunicorn + Uvicorn workers for production)
  - Command (dev): uvicorn test_server:app --reload --host 0.0.0.0 --port 8000
  - Production: gunicorn -k uvicorn.workers.UvicornWorker -w 4 test_server:app
- **HTTP/Web:**
  - Streaming: fastapi.responses.StreamingResponse or SSE depending on client (recommend chunked text/plain or server-sent events if you want event types).
- **Request validation & schemas:** pydantic models for request/response schemas.

3. Model Layer & Inference

- **Inference Provider:** Ollama (local LLM host)
  - Responsibilities: model hosting, local inference, model pulls/management.

    - Recommended models: llama3 (primary), optionally llama3.2, mistral, codellama.
    - Interaction method: call Ollama via CLI (ollama run) or HTTP client (if offering an API), preferably using a persistent client connection to avoid spawn overhead.
- **Model Integration Pattern:**
    - Create OllamaService wrapper in chatbot_module/service.py:
        - Handles model selection, prompt injection, streaming consumption, error handling, and timeouts.
        - Expose async generator that yields text chunks for the API streaming layer.
- **Prompt Management:** chatbot_module/prompts.py
    - Store system prompts, safety wrappers, and dynamic prompt composition logic.
- **Model resource controls:** Per-request timeouts, token limits, and concurrency limits to avoid OOM or saturation.

4. Frontend — UI & UX

- **Core stack:** Plain HTML/CSS/JS (as in README) or lightweight framework (React/Vite) if expanding.
    - Recommendation for production growth: use a minimal React app scaffolded with Vite for better componentization.
- **Design & Styling:** Gemini-inspired minimal theme (black & white)
    - Use CSS variables for theming; consider TailwindCSS for rapid styling if you adopt a framework.
- **Streaming handling:**
    - Browser-side fetch() + response.body.getReader() + TextDecoder to process streaming chunks with smooth incremental rendering.
- **Accessibility:** keyboard focus, screen-reader labels, high-contrast toggle if you add themes.

5. Dev Tools, Testing & Local Workflow

- **Dependency management:** pip + requirements.txt OR poetry for deterministic installs.
    - Recommended requirements.txt baseline:
    - fastapi>=0.95
    - uvicorn[standard]>=0.22
    - requests>=2.31
    - ollama>=0.1 # placeholder; use actual package name/version if available
    - python-dotenv>=1.0
    - pytest>=8.0
- **Testing:** pytest + httpx (async client) for endpoint & streaming tests.
    - Add verify_streaming.py to assert chunk arrival and ordering.
- **Linting/Formatting:** black, ruff, isort.
- **Local env:** .env with python-dotenv loader.

6. Deployment & Packaging

- **Containerization:** Docker
    - Dockerfile base: python:3.11-slim
    - Expose port 8000, install dependencies, run with uvicorn.
- **Process manager:** systemd (for Linux servers) or Docker Compose for local multi-service setups (e.g., if you containerize Ollama separately).
- **Reverse proxy (optional):** Nginx or Traefik for TLS termination and static file serving.
- **Example Docker run:**
  ```
  docker run -p 8000:8000 --env-file .env insight-sphere-ai
  ```
- **CI/CD:** GitHub Actions to run tests and build Docker images.

7. Security, Privacy & Moderation

- **Data handling:** All inference on-device — ensure logs do not persist PII unless explicit consent.
- **Transport security:** Use HTTPS in production (via reverse proxy).

- **Input moderation:** optional in-process filters or a lightweight rule-based filter before sending user text to model.
- **Rate-limiting & auth:**
    - Use API keys or session auth for multi-user setups.
    - Rate-limit per-client to avoid abuse (use slowapi/Redis for distributed rate limits).

8. Observability & Logging

- **Logging:** Structured logs (uvicorn + structlog or logging with JSON formatter).
- **Metrics:** Prometheus-compatible metrics via prometheus_client for:
    - Request counts, latency, active streaming connections, inference duration, model errors.
- **Tracing (optional):** OpenTelemetry for end-to-end traces across the service and model calls.

9. Hardware & Resource Guidance

- **Minimum (development):** 8GB RAM, 4 CPU cores — suitable for smaller models / testing.
- **Recommended (local production / Llama3):** 16–32GB RAM and a modern CPU; GPU not required for Ollama's CPU inference but improves latency if supported by model and Ollama installation.
- **Storage:** Keep model files on fast NVMe if available: 50+ GB free for model variants.

10. Example Config / Environment Variables

```
APP_HOST=127.0.0.1

APP_PORT=8000

OLLAMA_BASE_URL=http://127.0.0.1:11434

OLLAMA_MODEL=llama3

LOG_LEVEL=info

MAX_TOKEN_LIMIT=2048

REQUEST_TIMEOUT_SECONDS=60
```

```
ENABLE_MODERATION=true
```

11. Recommended Project Layout (quick)

```
Ai chatbot code/
├── chatbot_module/
│   ├── __init__.py
│   ├── prompts.py          # system & domain prompts
│   ├── service.py          # OllamaService + streaming generator
│   ├── routes.py           # FastAPI endpoints & schemas
│   └── utils.py            # helpers, moderation, rate-limit hooks
├── tests/
│   └── test_streaming.py
├── index.html
├── test_server.py
├── launch.py
├── Dockerfile
├── requirements.txt
└── .env
```

12. Quick Implementation Checklist (next actions)

1. Create OllamaService async generator that yields chunks (place in service.py).
2. Implement streaming endpoint using StreamingResponse in routes.py.
3. Add .env loader and config class.
4. Add verify_streaming.py & pytest tests for chunk order + latency assertions.
5. Add Dockerfile and a minimal systemd unit for non-container deployments.