# Chatbot Prototype – Technical Design Summary

## 1. Assumptions & Simplifications

* - The chatbot is a simplified proof-of-concept (POC) intended for demonstration purposes only, not production deployment.
* - It is hosted locally and intended to be walked through live by the developer; no external users or environments are involved.
* - SQLite was selected for its speed, zero-setup cost, and suitability for small-scale development.
* - User identification is limited to phone number and name, with no authentication or role-based access control.
* - Fraud prevention, monitoring, alerting, and structured error handling were excluded due to the low-risk, single-user nature of the demo.
* - GPT-4 was integrated to handle unseen queries under the assumption that API usage would remain within reasonable limits.
* - No caching, session state, or chat memory has been implemented; all interactions are stateless.
* - Security, cost estimation, and infrastructure reliability were not prioritized, given the prototype’s scope.
* - Non-functional enhancements (e.g., GPT-4 integration) were added to align the app with Cayu.ai’s GenAI focus.

## 2. Design Decisions & Technical Approach

* - Used a unified Next.js codebase to handle both frontend (React) and backend (API routes).
* - Prisma ORM was integrated with SQLite to enable clean schema management and rapid local development.
* - Users are onboarded via phone number and name; returning users are greeted by name, new users are prompted to provide their name.
* - Known queries are handled using hardcoded logic for deterministic responses.
* - Unrecognized questions are routed to GPT-4 via OpenAI’s API, with a system prompt tailored to Cayu.ai’s domain.
* - The app maintains no conversational memory; each interaction is handled independently.
* - The architecture is designed to be extensible toward RAG-based enhancements in the future.

## 3. System Workflow

1. User accesses the chatbot UI and enters their phone number.
2. If the number is recognized, the user is greeted using their stored name.
3. If not, the user is prompted to enter their name, and the information is saved.
4. The user asks a question.
5. The backend checks for a match in predefined hardcoded queries.
6. If matched, a pre-written response is returned.
7. If unmatched, the query is sent to GPT-4 with a Cayu-specific system prompt.
8. GPT-4’s response is relayed to the frontend.
9. All processing happens locally, with no external infrastructure.

## 4. Trade-offs & Rationale

* - GPT-4 provides rich, relevant responses, but incurs cost and latency.
* - Next.js enabled rapid development with a single stack, at the cost of backend flexibility.
* - Prisma accelerated database setup but has a learning curve due to its domain-specific schema language.
* - SQLite is ideal for local testing, but lacks scalability, concurrent writes, and cloud-readiness.
* - Skipping cloud deployment saved time and money, but reduced app accessibility.
* - Forgoing security and rate-limiting allowed faster prototyping, but would not be viable in a production context.
* - Absence of monitoring and logging simplifies the build but limits visibility into system behavior.
* - Hardcoded responses are fast but non-scalable; GPT-4 fallback adds generality at higher cost.
* - Lack of chat memory avoids complexity but limits user experience continuity.