SPEC-1-ChatGPT-Clone

Background

The goal is to create a ChatGPT-like conversational AI web application. The system should provide an intuitive chat interface for users, backed by a Laravel-based backend that manages authentication, session management, and API communication with an LLM provider. The frontend will be developed in Next.js to provide a modern, responsive, and real-time chat experience.

The product targets developers, businesses, and hobbyists who want a customizable ChatGPT-like tool with extendable backend features (user management, logging, usage tracking, etc.) and a production-ready frontend interface.

Requirements

Must Have

- User authentication and account management (login, signup, password reset).
- Real-time chat interface with AI responses.
- API integration with an LLM provider (e.g., OpenAI, Anthropic, or an open-source model).
- Session management (conversations stored per user).
- Basic usage tracking (messages sent, tokens used, timestamps).
- Mobile-responsive frontend UI (Next.js).
- Secure backend built with Laravel (API endpoints, validation, rate limiting).

Should Have

- Role-based access control (e.g., admin, premium user).
- Subscription/payment integration for premium access.
- Admin dashboard (user stats, usage logs, system health).
- Support for multiple AI models (selectable per chat).

Could Have

- File upload support (e.g., PDFs, images) for context-based chat.
- Team workspaces (shared conversations).
- Plugin/add-on system for extending capabilities.
- Multilingual UI support.

Won’t Have (for MVP)

- Voice-based interaction.
- Offline mode.
- Native mobile app (web app only for MVP).

Method

Architecture Overview
- Frontend (Next.js): Provides a responsive chat UI with real-time updates (via WebSockets or SSE).
- Backend (Laravel): Handles authentication, conversation storage, rate limiting, usage logging, and external API calls.
- Database (MySQL/Postgres): Stores users, sessions, and conversation history.
- LLM Provider: External API (e.g., OpenAI GPT-4 or Anthropic Claude).

PlantUML - High-level Architecture:
@startuml
actor User
User --> Frontend: Sends messages
Frontend --> Backend: API requests
Backend --> Database: Store/Retrieve chats
Backend --> LLM: Forward prompt
LLM --> Backend: AI Response
Backend --> Frontend: Return response
Frontend --> User: Display message
@enduml

Database Schema (MVP)
- users (id, name, email, password, role, created_at).
- conversations (id, user_id, title, created_at).
- messages (id, conversation_id, sender_type [user/ai], content, tokens_used, created_at).
- subscriptions (id, user_id, plan, status, created_at).

Key Libraries/Tools
- Laravel Sanctum/JWT for authentication.
- Laravel WebSockets or Pusher for real-time messaging.
- Next.js with Tailwind CSS for frontend UI.
- Prisma (optional) for frontend DB queries (if direct).

Implementation

1. Backend Setup (Laravel): Create authentication endpoints, build chat session + message storage, implement API connector for LLM, add rate limiting and logging.
2. Frontend Setup (Next.js): Build chat UI (input, message list, streaming responses), implement authentication (JWT/Session), connect WebSocket/SSE for real-time responses.
3. Database Setup: Define schema (users, conversations, messages, subscriptions), migrate tables.
4. Integration: Connect frontend → backend → LLM, test end-to-end chat flow.

Milestones

- Week 1–2: Backend setup (Laravel, authentication, DB schema).
- Week 3: Frontend setup (Next.js, chat UI).
- Week 4: Real-time messaging + LLM integration.
- Week 5: Admin dashboard & usage tracking.
- Week 6: Testing, bug fixes, and deployment.

Gathering Results

- Verify Must Have requirements via acceptance testing.
- Collect metrics: average response time, token usage, error rates.
- Gather user feedback for UI/UX improvements.
- Ensure production readiness with monitoring + logging.