Project 1: Basic LLM Chatbot

Objective:

Build a simple web chatbot where users can send prompts and receive responses from an LLM. This accelerated timeline focuses on core functionality while establishing essential development patterns. You will practice:

- API Integration: Calling external LLM APIs from Python
- Backend Development: Creating REST endpoints with FastAPI/Flask
- Frontend Development: Building a simple chat interface
- Environment Management: Secure API key handling

Core Requirements (Must-have)

Layer	Requirement
Backend	➢ Python 3.11 + FastAPI/Flask ➢ Expose POST /chat endpoint that accepts {"message": "<user text>"} and returns {"reply": "<llm response>"} ➢ Integrate with OpenAI GPT-4 or Google Gemini API ➢ Load API key from .env file (never commit secrets) ➢ Basic error handling for API failures
Frontend	➢ Streamlit - Python-only web framework ➢ Use st.chat_input() for user messages ➢ Use st.chat_message() to display conversation ➢ Add st.spinner() for loading states Simple, clean interface focused on functionality
Setup & Docs	➢ README.md with clear setup instructions ➢ requirements.txt with all dependencies ➢ .env.example showing required environment variables Basic project structure

Development Approach: Milestone-Based Progression

Focus on deliverable quality and comprehensive review compliance with rigid timelines.

Milestone 1: Foundation Setup & API Integration

Deliverables:

- Working development environment with proper project structure
- Basic LLM API integration (OpenAI/Gemini) with error handling
- Environment variable management with security compliance

- Initial Streamlit chat interface structure
- Git repository setup with proper .gitignore and README foundation

Review Requirements:

- Security Review: API keys properly secured, no secrets in git history
- Code Quality Review: Basic code organization and naming conventions
- AI Integration Review: Proper API integration patterns

Milestone 2: Core Chat Functionality

Deliverables:

- Complete chat interface with message history
- Comprehensive error handling for all API failure scenarios
- User experience enhancements (loading states, feedback)
- Session state management for conversation continuity
- Basic input validation and sanitization

Review Requirements:

- Performance Review: Response times and user experience optimization
- Security Review: Input validation and sanitization compliance
- Code Quality Review: Clean separation of concerns and maintainable code

Milestone 3: Application Readiness & Documentation

Deliverables:

- Comprehensive documentation (README, setup instructions, API documentation)
- Error handling and user feedback systems
- Code documentation and comments
- Testing and validation procedures
- Working demo with deployment preparation

Review Requirements:

- Architecture Review: Overall system design and structure assessment
- Code Quality Review: Final code quality and documentation standards
- AI Integration Review: Production-ready AI service integration
- Security Review: Complete security assessment and vulnerability scan

Stretch Goals (Nice-to-have)

- Chat History: Store conversation history in memory or simple JSON file
- Message Types: Support for different message types (user/assistant styling)
- Deployment: Deploy to Render, Vercel, or similar platform
- Enhanced UI: Add markdown rendering for code blocks and formatting

Technical Specifications

Quick Start Resources

- OpenAI API: https://platform.openai.com/docs/api-reference
- Google Gemini API: https://ai.google.dev/gemini-api/docs
- FastAPI Tutorial: https://fastapi.tiangolo.com/tutorial/
- Flask Quickstart: https://flask.palletsprojects.com/quickstart

Primary Objectives

- AI Integration Excellence | Security Compliance
- Code Quality Standards | Architecture Soundness | Performance Standards

Performance Standards

- API Response Time: Average < 3 seconds acceptance limit
- User Experience: Loading indicators, graceful error handling
- Resource Usage: Efficient memory and processing patterns

Success Criteria Checklist

- User can type a message and receive an LLM response
- API key is loaded from environment variables

- Comprehensive error handling
- Chat history is displayed in the interface
- Code is well-structured and documented
- README includes complete setup instructions and testing guide

Professional Skills Assessment

- Problem-Solving: Demonstrated through milestone challenge resolution
- Learning Agility: Shown through review feedback implementation
- Communication: Clear documentation and mentor interaction
- Quality Focus: Attention to review compliance and standards