Project 1: Basic LLM Chatbot

Objective (Why?)

Build a simple web chatbot where users can send prompts and receive responses from an LLM in just 2 days. 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. Each milestone must pass all relevant review templates.

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/

FAQ

- "Which LLM should I use?" Either OpenAI GPT-3.5/4 or Google Gemini - both are acceptable
- "Do I need to deploy?" Deployment is optional but recommended for demo purposes in further development project assignments
- "What if my API calls fail?" Implement basic error handling and show user-friendly error messages

Primary Objectives

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

Performance Standards

- API Response Time: Average < 3 seconds for 95% of requests
- Error Rate: < 5% failed API calls under normal conditions
- User Experience: Loading indicators, graceful error handling
- Resource Usage: Efficient memory and processing patterns

Assessment Scoring Guide

- 90-100: Exceptional (Production-ready, exceeds expectations)
- 80-89: Proficient (Meets all requirements, minor improvements needed)
- 75-79: Developing (Meets basic requirements, some improvements needed)
- Below 75: Needs Support (Requires additional work before project advancement)

Success Criteria Checklist

- User can type a message and receive an LLM response (90%+ success rate)
- API key is loaded from environment variables (100% compliance)
- Comprehensive error handling implemented (5+ scenarios covered)
- Chat history is displayed in the interface with proper formatting
- Code is well-structured and documented (75%+ code quality score)
- README includes complete setup instructions and testing guide

Project 1 → Project 2 Advancement Requirements

Review Template Compliance

- AI Integration Review
  - Focus: Clean AI service integration and error handling
  - Critical for Project 2's web scraping + AI analysis requirements
- Security Review
  - Focus: Environment security, input validation, no vulnerabilities
  - Foundation for all subsequent projects
- Code Quality Review
  - Focus: Organization, documentation, maintainable patterns
  - Builds foundation for complex projects ahead
- Architecture Review
  - Focus: Clean design patterns and scalable structure
  - Essential for full-stack development in further Project

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

Testing & Validation Procedures

Testing Requirements by Milestone

Milestone 1 Testing

Python
```python
# Security Testing
    # Verify no API keys in source code
    # Check .env file exists and .env.example provided
    # Validate git history contains no secrets

# AI Integration Testing
    # Test successful API calls
    # Test error handling scenarios
    # Validate response parsing
```

Milestone 2 Testing

Python
```python
# Performance Testing
    # Measure API response times
    # Test under various load conditions
    # Validate user experience metrics

# Comprehensive Error Testing
    # Invalid API key
    # Network timeout
    # Rate limit exceeded
    # Malformed responses
    # Empty/null inputs
```

Milestone 3 Testing

Python
```python
# Integration Testing
# End-to-end user flow testing
# Cross-browser compatibility
# Production deployment readiness
```

Code Scanning & Vulnerability Assessment

Required Scans Before Each Review

- Static Code Analysis: Use pylint/flake8 for code quality
- Security Scanning: Run bandit for security vulnerabilities
- Dependency Checking: Verify all dependencies are secure and updated
- Git History Scan: Ensure no secrets in commit history

Review Template Execution

Pre-Review Checklist

- All milestone deliverables completed
- Testing procedures executed and documented
- Code scanning completed with issues addressed
- Documentation updated and comprehensive
- Working demo prepared and tested

Review Execution Process

1. Load Templates: Open relevant review checklists from Templates folder
2. Execute Reviews: Follow systematic review process for each category
3. Document Findings: Complete all sections of review templates
4. Generate Action Points: Create specific, actionable improvement items
5. Score Assessment: Provide numerical scores for each review category
6. Advancement Decision: Determine readiness for next project based on scores

Task Tracking & Project Management Integration

Milestone 1: Foundation Setup & API Integration

Feature 1.1: Environment & Project Setup

Task ID: P1-M1-SETUP
Priority: Critical
Dependencies: None

Sub-tasks:

- P1-M1-SETUP-01: Create project directory structure
    - Description: Setup project folders, virtual environment, git repository
    - Acceptance Criteria:
        - Project folder with proper structure created
        - Python virtual environment activated
        - Git repository initialized with .gitignore
- P1-M1-SETUP-02: Configure environment variables
    - Description: Setup .env file management for API keys
    - Acceptance Criteria:
        - .env file created with API key placeholder
        - .env.example file created for documentation
        - python-dotenv dependency added
- P1-M1-SETUP-03: Install and configure dependencies
    - Description: Setup requirements.txt and install packages
    - Acceptance Criteria:
        - requirements.txt with all dependencies
        - All packages installed successfully
        - Dependency versions pinned
- P1-M1-SETUP-04: Create basic project structure
    - Description: Setup main application files and folders
    - Acceptance Criteria:

        - app.py or main.py created
        - config.py for configuration management
        - Basic folder structure (utils, services, etc.)

Feature 1.2: LLM API Integration

Task ID: P1-M1-API
Priority: Critical
Dependencies: P1-M1-SETUP

Sub-tasks:

- P1-M1-API-01: Implement API client service
    - Description: Create service class for LLM API communication
    - Acceptance Criteria:
        - Service class with proper abstraction
        - API key loading from environment
        - Basic API call functionality
- P1-M1-API-02: Implement error handling
    - Description: Add comprehensive error handling for API failures
    - Acceptance Criteria:
        - Handle network timeouts
        - Handle invalid API keys
        - Handle rate limiting
        - Handle malformed responses
- P1-M1-API-03: Test API integration
    - Description: Create test cases for API functionality
    - Acceptance Criteria:
        - Successful API call test
        - Error scenario tests
        - Response validation tests

Feature 1.3: Basic Streamlit Interface

Task ID: P1-M1-UI
Priority: High
Dependencies: P1-M1-API

Sub-tasks:

- P1-M1-UI-01: Create basic chat interface
    - Description: Setup Streamlit app with chat components
    - Acceptance Criteria:
        - Chat input component working
        - Message display area created
        - Basic styling applied
- P1-M1-UI-02: Integrate API with UI
    - Description: Connect chat interface to LLM API
    - Acceptance Criteria:
        - User can send messages
        - LLM responses displayed
        - Loading states implemented

Milestone 2: Core Chat Functionality

Feature 2.1: Enhanced Chat Experience

Task ID: P1-M2-CHAT
Priority: High
Dependencies: P1-M1-UI

Sub-tasks:

- P1-M2-CHAT-01: Implement conversation history
    - Description: Add persistent chat history within session
    - Acceptance Criteria:
        - Messages persist during session

        - Clear conversation history option
        - Proper message formatting
- P1-M2-CHAT-02: Add session state management
    - Description: Implement Streamlit session state for conversation
    - Acceptance Criteria:
        - Conversation persists on page refresh
        - Session state properly managed
        - Memory efficiency maintained
- P1-M2-CHAT-03: Enhance user experience
    - Description: Add loading indicators and feedback
    - Acceptance Criteria:
        - Loading spinner during API calls
        - Success/error message notifications
        - Response time display

Feature 2.2: Input Validation & Security

Task ID: P1-M2-SECURITY
Priority: Critical
Dependencies: P1-M2-CHAT

Sub-tasks:

- P1-M2-SECURITY-01: Implement input validation
    - Description: Add comprehensive input sanitization
    - Acceptance Criteria:
        - Message length validation
        - Special character sanitization
        - SQL injection prevention
        - XSS prevention
- P1-M2-SECURITY-02: Enhance API security
    - Description: Add rate limiting and request validation
    - Acceptance Criteria:
        - Basic rate limiting implemented

        - Request size validation
        - API key rotation capability

Milestone 3: Application Readiness & Documentation

Feature 3.1: Documentation & Testing

Task ID: P1-M3-DOCS
Priority: High
Dependencies: P1-M2-SECURITY

Sub-tasks:

- P1-M3-DOCS-01: Create comprehensive README
    - Description: Write detailed project documentation
    - Acceptance Criteria:
        - Installation instructions
        - Configuration guide
        - Usage examples
        - Troubleshooting section
- P1-M3-DOCS-02: Add code documentation
    - Description: Add docstrings and inline comments
    - Acceptance Criteria:
        - All functions have docstrings
        - Complex logic commented
        - Type hints added
- P1-M3-DOCS-03: Create testing procedures
    - Description: Document testing and validation steps
    - Acceptance Criteria:
        - Manual testing checklist
        - Automated test cases
        - Performance testing guide

Feature 3.2: Quality Assurance & Review Preparation

Task ID: P1-M3-QA

Priority: Critical

Dependencies: P1-M3-DOCS

Sub-tasks:

- P1-M3-QA-01: Run code quality scans
  - Description: Execute all required code scanning tools
  - Acceptance Criteria:
    - pylint score > 8.0
    - bandit security scan clean
    - pip-audit dependency check passed
- P1-M3-QA-02: Prepare demo and deployment
  - Description: Setup working demo and deployment preparation
  - Acceptance Criteria:
    - Local demo working perfectly
    - Deployment documentation ready
    - Demo scenarios prepared