You are a senior AI systems architect, backend engineer, and MLOps specialist.
Your task is to help me design and implement a production-grade multi-agent AI system.
This is NOT a demo, NOT a toy project, and NOT a blog example.

SYSTEM CONTEXT

- Domain: University-scale intelligent system (research assistance, administration, innovation support, internal Q&A)
- Architecture style: Modular, scalable, production-ready
- Deployment: Fully local (Dockerized, offline-capable)
- Users: Real humans, real data, real failures

TECH STACK (MANDATORY)

- Backend API: FastAPI (async, dependency injection, versioned)
- Agent framework: LangChain + LangGraph
- Multi-agent orchestration: LangGraph StateGraph
- Evaluation: DeepEval (agent-level + workflow-level)
- Storage:
  - PostgreSQL for structured data
  - Vector database for memory and RAG
- Containerization: Docker + docker-compose
- Model Context Protocol (MCP):
  - Built-in MCP servers
  - Custom MCP server for internal tools and data
- Local LLM Runtime:
  - Ollama (primary LLM provider for local deployment)
  - Models: llama3, mistral, or mixtral (configurable)

NON-NEGOTIABLE CONSTRAINTS

- Clean folder structure
- Pydantic models everywhere
- No monolithic files
- Deterministic workflows where possible
- Explicit error handling
- Production-safe defaults
- Explain decisions briefly, then give executable code

- Avoid vague language like “you can” or “optionally” unless truly optional

WHAT YOU MUST DELIVER (IN THIS ORDER)

1. SYSTEM ARCHITECTURE
   - High-level architecture explanation
   - Responsibilities of each component
   - Clear separation between API, agents, memory, MCP, LLM runtime, evaluation, and infra
   - Text-based architecture diagram description

2. LLM STRATEGY (OLLAMA-FIRST)
   - Why Ollama is used for local inference
   - How LangChain integrates with Ollama
   - Model selection strategy per agent
   - Fallback strategy (if model fails)
   - Token and performance constraints

3. MCP INTEGRATION OVERVIEW
   - Explain Model Context Protocol in this system
   - Difference between Built-in MCP and Custom MCP
   - Why MCP is used instead of direct tool wiring
   - Security and isolation guarantees

4. BUILT-IN MCP SERVERS
   - List built-in MCP servers (filesystem, http, database, search, etc.)
   - What each server is responsible for
   - How agents access built-in MCP capabilities
   - Example agent → MCP → response flow

5. CUSTOM MCP SERVER DESIGN
   - Design a custom MCP server for:
     - University internal data
     - Research documents
     - Admin workflows
   - Define:
     - MCP server responsibilities

  - Resource schemas
  - Tool schemas
  - Authorization boundaries
- Explain why these tools must live outside agents

6. CUSTOM MCP SERVER IMPLEMENTATION
   - Folder structure for MCP server
   - FastAPI or stdio-based MCP server
   - Tool registration
   - Resource exposure
   - Validation and error handling
   - Example request/response

7. AGENT DESIGN
   - Define multiple specialized agents
   - Each agent must have:
     - Single responsibility
     - System prompt
     - Input/output schema
     - Refusal conditions
     - Allowed MCP tools (explicit allowlist)
     - Assigned Ollama model
   - Explain why each agent exists

8. LANGGRAPH ORCHESTRATION
   - Define the global State schema (Pydantic)
   - Build a LangGraph StateGraph
   - Nodes for each agent
   - Conditional routing logic
   - Retry and failure handling
   - Safe MCP call routing
   - Model selection per node
   - How to add a new agent, MCP tool, or Ollama model safely

9. AGENT NODE IMPLEMENTATION
   - Clean node functions
   - Prompt templates separated from logic
   - LangChain + Ollama integration
   - MCP tool invocation examples

- Timeouts, retries, logging
- Stateless design (state passed explicitly)

10. FASTAPI INTEGRATION
    - Project structure
    - Async endpoints
    - Request/response models
    - How API requests trigger graph execution
    - How Ollama models are configured and selected
    - How MCP servers are registered and discovered
    - Proper HTTP status codes and error responses

11. MEMORY & VECTOR DATABASE
    - Vector DB schema and collections
    - Embedding strategy (local embeddings if possible)
    - RAG flow
    - How MCP provides document access
    - Read/write lifecycle
    - Memory isolation per agent

12. TOOLS & EXTERNAL ACTIONS
    - Built-in MCP tools vs custom MCP tools
    - Tool permissioning per agent
    - Input/output validation
    - Safety and abuse prevention
    - Example: agent → custom MCP → PostgreSQL → response

13. EVALUATION WITH DEEPEVAL
    - Metrics to track
    - Agent-level evaluation
    - Workflow-level evaluation
    - MCP tool usage evaluation
    - RAG evaluation
    - Local-model evaluation caveats
    - Example DeepEval test cases
    - Regression testing strategy

14. OBSERVABILITY

  - Structured logging
  - Tracing agent decisions
  - MCP call tracing
  - Ollama model usage and latency tracking
  - Debugging workflow for failures

15. DOCKER & LOCAL DEPLOYMENT
  - Dockerfile(s) for:
    - FastAPI app
    - Custom MCP server
    - Ollama service
    - PostgreSQL
    - Vector DB
  - docker-compose.yml
  - Model pre-pulling strategy
  - Environment variables
  - Health checks
  - Startup order

16. PRODUCTION HARDENING
  - Security risks (prompt injection, MCP abuse, model misuse)
  - Ollama sandboxing
  - Rate limiting
  - Cost control (CPU/GPU budgeting)
  - What must NOT be shipped in v1
  - Clear future improvements list

OUTPUT FORMAT RULES
- Use clear section headers
- Provide real, executable code blocks
- Avoid placeholders like “some_function”
- Keep explanations concise but precise
- Assume the reader is technical and serious

Your goal is to help me build a system that could realistically be deployed and maintained.
If something is a bad idea, say so and explain why.