Final Project Guidelines

Project Overview

This capstone project requires you to design and build a **functional AI application** as a team. You will create a production-ready web application that addresses a real-world problem or introduces an innovative solution using LLMs and the technical skills developed throughout the course.

Each team will develop a unique application based on a problem or opportunity you have identified. This project demonstrates your ability to:

- Design and implement AI-powered applications
- Work collaboratively in a team environment
- Apply professional software architecture patterns
- Deploy and monitor production AI systems
- Transform concepts into working software

Important: See the Deliverables section for submission deadlines and requirements.

Technical Requirements

Core Requirements

Backend: Python (Required)

- Python is the industry standard for AI application development
- Must integrate with an LLM API
- Recommended: Google Gemini (free tier available)
- Alternatives: OpenAI GPT, Anthropic Claude, or other major providers (with course professor approval)

Frontend: Your Choice

- Streamlit (covered in class) - simplest option for rapid development
- React, Vue, Next.js, or other modern frameworks
- Any technology that suits your project needs (with course professor approval)

AI Observability & Monitoring

- Langfuse (covered in class) or alternative observability tools (with course professor approval)
- Required for debugging and monitoring your AI application

Deployment

- Application must be publicly accessible
- Platform of your choice (Streamlit Cloud, Render, Vercel, AWS, etc.)

Documentation

- Root README.md with comprehensive project overview (see README_Template.md for example structure)
- docs/ARCHITECTURE.md explaining architecture decisions and technical choices

- Component-level documentation recommended for complex modules
- Complete setup and deployment instructions

Bonus Opportunities

Going beyond the core requirements is encouraged and rewarded. Examples include:

- **Building a REST API**: Professional API architecture demonstrating advanced development skills
- **Exceptional UI/UX**: Polished, intuitive interfaces that enhance user experience
- **Cross-disciplinary integration**: Incorporating machine learning models, predictive analytics, or concepts from other courses
- **Real-world data**: Using actual datasets rather than mock data (e.g., Portuguese market data for predictive models)
- **Other advanced features**: Implementing additional functionality that meaningfully enhances your application's value

Key principle: Innovation and extra effort are valued when they serve your project's goals. Always justify your technical decisions.

Important: Any technology choices outside the recommended stack require course professor approval before implementation. Reach out via email mcardoso@novaims.unl.pt or in person to discuss.

Project Scope

Minimum Viable Product (MVP)

Your application must demonstrate the skills taught in class. Design a project that naturally requires these techniques.

Required Components:

1. **AI Integration**: LLM integration with proper prompting and structured outputs when needed (e.g., extracting data, routing logic, function parameters)
2. **Conversational Interface**: Multi-turn dialogue with the AI
    - Users must be able to interact through chat-based conversation
    - Enables follow-up questions, exploring results, and refining outputs
    - Note: Very few projects genuinely don't benefit from conversational interaction. If you believe yours is an exception, reach out to discuss - you'll need strong justification
3. **Function Calling / Tools**: AI agent with custom tools that extend LLM capabilities
4. **Document Ingestion**: Process and utilize document content
    - Core feature (analyzing user documents) or supporting feature (company FAQs, knowledge base)
    - Must serve a purpose, not just exist to tick a box
5. **Clean Architecture**: Layered structure with separation of concerns
6. **Observability**: Tracing integrated throughout (Langfuse or approved alternative)

Recommended:

- **Multimodal processing**: If your project benefits from image/PDF analysis

Advanced (When It Adds Real Value):

- **RAG with vector databases**: For similarity matching and semantic search use cases (finding similar products, matching user queries to past support tickets, research paper recommendations). Note: With modern 200K+ token context windows, most document Q&A doesn't need RAG - you can process documents directly within context. Use RAG/vector search when you genuinely need "find similar items" functionality or have knowledge bases larger than context limits, not just to check a box.

Key Principle: Balance is critical. Don't add features just to meet requirements, but don't avoid techniques you've learned either. Equally important: don't overcomplicate features that work fine as-is. Apply techniques where they add value, keep things simple where they don't.

Evaluation Criteria

What We're Looking For:

Technical Implementation

- Clean architecture with proper separation of concerns
- Effective use of AI capabilities (LLM integration, prompting, structured outputs)
- Function calling/tools that genuinely extend your application
- Document ingestion that serves a purpose
- Proper observability and tracing throughout

Functionality & User Experience

- Application works reliably and solves the stated problem
- Conversational interface enables meaningful interaction
- Intuitive, polished user experience
- Features make sense for the problem you're solving

Innovation & Effort

- Going beyond minimum requirements where it adds value
- Creative solutions to real problems
- Technical depth and sophistication
- Evidence of research and learning beyond class material

Documentation & Professionalism

- Industry-grade README with clear setup instructions
- Architecture explanation and justification of technical choices
- Clean, well-organized code
- Professional presentation and defense

Presentation & Defense

- Compelling investor pitch
- Effective demonstration of key features

- Ability to defend technical decisions
- Clear communication of project value

Key Principle:

Quality over quantity. A well-executed application that thoughtfully applies appropriate techniques will score higher than a feature-bloated project that awkwardly forces every technique without purpose.

Deliverables

1. Technical Report

Due: December 7, 23:59

Submit via Moodle a detailed technical report outlining your project plan and implementation details.

Filename: `TeamName_TechnicalReport.pdf`

See the Technical Report Guidelines document for complete requirements.

2. Final Project Submission

Due: December 22, 23:59

Moodle Submission:

- Submit complete project as a zip file via Moodle
- **Filename:** `TeamName_FinalProject.zip`

GitHub Repository:

- Add `mb-cardoso` to your project repository as a collaborator
- Repository must include:
  - Complete source code
  - Root README.md with comprehensive overview and setup instructions
  - docs/ARCHITECTURE.md with architecture decisions and technical justifications
  - Component-level documentation (recommended for complex modules)
  - Environment setup instructions (dependencies, API keys, etc.)
  - Clear documentation of how to run the application locally

Deployed Application:

- Application must be deployed and publicly accessible
- Include deployment URL in README
- Platform choice is yours (Streamlit Cloud, Render, Vercel, AWS, etc.)

3. Project Defense

Date: January 17 (Saturday), in person, time slots will be attributed

Duration: 15-minute presentation + up to 15-minute Q&A (maximum 30 minutes total)

- **Investor Pitch**: Present your application as a startup seeking investment (presentation format of your choice)

- **Live Demo**: Demonstrate core functionality and key features
- **Technical Q&A**: Defend technical decisions, architecture choices, and implementation details
- **Evaluation Panel**: Course professor + AI industry professional
- **Team Participation**: All team members must attend and be prepared to answer questions about any aspect of the project

Getting Started

This is an overview of the entire project timeline, from initial conception to final defense.

1. Form Your Team & Choose Your Problem

- Form team composition
- Identify a real-world problem or innovation opportunity
- Brainstorm how AI can address this problem

2. Define Your Application Concept

- What problem does your application solve?
- Who are your users?
- What's the core value proposition?
- How will users interact with it?

3. Plan Your Architecture

- Review clean architecture materials from class
- Design your layered structure (UI, Service, AI, Tools)
- Identify which AI capabilities you need:
  - What tools/functions will extend the LLM?
  - What documents need to be processed?
  - Does your project benefit from multimodal processing?
  - Do you need RAG/vector database (e.g., semantic search for similarity matching)?

4. Start Building

- Set up project structure
- Build core functionality iteratively
- Integrate observability from the beginning
- Deploy early for testing and iteration

5. Technical Report Submission

- Document your architecture and implementation progress
- Justify your technical choices
- Outline remaining features and development plan

- See the Technical Report Guidelines document
- See Deliverables section for deadline and submission details

6. Continue Development Through December

- Complete remaining features
- Test and refine user experience
- Keep deployment updated
- Maintain clean, documented code

7. Final Submission

- Industry-grade README and documentation
- Complete, deployed application
- Clean codebase with proper setup instructions
- See Deliverables section for deadline and submission details

8. Project Defense

- Investor pitch presentation
- Live demonstration
- Technical Q&A defense
- See Deliverables section for date, format, and requirements

Resources & Documentation

Official Documentation

Google Gemini

- Google AI Studio
- Gemini API Documentation
- Google GenAI Python SDK

Streamlit

- Streamlit Documentation
- Streamlit Components
- Streamlit Cloud Deployment

Langfuse

- Langfuse Documentation
- Python SDK Integration
- Tracing Guide

Alternative LLM Providers

OpenAI

- OpenAI API Documentation

- Python SDK

Anthropic Claude

- Claude API Documentation
- Python SDK

Additional Tools

UV Package Manager

- UV Documentation

Database Options (if needed for your project)

- MongoDB Atlas + PyMongo
- PostgreSQL + psycopg2
- SQLite (built into Python, file-based)
- Local file-based storage (JSON, CSV, etc.)

ChromaDB (if using RAG/vector databases)

- ChromaDB Documentation

Course Materials

All course materials and examples are available on Moodle.

Documentation Templates

- README_Template.md - Example README structure for your project