Final Project Requirements

- Due Date: DEC. 19, 2025.

Project Options

You have three options for your final project. All options should have similar scope and leverage AI techniques covered in class (See Technical Stack and Required AI Technique below)

Option 1: Your Own Project

Design and build an application of your choice that incorporates the techniques we've covered.

Option 2: Complete the Receipts Application

Extend the receipt parsing application we started in class with at least one additional AI-powered feature

Option 3: Complete the Reflections Application

Finish the personal reflections app with embedding-based similarity search and recommendations.

Technical Stack Requirements

- **Frontend:** FastHTML
- **Backend:** FastAPI
- **Data Storage:** Database (e.g., PostgreSQL, SQLite) and/or Vector Store (e.g., ChromaDB)
- **AI Framework:** Pydantic AI with simple agents

Required AI Techniques

Your project must incorporate techniques we covered in class:

Structured Output: Using Pydantic models to constrain LLM responses for specific tasks like classification, data extraction, or validation. Example: classifying receipts as "grocery" vs "service" categories.

Retrieval Augmented Generation (RAG): Using embeddings and vector stores to retrieve relevant information before generating responses.

Vector Store Integration Ideas

For Reflections App:

- User enters a new reflection
- System converts reflection to embedding
- Search vector database for similar past reflections
- Display related reflections on the reflections page to help users see patterns or connections in their thinking

For Receipts App (ideas -- only one feeature that uses embeddings is required):

- **Semantic receipt search:** Store itemized receipt data as embeddings, allowing users to search by natural language queries like "show me all times I bought coffee" or "find receipts at Safeway"
- **Duplicate detection:** Compare new receipt embeddings against existing ones to flag potential duplicates
- **Smart categorization:** Use vector similarity to suggest categories for new items based on previously categorized purchases.

For Your Own Project:

Consider how vector stores could enable semantic search, similarity matching, or recommendation features relevant to your application domain.

Code Organization Requirements

Follow separation of concerns: Your code should be organized into logical modules, similar to how we structured the receipts application in class. For example:

- Database operations in one module
- Supabase/external service calls in another
- Frontend code separate from backend logic
- AI/agent functionality in dedicated modules

Do not mix concerns: Code that mixes database logic with frontend code, or combines multiple unrelated responsibilities in single files will be penalized. Each module should have a clear, focused purpose.

Keep it focused: Your code should include only what we've covered in class and what's necessary for your application to work. Don't include extensive testing suites, advanced deployment configurations, or other elements generated by LLMs that we haven't covered. We'll explore those topics in more advanced software engineering courses.

Project Deliverables

1. **Project Description Document:** A concise document (less than one page) describing:
- What your application does
- The AI techniques you're using and how
- Key features and functionality
2. **Class Diagram:** A UML class diagram showing the main classes in your application and their relationships.
3. **Sequence Diagrams:** At least two sequence diagrams illustrating the most important user interactions with your software. These should show how different components communicate to fulfill user requests.
4. **Code Organization Diagram:** A component/module diagram showing how your code is structured and organized. This should illustrate the separation of concerns in your application (e.g., database code, API layer, frontend, AI/agent modules, etc.).

Notes

- The use of Pydantic AI doesn't need to be complex or fully agentic - simple agent implementations with basic tool calling are acceptable
- More sophisticated agentic patterns (multiple tool calls, reasoning loops) are welcome but not required
- Focus on building a complete, working application that demonstrates understanding of the AI concepts
- Lastly, if you're unsure about how to organize your code or have questions about the requirements, please reach out to me or the TAs before starting implementation. We're happy to help you plan your approach.