# Take-Home Assignment

*RFP Analyzer Multi-Agent System*

**Assignment Overview**

**Domain:** Architecture & Interior Decor Industry

**Objective:** Build an AI-powered system that analyzes RFP documents and extracts structured requirements

**First Check-In:** Monday, December 1

**Submission Deadline:** Friday, December 5

## 1. Problem Statement

Interior design firms and architecture companies receive Request for Proposal (RFP) documents from clients seeking renovation or design services. These documents come in various formats and levels of detail, containing requirements about furniture, materials, colors, dimensions, brand preferences, and technical specifications.

Your task is to build a multi-agent system that automates the extraction and organization of requirements from these RFP documents. The system should parse unstructured document content, categorize extracted information, and present it in a structured format that allows for human review and refinement.

---

## 2. Functional Requirements

### 2.1 Document Input

- Accept RFP documents in PDF, DOCX, or plain text format
- Support single-file uploads through the UI
- Handle varying document structures (formal RFPs, email briefs, specification sheets)

### 2.2 Extraction Requirements

The system must extract and categorize the following from RFP documents:

| Category | Examples |
|---|---|
| Project Metadata | Project name, client type (residential/commercial), location, timeline, budget range |
| Space Requirements | Room types, dimensions, square footage |
| Item Requirements | Furniture, fixtures, appliances, decor items with quantities |
| Technical Specs [per item] [nullable] | Dimensions, weight limits, electrical requirements |
| Material Preferences [per item] [nullable] | Wood types, fabric, metal finishes, stone, etc. |

| | |
|---|---|
| Color Preferences [per item] [nullable] | Specific colors, palettes, themes (modern, rustic, minimalist) |
| Brand Preferences [per item] [nullable] | Preferred vendors, "or equivalent" notations, exclusions |
| Special Instructions [per item] [nullable] | Installation requirements, delivery constraints, accessibility needs |

### 2.3 Human-in-the-Loop (HITL) Interface

Users should be able to:

- View extracted requirements in an organized, editable format
- Confirm, modify, or reject individual extractions
- Add missing requirements manually or via prompting
- Export final validated requirements

---

## 3. Technical Requirements

### 3.1 Required Tech Stack

| Component | Technology |
|---|---|
| Agent Framework | PydanticAI |
| Backend | Python with FastAPI |
| Frontend | Streamlit or Gradio |
| Database | PostgreSQL |
| LLM | Any LLM of your choice |

### 3.2 Multi-Agent Architecture

Design and implement a multi-agent system with clear separation of concerns. Below is a suggested architecture that you may adapt or modify based on your design decisions:

- **Orchestrator Agent:** Manages workflow, routes documents, aggregates results
- **Document Parser Agent:** Extracts raw text, identifies document structure and sections
- **Project Classifier Agent:** Determines project type, scale, and domain
- **Requirements Extractor Agent:** Pulls out specific requirements by category
- **Specification Analyzer Agent:** Handles technical specs, dimensions, quantities
- **Preference Extractor Agent:** Identifies subjective preferences (colors, styles, brands)
- **Validation Agent:** Cross-checks extractions, flags conflicts, assigns confidence scores

### 3.3 Tool Implementation

The following are some suggestions for tool implementation, that you can modify based on your preferences and agent architecture:

**Document Processing:**

- parse_pdf(file_path) → DocumentContent
- parse_docx(file_path) → DocumentContent
- extract_tables(document) → List[Table]

**Data Extraction:**

- extract_dimensions(text) → List[Dimension]
- extract_quantities(text) → List[ItemQuantity]
- parse_budget_range(text) → BudgetRange

**Persistence:**

- save_extraction(project_id, data) → bool
- update_requirement(project_id, requirement_id, updates) → bool

---

## 4. API Specification

The following is a SUGGESTED API spec:

| Method | Endpoint | Description |
|---|---|---|
| POST | /api/projects/upload | Upload RFP document(s) |
| GET | /api/projects/{id} | Get project details |
| GET | /api/projects/{id}/analysis | Get extraction results |
| POST | /api/projects/{id}/analyze | Trigger analysis |
| PATCH | /api/projects/{id}/requirements/{req_id} | Update requirement (HITL) |
| POST | /api/projects/{id}/requirements | Add requirement manually |
| GET | /api/projects/{id}/export | Export requirements (JSON/CSV) |

---

## 5. UI Requirements

Build a functional UI using Streamlit or Gradio with the following screens:

**1. Upload Screen**

- File upload
- "Analyze" button to trigger processing

**2. Analysis Dashboard**

- Project metadata summary card
- Requirements grouped by category (can be collapsible sections or a simple JSON/markdown render)
- Confidence indicators (color-coded: green/yellow/red) [OPTIONAL]

**3. Requirement Edit View**

- Editable fields for requirement details
- Verify / Reject / Modify actions

**4. Export View**

- Format selection (JSON, CSV)
- Download button

---

## 6. Test Documents

You are provided with two sample RFP documents of varying complexity:

1. **Sample_RFP_Simple_HomeOffice.docx** — Single-room home office renovation (simple scenario)
2. **Sample_RFP_Medium_ApartmentDesign.docx** — Full apartment interior design (medium complexity)

Your system should successfully process both documents and extract relevant requirements. Use these to test and demonstrate your implementation.

---

## 7. Deliverables

| Item | Format | Required |
|---|---|---|
| GitHub repository with complete source code | Private repo (share view access with amansharma2910) | ✓ |
| README with setup & run instructions | Markdown | ✓ |
| Architecture diagram (agents, tools, data flow) | PNG/PDF/Mermaid | ✓ |
| Working demo against provided sample RFPs | Running application | ✓ |
| API documentation | Swagger/OpenAPI | ✓ |
| Design decisions document | Markdown (1-2 pages) | ✓ |
| Video walkthrough (5 min max) | MP4/Loom link | Optional, but preferred |

---

## 8. Evaluation Criteria

| Criteria | Weight | What We Look For |
|---|---|---|
| Agent Architecture | 25% | Clear separation of concerns, appropriate agent decomposition, clean orchestration |
| Tool Implementation | 20% | Robust tool functions, proper error handling, type safety |
| Extraction Quality | 20% | Accuracy on sample documents, edge case handling, confidence calibration |
| HITL Implementation | 15% | Intuitive feedback flow, state persistence, validation updates |
| Code Quality | 15% | Clean structure, typing, documentation, tests |
| UI/UX | 5% | Usability, clear information hierarchy, responsive feedback |

---

## 9. Scope Guidance

**Prioritize depth over breadth.** We'd rather see 2-3 well-implemented agents with clean orchestration than 7 shallow agents.

- Focus on extraction quality over UI polish
- A working MVP with clear extension points is better than an incomplete feature-rich system
- Document your design decisions and trade-offs
- If you run out of time, explain what you would have done differently

---

## 10. Bonus Challenges (Optional)

If you finish early or want to demonstrate additional skills:

1. **Image Analysis:** Some RFP documents can also contain images of reference products. Perform image analysis, extract reference images' features and metadata, which then can be used to augment extracted requirements.
2. **Authentication:** Basic user sessions with project history
3. **Async Processing:** Handle large documents with background processing and status updates

---

## 11. Questions?

If you have clarifying questions about the requirements, please reach out to aman@theagentic.ai (cc: mihir@theagentic.ai, drishtavya@theagentic.ai)

---