ChatGPT

AI-Powered RPA Flow Designer: Step-by-Step Implementation Plan

1. Environment Setup

- **Python & Tools:** Create a Python 3.11+ virtual environment and install required libraries. For example:

```
python3 -m venv venv
source venv/bin/activate
pip install fastapi uvicorn streamlit requests openai crewai streamlit-flow-component pdfkit
```

  This installs FastAPI (server), Streamlit (UI), CrewAI (multi-agent framework), the `streamlit-flow-component` (React Flow diagrams) and PDFKit bindings [1] [2].
- **API Keys:** Store API keys (OpenAI, vector DB) in a `.env` file or environment variables. Load them in code (e.g. using python-dotenv).
- **wkhtmltopdf:** Install the `wkhtmltopdf` binary, which PDFKit uses to render HTML to PDF. For example, on macOS use `brew install --cask wkhtmltopdf`, on Linux `sudo apt-get install wkhtmltopdf` [3]. Verify with `wkhtmltopdf --version`.
- **Project Structure:** Organize the repo into modules (e.g. `/backend` for FastAPI code, `/frontend` for Streamlit app, `/agents` for CrewAI definitions). Keep UI code separate from server logic to maintain modularity [4]. *Estimated time:* 1–2 days.

2. Build the Vector Database for RPA Documentation

- **Gather Docs:** Collect up-to-date action lists and parameters from Power Automate, UiPath, and Automation Anywhere 360 documentation (e.g. export to text or JSON).
- **Embeddings:** Use OpenAI embeddings (e.g. `text-embedding-ada-002`) to encode each action name, description, and parameters. Initialize the OpenAI client (e.g. `client = OpenAI(api_key=api_key)`) [5].
- **Vector DB:** Create a persistent vector store (ChromaDB or Qdrant). For each doc item, compute and upsert an embedding. (See [3] for an example of setting up ChromaDB with OpenAI embeddings.)
- **Testing Search:** Write a small query function that, given a natural-language keyword or action name, returns the nearest vector DB entries. This RAG datastore will power the Documentation Retriever agent. *Estimated time:* 2 days.

3. FastAPI Backend Development

- **Project Setup:** Scaffold a FastAPI app. Define endpoints for (a) receiving user queries, (b) returning structured responses or flow JSON, and (c) triggering PDF generation/download. Use Uvicorn as the ASGI server.

- **Business Logic Layer:** In FastAPI route handlers, call CrewAI agents or tasks. For example, when a user submits a query, create and kickoff a CrewAI task sequence. Ensure that none of these backend functions depend on Streamlit APIs [4]. Keep the FastAPI code focused on input/output of data (JSON) and orchestration.
- **REST API Calls:** The Streamlit frontend will call these endpoints (e.g. via `requests.post`). The UI thread should not block; use asynchronous calls or background threads if needed.
- **CORS & Config:** If deploying frontend/backend separately, configure CORS in FastAPI (`fastapi.middleware.cors`) to allow the Streamlit domain.
- **Caching:** For repeat queries or static data (like doc embeddings), use caching (e.g. `functools.lru_cache`) at the FastAPI level. This mimics Streamlit's caching but on the server side [4]. *Estimated time:* 1–2 days.

4. Streamlit Frontend (Chat UI & Flow Canvas)

- **Chat Interface:** Use Streamlit's built-in chat widgets. For example, use `st.chat_input` to get user text and `st.chat_message` to display messages [6]. Maintain a `st.session_state.messages` list of `{"role": "user/assistant", "content": ...}` to render the conversation.
- **Submitting Queries:** When the user sends a message, display it in the chat and send it to the FastAPI `/query` endpoint. Upon receiving the AI response (parsed JSON), display the structured steps or acknowledgments.
- **Flow Diagram:** Embed a React Flow canvas via the `streamlit-flow` component [2]. When the Flow Designer Agent outputs a graph structure (nodes/edges), pass it to this component. The user should be able to view (but not necessarily edit) the generated flowchart.
- **Flow Interaction (Optional):** Use the component's callback (it can return clicked node IDs) to allow clicking a node to see its details. For now, primary function is display.
- **PDF Download:** Add a button that calls a FastAPI endpoint (e.g. `/generate-pdf`). When clicked, Streamlit requests the PDF file and then offers it for download.
- **UI Logic:** Keep UI code separate from agent logic. The Streamlit app simply sends/receives data and renders chat/diagrams [4]. *Estimated time:* 1–2 days.

5. CrewAI Agents Implementation

5.1 Requirement Structuring Agent

- **Function:** Parses the user's natural-language request into a list of **structured automation steps** (e.g. bullet list of tasks).
- **Master Prompt:** Craft a system prompt like: *"You are an assistant that converts a user's automation request into an ordered list of clear, step-by-step tasks for an RPA workflow."* The LLM (GPT-5 Nano) should be instructed to output JSON or markdown list format.
- **Role & Goal:** Role = "Requirement Analyst"; Goal = "Extract and enumerate all required automation steps from the user's query."
- **Tools:** Use the OpenAI LLM itself (no external tools needed). You may also use CrewAI memory to recall previous tasks if iterative refinement is needed.
- **Implementation:** In a CrewAI Task, call this agent with `description` equal to the user's input. For example: `Task(description=user_query, agent=requirement_agent)`. The expected output is a structured list/JSON. The output feeds into the next agent. Test with sample queries to ensure reliability. *Estimated time:* 1 day.

5.2 Tool Mapper Agent

- **Function:** Takes the structured steps from the previous agent and **maps each to specific RPA tool actions** (naming the tool and action).
- **Master Prompt:** e.g. *"Map each of the following generic tasks to a specific action in Power Automate, UiPath, or Automation Anywhere 360. Provide the action name and necessary parameters."*
- **Role & Goal:** Role = "Tool Mapper"; Goal = "Translate high-level steps into concrete RPA actions per tool."
- **Tools:** Use GPT-5 Nano, and optionally the vector DB via the Documentation Retriever agent as a tool (see below). For example, the mapper can propose an action name, then query the doc DB to confirm it exists or get exact parameter names.
- **Implementation:** Create a CrewAI Task that takes the list of steps as input. For each step, the agent lists one or more tool actions (tool choice, action name, param placeholders). You may implement this agent with a loop or map over tasks. Test by verifying the mapper's output against known tool actions. *Estimated time:* 1–2 days.

5.3 Documentation Retriever Agent (RAG)

- **Function:** Enriches actions by retrieving **up-to-date action names and parameter details** from official documentation.
- **Master Prompt:** e.g. *"You have a vector database of RPA tool documentation. Given an action name or description, retrieve the exact official action name and parameter list."*
- **Role & Goal:** Role = "Documentation Retriever"; Goal = "Fetch precise action details from the docs DB."
- **Tools:** This agent's main tool is a *vector search* over the pre-built embeddings (the RAG vector DB). Implement a custom CrewAI Tool that performs semantic search: e.g., `search_tool.query(query_text) -> [top_docs]`.
- **Implementation:** For each action name proposed by the Tool Mapper, call the search tool with that name or step description to retrieve matching documentation entries. Then format and return the canonical action name and parameters. Ensure the vector DB returns high-confidence results (you might threshold by similarity score). *Estimated time:* 1 day.

5.4 Flow Designer Agent

- **Function:** Generates the **visual flow diagram data** (nodes, edges, labels) from the mapped and enriched tasks.
- **Master Prompt:** *"Create a flowchart structure (nodes and directional edges) representing the following sequence of RPA actions, including any splits or loops if needed."*
- **Role & Goal:** Role = "Flow Architect"; Goal = "Produce a JSON model of the workflow graph."
- **Tools:** The agent should output data in the format expected by the React Flow component (a list of nodes with `id/position/data` and edges linking them). You might incorporate a layout engine (e.g. ElkJS via the streamlit component) by specifying `x,y` positions for each node. Alternatively, let the React component auto-layout.
- **Implementation:** Pass the list of final actions (with labels) to this agent. It should output a JSON or Python dict with `nodes=[{id,label}], edges=[{source,target}]`. If using CrewAI, define a task with `agent=flow_agent` that consumes previous outputs. After running, send this JSON to the Streamlit app to render. Test by verifying the generated graph matches the intended sequence. *Estimated time:* 1 day.

5.5 PDF Generator Agent

- **Function:** Compiles the final flow and action details into a downloadable **PDF report**.

- **Master Prompt:** *"Using the provided workflow graph and action metadata, generate an HTML report summarizing each step and include the flow diagram image."* (Alternatively, this agent could output instructions for PDFKit.)
- **Role & Goal:** Role = "Report Builder"; Goal = "Produce a printable PDF summarizing the workflow."
- **Tools:** Use the `pdfkit` library. Steps: (1) Generate an HTML or Markdown summary of the actions (with descriptions/parameters). (2) Render the React Flow diagram to HTML (e.g. via `streamlit-flow` export or by re-generating in HTML/CSS). (3) Call `pdfkit.from_string(html_string, output_path)` to create the PDF [7].
- **Implementation:** After flow data is ready, the backend calls PDFKit. You may have an HTML template that includes both the flow (embedded as SVG/PNG) and a table/list of action metadata. Test by running on known workflows and inspecting the PDF. *Estimated time:* 1 day.

6. Maintain Modularity and Testability

- **Modular Code:** Keep each agent and service in its own Python module. For example, place CrewAI agent definitions in a separate `agents.py` or YAML file. FastAPI routes should only call service-layer functions. This separation makes unit testing easier [4].
- **Testing Agents:** Write unit tests for each agent's logic. For LLM-driven agents, consider mocking the LLM output (or use a small example prompt) to test the parsing and formatting steps. For the RAG agent, test that queries to the vector DB return correct hits.
- **Version Control:** Use Git branches for each major feature (e.g. a branch per agent or component). Review and test before merging.
- **Logging & Feedback:** Implement detailed logging in the FastAPI backend and CrewAI tasks (CrewAI can log each agent's steps). This aids debugging of the multi-agent flow.
- **Continuous Integration:** (Optional) Set up CI checks (e.g. pytest) to run basic tests on each commit. *Estimated time:* 1 day.

7. Deployment

- **Containerization:** Create a Dockerfile that runs both the FastAPI server and Streamlit app (possibly using separate containers and Docker Compose). Streamlit can serve the UI on port 8501 and FastAPI on, say, 8000. Configure the frontend to call the appropriate backend URL.
- **Cloud Hosting:** Deploy to a cloud platform. For example, use Heroku or AWS: run FastAPI on a public endpoint, and deploy the Streamlit app (Streamlit Sharing or another service) pointing to that endpoint. Alternatively, serve both under one domain with a proxy (NGINX).
- **Secrets:** Ensure API keys are set in the production environment securely.
- **Monitoring:** Add simple health endpoints (e.g. `/health`) and monitor logs. *Estimated time:* 1–2 days.

8. Agent Definitions (CrewAI)

Define each agent in CrewAI (YAML or code) with fields as shown below. You can refer to CrewAI docs for attribute meanings [8]. For clarity, here are example definitions:

- **Requirement Structuring Agent:**
- *Master Prompt:* "Convert the user's automation request into an ordered list of clear, step-by-step tasks for an RPA workflow."
- *Description:* Parses free-form input into structured task steps.
- *Role:* Requirement Analyst [8].

- *Goal:* Extract and enumerate automation steps from user input.
- *Tools:* GPT-5 Nano (OpenAI LLM) to interpret text.

- **Tool Mapper Agent:**

- *Master Prompt:* "Map each task to the corresponding RPA action (tool, action name, and parameters)."
- *Description:* Assigns specific RPA tool actions to the structured steps.
- *Role:* Action Mapper [8].
- *Goal:* Translate generic steps into concrete RPA tool actions.
- *Tools:* GPT-5 Nano; may call the Documentation Retriever for validation.

- **Documentation Retriever Agent:**

- *Master Prompt:* "Look up the official action name and parameters for the given RPA tool action in the documentation database."
- *Description:* Fetches precise action names and parameter details via semantic search.
- *Role:* Data Retriever [8].
- *Goal:* Retrieve updated documentation entries for each mapped action.
- *Tools:* Vector DB search tool (e.g. custom CrewAI Tool wrapping Qdrant/Chroma) [9].

- **Flow Designer Agent:**

- *Master Prompt:* "Generate a workflow diagram (nodes and edges) representing these RPA steps."
- *Description:* Builds the visual flowchart structure from the final task list.
- *Role:* Flow Architect [8].
- *Goal:* Produce a data model of the workflow graph for rendering.
- *Tools:* GPT-5 Nano for layout logic; possibly a graph layout engine (ElkJS via streamlit-flow).

- **PDF Generator Agent:**

- *Master Prompt:* "Compile the workflow diagram and action details into a formatted PDF report."
- *Description:* Creates the final PDF containing the flowchart and step descriptions.
- *Role:* Report Builder [8].
- *Goal:* Generate a ready-to-download PDF of the flow and metadata.
- *Tools:* PDFKit library (wkhtmltopdf) to render HTML/Markdown into PDF.

Each agent's *role* and *goal* guide its behavior; *tools* lists capabilities like the OpenAI API or database queries [8]. Use CrewAI's `Crew(kickoff)` or asynchronous tasks to orchestrate them in sequence, using task `context` to pass outputs (see CrewAI docs for chaining) [10].

References: We leverage proven tools and patterns – e.g. using Streamlit's chat widgets for the UI [6], separating frontend/back-end as advised [4], React Flow via the `streamlit-flow` component for diagrams [2], and Python-PDFKit (`wkhtmltopdf` backend) for PDF export [7] [3]. The modular, agentic pipeline follows CrewAI multi-agent RAG best practices [9] [11]. (Effort estimates assume a solo developer with strong Python skills.)

---

1 5 Building an Agentic Chatbot Framework with Multi-Agent Collaboration using Crew AI, FastAPI, and Streamlit | by Vijjeswarapu Surya Teja | Medium
https://medium.com/@surya.vijjeswarapu/building-an-agentic-chatbot-framework-with-multi-agent-collaboration-using-crew-ai-fastapi-and-617a15fbb1f4

2 New Component: Streamlit Flow - Beautiful, Interactive and Flexible Flow Diagrams in Streamlit - Custom Components - Streamlit
https://discuss.streamlit.io/t/new-component-streamlit-flow-beautiful-interactive-and-flexible-flow-diagrams-in-streamlit/67505

3 7 How to Generate PDFs from HTML with Python-PDFKit (HTML string, HTML File and URL) - APITemplate.io
https://apitemplate.io/blog/how-to-generate-pdfs-from-html-with-python-pdfkit/

4 Fastapi backend -> streamlit frontend? - Deployment - Streamlit
https://discuss.streamlit.io/t/fastapi-backend-streamlit-frontend/55460

6 Build a basic LLM chat app - Streamlit Docs
https://docs.streamlit.io/develop/tutorials/chat-and-llm-apps/build-conversational-apps

8 Agents - CrewAI
https://docs.crewai.com/en/concepts/agents

9 Simple Agentic RAG System - Qdrant
https://qdrant.tech/documentation/agentic-rag-crewai-zoom/

10 Tasks - CrewAI
https://docs.crewai.com/en/concepts/tasks

11 Agentic RAG using CrewAI. In this post, we will explore the... | by Ansuman Das | Medium
https://medium.com/@ansumandasiiit/agentic-rag-using-crewai-6a5f2d366020