ChatGPT

Project Plan: Agentic Workflow Webapp for Excel-Based Automation

---

Vision

Build a web application that allows users to create and manage custom agentic workflows. These workflows will process uploaded Excel files, extract relevant rows based on logic, and trigger automated actions like sending emails. Users should be able to:

- Log in securely.
- Define workflows using a visual drag-and-drop interface.
- Save and reuse workflows as templates.
- Upload new reports and automatically process them via linked workflows.

Initial Use Case

> **"Given a specific Excel file, extract relevant rows, email them to configured recipients, and save this logic as a template for future automation."**

Each user should be able to:

1. Upload and validate a report (Excel file).
2. Visually create a workflow (read file → extract rows → send email).
3. Save this as a reusable template.
4. Upload similar reports and run the workflow automatically.

⚖️ MVP Scope

- **User Authentication**: Users can sign up, log in, and manage their templates.
- **Template Management**:
- Save/load/update/delete workflows.
- Basic validation of input file matching the template structure.
- **Drag-and-Drop Workflow Builder**:
- Use React Flow to create visual workflows.
- Minimal components: Excel Reader, Row Filter, Email Sender.
- **Backend Execution Engine**:
- FastAPI parses the visual graph and triggers mocked/real processing.
- **Local Development Environment**:
- Run everything locally on M1 Max Mac using conda + VSCode.

Target Architecture (for MVP)

Frontend (React + React Flow):

- Drag and drop interface
- Node configuration UI (e.g., email credentials, row filter rules)
- Save graph as JSON

Backend (FastAPI):

- Auth APIs (mock initially)
- Endpoint to receive and parse workflow graph
- Executor module to run mocked agentic steps (eventually CrewAI-based)

Database (Post-MVP):

- Store users, templates, execution logs

Cloud Deployment (Later Phase):

- GCP or AWS (TBD)
- Possibly use Render/Fly.io for staging

📊 Tech Stack

- **Frontend**: React, React Flow, Vite, Tailwind CSS (optional)
- **Backend**: FastAPI, Pydantic, Uvicorn
- **Auth**: Clerk/Auth0/Supabase (TBD)
- **Execution Framework**: Mocked for MVP, CrewAI later
- **Database**: PostgreSQL (via Supabase or local)
- **Cloud**: Local now, GCP/AWS later

Immediate Next Steps

1. **Set up FastAPI backend** locally
2. **Create simple React + React Flow frontend**
3. Connect frontend to backend with one mocked flow:
4. Upload JSON graph → Backend parses it → Returns "executed"
5. Gradually increase graph complexity (file handling, logic, email)

Development Guidelines

- Go slow and step-by-step (since all tech is new)
- Build locally first
- Avoid cloud and database setup until local MVP is proven
- Document each module clearly and keep each one minimal

Long-Term Features (Deferred)

- OAuth login and cloud hosting
- Multiple node types (splitter, condition, database connector)
- Multi-user workflows and permissions
- Full CrewAI agent orchestration with tool memory
- CI/CD, staging/prod environments