Intelligent Workflows AI Planet - Architecture Design Document

- **Document Version:** 1.0
- **Last Updated:** January 18, 2026
- **Project:** Intelligent Workflows - No-Code AI Workflow Builder

1. Executive Summary

Intelligent Workflows AI Planet is a no-code, drag-and-drop workflow automation platform that enables users to create complex AI-powered workflows without writing code. The system integrates Large Language Models (LLMs), document processing, and visual workflow orchestration to provide an intuitive interface for building intelligent automation solutions.

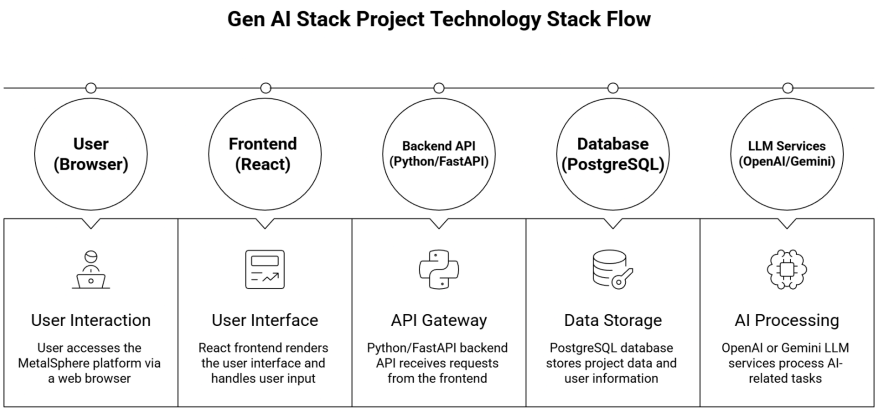
Key Features:

- Visual workflow designer with drag-and-drop interface.
- AI/LLM integration (GPT-4, Gemini, Claude).
- Document processing with embeddings.
- Node-based workflow execution.
- Real-time response generation.
- User authentication and workflow management.

2. High-Level Architecture (HLD)

2.1 System Overview

The platform follows a standard client-server model integrating external AI services.



Detailed Connection Flow:

- **Frontend (React)** communicates via **REST API / WebSocket** with -> **Backend (Python)**

- **Backend** reads/writes to -> **Database (PostgreSQL)**
- **Backend** sends prompts to -> **External LLM Services**

2.2 Layered Architecture

Presentation Layer

- **React Frontend (18.2.0):** Handles the visual workflow builder, node management, and real-time feedback.

API Layer

- **REST API:** Handles CRUD operations, Auth, and Node management.
- **WebSockets:** Handles real-time execution status and live streaming.

Business Logic Layer

- **Workflow Engine:** Validates structure and manages the execution pipeline.
- **AI Module:** Manages Prompts and LLM API integration.
- **Doc Processing:** Handles uploads, embeddings, and vector storage.

Data Layer

- **PostgreSQL:** Stores users, metadata, and history.
- **Vector Database:** Stores document embeddings for semantic search.

3. Low-Level Design (LLD)

3.1 Frontend Architecture

Component Directory Structure

- src/components/
  - WorkflowCanvas.jsx (Main visualization)
  - WorkflowNode.jsx (Node rendering)
  - NodeSettings.jsx (Configuration)
  - WorkflowControls.jsx (Save/Execute)
- src/pages/
  - WorkflowBuilderPage.jsx (Main interface)
- src/context/
  - WorkflowContext.jsx (State)

State Management Flow

The WorkflowContext manages the central state:

- **Data:** Nodes Array, Edges Array, Workflow Metadata (query, response).
- **Actions:** addNode -> updateNode -> executeWorkflow -> updateResponse.

Workflow Execution (Frontend Perspective)

1. **Input Phase:** User types in Query Node -> Stored in State.
2. **Execution Phase:** User clicks Execute -> isExecuting set to true.
3. **Processing:** Simulation or API call (approx 1000ms delay).
4. **Output Phase:** Response generated -> Output Node updates -> isExecuting set to false.

3.2 Backend Architecture

API Endpoints Overview

- POST /api/auth/login -> Authenticate User
- POST /api/workflows -> Create Workflow
- POST /api/workflows/{id}/execute -> Run Workflow Logic
- POST /api/documents/upload -> Process Files

Backend Execution Pipeline

The backend processes a workflow request in four distinct phases:

1. **Validation Phase:** Validate structure -> Check connections -> Validate inputs.
2. **Preparation Phase:** Load config -> Initialize executors -> Prepare data.
3. **Execution Phase (Chain):**
    - User Query Node (Extract text) ->
    - Knowledge Base Node (Search vectors/docs) ->
    - LLM Engine Node (Call AI API with context) ->
    - Output Node (Format result).
4. **Finalization Phase:** Log details -> Store history -> Return summary.

3.3 Database Schema

Key Tables:

- **Users:** id, username, password_hash
- **Workflows:** id, user_id, nodes (JSON), edges (JSON)
- **Executions:** id, workflow_id, input, output, status
- **Documents:** id, file_path, embeddings

4. Data Flow Architecture

4.1 User Query to Response Flow

This describes how a user's action translates to a result on screen.

User Input -> Click Execute -> Keyword Analysis/API Call -> Wait (Processing Delay) -> State Update -> Output Node Rerender

4.2 Backend Execution Flow

This describes the internal server logic during execution.

Receive API Request -> Validate Workflow Structure -> Set Status: RUNNING -> **[Execute Nodes]** -> Set Status: COMPLETED -> Save to DB -> Return JSON Response

5. Technology Stack

Component	Technology	Purpose
Frontend	React 18.2.0, ReactFlow, Tailwind CSS	UI & Visualization
Backend	Python 3.10+, FastAPI	REST API & Logic
Database	PostgreSQL 14+, SQLAlchemy	Data Persistence
AI/LLM	OpenAI SDK, Sentence-Transformers	Intelligence & Embeddings
DevOps	Docker, GitHub Actions	Deployment & CI/CD

6. Deployment Architecture

6.1 Production Traffic Flow

The flow of data in the production environment:

Internet -> CDN (Assets) -> Load Balancer (API Gateway) -> Nginx (Frontend) -> Backend Container (FastAPI) -> Database / Cache

6.2 Docker Strategy

- **Frontend Container:** Node 18 Alpine -> Build React App -> Serve Static Files.
- **Backend Container:** Python 3.10 Slim -> Install Requirements -> Run Uvicorn Server.

7. Future Enhancements

- **Advanced Logic:** Multi-step conditions (If/Else), Loops.
- **Integrations:** Webhooks, Scheduled triggers.
- **Enterprise:** SSO, Audit Logging, RBAC.
- **AI:** Custom model fine-tuning, support for local LLMs.