AI Agent Automation Workflow Spec (Replit Implementation)

# 🎯 Goal

Create an AI agent inside a Replit app that:  
- Accepts natural language input from users  
- Converts it into executable JSON workflows  
- Displays a visual workflow builder UI  
- Supports testing, editing, and saving  
- Integrates with apps (Slack, Airtable, etc.)

# 🔧 System Architecture Overview

User ➝ Frontend (React) ➝ Backend (FastAPI) ➝ GPT-4o ➝ Workflow JSON ➝ Database

# 🧾 Frontend (React)

Pages:  
- /: Prompt input + visual workflow builder  
- /workflows: List of saved workflows  
- /workflow/:id: Edit/view individual workflow  
  
Components:  
- PromptInput.jsx: Chat-like input for workflow commands  
- WorkflowPreview.jsx: Visual representation of steps  
- StepEditor.jsx: Edit individual steps (trigger/action)  
- WorkflowRunner.jsx: Run test data through the workflow

# 🧠 AI Agent Layer (GPT-4o)

System Prompt:  
"You are an AI workflow generator. Your job is to convert user commands into structured JSON-based workflows with clear triggers and steps."  
  
Sample User Input:  
“When a new Calendly booking is made, send a Slack message and update my Notion database.”  
  
Expected Output (JSON):  
{  
 "workflow\_name": "Calendly to Slack + Notion",  
 "trigger": { "app": "Calendly", "event": "New Booking", "params": { "calendar": "default" } },  
 "steps": [  
 { "app": "Slack", "action": "Send Message", "params": { "channel": "#bookings", "message": "New Calendly booking received!" } },  
 { "app": "Notion", "action": "Add Database Entry", "params": { "database\_id": "appointments", "fields": { "Name": "{{booking.name}}", "Email": "{{booking.email}}" } } }  
 ]  
}

# ⚙️ Backend (FastAPI)

Endpoints:  
- POST /generate-workflow: Accepts natural language input, returns JSON  
- POST /run-workflow: Simulates workflow execution  
- GET /workflow/:id: Retrieve stored workflow  
- POST /save-workflow: Save to DB  
  
Backend Flow:  
@app.post("/generate-workflow")  
def generate\_workflow(input\_text: str):  
 gpt\_response = call\_openai(input\_text)  
 return gpt\_response.json()  
  
@app.post("/run-workflow")  
def run\_workflow(workflow: dict):  
 for step in workflow["steps"]:  
 simulate(step)  
 return {"status": "Test Run Complete"}

# 🗃️ Database Structure (Supabase / Firebase)

Tables:  
- workflows: id, user\_id, name, json, created\_at  
- users: id, email, plan, created\_at

# 🔌 Integration Layer (Modular SDK-Style)

Structure:  
/integrations/  
 - slack.py  
 - notion.py  
 - calendly.py  
 - airtable.py  
  
Sample Integration (Slack):  
def send\_message(channel, message, token):  
 url = "https://slack.com/api/chat.postMessage"  
 headers = { "Authorization": f"Bearer {token}" }  
 payload = { "channel": channel, "text": message }  
 return requests.post(url, headers=headers, json=payload)

# 🧪 Workflow Simulation Engine

Mock Test Data (mock\_data.json):  
{ "booking": { "name": "John Smith", "email": "john@example.com" } }  
  
Simulation Logic:  
def simulate\_step(step, mock\_data):  
 for key, value in step["params"].items():  
 if isinstance(value, str) and "{{" in value:  
 step["params"][key] = replace\_template(value, mock\_data)  
 return step

# 🪄 Template Parser

def replace\_template(text, data):  
 matches = re.findall(r"\{\{(.\*?)\}\}", text)  
 for match in matches:  
 keys = match.strip().split('.')  
 val = data  
 for key in keys:  
 val = val.get(key, "")  
 text = text.replace(f"{{{{{match}}}}}", val)  
 return text

# 🧱 Replit Structure

/frontend  
 - App.jsx  
 - components/  
 - index.html  
  
/backend  
 - main.py (FastAPI)  
 - gpt\_agent.py  
 - integrations/  
 - simulation.py  
  
mock\_data.json  
.env (with OPENAI\_API\_KEY, etc.)  
replit.nix or requirements.txt

# ✅ Success Criteria

- GPT converts input to valid JSON  
- Frontend displays each step visually  
- Simulation engine runs mock data  
- Workflows can be saved  
- Modular integrations can be reused  
- Mobile responsive UI