💼 Building a Multi-Turn Job Application Agent Using LangChain & Gemini

Type	@datasciencebrain

This guide walks you through building an intelligent, multi-turn **Job Application Agent** that:

- Takes a job posting and your resume as input
- Summarizes the job requirements
- Tailors your resume for the job

- Generates a personalized cover letter
- Saves everything to a file using LangChain's tool-calling abilities

---

🗂️ Project Structure

```
job-application-agent/
│
├── main.py              # Main agent loop and logic
├── tools.py             # Tools: file read, save, and document generation
├── .env                 # API keys
├── resume.txt           # Your plain text resume
├── job_description.txt  # The job post text (manual paste or scrape)
├── requirements.txt     # Dependencies
└── applications/        # Folder for output resumes and cover letters
```

📦 requirements.txt

```
langchain
langchain-community
langchain-google-genai
python-dotenv
pydantic
```

Install with:

```
pip install -r requirements.txt
```

🔑 Step 1: Setup API Key

Create a `.env` file:

```
GOOGLE_API_KEY="your_google_gemini_key_here"
```

Load it in your code:

```
from dotenv import load_dotenv
load_dotenv()
```

🛠️ Step 2: Tool Implementations (tools.py)

Read Resume or Job Description

```
def read_file(file_path: str) → str:
    with open(file_path, "r", encoding="utf-8") as f:
        return f.read()
```

Save Cover Letter & Tailored Resume

```
from datetime import datetime
import os

def save_application(resume: str, cover_letter: str, job_title: str):
    timestamp = datetime.now().strftime("%Y%m%d_%H%M")
    folder = "applications"
    os.makedirs(folder, exist_ok=True)

    base_filename = os.path.join(folder, f"{job_title.replace(' ', '_')}_{timestam
p}")
    with open(base_filename + "_resume.txt", "w", encoding="utf-8") as f:
        f.write(resume)
    with open(base_filename + "_cover_letter.txt", "w", encoding="utf-8") as f:
        f.write(cover_letter)

    return f"Saved resume and cover letter for '{job_title}'"
```

Wrap as LangChain Tools

```
from langchain.tools import Tool

read_tool = Tool(
    name="read_file",
    func=read_file,
    description="Reads plain text from a given file path"
)

save_tool = Tool(
    name="save_application",
    func=save_application,
    description="Saves tailored resume and cover letter to files"
)

tools = [read_tool, save_tool]
```

🧠 Step 3: Agent Output Schema

```
from pydantic import BaseModel, Field
from typing import List

class ApplicationOutput(BaseModel):
    job_title: str
    tailored_resume: str
    cover_letter: str
```

🤖 Step 4: Agent Setup (main.py)

```
from langchain_google_genai import ChatGoogleGenerativeAI
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.output_parsers import PydanticOutputParser
from langchain.agents import create_tool_calling_agent, AgentExecutor
from tools import tools
from langchain_core.messages import HumanMessage, AIMessage

llm = ChatGoogleGenerativeAI(
    model="gemini-2.5-flash",
    temperature=0.3,
    max_output_tokens=1500,
)

parser = PydanticOutputParser(pydantic_object=ApplicationOutput)

prompt = ChatPromptTemplate.from_messages([
    ("system", """
You are a career assistant that helps tailor resumes and write cover letters for
job applications.
Use the provided resume and job description to:
1. Understand job requirements
2. Modify the resume to align with key skills
3. Generate a professional, personalized cover letter

Always save the result using the save_application tool.

Output only valid JSON in this format:
{format_instructions}
    """),
    ("placeholder", "{chat_history}"),
    ("human", "{query}"),
    ("placeholder", "{agent_scratchpad}"),
]).partial(format_instructions=parser.get_format_instructions())

agent = create_tool_calling_agent(llm=llm, prompt=prompt, tools=tools)

executor = AgentExecutor(agent=agent, tools=tools, verbose=True)

chat_history = []
```

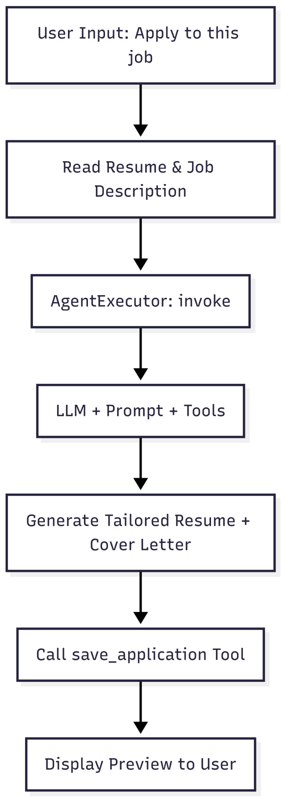
```
while True:
    query = input("You: ")
    if query.lower() in ["exit", "quit"]:
        break

    chat_history.append(HumanMessage(content=query))

    response = executor.invoke({
        "query": query,
        "chat_history": chat_history
    })

    try:
        parsed = parser.parse(response.get("output"))
        print("\n🎯 Job Title:", parsed.job_title)
        print("📄 Cover Letter Preview:\n", parsed.cover_letter[:500], "...")
        chat_history.append(AIMessage(content=parsed.cover_letter))
    except Exception as e:
        print("\n[Error parsing output]:", e)
        print("Raw:", response.get("output"))
```

🧭 Agent Workflow Diagram


User Input: Apply to this job
Read Resume & Job Description
AgentExecutor: invoke
LLM + Prompt + Tools
Generate Tailored Resume + Cover Letter
Call save_application Tool
Display Preview to User

🔄 Example Conversation

```
You: I want to apply to the role described in job_description.txt
🎯 Job Title: Data Analyst
📄 Cover Letter Preview:
Dear Hiring Manager,
I am excited to apply for the Data Analyst position at XYZ Corp...
```

✅ Final Output

The assistant:

- Reads your resume
- Analyzes the job post
- Tailors your resume content
- Writes a human-like cover letter

- Saves everything to the `applications/` folder
