JOMHACK
INSPIRE INNOVATION

PYTHON
BOOTCAMP

www.jomhack.com

AI AGENTS

JOMHACK
INSPIRE INNOVATION

Core Functions

- **Reason:** Thoughts about what to do
- **Action:** Interact with external tools
- Observe results from each actions
- Continue cycle until completing a task

Use Cases

- **Q&A:** Using search tool for current information
- **Automation:** Break down complex task into steps
- **Data Analysis:** Querying databases and processing results

R.A.G

Retrieval:

- Searches and fetches relevant information from external knowledge sources.

Augmented:

- Enriches the original prompt with retrieved information.

Generation:

- Produces the final answer using both the query and augmented context.

BACKEND

Libraries:

- pip install fastapi uvicorn python-dotenv pymongo langchain langchain-google-genai

API:

- MONGODB_ATLAS_CLUSTER_URI
- GOOGLE_API_KEY

Resources:

- https://www.mongodb.com/cloud/atlas
- https://aistudio.google.com/app/apikey

BACKEND

.env:

- GOOGLE_API_KEY=
- GEMINI_MODEL=gemini-2.0-flash
- MONGODB_ATLAS_CLUSTER_URI=

BACKEND

agents/chat_agent.py:

backend > agents > chat_agent.py > ...

```python
import os
import sys

sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(__file__))))

from langchain.agents import create_agent
from utils import llm, tools

SYSTEM_PROMPT = """
## PERSONA
You are a poetic AI assistant that always responds in rhymes.

## TASK
Your task is to assist users with their inquiries while adhering to your poetic nature.
Answer general questions directly without using tools.
ONLY use tools when the user explicitly asks about stored/personal data or posts.

## TOOL CALLING
**IMPORTANT**: Only use tools when the user asks about personal posts, database content, or stored articles.
For general questions, greetings, or casual conversation, answer directly WITHOUT using any tools.

## GUARDRAIL
Always respond in rhymes, no matter the question or task.
Politely refuse to answer anything related to violence, hate speech, or illegal activities.
"""
```

BACKEND

agents/chat_agent.py:

```python
def create_chat_agent():
    """Factory function to create a new chat agent with its own memory."""
    agent = create_agent(
        model=llm,
        tools=tools,
        system_prompt=SYSTEM_PROMPT,
    )

    return agent

# Uncomment below to test the agent directly

if __name__ == "__main__":
    test_agent = create_chat_agent()

    # Test general question (should NOT use database tool)
    print("\n=== Testing general question ===")
    response = test_agent.invoke({
        "messages": [{"role": "user", "content": "Tell me about Malaysia."}]
    })
    print("Response:", response['messages'][-1].content)

    # # Test database question (should use database tool)
    # print("\n=== Testing database question ===")
    # response = test_agent.invoke({
    #     "messages": [{"role": "user", "content": "Tell me about my personal posts"}]
    # })
    # print("Response:", response['messages'][-1].content)
```

BACKEND

agents/__init__.py:

backend > agents > __init__.py > ...

```python
from .chat_agent import create_chat_agent

__all__ = ["create_chat_agent"]
```

BACKEND

utils/llm.py:

backend > utils > llm.py > ...

```python
import os
from dotenv import load_dotenv
from langchain_google_genai import ChatGoogleGenerativeAI

load_dotenv()

api_key = os.getenv("GOOGLE_API_KEY")
model = os.getenv("GEMINI_MODEL")

if not api_key:
    raise ValueError("GOOGLE_API_KEY environment variable is not set")

llm = ChatGoogleGenerativeAI(
    model=model,
    api_key=api_key,
    temperature=0.2,
)
```

BACKEND

utils/tools.py:

backend > utils > tools.py > ...

```python
import os
from langchain_core.tools import tool
from pymongo import MongoClient
from dotenv import load_dotenv

load_dotenv()

@tool
def post_data_from_db(query: str) -> str:
    """
    Fetch posts from MongoDB database.

    ONLY use this tool when the user explicitly asks about:
    - Your personal posts
    - Posts from the database
    - Content you have written or saved
    - Blog posts or articles in your collection

    DO NOT use this tool for:
    - General knowledge questions
    - Greetings or casual conversation
    - Questions about topics not related to stored posts

    Args:
        query: The user's question about posts

    Returns:
        A formatted string of posts from the database or an error message
    """
    try:
        client = MongoClient(os.getenv("MONGODB_ATLAS_CLUSTER_URI"))
        db = client["example_db"]
        collection = db["posts"]

        # Fetch all posts (you can modify this to search by title, content, etc.)
        posts = list(collection.find({}).limit(5))  # Limit to 5 posts for brevity

        if posts:
            # Format the posts nicely
            result = []
            for post in posts:
                post_info = f"Title: {post.get('title', 'N/A')}, Content: {post.get('content', 'N/A')}"
                result.append(post_info)
            return "\n".join(result)
        else:
            return "No posts found in the database."
    except Exception as e:
        return f"Error fetching data: {str(e)}"
    finally:
        client.close()


tools = [post_data_from_db]
```

BACKEND

utils/__init__.py:

backend > utils > __init__.py > ...

```python
from .llm import llm
from .tools import tools

__all__ = [
  "llm",
  "tools"
  ]
```

BACKEND

api/main.py:

backend > api > main.py > ...

```python
from fastapi import FastAPI, HTTPException
from fastapi.middleware.cors import CORSMiddleware
from pydantic import BaseModel, Field
from typing import List, Optional
import sys
import os

# Add parent directory to path to import agents
sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(__file__))))

from agents import create_chat_agent

app = FastAPI(title="Chat API")

# Configure CORS
app.add_middleware(
    CORSMiddleware,
    allow_origins=["http://localhost:5173"],  # React Router dev server
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

```python
class ChatMessage(BaseModel):
    message: str = Field(..., description="User's message to the chatbot")

class ChatResponse(BaseModel):
    response: str = Field(..., description="Bot's response")
    error: Optional[str] = None

@app.get("/api/health")
async def health_check():
    """Health check endpoint"""
    return {"status": "ok"}
```

BACKEND

api/main.py:

```python
@app.post("/api/chat", response_model=ChatResponse)
async def chat(chat_message: ChatMessage):
    """
    Chat endpoint that processes user messages through the AI agent.
    """
    try:
        agent = create_chat_agent()
        # Use messages format as expected by create_agent
        result = agent.invoke({
            "messages": [{"role": "user", "content": chat_message.message}]
        })

        # Extract the last message content from the response
        last_message = result.get("messages", [])[-1] if result.get("messages") else None
        response_text = last_message.content if last_message else "I couldn't generate a response."

        return ChatResponse(
            response=response_text,
            error=None
        )
    except Exception as e:
        raise HTTPException(
            status_code=500,
            detail=f"Error processing message: {str(e)}"
        )

if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

BACKEND

Full Code:

- https://github.com/Bro-chill/full_stack_rag_chatbot