```
!mkdir -p app
!mkdir -p tests
```

```
!pip install fastapi uvicorn python-dotenv requests
```

```
Requirement already satisfied: fastapi in /usr/local/lib/python3.12/dist-packages (0.118.3)
Requirement already satisfied: uvicorn in /usr/local/lib/python3.12/dist-packages (0.38.0)
Requirement already satisfied: python-dotenv in /usr/local/lib/python3.12/dist-packages (1.2.1)
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packages (2.32.4)
Requirement already satisfied: starlette<0.49.0,>=0.40.0 in /usr/local/lib/python3.12/dist-packages (from fastapi) (0.48.0)
Requirement already satisfied: pydantic!=1.8,!=1.8.1,!=2.0.0,!=2.0.1,!=2.1.0,<3.0.0,>=1.7.4 in /usr/local/lib/python3.12/dist-packages (from
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.12/dist-packages (from fastapi) (4.15.0)
Requirement already satisfied: click>=7.0 in /usr/local/lib/python3.12/dist-packages (from uvicorn) (8.3.1)
Requirement already satisfied: h11>=0.8 in /usr/local/lib/python3.12/dist-packages (from uvicorn) (0.16.0)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests) (3.4.4)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests) (3.11)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests) (2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests) (2025.11.12)
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.12/dist-packages (from pydantic!=1.8,!=1.8.1,!=2.0.0,!=2.0.1
Requirement already satisfied: pydantic-core==2.41.4 in /usr/local/lib/python3.12/dist-packages (from pydantic!=1.8,!=1.8.1,!=2.0.0,!=2.0.1,
Requirement already satisfied: typing-inspection>=0.4.2 in /usr/local/lib/python3.12/dist-packages (from pydantic!=1.8,!=1.8.1,!=2.0.0,!=2.0
Requirement already satisfied: anyio<5,>=3.6.2 in /usr/local/lib/python3.12/dist-packages (from starlette<0.49.0,>=0.40.0->fastapi) (4.11.0)
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.12/dist-packages (from anyio<5,>=3.6.2->starlette<0.49.0,>=0.40.0->fas
```

```
%%writefile requirements.txt
fastapi==0.100.0
uvicorn==0.22.0
python-dotenv==1.0.0
requests==2.31.0
```

```
Writing requirements.txt
```

```
%%writefile app/__init__.py
__all__ = ["main", "agent", "tools", "memory"]
```

```
Writing app/__init__.py
```

```
%%writefile app/tools.py
from typing import Optional

# Simple knowledge base
KB = {
    "python": "Python is a high-level, interpreted programming language.",
    "fastapi": "FastAPI is a fast Python framework for building APIs.",
    "uvicorn": "Uvicorn is a lightning-fast ASGI server for Python.",
    "ai": "AI (Artificial Intelligence) is the simulation of human intelligence by machines."
}

def search_tool(query: str) -> Optional[str]:
    """Very simple keyword-based search tool."""
    q = query.lower().strip()

    if q in KB:
        return KB[q]

    for key, value in KB.items():
        if key in q:
            return value

    return None
```

```
Writing app/tools.py
```

```
%%writefile app/memory.py
from collections import deque
from typing import List

class ShortTermMemory:
    """Stores the last few messages to maintain context."""
    def __init__(self, capacity: int = 3):
        self.capacity = capacity
        self.messages = deque(maxlen=capacity)

    def add(self, message: str):
        self.messages.append(message)

    def get_context(self) -> List[str]:
        return list(self.messages)

    def clear(self):
```

```
        self.messages.clear()
```

```
Writing app/memory.py
```

```
%%writefile app/agent.py
from typing import Dict, Any
from .tools import search_tool
from .memory import ShortTermMemory

memory = ShortTermMemory(capacity=3)

def is_factual_question(message: str) -> bool:
    """Simple detection of factual questions using keywords."""
    message = message.lower()
    question_words = ["what", "when", "who", "where", "which", "explain", "define"]

    if any(message.startswith(w) for w in question_words):
        return True

    if "about" in message:
        return True

    return False

def generate_conversational_reply(message: str, context: list) -> str:
    """Very simple conversational response."""
    if "hi" in message.lower() or "hello" in message.lower():
        return "Hey! How can I help you today?"
    return "Got it! You can ask me a factual question like 'What is Python?'."

def agent_respond(message: str) -> Dict[str, Any]:
    memory.add(f"user: {message}")

    factual = is_factual_question(message)

    response = {
        "input": message,
        "factual": factual,
        "tool_used": False,
        "answer": None,
        "context": memory.get_context()
    }
```

```
    if factual:
        tool_answer = search_tool(message)
        if tool_answer:
            response["tool_used"] = True
            response["answer"] = tool_answer
            memory.add(f"agent: {tool_answer}")
            return response

    reply = generate_conversational_reply(message, memory.get_context())
    response["answer"] = reply
    memory.add(f"agent: {reply}")
    return response

def reset_memory():
    memory.clear()
```

```
Writing app/agent.py
```

```
%%writefile app/main.py
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
from .agent import agent_respond, reset_memory

app = FastAPI(title="AI Question-Answer Helper")

class ChatRequest(BaseModel):
    message: str

class ChatResponse(BaseModel):
    input: str
    factual: bool
    tool_used: bool
    answer: str
    context: list

@app.post("/chat", response_model=ChatResponse)
async def chat(req: ChatRequest):
    if not req.message.strip():
        raise HTTPException(status_code=400, detail="Message cannot be empty")

    reply = agent_respond(req.message)
    return ChatResponse(**reply)
```

```
@app.post("/reset-memory")
async def reset():
    reset_memory()
    return {"status": "ok", "message": "Memory cleared"}

# Run with:
# uvicorn app.main:app --reload --port=8000
```

```
Writing app/main.py
```

```
!pip install fastapi uvicorn nest_asyncio pyngrok
```

```
Requirement already satisfied: fastapi in /usr/local/lib/python3.12/dist-packages (0.118.3)
Requirement already satisfied: uvicorn in /usr/local/lib/python3.12/dist-packages (0.38.0)
Requirement already satisfied: nest_asyncio in /usr/local/lib/python3.12/dist-packages (1.6.0)
Requirement already satisfied: pyngrok in /usr/local/lib/python3.12/dist-packages (7.5.0)
Requirement already satisfied: starlette<0.49.0,>=0.40.0 in /usr/local/lib/python3.12/dist-packages (from fastapi) (0.48.0)
Requirement already satisfied: pydantic!=1.8,!=1.8.1,!=2.0.0,!=2.0.1,!=2.1.0,<3.0.0,>=1.7.4 in /usr/local/lib/python3.12/dist-packages (from
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.12/dist-packages (from fastapi) (4.15.0)
Requirement already satisfied: click>=7.0 in /usr/local/lib/python3.12/dist-packages (from uvicorn) (8.3.1)
Requirement already satisfied: h11>=0.8 in /usr/local/lib/python3.12/dist-packages (from uvicorn) (0.16.0)
Requirement already satisfied: PyYAML>=5.1 in /usr/local/lib/python3.12/dist-packages (from pyngrok) (6.0.3)
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.12/dist-packages (from pydantic!=1.8,!=1.8.1,!=2.0.0,!=2.0.1
Requirement already satisfied: pydantic-core==2.41.4 in /usr/local/lib/python3.12/dist-packages (from pydantic!=1.8,!=1.8.1,!=2.0.0,!=2.0.1,
Requirement already satisfied: typing-inspection>=0.4.2 in /usr/local/lib/python3.12/dist-packages (from pydantic!=1.8,!=1.8.1,!=2.0.0,!=2.0
Requirement already satisfied: anyio<5,>=3.6.2 in /usr/local/lib/python3.12/dist-packages (from starlette<0.49.0,>=0.40.0->fastapi) (4.11.0)
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.12/dist-packages (from anyio<5,>=3.6.2->starlette<0.49.0,>=0.40.0->fastap
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.12/dist-packages (from anyio<5,>=3.6.2->starlette<0.49.0,>=0.40.0->fas
```

```
import nest_asyncio
import uvicorn
from fastapi import FastAPI
from threading import Thread

nest_asyncio.apply()

app = FastAPI()

@app.get("/")
def home():
    return {"message": "API running!"}

def run():
```

```
    uvicorn.run(app, host="0.0.0.0", port=8000)

Thread(target=run).start()
```

```
INFO:     Started server process [164]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
```

```
import requests

response = requests.get("http://localhost:8000/")
response.json()
```

```
INFO:     127.0.0.1:37696 - "GET / HTTP/1.1" 200 OK
{'message': 'API running!'}
```

```
from app.agent import agent_respond

print(agent_respond("who is the president of nigeria"))
print(agent_respond("tell me something about football"))
print(agent_respond("how many states are in nigeria"))
print(agent_respond("ok cool, continue"))
```

```
{'input': 'who is the president of nigeria', 'factual': True, 'tool_used': False, 'answer': "Got it! You can ask me a factual question like
{'input': 'tell me something about football', 'factual': True, 'tool_used': False, 'answer': 'Hey! How can I help you today?', 'context': ['
{'input': 'how many states are in nigeria', 'factual': False, 'tool_used': False, 'answer': "Got it! You can ask me a factual question like
{'input': 'ok cool, continue', 'factual': False, 'tool_used': False, 'answer': "Got it! You can ask me a factual question like 'What is Pyth
```

```
import nest_asyncio
import uvicorn
from fastapi import FastAPI
from threading import Thread

nest_asyncio.apply()

app = FastAPI()

from app.agent import agent_respond
from pydantic import BaseModel

class ChatRequest(BaseModel):
```

```
    message: str

@app.post("/chat")
async def chat(req: ChatRequest):
    return agent_respond(req.message)

def run():
    uvicorn.run(app, host="0.0.0.0", port=8000)

Thread(target=run).start()
```

```
INFO:     Started server process [164]
```

```
import requests

response = requests.post(
    "http://127.0.0.1:8000/chat",
    json={"message": "who is the president of nigeria"}
)
response.json()
```

```
INFO:     127.0.0.1:50570 - "POST /chat HTTP/1.1" 404 Not Found
{'detail': 'Not Found'}
```

```
# Install necessary packages
!pip install fastapi uvicorn nest_asyncio requests

# ----------------------------
# Step 1: Define agent functions
# ----------------------------
memory = []

def add_to_memory(user_msg, assistant_msg):
    memory.append({"user": user_msg, "assistant": assistant_msg})
    if len(memory) > 5:
        memory.pop(0)

# Small knowledge base
knowledge_base = {
    "python": "Python is a high-level, interpreted programming language.",
    "fastapi": "FastAPI is a fast Python framework for building APIs.",
    "uvicorn": "Uvicorn is a lightning-fast ASGI server for Python.",
    "ai": "AI (Artificial Intelligence) is the simulation of human intelligence by machines."
```

```
    "ai": "AI (Artificial Intelligence) is the simulation of human intelligence by machines.",
}

def search_tool(query):
    q = query.lower().strip()
    return knowledge_base.get(q, None)

def agent_respond(message):
    # Check factual
    tool_answer = search_tool(message)
    if tool_answer:
        add_to_memory(message, tool_answer)
        return {
            "input": message,
            "factual": True,
            "tool_used": True,
            "answer": tool_answer,
            "context": memory
        }

    # Fallback conversational
    reply = f"Got it! You can ask me a factual question like 'What is Python?'."
    add_to_memory(message, reply)
    return {
        "input": message,
        "factual": False,
        "tool_used": False,
        "answer": reply,
        "context": memory
    }

# ----------------------------
# Step 2: Start FastAPI
# ----------------------------
import nest_asyncio
import uvicorn
from fastapi import FastAPI
from pydantic import BaseModel
from threading import Thread

nest_asyncio.apply()

app = FastAPI(title="AI Question-Answer Helper")

class ChatRequest(BaseModel):
    message: str
```

```
    message: str

@app.on_event("startup")
async def startup_event():
    print(" FastAPI server has started successfully!")

@app.post("/chat")
async def chat(req: ChatRequest):
    return agent_respond(req.message)

def run():
    # Changed port to 8001 to avoid conflicts
    uvicorn.run(app, host="0.0.0.0", port=8001)

# Start FastAPI in background thread
Thread(target=run).start()

# ----------------------------
# Step 3: Test multiple queries
# ----------------------------
import time
import requests

# Wait a few seconds for the server to start
time.sleep(3)

test_queries = [
    "What is Python?",
    "Tell me about AI",
    "Who is the president of Nigeria?",
    "Hello there!",
    "What is FastAPI?"
]

print("===== Testing /chat endpoint =====\n")
for query in test_queries:
    # Updated port in the request URL
    response = requests.post(
        "http://127.0.0.1:8001/chat",
        json={"message": query}
    )
    print(f"User: {query}")
    # Added error handling for responses that don't contain 'answer'
    try:
        print(f"Agent: {response.json()['answer']}\n")
    except KeyError:
```

```
    print(f"Agent: Error - Status Code: {response.status_code}, Detail: {response.json().get('detail', 'No detail provided')}\n")
except Exception as e:
    print(f"Agent: An unexpected error occurred: {e}\n")
```

```
Agent: Got it! You can ask me a factual question like 'What is Python?'.

INFO:     127.0.0.1:50212 - "POST /chat HTTP/1.1" 200 OK
User: Who is the president of Nigeria?
Agent: Got it! You can ask me a factual question like 'What is Python?'.

INFO:     127.0.0.1:50214 - "POST /chat HTTP/1.1" 200 OK
User: Hello there!
Agent: Got it! You can ask me a factual question like 'What is Python?'.

INFO:     127.0.0.1:50216 - "POST /chat HTTP/1.1" 200 OK
User: What is FastAPI?
Agent: Got it! You can ask me a factual question like 'What is Python?'.
```