Here's a **basic example** of creating an AI agent using **LangChain** with **Google Gemini**.

We'll use:

- LangChain
- Google Gemini

1️⃣ Install Dependencies

```
pip install langchain langchain-google-genai google-generativeai
```

2️⃣ Set Your Google API Key

```
export GOOGLE_API_KEY="your_api_key_here"
```

Or in Python:

```
import os
os.environ["GOOGLE_API_KEY"] = "your_api_key_here"
```

3️⃣ Basic Gemini LLM Example

```
from langchain_google_genai import ChatGoogleGenerativeAI

llm = ChatGoogleGenerativeAI(
    model="gemini-1.5-flash",   # or gemini-1.5-pro
    temperature=0.7
)

response = llm.invoke("Explain AI agents in simple terms.")
print(response.content)
```

4️⃣ Create a Simple AI Agent (With Tools)

Now let's create a basic agent that can use tools like math.

```
from langchain.agents import initialize_agent, Tool
from langchain_google_genai import ChatGoogleGenerativeAI
```

```
import math

# Initialize Gemini
llm = ChatGoogleGenerativeAI(
    model="gemini-1.5-flash",
    temperature=0
)

# Define a tool
def calculator_tool(query: str) -> str:
    try:
        return str(eval(query))
    except Exception:
        return "Invalid math expression"

tools = [
    Tool(
        name="Calculator",
        func=calculator_tool,
        description="Useful for solving math expressions."
    )
]

# Create agent
agent = initialize_agent(
    tools,
    llm,
    agent="zero-shot-react-description",
    verbose=True
)

# Run agent
response = agent.invoke("What is (25 * 4) + 10?")
print(response)
```

5️⃣ Modern Way (Recommended – LCEL + AgentExecutor)

```
from langchain.agents import AgentExecutor, create_react_agent
from langchain_google_genai import ChatGoogleGenerativeAI
from langchain import hub

llm = ChatGoogleGenerativeAI(model="gemini-1.5-flash")

# Pull react prompt
prompt = hub.pull("hwchase17/react")

tools = tools  # reuse from above

agent = create_react_agent(llm, tools, prompt)
agent_executor = AgentExecutor(agent=agent, tools=tools, verbose=True)
```

```
agent_executor.invoke({"input": "What is 45 divided by 5?"})
```

□ What This Agent Does

- Receives user query
- Decides if tool is needed
- Uses tool
- Returns final answer

That's a basic working AI agent using **LangChain + Gemini**.

If you'd like, I can next show you:

- □ Memory-enabled agent
- □ Web search agent
- □ FastAPI production-ready agent
- □ Multi-tool agent
- □ Conversational chatbot with Gemini

Just tell me your use case □

ok now give me latest updated way code with memory ,tools examples

Here is the **latest (LangChain v0.2+/v0.3 style)** way to build an AI agent using:

- LangChain
- Google Gemini

This example includes:

- □ Modern agent API (`create_tool_calling_agent`)
- □ Conversation memory
- □ Multiple tools
- □ Clean production-style structure

---

1 □ Install Dependencies

```
pip install -U langchain langchain-google-genai langchain-community
```

2️⃣ Set API Key

```
export GOOGLE_API_KEY="your_api_key_here"
```

or in Python:

```
import os
os.environ["GOOGLE_API_KEY"] = "your_api_key_here"
```

3️⃣ Full Working Example (Agent + Memory + Tools)

```
import os
from langchain_google_genai import ChatGoogleGenerativeAI
from langchain.tools import tool
from langchain.agents import (
    create_tool_calling_agent,
    AgentExecutor
)
from langchain.memory import ConversationBufferMemory
from langchain_core.prompts import ChatPromptTemplate, MessagesPlaceholder


# -------------------------
# 1. Initialize Gemini
# -------------------------
llm = ChatGoogleGenerativeAI(
    model="gemini-1.5-flash",
    temperature=0.3
)


# -------------------------
# 2. Define Tools
# -------------------------

@tool
def calculator(expression: str) -> str:
    """Useful for solving math expressions."""
    try:
        return str(eval(expression))
    except Exception:
        return "Invalid math expression."


@tool
def word_counter(text: str) -> str:
    """Counts number of words in a text."""
    return f"Word count: {len(text.split())}"
```

```
tools = [calculator, word_counter]


# -------------------------
# 3. Create Prompt Template
# -------------------------

prompt = ChatPromptTemplate.from_messages(
    [
        ("system", "You are a helpful AI assistant."),
        MessagesPlaceholder(variable_name="chat_history"),
        ("human", "{input}"),
        MessagesPlaceholder(variable_name="agent_scratchpad"),
    ]
)


# -------------------------
# 4. Add Memory
# -------------------------

memory = ConversationBufferMemory(
    memory_key="chat_history",
    return_messages=True
)


# -------------------------
# 5. Create Agent
# -------------------------

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

agent_executor = AgentExecutor(
    agent=agent,
    tools=tools,
    memory=memory,
    verbose=True
)


# -------------------------
# 6. Run Agent
# -------------------------

response = agent_executor.invoke(
    {"input": "My name is Sagar. What is 45 * 2?"}
)
print(response["output"])

# Memory test
```

```
response = agent_executor.invoke(
    {"input": "What is my name?"}
)
print(response["output"])
```

□ What This Example Shows

□ Tool Calling

Gemini automatically decides when to:

- Use `calculator`
- Use `word_counter`
- Or answer directly

□ Memory

The agent remembers:

```
"My name is Sagar"
```

and can answer:

```
"What is my name?"
```

Below is a **modern production-style RAG Agent using Gemini + FastAPI** built with:

- LangChain
- Google Gemini
- FastAPI

You'll get:

- RAG (Retrieval-Augmented Generation)
- Tool-calling Agent
- Vector DB (FAISS)
- Memory
- FastAPI REST endpoint

---

1 Install Dependencies

```
pip install -U langchain langchain-google-genai langchain-community faiss-cpu
fastapi uvicorn pydantic
```

---

2 Project Structure

```
project/
├── app.py
├── rag_agent.py
├── data.txt
```

---

3 Create RAG Agent (rag_agent.py)

```
import os
from langchain_google_genai import (
    ChatGoogleGenerativeAI,
    GoogleGenerativeAIEmbeddings,
)
from langchain.vectorstores import FAISS
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.document_loaders import TextLoader
from langchain.tools import tool
```

```
from langchain.agents import create_tool_calling_agent, AgentExecutor
from langchain.memory import ConversationBufferMemory
from langchain_core.prompts import ChatPromptTemplate, MessagesPlaceholder

# ----------------------
# 1. Load & Split Data
# ----------------------

loader = TextLoader("data.txt")
documents = loader.load()

splitter = RecursiveCharacterTextSplitter(
    chunk_size=1000,
    chunk_overlap=200
)

docs = splitter.split_documents(documents)

# ----------------------
# 2. Create Vector Store
# ----------------------

embeddings = GoogleGenerativeAIEmbeddings(
    model="models/embedding-001"
)

vectorstore = FAISS.from_documents(docs, embeddings)
retriever = vectorstore.as_retriever(search_kwargs={"k": 3})

# ----------------------
# 3. Create Gemini LLM
# ----------------------

llm = ChatGoogleGenerativeAI(
    model="gemini-1.5-flash",
    temperature=0.3
)

# ----------------------
# 4. RAG Tool
# ----------------------

@tool
def knowledge_base(query: str) -> str:
    """Searches the knowledge base for relevant information."""
    docs = retriever.get_relevant_documents(query)
    return "\n\n".join([doc.page_content for doc in docs])

tools = [knowledge_base]

# ----------------------
# 5. Prompt Template
# ----------------------

prompt = ChatPromptTemplate.from_messages(
    [
        ("system", "You are a helpful assistant. Use the knowledge_base tool
```

```
when needed."),
        MessagesPlaceholder(variable_name="chat_history"),
        ("human", "{input}"),
        MessagesPlaceholder(variable_name="agent_scratchpad"),
    ]
)

# -----------------------
# 6. Memory
# -----------------------

memory = ConversationBufferMemory(
    memory_key="chat_history",
    return_messages=True
)

# -----------------------
# 7. Create Agent
# -----------------------

agent = create_tool_calling_agent(llm, tools, prompt)

agent_executor = AgentExecutor(
    agent=agent,
    tools=tools,
    memory=memory,
    verbose=False
)

def ask_agent(query: str):
    response = agent_executor.invoke({"input": query})
    return response["output"]
```

4️⃣ FastAPI Deployment (app.py)

```
from fastapi import FastAPI
from pydantic import BaseModel
from rag_agent import ask_agent

app = FastAPI(title="Gemini RAG Agent API")

class QueryRequest(BaseModel):
    question: str

@app.post("/ask")
def ask_question(request: QueryRequest):
    answer = ask_agent(request.question)
    return {"answer": answer}
```

5️⃣ Run Server

```
uvicorn app:app --reload
```

Open:

```
http://127.0.0.1:8000/docs
```

FastAPI auto-generates Swagger UI □

□ How This RAG Works

1. User sends question
2. Agent decides to call `knowledge_base`
3. FAISS retrieves top 3 chunks
4. Gemini generates contextual answer
5. Memory keeps conversation history

Let's go **step by step** to install and run **FastAPI** and use it with the RAG agent example I gave. I'll make it beginner-friendly but production-ready.

1□ Install FastAPI & Uvicorn

FastAPI is the web framework; **Uvicorn** is the ASGI server that runs it.

Open your terminal and run:

```
pip install fastapi uvicorn
```

If you also need Pydantic (used for request/response models):

```
pip install pydantic
```

2□ Create Your Project Files

For the RAG agent we made earlier, structure your project like this:

```
my_rag_agent/
│
├── app.py           # FastAPI server
├── rag_agent.py     # Agent code with Gemini + RAG
├── data.txt         # Your knowledge base content
```

- `data.txt` can contain any text your RAG agent will search.
- `rag_agent.py` contains the code for LLM + FAISS + tools.
- `app.py` will expose a REST API.

---

3 FastAPI Server (`app.py`)

Here's a minimal working version using the RAG agent:

```
from fastapi import FastAPI
from pydantic import BaseModel
from rag_agent import ask_agent  # import your agent function

app = FastAPI(title="Gemini RAG Agent API")

class QueryRequest(BaseModel):
    question: str

@app.post("/ask")
def ask_question(request: QueryRequest):
    """Endpoint to ask a question to the RAG agent"""
    answer = ask_agent(request.question)
    return {"answer": answer}
```

---

4 Run FastAPI

In your terminal, run:

```
uvicorn app:app --reload
```

Explanation:

- `app` → the filename `app.py`
- `--reload` → auto-reloads server on code changes
- Server will run at `http://127.0.0.1:8000`

---

5 Test the API

Open your browser:

```
http://127.0.0.1:8000/docs
```

- FastAPI automatically generates Swagger UI.
- You'll see `/ask` endpoint.

- Click **Try it out**, enter JSON like:

```
{
  "question": "What is the capital of France?"
}
```

- Press **Execute**, and the Gemini RAG agent will respond.

---

6 Example Using Python Requests

```
import requests

url = "http://127.0.0.1:8000/ask"
data = {"question": "How many words are in 'FastAPI is amazing'?"}

response = requests.post(url, json=data)
print(response.json())
```