Houston Community College

Personal Wellness Agent
AI-Agent using RAG

Author: Trevon Woods
Team Members: Jiri Musil, Brian Martinez

Data Science - ITAI - 2377

Vishwa Rao

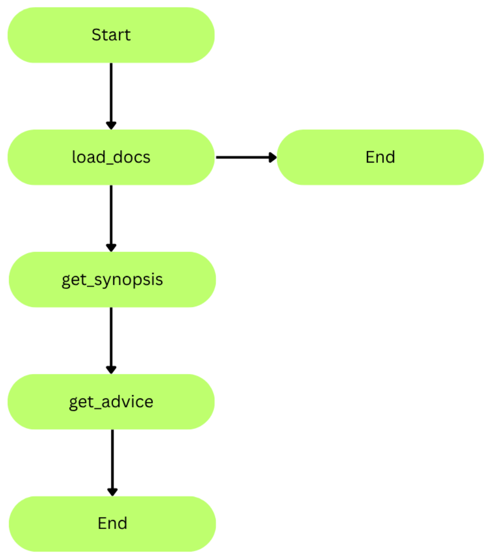
Due: 08/08/2025

Acknowledgements

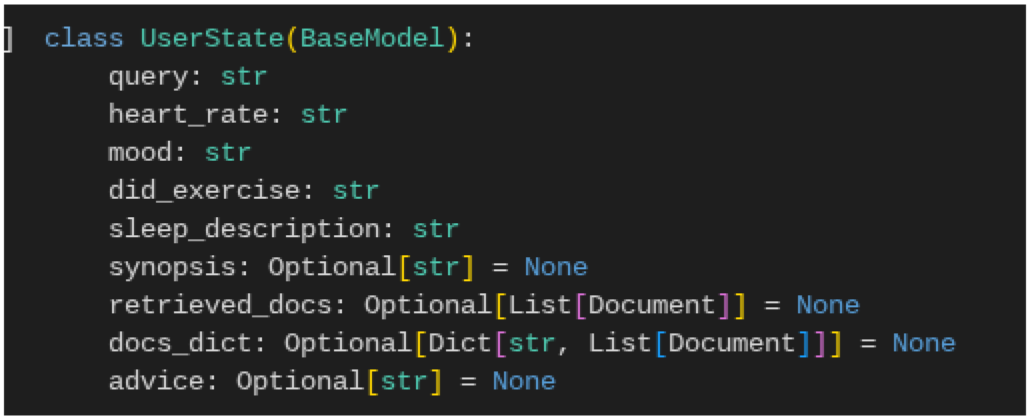
I want to thank the members of my group that communicated and did their parts. Participating members for this assignment were Jiri Musil and Brian Martinez. They handled the data collection from numerous sources and compiled everything into neat organized folders. This allowed me to efficiently finish coding and testing the agent. This was an exciting project and experience.

Key Frameworks, Libraries, API's Used

In this assignment I used a couple key frameworks and one key api and library. The frameworks I used were LangChain and LangGraph. LangChain is an open-source, modular framework for building applications powered by large language models. It's designed for chaining prompts, integrating tools, accessing data sources, and building linear task pipelines across Python or JavaScript. It has a vast ecosystem with pre-built integrations and strong community support. LangGraph is a graph-based, stateful orchestration framework built on top of LangChain, used for complex, dynamic, and long-running agent workflows. I took advantage of LangChains packages and tools to load documents, preprocess the documents, create embeddings, and implement the vector store to store embeddings. I also used LangChain to enable usage of the OpenAI API so I could use the GPT-4o model for text generation and understanding. LangGraph was used to create the workflow of the agent.



I also made use of the Pydantic library to handle the agent’s state. It is a Python library for data validation and settings management that uses Python type hints to enforce structure and correctness. It lets me define classes that automatically parse input data, validate the data against the specified types and constraints, convert types when possible, and it gives helpful error messages when validation fails.



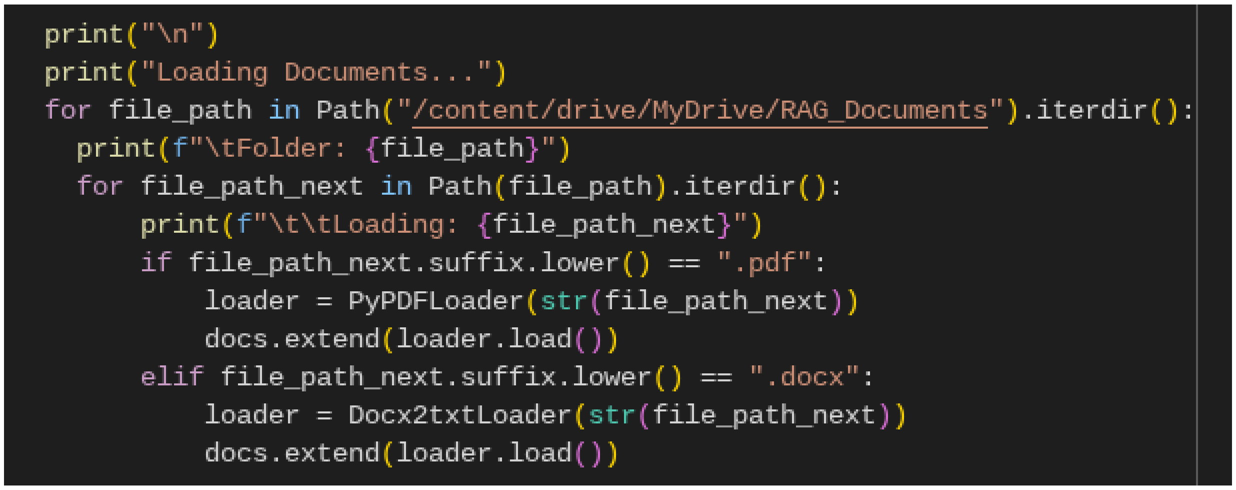
```
] class UserState(BaseModel):
      query: str
      heart_rate: str
      mood: str
      did_exercise: str
      sleep_description: str
      synopsis: Optional[str] = None
      retrieved_docs: Optional[List[Document]] = None
      docs_dict: Optional[Dict[str, List[Document]]] = None
      advice: Optional[str] = None
```

Deliverable Completions

For the midterm my team and I specified the features that will be implemented for the agent. All of the non-bonus features were implemented successfully. I didn’t get around to implementing extraction of wearable trackable data through Apple HealthKit or Oura Ring APIs. That will be one of many future additions to our agent’s capabilities.

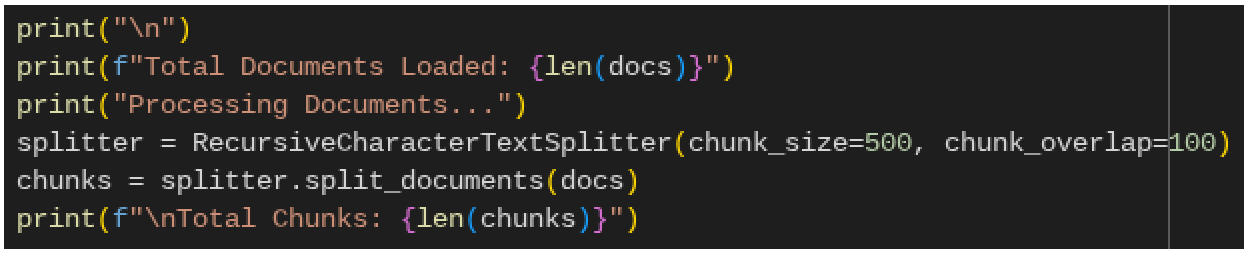
Key features implemented:

1. Load raw data



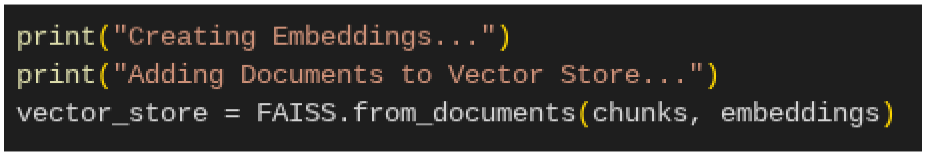
```
print("\n")
print("Loading Documents...")
for file_path in Path("/content/drive/MyDrive/RAG_Documents").iterdir():
  print(f"\tFolder: {file_path}")
  for file_path_next in Path(file_path).iterdir():
      print(f"\t\tLoading: {file_path_next}")
      if file_path_next.suffix.lower() == ".pdf":
          loader = PyPDFLoader(str(file_path_next))
          docs.extend(loader.load())
      elif file_path_next.suffix.lower() == ".docx":
          loader = Docx2txtLoader(str(file_path_next))
          docs.extend(loader.load())
```

2. Pre-RAG Data Preprocessing



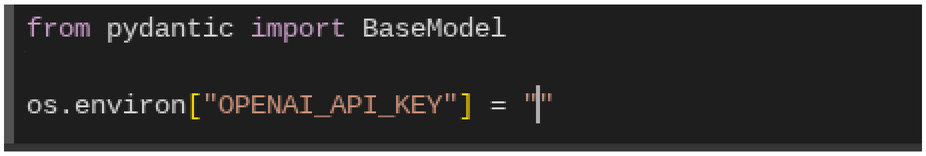
```python
print("\n")
print(f"Total Documents Loaded: {len(docs)}")
print("Processing Documents...")
splitter = RecursiveCharacterTextSplitter(chunk_size=500, chunk_overlap=100)
chunks = splitter.split_documents(docs)
print(f"\nTotal Chunks: {len(chunks)}")
```

3. RAG Vector Store Setup
4. Load preprocessed data to vector store (creates embeddings before loading)



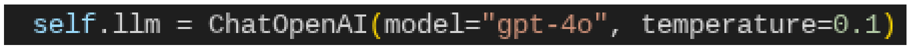
```python
print("Creating Embeddings...")
print("Adding Documents to Vector Store...")
vector_store = FAISS.from_documents(chunks, embeddings)
```

5. LLM API setup for prompting, execution, and generation



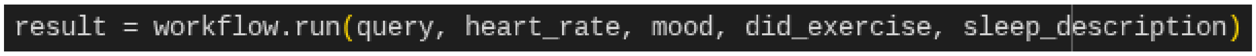
```python
from pydantic import BaseModel

os.environ["OPENAI_API_KEY"] = ""
```



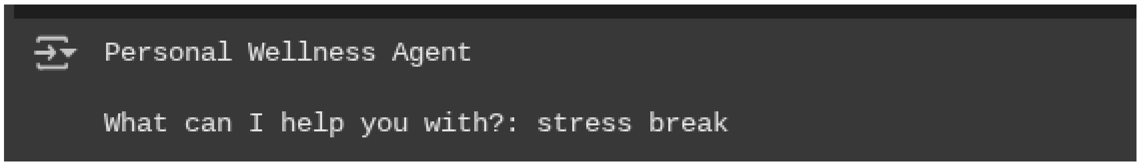
```python
self.llm = ChatOpenAI(model="gpt-4o", temperature=0.1)
```

6. Agent wake up



```python
result = workflow.run(query, heart_rate, mood, did_exercise, sleep_description)
```

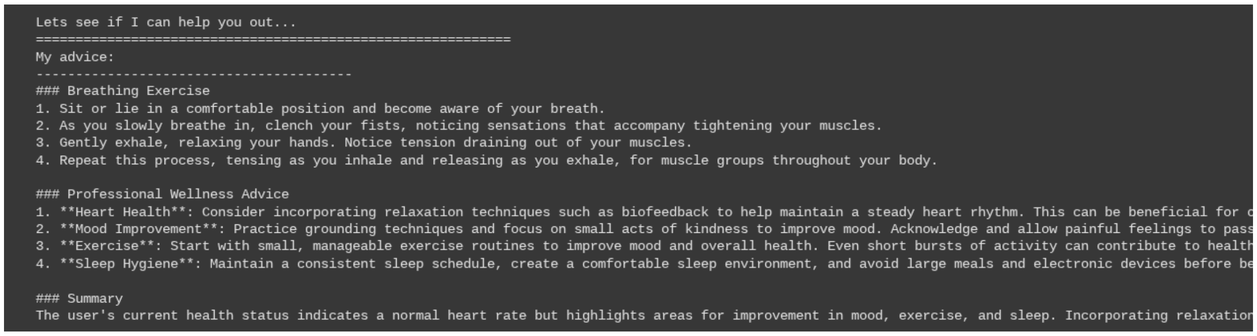
7. User input



```
Personal Wellness Agent

What can I help you with?: stress break
```

8. Generate output to a text file or to terminal for viewing



```
Lets see if I can help you out...
============================================================
My advice:
----------------------------------------
### Breathing Exercise
1. Sit or lie in a comfortable position and become aware of your breath.
2. As you slowly breathe in, clench your fists, noticing sensations that accompany tightening your muscles.
3. Gently exhale, relaxing your hands. Notice tension draining out of your muscles.
4. Repeat this process, tensing as you inhale and releasing as you exhale, for muscle groups throughout your body.

### Professional Wellness Advice
1. **Heart Health**: Consider incorporating relaxation techniques such as biofeedback to help maintain a steady heart rhythm. This can be beneficial for c
2. **Mood Improvement**: Practice grounding techniques and focus on small acts of kindness to improve mood. Acknowledge and allow painful feelings to pass
3. **Exercise**: Start with small, manageable exercise routines to improve mood and overall health. Even short bursts of activity can contribute to health
4. **Sleep Hygiene**: Maintain a consistent sleep schedule, create a comfortable sleep environment, and avoid large meals and electronic devices before be

### Summary
The user's current health status indicates a normal heart rate but highlights areas for improvement in mood, exercise, and sleep. Incorporating relaxation
```

If you look at feature 7 you'll see where I prompted the agent with the phrase "stress break". As you can see the generated output incorporates the breathing exercise. Also from the generated output you will see that the agent gives numbered advice snippets based on the users query and health data.

Challenges

I had a few challenges while coding this agent. In the beginning, creating a consistent langgraph flow for the agent was tricky. Many times I had to redo the whole thing. After a few youtube videos and research online I was able to come up with an efficient flow. Prompting was also a big challenge. I initially thought prompting would be easy but if you aren't explicit in your prompt instructions, the LLM will start hallucinating or generating outlandish things. I wish I had screenshots to show. Some of the generated responses in the beginning were comical.