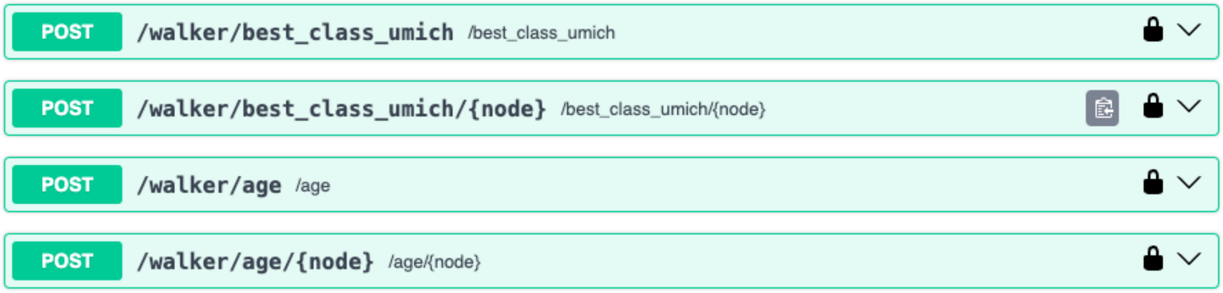
Part 1

- Updated `server.jac` with 2 additional walkers - also in server-part1.jac

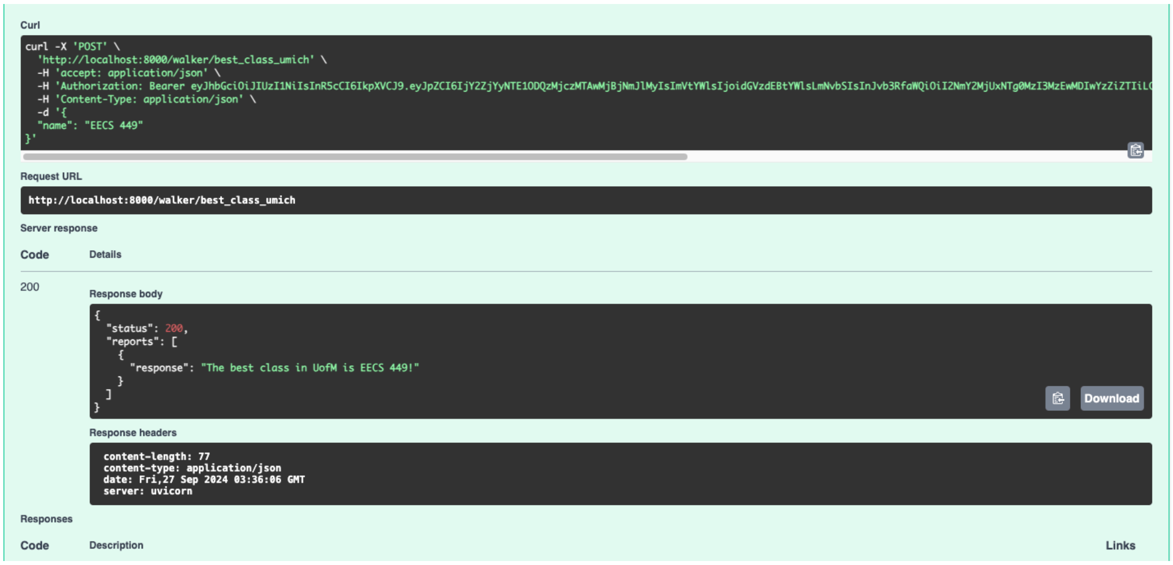
```
walker interact {
   can return_message with `root entry {
       report {
           "response": "Hello, world!"
       };
   }
}

walker interact_with_body {
   has name: str;

   can return_message with `root entry {
       report {
           "response": "Hello, " + self.name + "!"
       };
   }
}

walker best_class_umich {
   has name: str;

   can return_message with `root entry {
       report {
           "response": "The best class in UofM is " + self.name + "!"
       };
   }
}

walker age {
   has age: str;

   can return_message with `root entry {
       report {
           "response": "Feeling like " + self.age + "!"
       };
   }
}
```

    -

- Screenshot of the Swagger documentation showing the new walkers you implemented


POST
/walker/best_class_umich /best_class_umich
POST
/walker/best_class_umich/{node} /best_class_umich/{node}
POST
/walker/age /age
POST
/walker/age/{node} /age/{node}

- Screenshot of the response from the API when you call the new walkers



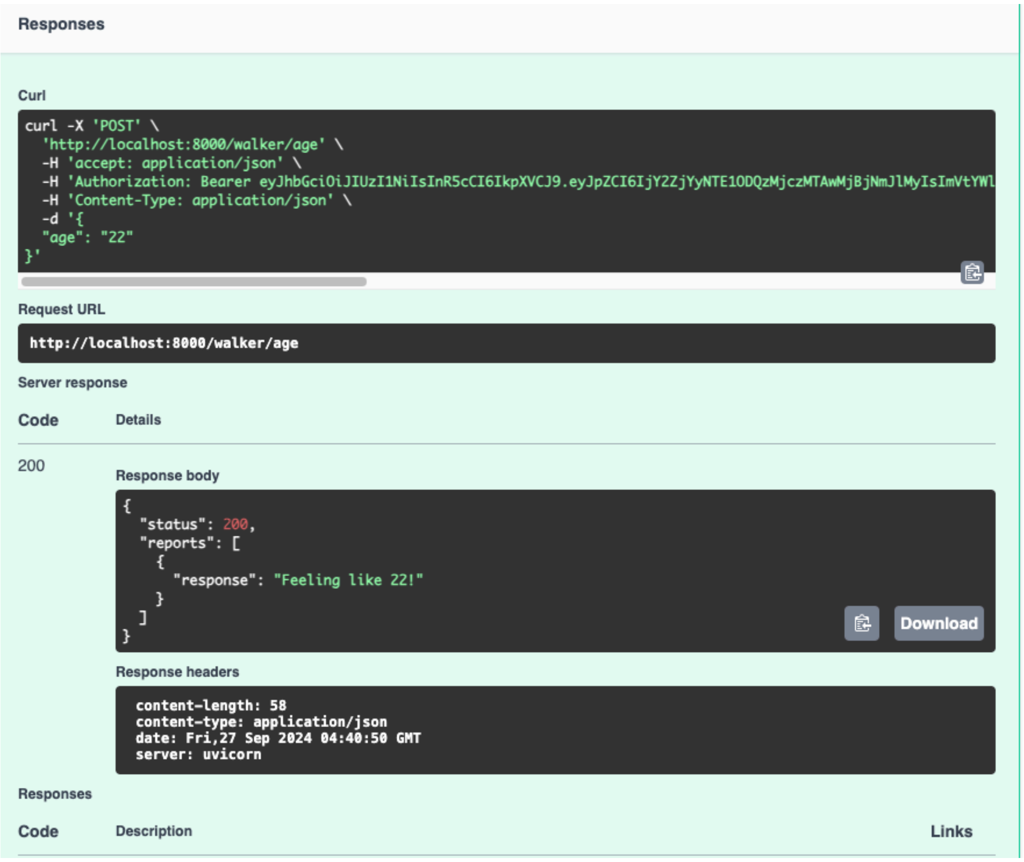
```
curl -X 'POST' \
  'http://localhost:8000/walker/best_class_umich' \
  -H 'accept: application/json' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY2ZjYyNTE1ODQzMjczMTAwMjBjNmJlMyIsImVtYWlsIjoidGVzdEBtYWlsLmNvbSIsInJvb3RfaWQiOiI2NmY2MjUxNTg0MzI3MzEwMDIwYzZiZTIiL
  -H 'Content-Type: application/json' \
  -d '{
  "name": "EECS 449"
}'
```

```
http://localhost:8000/walker/best_class_umich
```

```
{
  "status": 200,
  "reports": [
    {
      "response": "The best class in UofM is EECS 449!"
    }
  ]
}
```

```
content-length: 77
content-type: application/json
date: Fri,27 Sep 2024 03:36:06 GMT
server: uvicorn
```

    -

- 
Responses

Curl

```
curl -X 'POST' \
  'http://localhost:8000/walker/age' \
  -H 'accept: application/json' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY2ZjYyNTE1ODQzMjczMTAwMjBjNmJlMyIsImVtYWl
  -H 'Content-Type: application/json' \
  -d '{
  "age": "22"
}'
```

Request URL

```
http://localhost:8000/walker/age
```

Server response

Code	Details
200	Response body

```
{
  "status": 200,
  "reports": [
    {
      "response": "Feeling like 22!"
    }
  ]
}
```

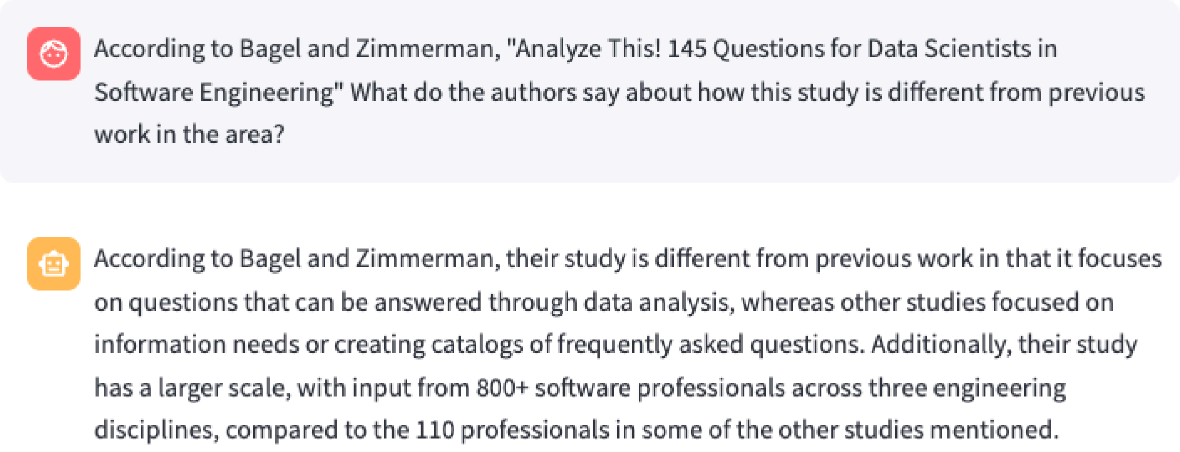
Response headers

```
 content-length: 58
 content-type: application/json
 date: Fri,27 Sep 2024 04:40:50 GMT
 server: uvicorn
```

Responses

Part 2

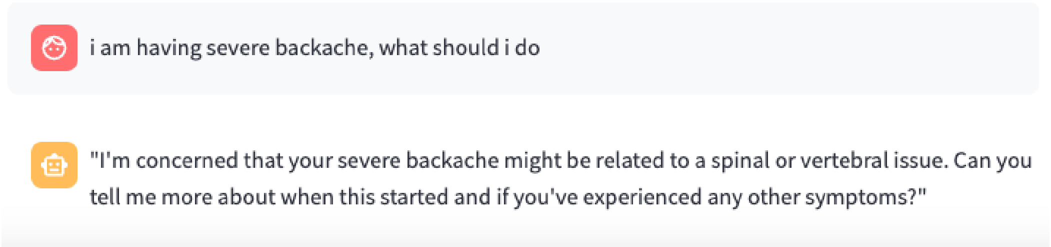
- Added Bagel and Zimmerman, "Analyze This! 145 Questions for Data Scientists in Software Engineering" as custom document, and here’s the chatbot answer



According to Bagel and Zimmerman, "Analyze This! 145 Questions for Data Scientists in Software Engineering" What do the authors say about how this study is different from previous work in the area?

According to Bagel and Zimmerman, their study is different from previous work in that it focuses on questions that can be answered through data analysis, whereas other studies focused on information needs or creating catalogs of frequently asked questions. Additionally, their study has a larger scale, with input from 800+ software professionals across three engineering disciplines, compared to the 110 professionals in some of the other studies mentioned.

- Using Ollama - clinical_medicine.pdf

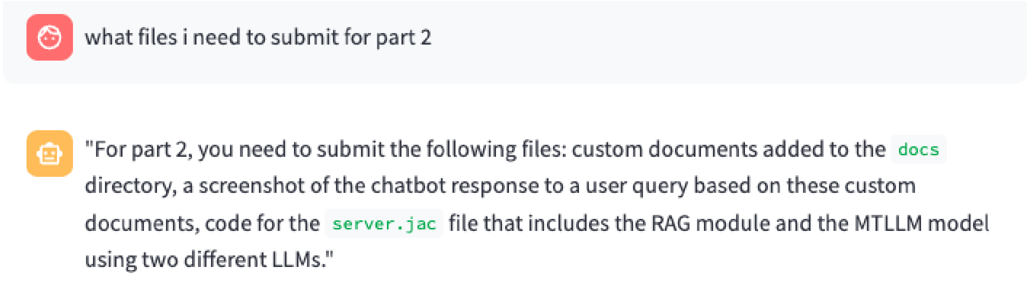


i am having severe backache, what should i do

"I'm concerned that your severe backache might be related to a spinal or vertebral issue. Can you tell me more about when this started and if you've experienced any other symptoms?"

    -

- Using openai - 449_part2_spec.pdf



what files i need to submit for part 2

"For part 2, you need to submit the following files: custom documents added to the `docs` directory, a screenshot of the chatbot response to a user query based on these custom documents, code for the `server.jac` file that includes the RAG module and the MTLLM model using two different LLMs."

    -

- Part 2 code - also in server-part2.jac

```
# ------------For part 2------------
# First LLM: For using OpenAI
# import:py from mtllm.llms {OpenAI}
# glob llm = OpenAI(model_name='gpt-3.5-turbo');

# Second LLM: For using Ollama
import:py from mtllm.llms {Ollama}
glob llm = Ollama(model_name='llama3.1');

import:jac from rag {RagEngine}
glob rag_engine:RagEngine = RagEngine();

node Session {
   has id: str;
   has chat_history: list[dict];
   has status: int = 1;

   can 'Respond to message using chat_history as context and agent_role as the goal of the agent'
   llm_chat(
       message:'current message':str,
       chat_history: 'chat history':list[dict],
       agent_role:'role of the agent responding':str,
       context:'retrieved context from documents':list
   ) -> 'response':str by llm();
}
```

```
walker interact {
   has message: str;
   has session_id: str;

   can init_session with `root entry {
        visit [-->](`?Session)(?id == self.session_id) else {
           session_node = here ++> Session(id=self.session_id, chat_history=[],
status=1);
           print("Session Node Created");

           visit session_node;
       }
   }

   can chat with Session entry {
       here.chat_history.append({"role": "user", "content": self.message});
       data = rag_engine.get_from_chroma(query=self.message);
       response = here.llm_chat(
           message=self.message,
           chat_history=here.chat_history,
           agent_role="You are a conversation agent designed to help users with their
queries based on the documents provided",
           context=data
       );

       here.chat_history.append({"role": "assistant", "content": response});

       report {"response": response};
   }
}
```

Part 3

- I added a clarification state to the enum, the state represent that the chatbot requires additional information to accurately understand the users' query. This can occur when user's query is vague or unreadable, or the chatbot's processing logic fails to determine the user's intent

  ```
  enum ChatType {
     RAG : 'Need to use Retrievable information in specific documents to respond'
  = "RAG",
     QA : 'Given context is enough for an answer' = "user_qa",
  ```

```
   Clarification : 'Requesting more information or details from the user' = "clarification"
}
```

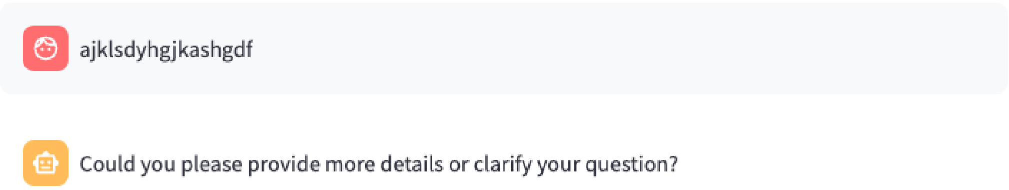
- Custom code node -
    - The respond with infer entry block includes a method ask_for_clarification that constructs a response prompting the user for more details. The response is generated as: "Could you please provide more details or clarify your question?”

```
node ClarificationChat :Chat: {
   has chat_type: ChatType = ChatType.Clarification;

   can respond with infer entry {
       can 'Request additional information from the user for better understanding'
       ask_for_clarification(message:'current message':str) -> 'response':str {
           response = "Could you please provide more details or clarify your question?";
           return response;
       }

       here.response = ask_for_clarification(here.message);

       report {
           "response": here.response
       };
   }
}
```

- Custom state to the routing
    - The route method remains the same, but it now includes my CustomChat in the routing process. The classification variable will determine the appropriate chat node to visit based on the user’s query. It’s added after QAchat, so it will be visit after QA failed to generate an response with llm

```
   can init_router with `root entry {
       visit [-->](`?Router) else {
           router_node = here ++> Router();
           router_node ++> RagChat();
           router_node ++> QAChat();
           router_node ++> ClarificationChat();
           visit router_node;
       }
   }
```

-
- screenshot of a conversation with your chatbot showing you custom dialogue routing in action.



ajklsdyhgjkashgdf

Could you please provide more details or clarify your question?

-
- Code - Also in server-part3.jac

```
import:py from mtllm.llms {Ollama}
glob llm = Ollama(model_name='llama3.1');

import:jac from rag {RagEngine}
glob rag_engine:RagEngine = RagEngine();


walker interact {
   has message: str;
   has session_id: str;

   can init_session with `root entry {
        visit [-->](`?Session)(?id == self.session_id) else {
           session_node = here ++> Session(id=self.session_id, chat_history=[],
status=1);
           print("Session Node Created");

           visit session_node;
       }
   }
}

node Session {
   has id: str;
   has chat_history: list[dict];
   has status: int = 1;

   can 'Respond to message using chat_history as context and agent_role as the
goal of the agent'
   llm_chat(
       message:'current message':str,
       chat_history: 'chat history':list[dict],
       agent_role:'role of the agent responding':str,
       context:'retrieved context from documents':list
   ) -> 'response':str by llm();
   # added for part 3
   can chat with interact entry {
       self.chat_history.append({"role": "user", "content": here.message});
       response = infer(message=here.message, chat_history=self.chat_history)
spawn root;
       self.chat_history.append({"role": "assistant", "content":
response.response});

       report {
           "response": response.response
       };
   }
}


enum ChatType {
   RAG : 'Need to use Retrievable information in specific documents to respond'
= "RAG",
   QA : 'Given context is enough for an answer' = "user_qa",
   Clarification : 'Requesting more information or details from the user' =
"clarification"
}

node Router {
   can 'route the query to the appropriate task type'
   classify(message:'query from the user to be routed.':str) -> ChatType by
llm(method="Reason", temperature=0.0);
}

node Chat {
   has chat_type: ChatType;
}

walker infer {
   has message:str;
   has chat_history: list[dict];

   can init_router with `root entry {
       visit [-->](`?Router) else {
           router_node = here ++> Router();
           router_node ++> RagChat();
           router_node ++> QAChat();
           router_node ++> ClarificationChat();
           visit router_node;
       }
   }
   can route with Router entry {
       classification = here.classify(message = self.message);
       visit [-->](`?Chat)(?chat_type==classification);
   }
}

node RagChat :Chat: {
   has chat_type: ChatType = ChatType.RAG;

   can respond with infer entry {
       can 'Respond to message using chat_history as context and agent_role as
the goal of the agent'
       respond_with_llm(   message:'current message':str,
                   chat_history: 'chat history':list[dict],
                   agent_role:'role of the agent responding':str,
                   context:'retirved context from documents':list
                       ) -> 'response':str by llm();
       data = rag_engine.get_from_chroma(query=here.message);
       here.response = respond_with_llm(here.message, here.chat_history, "You
are a conversation agent designed to help users with their queries based on the
documents provided", data);
   }
}

node QAChat :Chat: {
   has chat_type: ChatType = ChatType.QA;

   can respond with infer entry {
       can 'Respond to message using chat_history as context and agent_role as
the goal of the agent'
       respond_with_llm(   message:'current message':str,
           chat_history: 'chat history':list[dict],
           agent_role:'role of the agent responding':str
               ) -> 'response':str by llm();
       here.response = respond_with_llm(here.message, here.chat_history,
agent_role="You are a conversation agent designed to help users with their
queries");
```

```
    }
}

node ClarificationChat :Chat: {
    has chat_type: ChatType = ChatType.Clarification;

    can respond with infer entry {
        can 'Request additional information from the user for better understanding'
        ask_for_clarification(message:'current message':str) -> 'response':str {
            response = "Could you please provide more details or clarify your question?";
            return response;
        }

        here.response = ask_for_clarification(here.message);

        report {
            "response": here.response
        };
    }
}
```