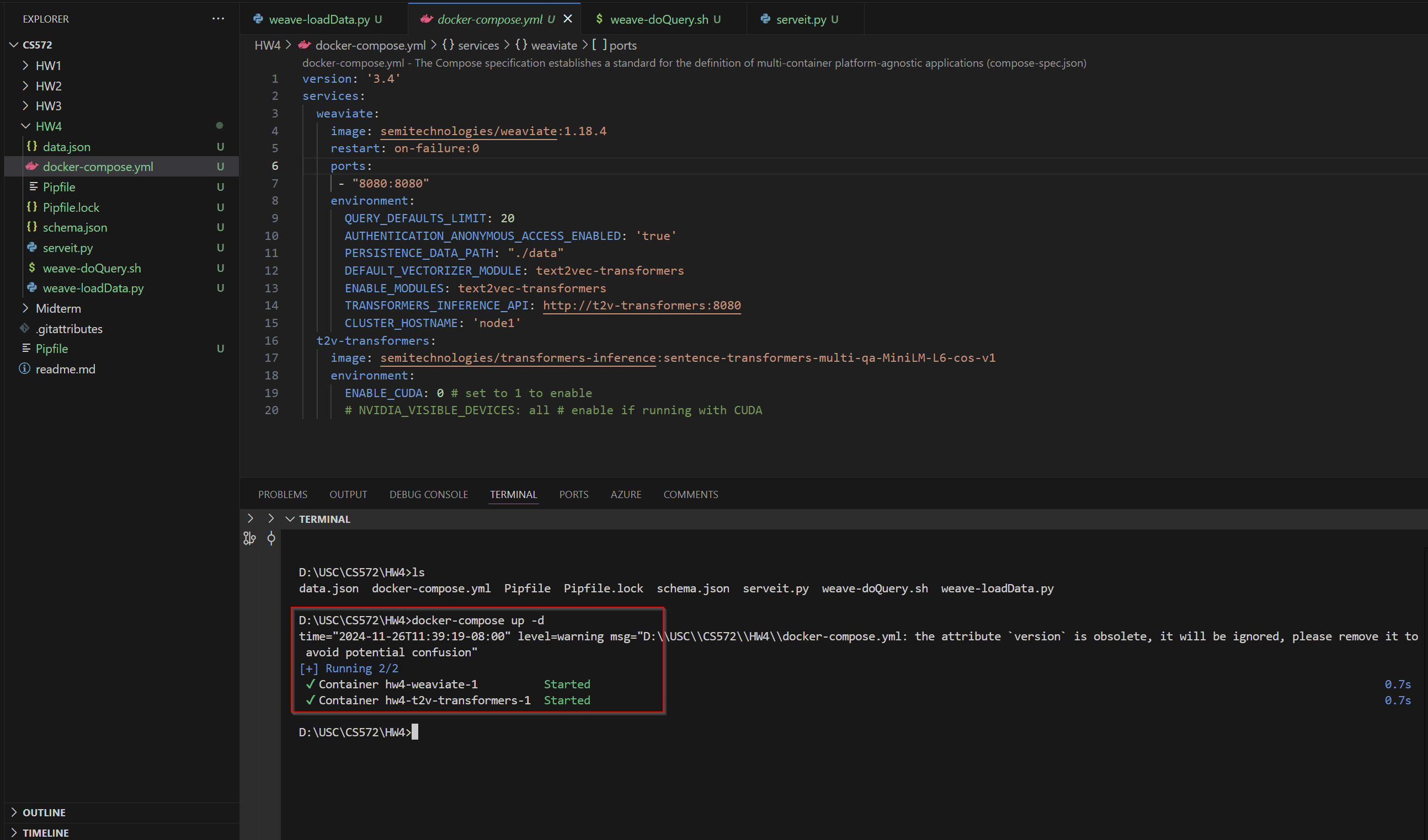
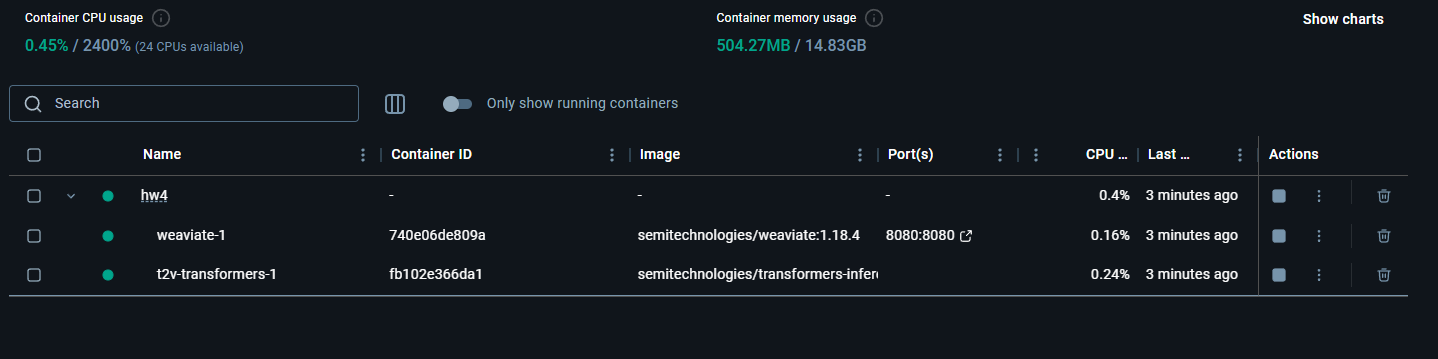
# Question 1:

1. Installing Weaviate and a vectorizer module  
     
   
2. Loading data to search for  
   A screenshot of a computer

   Description automatically generated
3. Querying our vectorized data  
   A screen shot of a computer

   Description automatically generated  
   Loading my own data:  
   A screen shot of a computer program

   Description automatically generated  
   Loading the data:  
   A screenshot of a computer program

   Description automatically generated  
   Querying the data with two brands:  
   A screenshot of a computer

   Description automatically generated

# Question 2:

PDF used: <https://apps.irs.gov/app/vita/content/globalmedia/4491_income.pdf>  
A document with a questionnaire

Description automatically generated with medium confidence

Using the file:

A screenshot of a computer

Description automatically generated

Single query:  
A screenshot of a computer

Description automatically generated

More queries:

A screenshot of a computer

Description automatically generated

# Question 3:

Huggingface space used-> <https://huggingface.co/spaces/MuntasirHossain/RAG-PDF-Chatbot>

Pdf uploaded -> <https://apps.irs.gov/app/vita/content/globalmedia/4491_income.pdf>

A screenshot of a chat

Description automatically generated

* Describe the RAG steps involved (ie. the Python calls) - you don't need to do this in exhaustive detail, but you DO need to describe the workflow (that starts with ingesting the input, ending with output generation)
  + Functions in the files.py for RAG
    - def initialize\_database(list\_file\_obj, progress=gr.Progress()) / def create\_db(splits):
      * This function generates vector embeddings for the document chunks using HuggingFaceEmbeddings
      * It also creates a vector database using FAISS, storing the embeddings for efficient similarity retrieval.
    - def initialize\_LLM(llm\_option, llm\_temperature, max\_tokens, top\_k, vector\_db, progress=gr.Progress()): / def initialize\_llmchain(llm\_model, temperature, max\_tokens, top\_k, vector\_db, progress=gr.Progress())
      * Helps choose a language model (e.g., Meta-Llama or Mistral) via the Gradio interface.
      * Helps configure model parameters (temperature, max tokens, top-k sampling) through sliders.
      * It uses HuggingFaceEndpoint to integrate with the selected LLM and creates a ConversationalRetrievalChain with a retriever from the vector database and a memory buffer for chat history.
    - def conversation(qa\_chain, message, history): / def format\_chat\_history(message, chat\_history):
      * This function submits questions via Gradio interface and retrieve relevant document chunks from the vector database as context for LLM.
      * It generates a response using the retrieved context and include source for transparency.
* Describe how the UI works (what calls do whatwhat) - the file component, input, output. The UI is generated via Gradio [https://www.gradio.app/] or Streamlit [https://streamlit.io/], so this exercise is to get you familiar with their basics :)
  + Following functions are used in the code to create UI:
    - gr.Files, this is used to upload PDFs, it is interactive and triggers the initialize database function, and status is updated in the textbox
    - gr.radio and gr.Slider is used to select the parameters of the charbot. The radio button is used to select between the models and the slider is used to adjust parameters like temperature, Max tokens and top-k.
    - gr.Textbox, gr.Chatbot and gr.Accordion are used to create the interactive chat elements, and update and display the chat
    - Functions like gr.Row and gr.Columns are used to create the layout of the whole UI