1Q AI/ML Quiz

In this project we will see a very simple implementation of a Quiz or in more simple terms a prompt in the form of a question. The User will answer the prompt to the best of their knowledge. Based on the answer the User receives a feedback and the rating for the answer. I have **Hosted** it locally.

This is a very basic Quiz which can and will be developed to its full potential as an AI/ML quiz bot. This was implemented using a very simple frontend made with ReactJS, HTML and CSS. Also a Web Socket connecting the client and the server. I have written the backend using python.
The aim of the project is to utilize a ML model or a fine tuned LLM to generate Questions based on the topic and difficulty level the user chooses. The data however can be fed as documents (pdf) or in the form of External URL's. Please note the project you are about to view is in its early stages.

Mainly I have Used Uvicorn, as the asynchronous interface for the FastAPI and the system.

- Uvicorn is built on asynchronous programming principles, which means it can handle many requests concurrently without blocking. This makes it well-suited for handling high-throughput applications or applications that need to handle multiple tasks at the same time, such as web servers that handle lots of API calls, or WebSocket communication.

- FastAPI is designed to work asynchronously using Python's `async` and `await` keywords, and Uvicorn is the server that can run such applications efficiently.

For the purpose of the ASSIGNMENT and its deliverables I have utilized:

- **FastAPI:** The FastAPI framework is used for building the web application. It's known for being fast, intuitive, and modern, especially for building APIs.

- **WebSocket:** WebSockets are used for full-duplex communication between the client (browser) and server. This allows real-time communication without constantly polling the server.

- **HTMLResponse:** This allows us to send HTML as a response in FastAPI.

- **Transformers:** This library by Hugging Face is used for natural language processing (NLP) tasks. It provides pre-trained models that can be fine-tuned for various tasks like text generation, question answering, etc.

- **PyMuPDF (fitz):** This library is used to extract text from PDFs. It allows us to read the contents of the PDF, including text and metadata.

- **SentenceTransformers:** This is a library to convert sentences into vector embeddings, which can then be compared using cosine similarity to evaluate answers.

Lets look at the two models used:

- **SentenceTransformer:** The **all-MiniLM-L6-v2** model is a smaller, faster version of the Sentence-BERT models that is optimized for producing high-quality sentence embeddings. These embeddings are used to measure the semantic similarity between the user's answer and the expected answer in the **evaluate_answer** function.

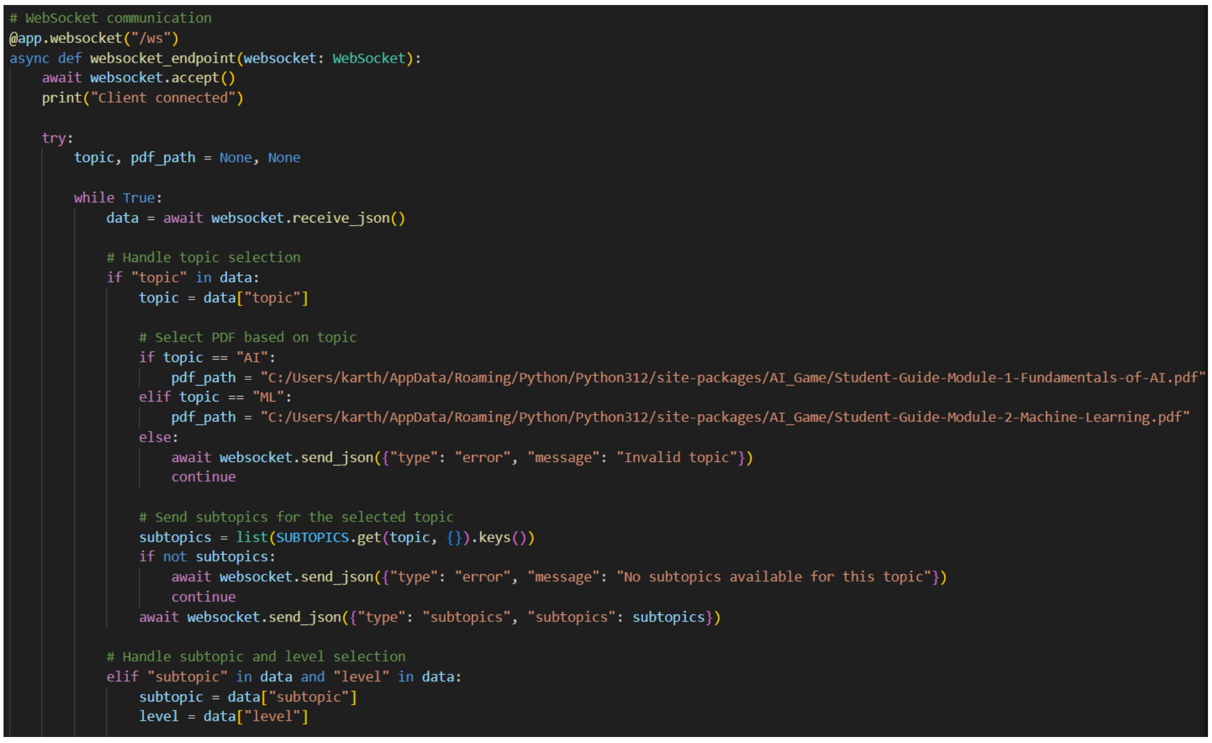
- **T5Tokenizer & T5ForConditionalGeneration:**

    - T5 (Text-to-Text Transfer Transformer) is a model that converts one form of text input to another form of text output. Here, it's used to generate questions based on extracted text from the PDFs.
    - The **valhalla/t5-small-qg-prepend** model is a pre-trained T5 model designed specifically for **question generation**. The tokenizer converts text into a format the model understands, and the model generates questions based on this input text.

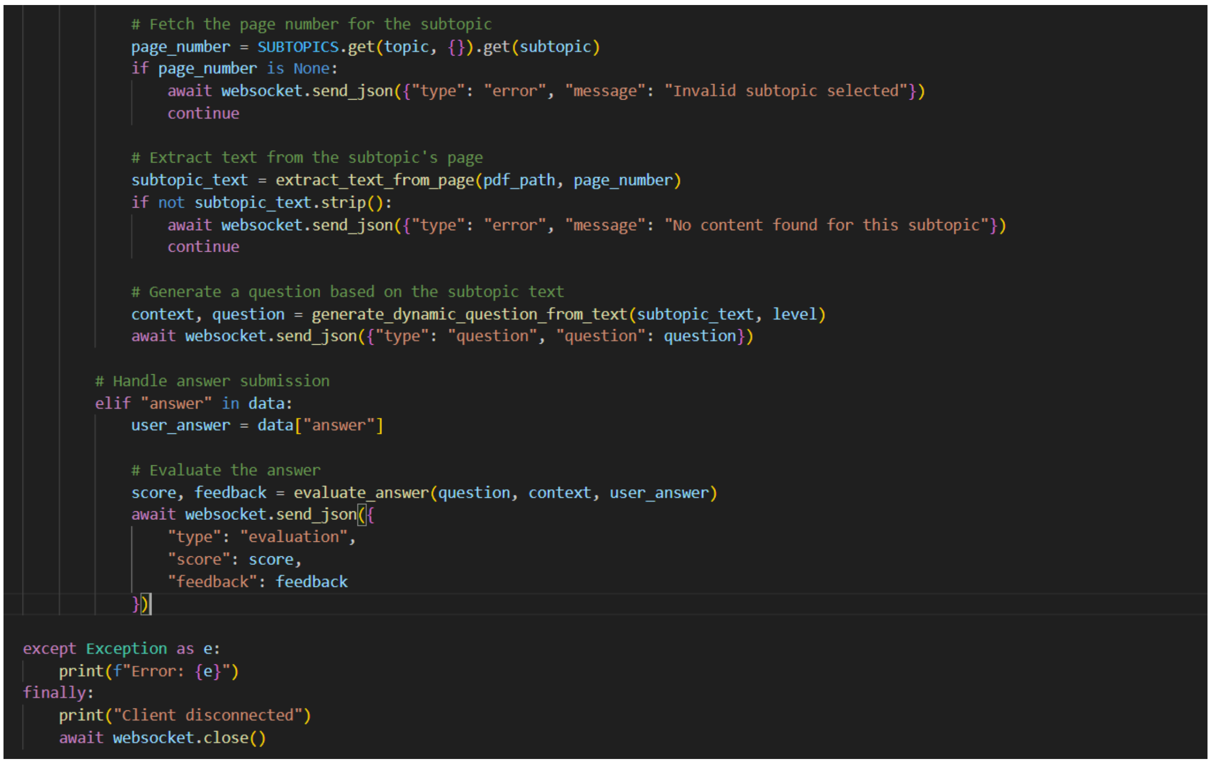
In this project for the ease of data retrieval I have used PDF's for both AI and ML related question. I have extracted the information respectively and generated the questions.

WebSocket Endpoint:

- This is the main function that handles the WebSocket communication.
- The client (e.g., a browser) connects to this WebSocket using the `/ws` endpoint.
- The `websocket.accept()` method is called to accept the WebSocket connection.
- The function listens for incoming messages (in JSON format) from the client, which may include the topic, subtopic, or user answer.



```python
# WebSocket communication
@app.websocket("/ws")
async def websocket_endpoint(websocket: WebSocket):
    await websocket.accept()
    print("Client connected")

    try:
        topic, pdf_path = None, None

        while True:
            data = await websocket.receive_json()

            # Handle topic selection
            if "topic" in data:
                topic = data["topic"]

                # Select PDF based on topic
                if topic == "AI":
                    pdf_path = "C:/Users/karth/AppData/Roaming/Python/Python312/site-packages/AI_Game/Student-Guide-Module-1-Fundamentals-of-AI.pdf"
                elif topic == "ML":
                    pdf_path = "C:/Users/karth/AppData/Roaming/Python/Python312/site-packages/AI_Game/Student-Guide-Module-2-Machine-Learning.pdf"
                else:
                    await websocket.send_json({"type": "error", "message": "Invalid topic"})
                    continue

                # Send subtopics for the selected topic
                subtopics = list(SUBTOPICS.get(topic, {}).keys())
                if not subtopics:
                    await websocket.send_json({"type": "error", "message": "No subtopics available for this topic"})
                    continue
                await websocket.send_json({"type": "subtopics", "subtopics": subtopics})

            # Handle subtopic and level selection
            elif "subtopic" in data and "level" in data:
                subtopic = data["subtopic"]
                level = data["level"]
```

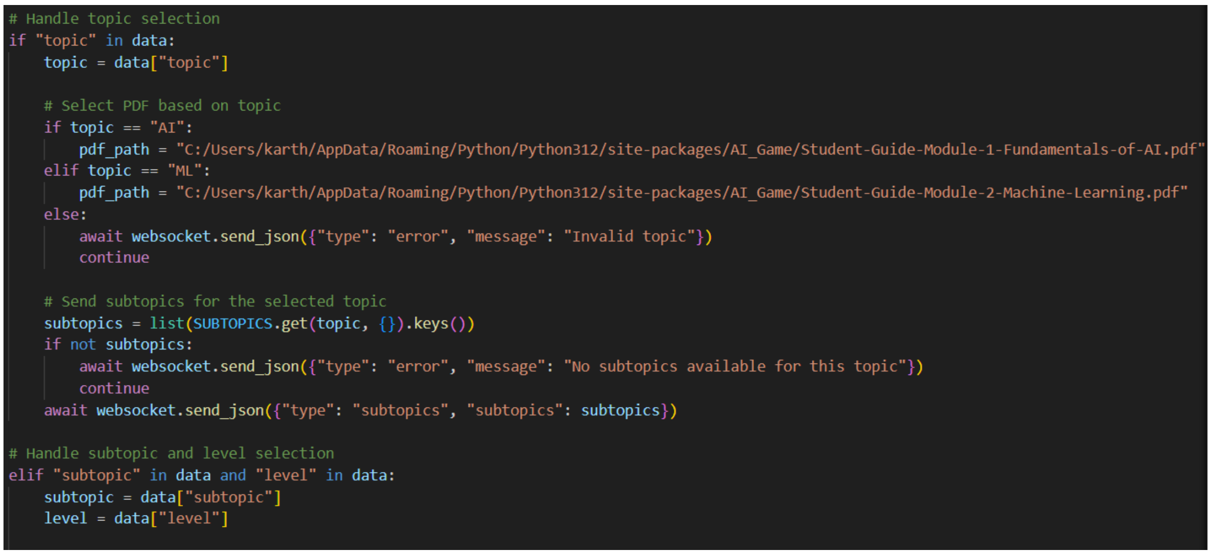


```python
            # Fetch the page number for the subtopic
            page_number = SUBTOPICS.get(topic, {}).get(subtopic)
            if page_number is None:
                await websocket.send_json({"type": "error", "message": "Invalid subtopic selected"})
                continue

            # Extract text from the subtopic's page
            subtopic_text = extract_text_from_page(pdf_path, page_number)
            if not subtopic_text.strip():
                await websocket.send_json({"type": "error", "message": "No content found for this subtopic"})
                continue

            # Generate a question based on the subtopic text
            context, question = generate_dynamic_question_from_text(subtopic_text, level)
            await websocket.send_json({"type": "question", "question": question})

        # Handle answer submission
        elif "answer" in data:
            user_answer = data["answer"]

            # Evaluate the answer
            score, feedback = evaluate_answer(question, context, user_answer)
            await websocket.send_json({
                "type": "evaluation",
                "score": score,
                "feedback": feedback
            })

except Exception as e:
    print(f"Error: {e}")
finally:
    print("Client disconnected")
    await websocket.close()
```

Lets take a look at how the question is generated:

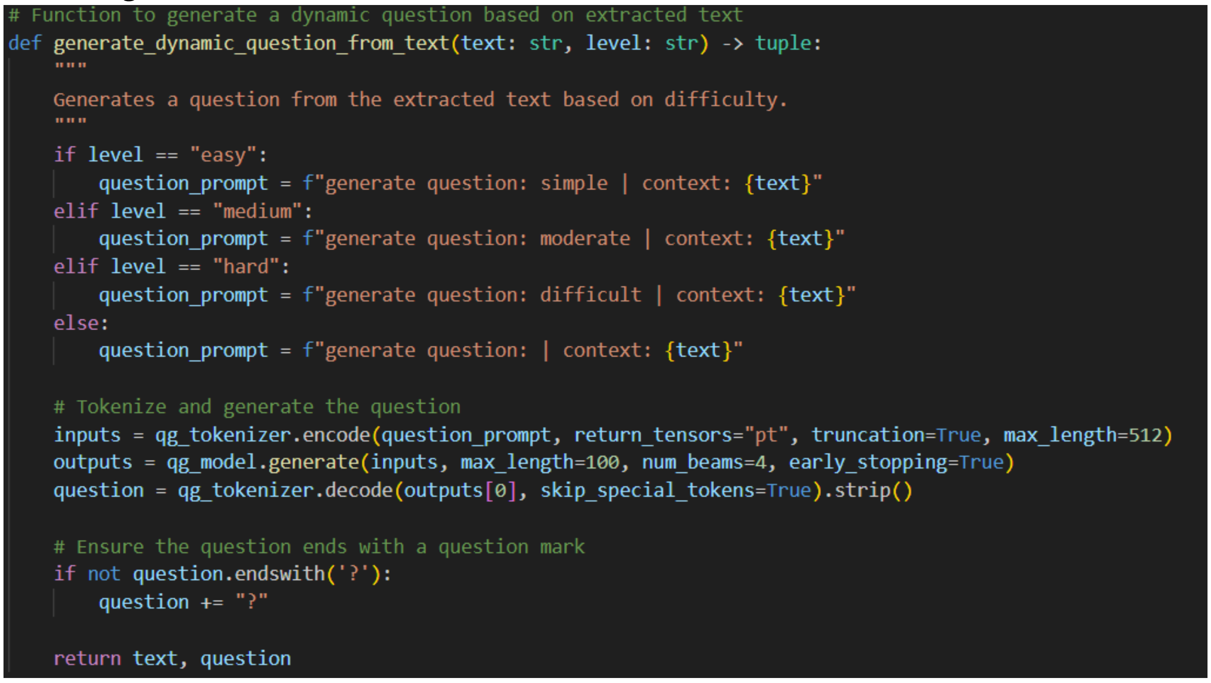
Topic Selection:

- The client sends a selected topic (either "AI" or "ML"), and based on this, the server loads the appropriate PDF (`pdf_path`).
- It checks if the topic is valid and sends the corresponding subtopics using *SUBTOPICS*.get(topic).
- If no subtopics are available, it sends an error message.



```python
# Handle topic selection
if "topic" in data:
    topic = data["topic"]

    # Select PDF based on topic
    if topic == "AI":
        pdf_path = "C:/Users/karth/AppData/Roaming/Python/Python312/site-packages/AI_Game/Student-Guide-Module-1-Fundamentals-of-AI.pdf"
    elif topic == "ML":
        pdf_path = "C:/Users/karth/AppData/Roaming/Python/Python312/site-packages/AI_Game/Student-Guide-Module-2-Machine-Learning.pdf"
    else:
        await websocket.send_json({"type": "error", "message": "Invalid topic"})
        continue

    # Send subtopics for the selected topic
    subtopics = list(SUBTOPICS.get(topic, {}).keys())
    if not subtopics:
        await websocket.send_json({"type": "error", "message": "No subtopics available for this topic"})
        continue
    await websocket.send_json({"type": "subtopics", "subtopics": subtopics})

# Handle subtopic and level selection
elif "subtopic" in data and "level" in data:
    subtopic = data["subtopic"]
    level = data["level"]
```

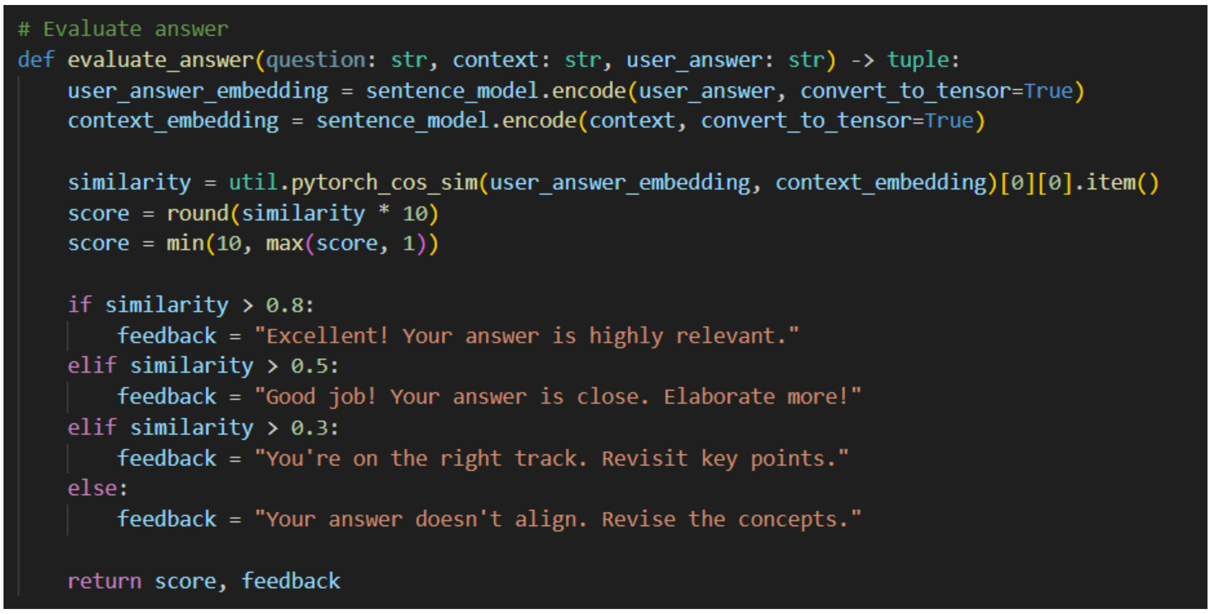
Now the generation function:



```python
# Function to generate a dynamic question based on extracted text
def generate_dynamic_question_from_text(text: str, level: str) -> tuple:
    """
    Generates a question from the extracted text based on difficulty.
    """
    if level == "easy":
        question_prompt = f"generate question: simple | context: {text}"
    elif level == "medium":
        question_prompt = f"generate question: moderate | context: {text}"
    elif level == "hard":
        question_prompt = f"generate question: difficult | context: {text}"
    else:
        question_prompt = f"generate question: | context: {text}"

    # Tokenize and generate the question
    inputs = qg_tokenizer.encode(question_prompt, return_tensors="pt", truncation=True, max_length=512)
    outputs = qg_model.generate(inputs, max_length=100, num_beams=4, early_stopping=True)
    question = qg_tokenizer.decode(outputs[0], skip_special_tokens=True).strip()

    # Ensure the question ends with a question mark
    if not question.endswith('?'):
        question += "?"

    return text, question
```

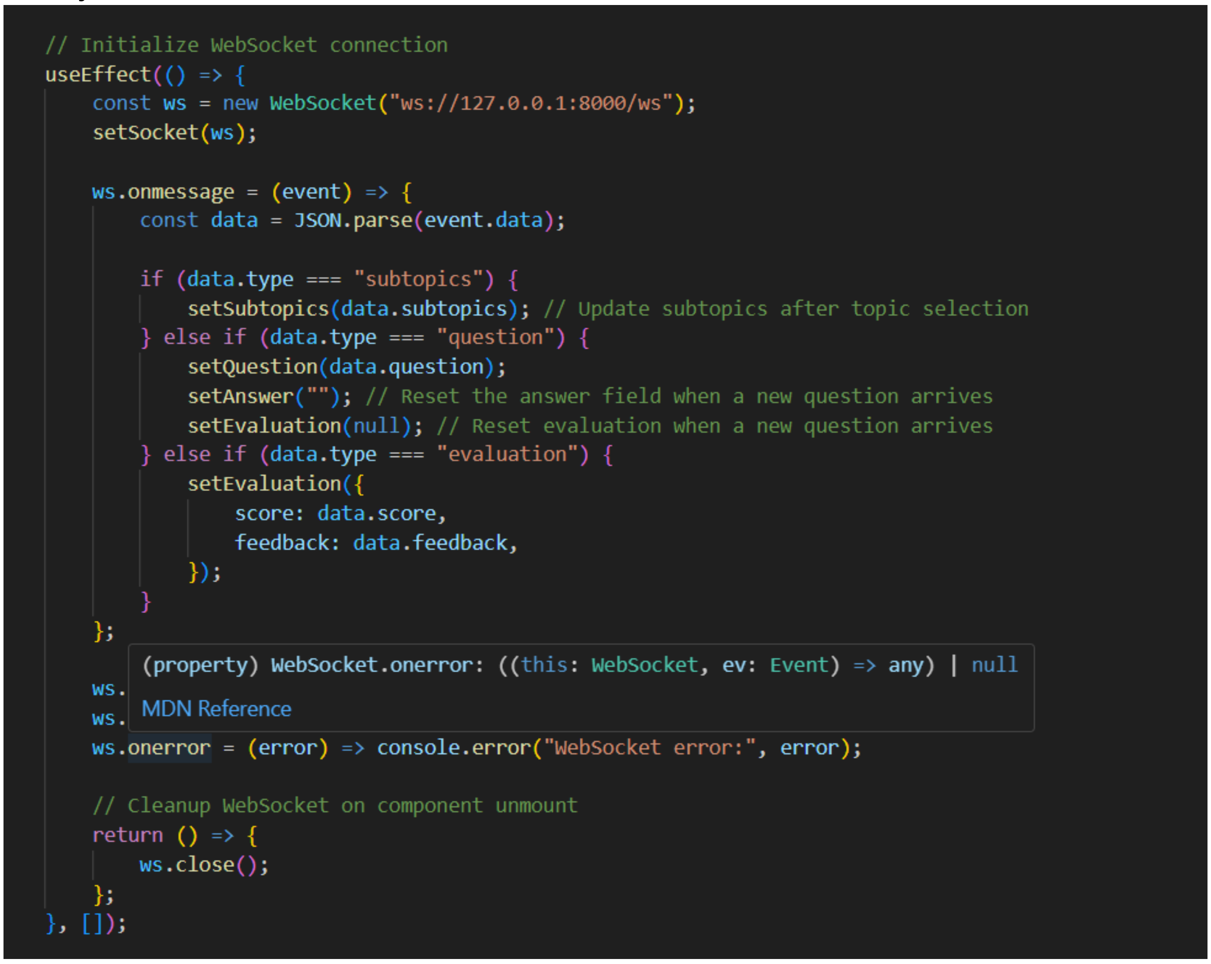
Now lets take a look at how the **answer** is **evaluated** and the logic behind it:
Answer Evaluation:

- The user’s answer is sent to the backend for evaluation.
- The evaluate_answer function compares the user’s answer with the expected context using semantic similarity (via Sentence Transformers). The function returns a *score* (0 to 10) and *feedback* based on the similarity.



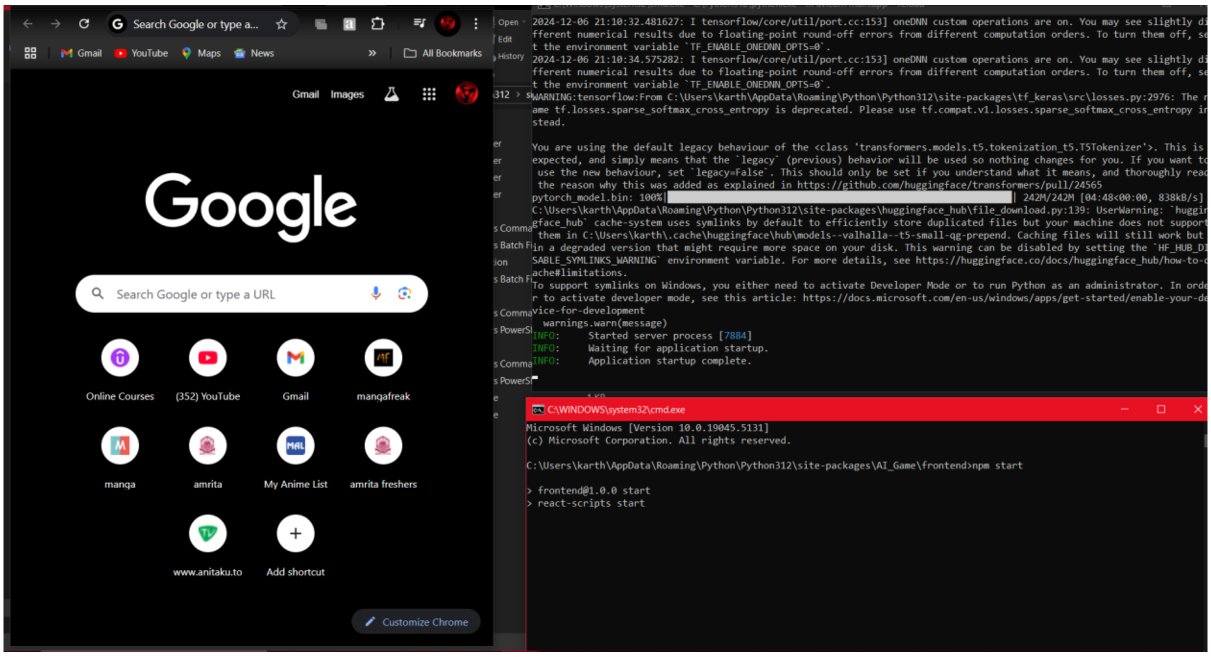
```python
# Evaluate answer
def evaluate_answer(question: str, context: str, user_answer: str) -> tuple:
    user_answer_embedding = sentence_model.encode(user_answer, convert_to_tensor=True)
    context_embedding = sentence_model.encode(context, convert_to_tensor=True)

    similarity = util.pytorch_cos_sim(user_answer_embedding, context_embedding)[0][0].item()
    score = round(similarity * 10)
    score = min(10, max(score, 1))

    if similarity > 0.8:
        feedback = "Excellent! Your answer is highly relevant."
    elif similarity > 0.5:
        feedback = "Good job! Your answer is close. Elaborate more!"
    elif similarity > 0.3:
        feedback = "You're on the right track. Revisit key points."
    else:
        feedback = "Your answer doesn't align. Revise the concepts."

    return score, feedback
```

Finally lets take a look at the **frontend websocket** initialization:



```
// Initialize WebSocket connection
useEffect(() => {
    const ws = new WebSocket("ws://127.0.0.1:8000/ws");
    setSocket(ws);

    ws.onmessage = (event) => {
        const data = JSON.parse(event.data);

        if (data.type === "subtopics") {
            setSubtopics(data.subtopics); // Update subtopics after topic selection
        } else if (data.type === "question") {
            setQuestion(data.question);
            setAnswer(""); // Reset the answer field when a new question arrives
            setEvaluation(null); // Reset evaluation when a new question arrives
        } else if (data.type === "evaluation") {
            setEvaluation({
                score: data.score,
                feedback: data.feedback,
            });
        }
    };
       (property) WebSocket.onerror: ((this: WebSocket, ev: Event) => any) | null
    ws.
    ws.  MDN Reference
    ws.onerror = (error) => console.error("WebSocket error:", error);

    // Cleanup WebSocket on component unmount
    return () => {
        ws.close();
    };
}, []);
```

With this overview I would like to share some results on the projects development so far:
1st we initiate the connection between server and frontend:

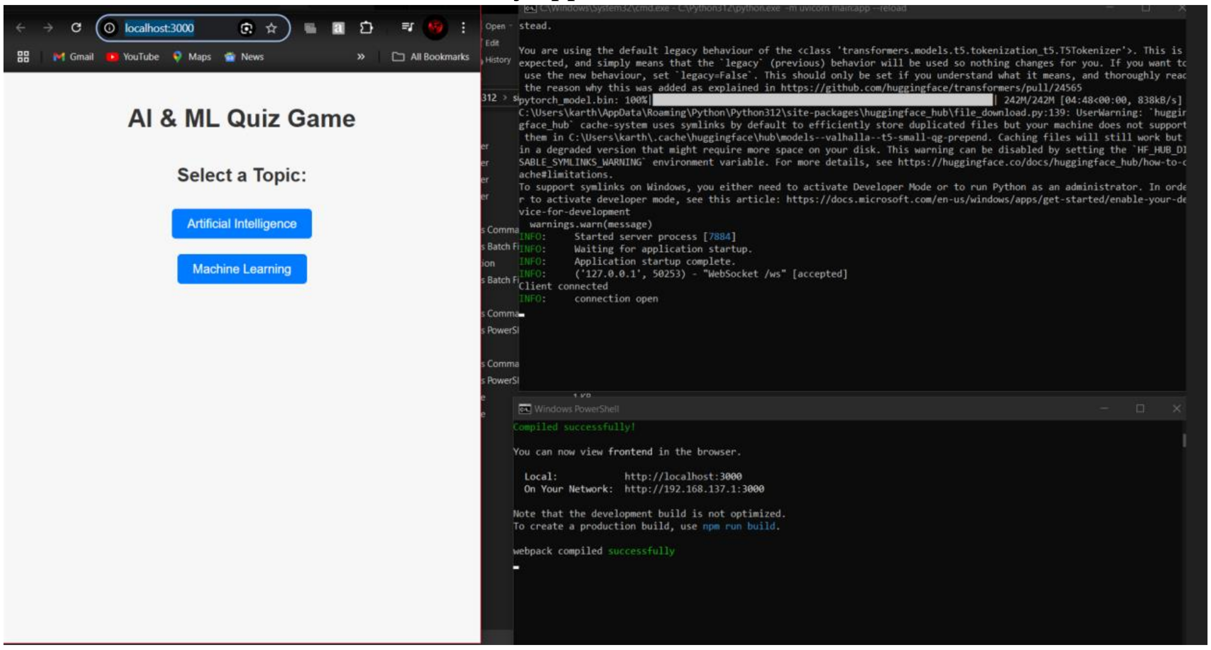


```
2024-12-06 21:10:32.481627: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly di
fferent numerical results due to floating-point round-off errors from different computation orders. To turn them off, se
t the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2024-12-06 21:10:34.575282: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly di
fferent numerical results due to floating-point round-off errors from different computation orders. To turn them off, se
t the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
WARNING:tensorflow:From C:\Users\karth\AppData\Roaming\Python\Python312\site-packages\tf_keras\src\losses.py:2976: The n
ame tf.losses.sparse_softmax_cross_entropy is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy in
stead.

You are using the default legacy behaviour of the <class 'transformers.models.t5.tokenization_t5.T5Tokenizer'>. This is
expected, and simply means that the `legacy` (previous) behavior will be used so nothing changes for you. If you want to
 use the new behaviour, set `legacy=False`. This should only be set if you understand what it means, and thoroughly read
 the reason why this was added as explained in https://github.com/huggingface/transformers/pull/24565
pytorch_model.bin: 100%|████████████████████████████████████████| 242M/242M [04:48<00:00, 838kB/s]
C:\Users\karth\AppData\Roaming\Python\Python312\site-packages\huggingface_hub\file_download.py:139: UserWarning: `huggin
gface_hub` cache-system uses symlinks by default to efficiently store duplicated files but your machine does not support
 them in C:\Users\karth\.cache\huggingface\hub\models--valhalla--t5-small-qg-prepend. Caching files will still work but
in a degraded version that might require more space on your disk. This warning can be disabled by setting the `HF_HUB_DI
SABLE_SYMLINKS_WARNING` environment variable. For more details, see https://huggingface.co/docs/huggingface_hub/how-to-c
ache#limitations.
To support symlinks on Windows, you either need to activate Developer Mode or to run Python as an administrator. In orde
r to activate developer mode, see this article: https://docs.microsoft.com/en-us/windows/apps/get-started/enable-your-de
vice-for-development
  warnings.warn(message)
INFO:     Started server process [7884]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
```

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.19045.5131]
(c) Microsoft Corporation. All rights reserved.

C:\Users\karth\AppData\Roaming\Python\Python312\site-packages\AI_Game\frontend>npm start

> frontend@1.0.0 start
> react-scripts start
```

On the right top we have the backend being started run by the command -> uvicorn main:app –reload.
Bottom right we have the Frontend -> npm start

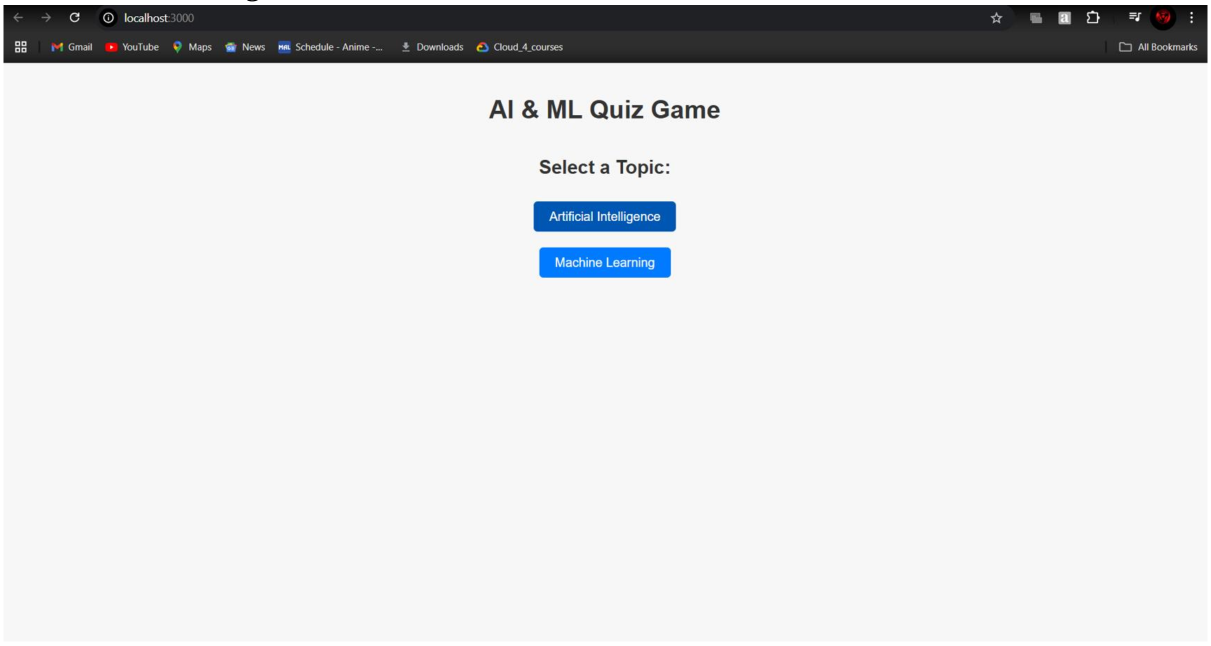
Now on localhost:3000 I have hosted my application:



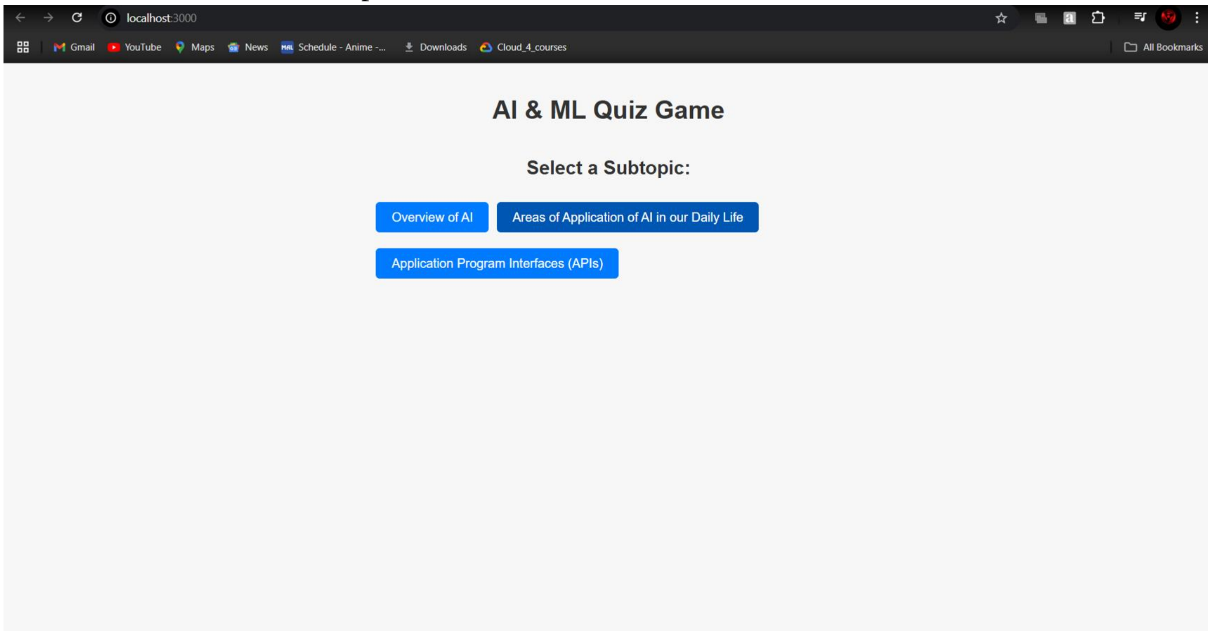
```
C:\Windows\System32\cmd.exe - C:\Python312\python.exe -m uvicorn main:app --reload
stead.

You are using the default legacy behaviour of the <class 'transformers.models.t5.tokenization_t5.T5Tokenizer'>. This is
expected, and simply means that the `legacy` (previous) behavior will be used so nothing changes for you. If you want to
 use the new behaviour, set `legacy=False`. This should only be set if you understand what it means, and thoroughly read
 the reason why this was added as explained in https://github.com/huggingface/transformers/pull/24565
pytorch_model.bin: 100%|████████████████████████████████████████| 242M/242M [04:48<00:00, 838kB/s]
C:\Users\karth\AppData\Roaming\Python\Python312\site-packages\huggingface_hub\file_download.py:139: UserWarning: `huggin
gface_hub` cache-system uses symlinks by default to efficiently store duplicated files but your machine does not support
 them in C:\Users\karth\.cache\huggingface\hub\models--valhalla--t5-small-qg-prepend. Caching files will still work but
in a degraded version that might require more space on your disk. This warning can be disabled by setting the `HF_HUB_DI
SABLE_SYMLINKS_WARNING` environment variable. For more details, see https://huggingface.co/docs/huggingface_hub/how-to-c
ache#limitations.
To support symlinks on Windows, you either need to activate Developer Mode or to run Python as an administrator. In orde
r to activate developer mode, see this article: https://docs.microsoft.com/en-us/windows/apps/get-started/enable-your-de
vice-for-development
  warnings.warn(message)
INFO:     Started server process [7884]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     ('127.0.0.1', 50253) - "WebSocket /ws" [accepted]
Client connected
INFO:     connection open
```

```
Windows PowerShell
Compiled successfully!

You can now view frontend in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.137.1:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

Now lets run through one round of the Quiz:

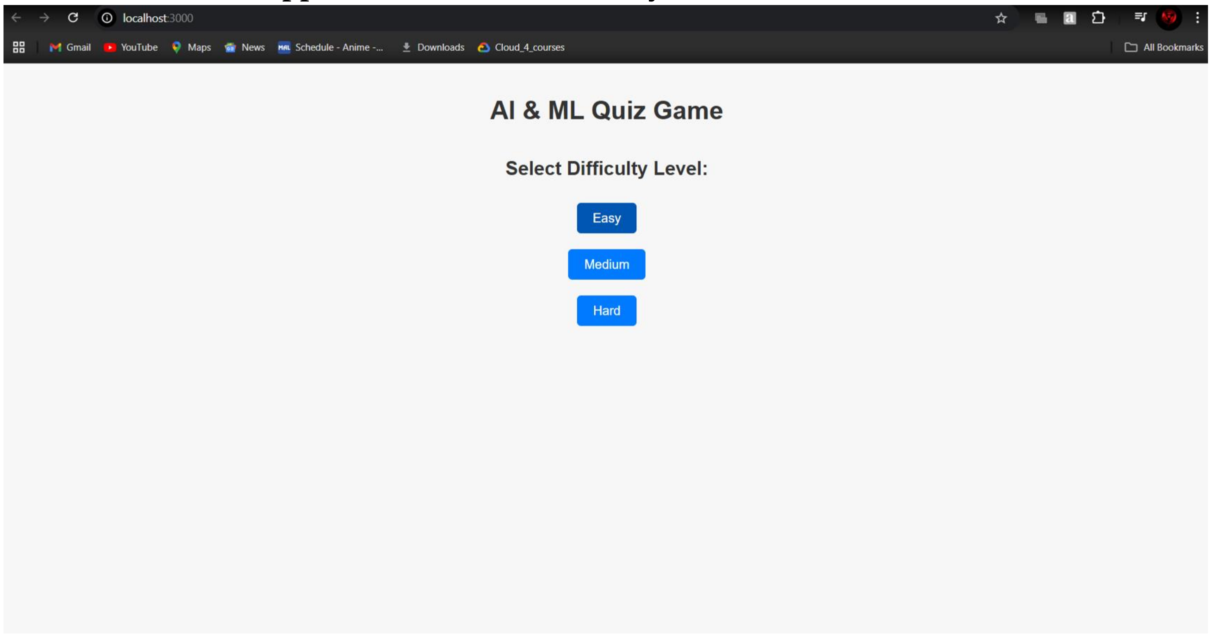


We chose AI as our main topic :

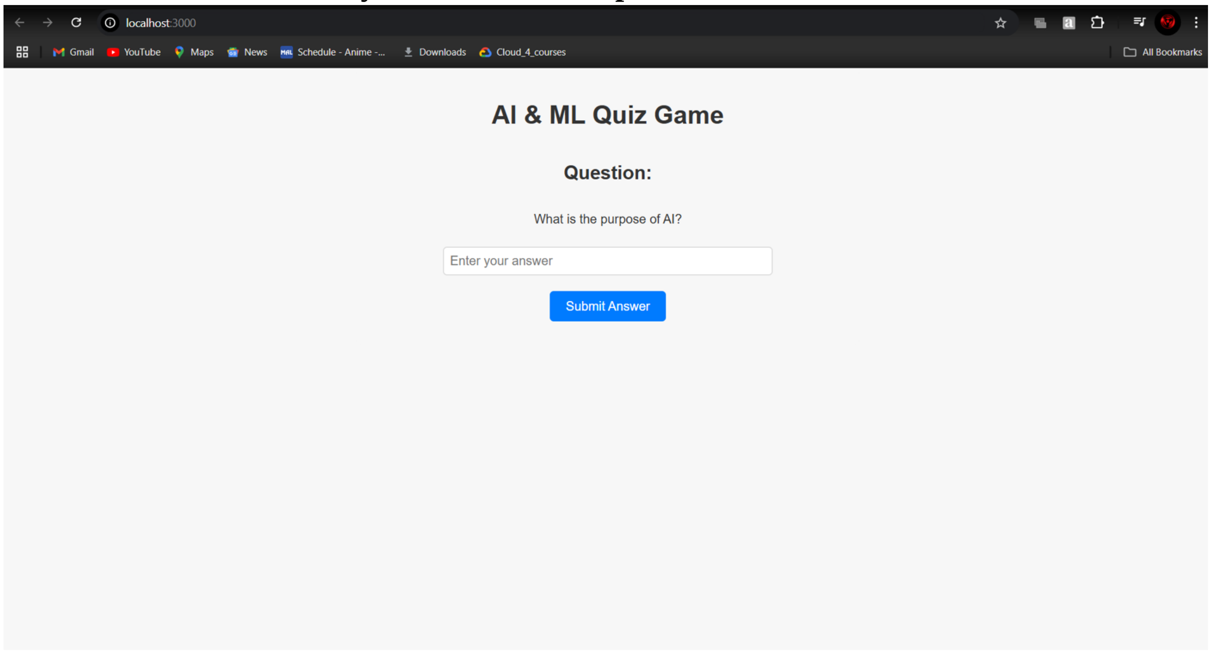


For both AI and ML I have given 3 subtopics each to chose from. Note: this will be improvised. Due to a limited time I have resorted to only 3 topics. These 3 topics are derived from the subtopics available in the pdf. We then feed only the necessary bit of information from the respective subtopic to the **transformer** to generate the question based on the content given.

We chose **Areas of Application of AI in our daily life:**



Now we chose the difficulty level of the subtopic chosen:



AI & ML Quiz Game

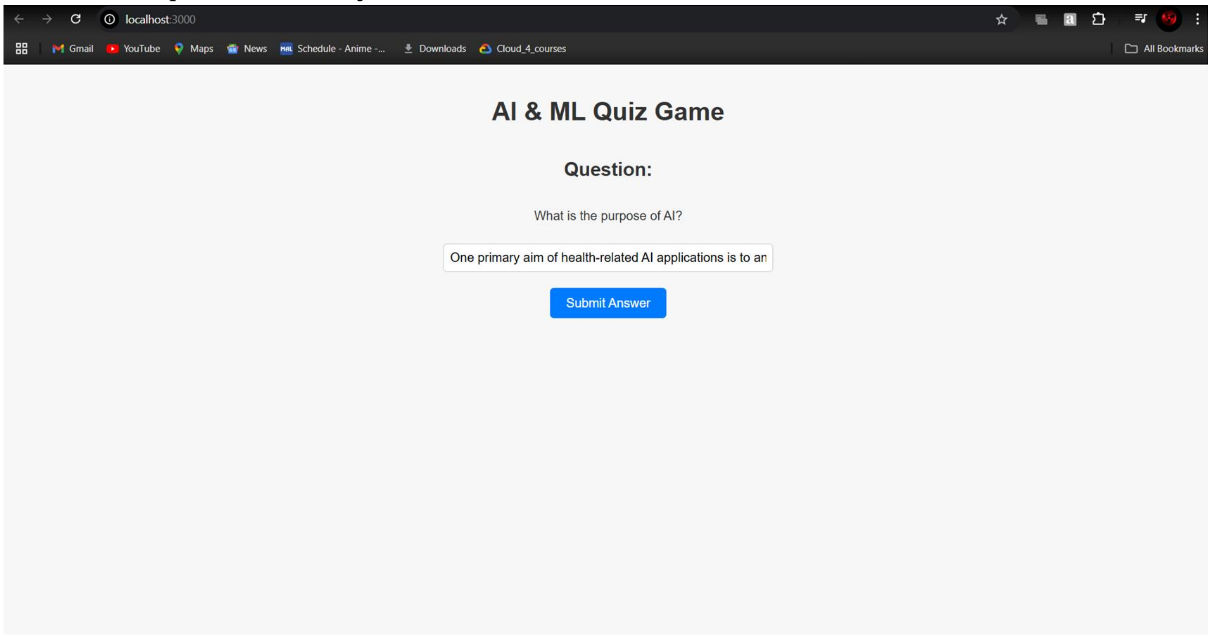
Question:

What is the purpose of AI?

Enter your answer

Submit Answer

Initially I had used Websites as reference. I had used Wikipedia api to dynamically chose the content and derive a question by feeding the content to the transformer. But that had a lot of delay in generating the question. To simplify it, I had used pdf's which give the question in less than 2 secs max.
Now to this question lets say I answer:



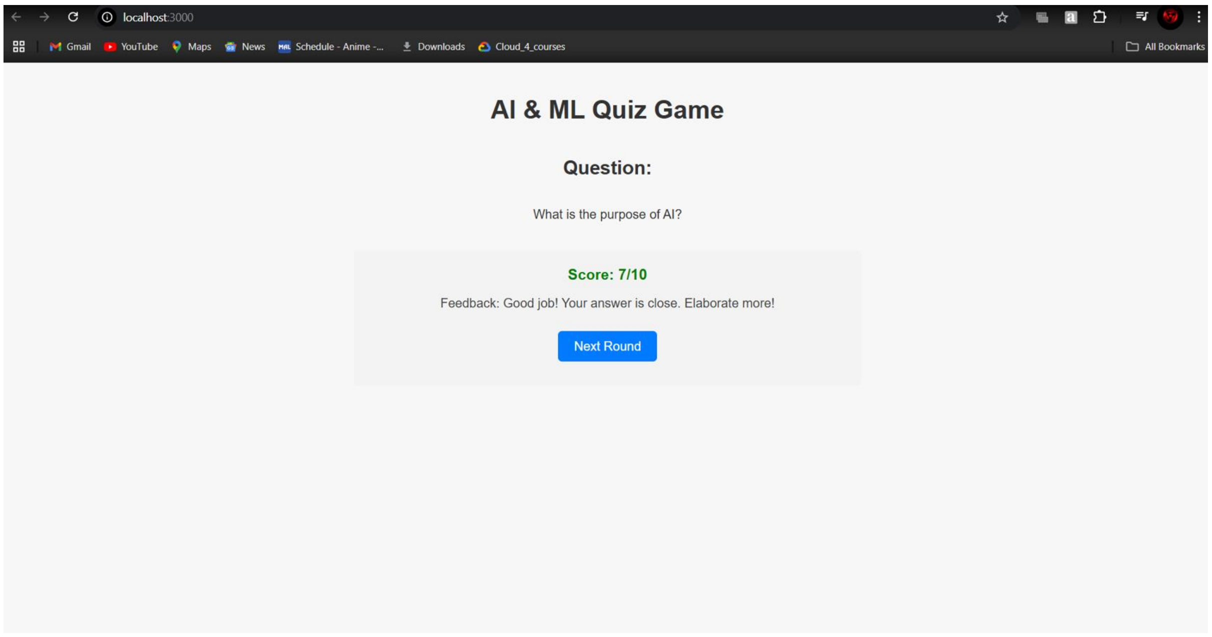
AI & ML Quiz Game

Question:

What is the purpose of AI?

One primary aim of health-related AI applications is to an

Submit Answer


AI & ML Quiz Game

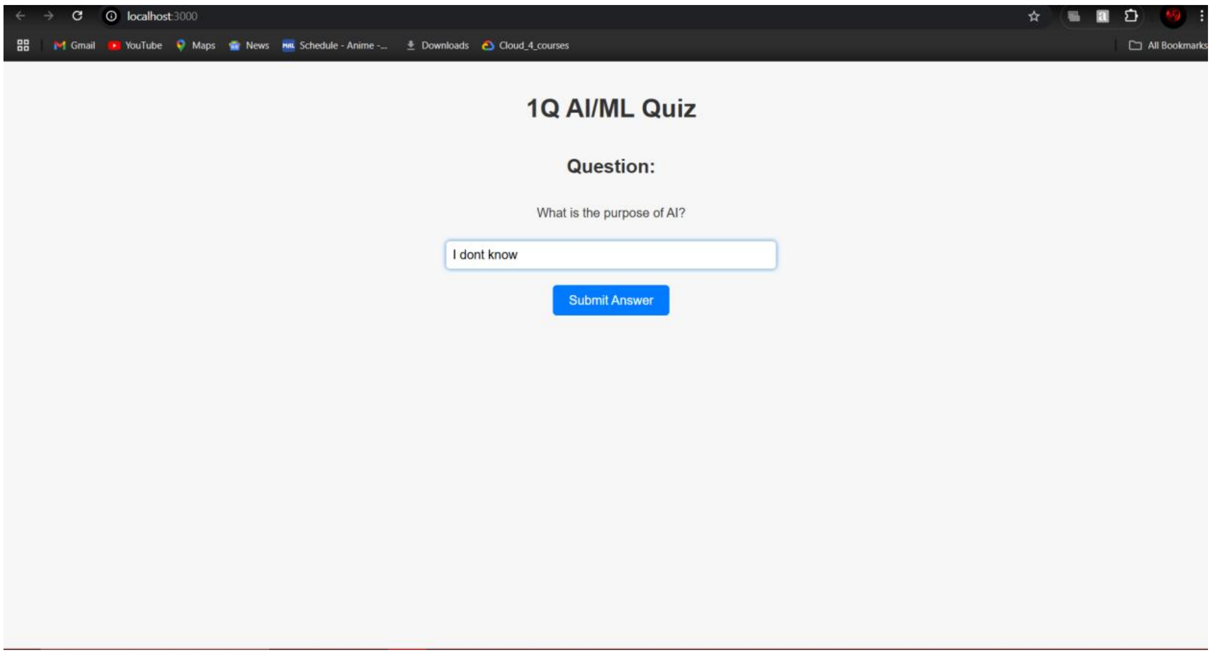
Question:

What is the purpose of AI?

Score: 7/10

Feedback: Good job! Your answer is close. Elaborate more!

Based on my answer I get rated dynamically using the semantic analysis. Now lets see a scenario where I give a weird answer:

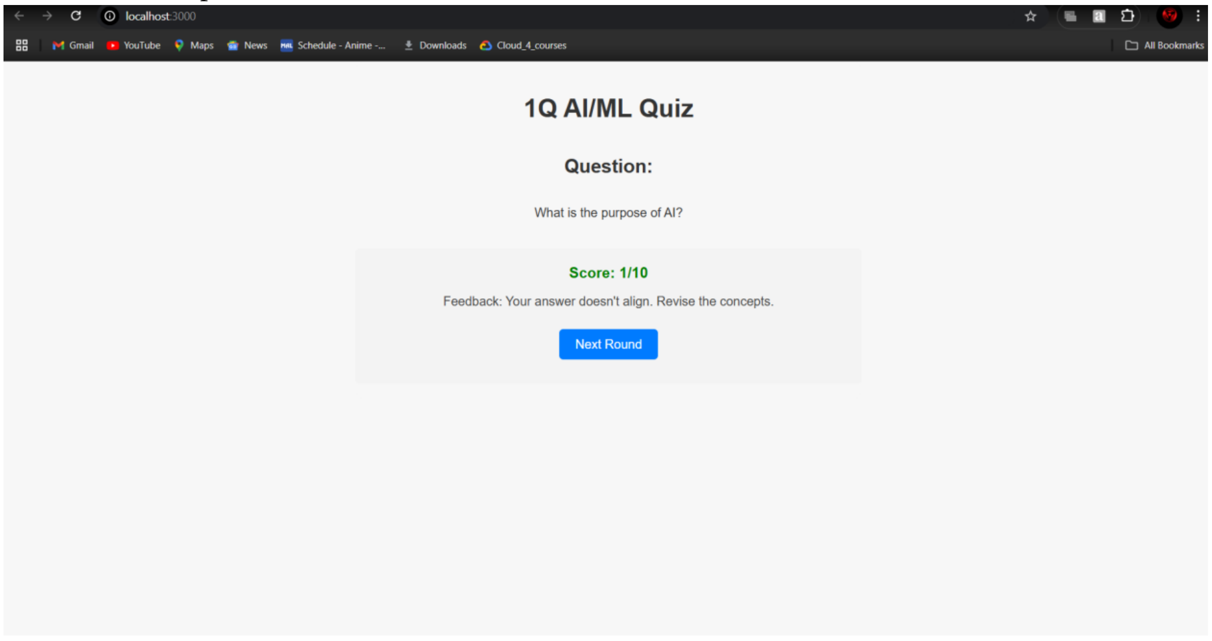

1Q AI/ML Quiz

Question:

What is the purpose of AI?

I dont know

Lets see the response:



1Q AI/ML Quiz

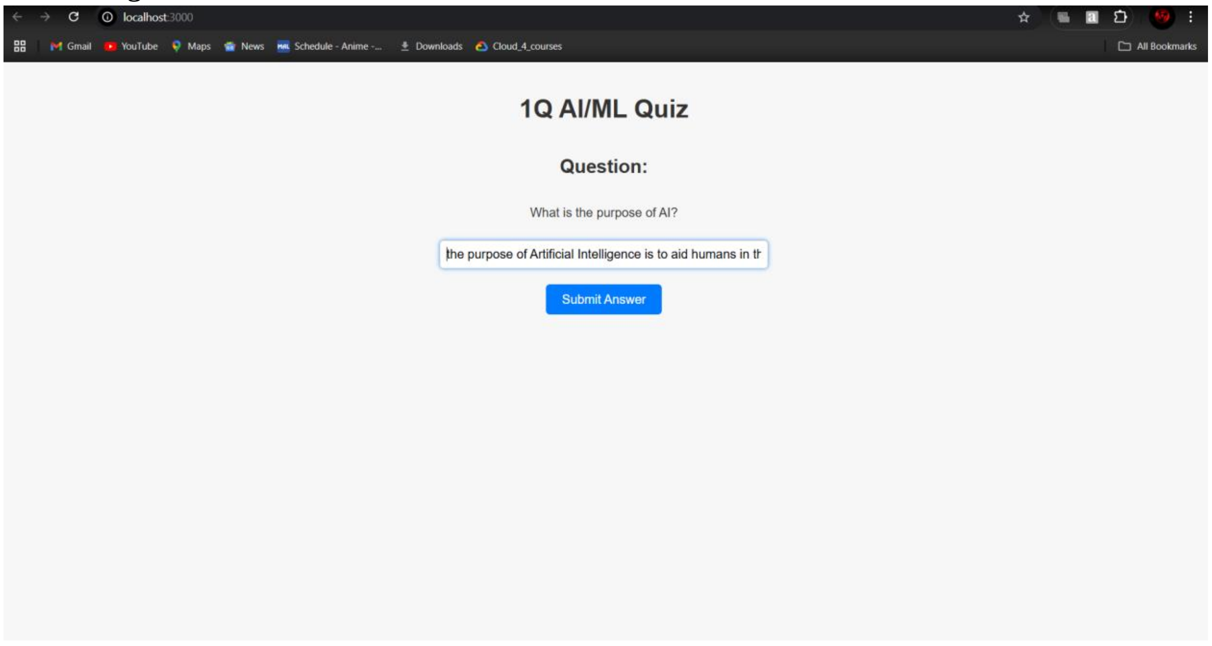
Question:

What is the purpose of AI?

Score: 1/10

Feedback: Your answer doesn't align. Revise the concepts.

Now lets give an answer somewhat relatable:



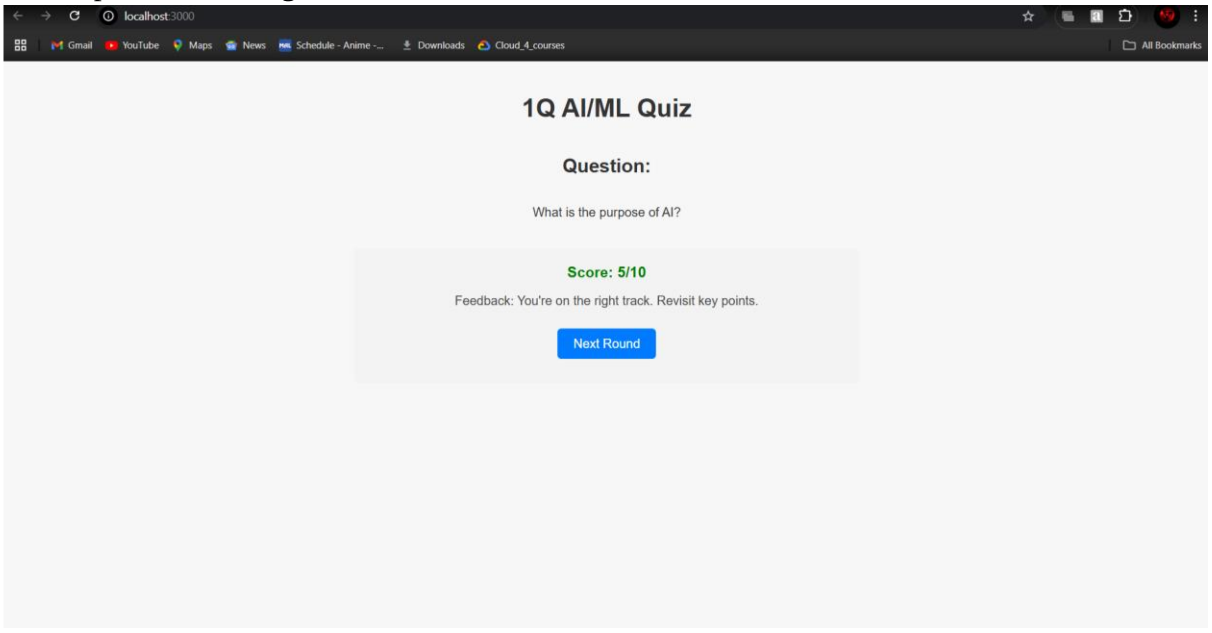
1Q AI/ML Quiz

Question:

What is the purpose of AI?

the purpose of Artificial Intelligence is to aid humans in th

The ouput we should get should be better:



1Q AI/ML Quiz

Question:

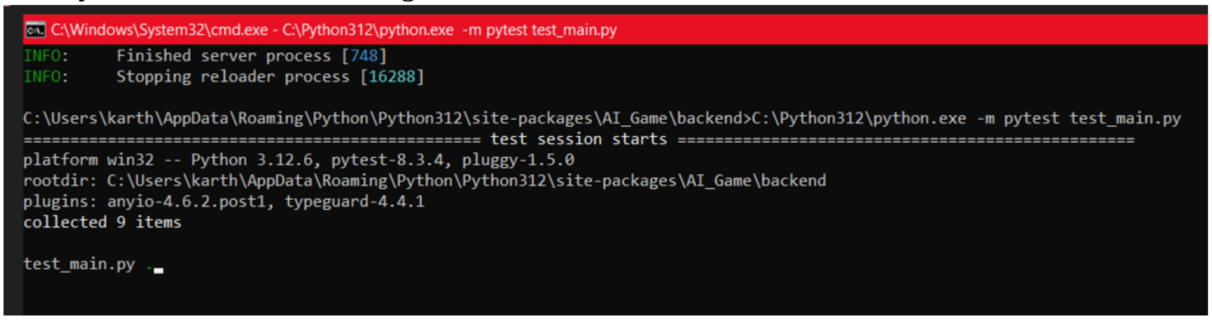
What is the purpose of AI?

Score: 5/10

Feedback: You're on the right track. Revisit key points.

Yes, it is slightly better. The algorithm uses cosine similarity to determine the similarness to the test in the content derived from pdf and the answer we input. Ofcourse if the data set is broader we can definitely get better results. Since this is still in the beginning stages and within a short period of time, this is the progress of the project.

Finally lets move to Unit testing:



```
C:\Windows\System32\cmd.exe - C:\Python312\python.exe  -m pytest test_main.py
INFO:     Finished server process [748]
INFO:     Stopping reloader process [16288]

C:\Users\karth\AppData\Roaming\Python\Python312\site-packages\AI_Game\backend>C:\Python312\python.exe -m pytest test_main.py
================================================= test session starts =================================================
platform win32 -- Python 3.12.6, pytest-8.3.4, pluggy-1.5.0
rootdir: C:\Users\karth\AppData\Roaming\Python\Python312\site-packages\AI_Game\backend
plugins: anyio-4.6.2.post1, typeguard-4.4.1
collected 9 items

test_main.py .
```

The result is test_main.py .<- this small dot at the end in the image is actually green. If we zoom it we can see. That indicates all the test cases have passed.

The test are attached in the backend folder within the Main AI_quiz project drectory.