Task 6: Building a Small End-to-End Application

Objective:

To build a small application using skills learned in all of the previous tasks which will serve some specific use-case.

Use Case Information:

Question Generation Application/system

- The objective of this task is to build a *web application that can generate multiple-choice questions (MCQs)* based on the content of a PDF document and a specified topic.

- The application uses FastAPI for the backend service to process the PDF and generate questions, and Flask to create a simple web interface for users to interact with the service.

Practical Use Case:

- The system allows users to generate MCQs from PDF documents based on a given topic. This can be particularly useful for educators, students, and content creators who need to create quizzes or study materials efficiently.

Requirements and Expected Functionalities:

- **File Upload**: Ability to upload a PDF document.

- **Topic Specification**: Input field to specify the topic for generating questions.

- **Question Generation**: Backend service to extract text, generate embeddings, find relevant content, and produce MCQs.

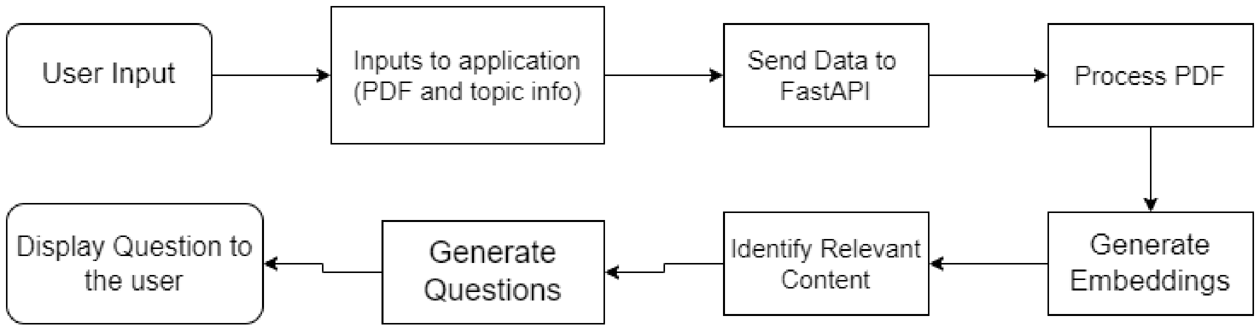
- **Web Interface**: Simple UI to input data and display generated questions.

Architecture of Application:

- **Frontend**: Flask application serving an HTML form.

- **Backend**: FastAPI application handling the processing of the PDF and generation of questions using GenAI models.

- **Data Flow**:
    1. User inputs the PDF link and topic in the Flask form.
    2. Flask sends the input data to FastAPI.
    3. Next we process the PDF, generate embeddings, and identify relevant content.
    4. Next we generate questions using the GenAI model and return them to Flask.
    5. Flask displays the generated questions to the user.


User Input
Inputs to application (PDF and topic info)
Send Data to FastAPI
Process PDF
Generate Embeddings
Identify Relevant Content
Generate Questions
Display Question to the user

Steps Taken:

Step - 1: Extracting text from the PDF

- We start by extracting the text from a PDF document and combining it to form a large text. This is done using the PyPDF2 library.

```
text = ""
reader = PdfReader(file_link)
for page_num in range(len(reader.pages)):
  page = reader.pages[page_num]
  text = text + str(page.extract_text())
```

Step - 2: Chunk the large document

- Defines a function to split a large text into smaller chunks based on a maximum chunk size.By using the nltk library.

```python
def chunk_document(text, max_chunk_size=1000):
    sentences = sent_tokenize(text)
    chunks = []
    current_chunk = []
    current_chunk_size = 0

    for sentence in sentences:
        sentence_size = len(sentence)
        if current_chunk_size + sentence_size > max_chunk_size:
            chunks.append(" ".join(current_chunk))
            current_chunk = [sentence]
            current_chunk_size = sentence_size
        else:
            current_chunk.append(sentence)
            current_chunk_size += sentence_size

    if current_chunk:
        chunks.append(" ".join(current_chunk))

    return chunks
```

Step - 3: Extract relevant chunks

- A function to create embeddings for a list of text chunks.

```python
def create_embedding(chunks):
    embedding_list = []
    for chunk in chunks:
        chunk_embedding = get_embedding(chunk)
        embedding_list.append(chunk_embedding)

    return embedding_list
```

- Calculates the cosine similarity between each chunk and the topic, and identifies the most relevant chunks.

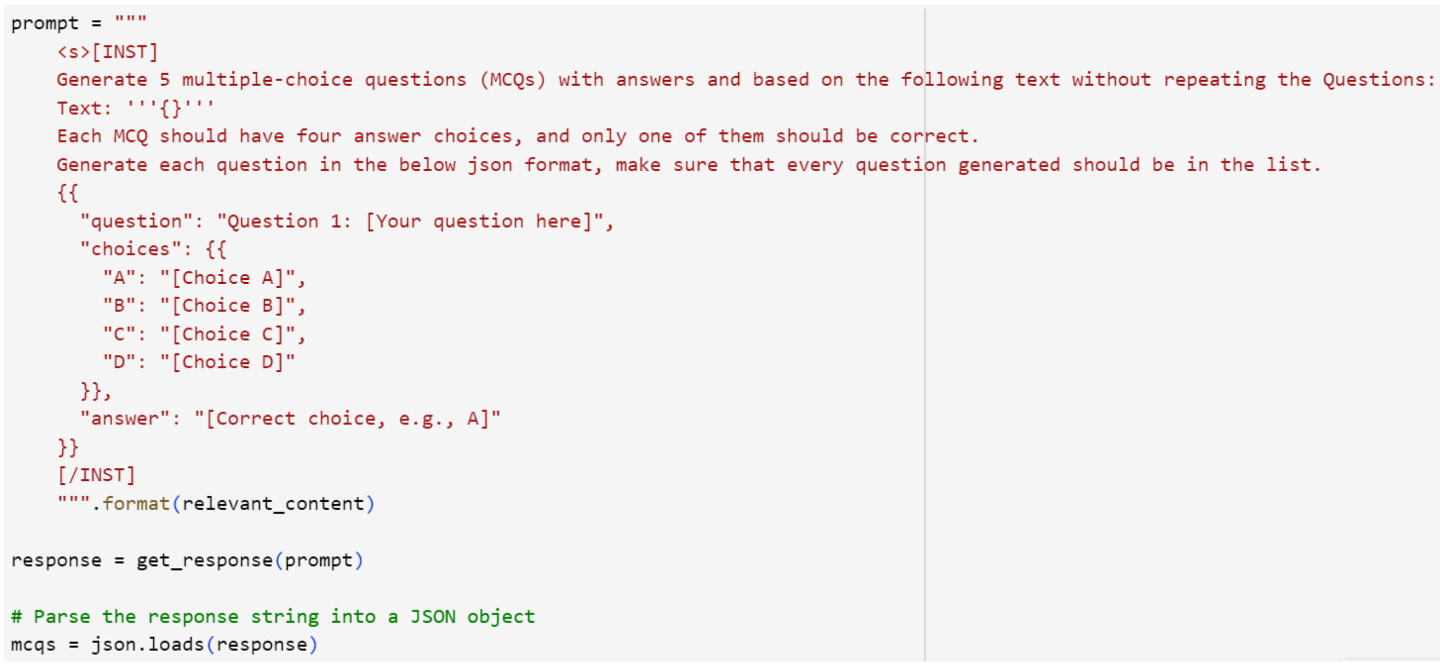
```python
# Calculate the cosine similarity between each chunk embedding and the topic embedding
topic_embedding = np.array(topic_embedding).reshape(1, -1)
similarities = cosine_similarity(chunk_embeddings, topic_embedding)

# Flatten the similarities array and get the top 2 most similar chunks
similarities = similarities.flatten()
top_indices = np.argsort(similarities)[-2:][::-1]
relevant_chunks = [chunks[i] for i in top_indices]
relevant_content = str(relevant_chunks[0] + relevant_chunks[1])
```

Step - 4: Generate Questions:

- Generates multiple-choice questions based on the relevant chunks using OpenAI's API and formats the response as JSON.

Prompt used for the above:



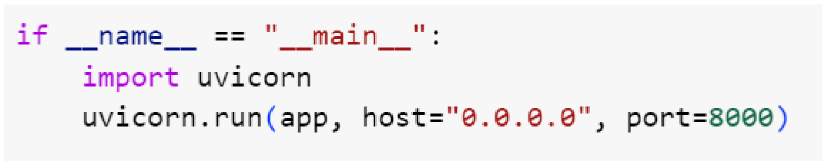
```
prompt = """
    <s>[INST]
    Generate 5 multiple-choice questions (MCQs) with answers and based on the following text without repeating the Questions:
    Text: '''{}'''
    Each MCQ should have four answer choices, and only one of them should be correct.
    Generate each question in the below json format, make sure that every question generated should be in the list.
    {{
      "question": "Question 1: [Your question here]",
      "choices": {{
        "A": "[Choice A]",
        "B": "[Choice B]",
        "C": "[Choice C]",
        "D": "[Choice D]"
      }},
      "answer": "[Correct choice, e.g., A]"
    }}
    [/INST]
    """.format(relevant_content)

response = get_response(prompt)

# Parse the response string into a JSON object
mcqs = json.loads(response)
```

Running the Application:

- Runs the FastAPI application using Uvicorn, a lightning-fast ASGI server.

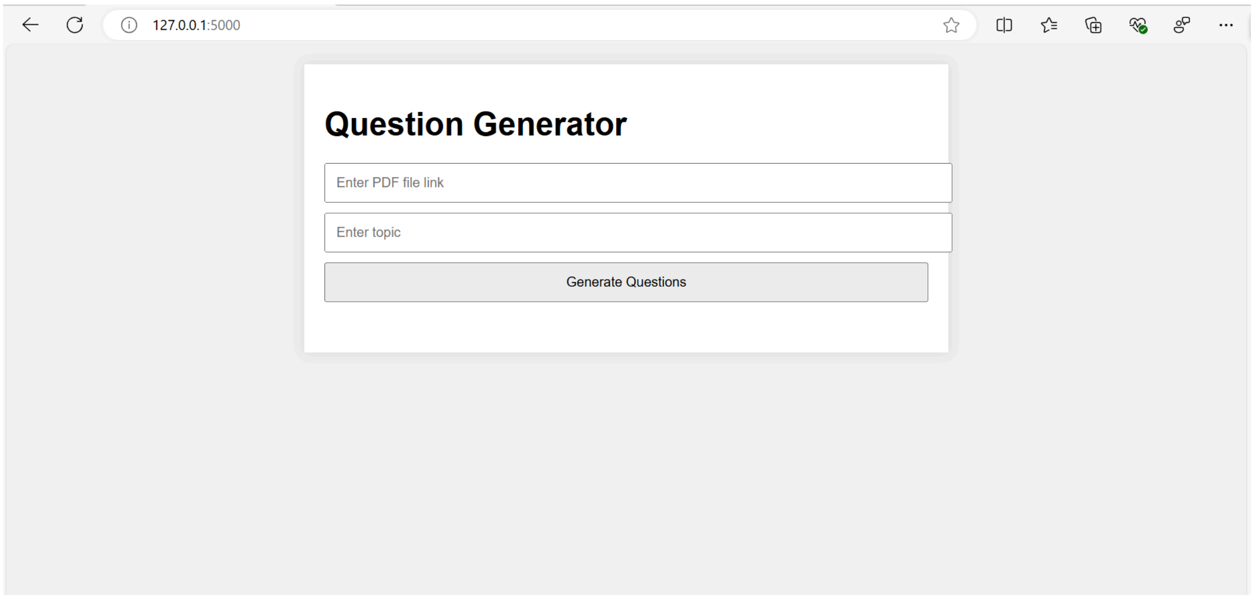


```
if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

After this a basic UI is created using HTML for frontend and Flask for the backend. The code for the UI is written using GPT.

Testing and Results:

UI:


127.0.0.1:5000

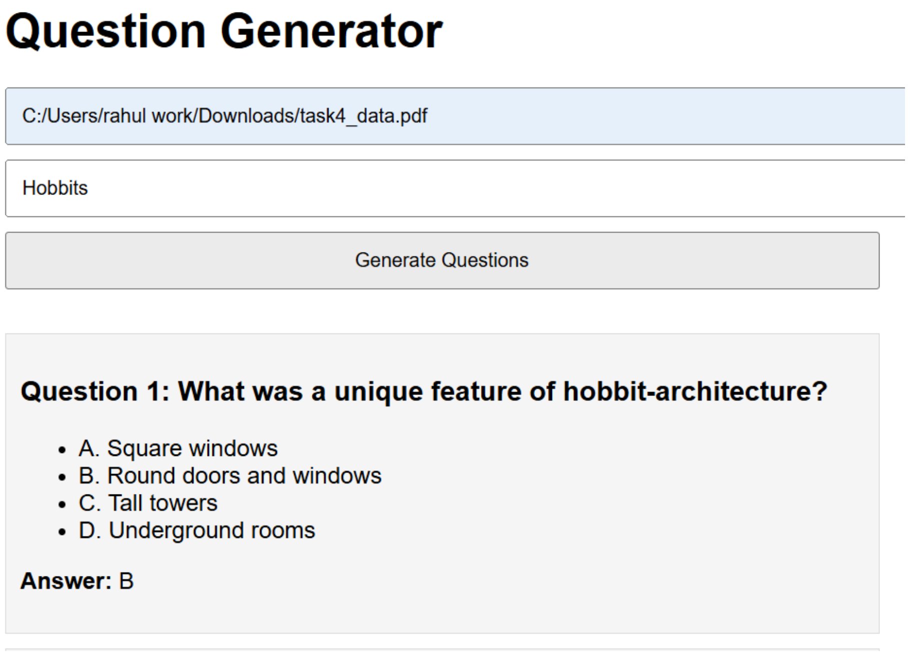
Question Generator

Enter PDF file link

Enter topic

Generate Questions

As the user Inputs PDF link and the topic on which the user wants to generate MCQ questions.


Question Generator

C:/Users/rahul work/Downloads/task4_data.pdf

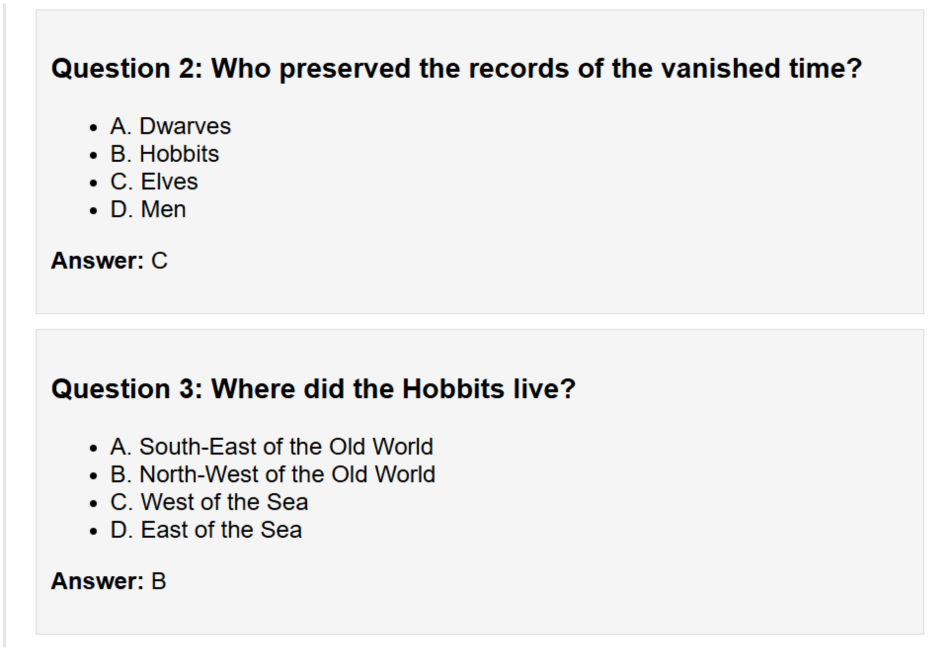
Hobbits

Generate Questions

Question 1: What was a unique feature of hobbit-architecture?

- A. Square windows
- B. Round doors and windows
- C. Tall towers
- D. Underground rooms

Answer: B



Question 2: Who preserved the records of the vanished time?

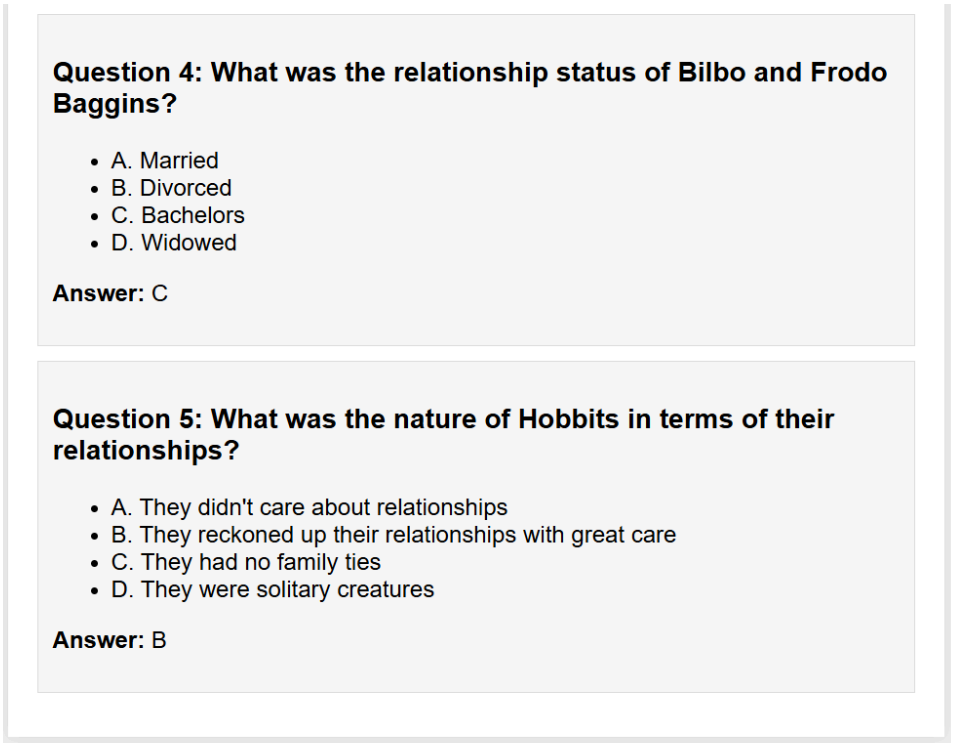
- A. Dwarves
- B. Hobbits
- C. Elves
- D. Men

Answer: C

Question 3: Where did the Hobbits live?

- A. South-East of the Old World
- B. North-West of the Old World
- C. West of the Sea
- D. East of the Sea

Answer: B



Question 4: What was the relationship status of Bilbo and Frodo Baggins?

- A. Married
- B. Divorced
- C. Bachelors
- D. Widowed

Answer: C

Question 5: What was the nature of Hobbits in terms of their relationships?

- A. They didn't care about relationships
- B. They reckoned up their relationships with great care
- C. They had no family ties
- D. They were solitary creatures

Answer: B

Conclusions and future scope:

We have successfully implemented a FastAPI backend service for generating multiple-choice questions (MCQs) from PDF documents based on a specified topic.

Future Scope:

- We are taking the pdf link as the input but we can improve this web application by giving the option of uploading the file.
- We can add different types of questions like (Fillups, T/F questions etc).

Challenges Faced:

- One of the main challenges faced was to tune the prompt to generate the correct response in the correct format we want, so we used different prompting techniques to solve this issue.