Large Scale Exam Grading Using Cloud

Presented by:

Sharwin Neema

Saket Nerurkar

Yash Aditya

Ashmit Khandelwal

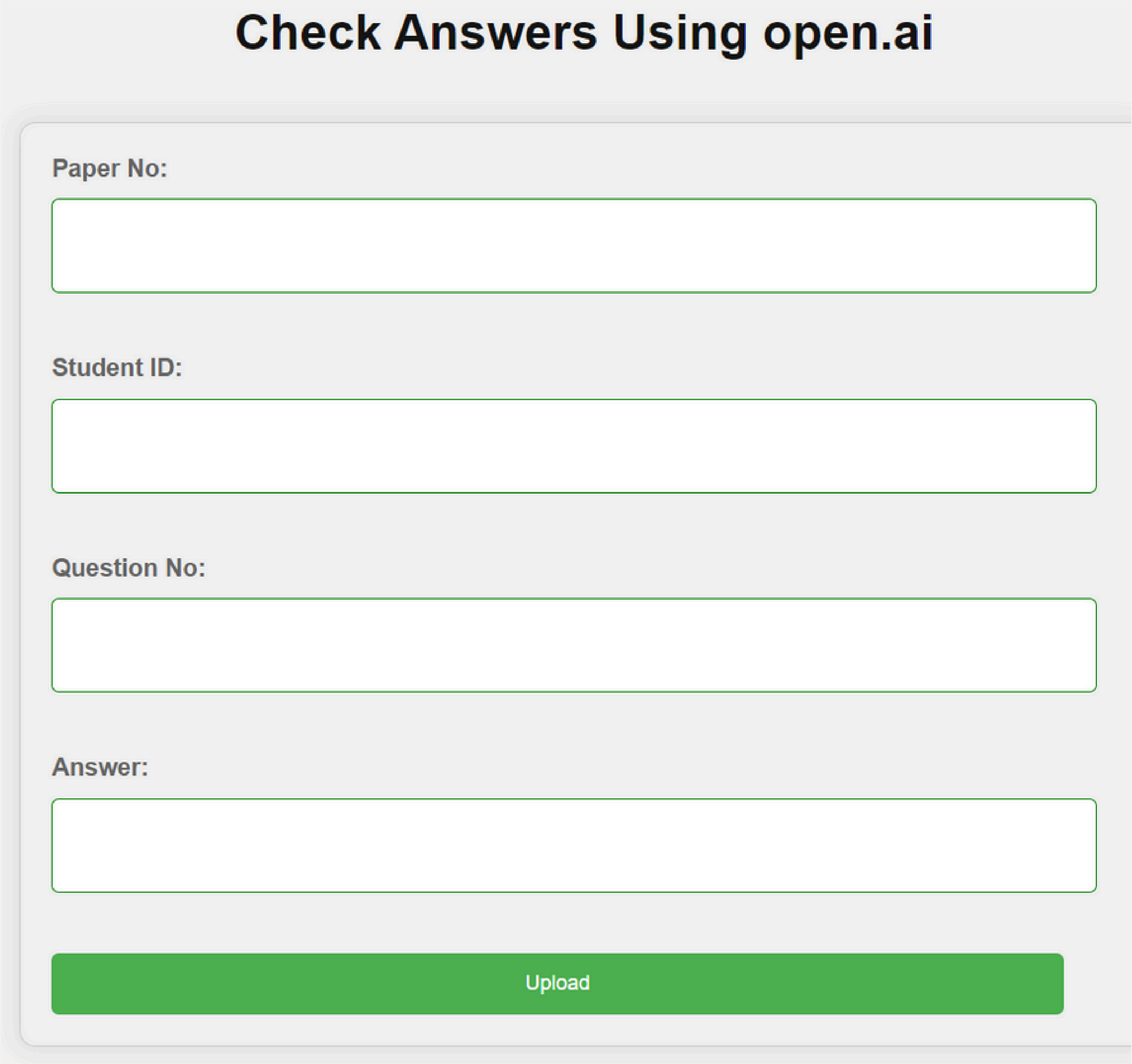
Hardav Raval

Github :

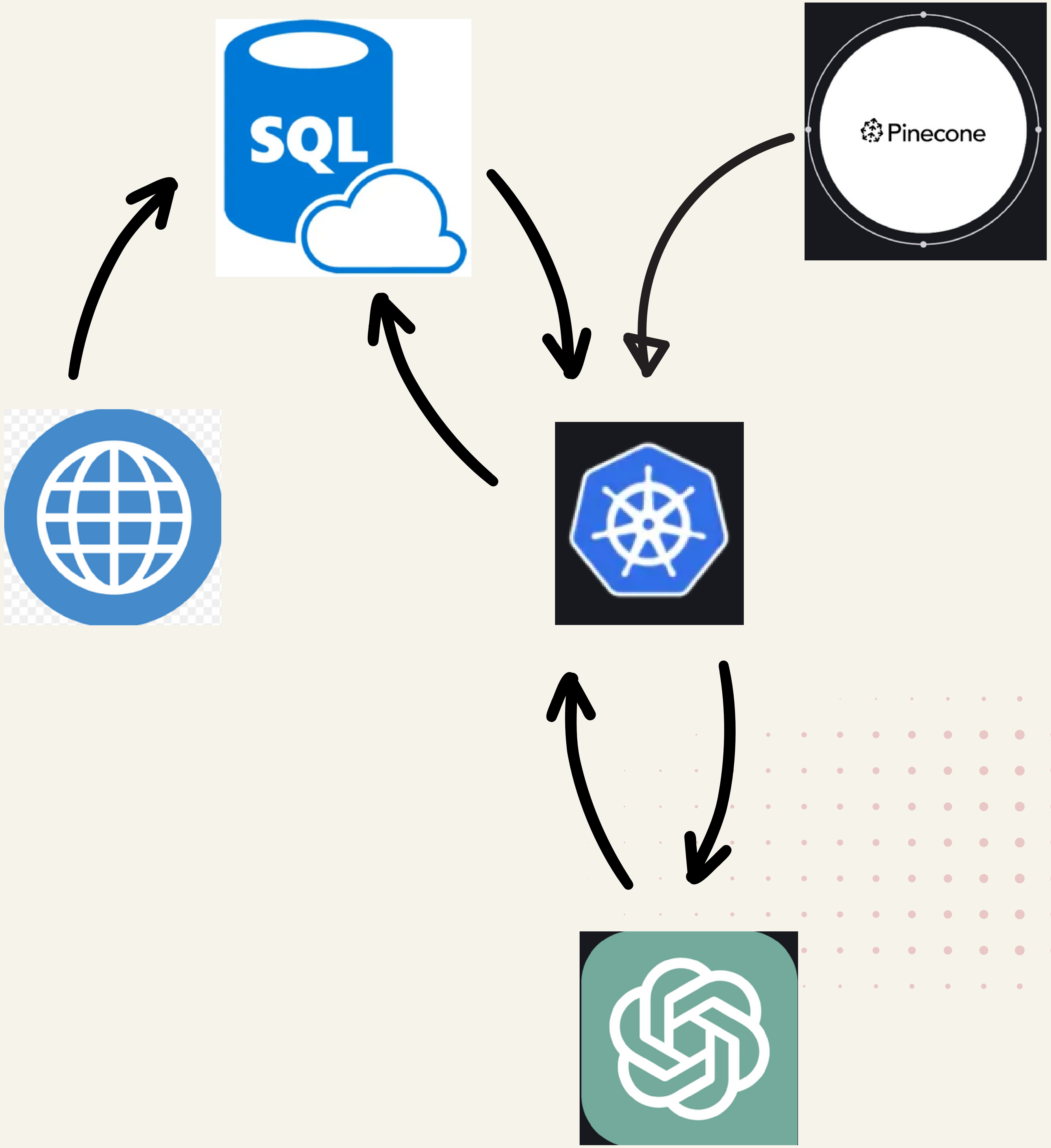
https://github.com/SaketNer/RCAzureGradert

PROJECT OVERVIEW

- The project is to create a scalable paper grading system using Cloud and OpenAi.
- We have used Azure Kubernetes, Azure Web App, Openai, and Pinecone.
- The answers, student ID, paper number, and question number are stored in an Azure SQL database. The official answers to the exam questions are also uploaded to the database.



- Utilizes a subject-related book uploaded to Pinecone for reference during evaluation.
- Kubernetes manages retrieval of student answers and official answers from the database.
- Kubernetes also fetches relevant context from the Pinecone book.
- ChatGPT is invoked by Kubernetes to evaluate student answers using the collected data.
- Marks assigned by ChatGPT are stored in the Azure SQL database.



SYSTEM ARCHITECTURE

The system architecture consists of the following components:

1. **Website**: A website that allows students to submit their descriptive answers to exam questions. The website also collects the student ID, paper number, and question number for each answer.
2. **Azure SQL Database**: A relational database in Azure that stores the student answers, official answers, and other relevant data.
3. **Pinecone**: A vector database that stores the book related to the exam subject. The context from the book is retrieved using Pinecone.
4. **Kubernetes**: A container orchestration system that pulls requests from the Azure SQL database and calls ChatGPT to evaluate the student answers.
5. **ChatGPT**: A language model developed by OpenAI that is used to evaluate the student answers and assign marks.

Azure Functions Vs Kubernetes

We chose Kubernetes for our model due to several key benefits:

- **Flexibility**: Kubernetes provides more flexibility and control over the deployment and scaling of the application compared to Azure Functions. With Kubernetes, we can easily configure and manage the resources required for the application, such as CPU, memory, and storage.

- **Scalability:** Kubernetes is designed to handle large-scale, distributed applications, making it a better choice for our project, which involves evaluating a large number of student answers.

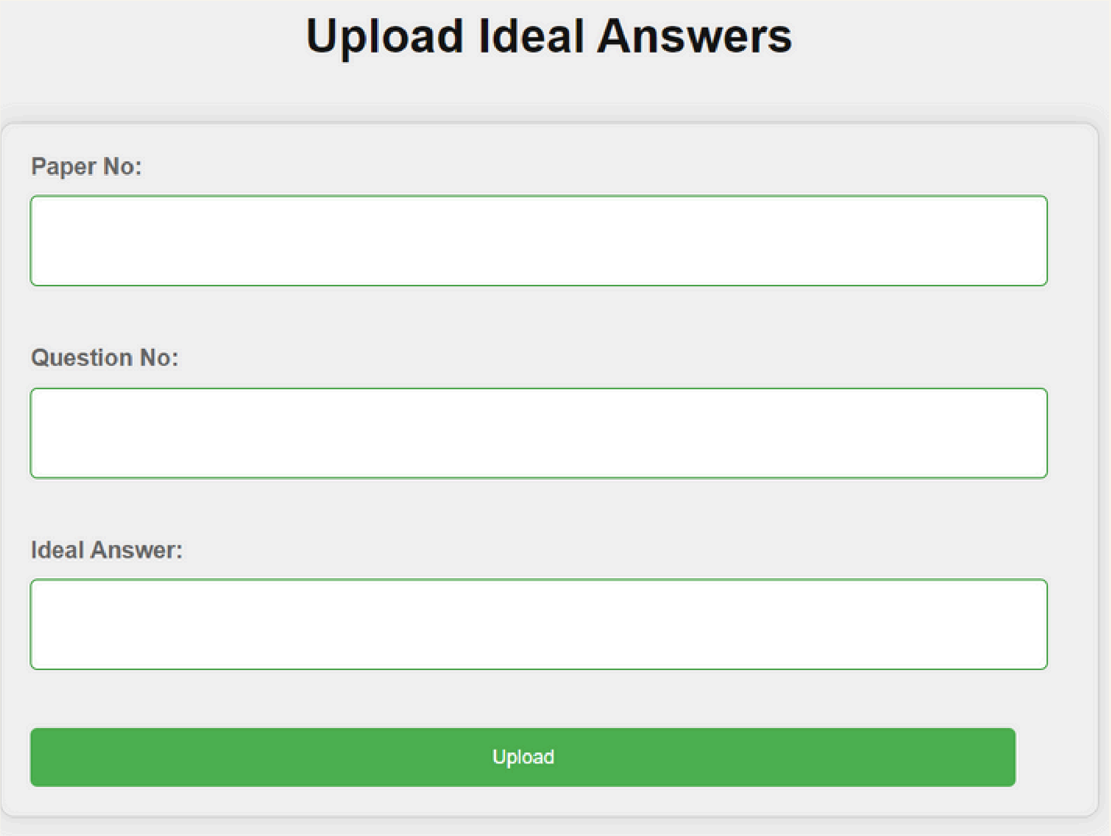
Feature	Azure Functions	Kubernetes
Deployment	Easy	Complex
Management	Simple	Advanced
Customization	Limited	High
Scalability	Automatic	Manual
Cost	Pay-per-use	Upfront costs
Integration	Azure-focused	Multi-cloud

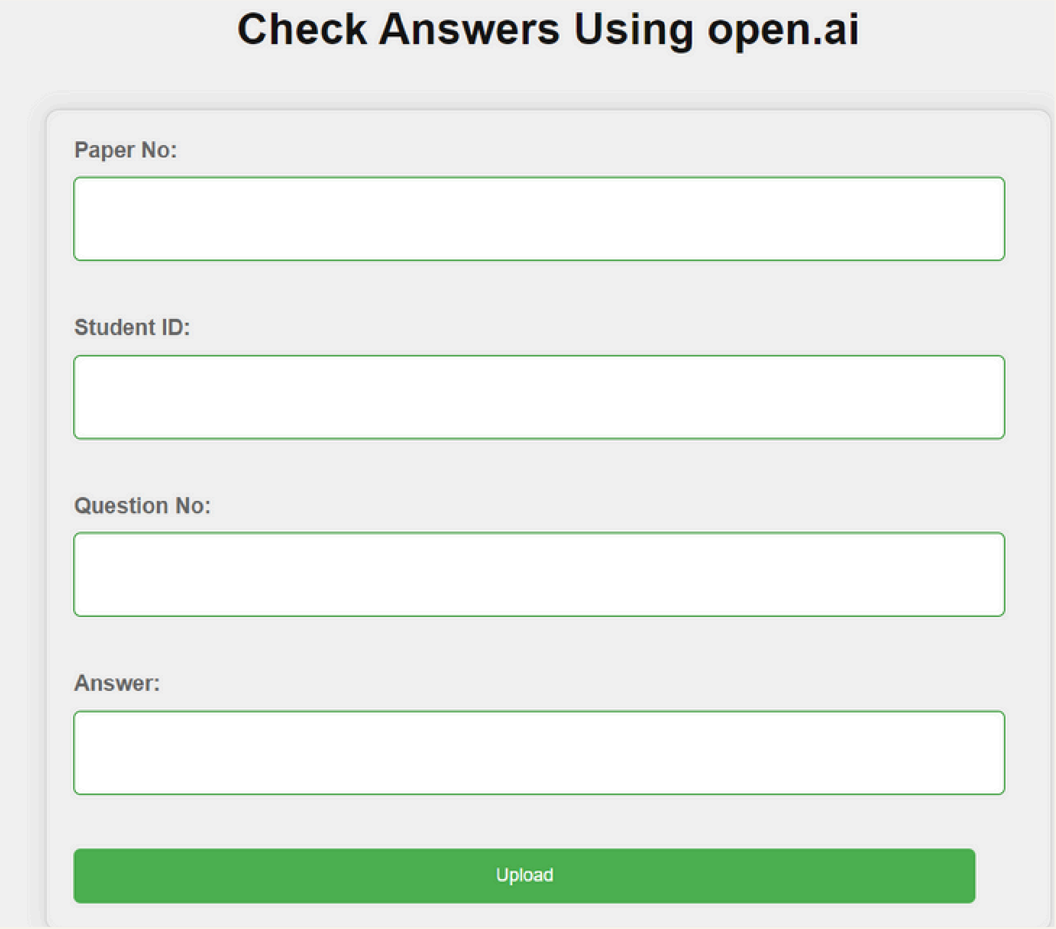
- **Integration:** Kubernetes provides a rich ecosystem of tools and integrations, making it easier to connect with other services and systems.
- **Portability:** Kubernetes is a cloud-native platform that can run on any cloud provider or on-premises infrastructure, providing greater portability and flexibility for the application.

WORKFLOW

We've developed two user-friendly websites to streamline the exam process:
one for students to upload their answers and **another for professors** to submit official answers.

These platforms were crafted with **HTML, CSS, and Flask**, ensuring intuitive navigation and seamless functionality for both students and professors.





- The data collected from these websites is stored in separate tables within the **Azure SQL database:**

- one table for student answers and another for the ideal answers.

o	student_id	question_no	answer
	1	1	cookie technology
	2	1	Cookie technology
	1	2	three-way handsha
	2	2	In the "three-way h

Student answers

paper_no	question_no	answer
1	1	cookie technology h...
1	2	three-way handshak...

Ideal answers

Executing **Kubernetes** code orchestrates the invocation of **ChatGPT**, which is provided with **student answers**, **ideal answers**, and the **grading scheme.**

Subsequently, **ChatGPT** autonomously assigns grades to students along with **detailed explanations** for each assessment.

```
def upload_marks(paper_no,student_id,quesntion_no,marks,reason):
    cursor = conn.cursor()
    print("uploading")
    cursor.execute("INSERT INTO final_marks (paper_no,student_id,question_no,marks,reason) VALUES


def check_ans(ideal_answer, student_answer):
    context = get_context(ideal_answer)
    text = f"You are a highly experienced professor in the field of computer science, tasked wit
    #print(text)
    client = OpenAI(api_key="sk-proj-W0lHK8EgUiZY0QxMtYc8T3BlbkFJFDOiSZnoED53RX4pf14N")

    completion = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=[
            {"role": "system", "content": "You are a professor grading answer papers of college 
            {"role": "user", "content":text}
        ],
        temperature= 0
    )
    print(completion.choices[0].message.content)
    split_text = completion.choices[0].message.content.split(',', 1)
    marks_split = split_text[0].split(' ')
    marks = marks_split[1]
    print("marks give are ", int(marks))
```

Utilizing Python scripts, we fetch and compute grades for each professor, meticulously storing them in designated SQL tables.

	student_id	question_no	marks	reason
	1	1	6	Reasons: The studen...
	2	1	8	Reasons: The studen...
	1	2	8	Reasons: The studen...
	2	2	8	Reasons: The studen...

Getting marks from the sql databse for each proff

```
Attempting Connection
connection done
   student_id  marks
0           1    104
1           2    106
(.venv) saket@Sakets-MacBook
```

Alternative Approach using LangChain

We alternatively tried 2 more approaches :

1. Using a simple retriever for RAG, the context is derived from the ideal answer as reference. Then making a call to chatGPT to return us a score and explaination behind the score. This we tested out however the results were not very satisfactory. Much more work was needed on the prompt.
2. Creating a Sequential chain which would perform RAG in the first chain and then using that context call the second chain along with the ideal and student answer as the input to then return a score and explanation for the same. We ran into a roadblock here as we were unable to figure out how to pass in the context from the first chain along with the student and ideal answers that we were passing when the whole chain was first called to the second chain.

Link for the colab notebook:

https://colab.research.google.com/drive/1EZD9HaJBCR8LzjrDw0iTXO-3xoIk4TJj?authuser=0#scrollTo=ZOsNsdrGMrPd

RAG.ipynb

File Edit View Insert Runtime Tools Help All changes saved

\+ Code + Text

FINAL FUNCTION

```python
import warnings
warnings.filterwarnings('ignore')

import os
openai_api_key="sk-proj-W0lHK8EgUiZY0QxMtYc8T3BlbkFJFDOiSZnoED53RX4pf14N"
#!pip install langchain
#!pip install faiss-cpu
#!pip install -U langchain-openai
#from dotenv import load_dotenv, find_dotenv
#_ = load_dotenv(find_dotenv()) # read local .env file

from langchain_openai import ChatOpenAI
llm_model = "gpt-3.5-turbo"
llm = ChatOpenAI(temperature=0.9, model=llm_model, openai_api_key=openai_api_key)

from langchain.chains.combine_documents import create_stuff_documents_chain
from langchain.embeddings import OpenAIEmbeddings
from langchain.prompts import PromptTemplate
from langchain.chat_models import ChatOpenAI
from langchain.prompts import ChatPromptTemplate
from langchain.chains import LLMChain , SequentialChain
from langchain_community.vectorstores import FAISS
from langchain.schema import Document
from langchain_core.messages import HumanMessage

#text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=200)
#splits = text_splitter.split_documents(docs)
#vectorstore = Chroma.from_documents(documents=splits, embedding=OpenAIEmbeddings())
```

```python
document = Document(page_content="The Indian Independence Movement, was a series of historic events in South Asia with the ul

# get answers from sql
ideal_answer = "India became independent in 1947 from British Rule. "
student_answer = "India became independent in 1962 from Chinese Rule. The Nationalistic movement started in Uttar Pradesh"

# Embed the document
documents = [document]
embeddings = OpenAIEmbeddings(openai_api_key=openai_api_key)
vectorstore = FAISS.from_documents(documents=documents, embedding=embeddings)


#embed = embeddings.embed_query(ideal_answer)
retriever = vectorstore.as_retriever()

llm = ChatOpenAI(temperature=0.9, model=llm_model,openai_api_key=openai_api_key)

context = retriever.invoke(ideal_answer)
print(context)
qa_template = PromptTemplate(
    template=""" You are an answer evaluation agent. \
     Evaluate the student's answer :{student_answer} based on the \
    ideal answer: {ideal_answer} and {context}.
    Return a JSON object with a score (integer between 0 and 5) and an explanation (string). \
    """,
    input_variables=["student_answer", "ideal_answer", "context"]
)

# Format the prompt with specific input values
qa_prompt = qa_template.format(student_answer=student_answer, ideal_answer=ideal_answer, context=context)

messages = [
    HumanMessage(content=qa_prompt),
]

output = llm(messages)
print(output)
```

THANK YOU