InsightBot

NLP Powered AI ChatBot

o5 December, 2024

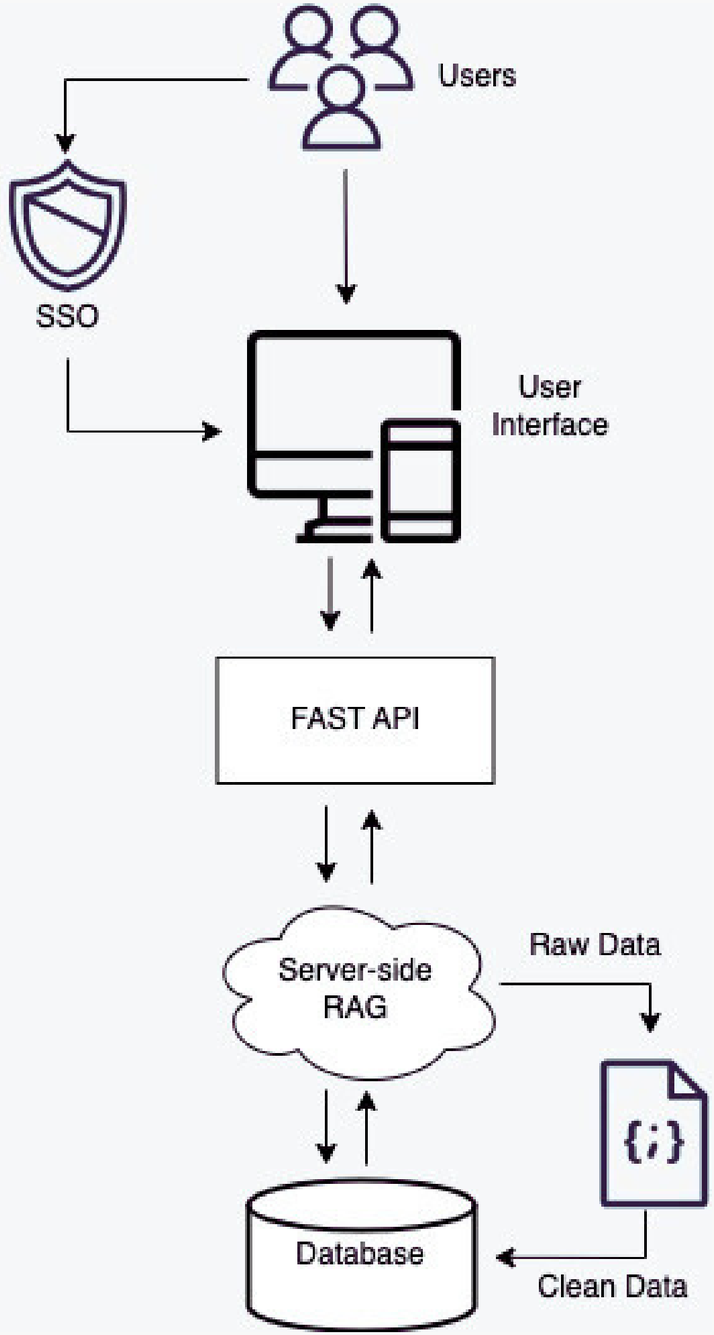
Presented By:
Soumya Bharathi Vetukuri
Rutuja Patil
Shubham Kothiya
Mann Nada

OBJECTIVES

- **Providing Instant Assistance:** Offering real-time responses to student queries related to academics, campus resources, events, and administrative services.
- **Centralizing Information:** Acting as a single point of contact for all university-related inquiries, eliminating the need to navigate multiple platforms or resources.
- **Improving Efficiency:** Reducing response times for common student concerns, such as course schedules, fee payment processes, and exam details, thereby enhancing productivity.
- **Personalized Support:** Utilizing AI to provide customized recommendations, such as suggesting campus events, academic resources, or career opportunities based on user input.
- **Enhancing Student Experience:** Simplifying the resolution of common challenges faced by students and improving their overall satisfaction with university services.

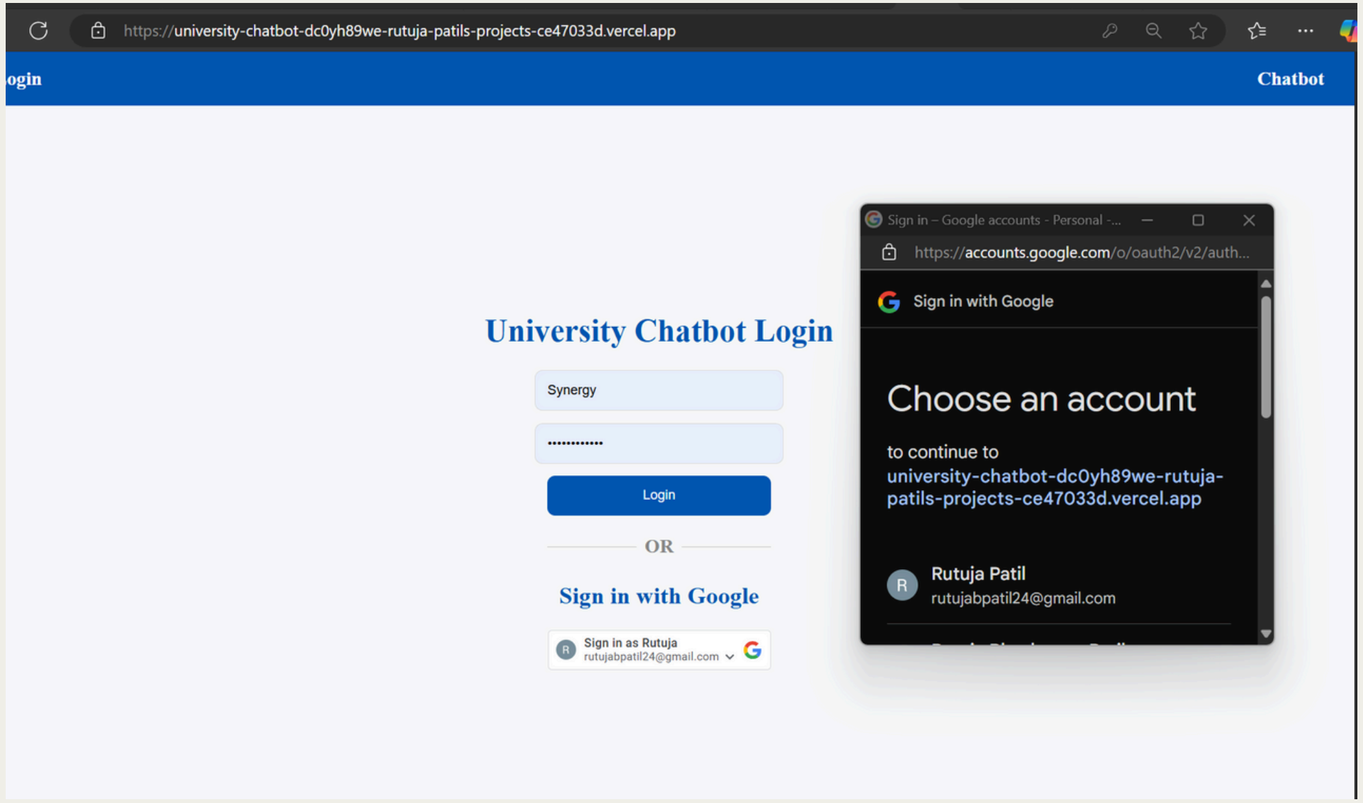
HIGH LEVEL ARCHITECTURE

- **Login Page:** User Credentials or SSO using Google
- **User Interface:** Chatbot Interface, Query Input and Response
- **Fast API:** Communication between the UI and database
- **Server-Side RAG:** AI-powered engine of the chatbot
- **Database:** MongoDB Cluster storing university information
- **CI-CD**: GitHub and Jenkins Integration, AWS Instance
- **Enhancement**: Google Calendar



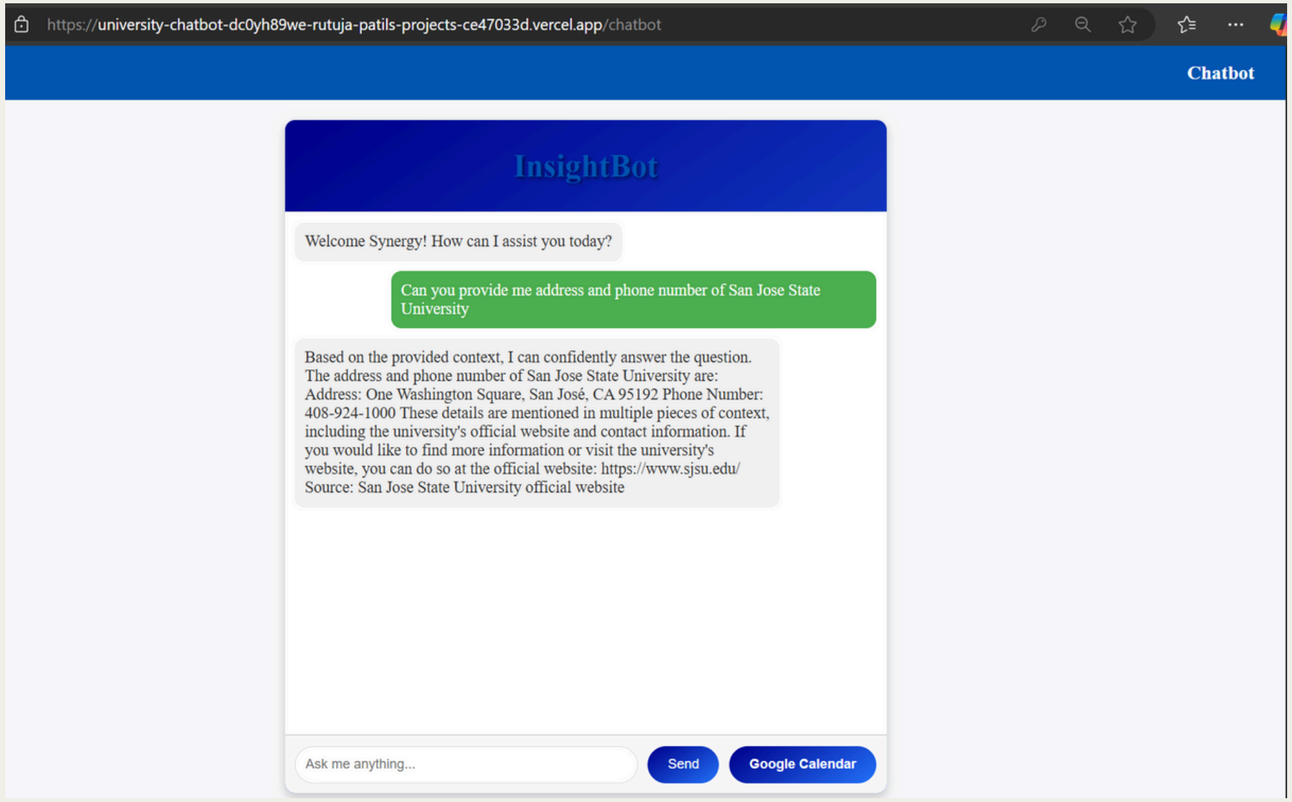
LOGIN - SSO

- **Access User Interface via HTTPS.**
- **User Credentials:** Existing User's Username, Password
- **Google OAuth SSO Integration**



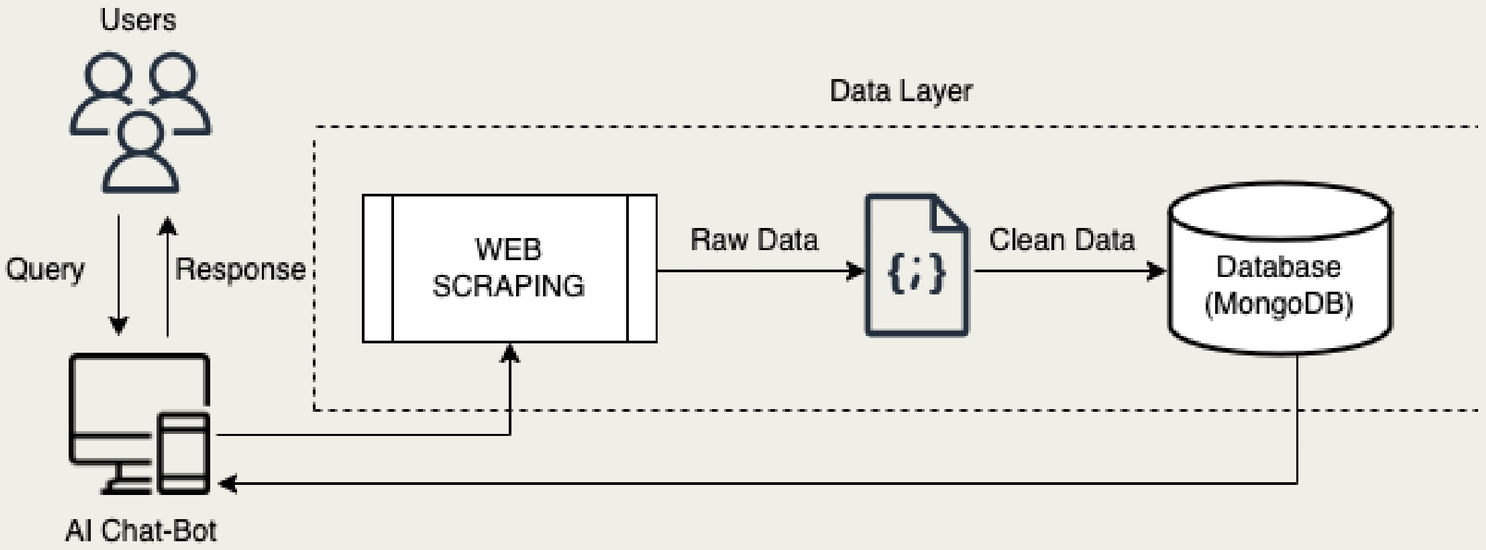
CHATBOT USER INTERFACE

- Userfriendly query and response based interface for students.
- Used Axios to make HTTP POST requests to the Fast API backend for real-time query resolution.
- Managed API responses and displayed bot replies in the chat window.
- Added a dedicated button in the UI, redirecting users to Google Calendar for event scheduling.



FAST API

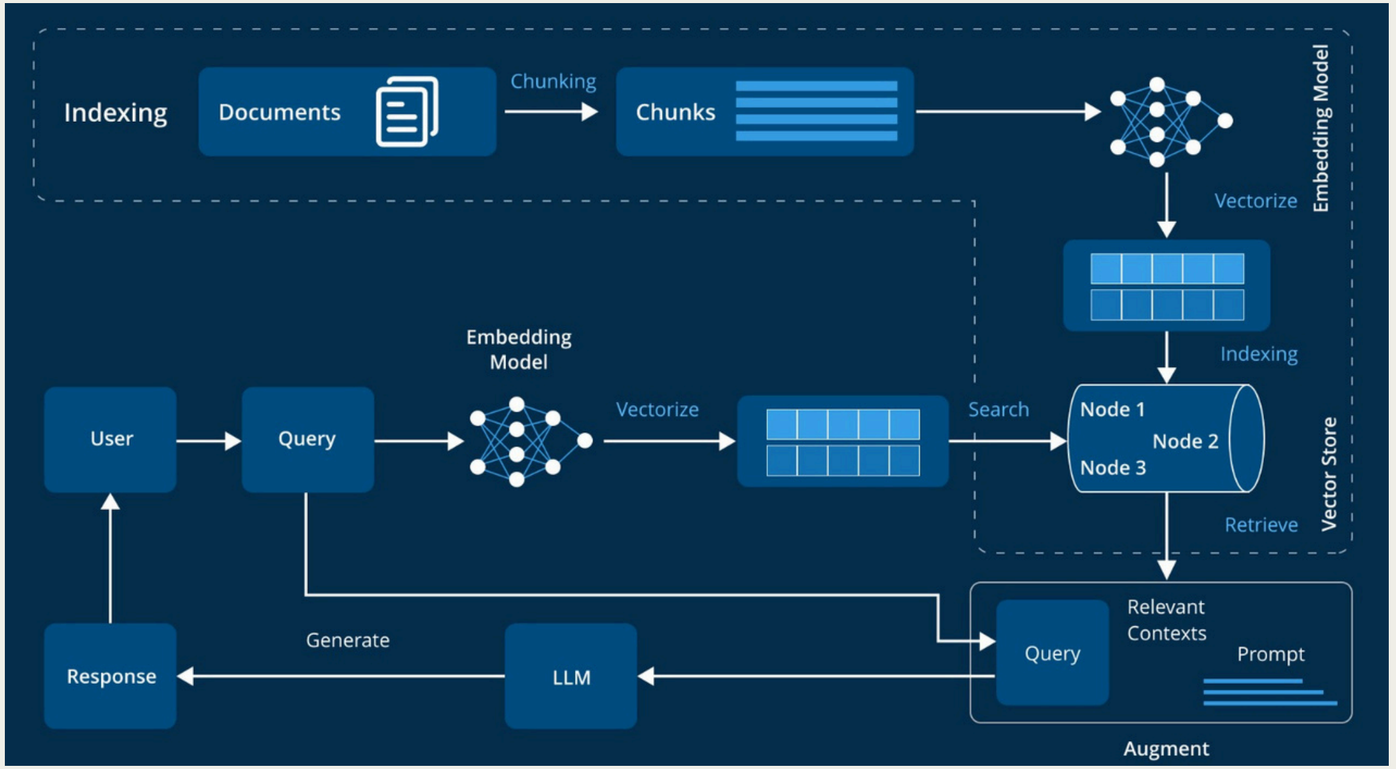
- High-performance web framework for building HTTP-based service APIs in Python.
- Automatic validation, documentation and type inference.
- UI <> API <> Web Service

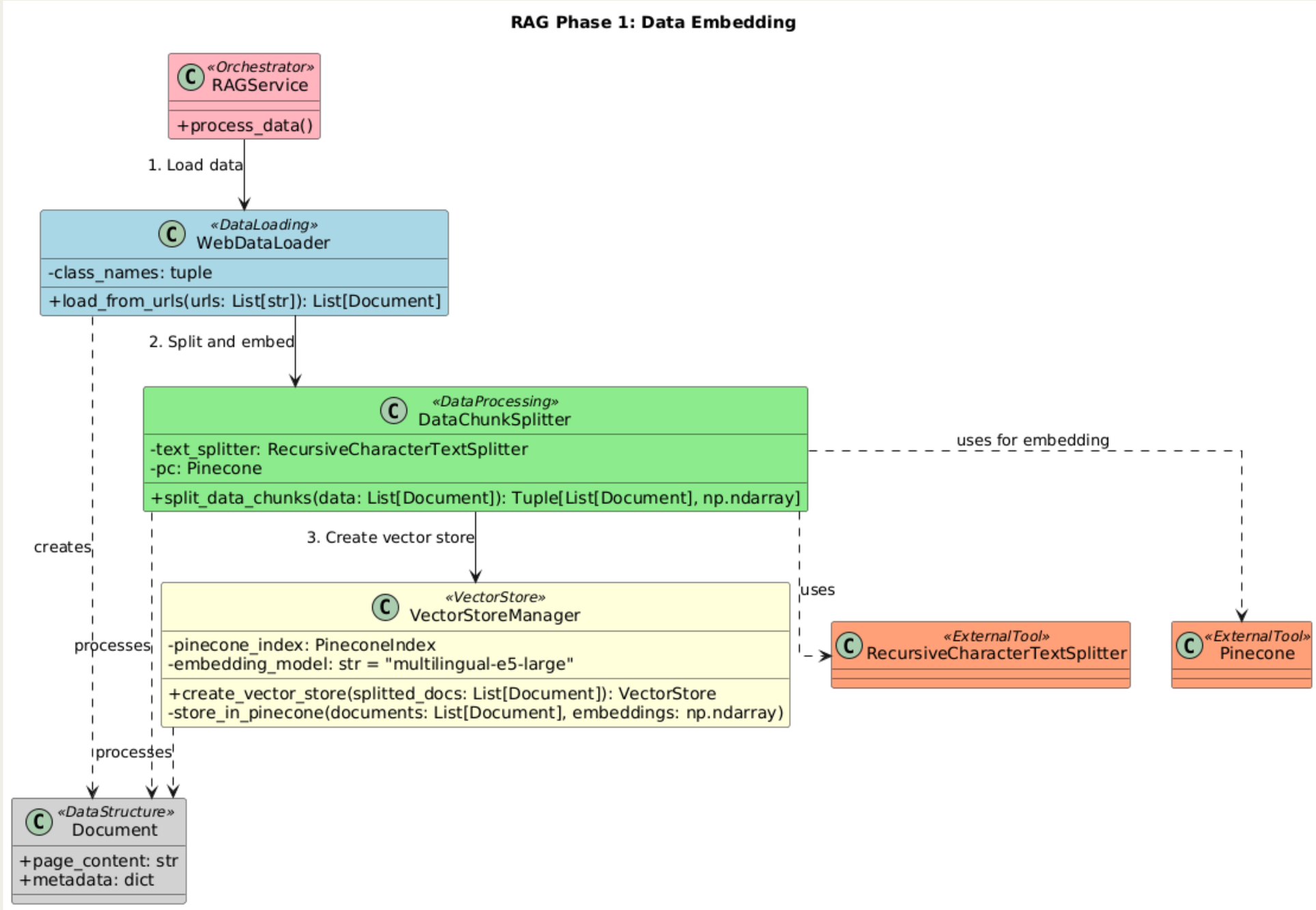


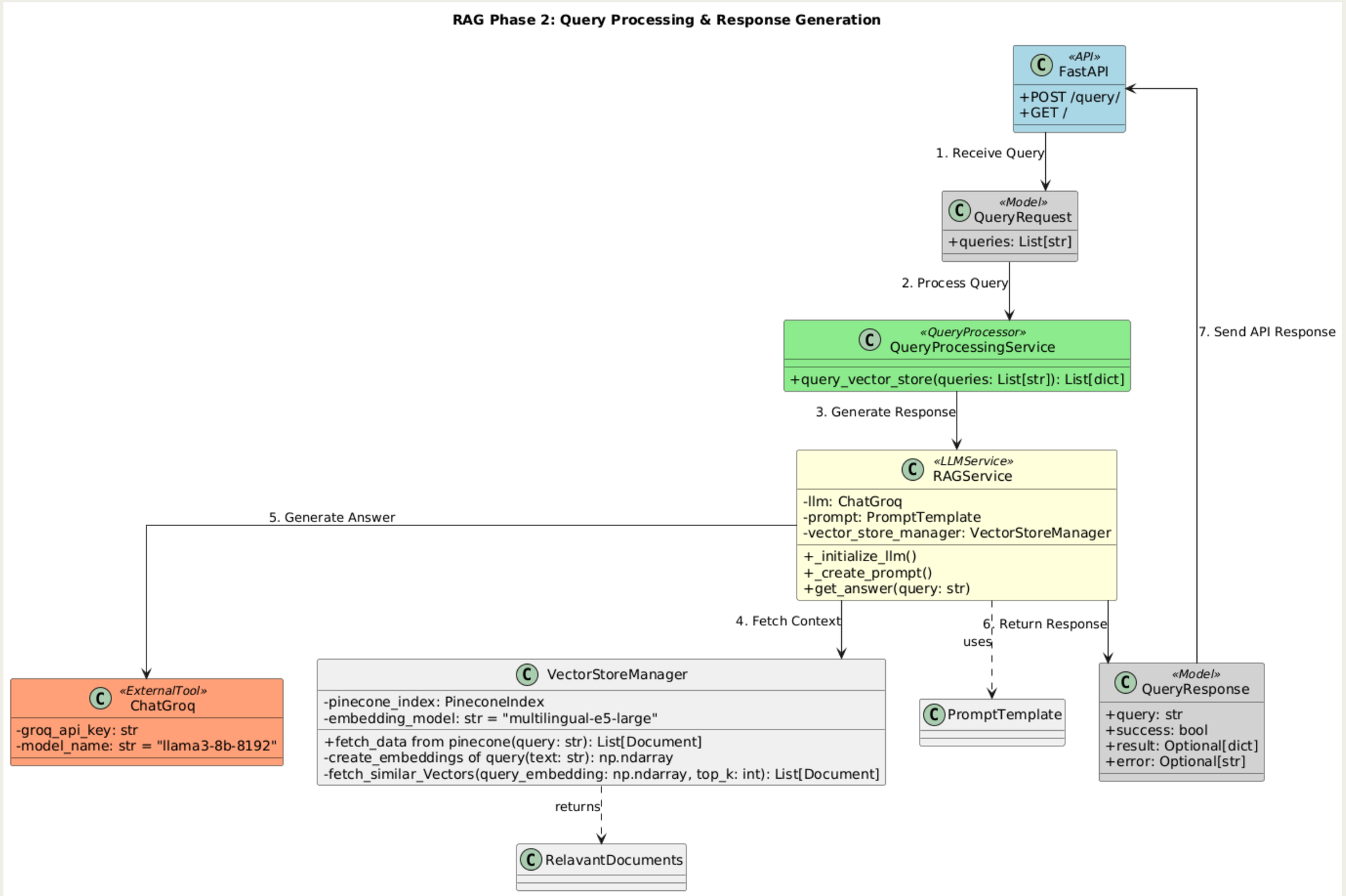

```
try {
  setLoading(true);
  const response = await axios.post("http://127.0.0.1:8000/query/", {
    queries: [userInput],
  });
```

SERVER SIDE - RAG

- AI framework that combines large language models (LLMs) with information retrieval systems to improve the accuracy and relevance of text.
- RAG architecture works by: Retrieving information: Using search algorithms to query external data like databases, knowledge bases, and web pages
- Pre-processing: Preparing the retrieved information
- Integrating: Incorporating the pre-processed information into the LLM

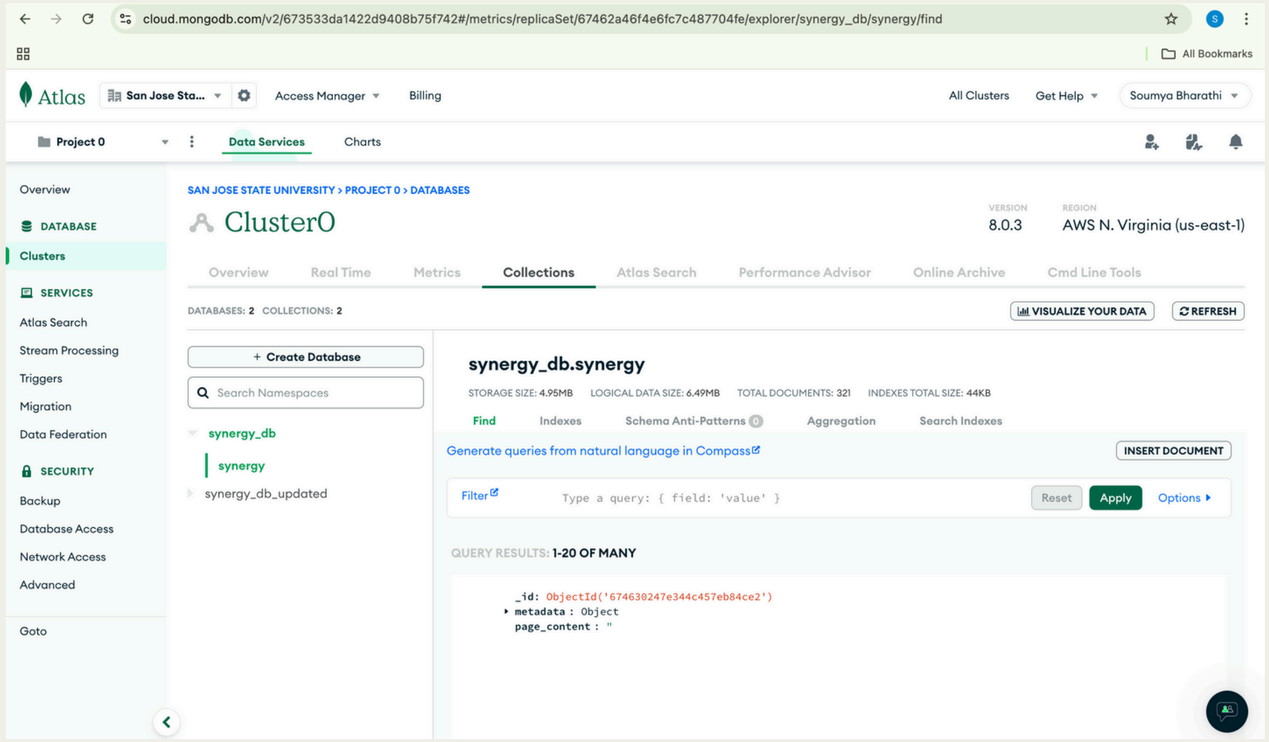



RAG Phase 1: Data Embedding
«Orchestrator»
RAGService
+process_data()
1. Load data
«DataLoading»
WebDataLoader
-class_names: tuple
+load_from_urls(urls: List[str]): List[Document]
2. Split and embed
«DataProcessing»
DataChunkSplitter
-text_splitter: RecursiveCharacterTextSplitter
-pc: Pinecone
+split_data_chunks(data: List[Document]): Tuple[List[Document], np.ndarray]
uses for embedding
creates
3. Create vector store
«VectorStore»
VectorStoreManager
-pinecone_index: PineconeIndex
-embedding_model: str = "multilingual-e5-large"
+create_vector_store(splitted_docs: List[Document]): VectorStore
-store_in_pinecone(documents: List[Document], embeddings: np.ndarray)
uses
processes
«ExternalTool»
RecursiveCharacterTextSplitter
«ExternalTool»
Pinecone
processes
«DataStructure»
Document
+page_content: str
+metadata: dict

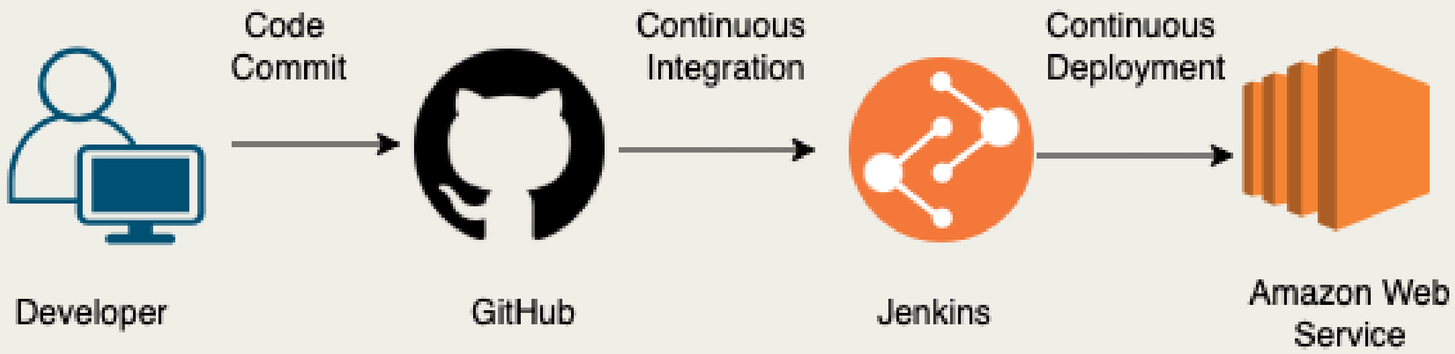

RAG Phase 2: Query Processing & Response Generation
«API»
FastAPI
+POST /query/
+GET /
1. Receive Query
«Model»
QueryRequest
+queries: List[str]
2. Process Query
«QueryProcessor»
QueryProcessingService
+query_vector_store(queries: List[str]): List[dict]
7. Send API Response
3. Generate Response
«LLMService»
RAGService
-llm: ChatGroq
-prompt: PromptTemplate
-vector_store_manager: VectorStoreManager
+_initialize_llm()
+_create_prompt()
+get_answer(query: str)
5. Generate Answer
4. Fetch Context
6. Return Response
uses
«ExternalTool»
ChatGroq
-groq_api_key: str
-model_name: str = "llama3-8b-8192"
VectorStoreManager
-pinecone_index: PineconeIndex
-embedding_model: str = "multilingual-e5-large"
+fetch_data from pinecone(query: str): List[Document]
-create_embeddings of query(text: str): np.ndarray
-fetch_similar_Vectors(query_embedding: np.ndarray, top_k: int): List[Document]
PromptTemplate
«Model»
QueryResponse
+query: str
+success: bool
+result: Optional[dict]
+error: Optional[str]
returns
RelavantDocuments

DATABASE

- Output Of RAG web scrape: The raw data file is retrieved.
- Clean the raw data using Python code.
- The cleaned data is fed to Mongo DB database.
- Org > Project > Database > Collections
- A collection is a set of documents, which is similar to a table in a relational database.

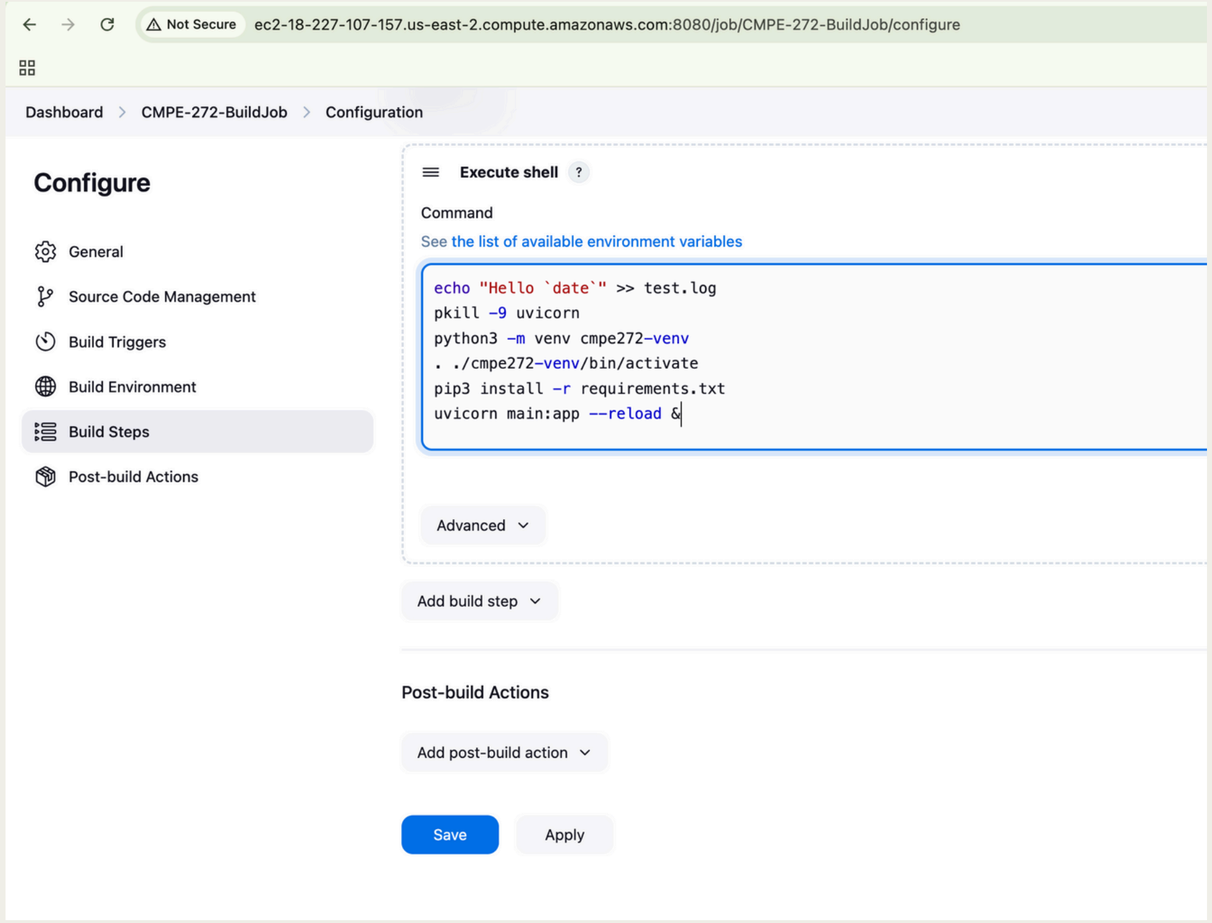


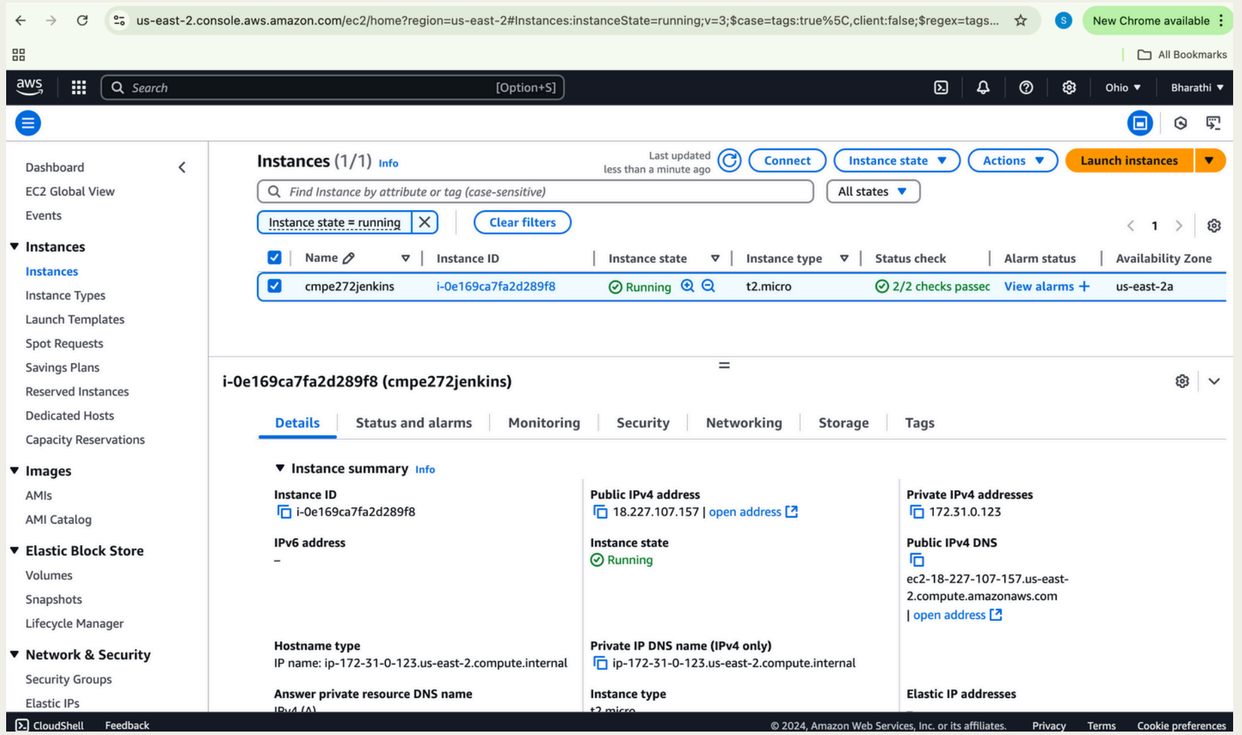
CI-CD PIPELINE



- GitHub Code Repository
- Integrated Jenkins with GitHub Repo for Continuous Integration
- Code Commit to GitHub repo invokes Jenkins Build Job
- Continuous Deployment to Ubuntu Server hosted in AWS

CI-CD SETUP


Not Secure
ec2-18-227-107-157.us-east-2.compute.amazonaws.com:8080/job/CMPE-272-BuildJob/configure
Dashboard
CMPE-272-BuildJob
Configuration
Configure
General
Source Code Management
Build Triggers
Build Environment
Build Steps
Post-build Actions
Execute shell
Command
See the list of available environment variables
echo "Hello `date`" >> test.log
pkill -9 uvicorn
python3 -m venv cmpe272-venv
. ./cmpe272-venv/bin/activate
pip3 install -r requirements.txt
uvicorn main:app --reload &
Advanced
Add build step
Post-build Actions
Add post-build action
Save
Apply


us-east-2.console.aws.amazon.com/ec2/home?region=us-east-2#Instances:instanceState=running;v=3;$case=tags:true%5C,client:false;$regex=tags...
New Chrome available
Search
[Option+S]
Ohio
Bharathi
Dashboard
EC2 Global View
Events
Instances
Instances
Instance Types
Launch Templates
Spot Requests
Savings Plans
Reserved Instances
Dedicated Hosts
Capacity Reservations
Images
AMIs
AMI Catalog
Elastic Block Store
Volumes
Snapshots
Lifecycle Manager
Network & Security
Security Groups
Elastic IPs
Instances (1/1) Info
Last updated less than a minute ago
Connect
Instance state
Actions
Launch instances
Find Instance by attribute or tag (case-sensitive)
All states
Instance state = running
Clear filters
Name
Instance ID
Instance state
Instance type
Status check
Alarm status
Availability Zone
cmpe272jenkins
i-0e169ca7fa2d289f8
Running
t2.micro
2/2 checks passec
View alarms
us-east-2a
i-0e169ca7fa2d289f8 (cmpe272jenkins)
Details
Status and alarms
Monitoring
Security
Networking
Storage
Tags
Instance summary Info
Instance ID
i-0e169ca7fa2d289f8
Public IPv4 address
18.227.107.157 | open address
Private IPv4 addresses
172.31.0.123
IPv6 address
–
Instance state
Running
Public IPv4 DNS
ec2-18-227-107-157.us-east-2.compute.amazonaws.com | open address
Hostname type
IP name: ip-172-31-0-123.us-east-2.compute.internal
Private IP DNS name (IPv4 only)
ip-172-31-0-123.us-east-2.compute.internal
Answer private resource DNS name
Instance type
Elastic IP addresses
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

Demo

Enhancements and Future Scope

- Help students by integrating more universities information.
- Improvise the RAG retrieval system to generate more precise results.
- Enhance by adding more additional features like Payment System, Google Maps for university location etc.
- Include a feedback mechanism for students to rate responses or provide feedback to improve the chatbot’s learning.
- Add multilingual capabilities to cater to a diverse student population by integrating language models

Thank you!

- TEAM SYNERGY