**Database Design (MySQL/PostgreSQL):**

* **Schema Design**: You need two main tables for this project: products and suppliers.

**Supplier Table**:

sql

Copy code

CREATE TABLE suppliers (

id INT PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR(255) NOT NULL,

contact\_info TEXT,

product\_categories\_offered TEXT

);

**Product Table**:

sql

Copy code

CREATE TABLE products (

id INT PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR(255) NOT NULL,

brand VARCHAR(255),

price DECIMAL(10, 2),

category VARCHAR(255),

description TEXT,

supplier\_id INT,

FOREIGN KEY (supplier\_id) REFERENCES suppliers(id)

);

**Populate with Sample Data**: Populate both tables with mock data for testing. For instance, a few sample suppliers and their associated products.

**2. Backend Setup (Python & FastAPI):**

* **FastAPI Setup**: Install dependencies:

bash

Copy code

pip install fastapi sqlalchemy mysql-connector-python langgraph requests uvicorn

* **Create FastAPI App**:
  + Start by creating the FastAPI app to expose endpoints for the chatbot. You'll need at least one endpoint to accept queries from the frontend.

Example API endpoint:

python

Copy code

from fastapi import FastAPI

from pydantic import BaseModel

import langgraph

import requests

app = FastAPI()

class Query(BaseModel):

query: str

@app.post("/api/chat")

async def chat(query: Query):

# Here, integrate LangGraph to process the query and return a response.

# For now, return a dummy response

response = "Fetching product and supplier info based on query: " + query.query

return {"response": response}

**3. LangGraph Setup:**

* **Install LangGraph**:

bash

Copy code

pip install langgraph

* **LangGraph Node for Querying Database**: You'll create a LangGraph node to interact with the database to fetch supplier and product details.

Example LangGraph node:

python

Copy code

from langgraph import LangGraph

from sqlalchemy.orm import sessionmaker

from sqlalchemy import create\_engine

graph = LangGraph()

DATABASE\_URL = "mysql+mysqlconnector://username:password@localhost/dbname"

engine = create\_engine(DATABASE\_URL)

SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

@graph.node

def get\_product\_details(product\_id: int):

session = SessionLocal()

product = session.query(Product).filter(Product.id == product\_id).first()

return product

* **LangGraph for LLM Integration**: You can use an open-source LLM (e.g., Hugging Face's GPT-2, LLaMA 2) to summarize or enhance the information retrieved from the database. Integrate LLM into the LangGraph nodes.

**1. React Setup:**

* **Install React and Axios**: Use create-react-app to set up your project:

bash

Copy code

npx create-react-app chatbot-app

cd chatbot-app

npm install axios @mui/material

* **Create Chat Interface**: Set up a basic UI to allow users to input queries and display chatbot responses. Use Material UI for components like TextField and List.

Example:

js

Copy code

import React, { useState } from 'react';

import { TextField, Button, List, ListItem } from '@mui/material';

import axios from 'axios';

const Chatbot = () => {

const [query, setQuery] = useState('');

const [chatHistory, setChatHistory] = useState([]);

const handleQuerySubmit = async () => {

const response = await axios.post('http://localhost:8000/api/chat', { query });

setChatHistory([...chatHistory, { user: query, bot: response.data.response }]);

setQuery('');

};

return (

<div>

<List>

{chatHistory.map((chat, index) => (

<ListItem key={index}>

<div><strong>You:</strong> {chat.user}</div>

<div><strong>Bot:</strong> {chat. Bot}</div>

</List Item>

))}

</List>

<Text Field

value={query}

unchanged={(e) => setQuery(e.target.value)}

label="Ask a question"

fullWidth

/>

<Button onClick={handleQuerySubmit}>Ask</Button>

</div>

);

};

export default Chatbot;

**2. Connecting Frontend to Backend:**

* Use Axios to send requests from the React frontend to the FastAPI backend. The backend will then use LangGraph to process the query and return the appropriate response.

**1. Implement Query Logic:**

* In the FastAPI backend, refine the query processing logic to fetch supplier/product data from the database based on the user's query.

Example for handling queries like "Show me all products under brand X":

python

Copy code

@app.post("/api/chat")

async def chat(query: Query):

if "products under brand" in query.query:

brand = query.query.split("under brand")[-1].strip()

products = db.session.query(Product).filter(Product.brand == brand).all()

return {"response": [product.name for product in products]}

# Add more conditional logic for other queries

**2. LLM Integration for Summarization:**

* Once data is retrieved, pass it to the LLM for summarization. You can use Hugging Face's API or an open-source model for this task.

Example with Hugging Face API:

python

Copy code

def summarize\_with\_llm(text):

response = requests.post(

"https://api-inference.huggingface.co/models/gpt-2",

headers={"Authorization": "Bearer YOUR\_HUGGINGFACE\_API\_KEY"},

json={"inputs": text}

)

return response.json()[0]["generated\_text"]

}

Name: Varun s

Whatsapp number:8762829241