NextGenLead_4_Modified

The project will be divided into four main files:

1. **app.py**: Main application file with API routes.
2. **models.py**: Contains database models.
3. **config.py**: Database configuration.
4. **requirements.txt**: File for project dependencies.

PROJECT STRUCTURE:

```
customer_service/
├── app.py # Main application file
├── models.py # SQLAlchemy models
├── config.py # Database configuration
└── requirements.txt # Dependency file
```

1. `config.py`: Database Configuration

This file contains the PostgreSQL connection configuration.

```python
# config.py

"""
Database configuration for connecting to PostgreSQL using SQLAlchemy.
"""

DATABASE_URL = 'postgresql://postgres:1234@localhost:5432/postgres'
```

Explanation:

- We define the `DATABASE_URL` that holds the connection string for the PostgreSQL database. This string includes the username, password, host, and database name.

2. `models.py`: Defining the Database Models

This file contains the SQLAlchemy models for `productenquiryforms_4` and `customerdetails_4`.

```python
# models.py

from sqlalchemy import Column, String, Integer, Date, BOOLEAN, BIGINT, Text, create_er
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker
from config import DATABASE_URL

"""
SQLAlchemy models for the 'productenquiryforms_4' and 'customerdetails_4' tables.
"""

# Create base class for models
Base = declarative_base()
```

```
# Create engine to connect to PostgreSQL
engine = create_engine(DATABASE_URL, echo=True)

# Create session for database interaction
Session = sessionmaker(bind=engine)
session = Session()

# Define the ProductEnquiryForms_4 model
class ProductEnquiryForms_4(Base):
    """
    Represents the 'productenquiryforms_4' table in the database.
    This table stores customer product inquiries.
    """
    __tablename__ = 'productenquiryforms_4'

    CustomerName = Column("customername", String)
    Gender = Column("gender", String)
    Age = Column("age", Integer)
    Occupation = Column("occupation", String)
    MobileNo = Column("mobileno", BIGINT, primary_key=True)
    Email = Column("email", String)
    VehicleModel = Column("vechiclemodel", String)
    State = Column("state", String)
    District = Column("district", String)
    City = Column("city", String)
    ExistingVehicle = Column("existingvehicle", String)
    DealerState = Column("dealerstate", String)
    DealerTown = Column("dealertown", String)
    Dealer = Column("dealer", String)
    BriefAboutEnquiry = Column("briefaboutenquiry", Text)
    ExpectedDateOfPurchase = Column("expecteddateofpurchase", Date)
    IntendedUsage = Column("intendedusage", String)
    SentToDealer = Column("senttodealer", BOOLEAN)
    DealerCode = Column("dealercode", String)
    LeadId = Column("leadid", String)
    Comments = Column("comments", Text)
    CreatedDate = Column("createddate", Date)
    IsPurchased = Column("ispurchased", BOOLEAN)

# Define the CustomerDetails_4 model
class CustomerDetails_4(Base):
    """
    Represents the 'customerdetails_4' table in the database.
    This table stores customer details related to inquiries.
    """
    __tablename__ = 'customerdetails_4'

    LeadId = Column("leadid", String)
    CustomerName = Column("customername", String)
    MobileNo = Column("mobileno", BIGINT, primary_key=True)
    City = Column("city", String)
    Dealer = Column("dealer", String)
    DealerCode = Column("dealercode", String)
    SentToDealer = Column("senttodealer", BOOLEAN)
```

```
# Create the tables in PostgreSQL
Base.metadata.create_all(engine)
```

- **Explanation**:
  - **Base**: The base class for all models using SQLAlchemy's `declarative_base()`.
  - **Session**: Creates a session to interact with the database.
  - **ProductEnquiryForms_4**: Model representing a table that stores customer product inquiry information.
  - **CustomerDetails_4**: Model representing a table that stores customer details.
  - `Base.metadata.create_all(engine)` creates both tables in the PostgreSQL database if they don't already exist.

3. `app.py`: Main Application with API Routes

```
# app.py

from flask import Flask, request, jsonify
from models import ProductEnquiryForms_4, CustomerDetails_4, session

"""
Main Flask application with API routes for managing records in the database.
"""

app = Flask(__name__)

# Route to insert records into both tables
@app.route('/post_records', methods=['POST'])
def post_records():
    """
    Inserts records into the 'productenquiryforms_4' and 'customerdetails_4' tables.

    Request:
    POST /post_records
    JSON Body:
    {
        "customername": "John Doe",
        "gender": "Male",
        "age": 30,
        "occupation": "Engineer",
        "mobileno": 9876543210,
        "email": "johndoe@example.com",
        "vechiclemodel": "Model X",
        "state": "California",
        "district": "District A",
        "city": "San Francisco",
        "existingvehicle": "None",
        "dealerstate": "California",
        "dealertown": "Town A",
        "dealer": "Dealer ABC",
        "briefaboutenquiry": "Interested in purchasing a new car",
        "expecteddateofpurchase": "2024-12-01",
        "intendedusage": "Personal",
```

```
        "senttodealer": false,
        "dealercode": "ABC123",
        "leadid": "LEAD12345",
        "comments": "Looking for more details",
        "createddate": "2024-09-07",
        "ispurchased": false,

        "customer_leadid": "LEAD12345",
        "customer_city": "San Francisco",
        "customer_dealer": "Dealer ABC",
        "customer_dealercode": "ABC123",
        "customer_senttodealer": false
    }
    """
    data = request.get_json()
    try:
        # Insert into ProductEnquiryForms_4
        product_record = ProductEnquiryForms_4(
            CustomerName=data['customername'],
            Gender=data['gender'],
            Age=data['age'],
            Occupation=data['occupation'],
            MobileNo=data['mobileno'],
            Email=data['email'],
            VehicleModel=data['vechiclemodel'],
            State=data['state'],
            District=data['district'],
            City=data['city'],
            ExistingVehicle=data['existingvehicle'],
            DealerState=data['dealerstate'],
            DealerTown=data['dealertown'],
            Dealer=data['dealer'],
            BriefAboutEnquiry=data['briefaboutenquiry'],
            ExpectedDateOfPurchase=data['expecteddateofpurchase'],
            IntendedUsage=data['intendedusage'],
            SentToDealer=data['senttodealer'],
            DealerCode=data['dealercode'],
            LeadId=data['leadid'],
            Comments=data['comments'],
            CreatedDate=data['createddate'],
            IsPurchased=data['ispurchased']
        )
        session.add(product_record)

        # Insert into CustomerDetails_4
        customer_record = CustomerDetails_4(
            LeadId=data['customer_leadid'],
            CustomerName=data['customername'],
            MobileNo=data['mobileno'],
            City=data['customer_city'],
            Dealer=data['customer_dealer'],
            DealerCode=data['customer_dealercode'],
            SentToDealer=data['customer_senttodealer']
        )
        session.add(customer_record)
```

```
        session.commit()
        return jsonify({"message": "Records added successfully!"}), 201
    except Exception as e:
        session.rollback()
        return jsonify({"error": str(e)}), 500

# Route to fetch all records
@app.route('/get_records', methods=['GET'])
def get_records():
    """
    Retrieves all records from 'productenquiryforms_4' and 'customerdetails_4' tables.
    """
    try:
        product_records = session.query(ProductEnquiryForms_4).all()
        customer_records = session.query(CustomerDetails_4).all()

        # Remove '_sa_instance_state' before returning the records
        product_list = [{k: v for k, v in record.__dict__.items() if k != '_sa_instanc
        customer_list = [{k: v for k, v in record.__dict__.items() if k != '_sa_instar

        return jsonify({
            "ProductEnquiryForms_4": product_list,
            "CustomerDetails_4": customer_list
        }), 200
    except Exception as e:
        return jsonify({"error": str(e)}), 500

# Run the application
if __name__ == '__main__':
    app.run(debug=True)
```

Explanation:

- **post_records**: Inserts data into both the `productenquiryforms_4` and `customerdetails_4` tables using JSON input.
- **get_records**: Retrieves all records from both tables and returns them as JSON after cleaning up the SQLAlchemy instance state.

Here's the continuation of the project setup with the `requirements.txt` file for managing dependencies, and a summary of how everything works together.

4. `requirements.txt`: Managing Dependencies

This file lists the project dependencies, so when others want to run the project, they can easily install them using `pip`.

```
Flask==2.0.1
SQLAlchemy==1.4.22
psycopg2==2.9.1
flask-restful==0.3.9
```

- **Flask**: Lightweight WSGI web application framework for creating the APIs.

- **SQLAlchemy**: Python SQL toolkit for working with databases using ORM.
- **psycopg2**: PostgreSQL adapter for Python to interact with the PostgreSQL database.
- **flask-restful**: Extension for Flask to create RESTful APIs.

1. `config.py`

- This file contains the configuration for connecting to the PostgreSQL database.
  - `DATABASE_URL`: A string containing the connection information for PostgreSQL.

2. `models.py`

- **Base**: The `declarative_base()` function from SQLAlchemy is used to define the base class for all ORM models.
- **engine**: The `create_engine` function creates a connection to the PostgreSQL database, using the connection URL defined in `config.py`.
- **Session**: Creates a session to interact with the database.
- **ProductEnquiryForms_4**: This class represents the `productenquiryforms_4` table. Each column is defined with a type (e.g., `String`, `Integer`, etc.).
  - The `MobileNo` column serves as the primary key, uniquely identifying each row.
- **CustomerDetails_4**: Represents the `customerdetails_4` table. The `MobileNo` column is also used as the primary key here.
- **Base.metadata.create_all(engine)**: This command creates both tables in the PostgreSQL database if they don't already exist.

3. `app.py`

- **Flask Setup**: Initializes the Flask application (`app = Flask(__name__)`).
- **Session**: The session object from `models.py` is imported for interaction with the database.
- **Route: `/post_records`**
  - This route handles the `POST` request to insert records into both the `productenquiryforms_4` and `customerdetails_4` tables.
  - **Logic**:
    - Data is extracted from the request JSON using `request.get_json()`.
    - Two records are created:
      1. One in the `productenquiryforms_4` table.
      2. One in the `customerdetails_4` table.
    - These records are added to the database session (`session.add()`).
    - After successful addition, `session.commit()` saves the changes.
    - If an error occurs, the session is rolled back (`session.rollback()`) to maintain database integrity.
- **Route: `/get_records`**
  - This route handles the `GET` request to fetch all records from both tables.
  - **Logic**:
    - All records are fetched using `session.query()` for both tables.
    - Before returning the records as JSON, the `_sa_instance_state` attribute (an internal SQLAlchemy attribute) is removed from each dictionary to avoid serialization errors.
    - The response contains two lists of records: one for `productenquiryforms_4` and one for

`customerdetails_4`.

- **Route: `/update_record/<mobileno>`**
  - This route handles the `PUT` request to update records based on the `MobileNo` provided in the URL.
  - **Logic**:
    - The record is queried by `MobileNo`. If found, it is updated with the new data provided in the request body (`data.items()`).
    - Changes are committed to the database after updating.
- **Route: `/delete_record/<mobileno>`**
  - This route handles the `DELETE` request to remove a record based on the provided `MobileNo`.
  - **Logic**:
    - The record is queried by `MobileNo`. If found, it is deleted from the session.
    - Changes are committed to remove the record permanently from the database.

4. `requirements.txt`

- This file is used to install the necessary libraries using `pip install -r requirements.txt`. It ensures the project dependencies (Flask, SQLAlchemy, psycopg2, etc.) are installed in your Python environment.

HOW IT ALL WORKS TOGETHER (SUMMARY FOR PRESENTATION)

1. **Database Configuration**: The PostgreSQL connection string is defined in `config.py`, enabling the app to connect to the database.
2. **Models and Tables**:
   - `models.py` defines two SQLAlchemy ORM models: `ProductEnquiryForms_4` and `CustomerDetails_4`. These classes map to tables in the PostgreSQL database.
   - These tables are created automatically when the application runs, ensuring the database is ready for data insertion.

1. **API Routes**:
   - The application (`app.py`) contains several routes:
     - `/post_records`: Handles the insertion of records into the two tables.
     - `/get_records`: Fetches and returns all the records from both tables.
     - `/update_record/<mobileno>`: Updates a record in the `productenquiryforms_4` table based on the mobile number.
     - `/delete_record/<mobileno>`: Deletes a record from the `productenquiryforms_4` table based on the mobile number.

1. **Integration with PostgreSQL**:
   - The app uses SQLAlchemy ORM to interact with PostgreSQL. Operations such as `INSERT`, `UPDATE`, `DELETE`, and `SELECT` are done using SQLAlchemy's query API.

1. **Flask Framework**:
   - Flask is used to expose these operations as RESTful API endpoints, allowing users to interact with the database via HTTP requests (e.g., using Postman).