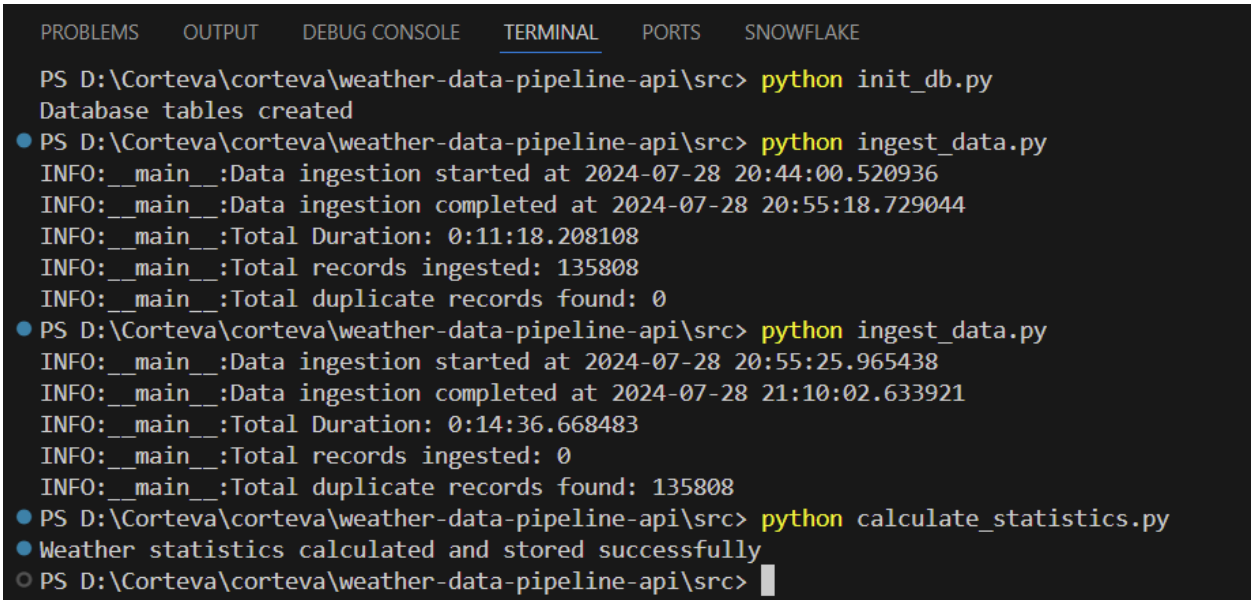
Problem 1:

Choose a database to use for this coding exercise (SQLite, Postgres, etc.). Design a data model to represent the weather data records. If you use an ORM, your answer should be in the form of that ORM's data definition format. If you use pure SQL, your answer should be in the form of DDL statements.

```
from extensions import db


class WeatherData(db.Model):
    __tablename__ = 'weather_data'
    id = db.Column(db.Integer, primary_key=True)
    station_id = db.Column(db.String(50), nullable=False)
    date = db.Column(db.Date, nullable=False)
    max_temp = db.Column(db.Integer)
    min_temp = db.Column(db.Integer)
    precipitation = db.Column(db.Integer)
```

Problem 2:



Problem 3:

##Model Definition

```
class WeatherStatistics(db.Model):
    __tablename__ = 'weather_statistics'
    id = db.Column(db.Integer, primary_key=True)
    station_id = db.Column(db.String(50), nullable=False)
    year = db.Column(db.Integer, nullable=False)
    avg_max_temp = db.Column(db.Float)
    avg_min_temp = db.Column(db.Float)
    total_precipitation = db.Column(db.Float)
```

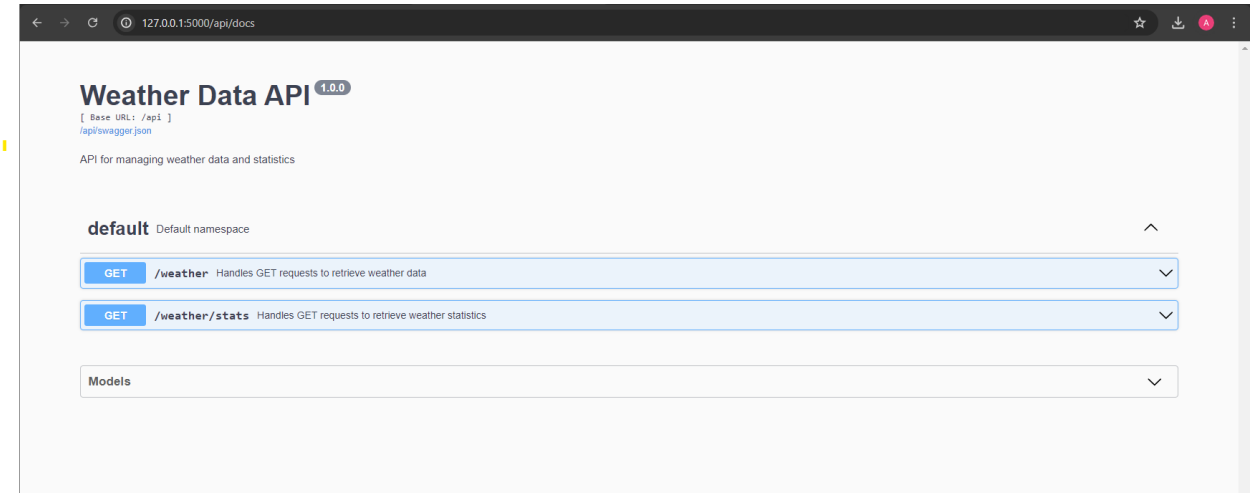
##Code to calculate statistics

```
from datetime import datetime
from sqlalchemy import func
from app import create_app
from extensions import db
from models import WeatherData, WeatherStatistics

def calculate_statistics():
    app = create_app()
    with app.app_context():
        stations = db.session.query(WeatherData.station_id).distinct()
        for station in stations:
            station_id = station[0]
            years = db.session.query(func.extract('year',
WeatherData.date).distinct()).filter_by(station_id=station_id).all()
            for year in years:
                year = int(year[0])
                avg_max_temp = db.session.query(func.avg(WeatherData.max_temp)).filter(
                    WeatherData.station_id == station_id,
```

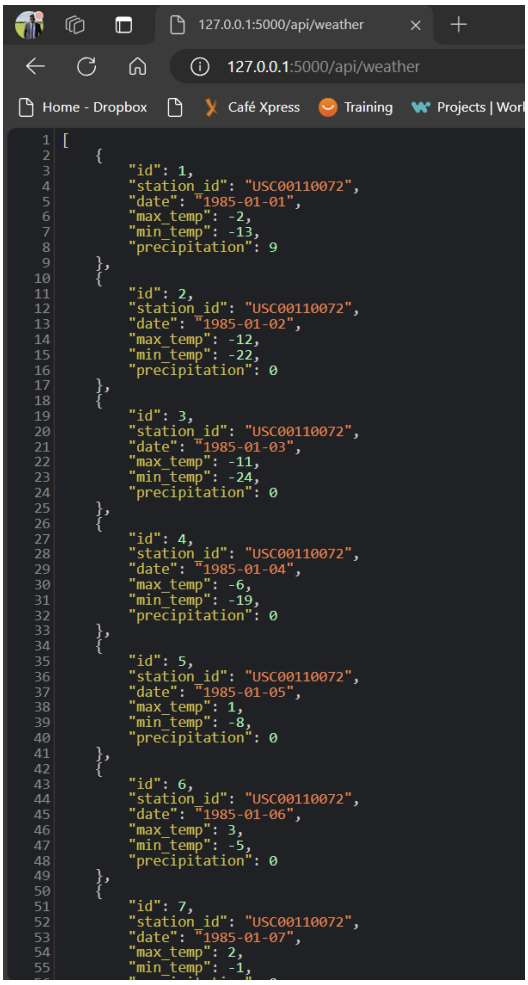
```
                    func.extract('year', WeatherData.date) == year,
                    WeatherData.max_temp != None
                ).scalar()
                avg_min_temp = db.session.query(func.avg(WeatherData.min_temp)).filter(
                    WeatherData.station_id == station_id,
                    func.extract('year', WeatherData.date) == year,
                    WeatherData.min_temp != None
                ).scalar()
                total_precipitation =
db.session.query(func.sum(WeatherData.precipitation)).filter(
                    WeatherData.station_id == station_id,
                    func.extract('year', WeatherData.date) == year,
                    WeatherData.precipitation != None
                ).scalar()

                weather_stats = WeatherStatistics(
                    station_id=station_id,
                    year=year,
                    avg_max_temp=avg_max_temp,
                    avg_min_temp=avg_min_temp,
                    total_precipitation=total_precipitation
                )
                db.session.add(weather_stats)
        db.session.commit()
        print("Weather statistics calculated and stored successfully")

if __name__ == '__main__':
    calculate_statistics()
```

Problem 4:

Swagger API Documentation



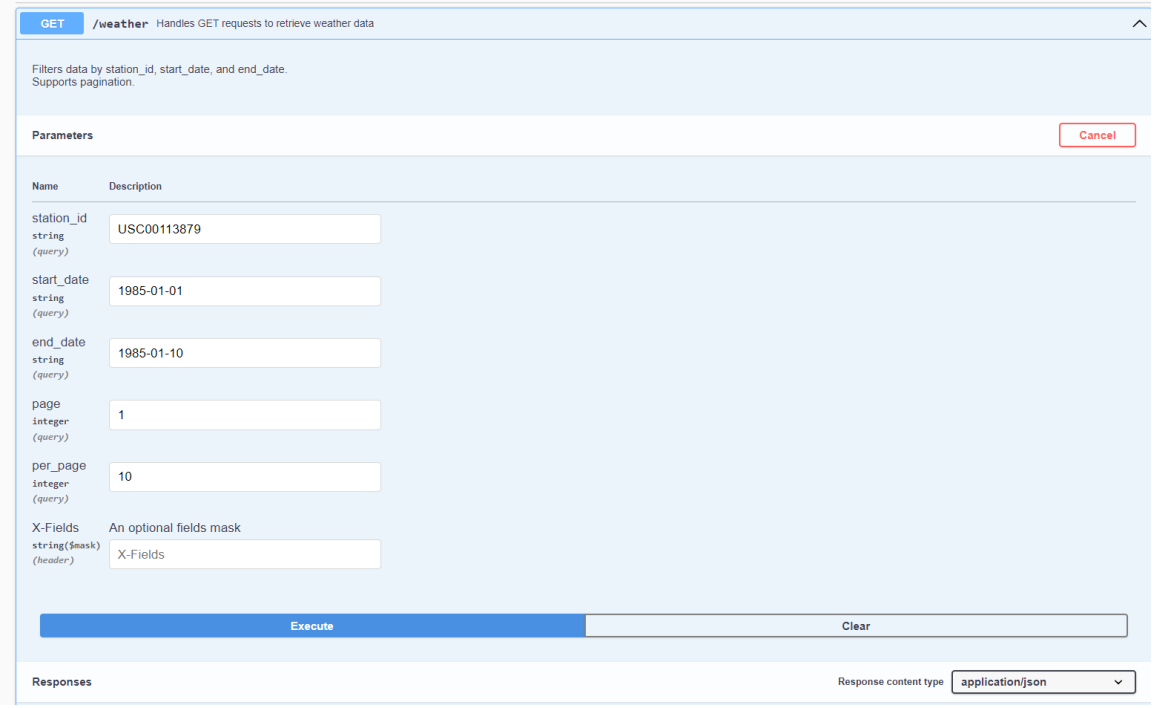
127.0.0.1:5000/api/weather

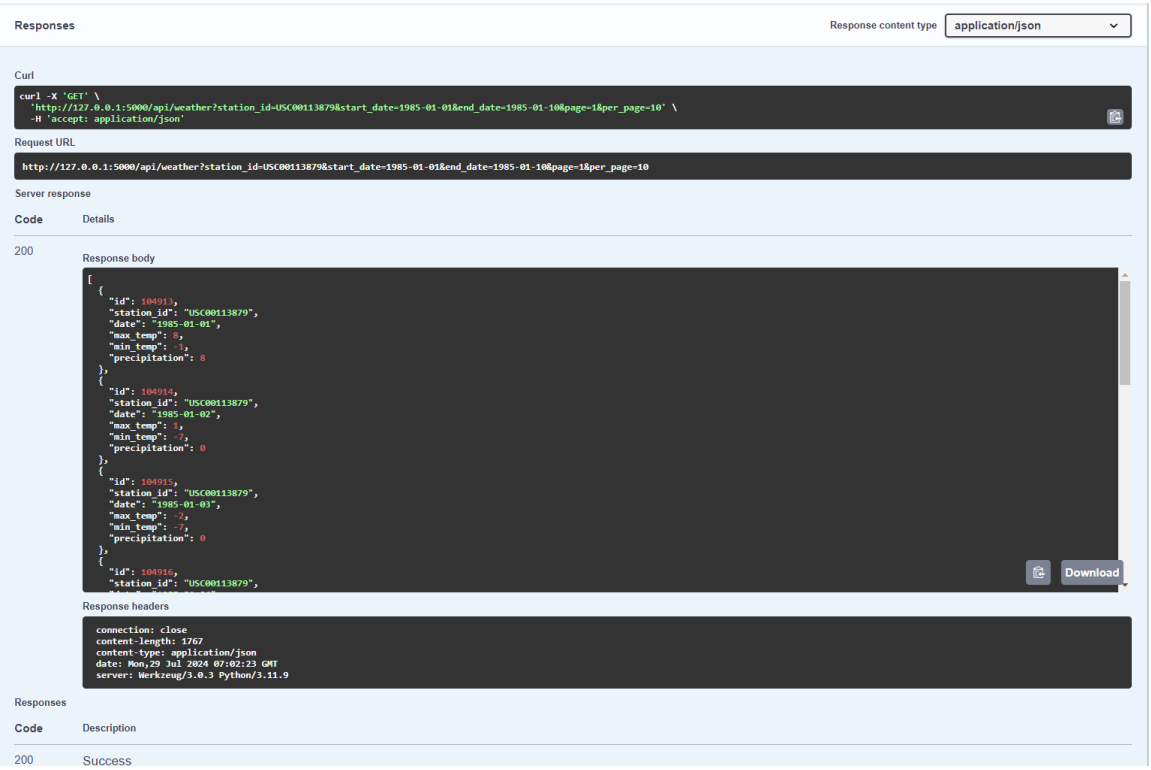


127.0.0.1:5000/api/weather/stats

GET endpoint /api/weather:

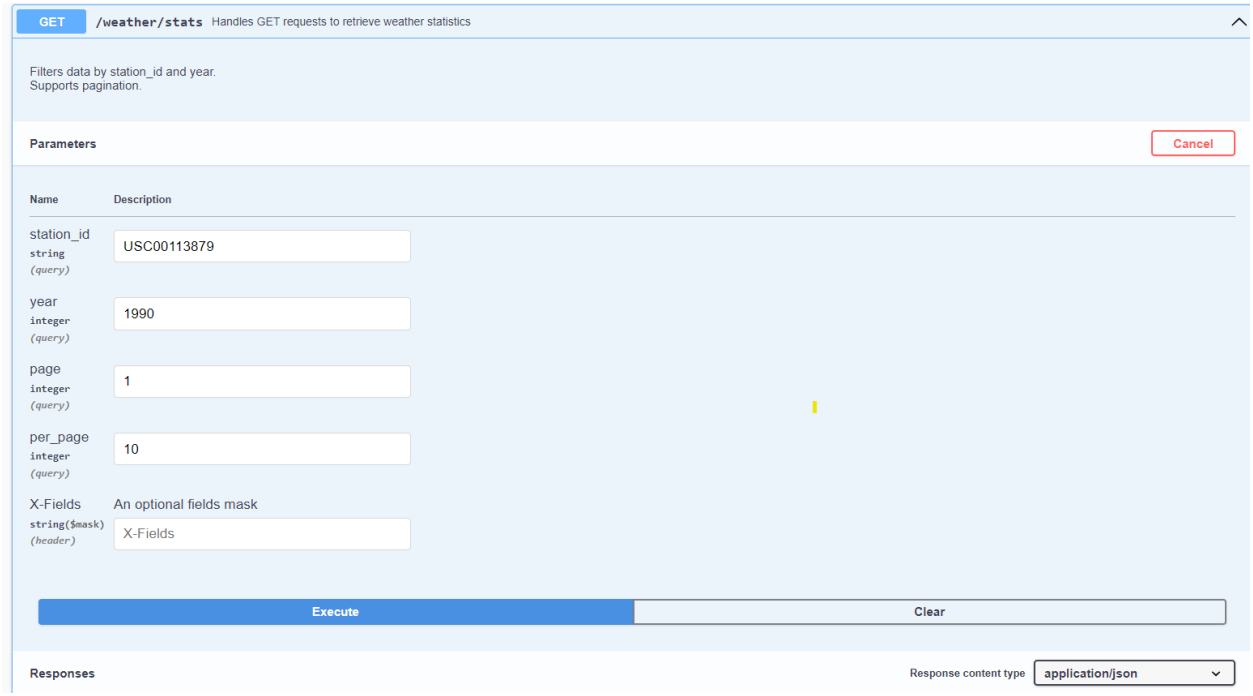
Clients can filter by date and station ID as shown below.

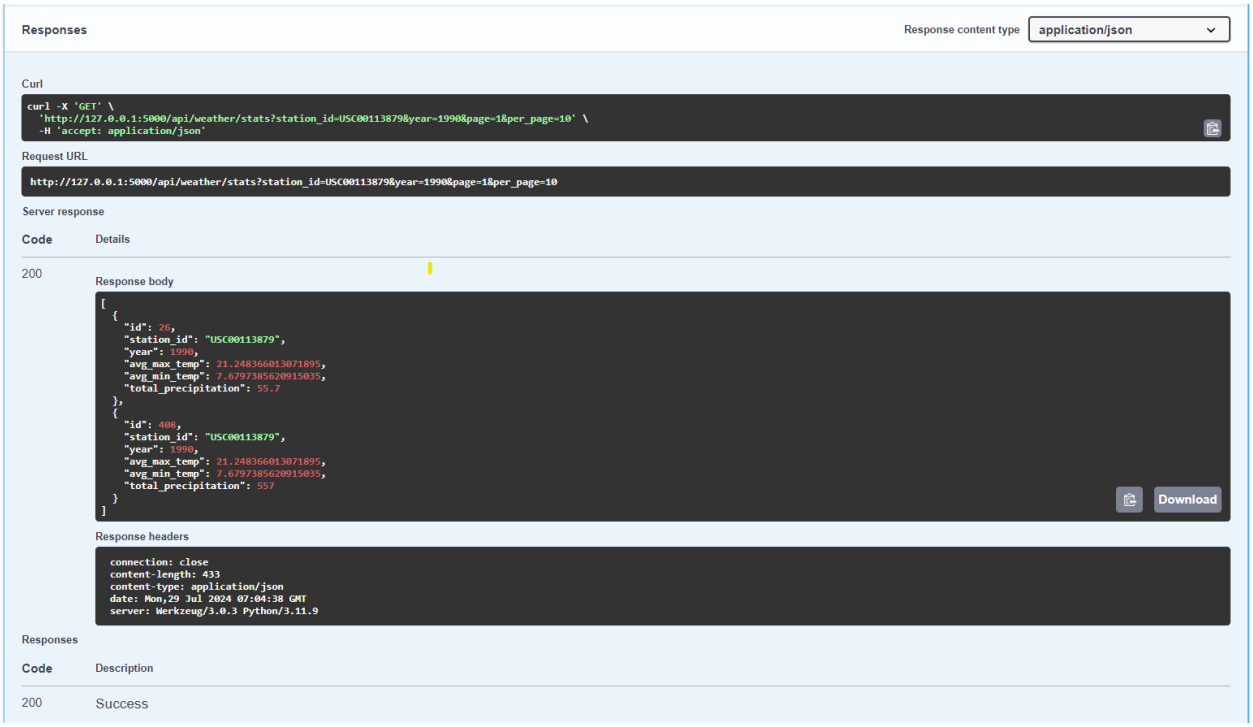




GET endpoint /api/weather/stats:

Clients can filter by year and station ID as shown below.





Extra Credit - Deployment:

To deploy the API, database, and scheduled data ingestion code in the cloud using AWS, we can use the following tools and services:

1. **API Deployment:** We can use AWS Elastic Beanstalk to deploy and manage the Flask API. It handles the infrastructure and scaling automatically.
2. **Database:** For DB, we can use Amazon RDS to set up a managed PostgreSQL database. It handles backups, updates, and scaling.
3. **Scheduled Data Ingestion:** AWS Lambda can be used to run the data ingestion code and we can schedule the Lambda function with Amazon CloudWatch Events to run at specific intervals.

This approach leverages AWS services to manage infrastructure, scalability, and scheduling, allowing you to focus on your application logic.

GitHub Repository:

https://github.com/Ashray3096/weather-data-pipeline-api/tree/main