Answers

Extra Credit - Deployment

To deploy the API, database, and a scheduled version of the data ingestion code on AWS, the following AWS services can be used:

1. **Database:** Amazon RDS (Relational Database Service) can be used to create a managed database instance like PostgreSQL or MySQL.
2. **Data Ingestion:** AWS Lambda can be used to create a serverless function for the data ingestion script, and Lambda function can be scheduled to run periodically using AWS CloudWatch Events.
3. **API:** The Django application can be deployed using AWS Elastic Beanstalk, which automatically handles the deployment, scaling, and monitoring of application.
4. **Storage:** Raw weather data files can be stored in an Amazon S3 bucket.
5. **Infrastructure as Code:** AWS CloudFormation or Terraform can be used to define and provision the AWS infrastructure as code.
6. **CI/CD:** AWS CodePipeline can be used to set up a continuous integration and continuous deployment pipeline for the project.

This approach ensures that the deployment of the project will be scalable, reliable, and maintainable in the cloud.

Problem 1:

For the weather data coding assessment, I used SQLite as the database as it is an ideal choice for small to medium-sized applications, prototyping, and testing, and easy to set up and use.

I designed a data model to represent weather data records using SQLAlchemy, which is an Object-Relational Mapping (ORM) tool in Python. The weather records are stored in the following table:

weather_records Table:

The WeatherRecord table is designed to store daily weather data for different stations. Each record in this table represents weather data for a specific station on a specific date. The fields include:

id: An auto-incrementing primary key.

station_id: A string representing the unique identifier of the weather station, cannot be null.

date: A date field representing the date of the weather record, cannot be null.

max_temp: A float representing the maximum temperature recorded on that date, in degrees Celcius.

min_temp: A float representing the minimum temperature recorded on that date, in degrees Celcius.

precipitation: A float representing the precipitation recorded on that date, in centimeters.

A unique constraint on (station_id, date) to ensure there will be no duplicate entries for the same station_id and date.

src/create_session.py contains the script to establish a session and create and connect to the database. src/data_models.py will create tables in the database.

Problem 2:

src/data_ingestion.py contains code to ingest weather data from the wx_data directory containing raw text files into the weather_data database. Missing values represented by -9999 are changed to null, the units for maximum temperature, minimum temperature and amount of precipitation are corrected and then data is inserted using bulk insert to commit multiple records. To avoid duplicates when code is run multiple times, an IntegrityError is checked against the unique constraint (station_id, date). If this constraint is violated, the record is skipped and not inserted. Logging has been implemented to record start times, end times, number of records ingested for every file, along with number of duplicates detected, if any. The logs are written to src/data_ingestion.log, as shown below.

The weather_records table is shown below:

Table: weather_records

Filter in any column

	id	station_id	date	max_temp	min_temp	precipitation
	Filter	Filter	Filter	Filter	Filter	Filter
385344	385344	USC00120177	2010-09-07	24.4	10.6	0.0
385345	385345	USC00120177	2010-09-08	23.9	7.8	0.0
385346	385346	USC00120177	2010-09-09	27.2	11.7	0.0
385347	385347	USC00120177	2010-09-10	NULL	NULL	0.0
385348	385348	USC00120177	2010-09-11	22.2	15.6	10.7
385349	385349	USC00120177	2010-09-12	26.7	11.1	0.0
385350	385350	USC00120177	2010-09-13	NULL	NULL	0.0
385351	385351	USC00120177	2010-09-14	26.1	12.8	0.0
385352	385352	USC00120177	2010-09-15	30.0	11.7	0.0
385353	385353	USC00120177	2010-09-16	26.7	15.6	0.0
385354	385354	USC00120177	2010-09-17	23.9	11.1	0.0
385355	385355	USC00120177	2010-09-18	28.9	19.4	0.0
385356	385356	USC00120177	2010-09-19	27.2	16.1	0.0
385357	385357	USC00120177	2010-09-20	28.9	16.1	0.0
385358	385358	USC00120177	2010-09-21	33.3	19.4	0.0
385359	385359	USC00120177	2010-09-22	22.8	17.2	8.9
385360	385360	USC00120177	2010-09-23	33.3	17.2	0.0
385361	385361	USC00120177	2010-09-24	29.4	17.2	0.0
385362	385362	USC00120177	2010-09-25	20.6	11.1	0.0

Problem 3:

To calculate weather data statistics, I defined another table in the src/data_models.py file, which is detailed below:

weather_stats Table:

The weather_stats table is designed to store aggregated weather statistics, for different weather stations annually. Each record in this table represents statistical data for a specific station in a specific year. The fields include:

id: An auto-incrementing primary key.

station_id: A string representing the unique identifier of the weather station, cannot be null.

year: An integer representing the year of the statistics, cannot be null.

avg_max_temp: A float representing the average maximum temperature for that year (in degrees Celsius).

avg_min_temp: A float representing the average minimum temperature for that year (in degrees Celsius).

total_precipitation: A float representing the total precipitation for that year (in centimeters).

Statistics that could not be calulated, that had missing data, are indicated by NULL in this table. The script to calculate these statistics and store them in the database can be found in src/statistical_analysis_weather_data.py file. The table weather_stats is shown below:

Table: weather_stats

Filter in any column

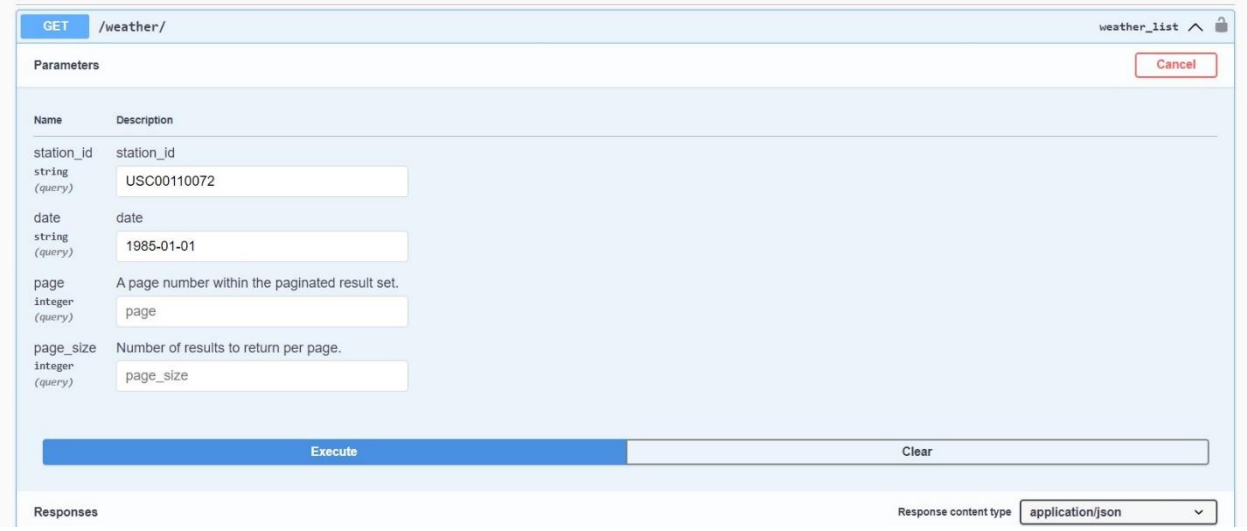
	id	station_id	year	avg_max_temp	avg_min_temp	total_precipitation
	Filter	Filter	Filter	Filter	Filter	Filter
225	225	USC00112348	2003	15.153698630137	3.30684931506849	716.6
226	226	USC00112348	2004	15.1038356164384	3.88598901098901	878.6
227	227	USC00112348	2005	17.3889221556886	5.62814371257485	753.5
228	228	USC00112348	2006	15.5841095890411	4.83424657534247	1100.1
229	229	USC00112348	2007	15.3591780821918	4.11095890410959	1006.3
230	230	USC00112348	2008	13.9151260504202	2.43352112676056	1210.7
231	231	USC00112348	2009	15.1867867867868	3.95855855855856	1186.6
232	232	USC00112348	2011	14.5049180327869	2.24426229508197	0.0
233	233	USC00112348	2012	18.9463855421687	6.49337349397589	*NULL*
234	234	USC00112348	2014	17.1645502645503	7.16560846560847	651.4
235	235	USC00112483	1985	18.5432876712329	6.97068493150685	1320.9
236	236	USC00112483	1986	19.8298630136986	8.30246575342466	1032.6
237	237	USC00112483	1987	20.3852054794521	8.14356164383561	1047.5
238	238	USC00112483	1988	19.9901639344262	6.77185792349726	1042.9
239	239	USC00112483	1989	18.5549723756906	6.89337016574586	976.6
240	240	USC00112483	1990	20.0772602739726	8.61369863013699	1487.0
241	241	USC00112483	1991	19.8980821917808	9.08219178082192	1031.2

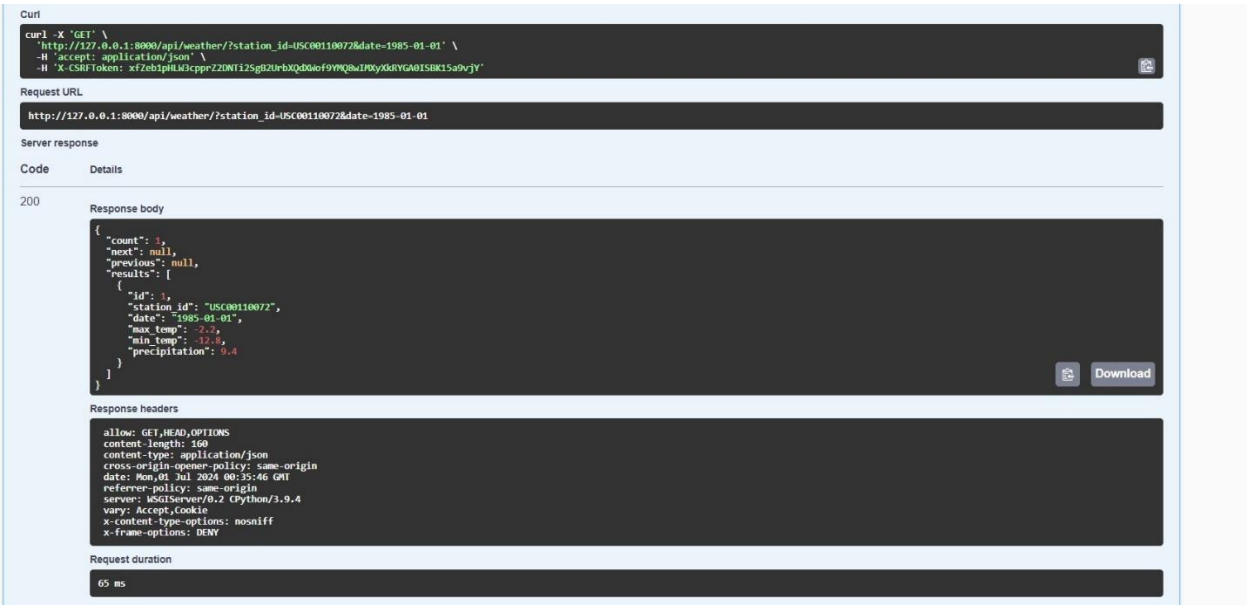
Problem 4:

I chose Django REST Framework for the API Development. A REST API with the /api/weather and /api/weather/stats GET Endpoints returns JSON-formatted responses of the ingested and calculated statistical data from the database, which is paginated (page number and number of

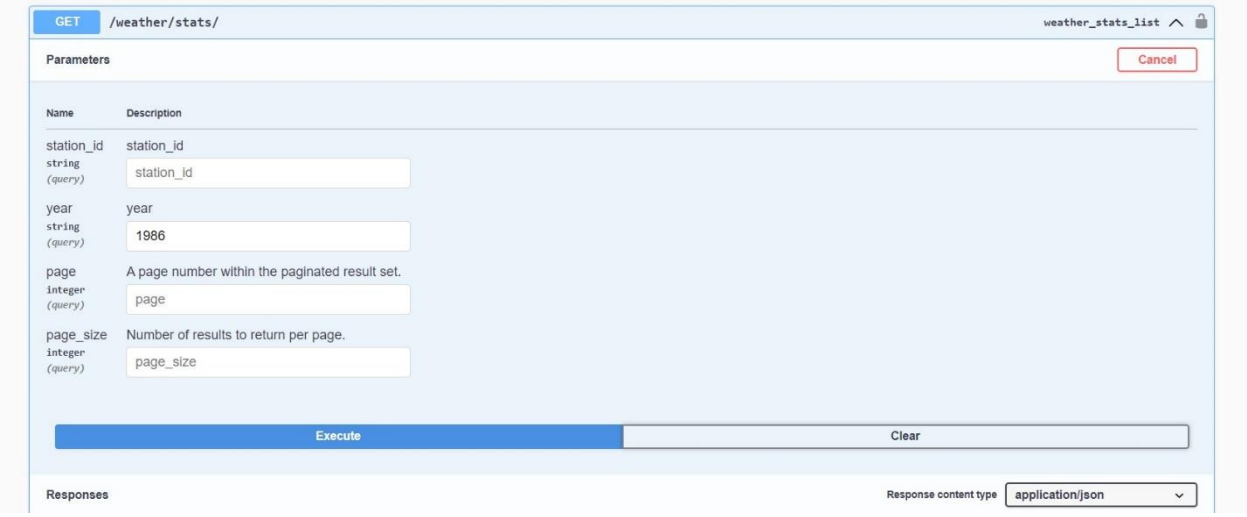
results per page can be chosen) and can be filtered by station_id and date for the /api/weather GET Endpoint and by station_id and year for the /api/weather/stats GET Endpoint. Swagger documentation for the API can be accessed from the /swagger endpoint. All files to setup and run this API locally, along with unit tests can be found in the weather_api directory.

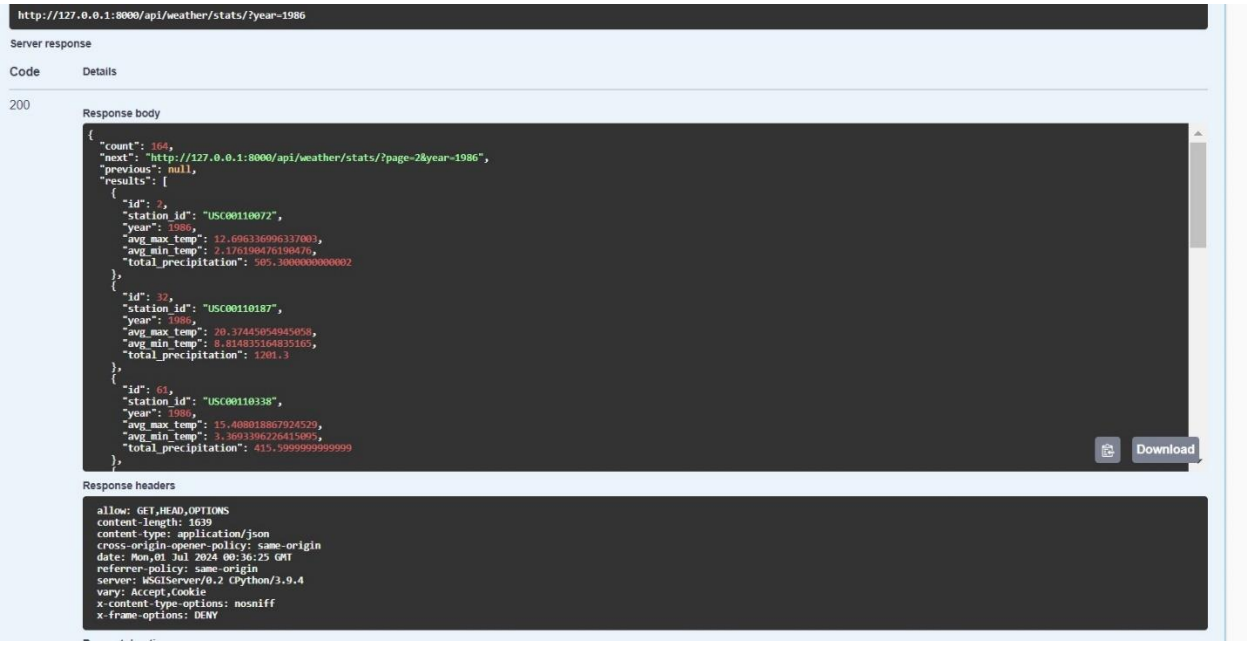
/api/weather GET Endpoint:





/api/weather/stats GET Endpoint:


GET
/weather/stats/
weather_stats_list
Parameters
Cancel
Name
Description
station_id
string
(query)
station_id
station_id
year
string
(query)
year
1986
page
integer
(query)
A page number within the paginated result set.
page
page_size
integer
(query)
Number of results to return per page.
page_size
Execute
Clear
Responses
Response content type
application/json


http://127.0.0.1:8000/api/weather/stats/?year=1986
Server response
Code
Details
200
Response body
{
"count": 164,
"next": "http://127.0.0.1:8000/api/weather/stats/?page=2&year=1986",
"previous": null,
"results": [
{
"id": 2,
"station_id": "USC00110072",
"year": 1986,
"avg_max_temp": 12.696336096337003,
"avg_min_temp": 2.176190476190476,
"total_precipitation": 505.3000000000002
},
{
"id": 32,
"station_id": "USC00110187",
"year": 1986,
"avg_max_temp": 20.37445054945058,
"avg_min_temp": 8.814835164835165,
"total_precipitation": 1201.3
},
{
"id": 61,
"station_id": "USC00110338",
"year": 1986,
"avg_max_temp": 15.408018867924529,
"avg_min_temp": 3.369339622641509,
"total_precipitation": 415.5999999999999
},
Download
Response headers
allow: GET,HEAD,OPTIONS
content-length: 1639
content-type: application/json
cross-origin-opener-policy: same-origin
date: Mon,01 Jul 2024 00:36:25 GMT
referrer-policy: same-origin
server: WSGIServer/0.2 CPython/3.9.4
vary: Accept,Cookie
x-content-type-options: nosniff
x-frame-options: DENY