Implementing Nowcasting Intelligent System for Media

[Team Members](#_2et92p0)

[Part 1: Designing the REST API with Logic Enhancement](#_tyjcwt)

[Operations supported by API](#_1t3h5sf)

[GeoPy distance logic and Nearest Location](#_4d34og8)

[Positive Test Cases](#_2s8eyo1)

[Negative Test Cases](#_26in1rg)

[Part 2: New Client (Streamlit)](#_35nkun2)

[Part 3: Host the API](#_1ksv4uv)

[Deployment on Cloud Run](#_nw0zf9470ah4)

[1. Creating a DockerFile](#_gl1ke927hs22)

[2. Build a Container Image](#_2edsrut1vo0q)

[3. Deploy the Container Image](#_6bsn1547ui4x)

[Part 4 : Securing the API using JWT Tokens](#_vhajovqscjih)

[JWT Authentication](#_7ufqh53d0tj)

[JWT Handler](#_6y5s84yt30lk)

[JWT Secret and Algorithm](#_z41x55qzjk7x)

[User Registration and Login](#_3m2xlgrb3ogi)

[Securing Routes](#_h9gzlljex4mp)

[JWT Bearer](#_7v5zoileupyp)

[Dependency Injection](#_kpe1j35uml8u)

[Part 5 : Airflow Pipeline](#_ljserqgf6nmu)

[Conclusion](#_2jxsxqh)

[References](#_z337ya)

# 

Implementing Nowcasting Intelligent System for Media

# 

# Team Members

| NAME | NUID |
| --- | --- |
| PRIYANKA DILIP SHINDE | 001524484 |
| KSHITIJ ZUTSHI | 001021288 |
| DWITHIKA SHETTY | 002114630 |

# Part 1: Designing the REST API with Logic Enhancement

# 

## Operations supported by API

1. Endpoint 1: **/nowcast/{img\_Id}**

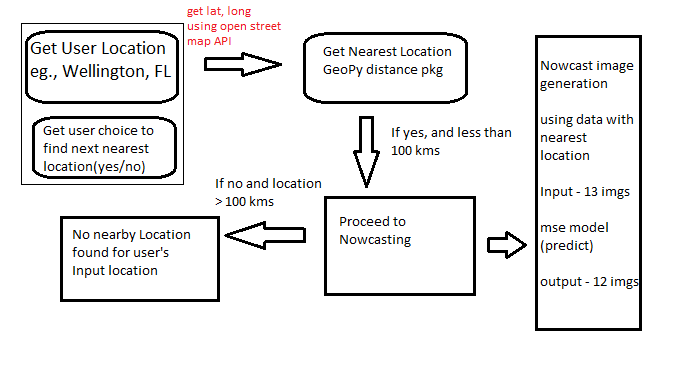
To render next hour’s nowcasting 12 output images based on their image id that ranges from 0-11 where each

image id represents 5 min interval of nowcasted image.

1. Endpoint 2: **/weatherviz/{location, nearest\_location}**

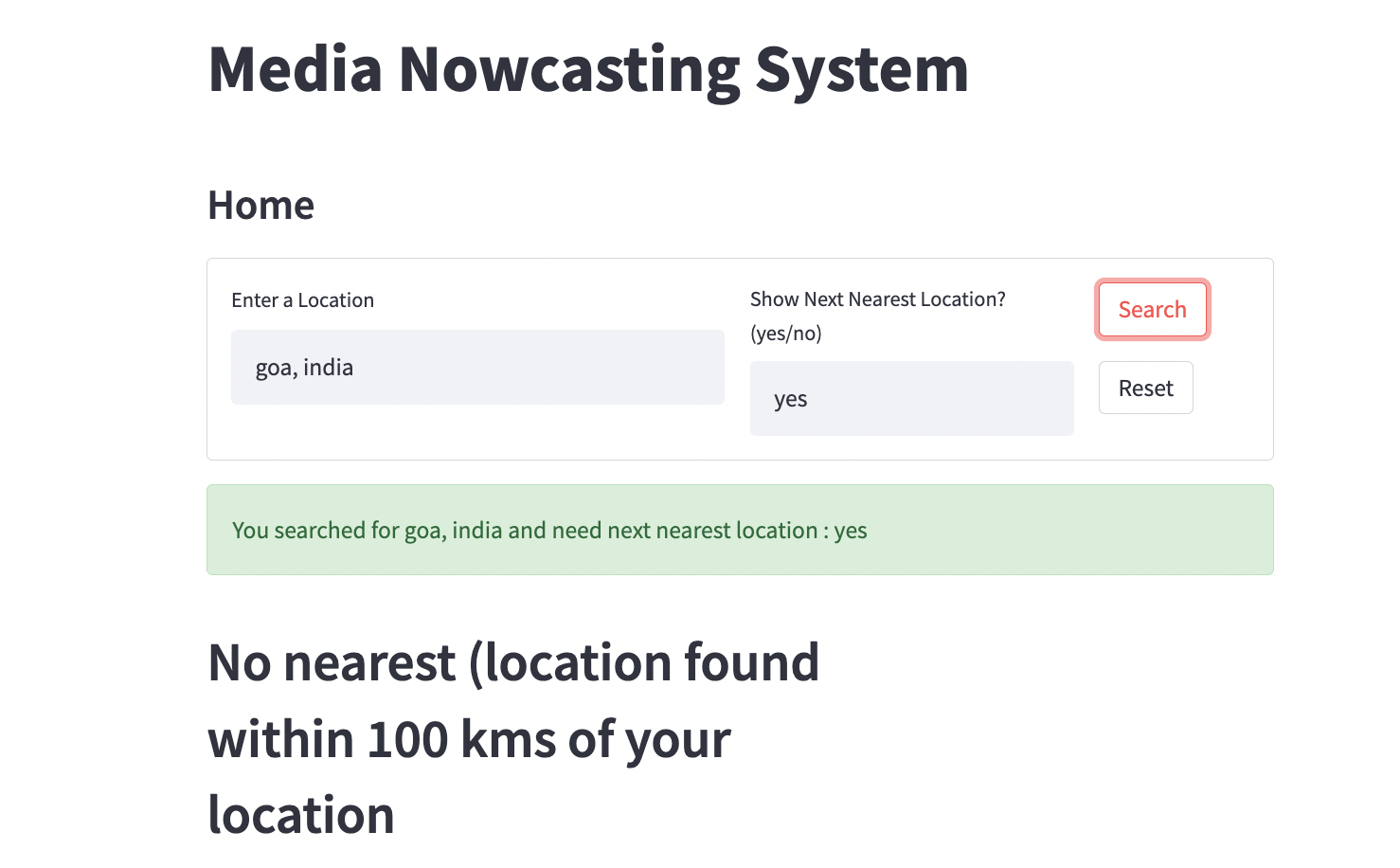
## GeoPy distance logic and Nearest Location

To display the nowcast images gif based on the location and if the location does not exist displaying the nowcast images gif for nearest location based on user request.

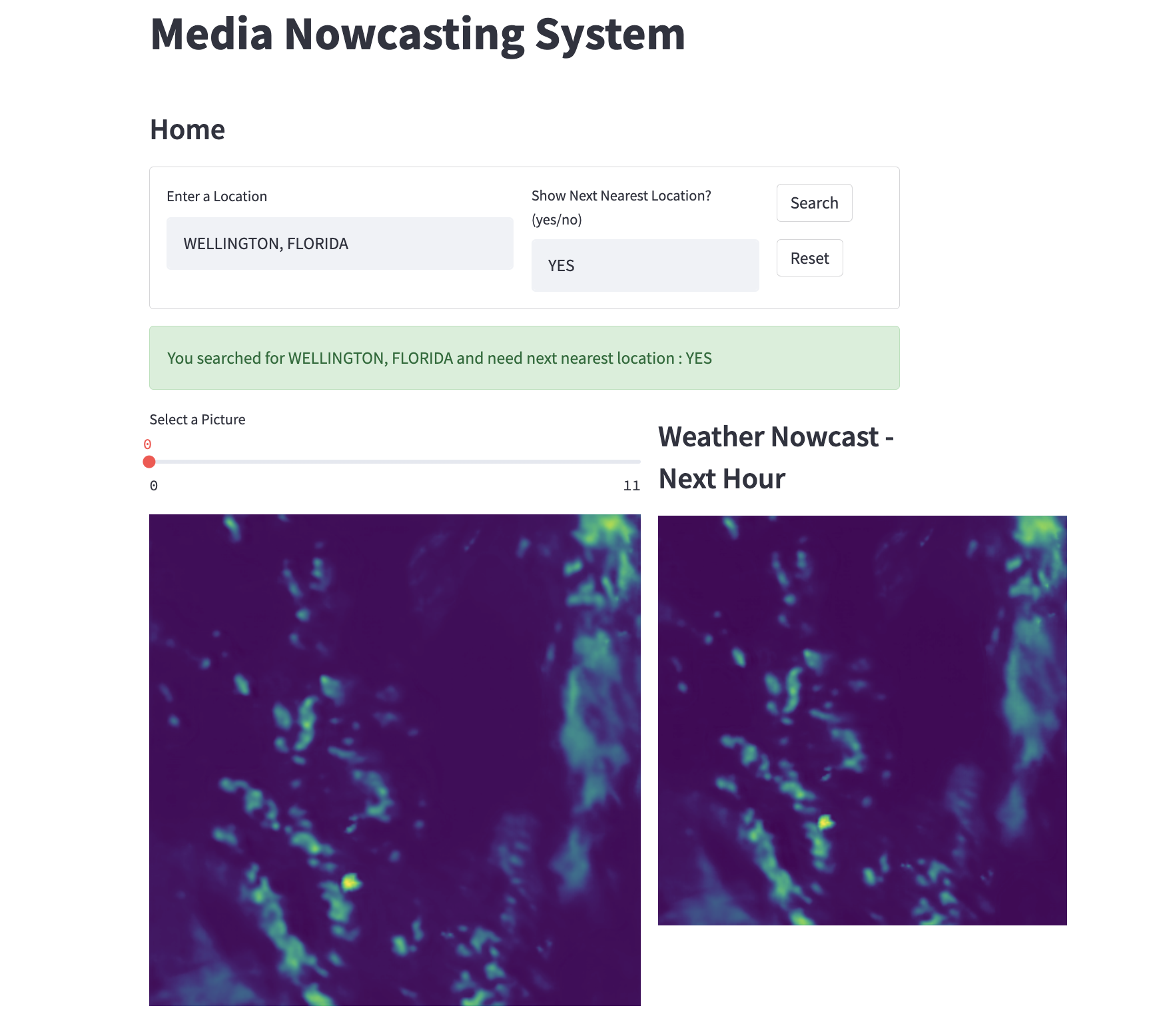


## Positive Test Cases

### If the location is beyond 100 km it will display no nearest location found.

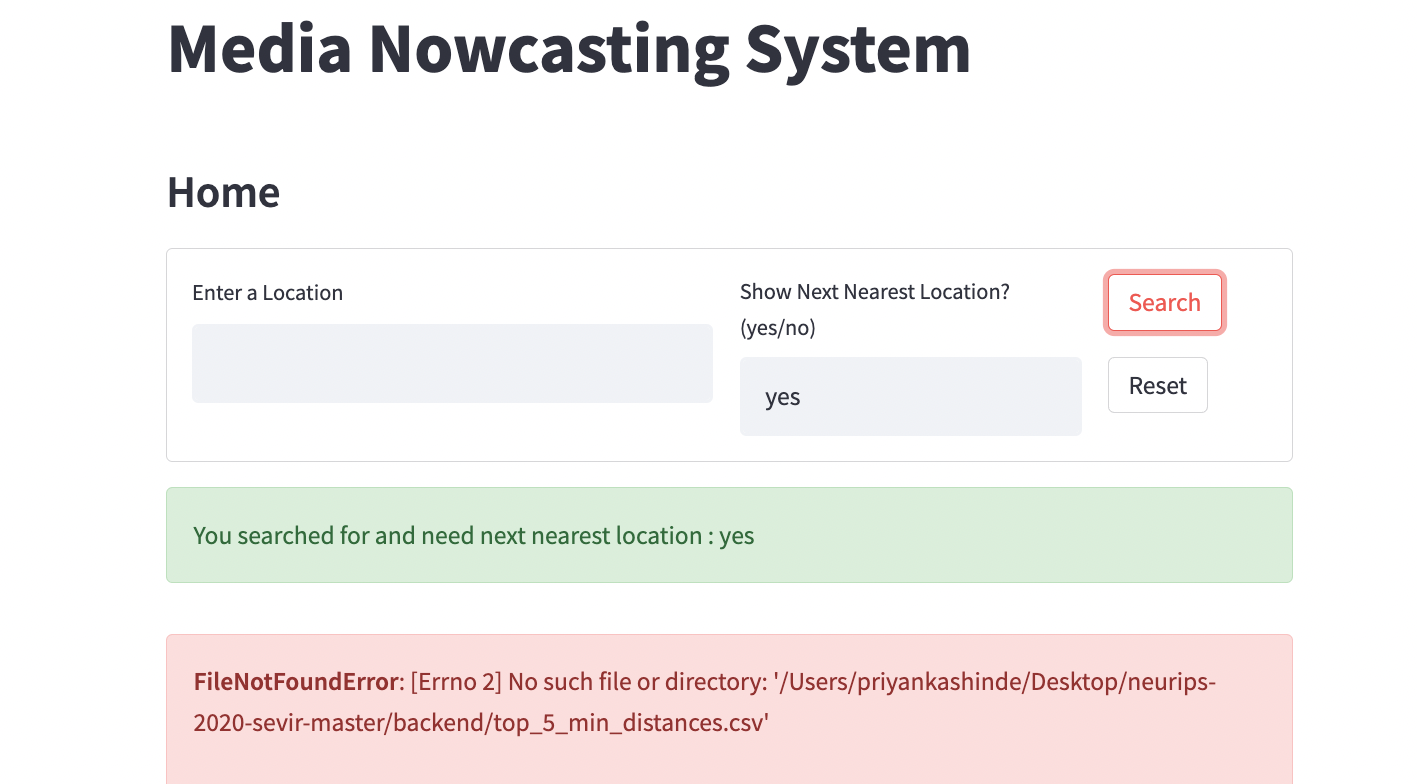


### Giving inputs in caps still gives results.



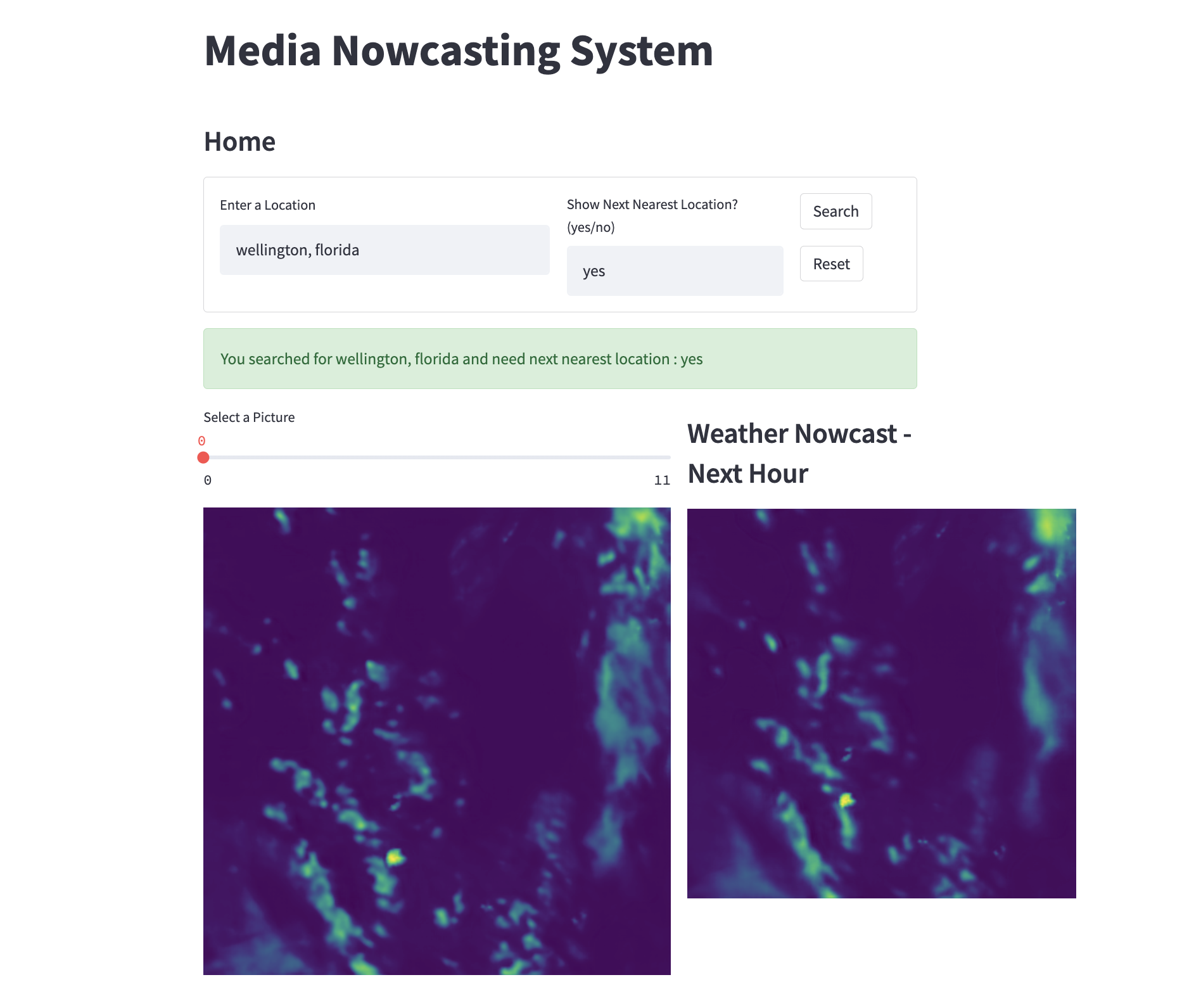
## Negative Test Cases

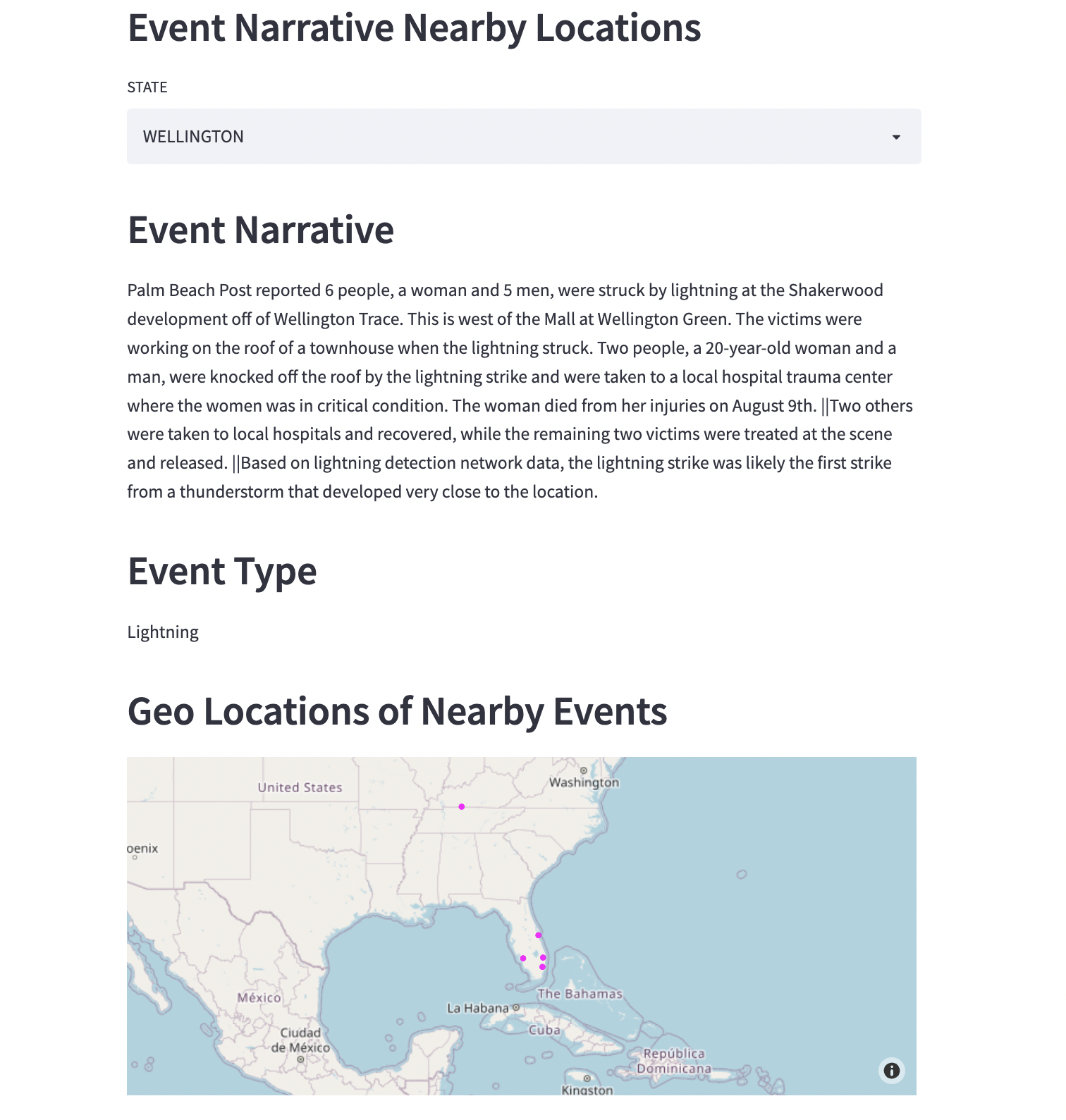
### Giving null values in input field throws error.



# Part 2: New Client (Streamlit)

Integrating Fast API with Streamlit where users can generate nowcast, filter event narrative, and get geographic representation of storm events happening based on location provided and if the location does not exist user can choose to view the nowcast of nearest location.





# Part 3: Host the API

Deployment of FAST API on GCP Cloud Run

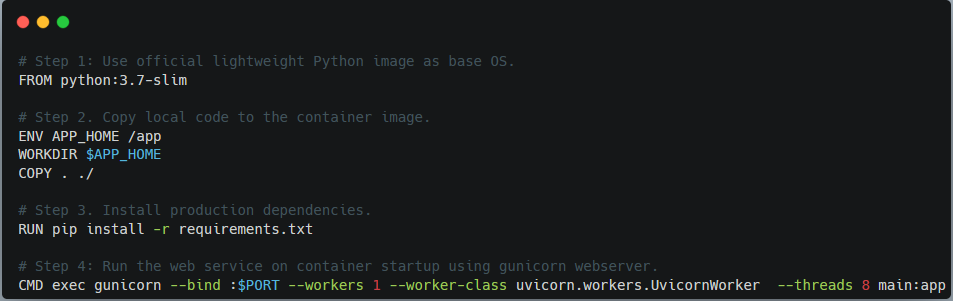
## Deployment on Cloud Run

Google Cloud Run is a fully-managed compute platform for deploying and scaling stateless containerized applications quickly and securely.

To deploy our FAST API Nowcasting application to Cloud Run, we need to containerize the application using Docker. This process would give us a container image that can be deployed to Cloud Run.

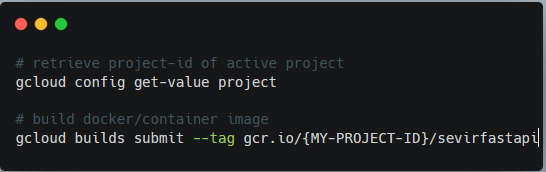
### 1. Creating a DockerFile

The first step to containerizing our application is to create a Dockerfile in our source code directory.



### 2. Build a Container Image

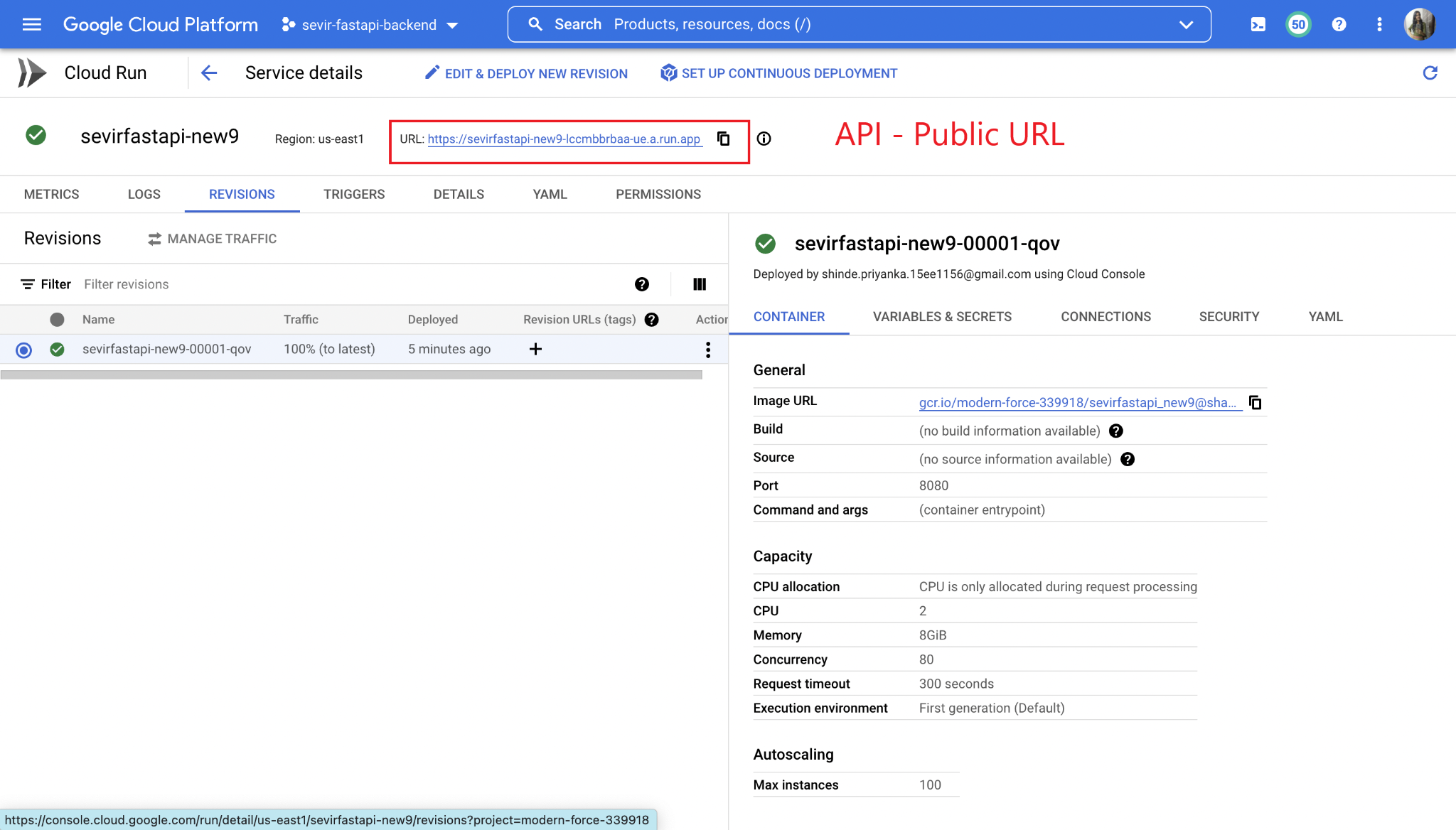
A container (docker) image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries, and settings.



### 3. Deploy the Container Image

The final step is to deploy our serverless app on Cloud Run. To achieve this, we need to create a Cloud Run Service. The service would make use of our image on the Container Registry to deploy the application for us.

When the deployment is successful, Cloud Run would automatically generate a web URL that can be used to access the application. If needed, we can also map a custom domain with the Cloud Run service.



**NOTE : Issues faced with reading some H5 files, else data is getting pushed in GCS Bucket**

# Part 4 : Securing the API using JWT Tokens

#### **JWT Authentication**

In this section, we'll create a JWT token handler and a class to handle bearer tokens.

Before beginning, install PyJWT, for encoding and decoding JWTs. We'll also be using python-decouple for reading environment variables.

#### **JWT Handler**

The JWT handler will be responsible for signing, encoding, decoding, and returning JWT tokens. In the code, we imported the time, typing, jwt, and decouple modules. The time module is responsible for setting an expiry for the tokens. Every JWT has an expiry date and/or time where it becomes invalid. The jwt module is responsible for encoding and decoding generated token strings. Lastly, the token\_response function is a helper function for returning generated tokens.

#### **JWT Secret and Algorithm**

Next, create an environment file called .env in the base directory:

secret=please\_please\_update\_me\_please

algorithm=HS256

The secret key is used for encoding and decoding JWT strings.

The algorithm value on the other hand is the type of algorithm used in the encoding process.

In the **signJWT** function, we defined the payload, a dictionary containing the user\_id passed into the function, and an expiry time of ten minutes from when it is generated. Next, we created a token string consisting of the payload, the secret, and the algorithm type and then returned it.

The **decodeJWT** function takes the token and decodes it with the aid of the jwt module and then stores it in a decoded\_token variable. Next, we return decoded\_token if the expiry time is valid, otherwise, we return None.

#### **User Registration and Login**

Moving along, let's wire up the routes, schemas, and helpers for handling user registration and login.

#### **Securing Routes**

With the authentication in place, let's secure the create route.

#### **JWT Bearer**

Now we need to verify the protected route, by checking whether the request is authorized or not. This is done by scanning the request for the JWT in the Authorization header. FastAPI provides the basic validation via the HTTPBearer class. We can use this class to extract and parse the token. Then, we'll verify it using the decodeJWT function defined in app/auth/auth\_handler.py.

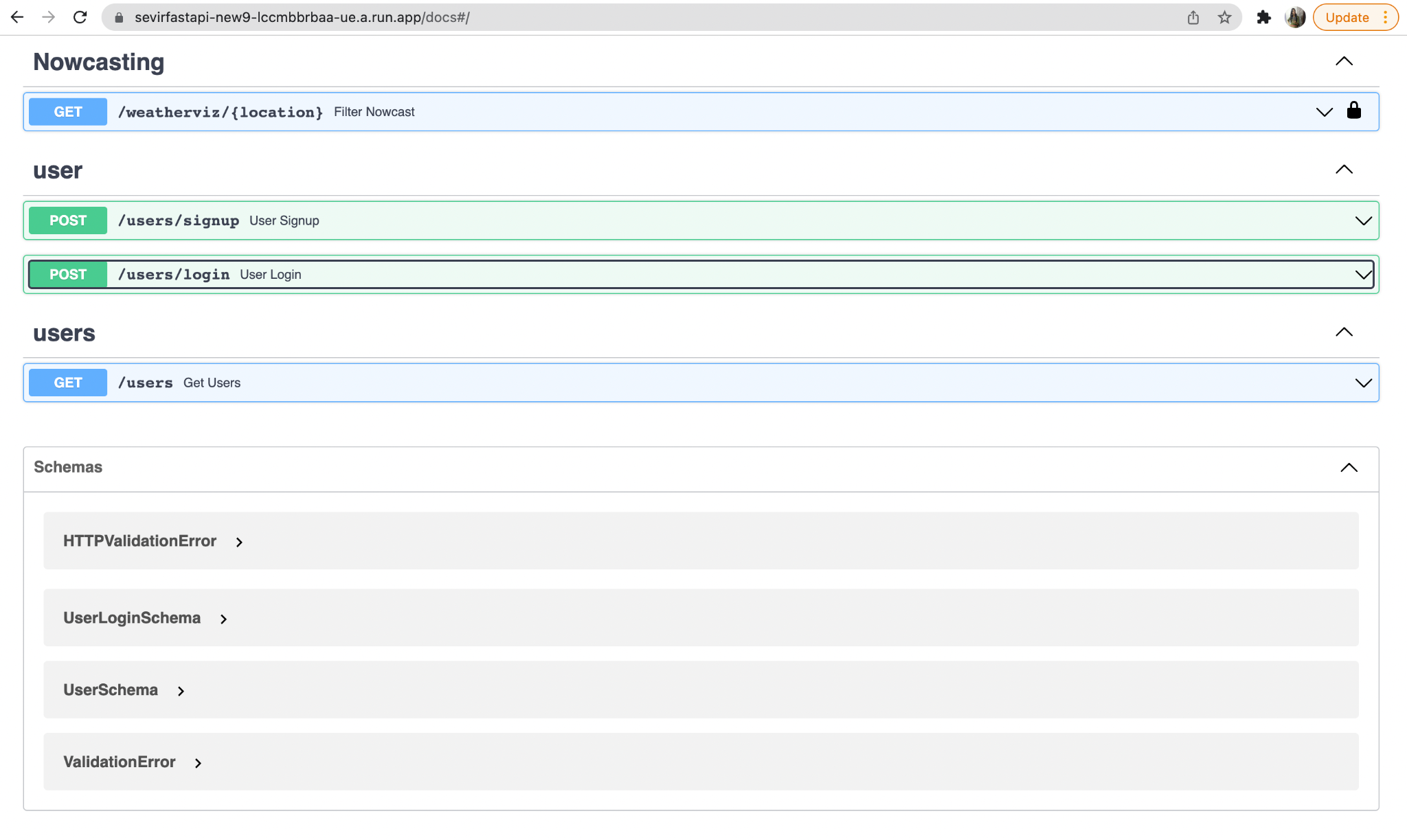
Create a new file in the "auth" folder called auth\_bearer.py

#### **Dependency Injection**

To secure the routes, we'll leverage dependency injection via FastAPI's Depends.

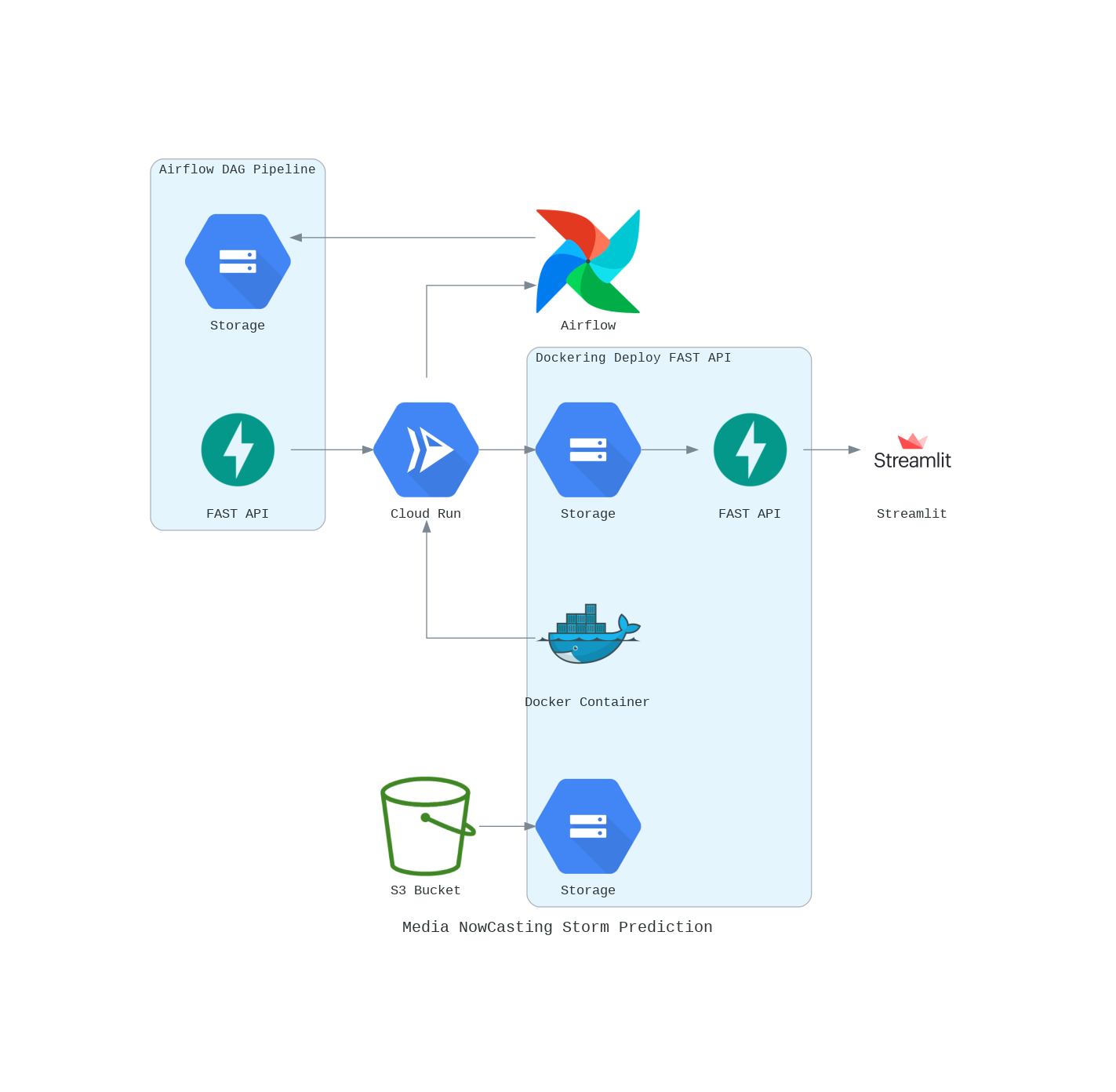
Start by updating the imports by adding the JWTBearer class as well as Depends

In the POST route, add the dependencies argument to the @app property



# Part 5 : Airflow Pipeline

We create a list of 50 locations spanning across the US for which SEVIR is designed. For these locations, the Airflow pipeline will create the nowcast every hour and store it in a cloud location (Google cloud storage). The airflow dashboard will show update <timestamp>



# 

# Conclusion

1. Did logic enhancement by allowing the user to generate a nowcast based on choice of location and if the location does not exist asked the user if they want the nearest location nowcast.
2. Integrated Fast API with Streamlit.

# References

1. <https://realpython.com/fastapi-python-web-apis/#path-parameters-get-an-item-by-id>
2. <https://learning.postman.com/docs/getting-started/introduction/>
3. <https://www.youtube.com/watch?v=vpTAqnAbowo>
4. <https://github.com/gretelai/gretel-blueprints>
5. <https://nbviewer.org/github/MIT-AI-Accelerator/eie-sevir/blob/master/examples/SEVIR_Tutorial.ipynb>
6. <https://gretel.ai/blog/walkthrough-create-synthetic-data-from-a-dataframe-or-csv>