

# Final Report

## Project title

Weather History

- Build a weather data warehouse with MongoDB

## Team member

Li-Chi Chang ([chanl01@pfw.edu](mailto:chanl01@pfw.edu))

## Detailed description of your project

This Weather History can be divided into three parts. Each part has its unique functionality. Load Data Timer is for data collecting. Json to Database is for importing data to MongoDB. Weather Report Web Server is for data presentation website.

- Load Data Timer
  - Use **Google Apps Script** and **Google Drive** to build a crawler with timer.
    - Apps script is a software-embedded programming language based on JavaScript.
    - I embedded a script function in a timer on Google Drive. While the timer is triggered, the function crawls data through web APIs and saves this data into a Google Drive folder.
    - This method can build a simple crawler server.
  - Crawling the yesterday weather data of **several cities every 12 hours**.
    - OpenWeatherMap supports historical data API. Get requests with my key can return a 24-hour hourly weather data in json format.
    - In a loop, crawls all cities I want.
    - The cities list:
      - Fort Wayne
      - Taipei
- Json to Database
  - Translate and save json data into **MongoDB** using **Python** package pymongo.
  - To avoid the duplication, use UTC timestamp as document ID. And before inserting, check the ID is exist or not. If the ID exists in the collection, just simply skip the value.
  - The hourly data contains:
    - UTC time
    - Temperature
    - Feels like
    - Pressure
    - Humidity
    - Dew point
    - Clouds
    - Visibility
    - Wind speed
    - Wind degree
    - Weather description
- Weather Report Web Server
  - Use **Python** to program, summarize and visualize statistic data.
  - Build a server using Django. Show the whole information on it.
  - This website also supports query functions.

## Data requirement description for your system

### Load Data Timer

Json format is native supported by node JS and JavaScript. And it should be the format like below:

```
{
  "dt": 1615939200,
  "temp": 294.08,
  "feels_like": 295,
  "pressure": 1012,
  "humidity": 78,
  "dew_point": 290.1,
  "clouds": 20,
  "visibility": 10000,
  "wind_speed": 2.06,
  "wind_deg": 290,
  "weather": [{
    "id": 801,
    "main": "Clouds",
    "description": "few clouds",
    "icon": "02d"
  }]
}
```

In Weather History, the Load Data Timer gets this kind of data from OpenWeatherMap. If the data source changes, it is acceptable. We only need to change the code in Json to Database.

### Json to Database

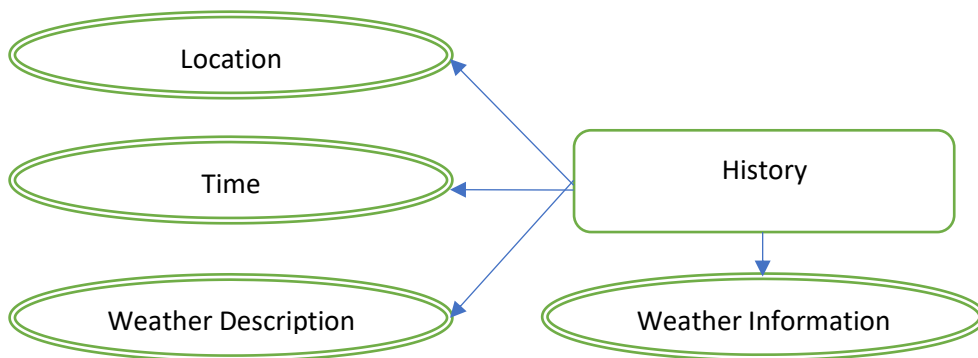
In this part, the data is saved in the below format.

```
{
  "_id": {"$oid": "6080f63b48deeb14f6881f73"},
  "location": "FortWayne",
  "year": 2021,
  "month": 4,
  "day": 9,
  "hour": 20,
  "dt": 1618012800,
  "temp": 293.95,
  "feels_like": 289.9,
  "pressure": 1006,
  "humidity": 35,
  "dew_point": 277.95,
  "clouds": 20,
  "visibility": 16093,
  "wind_speed": 4.12,
  "wind_deg": 250,
  "description_id": 801
}
```

This part is the core feature in Weather History. It is very hard to change the column name, but we can simply add new columns in MongoDB. By adding new columns, Weather History can add new features in frontend.

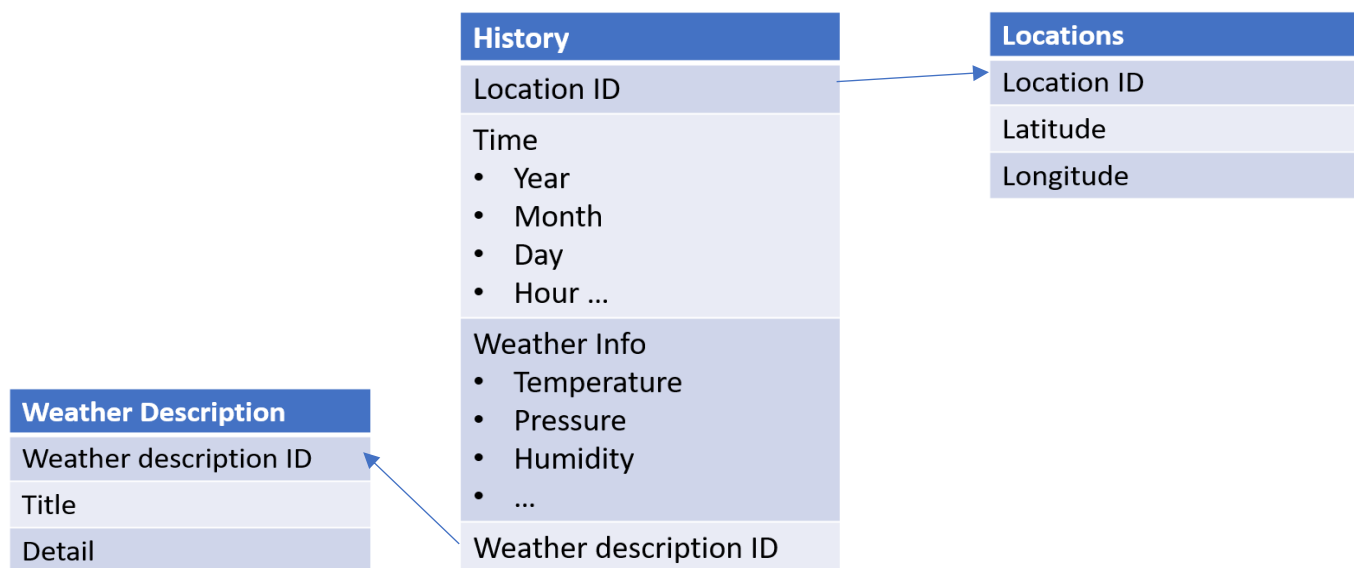
## Conceptual database schema

Because this is a warehouse system, the conceptual database schema is very simple.



## Logical / Physical database schema

The logical database schema shows that there are three collections. History collection is for fact table. For Weather History project, latitude and longitude are not used a lot in frontend, so I simply move them into another collection. Weather Description collection is a fixed collection and contains the summary descriptions for each weather situation.



## System functions

- Get latest weather records of several cities and update them to frontend periodically.
  - This part is done in Load Data Timer.
- Show current weather situation of several cities.
  - This part is done in Weather Report Web Server.
- Historical highest temperatures of several cities.
  - This part is done in Weather Report Web Server.
- Query functions and query web API.
  - This part is done in Weather Report Web Server.

## Database technology

- Data Warehousing
  - MongoDB

## Detail development environment description

- Google Apps Script
  - Load Data Timer
  - In this part, Load Data Timer needs a free google driver account.
  - We also need a free OpenWeatherMap API access key
  - Load Data Timer also needs a time trigger in Google App Script Console.
  - Package:
    - UrlFetchApp
- Python
  - Json To Database
    - Packages:
      - JSON
      - pymongo
  - Django Web Server
    - Package:
      - Django

## Data - real data source or synthetic generation description

- Openweathermap api document
  - 5 days historical data [One Call API - OpenWeatherMap](#)
  - Weather condition id code/ icon [Weather Conditions - OpenWeatherMap](#)

The data can be reached by API calls follow the links above. In Weather History, we use this link to get historical data:

`https://api.openweathermap.org/data/2.5/onecall/timemachine?lat={ lat }&lon={ lon }&dt={ dt }&appid={ API key }`

The return value is a JSON file like below.

```
{
  "lat": 25.0259,
  "lon": 121.6516,
  "timezone": "Asia/Taipei",
  "timezone_offset": 28800,
  "current": {
    "dt": 1619926344,
    ...
  },
  "hourly": [
    {
      "dt": 1619913600,
      ...
    },
    {
      "dt": 1619917200,
      ...
    },
    ...
  ]
}
```

In this project, I mainly use the data in hourly. Every API query contains 24 hours data in this location.

After Json to Database, these hourly data will be imported to MongoDB. In the MongoDB, the data is like below:

#### One record in History Collection

```
{
  "_id": {"$oid": "6080f63b48deeb14f6881f73"},
  "location": "FortWayne",
  "year": 2021,
  "month": 4,
  "day": 9,
  "hour": 20,
  "dt": 1618012800,
  "temp": 293.95,
  "feels_like": 289.9,
  "pressure": 1006,
  "humidity": 35,
  "dew_point": 277.95,
  "clouds": 20,
  "visibility": 16093,
  "wind_speed": 4.12,
  "wind_deg": 250,
  "description_id": 801
}
```

#### One record in Description Collection

```
{
  "_id": 200,
  "main": "Thunderstorm",
  "description": "thunderstorm with light rain"
}
```

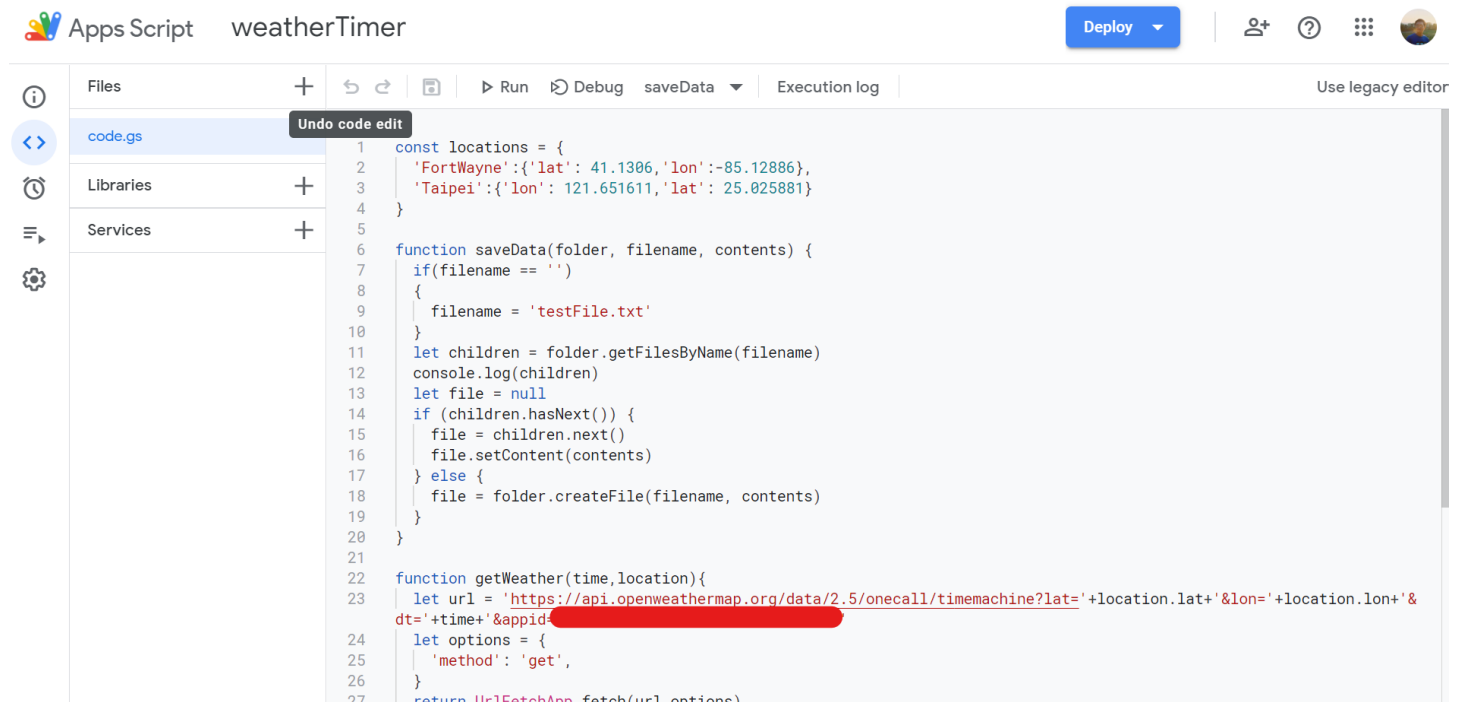
#### One record in Locations Collection

```
{
  "_id": {
    "$oid": "6081de70e5c496cf2505d585"
  },
  "location": "FortWayne",
  "lat": 41.1306,
  "lon": -85.12886
}
```

## The prototype system's functions and demo scenario

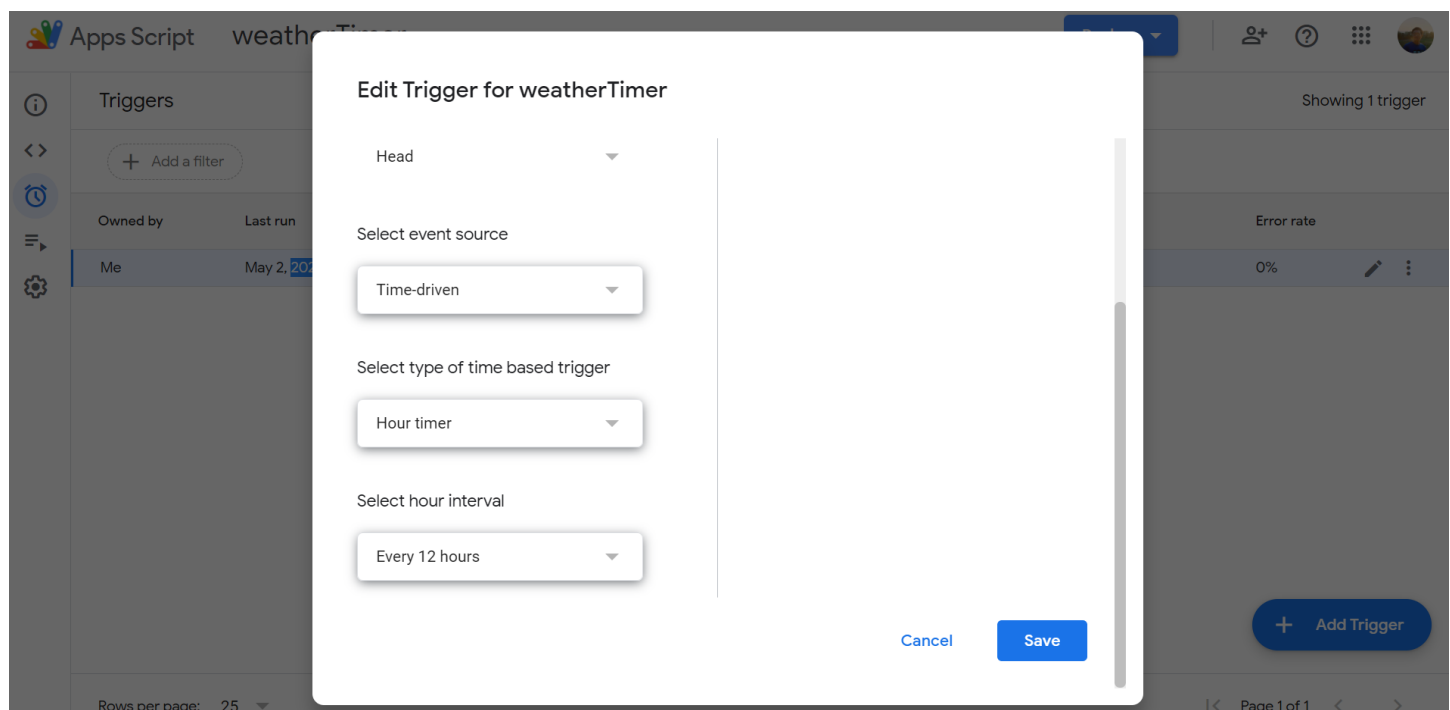
In this part, I will show the system's functions and simple demo screenshots.

- Load Data Timer
  - This part needs to be programmed on google drive.
  - Below shows the code on google drive. The “getWeather” method is an API crawler. Using this function my Google drive can run and get data from OpenWeatherMap and save data into google drive.



```
1 const locations = {
2   'FortWayne': {'lat': 41.1306, 'lon': -85.12886},
3   'Taipei': {'lon': 121.651611, 'lat': 25.025881}
4 }
5
6 function saveData(folder, filename, contents) {
7   if(filename == '')
8   {
9     filename = 'testFile.txt'
10  }
11  let children = folder.getFilesByName(filename)
12  console.log(children)
13  let file = null
14  if (children.hasNext()) {
15    file = children.next()
16    file.setContent(contents)
17  } else {
18    file = folder.createFile(filename, contents)
19  }
20 }
21
22 function getWeather(time, location){
23   let url = 'https://api.openweathermap.org/data/2.5/onecall/timemachine?lat='+location.lat+'&lon='+location.lon+'&dt='+time+'&appid='
24   let options = {
25     'method': 'get',
26   }
27   return UrlFetchApp.fetch(url, options)
```

- After programming, I need to set the timer to launch this method periodically. Below shows that Google AppScript can set a time-driven trigger.





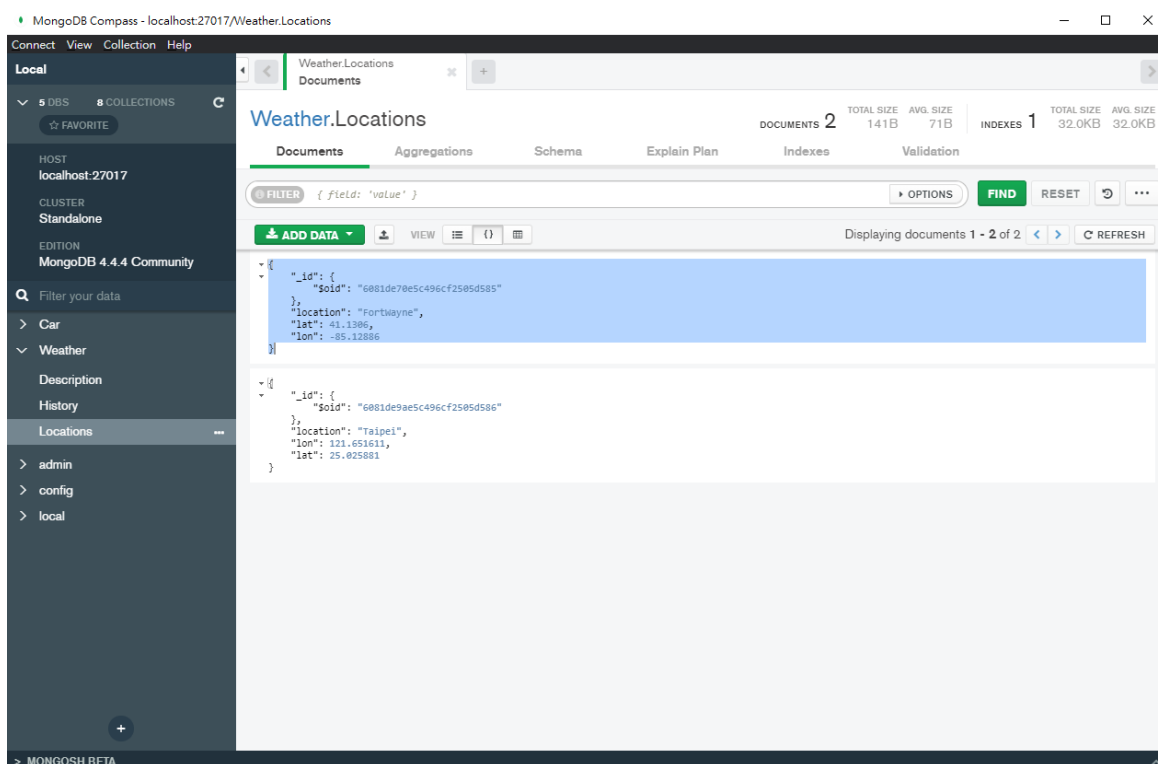
- I use “Backup and Sync” of Google to sync my weather JSON files to local drive.
- Json to Database
  - Below shows that how this parser importing data.
  - If the system host has a long run server, we can combine the Load Data Timer and this Json to Database into a system service. This way can make it totally automate system.

```

1  from pymongo import MongoClient
2  from os import listdir
3  from os.path import join
4  from datetime import datetime
5  import json
6
7  citylist = ['FortWayne', 'Taipei']
8
9  client = MongoClient('localhost:27017')
10 weatherdb=client['Weather']
11 Historycollect = weatherdb['History']
12
13 data_dir = 'sampledata'
14
15 > def insert_json_data(collect, cityname):...
16
17 # mian
18 for city in citylist:
19     insertNum = insert_json_data(Historycollect, city)
20     print(city + ' total insert num:', insertNum)

```

- We can manage the documents and collections on MongoDB Compass.



- Weather Report Web Server
  - This server is powered by Django. One feature of Django is its url paths. In this project, there are 4 url patterns for 4 functions.
    - Main page in '/'
      - This page shows the links of cities and search pages.

## Weather Histroy

please select a location

- [FortWayne](#)
- [Taipei](#)
- [search engine](#)

- Cities in '/FortWayne' and '/Taipei' (it depends on locations collection)
  - In each city page, it shows current weather by the API from above. It also shows the weather history view from MongoDB like below.

---

### Fort Wayne

#### Current weather

Current Local time: 2021-05-02 21:05:25

Temp: 68.32 °F ( 20.18 °C / 293.33 °K)

Feel like: 67.62 °F ( 19.79 °C / 292.94 °K)

Humidity: 59

Wind speed: 7.2

#### Weather Histroy View (2/14/2021 ~ 4/21/2021)

Highest temp: 79.11 °F ( 26.17 °C / 299.32 °K) [occur here](#)

Lowest temp: 0.14 °F ( -17.7 °C / 255.45 °K) [occur here](#)

Average temp: 41.37 °F ( 5.2 °C / 278.35 °K)

Average temp in Spring 45.51 °F

Average temp in Summer no data °F

Average temp in Fall no data °F

Average temp in Winter no data °F

[Back to main](#)

- In the summary view, we can see that if the column is from specific one record, there is a link called "occur here". Click will show that record in a single page. This page is in '/history/'

- Each record in '/history/{ record object id }'
  - This page show a single record.

---

## FortWayne

### One Record - Weather Histroy

Local Time: 2021/4/7 16:00

Temp: 79.11 °F ( 26.17 °C / 299.32 °K)

Feel like: 73.08 °F ( 22.82 °C / 295.97 °K)

Pressure: 1010

Dew point: 283.9

Humidity: 38

Wind speed: 5.14

In Simple: Clouds

Description: scattered clouds: 25-50%

[Back to main](#)

- Query Search page in '/search'
  - This page shows a flex query page. Users can select needed columns and query the data.

## Search

Please fill out your bounds, or left blank if you don't need it.

Location:

Local Time:  ~

Temp in °K:  ~

Feel like temp in °K:  ~

Pressure:  ~

Dew point:  ~

Humidity:  ~

Wind speed:  ~

Description:

[Back to main](#)

- After clicking submit, this page can show the result view and the records including in this query.

## Search

Please fill out your bounds, or left blank if you don't need it.

Location:

Local Time:  ~

Temp in °K:  ~

Feel like temp in °K:  ~

Pressure:  ~

Dew point:  ~

Humidity:  ~

Wind speed:  ~

Description:

Highest temp: 46.22 °F ( 7.9 °C / 281.05 °K) [occur here](#)

Lowest temp: 21.52 °F ( -5.82 °C / 267.33 °K) [occur here](#)

Average temp: 31.95 °F ( -0.03 °C / 273.12 °K)

Highest feel like temp: 37.06 °F ( 2.81 °C / 275.96 °K) [occur here](#)

Highest temp: 46.22 °F ( 7.9 °C / 281.05 °K) [occur here](#)

Lowest temp: 21.52 °F ( -5.82 °C / 267.33 °K) [occur here](#)

Average temp: 31.95 °F ( -0.03 °C / 273.12 °K)

Highest feel like temp: 37.06 °F ( 2.81 °C / 275.96 °K) [occur here](#)

Lowest feel like temp: 11.88 °F ( -11.18 °C / 261.97 °K) [occur here](#)

Average feel like temp: 21.33 °F ( -5.93 °C / 267.22 °K)

46 Records

[FortWayne 3/31/2021 22:00](#)

[FortWayne 3/31/2021 23:00](#)

[FortWayne 4/1/2021 0:00](#)

[FortWayne 4/1/2021 1:00](#)

[FortWayne 4/1/2021 2:00](#)

[FortWayne 4/1/2021 3:00](#)

[FortWayne 4/1/2021 4:00](#)

[FortWayne 4/1/2021 5:00](#)

- Query Search API function in '/search\_post'
  - The output of this API is a HTML file. The same as above.

Highest temp: 46.22 °F ( 7.9 °C / 281.05 °K) [occur here](#)

Lowest temp: 21.52 °F ( -5.82 °C / 267.33 °K) [occur here](#)

Average temp: 31.95 °F ( -0.03 °C / 273.12 °K)

Highest feel like temp: 37.06 °F ( 2.81 °C / 275.96 °K) [occur here](#)

Lowest feel like temp: 11.88 °F ( -11.18 °C / 261.97 °K) [occur here](#)

Average feel like temp: 21.33 °F ( -5.93 °C / 267.22 °K)

46 Records

[FortWayne 3/31/2021 22:00](#)

[FortWayne 3/31/2021 23:00](#)

[FortWayne 4/1/2021 0:00](#)

[FortWayne 4/1/2021 1:00](#)

[FortWayne 4/1/2021 2:00](#)

[FortWayne 4/1/2021 3:00](#)

[FortWayne 4/1/2021 4:00](#)

[FortWayne 4/1/2021 5:00](#)

## Discussion and limitations of your work

This Weather History has its pros and cons. The good parts are:

- Easy to import new locations
  - If system host wants to add more locations, it needs only one step on a long run server version: add cities information into locations collection.
  - If system host does not own a long run server, this system need to change 2 parts: add cities information on Load Data Timer and locations collection.
- Easy to implement new functions.
  - Can slightly change the front page and add a function at backend to build a new feature.
  - Because Django's MVC structure, adding new feature only needs to add functions in model part and add view functions in view part.
- Easy to deploy.
  - MongoDB, Django are cross platform tools.
  - These tools can be found on Linux and Windows.
  - Comparing to other tools and databases, MongoDB and Django are lite weight but powerful tools.

On the other side, it has its limitations:

- Only have 3 months / 2 locations dataset
  - Need time to collect.
  - Need to pay for older history records on OpenWeatherMap
- Show Table or dynamic graph on Django
  - Not easy to show real time analysis figure on web page using Django.
  - It may be deployed on a heavy weight web server.
- JSON to DB part still not an automatic process in without long run server version.
  - If want to automate the process, need a long run server and deploy it as a system service.

## Conclusion and possible future works (directions)

Conclusions

- MongoDB support vary Query functions. It is easy to build a frontend query APIs and interfaces.
- This project has high scalability, can easily add extensions like more locations.
- If budget is enough, we can use a long run server to build a total automatic server.

Future Works

- Show Table or dynamic graph
- Use Box plot instead of text description
- Add more locations
- Automate all the process

## Appendix

- Openweathermap api document
  - Free account quotas [Pricing - OpenWeatherMap](#)
  - 5 days historical data [One Call API - OpenWeatherMap](#)
  - Weather condition id code/ icon [Weather Conditions - OpenWeatherMap](#)
- Google App Script document
  - Free account quotas [Quotas | Apps Script | Google Developers](#)
  - Code document [Overview of Google Apps Script | Google Developers](#)
- MongoDB document
  - Installation [MongoDB Community Download | MongoDB](#)
  - APIs for Python and Node.js [The MongoDB 4.4 Manual — MongoDB Manual](#)