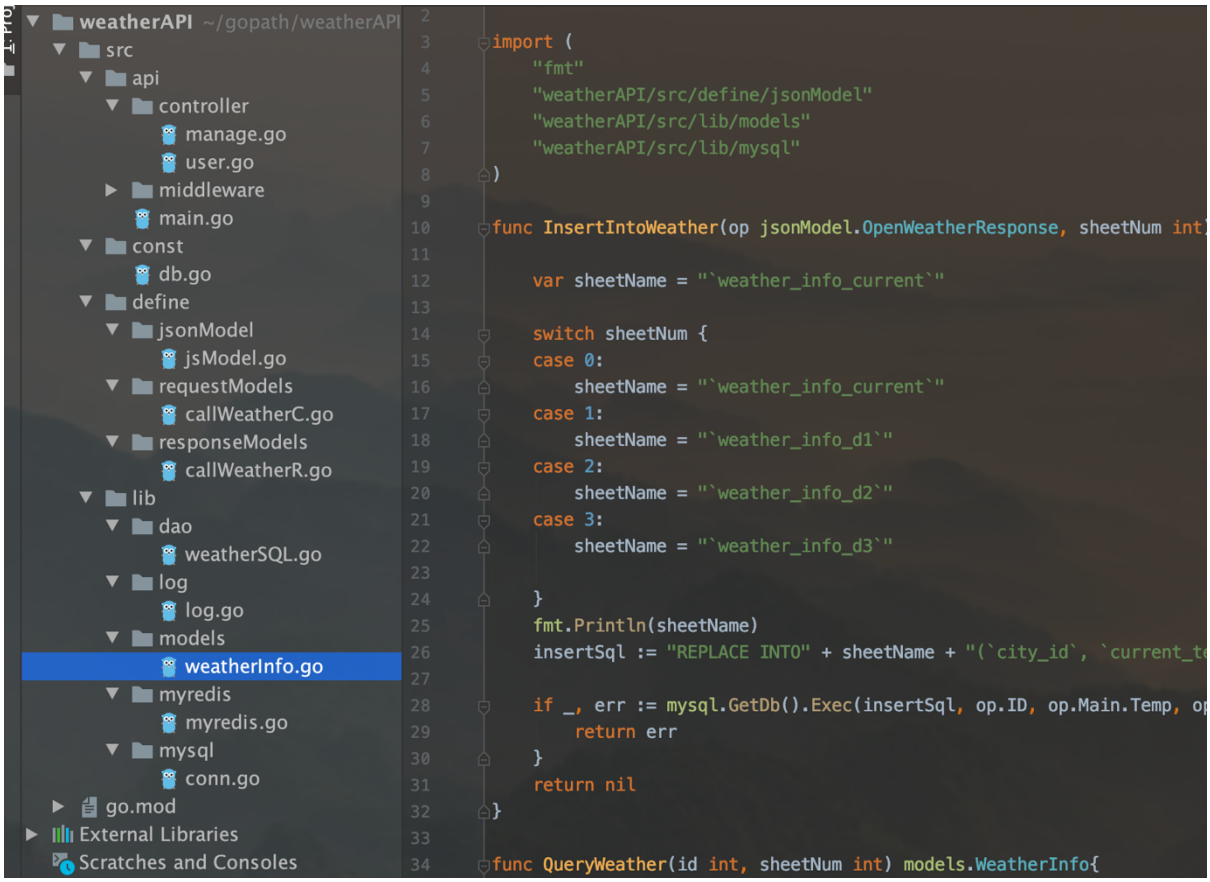
Weather API Doc

Codes Structure:



APIs:

- Query weather
  User requests a post with API to get a JSON package like:


```
{
    "citylds" : [2150126, 2150106, 2150096]
}
```


Body 200 OK 3 ms 1.93 KB Save Re

Pretty Raw Preview Visualize JSON


```
{
    "2150096": [
        {
            "city_id": 2150096,
            "current_temperature": 292.59,
            "weather_condition": "Clear",
            "weather_code": 800,
            "highest_tp": 292.59,
            "lowest_tp": 292.59,
            "dt": 1612868361
        },
        {
            "city_id": 2150096,
            "current_temperature": 292.59,
            "weather_condition": "Clear",
            "weather_code": 800,
            "highest_tp": 292.59,
            "lowest_tp": 292.59,
            "dt": 1612868361
        },
        {
            "city_id": 2150096,
            "current_temperature": 292.59,
            "weather_condition": "Clear",
            "weather_code": 800,
            "highest_tp": 292.59,
            "lowest_tp": 292.59,
            "dt": 1612868361
        },
        {
            "city_id": 2150096,
            "current_temperature": 292.59,
            "weather_condition": "Clear",
            "weather_code": 800,
            "highest_tp": 292.59,
            "lowest_tp": 292.59,
            "dt": 1612868361
        }
    ],
    "2150106": [
        {
            "city_id": 2150106,
            "current_temperature": 292.79,
            "weather_condition": "Clouds",
            "weather_code": 804,
            "highest_tp": 293.15,
            "lowest_tp": 292.04,
            "dt": 1612868342
        },
        {
            "city_id": 2150106,
            "current_temperature": 292.79,
            "weather_condition": "Clouds",
            "weather_code": 804,
            "highest_tp": 293.15,
            "lowest_tp": 292.04,
            "dt": 1612868342
        },
        {
            "city_id": 2150106,
            "current_temperature": 292.79,
            "weather_condition": "Clouds",
            "weather_code": 804,
            "highest_tp": 293.15,
            "lowest_tp": 292.04,
            "dt": 1612868342
        },
        {
            "city_id": 2150106,
            "current_temperature": 292.79,
            "weather_condition": "Clouds",
            "weather_code": 804,
            "highest_tp": 293.15,
            "lowest_tp": 292.04,
            "dt": 1612868343
        }
    ],
    "2150126": [
        {
            "city_id": 2150126,
            "current_temperature": 297.04,
```


the frontEnd can decode this JSON package as a list, which is a convenience for searching and save to list, and also swipe the screen to left or right

- Async weather via open weather API into MySQL

  Just database design fulfills the free plan of OpenWeather. I can not get history data via API with multiple requests. The API gave the same weather, although I input a different timestamp. So the request weather API returns the same data but with a sequence from the current day to the future three days.

```
[GIN-debug] POST   /api/v1/user/callWeather  --> weatherAPI/src/api/controller.CallWeather
[GIN-debug] Environment variable PORT is undefined. Using port :8080 by default
[GIN-debug] Listening and serving HTTP on :8080
http://api.openweathermap.org/data/2.5/weather?APPID=5013a32cc734deec3c8fefbbef1fab5f&dt=1
1612867266 1612780866
::1 - [Tue, 09 Feb 2021 10:41:07 UTC] "POST /api/v1/asyncWeather HTTP/1.1 200 579.893096ms
{"level":"info","ts":1612867267.150659,"caller":"middleware/middleware.go:17","msg":"::1 2
http://api.openweathermap.org/data/2.5/weather?APPID=5013a32cc734deec3c8fefbbef1fab5f&dt=1
1612867235 1612780877
::1 - [Tue, 09 Feb 2021 10:41:17 UTC] "POST /api/v1/asyncWeather HTTP/1.1 200 244.785822ms
{"level":"info","ts":1612867277.325788,"caller":"middleware/middleware.go:17","msg":"::1 2
http://api.openweathermap.org/data/2.5/weather?APPID=5013a32cc734deec3c8fefbbef1fab5f&dt=1
1612867266 1612694485
::1 - [Tue, 09 Feb 2021 10:41:25 UTC] "POST /api/v1/asyncWeather HTTP/1.1 200 248.821239ms
{"level":"info","ts":1612867285.798831,"caller":"middleware/middleware.go:17","msg":"::1 2
```

2: Favorites 7: Structure

Terminal 3: Find 6: TODO

Database Implement:

I first use XORM to create tables with structs

```go
type WeatherInfoCurrent struct {
    CityId             int    `xorm:"pk" json:"city_id"`
    CurrentTemperature float64  `xorm:"notnull" json:"current_temperature"`
    WeatherCondition   string  `xorm: "notnull" json:"weather_condition"`
    WeatherCode        int    `xorm: "notnull" json:"weather_code"`
    HighestTP          float64 `xorm:"notnull" json:"highest_tp"`
    LowestTP           float64 `xorm: "notnull" json:"lowest_tp"`
    DT                 int    `xorm:"updated" json:"dt"`
}
type WeatherInfoD1 struct {
    CityId             int    `xorm:"pk" json:"city_id"`
    CurrentTemperature float64  `xorm:"notnull" json:"current_temperature"`
    WeatherCondition   string  `xorm: "notnull" json:"weather_condition""`
    WeatherCode        int    `xorm: "notnull" json:"weather_code"`
    HighestTP          float64 `xorm:"notnull" json:"highest_tp"`
    LowestTP           float64 `xorm: "notnull" json:"lowest_tp"`
    DT                 int    `xorm:"notnull" json:"dt"`
}
type WeatherInfoD2 struct {
    CityId             int    `xorm:"pk" json:"city_id"`
    CurrentTemperature float64  `xorm:"notnull" json:"current_temperature"`
    WeatherCondition   string  `xorm: "notnull" json:"weather_condition""`
    WeatherCode        int    `xorm: "notnull" json:"weather_code"`
    HighestTP          float64 `xorm:"notnull" json:"highest_tp"`
    LowestTP           float64 `xorm: "notnull" json:"lowest_tp"`
    DT                 int    `xorm:"notnull" json:"dt"`
}
type WeatherInfoD3 struct {
    CityId             int    `xorm:"pk" json:"city_id"`
    CurrentTemperature float64  `xorm:"notnull" json:"current_temperature"`
    WeatherCondition   string  `xorm: "notnull" json:"weather_condition""`
    WeatherCode        int    `xorm: "notnull" json:"weather_code"`
    HighestTP          float64 `xorm:"notnull" json:"highest_tp"`
    LowestTP           float64 `xorm: "notnull" json:"lowest_tp"`
```

WeatherInfoCurrent

I create four tables: weatherInfoCurtent, weatherInfoD1, weatherInfoD2, weatherInfoD3.

But in the implementation of data query and insert, I use native sql, because ORM structure is complicated, and I needed to write four structure for four tables for each API(async and query)

Optional task:

I used Redis into Gin, but the result was not well

```
(Decex) ✘ rongbaizhang@192-168-1-108 ~/PycharmProjects/SQLtask/weather python apiPerformance.py
0:00:05.750877 with redis 1000times
0:00:04.873221 without redis 1000times
(Decex) rongbaizhang@192-168-1-108 ~/PycharmProjects/SQLtask/weather python apiPerformance.py
0:01:03.406530 with redis 10000times
0:00:57.599475 without redis 10000times
(Decex) rongbaizhang@192-168-1-108 ~/PycharmProjects/SQLtask/weather python apiPerformance.py
0:11:26.807948 with redis 100000times
0:09:57.382788 without redis 100000times
(Decex) rongbaizhang@192-168-1-108 ~/PycharmProjects/SQLtask/weather python apiPerformance.py
```

```go
} else {
    DataWeather := make(map[int][]models.WeatherInfo)
    conn := myredis.RedisDefaultPool.Get()
    defer conn.Close()
    for _, id := range stat.CityIds {
        s := make([] models.WeatherInfo, 0)
        redisKey := "WC" + strconv.Itoa(id)
        ret, err := redis.Bytes(conn.Do( commandName: "get", redisKey))
        weatherObj := []models.WeatherInfo{}
        if err != nil {
            for i := 0; i < 4; i++ {
                res := dao.QueryWeather(id, i)
                s = append(s, res)
            }
            retDate, _ := ffjson.Marshal(s)
            conn.Do( commandName: "setex", redisKey, 2000, retDate)

            DataWeather[id] = s

        }else {
            ffjson.Unmarshal(ret, &weatherObj)

            DataWeather[id] = weatherObj
        }
```