成都理工大學中英合作办学
CDUT Sino-British Collaborative Education

OXFORD BROOKES UNIVERSITY

UNDERGRADUATE PROJECT PROPOSAL

Project Title:	**Web- based Weather forecasting system**
Surname:	**Li**
First Name:	**Rox**
Student Number:	**201918020305**
Supervisor Name:	**Aymen Chebira**
Module Code:	**CHC 6096**
Module Name:	**Project**
Date Submitted:	**January 13, 2023**

Table of Contents

1 Introduction

1.1 Background (overview of topic and motivation)

Weather forecasting is an important and indispensable procedure in people's daily lives and, according to Selvam and Brorsson (2022), weather and climate science play a crucial role in all forms of life.

1.2 Aim

It can be used to assess the changes that occur in the current atmospheric conditions, so it is very important to predict the weather more accurately.

1.3 Objectives

After some research I found that most weather forecasting systems require downloading a mobile application, so the need to open the website directly for easy real-time viewing is indispensable. Therefore, I decided to use a set of distributed microservices Spring Cloud to complete a convenient and fast Web-based weather forecasting system. This will be a microservices project based on Spring Cloud's microservices architecture and the components or modules covered in it are built on Spring Boot.

Here's what I need to finish in this semester:

Understanding Spring Boot, environment setup, getting started projects

Build development environment and develop controller using Gradle

Create project

Boosting Concurrent Access to Applications with redis

Using Quartz scheduler to get weather data at regular intervals

Create UI for weather forecast

Understanding monolithic and microservice architectures

Weather forecasting system microservice (1) architecture design

Weather Forecasting System Microservices (2) Weather Data Collection Microservices

Weather Forecasting System Microservices (3) Weather Data API Microservices

Weather Forecasting System Microservices (4) Weather Forecasting Microservices

Weather Forecasting System Microservices (5) City Data API Microservices

1.4 Project Overview

1.4.1 Scope

The system will be used in the Internet weather forecast website, when clicked into the website will be able to see all the searchable weather information on the home page. To sum of, I will implement the following seven application frameworks in general: registry - service registration and discovery, config - external configuration, monitor - monitoring, zipkin - distributed tracking, gateway - interface gateway to proxy all microservices, auth-service - OAuth2 authentication service, svc-service - business service. Based on these services, I hope to provide a convenient Web-based weather forecasting system for users who do not want to download mobile applications. This will be a convenient web-based weather forecasting system for all users who want to know what the weather is like right now and what the weather will be like in the future.

1.4.2 Audience

The weather forecast is relevant to all people's lives and it can provide weather related alerts to all people. Whether you are at home or need to go out (work, travel, etc.), you can be proactive for different situations.

2 Background Review

2.1 Summary of existing approaches

Compare my choice of SpringCloud technology, the techniques that can be seen in the current market to develop weather forecasting systems are 1. The use of ARIMA technique to develop weather forecasting tools, Autoregressive Integrated Moving Average (ARIMA) is a data mining technique commonly used for time series analysis and future forecasting (Shivhare et al., 2019). ; 2. now the mainstream microservices framework Dubbo, which uses a custom Dubbo protocol to achieve remote communication, is a typical RPC invocation scheme(Zhao, Jiang and Zhao, 2020). And SpringCloud uses Feign is based on Rest style invocation;

Compare this to SpringCloud+SpringBoot which I will use to implement microservices development. Specifically, SpringCloud has the core technology for microservices development: RPC remote call technology; SpringBoot's web component integrates SpringMVC by default, which enables lightweight HTTP+JSON transfer and writing microservice interfaces, so SpringCloud relies on SpringBoot framework for micro So SpringCloud relies on SpringBoot framework for microservices development.			
Using the SpringBoot	Spring Boot Web Starter and Spring Boot Data Redis Starter running on Redis.	By default, spring-boot-starter-web provides a set of HttpMessageconverters for type conversion of request parameters and response results.	SpringBoot avoids a lot of development environment import and saves testing time and effort, Springcloud is a microservice solution - RPC remote calls, relationship Springcloud depends on SpringBoot (SpringMVC for web components). Springcloud will depend on with SpringBoot, because Springcloud write interface is SpringMVC interface, but due to the rapid iteration of the

			version, some modules will be very large changes, but as a simple implementation of the weather forecast site will not be affected too much.

2.2 Brief summary of related literature

Regarding Ramachandran et al.'s Healthy Weather app (2021), it is proposed that the app is based on to Android users, so I still prioritize the development of a weather forecasting website when targeting the relevant audience for all masses.

Technology reference	Summary	Evaluation	Reflection
Using the Android Volley Library	The Healthy Weather app uses an HTTP library called Volley Library. Volley's two main classes, Request Queue and Request, are used so that all the necessary information for making network API calls is stored in them.	It makes networking very easy and fast	Volley is designed to be ideal for network operations that do not involve large amounts of data but frequent communication, while for network operations that involve large amounts of data, such as downloading files, Volley performs very poorly.
Using the Flask Framework	Flask is a lightweight Python based web framework that supports Python 2 and Python 3 and is easy to use and suitable for rapid development.	It enables fast and easy production of web applications with the ability to improve complex applications.	It is too lightweight and requires more call experience for the developer.

3 Project Technical Progress

3.1 Methodology

3.1.1 Approach

According to the explanatory model of Health Weather App by Ramachandran et al. (2021), the health weather app provides a lot of relevant information in a combination of graphics and text so that it is not challenging for the user to understand. However, after reviewing the article, there are no restrictions for people living at home to use the app, such as download time and mobile device storage space, but for travelers or tourists, an app needs to be downloaded and installed in advance, which proves that it requires the mobile device to provide the appropriate memory storage space in a timely manner. This is a problem that I do not have to consider for the web-based weather forecasting I am going to implement.

Configuration files

The default configuration file for Spring Boot is properties, where we can define various configuration information such as container port numbers, database connection information, logging levels, etc., depending on the different Starter modules we introduce. I also found that the configuration file also supports the use of YAML files. After comparison it became apparent that the YAML configuration information was more legible using stepped indentation and that the amount of characters in the configuration content was significantly reduced.

In some special cases, we want some parameters to be loaded with more than a fixed value each time, such as keys, service ports, etc. In Spring Boot's property configuration file, it is possible to generate random int values, long values or string strings by using the ${random} configuration, so that we can easily generate properties randomly through the configuration, rather than coding the logic in the application to implement them. This configuration allows scenarios such as application ports to be set to avoid the hassle of port conflicts when debugging locally.

application-dev.properties: development environment

application-test.properties: test environment

application-prod.properties: production environment

Spring Boot uses a rather unusual order of loading properties as follows.

1. parameters passed in on the command line.

2. properties in SPRING_APPLICATION_JSON, which are configured in JSON format in the system environment variables.

3. The JNDI property in java:comp/env.

4. Java's system properties, the contents of which can be obtained via System.getProperties().

5. environment variables of the operating system.

6. random properties configured via random.*.

7. The contents of a configuration file for a different {profile} environment, such as application-{profile}.properties or a YAML defined configuration file, located outside the current application jar package.

8. The contents of a profile for a different {profile} environment, such as application-{profile}.properties or a YAML-defined profile, located within the current application jar package.

9. application.properties and YAML configuration content located outside the current application jar package.

10. application.properties and YAML configuration content located inside the current application jar package.

11. properties defined by the @PropertySource annotation in the class modified by the @Configuration annotation.

12. the application default properties, using the content defined by SpringApplication.setDefaultProperties.

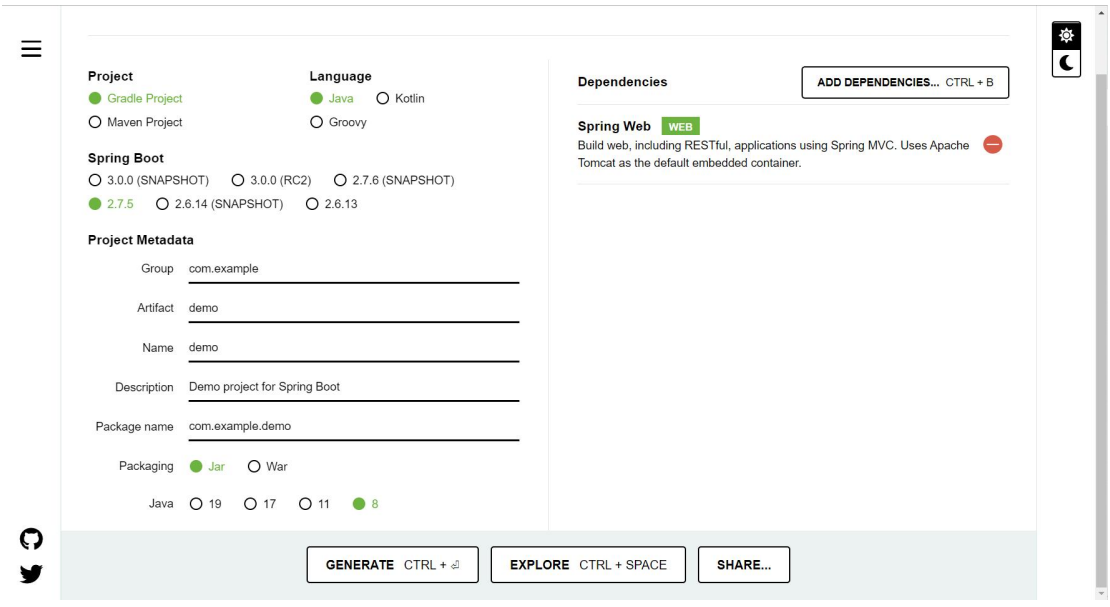
When we decided to use Spring Boot as a microservices framework, apart from its powerful rapid development features, it was also because it provides a special dependency module spring-boot-starter-actuator in the Starter POMs, the introduction of which automatically provides Spring Boot-built applications with a set of endpoints for monitoring endpoints for applications built with Spring Boot. Spring Cloud further extends this module when implementing individual microservice components, for

example by adding more metrics and metrics to the native endpoints (e.g. the /health endpoint when integrating with Eureka), and by providing more free endpoints depending on the component (e.g. /routine for the API gateway component). Zuul provides the /routes endpoint to return routing information).

The implementation of the spring-boot-starter-actuator module can be effective for small to medium sized teams implementing microservices, eliminating or significantly reducing the amount of development required for monitoring systems to capture application metrics. Of course, it is not a panacea and sometimes simple extensions are needed to help us personalise our monitoring needs.

3.1.2 Technology

Flask is a framework written in Python language and easy to understand the code , it is a language compared to the requirements of SpringBoot2.7.5 higher, because it is too lightweight , so developers need to have more experience in the development of third-party library integration calls(Mufid et al., 2019). For comparison, Due to the large base size, data and frequent operations required by the system (pei and peng, 2022), I used the SpringCloud microservices framework when choosing the technology to serve the web-based Weather forecasting system. The Gradle project language is Java. Dependency tool I choose Spring Web.



(Figure1: Spring Initializr)

Spring Initializr open source project .

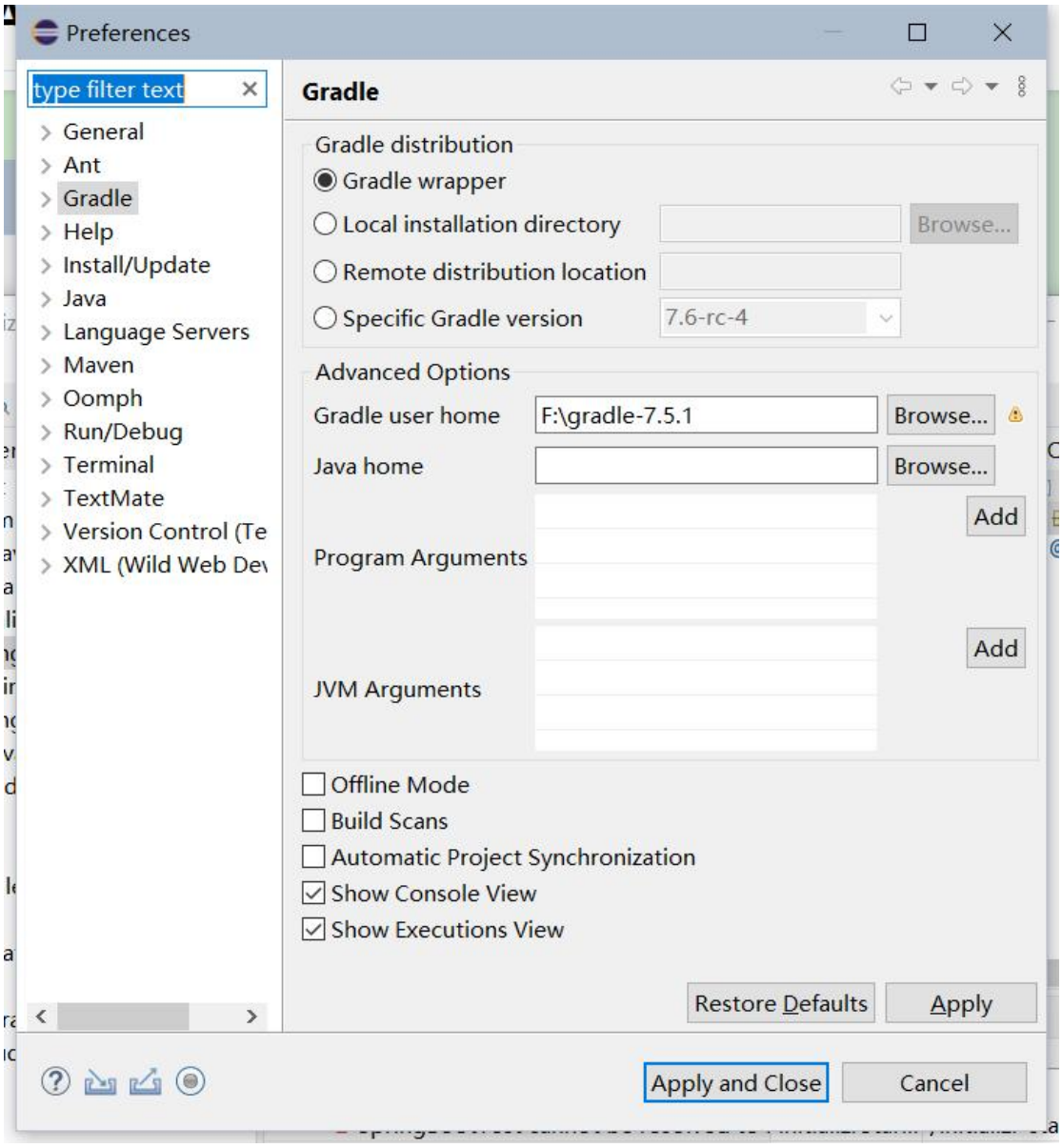
.gitignore: versioning the ignore file

build.gradle: gradle project configuration file (core)

gradlew: execution script under Linux

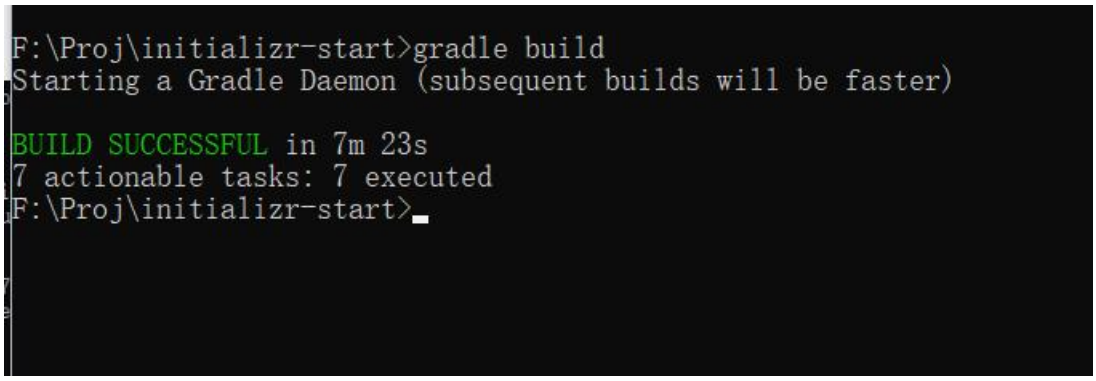
gradlew.bat: execution script under Windows

Eclipse Gradle development environment configuration:



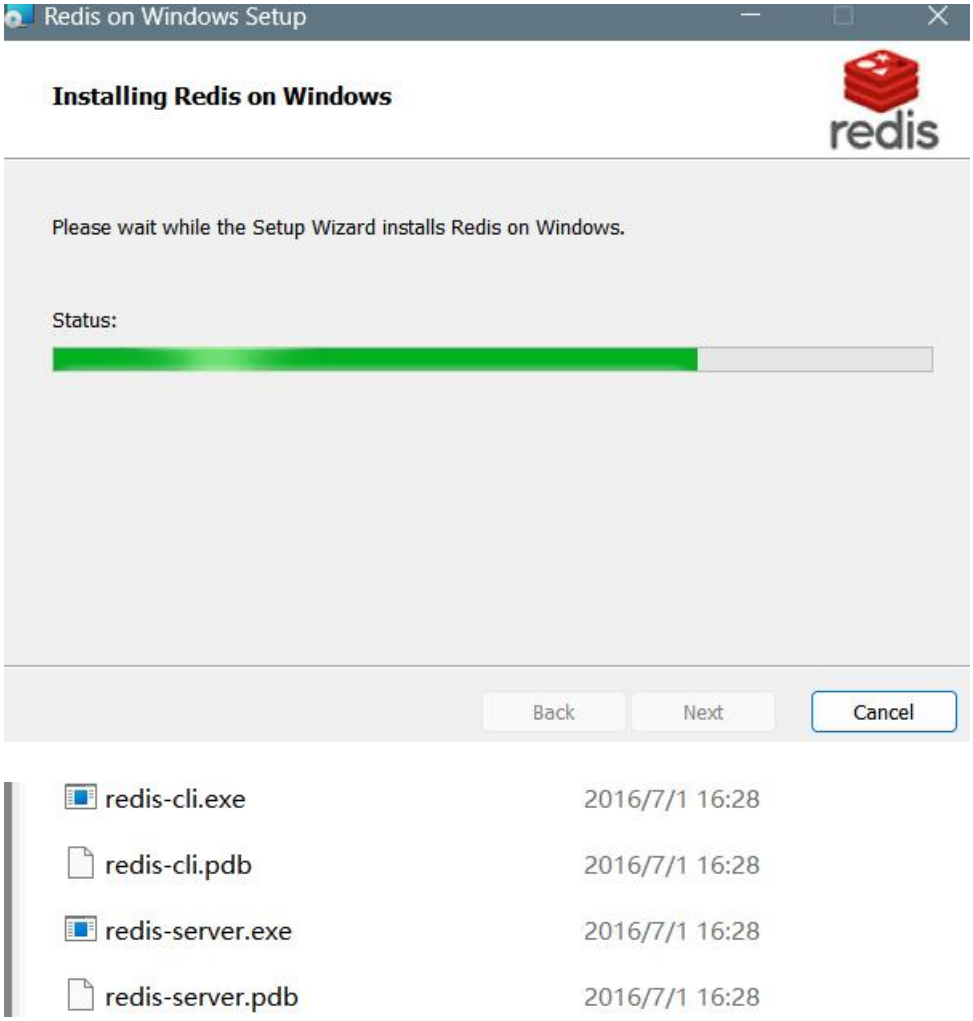
(Figure2: Gradle)

Compile the project Gradle build - compile successfully:



(Figure3: Gradle)

Install and run Redis



(Figure4: Redis)

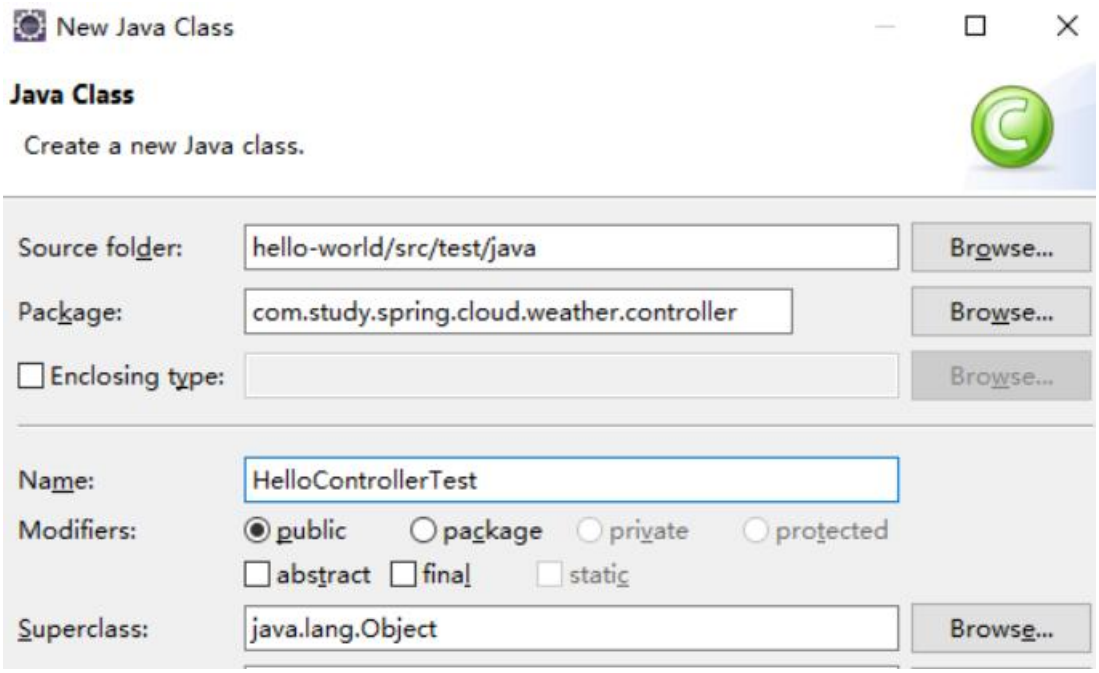
```
        //Redis
compile 'org.springframework.boot:spring-boot-starter-data-redis'

        //Quartz
compile 'org.springframework.boot:spring-boot-starter-quartz'
```

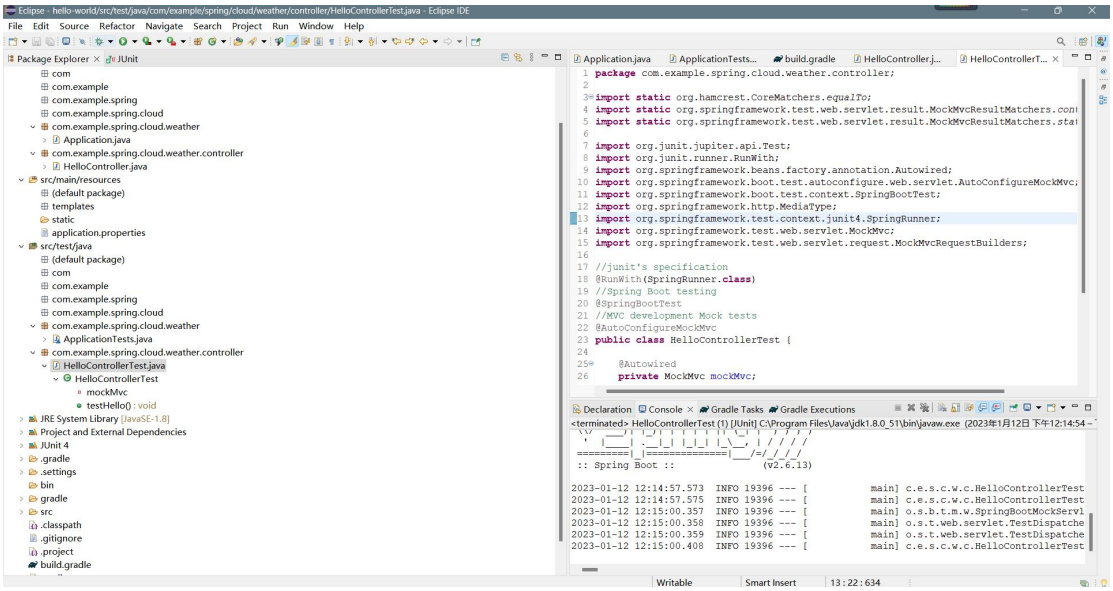
3.2 Testing and Evaluation

Writing Test Cases (Hello-world)

Create a new controller package:

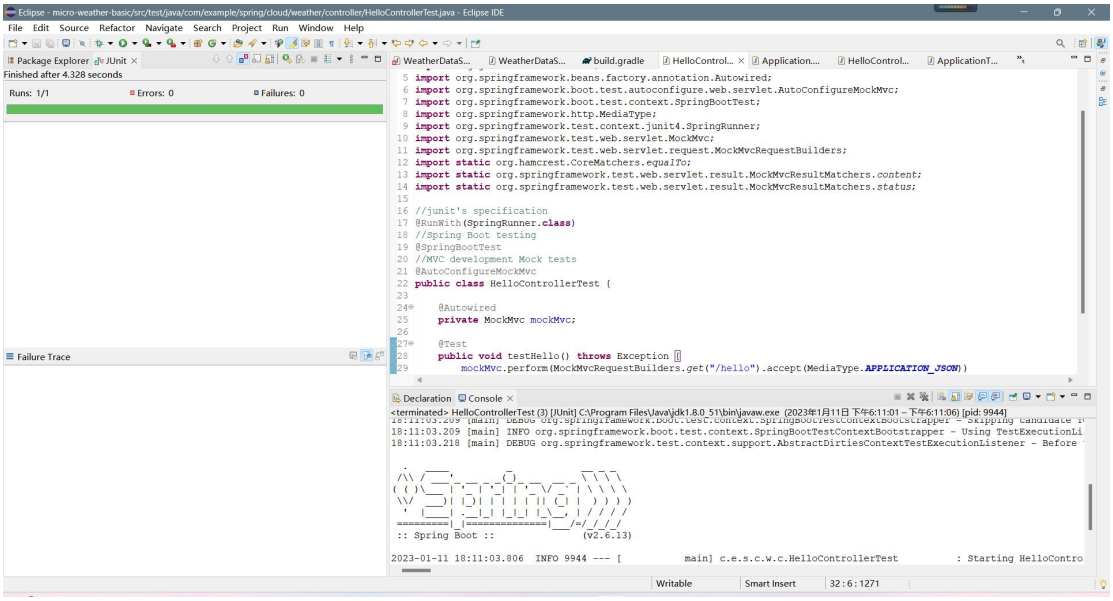


(Figure5)



(Figure6)

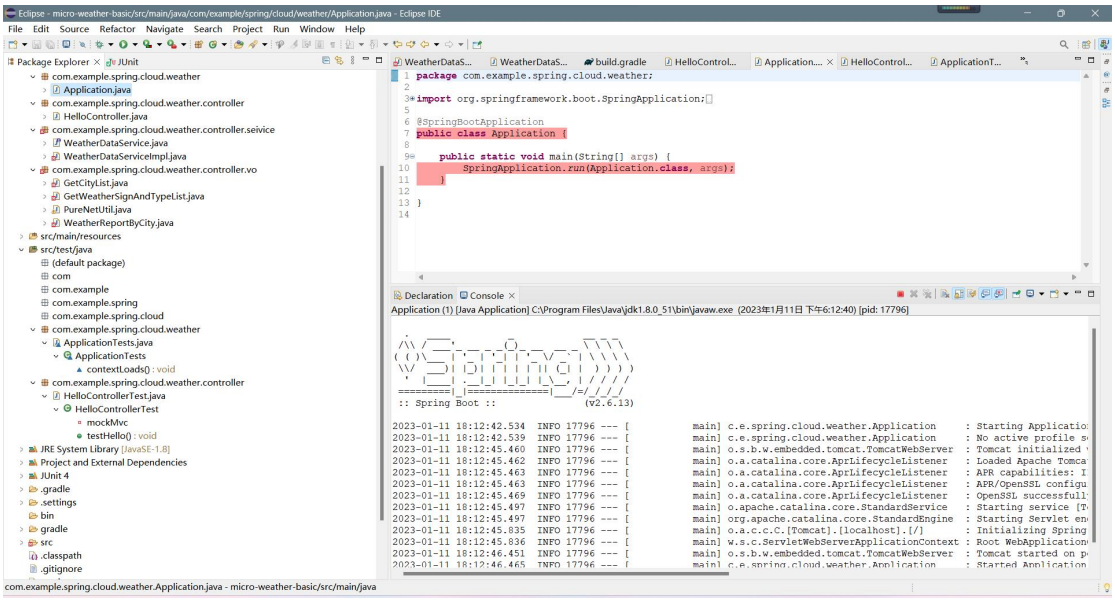
Run test class results display:



(Figure7)

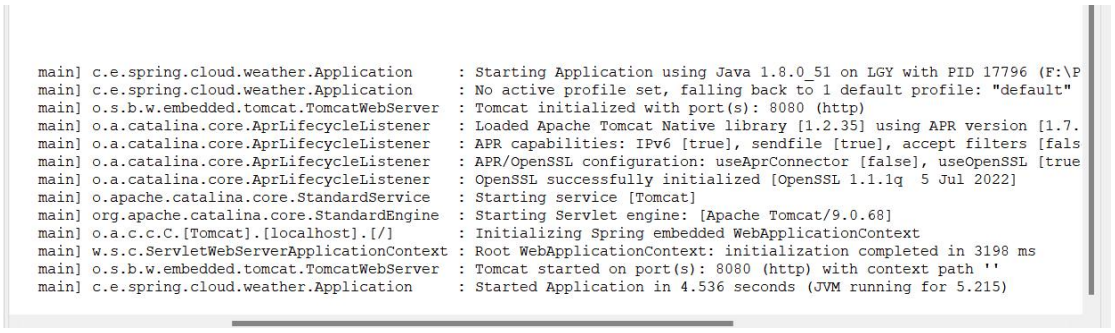
Running Spring Boot

Right-click to Run as Java Application



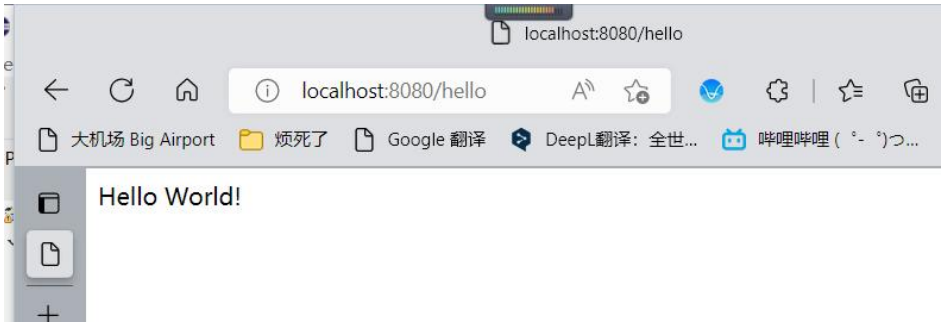
(Figure8)

Tomcat on port 8080:



(Figure9)

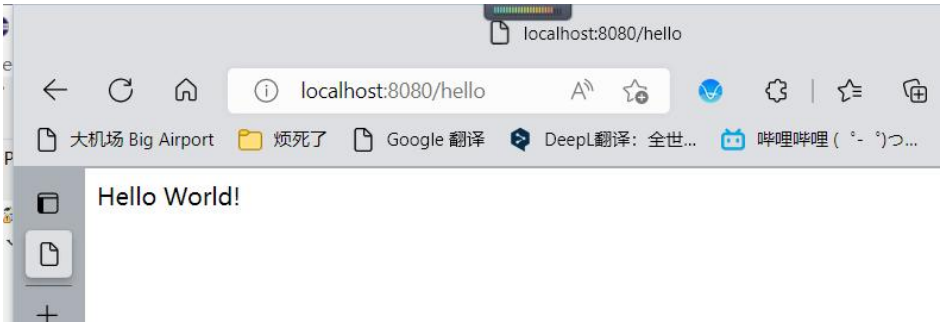
localhost:8080/hello



(Figure10)

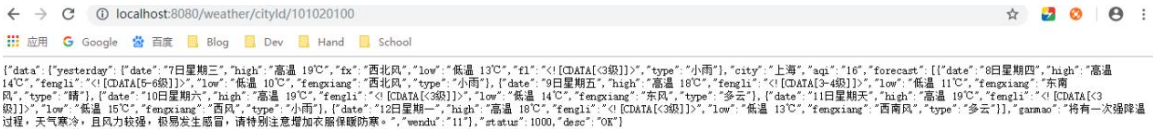
3.3 Design and Implementation

localhost:8080/hello



(Figure11)

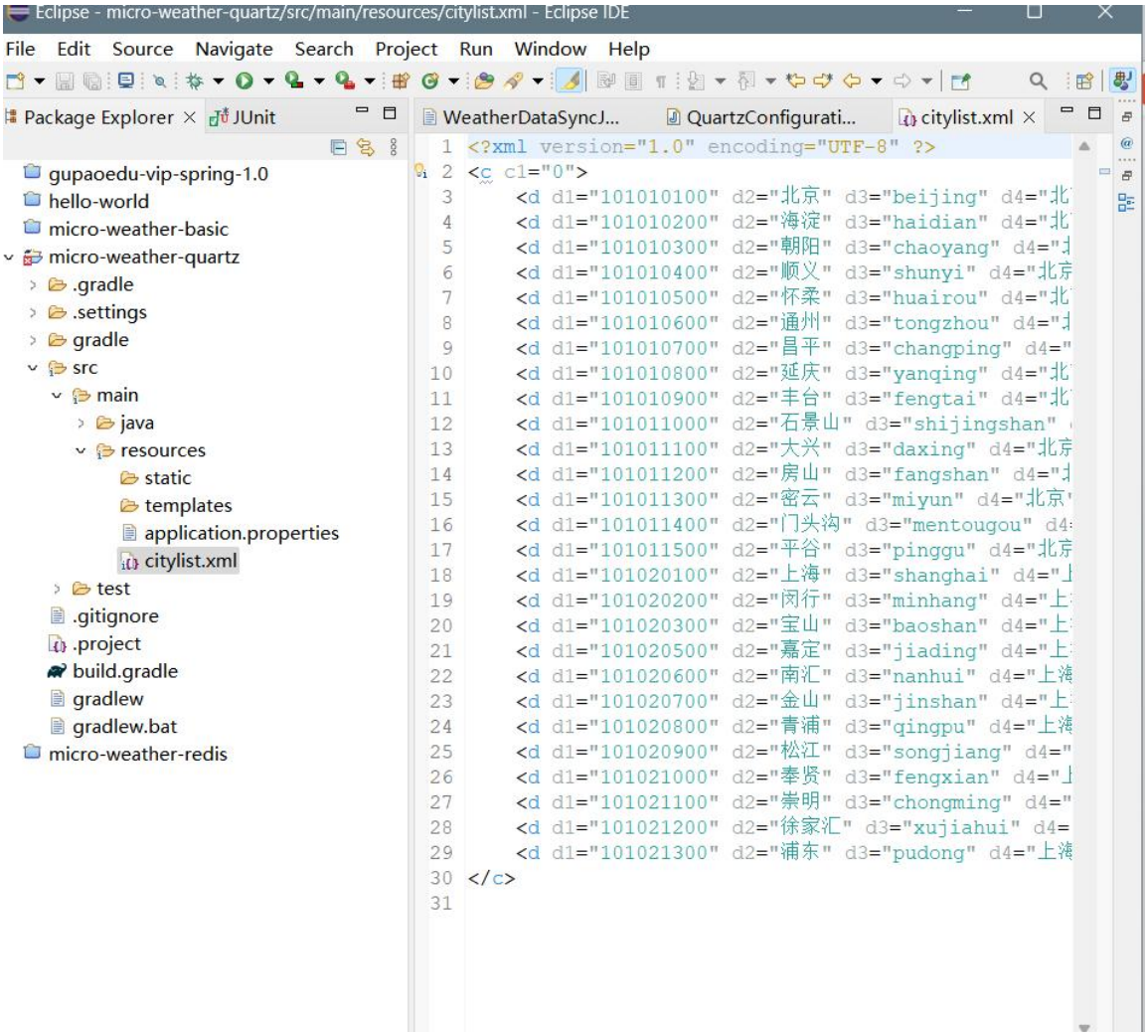
Use Redis to improve the concurrent access capability of applications:



(Figure12)

The quartz scheduler acquires weather data on a regular basis:

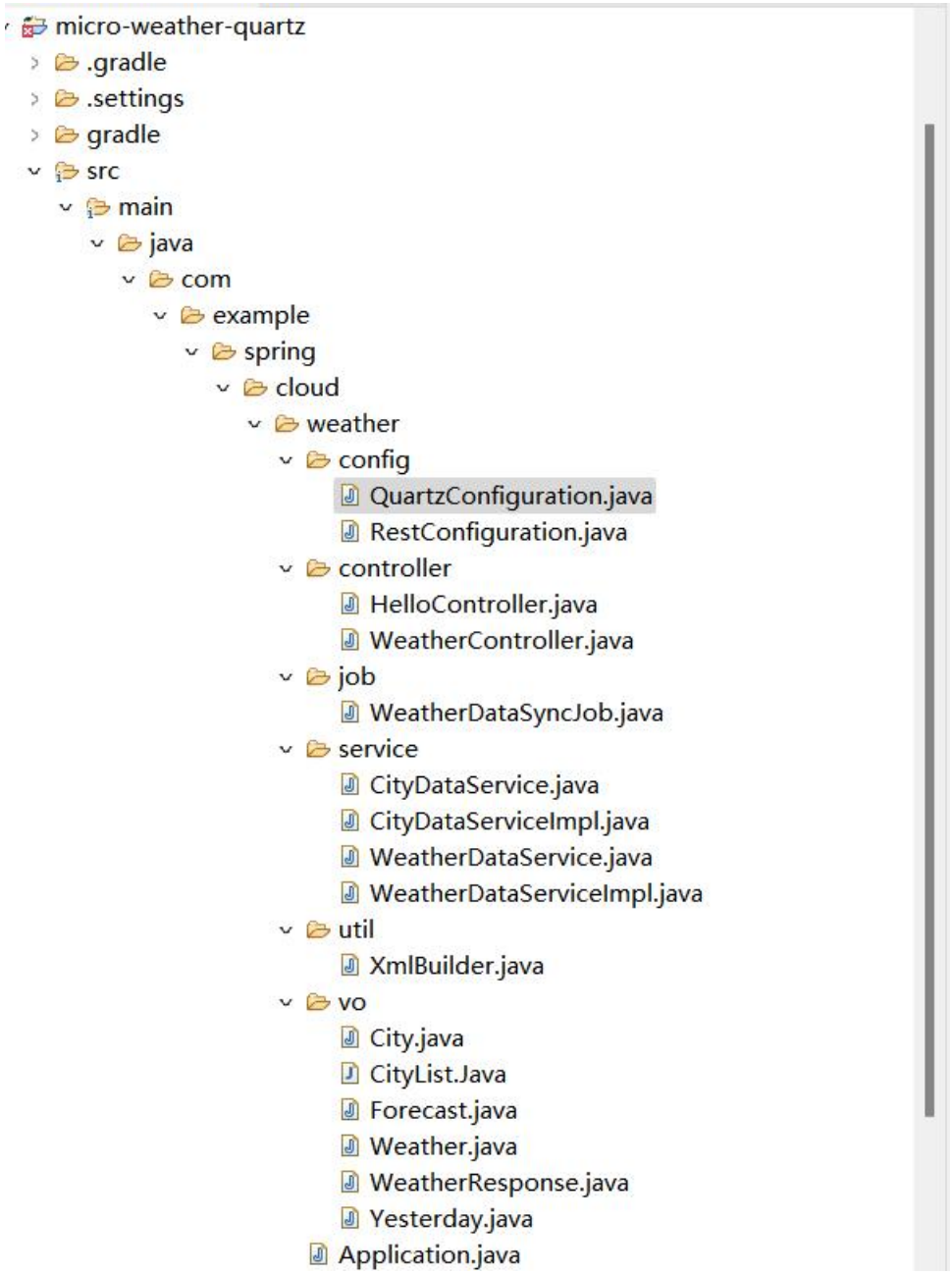
The citylist:



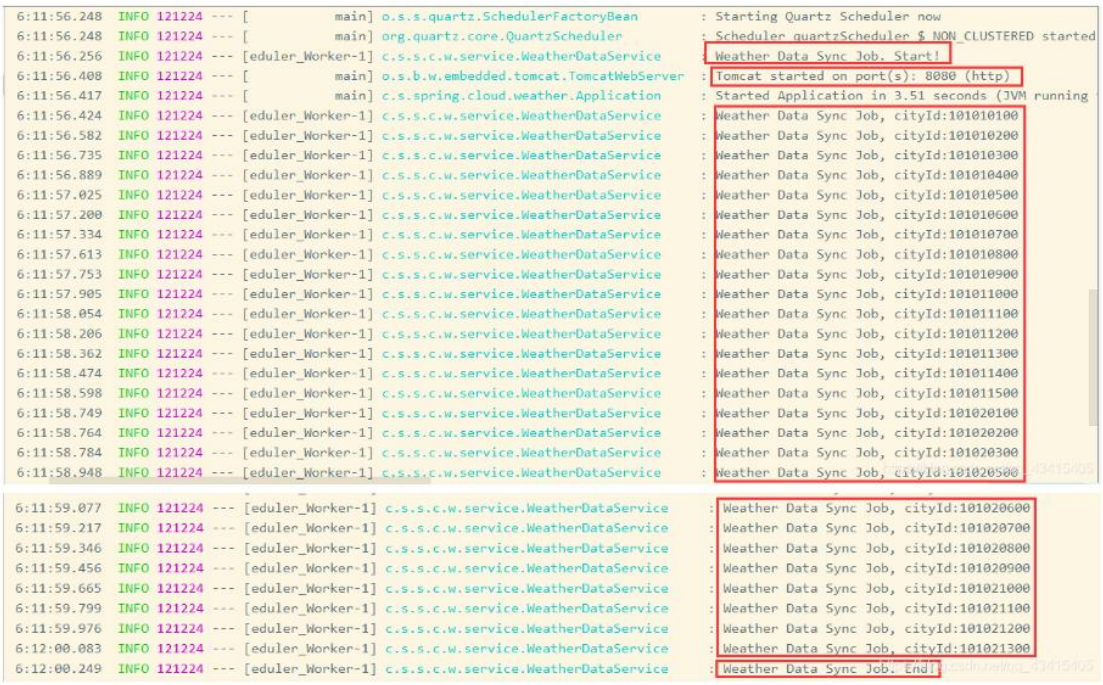
(Figure13)

Run Redis

Final running result:

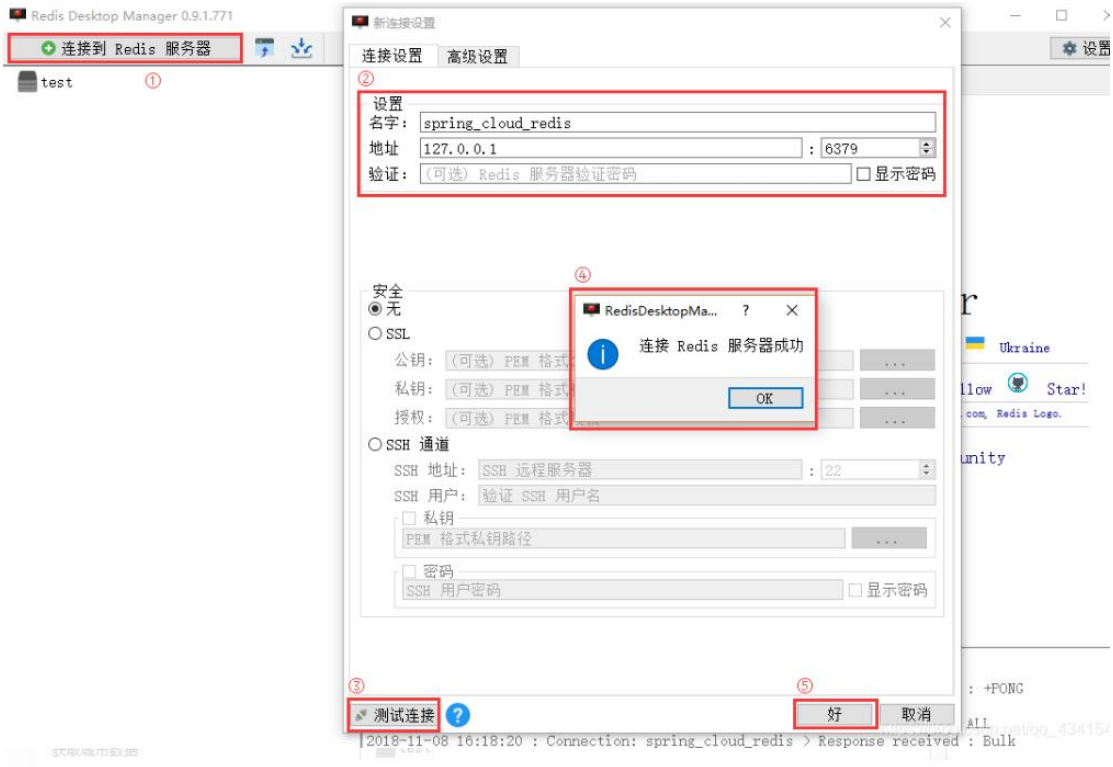

micro-weather-quartz
.gradle
.settings
gradle
src
main
java
com
example
spring
cloud
weather
config
QuartzConfiguration.java
RestConfiguration.java
controller
HelloController.java
WeatherController.java
job
WeatherDataSyncJob.java
service
CityDataService.java
CityDataServiceImpl.java
WeatherDataService.java
WeatherDataServiceImpl.java
util
XmlBuilder.java
vo
City.java
CityList.Java
Forecast.java
Weather.java
WeatherResponse.java
Yesterday.java
Application.java

(Figure14)



(Figure15)

Use Redis Desktop Manager：



(Figure16)

Create UI for weather forecast (tentative Chinese system)：

Features of the weather forecast service

Search by different cities

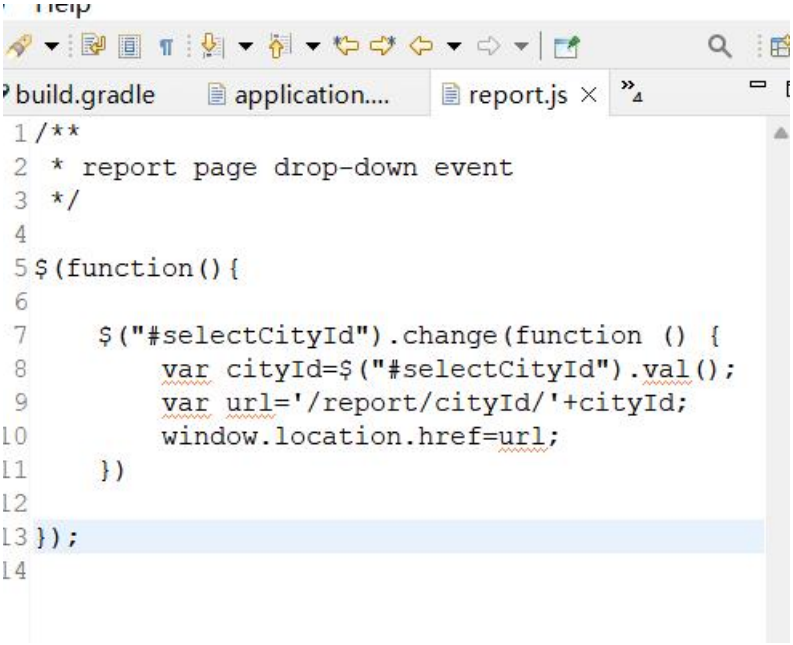
Check the weather information for the last few days

Simple and elegant interface

API of the weather forecast service

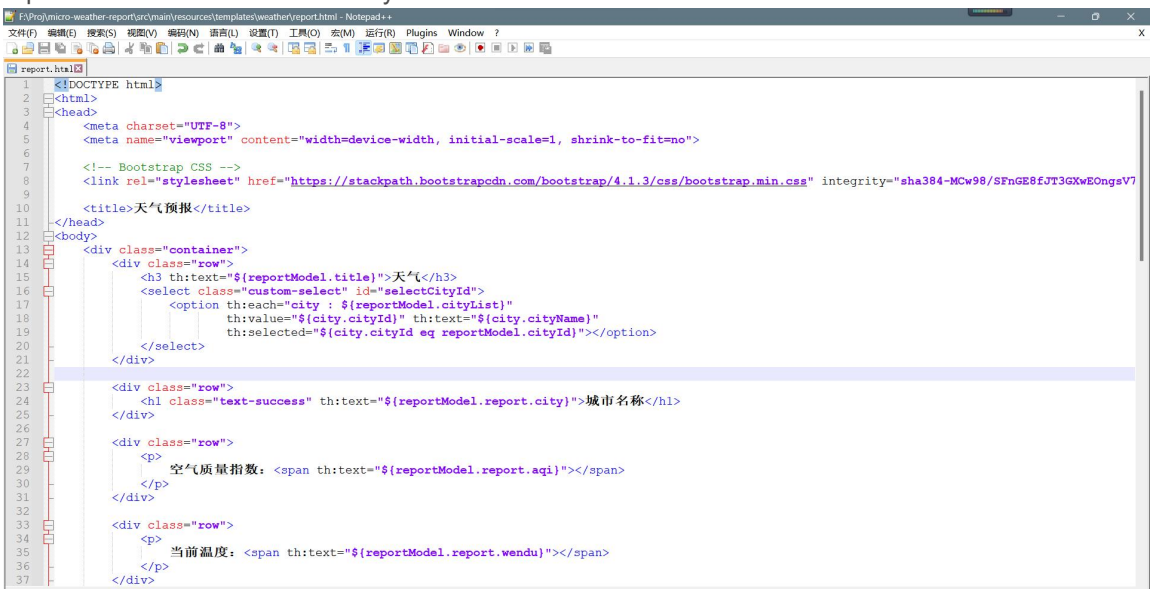
Get the weather information for the city ID: GET/report/cityId/{cityId}

Create a js directory in the static directory of resources, then create a weather directory in the js directory, and a new report.js file in the weather directory:



(Figure17)

Create the weather directory in the templates directory of resources and a new front-end page report.html in the weather directory：



```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css" integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7

    <title>天气预报</title>
</head>
<body>
    <div class="container">
        <div class="row">
            <h3 th:text="${reportModel.title}">天气</h3>
            <select class="custom-select" id="selectCityId">
                <option th:each="city : ${reportModel.cityList}"
                        th:value="${city.cityId}" th:text="${city.cityName}"
                        th:selected="${city.cityId eq reportModel.cityId}"></option>
            </select>
        </div>

        <div class="row">
            <h1 class="text-success" th:text="${reportModel.report.city}">城市名称</h1>
        </div>

        <div class="row">
            <p>
                空气质量指数：<span th:text="${reportModel.report.aqi}"></span>
            </p>
        </div>

        <div class="row">
            <p>
                当前温度：<span th:text="${reportModel.report.wendu}"></span>
            </p>
        </div>
```

(Figure18)

运行结果：

(Figure19)

(Figure20)

4 Project Management

4.1 Activities: tasks required to complete each objective

Technology Stack:

Spring boot - entry-level micro framework for microservices, used to simplify the initial build of Spring applications and the development process.

Eureka - Cloud Service Discovery, a REST-based service for locating services for cloud middle-tier service discovery and failover.

Spring Cloud Config - Configuration management toolkit that allows you to put configuration on remote servers and centralize cluster configuration, currently supporting local storage, Git, and Subversion.

Hystrix - Fuse, a fault-tolerant management tool designed to provide greater fault tolerance for latency and failure by controlling the nodes of services and third-party libraries through a fusing mechanism.

Zuul - Zuul is a framework for providing dynamic routing, monitoring, elasticity, security and other edge services on the cloud platform. zuul is the front door to all requests on the back end of the web site for devices and Netflix streaming applications.

Spring Cloud Bus - event, message bus for propagating state changes in clusters (e.g., configuration change events) , which can be used in conjunction with Spring Cloud Config for hot deployment .

Spring Cloud Sleuth - Log collection toolkit that encapsulates Dapper and log-based tracing as well as Zipkin and HTrace operations, enabling a distributed tracing solution for SpringCloud applications.

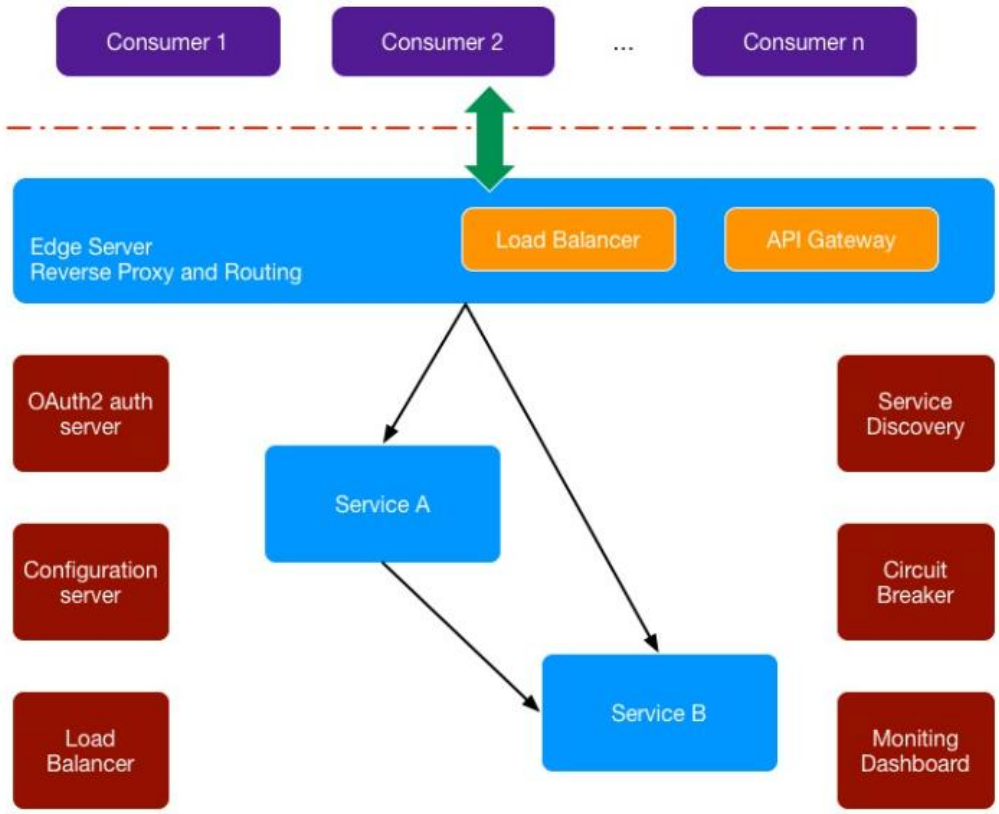
Ribbon - Provides cloud-based load balancing with a choice of load balancing policies that can be used with service discovery and circuit breakers.

Turbine - Turbine is a tool for aggregating event stream data sent by servers to monitor the metrics of hystrix under a cluster.

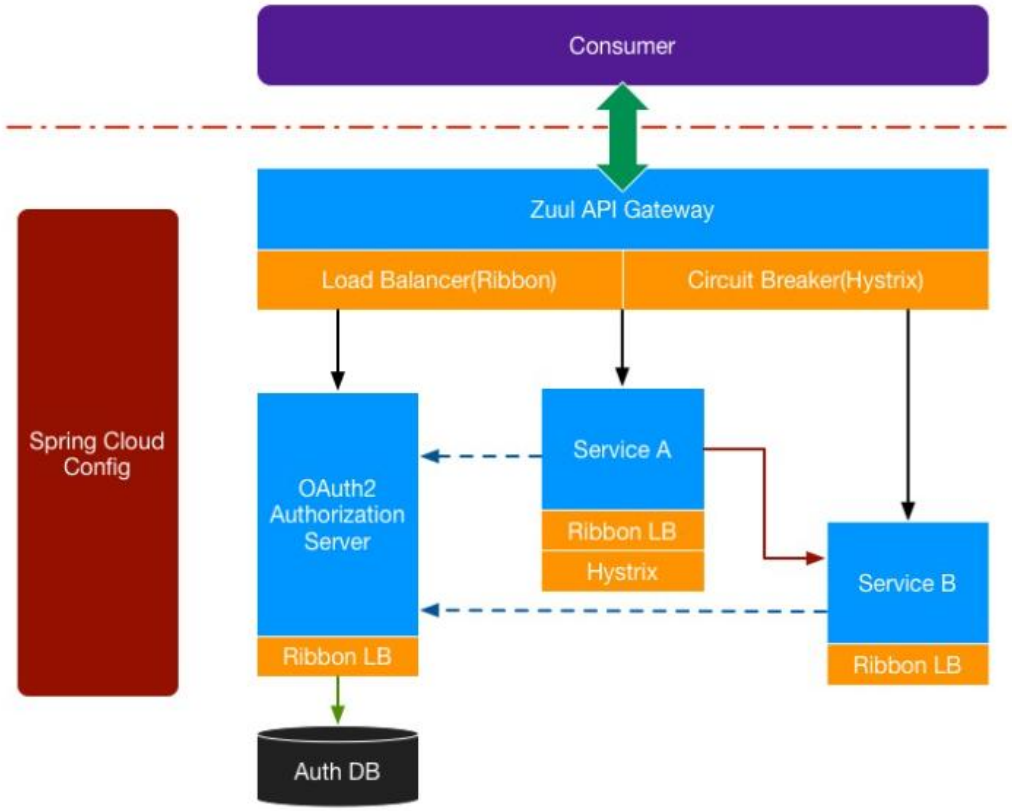
Spring Cloud Stream - Spring data streaming operations development kit that encapsulates sending and receiving messages with Redis, Rabbit, Kafka, etc.

Feign - Feign is a declarative, templated HTTP client.

Spring Cloud OAuth2 - A security toolkit based on Spring Security and OAuth2 to add security controls to your applications.



(Figure21: System Architecture)



(Figure 22: Application Components)

4.2 Schedule

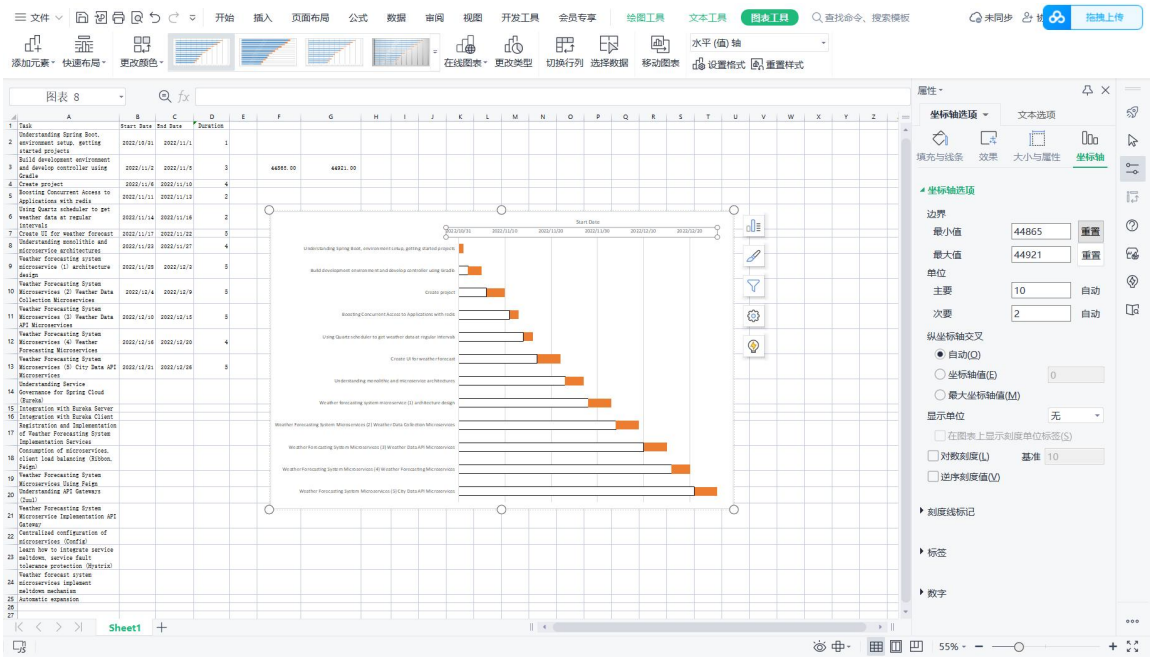
Tasks:

1. Understanding Spring Boot, environment setup, getting started projects
2. Build development environment and develop controller using Gradle
3. Create project
4. Boosting Concurrent Access to Applications with redis
5. Using Quartz scheduler to get weather data at regular intervals
6. Create UI for weather forecast
7. Understanding monolithic and microservice architectures
8. Weather forecasting system microservice (1) architecture design
9. Weather Forecasting System Microservices (2) Weather Data Collection Microservices

10. Weather Forecasting System Microservices (3) Weather Data API Microservices
11. Weather Forecasting System Microservices (4) Weather Forecasting Microservices
12. Weather Forecasting System Microservices (5) City Data API Microservices
13. Understanding Service Governance for Spring Cloud (Eureka)
14. Integration with Eureka Server
15. Integration with Eureka Client
16. Registration and Implementation of Weather Forecasting System Implementation Services
17. Consumption of microservices, client load balancing (Ribbon, Feign)
18. Weather Forecasting System Microservices Using Feign
19. Understanding API Gateways (Zuul)
20. Weather Forecasting System Microservice Implementation API Gateway
21. Centralized configuration of microservices (Config)
22. Learn how to integrate service meltdown, service fault tolerance protection (Hystrix)
23. Weather forecast system microservices implement meltdown mechanism
24. Automatic expansion

	A	B	C	D
1	Task	Start Date	End Date	Duration
2	Understanding Spring Boot, environment setup, getting started projects	2022/10/31	2022/11/1	1
3	Build development environment and develop controller using Gradle	2022/11/2	2022/11/5	3
4	Create project	2022/11/6	2022/11/10	4
5	Boosting Concurrent Access to Applications with redis	2022/11/11	2022/11/13	2
6	Using Quartz scheduler to get weather data at regular intervals	2022/11/14	2022/11/16	2
7	Create UI for weather forecast	2022/11/17	2022/11/22	5
8	Understanding monolithic and microservice architectures	2022/11/23	2022/11/27	4
9	Weather forecasting system microservice (1) architecture design	2022/11/28	2022/12/3	5
10	Weather Forecasting System Microservices (2) Weather Data Collection Microservices	2022/12/4	2022/12/9	5
11	Weather Forecasting System Microservices (3) Weather Data API Microservices	2022/12/10	2022/12/15	5
12	Weather Forecasting System Microservices (4) Weather Forecasting Microservices	2022/12/16	2022/12/20	4
13	Weather Forecasting System Microservices (5) City Data API Microservices	2022/12/21	2022/12/26	5
14	Understanding Service Governance for Spring Cloud (Eureka)			
15	Integration with Eureka Server			
16	Integration with Eureka Client			
17	Registration and Implementation of Weather Forecasting System Implementation Services			
18	Consumption of microservices, client load balancing (Ribbon, Feign)			
19	Weather Forecasting System Microservices Using Feign			
20	Understanding API Gateways (Zuul)			
21	Weather Forecasting System Microservice Implementation API Gateway			
22	Centralized configuration of microservices (Config)			
23	Learn how to integrate service meltdown, service fault tolerance protection (Hystrix)			
24	Weather forecast system microservices implement meltdown mechanism			
25	Automatic expansion			

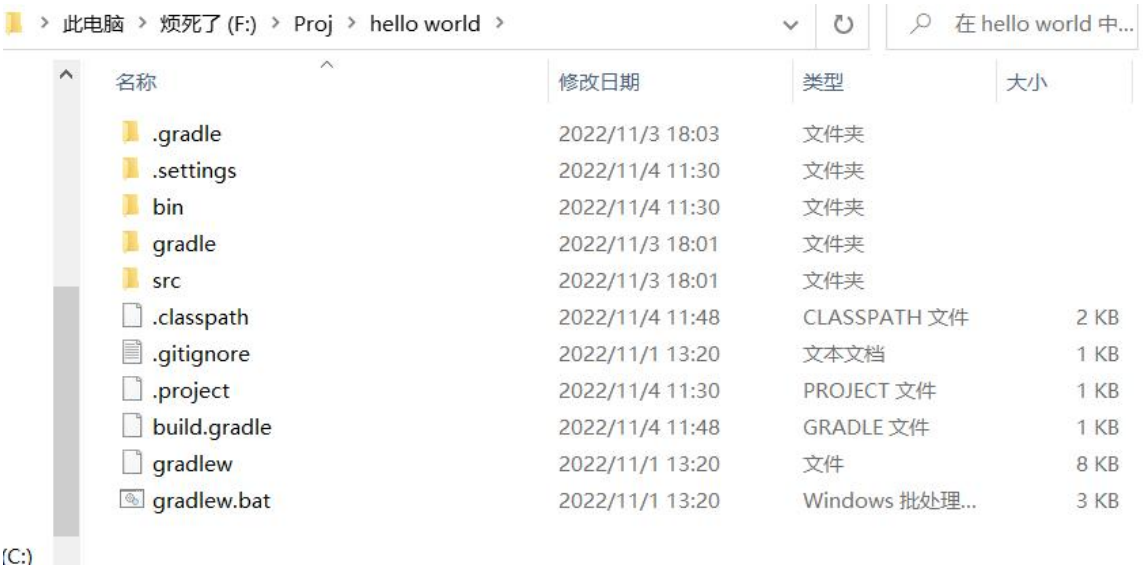
(Figure23: Tasks, Start Date, End Date)



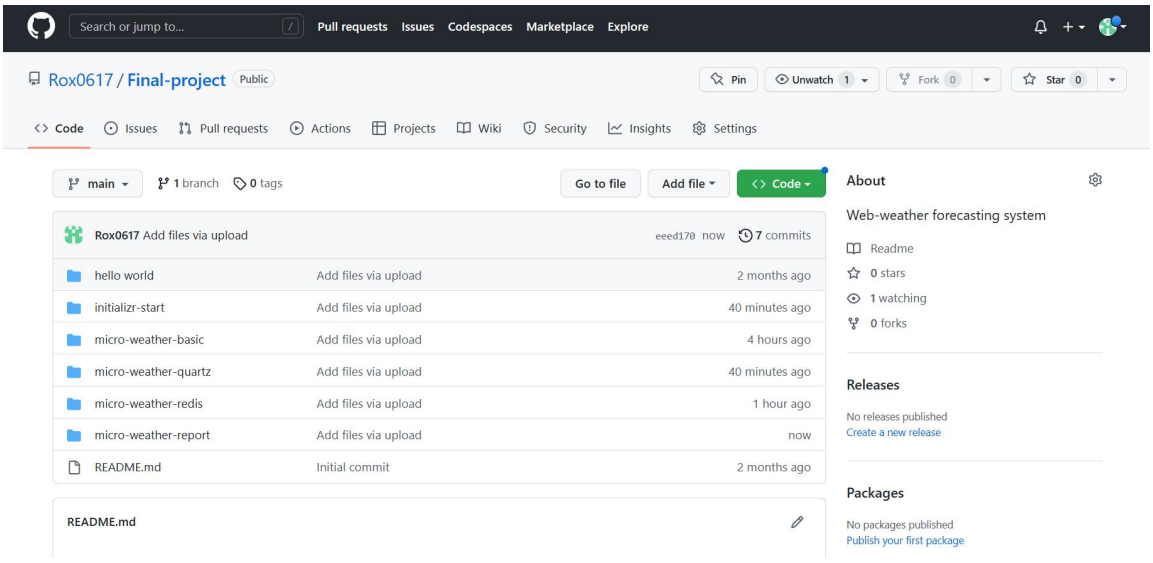
(Figure24: Project-gantt Full flow)

4.3 Data management plan

My understanding of Spring Boot was followed by a Gradle environment build and modification of the Hello World starter project.



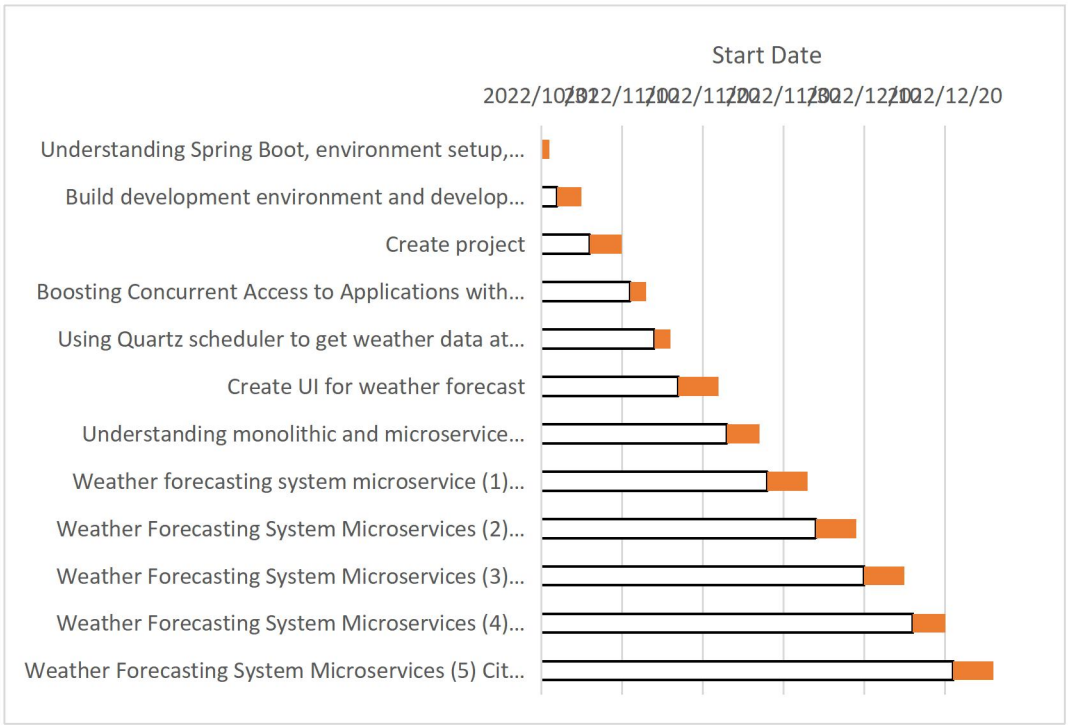
(Figure25: The local file)



(Figure26: The github file:https://github.com/Rox0617/Final-project)

4.4 Deliverables

The task 1 to 12 in this semester



(Figure27: Chart)

5 References

ABCDEFGHIJKLMNOPQRSTUVWXYZ

1. H. Zhao, Y. Jiang and X. Zhao. (2020). "Design and research of University intelligent education cloud platform based on Dubbo microservice framework," 2020 5th International Conference on Mechanical, Control and Computer Engineering (ICMCCE), 2020, pp. 870-874, doi: 10.1109/ICMCCE51767.2020.00191.
2. K. P. Selvam and M. Brorsson, "Performance Modeling of Weather Forecast Machine Learning for Efficient HPC," 2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS), 2022, pp. 1268-1269, doi: 10.1109/ICDCS54860.2022.00127.
3. M. R. Mufid, A. Basofi, M. U. H. Al Rasyid, I. F. Rochimansyah and A. rokhim, "Design an MVC Model using Python for Flask Framework Development," 2019 International Electronics Symposium (IES), 2019, pp. 214-219, doi: 10.1109/ELECSYM.2019.8901656.
4. M. R. Mufid, A. Basofi, M. U. H. Al Rasyid, I. F. Rochimansyah and A. rokhim, "Design an MVC Model using Python for Flask Framework Development," 2019 International Electronics Symposium (IES), 2019, pp. 214-219, doi: 10.1109/ELECSYM.2019.8901656.
5. R. Ramachandran. et al.(2021) "Health Weather App," 2021 2nd Global Conference for Advancement in Technology (GCAT), 2021, pp. 1-6, doi: 10.1109/GCAT52182.2021.9587785.
6. Shivhare, N., Rahul, A. K., Dwivedi, S. B., & Dikshit, P. K. S. (2019). “ARIMA based daily weather forecasting tool: A case study for Varanasi”, Mausam, 70(1), 133–140. https://doi.org/10.54302/mausam.v70i1.179.