# **WEATHER SERVICE PROJECT REPORT**

## **PROJECT DESCRIPTION**

The Weather Service Application has two components:

### **Data collector**

It populates the datastore with all the valid zip codes in Texas. The zip codes in the datastore are then used as an input for calling the Yahoo weather API to retrieve weather data (low temp, high temp) for the input zip code and the results are populated into the data store.

### **Web service**

This service exposes two REST APIs and interacts with the data collector to retrieve the weather for a location. The first API accepts a zip code parameter and returns the weather information corresponding to that location. The second API accepts an array of zip codes and returns the weather data corresponding to all those zip codes.

## **IMPLEMENTATION**

Following have been used for implementation:

* Languages & Frameworks : Java, Spring Boot, Jersey, Maven
* Database : MySQL
* Server : Apache Tomcat

### **Data Collector**

This application uses the Spring JDBC data source initializer feature to load the MySQL database with zip codes corresponding to Texas state. The schema for creating the tables and initializing the database are added to the schema.sql and the data.sql files respectively.

The Tomcat JDBC connection pool which is the default data source used by Spring is replaced with HikariCp for better performance and its properties are configured in the application.properties file

Data collector populates the database with the weather information through the following steps:

1. The zip codes that were previously stored into the database are collected into a list.
2. The zip codes are then processed concurrently by using Multi-threading. Thread count is made configurable.
3. Each thread from the thread pool does the following:
4. Calls the Yahoo weather API and gets the response data
5. If the response status is success, then the response entity, which is a JSON string, is parsed and mapped to the WeatherData object using the Jackson library.
6. The zip code of the WeatherData object is then set and stored in the MySQL database using the Spring JDBC Template.

### **Web Service**

Two RESTful APIs are built using Jersey to support Web Service. Web Service uses the Basic Auth protocol for authorizing the users to access the resources. For the demo purpose, the username and the password are configured using the properties file.

* Failure to authenticate results in ‘401 unauthorized’ status.
* Passing null parameters results in ‘400 bad request’ status
* Passing an invalid zip code results in ‘404 request not found’ status

### **Integration**

Data Collector and Web Service are integrated using the Rmi infrastructure provided by Spring. The Data Collector application exposes one of its interfaces transparently by configuring the RmiServiceExporter to serve the weather data requests from the Web Service component. Web Service uses the RmiProxyFactory bean to communicate with the Data collector. All the Rmi related configurations are made configurable.

## **TESTING**

Following steps can be used to execute and test the project:

1. Download the attached zipcodes.txt file to your machine
2. Configure the data.sql schema file to change the path for the zip codes file to the location on your machine.
3. Create the database ‘weatherinfo’ in the MySQL Database
4. Go to the folder containing the pom.xml file for the Data Collector application and start the application using the following command:

mvn spring-boot:run

1. Build the Web Service application
2. Deploy the war file to the servlet container
3. Run the server

Following are the usage guidelines for accessing the REST APIs

1. API that returns weather info for a single zip code:

* Return value: JSON response containing the weather info for the given zip code
* Resource url: [http://host:port/weatherinfo/v1/forecast/{zipcode}](http://host:port/weatherinfo/v1/forecast/%7bzipcode%7d)
* Method type: GET
* Authentication: Required (For the demo purpose, username = user, password = password)
* Path parameter: zip code

Example: <http://localhost:8083/v1/forecast/73301>  
Response: {  
            "high": 30,  
            "low": 16,  
            "zipCode": "73301",  
            "Weather Condition": "Sunny"

}

1. API that returns weather info for multiple locations.

* Return value: JSON array containing weather info for multiple locations based on their zip codes
* Resource url: <http://host:port/weatherinfo/v1/forecast/zipcode=value&zipcode=value>
* Method type: GET
* Authentication: Required

Example: <http://localhost:8083/weatherinfo/v1/forecast?zipcode=73301&zipcode=75124>  
Response: {  
        [  
            {  
                "high": 30,  
                "low": 16,  
                "zipCode": "73301",  
                "Weather Condition": "Sunny"  
            },  
            {  
                "high": 28,  
                "low": 16,  
                "zipCode": "75124",  
                "Weather Condition": "Sunny"  
            }  
        ]  
     }

## **POTENTIAL IMPROVEMENTS**

Looking ahead, following could improve various aspects of the application

* Due to lack of sufficient documentation regarding the Yahoo two-legged OAuth APIs, it could not be implemented, but given the sufficient documentation, this would improve the security.
* Retry logic could be implemented on the client side to handle failure requests.
* A scheduler could be implemented in the Data Collector service to fetch the weather data daily.
* TTL of the response could be handled.
* Pagination could be implemented for the multiple location Web Service to improve performance.
* Integration with logger to make the application production ready.