

User Guide for Weather Report APIs

1. Download Source Code

The source code can be downloaded from GitHub repository via this link:

<https://github.com/tnphung/weather-app>

2. Run The Application

Once the source code is downloaded it can be run straight away. But before running the application, please be sure that the JDK 17 is already installed.

Execute the following commands to start the application Weather App.

```
C:\temp\Weather App\target>java -jar weather-app-1.0.jar
```

The result should look like so:

```

  2024-09-20T11:15:27.235+10:00 INFO 6320 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 4099 ms
  2024-09-20T11:15:27.319+10:00 INFO 6320 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
  2024-09-20T11:15:27.930+10:00 INFO 6320 --- [main] com.zaxxer.hikari.pool.HikariPool : HikariPool-1 - Added connection conn0: url=jdbc:h2:mem:testdb user=SA
  2024-09-20T11:15:27.954+10:00 INFO 6320 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
  2024-09-20T11:15:27.969+10:00 INFO 6320 --- [main] o.s.b.a.h2.H2ConsoleAutoConfiguration : H2 console available at '/h2-console', Database available at 'jdbc:h2:mem:testdb'
  2024-09-20T11:15:28.309+10:00 INFO 6320 --- [main] o.hibernate.jpa.internal.util.LogHelper : HHH000204: Processing PersistenceUnitInfo [name: default]
  2024-09-20T11:15:28.541+10:00 INFO 6320 --- [main] org.hibernate.Version : HHH000041: Hibernate ORM core version 6.5.2.Final
  2024-09-20T11:15:28.668+10:00 INFO 6320 --- [main] o.h.c.internal.RegionFactoryInitiator : HHH000026: Second-level cache disabled
  2024-09-20T11:15:29.525+10:00 INFO 6320 --- [main] o.s.o.j.n.SpringPersistenceUnitInfo : No LoadTimeWeaver setup; ignoring JPA class transformer
  2024-09-20T11:15:31.151+10:00 INFO 6320 --- [main] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000489: No JTA platform available (set 'hibernate.transaction.jta.platform' to enable JTA platform integration)
  2024-09-20T11:15:31.154+10:00 INFO 6320 --- [main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
  2024-09-20T11:15:32.876+10:00 WARN 6320 --- [main] jpa.hbm3Configuration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during view rendering. Explicitly configure spring.jpa.open-in-view to disable this warning
  2024-09-20T11:15:34.762+10:00 INFO 6320 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 5000 (http) with context path '/'
  2024-09-20T11:15:34.868+10:00 INFO 6320 --- [main] weatherapi.WeatherApiApplication : Started WeatherApiApplication in 13.047 seconds (process running for 14.307)

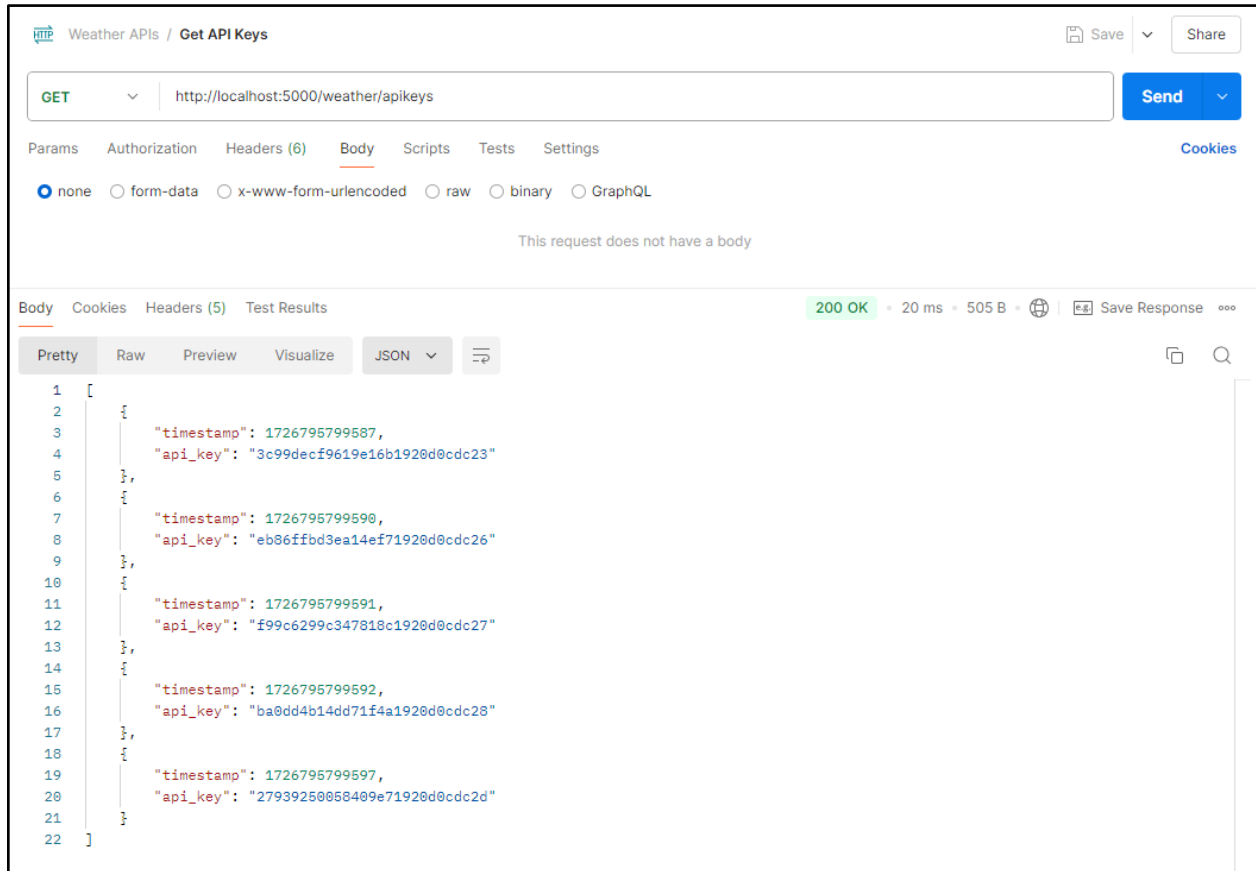
```

Please note that the embedded Tomcat run on port 5000.

3. Calling Weather Report APIs

It is recommended to use the tool Postman to call the APIs. Below are illustrations of how to send a GET request to a list of five API keys and weather report description.

The GET request end point to get API keys:**<http://localhost:5000/weather/apikeys>**



http://localhost:5000/weather/report?q=paris,france&apiKey=3c99decf9619e16b1920d0cdc23

Weather APIs / Get Weather Report

SaveSend

GET

http://localhost:5000/weather/report?q=paris,france&apiKey=3c99decf9619e16b1920d0cdc23

Send

Params

Authorization

Headers (6)

Body

Scripts

Tests

Settings

Cookies

Query Params

<input checked="" type="checkbox"/> Key	Value	Description	⋮ Bulk Edit
<input checked="" type="checkbox"/> q	paris,france		
<input checked="" type="checkbox"/> apiKey	3c99decf9619e16b1920d0cdc23		
<input type="checkbox"/> Key	Value	Description	

Body

Cookies

Headers (5)

Test Results

200 OK

15 ms

191 B

🌐

📄 Save Response

⋮

Pretty

Raw

Preview

Visualize

JSON

⌵

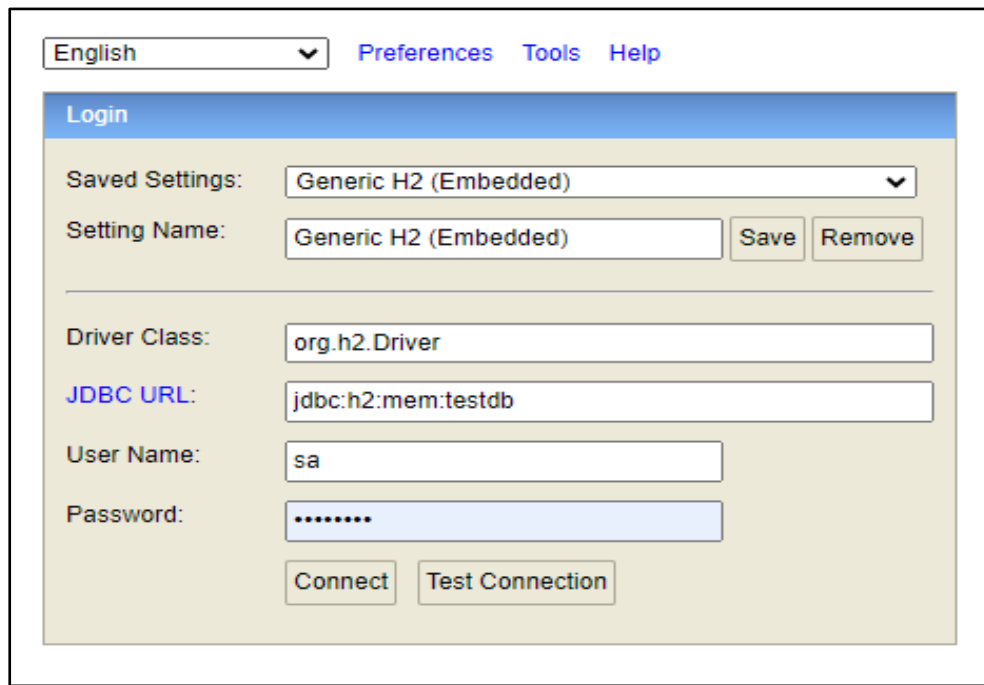
⌵

```
1 {
2   |   "description": "clear sky"
3 }
```

4. H2 Database

When a user gets a weather report for the first time, the report data is stored the H2 database. To inspect the weather reports, please login to the local H2 database by following these steps:

- Enter this URL on a browser: <http://localhost:5000/h2-console>
- Enter the details in the login pop-up window as shown below:
The password is “password”.



The screenshot shows the H2 Database console interface. At the top, there is a language dropdown menu set to 'English' and three links: 'Preferences', 'Tools', and 'Help'. Below this is a 'Login' section with a blue header. Inside the login section, there are several fields and buttons:

- Saved Settings:** A dropdown menu showing 'Generic H2 (Embedded)'.
- Setting Name:** A text input field containing 'Generic H2 (Embedded)', followed by 'Save' and 'Remove' buttons.
- Driver Class:** A text input field containing 'org.h2.Driver'.
- JDBC URL:** A text input field containing 'jdbc:h2:mem:testdb'.
- User Name:** A text input field containing 'sa'.
- Password:** A text input field with masked characters (dots).
- At the bottom of the login section are two buttons: 'Connect' and 'Test Connection'.

Once logged in, execute some “SELECT” statements as shown below:

The screenshot shows a database client interface with a sidebar on the left displaying the database structure: jdbc:h2:mem:testdb, API_KEY, COUNTRY, WEATHER_REPORT_DETAILS, INFORMATION_SCHEMA, Users, and H2 2.1.214 (2022-06-13). The main area contains a SQL statement editor with two queries. The first query, 'SELECT * FROM API_KEY;', is executed, resulting in a table with 5 rows and 3 columns: API_KEY, TIMESTAMP, and NUMBER_OF_TIME_USED. The second query, 'SELECT * FROM WEATHER_REPORT_DETAILS;', is also executed, resulting in a table with 1 row and 5 columns: ID, CITY, COUNTRY, DESCRIPTION, and TIMESTAMP.

Run Run Selected Auto complete Clear SQL statement:

```
SELECT * FROM API_KEY ;
```

```
SELECT * FROM WEATHER_REPORT_DETAILS |
```

```
SELECT * FROM API_KEY ;
```

API_KEY	TIMESTAMP	NUMBER_OF_TIME_USED
f7a12491104ccca21920d151bac	1726796340140	1
937be0c768b882571920d151bae	1726796340142	0
fe3bcd3b6492a47d1920d151bb0	1726796340144	0
e9c46dda3045224f1920d151bb4	1726796340148	0
445bed030e4f577a1920d151bb5	1726796340149	0

(5 rows, 1 ms)

```
SELECT * FROM WEATHER_REPORT_DETAILS;
```

ID	CITY	COUNTRY	DESCRIPTION	TIMESTAMP
1	Paris	France	clear sky	1726795695515

(1 row, 1 ms)

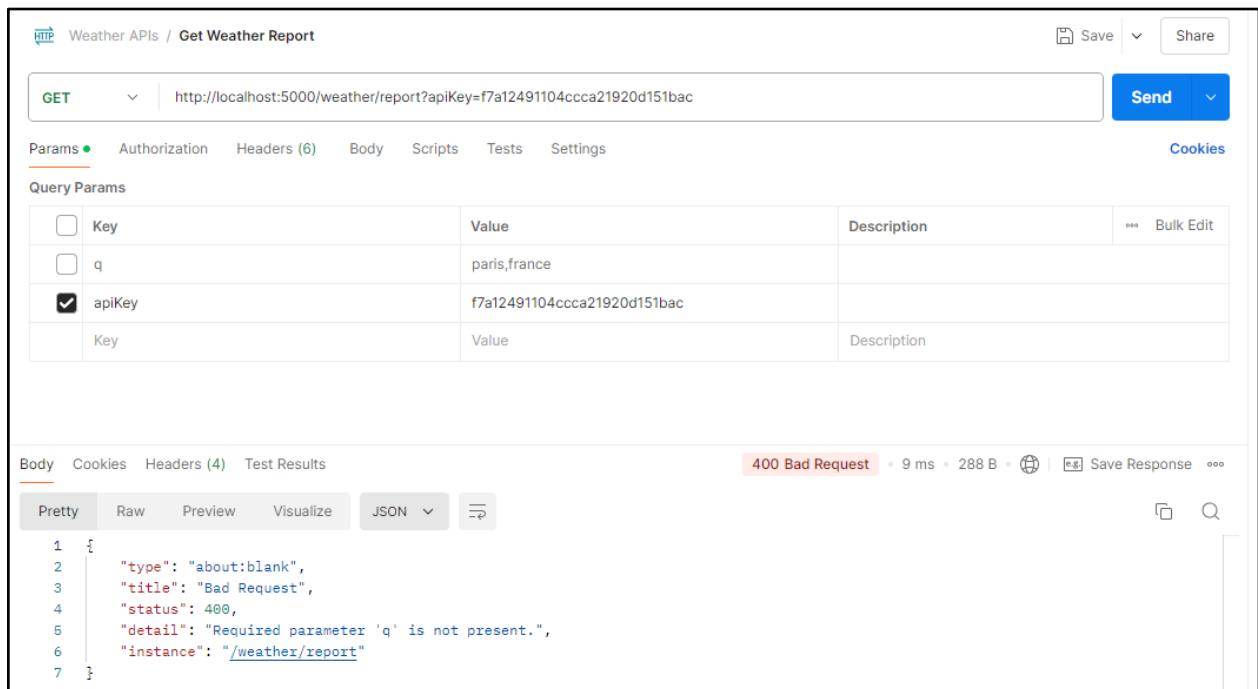
The above image shows a table of five API keys and a weather report for city Paris in France.

5. Testing

This section shows some “unhappy path” testings on the Weather Report API.

5.1 Testing Parameter “q”

The API will return the following error when the parameter “q” does not exist in the URL



The screenshot shows a REST client interface for a "Weather APIs / Get Weather Report" endpoint. The request method is GET, and the URL is `http://localhost:5000/weather/report?apiKey=f7a12491104ccca21920d151bac`. The "Query Params" section shows a table with the following data:

Key	Value	Description
q	paris,france	
apiKey	f7a12491104ccca21920d151bac	

The response is a 400 Bad Request, returned in 9 ms with a body size of 288 B. The response body is shown in JSON format:

```
1 {
2   "type": "about:blank",
3   "title": "Bad Request",
4   "status": 400,
5   "detail": "Required parameter 'q' is not present.",
6   "instance": "/weather/report"
7 }
```

5.2 Testing Parameter “apiKey”

The API will return the following error when the parameter “apiKey” does not exist in the URL

The screenshot shows a REST client interface for a "Weather APIs / Get Weather Report" endpoint. The method is GET and the URL is `http://localhost:5000/weather/report?q=paris,france`. The "Query Params" section contains a table with the following data:

Key	Value	Description
q	paris,france	
apiKey	f7a12491104ccca21920d151bac	

The response status is "400 Bad Request" with a 6 ms duration and 293 B size. The response body is shown in JSON format:

```
1 {
2   "type": "about:blank",
3   "title": "Bad Request",
4   "status": 400,
5   "detail": "Required parameter 'apiKey' is not present.",
6   "instance": "/weather/report"
7 }
```

5.3 Testing Query

The following error will be returned when the country name is missing from the URL.

The screenshot shows the same REST client interface, but the URL is `http://localhost:5000/weather/report?q=paris&apiKey=f7a12491104ccca21920d151bac`. The "Query Params" section contains a table with the following data:

Key	Value	Description
q	paris	
apiKey	f7a12491104ccca21920d151bac	

The response status is "400 Bad Request" with a 7 ms duration and 282 B size. The response body is shown in JSON format:

```
1 {
2   "type": "about:blank",
3   "title": "Bad Request",
4   "status": 400,
5   "detail": "Missing city and/or country name",
6   "instance": "/weather/report"
7 }
```

The same error will be returned when the city name is missing from the URL:

The screenshot shows a REST client interface for a 'Weather APIs / Get Weather Report' endpoint. The method is GET and the URL is `http://localhost:5000/weather/report?q=united states&apiKey=f7a12491104ccca21920d151bac`. The 'Query Params' section shows a table with three parameters: 'Key', 'q' (value: 'united states'), and 'apiKey' (value: 'f7a12491104ccca21920d151bac'). The 'Body' tab is selected, showing a JSON response with a 400 status and a message: 'Missing city and/or country name'. The response is as follows:

```
1 {
2   "type": "about:blank",
3   "title": "Bad Request",
4   "status": 400,
5   "detail": "Missing city and/or country name",
6   "instance": "/weather/report"
7 }
```

And the same error will be displayed when the query is empty.

This screenshot is identical to the previous one, but the URL is `http://localhost:5000/weather/report?q=&apiKey=f7a12491104ccca21920d151bac`, where the query parameter 'q' is empty. The 'Query Params' table shows 'q' with an empty value. The JSON response in the 'Body' tab is the same: a 400 status with the message 'Missing city and/or country name'.

Extra tests are welcome for this API.