**Create a component , which can render the weather of a city.End user should be able to enter the city name in the component and click submit, system should display the weather information of that city .**

**CASE-1:**

**AJAX CODE FOR CALLING THE SERVLET**

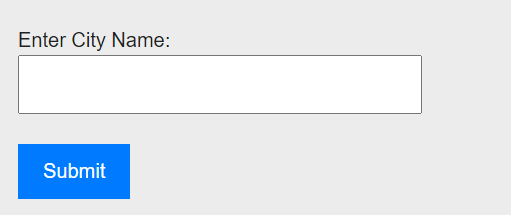
**<form id="cityForm" action="/bin/submit\_city1232" method="post">**

**<label for="cityInput">Enter City Name:</label><br>**

**<input type="text" id="cityInput" name="cityName" required><br><br>**

**<input type="submit" id="submitBtn" value="Submit">**

**</form>**

****

**CASE-2:**

ServletResolverConstants.***SLING\_SERVLET\_PATHS*** + "=/bin/submit\_city1232"

**It triggers the servlet when we click the submit button on the page**

**CASE-3:**

**public** **class** WeatherServlet **extends** SlingAllMethodsServlet

**<form id="cityForm" action="/bin/submit\_city1232" method="post">**

**<label for="cityInput">Enter City Name:</label><br>**

**<input type="text" id="cityInput" name="cityName" required><br><br>**

**<input type="submit" id="submitBtn" value="Submit">**

**</form>**

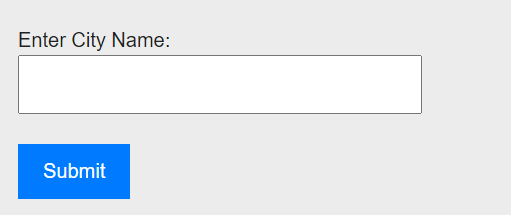
**SlingAllMethodsServlet used for post request**

**Form Submissions:**

**When a user submits a form on a website, such as a registration form, login form, or contact form, the data entered by the user is typically sent to the server via a POST request.**

**CASE-4:**

String city = request.getParameter("cityName");

****

**When we enter the city name in frontend , Backend takes the cityname using getParameter(“cityName”) method.**

**CASE-5:**

WeatherData weatherData = weatherService.getWeatherData(city);

WeatherData weatherData:

We declares the variable here weatherData and return type we mentioned as a class name WeatherData

weatherService.getWeatherData(city);

**Here** weatherService is a service class name , In the weatherService class it calls the “getWeatherData(city)” method.

**public** WeatherData getWeatherData(String city) **throws** IOException

****

**CASE-6:**

String description = "Sunny";

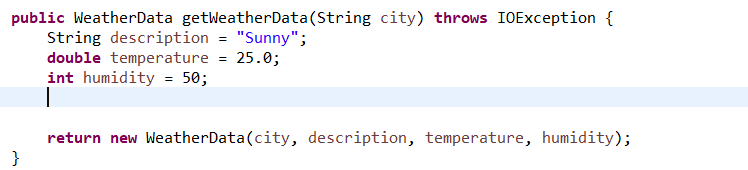
**double** temperature = 25.0;

**int** humidity = 50;

**We are giving the city name in the front end based on that it needs to give the weather details using api key , If api key not working It returns the default values what we have defined in the code .**

**CASE-6a:**

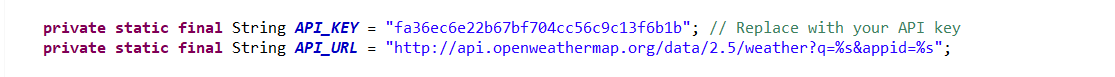
**For suppose api key is not working, The Process happening like this**

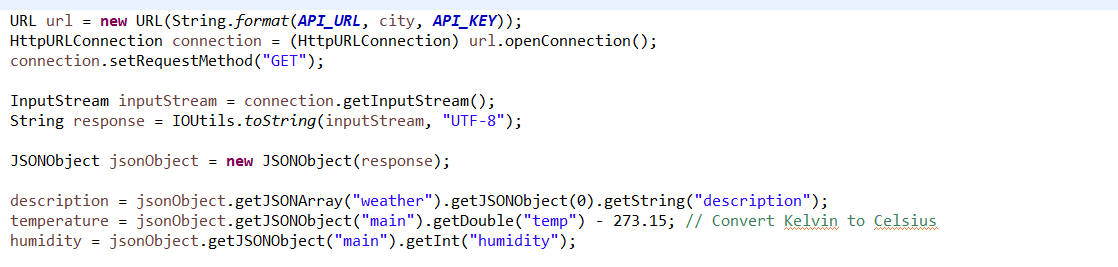
****

**When “**getWeatherData(city)” method triggers it checks the code in the method ,there is no api code, It process the default values .

**CASE-7:**

**If api code there it can take the weather from here**

****

****

**URL url = new URL(String.format(API\_URL, city, API\_KEY));**

1. **URL: This is a class in Java that represents a Uniform Resource Locator, which is a pointer to a resource on the World Wide Web. It is typically used to open a connection to the resource specified by the URL.**
2. **String.format(API\_URL, city, API\_KEY): This is using the String.format() method to format a string with dynamic values. It looks like API\_URL is a string containing a placeholder (%s), which will be replaced by the values of city and API\_KEY.**
3. **API\_URL: This seems to be a constant or a variable holding the base URL of an API. It likely contains a placeholder for the city name and another for the API key.**
4. **city and API\_KEY: These are variables containing the values to be substituted into the placeholders in the API\_URL string.**

**HttpURLConnection connection = (HttpURLConnection) url.openConnection();**

**This line of code is also written in Java and is used to establish a connection to the URL previously constructed.**

1. **HttpURLConnection: This is a class in Java that represents a communication link between the URL and the application. It is used to send and receive data between the application and the URL resource.**
2. **url.openConnection(): This method is called on the URL object (url) that was previously created. It returns a URLConnection object that represents the connection to the URL. Since url.openConnection() returns a URLConnection object and not specifically an HttpURLConnection object, it needs to be cast to HttpURLConnection in order to use methods specific to HTTP connections.**

**InputStream inputStream = connection.getInputStream();**

**This line of code is obtaining the input stream from the HttpURLConnection object, which represents the stream of data coming from the URL connection**

1. **InputStream: This is a class in Java that represents an input stream of bytes. It's used for reading data from various sources, such as files, network connections, or other streams.**
2. **connection.getInputStream(): This method is called on the HttpURLConnection object (connection) to retrieve the input stream associated with the connection. This input stream allows you to read data from the URL.**

**So, this line of code is creating an InputStream object named inputStream by obtaining the input stream from the HttpURLConnection object. This input stream can then be used to read data from the URL connection.**

**JSONObject jsonObject = new JSONObject(response);**

**This line of code appears to be using the JSONObject class from a JSON parsing library to parse a JSON-formatted response obtained from the URL connection**

1. **JSONObject: This likely refers to a class from a JSON parsing library in Java, such as org.json.JSONObject. This class is used to represent a JSON object, allowing you to easily work with JSON data in Java.**
2. **response: This seems to be a string variable containing the response received from the URL connection. Presumably, this response is in JSON format.**

**So, this line of code is creating a JSONObject named jsonObject by parsing the JSON-formatted response string response. Once parsed, you can access the data within the JSON object using methods provided by the JSONObject class.**

**Top of Form**