REST Applications - In-house Development and External Use

REST: An Architectural Style for the Modern Web

REST, short for "**Representational State Transfer**," is an architectural style that has evolved over the years from a simple concept into a cornerstone of modern web development. The focus is on **machine-to-machine communication**. Text messages are exchanged using the **HTTP protocol**. In a RESTful architecture, every resource to be accessed is identified by a unique address (**URI – Uniform Resource Identifier**). The REST paradigm evolved from the HTTP Object Model designed by Roy Fielding in 1994.

We will use the Apache2 web server to integrate a simple web service. The request is made via an **HTTP Request**, and the web service returns a **JSON object**. JSON is very easy for a custom program to read and interpret, but other formats, such as XML, can also be used.

Temperature Conversion Application: Celsius <--> Fahrenheit

We will create a **PHP REST application** to convert a temperature from degrees Celsius (°C) to Fahrenheit (°F) or from Fahrenheit to Celsius.

To do this, you will use the following URL: `./tempUmrechner.php?value=xxx.xx&unit=celsius`.

We use the following formulas: $\text{°F} = \text{°C} \times 1.8 + 32$ or $\text{°C} = (\text{°F} - 32) / 1.8$.

Implementation

Create a folder named **"rest"** in the web directory. Create a file named `tempUmrechner.php` in the new directory and add the following code:

```php
<?php
    $unit = "";
    $value = "";

    // Define function to return a JSON error
    function sendError($statusCode, $message) {
        header('Content-Type: application/json');
        http_response_code($statusCode);
        $error_result = array("status" => "error", "message" => $message);
        echo json_encode($error_result);
        exit;
    }

    // 1. Check for required parameters
    if (!isset($_GET['value']) || !isset($_GET['unit'])) {
        sendError(400, "Missing parameters. 'value' and 'unit' are required.");
    }
```

```
    // 2. Check for valid value
    $value = $_GET['value'];
    if (!is_numeric($value)) {
        sendError(400, "The parameter 'value' must be numeric.");
    }

    // 3. Check for valid unit
    $unit = strtolower($_GET['unit']); // Convert unit to lowercase for robust
checking
    if ($unit !== 'celsius' && $unit !== 'fahrenheit') {
        sendError(400, "Invalid value for 'unit'. Allowed are 'celsius' or
'fahrenheit'.");
    }

    if($unit == "celsius"){
        header('Content-Type: application/json');
        http_response_code(200);
        $result = array(
            "unit" => "fahrenheit",
            "value" => ($value * 1.8 + 32)
        );
        echo json_encode($result);
        exit;

    }elseif($unit == "fahrenheit"){
        header('Content-Type: application/json');
        http_response_code(200);
        $result = array(
            "unit" => "celsius",
            "value" => (($value - 32) / 1.8)
        );
        echo json_encode($result);
        exit;
    }
?>
```

Note that we return `'Content-Type: application/json'` in the header and convert the return array into a JSON object.

Now you can test the application. To do this, call the application in the browser: `http://<ip-webcontainer>/rest/tempUmrechner.php?value=20&unit=celsius`. We are querying the conversion of 20°Celsius to Fahrenheit.

The response is:

```
{
    "unit":"fahrenheit",
    "value":68
}
```

REST Client

A **REST client** can be written in any language. The client only needs to adhere to the form of the request (HTTP or HTTPS) and have knowledge of the response structure.

> **Note:** You can execute these examples on your client PC.

Java Example

```
package va_rest;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;

public class TempUmrechnerApiCaller {
    public static void main(String[] args) {
        try {
            // Replace with the actual URL of your API
            String urlString = "http://<ip-webcontainer>/rest/tempUmrechner.php?
value=20&unit=celsius";
            URL url = new URL(urlString);
            HttpURLConnection connection =
(HttpURLConnection)url.openConnection();

            BufferedReader in = new BufferedReader(
                    new InputStreamReader(connection.getInputStream()));

            int responseCode = connection.getResponseCode();
            System.out.println("Response Code: " + responseCode);

            String inputLine;
            StringBuffer content = new StringBuffer();
            while ((inputLine = in.readLine()) != null) {
                content.append(inputLine);

            }
            in.close();


            // Here you can parse the JSON string and process the result
            System.out.println(content.toString());

        } catch (IOException e) {
            System.out.println(e);
        }
    }
}
```

To work with JSON in Java, you need a separate library; I use GSON from Google.

Python Example

```
import requests
import json
from pprint import pprint

# ------------------------------------------------------------
# Configuration
# ------------------------------------------------------------

# NOTE: Replace <ip-webcontainer> with the actual IP address of your LXC container
BASE_URL = "http://<ip-webcontainer>/rest/tempUmrechner.php"


def get_temperature_conversion(value, unit):
    """
    Calls the REST service for temperature conversion.

    :param value: The temperature value to be converted (float or int).
    :param unit: The source unit ('celsius' or 'fahrenheit').
    :return: The converted result as a dictionary or an error message.
    """

    # Define URL parameters
    params = {
        "value": value,
        "unit": unit
    }

    print(f"Sending Request: {BASE_URL}?value={value}&unit={unit}")

    try:
        # Execute the GET request
        response = requests.get(BASE_URL, params=params)

        # 1. Check the HTTP Status Code
        if response.status_code == 200:
            print("Status: OK (200)")
            # Read and return the JSON result
            return response.json()

        elif response.status_code == 400:
            # Bad Request, e.g., invalid unit or value
            print(f"Status: Bad Request ({response.status_code})")

            # The API returns an explanatory JSON body in case of error
            try:
                error_data = response.json()
                print("API Error Message:")
```

```
                # Use pprint for clean output of the error
                pprint(error_data)
                return None
            except json.JSONDecodeError:
                print("API error could not be read as JSON.")
                return None

        else:
            # Unexpected Status Code (e.g., 500 Server Error)
            print(f"Unexpected Status: {response.status_code}")
            return None

    except requests.exceptions.ConnectionError:
        print("ERROR: Could not establish connection to the server. Check
IP/Port.")
        return None
    except requests.exceptions.RequestException as e:
        print(f"An error occurred: {e}")
        return None


# ------------------------------------------------------------
# Examples
# ------------------------------------------------------------

if __name__ == "__main__":

    print("--- 1. Successful Call (20°C to F) ---")
    result_ok = get_temperature_conversion(value=20, unit="celsius")
    if result_ok:
        print(f"Result: {result_ok['value']} {result_ok['unit'].upper()}")

    print("\n" + "="*50 + "\n")

    print("--- 2. Successful Call (68°F to C) ---")
    result_ok_reverse = get_temperature_conversion(value=68, unit="fahrenheit")
    if result_ok_reverse:
        print(f"Result: {result_ok_reverse['value']}
{result_ok_reverse['unit'].upper()}")

    print("\n" + "="*50 + "\n")

    print("--- 3. Failed Call (Invalid Value) ---")
    # The value "abc" is not numeric. This should trigger 400 Bad Request.
    get_temperature_conversion(value="abc", unit="celsius")

    print("\n" + "="*50 + "\n")

    print("--- 4. Failed Call (Invalid Unit) ---")
    # The unit is unknown. This should trigger 400 Bad Request.
    get_temperature_conversion(value=10, unit="kelvin")
```

Conclusion

With **REST Web Services**, you have a powerful tool that works with a simple protocol (HTTP). You can define your own services, use database data for responses, or exchange data.

Crucially, programs can work **independently** of each other—essential for applications using a **Microservice Architecture**. This is an architectural approach where a large application is broken down into a collection of small, loosely coupled, and independently deployable services.
