JavaScript - REST API's and User Data on the Client Side

Eli the Computer Guy - December 5, 2024
Version 1

Note:
We use Template Literals for concatenation in these examples. You open a Template Literal with a ` (back tick) not a ' (single or double quotation mark), and add the variable values using ${variable}.

`Hello ${name}`;

Note 2:
These projects can be written and run from your computer with no need for a web server. Simply make sure they have the HTML extension, and then open the script with a web browser.

Console

The console allows you to view information that you may not want printed out in HTML. You generally click on "view source" in a browser and then navigate to console to view messages.

navigator Values

By calling navigator properties you can access information about the system that the user is using to view your web page.

- userAgent: navigator.userAgent
- language: navigator.language
- platform: navigator.platform
- online: navigator.onLine
- memory: navigator.deviceMemory
- cores: navigator.hardwareConcurrency

browser-info.html

```
<div id="display"></div>

<script>
    const display = document.getElementById('display');
    display.innerHTML = `User Agent -- ${navigator.userAgent}<br>
                         Language -- ${navigator.language}<br>
                         Platform -- ${navigator.platform}<br>
                         CPU's -- ${navigator.hardwareConcurrency}`;
</script>
```

screen / window Values

You can access how large a users screen is and how big the browser window is as the user views your site.

```
<div id="display"></div>

<script>
    const display = document.getElementById('display');
    display.innerHTML =
        `<p>Total Screen Size: ${screen.width} x ${screen.height}</p>
        <p>Available Screen Size: ${screen.availWidth} x $
{screen.availHeight}</p>
        <p>Viewport Size: ${window.innerWidth} x ${window.innerHeight}</
p>`;
</script>
```

REST API

fetch() function is used to get the response from a web page like **requests.get()** in Python.

When the value come back you assign a variable name for the **.then** step to check if the response was OK, and then format the data appropriately as either **text()** or **json()**.

The next **.then** step allows you to assign a name to the variable value and then do something with it such as print out to the HTML page. (Instead of "data" you could call the variable "bob")

The final step is **.catch** where if there is an error it is instructions of what to do such as print out to the console.

=> is a shortcut to send variable values to a function.

Text Response

This example finds your external IP Address from the ipify.org REST API when you request a text response. You will only receive your IP Address with no additional data.

rest-ip-html.html

```
<div id="display"></div>

<script>
    const display = document.getElementById('display');

    fetch('https://api.ipify.org')
    .then((response) => {
        if (!response.ok) {
            throw new Error(`HTTP error! Status: ${response.status}`);
        }
        return response.text();
    })
    .then((data) => {
        display.innerHTML = `<h1>IP: ${data}</h1>`
    })
    .catch((error) => {
        console.error('Error fetching data:', error);
    });
</script>
```

JSON Response

When you are making a request for a JSON response you need to use **.json()** for the data to be accessible as JSON.

JSON.stringify(variable, null, indent) turns your JSON value into formatted text like prettify with **json.dump()** in Python.

Wrap the response with in **<pre>** tags to display the JSON as formatted text with appropriate indents within HTML.

To access JSON values you simply call them by using the variable name, period, and then the index name. So when we receive the IP Address from ipify.org as JSON we can get the value by calling **data.ip** .

rest-ip-json.html

```
<div id="display"></div>

<script>
    const display = document.getElementById('display');

    fetch('https://api.ipify.org?format=json')
    .then((response) => {
        if (!response.ok) {
            throw new Error(`HTTP error! Status: ${response.status}`);
        }
        return response.json();
    })
    .then((data) => {
        display.innerHTML = `<h1>IP: ${data.ip}</h1>
                             <pre>${JSON.stringify(data, null, 2)}</pre>`
    })
    .catch((error) => {
        console.error('Error fetching data:', error);
    });
</script>
```

This is an example using a joke API. You receive a JSON response with multiple values. We will print out the prettified version of the JSON, and then print out the Setup and Punchline of the Joke.

https://www.postman.com/cs-demo/public-rest-apis/request/64pqonf/random-joke

rest-joke.html

```
<div id="display"></div>

<script>
    const display = document.getElementById('display');

    fetch('https://official-joke-api.appspot.com/random_joke')
    .then((response) => {
        if (!response.ok) {
            throw new Error(`HTTP error! Status: ${response.status}`);
        }
        return response.json();
    })
    .then((data) => {
        display.innerHTML = `${JSON.stringify(data, null, 2)}<br>
        Setup: ${data.setup}<br>
        Punchline: ${data.punchline}<br>`
    })
    .catch((error) => {
        console.error('Error fetching data:', error);
    });
</script>
```

Multilevel JSON

In this example we access a REST API of Cat Facts. It responds with 5 facts to your request.

The response is enclosed within square brackets [] with means those indexes are numbered. Then individual facts are enclosed with curly brackets {} which means they are accessed with a name.

So the third cat fact would be accessed with **data[2].text** . (With indexes numbering starts at 0)

https://www.postman.com/cs-demo/public-rest-apis/request/69ieesi/cat-facts

rest-cat.html

```
<div id="display"></div>

<script>
    const display = document.getElementById('display');

    fetch('https://cat-fact.herokuapp.com/facts/')
    .then((response) => {
        if (!response.ok) {
            throw new Error(`HTTP error! Status: ${response.status}`);
        }
        return response.json();
    })
    .then((data) => {
        display.innerHTML = `<pre>${JSON.stringify(data, null, 2)}
<pre><br>
        <h1>${data[2].text}</h1>`
    })
    .catch((error) => {
        console.error('Error fetching data:', error);
    });
</script>
```

Iterating Through JSON Records

In Javascript you use the map() function to loop through an array to pull out values of different records.

map() creates an array of the results so we use the **join()** function to join the array as a single string of text.

rest-cat-map.html

```
<div id="display"></div>

<script>
    const display = document.getElementById('display');

    fetch('https://cat-fact.herokuapp.com/facts/')
    .then((response) => {
        if (!response.ok) {
            throw new Error(`HTTP error! Status: ${response.status}`);
        }
        return response.json();
    })
    .then((data) => {
        const factsHtml = data.map((fact) => {
            return `<p>${fact.text} -- ${fact.createdAt}</p>`;
        }).join('');

        display.innerHTML = factsHtml;
    })
    .catch((error) => {
        console.error('Error fetching data:', error);
    });
</script>
```

Sending Data From Javascript

You can collect data about a users web browser environment and then send that data to your server.

Cross Site Scripting

XSS is when a script on one server tries to communicate with another server. Browsers stop many of these communications be default. You need to either ask permission to send data, or find a work around.

We will be using fetch() with GET values to be able to send data.

The HTML page does not need to be hosted on a server. You can simply save it to your desktop and open it from there.

Sending a GET with fetch()

we use **fetch()** to send data to a server using GET. We grab information about the user agent with **navigator.userAgent**, and the screen size of the machine.

send-data.html

```
<h1>Getting Your Data App</h1>

<script>
    fetch(`http://localhost:8080/receive?userAgent=${navigator.userAgent}
&width=${screen.width}&height=${screen.height}`)
</script>
```

This is a simple Bottle Web App script that will receive the data from the javascript.

```
python3 -m pip install bottle
```

Make sure the script is running and then when you open/ refresh the send-data.html web page the GET values will be printed in the terminal

bottle-receive.py

```
from bottle import request, run, get

@get('/receive')
def receive_post():
    print(f'data -- {request}')
    print()
    print(request.query.get('userAgent'))
    print(request.query.get('width'))
    print(request.query.get('height'))

run(host='localhost', port=8080)
```

Sending Supplemental Data from REST API's

In this example we use the ip-api.com REST API to find the city of the user, and then we send that value along with the user agent to the web server.

NOTE: From a security standpoint imagine you visit a site while using a VPN to hide your location. You close your VPN, but don't close the web page. If it is refreshed or the Javascript runs then your real location will be sent to the server.

spy-script.html

```
<h1>Spy Script</h1>
<div id="display"></div>

<script>
    const display = document.getElementById('display');

    fetch('http://ip-api.com/json/')
    .then((response) => {
        if (!response.ok) {
            throw new Error(`HTTP error! Status: ${response.status}`);
        }
        return response.json();
    })
    .then((data) => {
        console.log(data);
        display.innerHTML = `
        User Agent: ${navigator.userAgent}<br>
        City: ${data.city}<br>`

        fetch(`http://localhost:8080/receive?userAgent=$
{navigator.userAgent}&city=${data.city}`)
    })
    .catch((error) => {
        console.error('Error fetching IP data:', error);
    });
</script>
```

This bottle script will print out the user agent, and the city that is submitted from the HTML web page.

bottle-receive-rest.py

```
from bottle import request, run, get

@get('/receive')
def receive_post():
    print(f'data -- {request}')
    print()
    print(request.query.get('userAgent'))
    print(request.query.get('city'))

run(host='localhost', port=8080)
```