HTTP requests

HTTP stands for Hypertext Transfer Protocol and is used to structure requests and responses over the internet. HTTP requires data to be transferred from one point to another over the network.

The transfer of resources happens using TCP (Transmission Control Protocol). TCP manages the channels between your browser and the server. TCP is used to manage many types of internet connections in which one computer or device wants to send something to another. HTTP is the command language that the devices on both sides of the connection must follow in order to communicate.

HTTP and TCP

When you type an address into your browser, you are commanding it to open a TCP channel to the server that responds to that URL (or Uniform Resource Locator). A URL is like your home address or phone number because it describes how to reach you.

In this situation, your computer, which is making the request, is called the client. The URL you are requesting is the address that belongs to the server. Once the TCP connection is established, the client sends a HTTP *GET* request to the server to retrieve the webpage it should display. After the server has sent the response, it closes the TCP connection. If you open the website in your browser again, or if your browser automatically requests something from the server, a new connection is opened which follows the same process described above. GET requests are one kind of HTTP method a client can call.

The GET request

Suppose you want to check out the latest course offerings from http://tec.com. After you type the URL into your browser, your browser will extract the **`http`** part and recognize that it is the name of the network protocol to use. Then, it takes the domain name from the URL, in this case “tec.com”, and asks the internet Domain Name Server to return an Internet Protocol (IP) address.

Now the client knows the destination’s IP address. It then opens a connection to the server at that address, using the **`http`** protocol as specified. It will initiate a GET request to the server which contains the IP address of the host and optionally a data payload. The GET request contains the following text:

```
GET / HTTP/1.1
Host: www.tec.com
```

The GET request

This identifies the type of request, the path on www.tec.com (in this case, “/“) and the protocol “HTTP/1.1.”

The second line of the request contains the address of the server which is `"www.tec.com"`. If the server is able to locate the path requested, the server might respond with the header:

```
HTTP/1.1 200 OK
Content-Type: text/html
```

This header is followed by the content requested, which in this case is the information needed to render www.tec.com.

The first line of the header, **`HTTP/1.1 200 OK`**, is confirmation that the server understands the protocol that the client wants to communicate with (**`HTTP/1.1`**), and an HTTP status code signifying that the resource was found on the server. The second line, **`Content-Type: text/html`**, shows the type of content that it will be sending to the client.

If the server is not able to locate the path requested by the client, it will respond with the header:

```
HTTP/1.1 404 NOT FOUND
```

Introduction to requests

There are many types of HTTP requests. The four most commonly used types of HTTP requests are GET, POST, PUT, and DELETE.

JavaScript uses an event loop to handle asynchronous function calls. When a program is run, function calls are made and added to a stack. The functions that make requests that need to wait for servers to respond then get sent to a separate queue. Once the stack has cleared, then the functions in the queue are executed.

Web developers use the event loop to create a smoother browsing experience by deciding when to call functions and how to handle asynchronous events.

GET requests using fetch

The `fetch()` function:

- Creates a request object that contains relevant information that an API needs.
- Sends that request object to the API endpoint provided.
- Returns a promise that ultimately resolves to a response object, which contains the status of the promise with information the API sent back.

```
fetch('http://api-to-call.com/endpoint').then(response => {
  if (response.ok) {
    return response.json();
  }
  throw new Error('Request failed!');
}, networkError => console.log(networkError.message)
).then(jsonResponse => {
    // Code to execute with jsonResponse
});
```

sends request

converts response object to JSON

handles errors

handles success

Activity

POST requests using fetch

Take a look at the diagram to the right. It has the boilerplate code for a POST request using `fetch()`.

Notice that the `fetch()` call takes two arguments: an endpoint and an object that contains information needed for the POST request.

The object passed to the `fetch()` function as its second argument contains two properties: `method`, with a value of `'POST'`, and `body`, with a value of `JSON.stringify({id: '200'});`. This second argument determines that this request is a POST request and what information will be sent to the API.

A successful POST request will return a response body, which will vary depending on how the API is set up.

Activity