Fetch & CORS

Summary: in this tutorial, you will explore CORS (Cross-Origin Resource Sharing) and how to configure the webserver to enable CORS for an origin to fetch data.

What is origin

An origin refers to a unique combination of three components:

- **Protocol (scheme)**: The protocol specified in the URL such as `HTTP`, `HTTPS`, `FTPS`, etc.
- **Domain (host)**: This is the domain name or IP address such as `javascripttutorial.net` or `127.0.0.1`
- **Port**: This is the port number used to communicate with the webserver. If you don't specify a port, the default port for the protocol will be used e.g., port `443` for `HTTPS`, port `80` for `HTTP`.

For example, `https://www.javascripttutorial.net/api/` is an origin where:

- `HTTPS` is the protocol.
- `www.javascriptttorial.net` is the domain name.
- `443` is the default port for `HTTPS` protocol.

The `https://api.javascripttutorial.net/` is a different origin because the domain name `api.javascripttutorial.net` is different from the domain name `www.javascripttutorial.net`.

The origin is important because the web browser uses it to enforce security policy which is known as CORS.

What is CORS

CORS (Cross-Origin Resource Sharing) is a security feature of web browsers, which prevents an origin from making unauthorized requests to another origin.

The browser allows a site to make HTTP requests to its own origin. But, if that site attempts to request a different origin (cross-origin), the browser blocks this action by default to protect against security risks like cross-site scripting (XSS) attacks.

Let's take a look at an example to have a better understanding of how CORS works.

Setting a simple web server

We'll set up a simple Express web server in Node.js.

Step 1. Create a new directory to store project files:

```
mkdir webserver
cd webserver
```

Step 2. Init the project by running the `npm init` command on your terminal:

```
npm init --yes
```

This command will create the package.json file in the project directory.

Step 3. Install `express` package:

```
npm install express
```

Step 4. Configure the `package.json` by adding the `type : "module"` :

```
"type": "module"
```

This will allow you to use ES6 modules in your Node.js project.

And also modify the `script` section:

```
"scripts": {
    "start": "node index.js"
```

```
}
```

By modifying the `scripts` section, you can execute the `npm start` command in your terminal to run the `index.js` file:

Step 5. Create a new file called `index.js` with the following code:

```
import express from 'express';
const app = express();

app.get('/', (req, res) => {
  res.send('Hello, World!');
});

app.listen(3000, () => {
  console.log('Server is running on port 3000');
});
```

In `index.js` file, we create an Express web server that accepts a `GET` request to the route `http://localhost:3000/` and returns the `'Hello, World!'` message.

Step 6. Start the webserver by running the following command in your terminal:

```
npm start
```

Step 7. Open `http://localhost:3000/` on the web browser, you'll see the JSON response:

```
{
  "message": "Hello, World!"
}
```

Creating a JavaScript app

We'll create a simple JavaScript app that will call the API endpoint `http://localhost:3000/` using the fetch() method.

Step 1. Create a new directory to store the JavaScript app files:

```
mkdir app
cd app
```

Step 2. Create an `index.html` file:

```
<!DOCTYPE html>
<html lang="en">

    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Fetch API - CORS</title>
    </head>

    <body>
        <button id="btn">Fetch</button>
        <p id="message"></p>
        <script src="app.js"></script>
    </body>

</html>
```

The `index.html` file includes the `app.js` within the same directory.

Step 3. Create an `app.js` file in the project directory:

```
const btn = document.getElementById('btn');
const messageElem = document.getElementById('message');

btn.addEventListener('click', async () => {
  // reset the message
  messageElem.innerHTML = '';
  try {
    // call the API
    const response = await fetch('http://localhost:3000/');
```

```
    const data = await response.json();

    // update the message
    messageElem.innerHTML = data.message;
  } catch (err) {
    messageElem.innerHTML = err.message;
  }
});
```

Step 4. Open the `index.html` (using a live-server extension on VS code) and click the Fetch button, you'll see the message:

```
Failed to fetch
```

When you open the console window, you'll see the following error message:

```
Access to fetch at 'http://localhost:3000/' from origin 'http://127.0.0.1:5501' has been bloc
```

The reason is that we are making an HTTP request from the origin `http://127.0.0.1:5501` to the server at the origin `http://localhost:3000/` and the browser blocks it.

Enable CORS on the Web Server

Step 1. Open the terminal and install `cors` package in the Node.js project:

```
npm install cors
```

Step 2. Modify the `index.js` file to enable CORS from all origins:

```
import express from 'express';
import cors from 'cors';

const app = express();

app.use(cors());
```

```
app.get('/', (req, res) => {
  res.send({ message: 'Hello, World!' });
});

app.listen(3000, () => {
  console.log('Server is running on port 3000');
});
```

How it works.

First, import `cors` function from the `cors` package:

```
import cors from 'cors';
```

Second, call the `cors()` function and pass it returns value to the `app.use` () method:

```
app.use(cors());
```

This line of code instructs the Express web server to add the following entry to the header of the HTTP responses, which allows the API endpoint to be called from any origin:

```
Access-Control-Allow-Origin: *
```

The asterisk (`*`) means any origin.

Step 3. Restart the web server by stopping it (press Ctrl-C) and starting it again:

```
npm start
```

Step 4. Click the Fetch button on the JavaScript app. You'll see the `Hello, World!` message on the web browser:

```
Hello, World!
```

If you examine the header of the HTTP response, you'll see the following entry:

```
Access-Control-Allow-Origin: *
```

Note that you can configure the webserver to allow CORS from a specific origin such as `http://127.0.0.1:5501` by adding it to the `cors()` function:

```
// ...

app.use(
  cors({
    origin: 'http://127.0.0.1:5501',
  })
);
//  ...
```

> Note that the origin has no trailing slash (`/`) because the origin sent by the browser may not include it.

In this case, the web server will add the following entry in the header of the HTTP response:

```
Access-Control-Allow-Origin: http://127.0.0.1:5501/
```

Download the project source code

Click here to download the project source code

When you extract the zip file, you'll see two directories:

- webserver
- app

To start the web server, you need to:

First, navigate to the `webserver` directory:

```
cd webserver
```

Second, run the command `npm install` to install dependencies:

```
npm install
```

Third, start the web server:

```
npm start
```

Summary

- An origin is defined by a combination of three components: domain, protocol, and port.
- CORS (Cross-Origin Resource Sharing) is a security feature built into web browsers that, by default, prevents an origin from making requests to a different origin.
- To allow requests from any origin, configure your web server to include the header `Access-Control-Allow-Origin: *` in its HTTP responses.