Requests in JavaScript

Overview

Topics

- URLs and Requests Overviews
- Synchronous Code, Asynchronous Code, and Promises
- JSON
- Axios

Let's Review URLs

What is a URL?

URL stands for `Uniform Resource Locator`.

The URL is a mixture of the protocol, domain, and endpoint being requested.

https://devmountain.com/about.html

Above is an example of a `URL`.

- Protocol - https://
- Domain - devmountain.com
- Endpoint - /about.html

So this URL is making an HTTP request to the devmountain.com domain (which represents an IP address) and is asking for an HTML document using the specified endpoint.

HTTP(S)

HTTP stands for Hyper Text Transfer Protocol.

The (S) stands for secure. If a web sites url is prefixed with HTTPS then it will usually have a green lock next to it meaning that this site is secure to send information to it over the interwebs.

This is the protocol that we follow when making a network or an http request.

We do this by making an http request to a URL prefixed with http://.

Query

The query of the request is extra information that we can apply to the URL. This is usually used to find specific values.

Queries start with a ? then use a key value pair.

Example: http://example.com/people/?name=cameron

Above, I'm looking for a user named 'cameron'

The server should be set up to handle the query, if it isn't, we'll get an error.

Params

The params are extra parts of our URL to send data. It works like a query almost, but we don't need a ? or a key/value pair.

We just prefix the param with a backslash.

Example: http://example.com/people/riley

Above, 'riley' is the param.

The server should be set up to handle the parameter, if it isn't, we'll get an error.

Parts Of A Request

Header

Similar to the ***head*** of an HTML document, the header is the part of the request that holds information about the request we are making. This could be status codes, content type, when the request was made, etc.

We used a header in our ***curl*** commands when we learned about how the web works.

Body

The body is an optional part of the request. This is where we will store the data that we want to send through the request.

The body does not always have to have information inside of it. It's okay to keep it empty if need be.

A good example of this is if we have a form on our webpage and we hit the submit button, it then will perform a request and send the information that we typed into the form through the body of the request.

In JavaScript, we'll send an object for the body.

JSON

What is JSON?

`JSON` is the format that we can use to structure our data that is being sent in the request.

JSON is short for `JavaScript Object Notation`. This is how we will transfer information between different languages.

JSON looks very similar to a JavaScript object, but the key/value pairs are wrapped in double quotes, except for numbers.

```
{
  "name": "jacob",
  "age": 22,
  "hobbies": ["snowboarding", "video games", "cars"],
  "car": {
    "make": "subaru",
    "year": 2014
  }
}
```

Notice how we can still send arrays and objects inside of JSON.

We also can *not* have a trailing comma on our object.

Every request that we send will be parsed into JSON so that the server can understand it.

How Code Runs

Synchronous

JavaScript is a synchronous language. This means that it can only have one thing happen at a time.

This causes a problem for us when we need to make an HTTP request to a server because requests can sometimes take a large amount of time to receieve a response. Since JavaScript is synchronous, it will just make the request then think it's done. Meaning that when we receive a response from the request it will not do anything with that response.

So we need to make our code run asynchronously.

Asynchronous

Asynchronous JavaScript will allow us to make a request then execute the rest of our code, then once we receive a response from our request, it will handle it.

This makes it so our application can still run and perform correctly.

Promises

A promise is a special object in JavaScript that will hold a response object from an http request. We can use them to handle asynchronous actions.

Putting it All Together

We'll be using a package called axios to make server requests. Axios helps us make HTTP requests using URLs and communicates with the server using JSON. It works asynchronously and returns promises. We'll be making axios calls in our JavaScript file(s) on the front end – these requests are what connect the front and back ends of an application. Then we'll use the DOM manipulation we learned to change the page based on the response.

Axios

Installing Axios

To use axios, we first need to install `axios` from the package manager.

In your terminal, make sure that you are in the current project directory, then run:

`npm install axios`

This will install the library into your project so we have access to the built-in methods needed.

When you're starting with someone else's code, running `npm install` will go and find a list of needed packages from the `package.json` file and install them.

Import Axios

In your HTML file, add a script tag. We'll set the ***src*** attribute of this script tag to be the main axios file.

This script tag should come before your own JavaScript files.

```
<script src="./node_modules/axios/dist/axios.min.js"></script>
```

This will bring it into our code so we can use it.

Using Axios

Axios returns a `promise` in JavaScript. Remember: A promise is a special object in JavaScript that will hold a response object from an http request.

This allows us to asynchronously handle JavaScript code.

We can then use this `promise` object to handle data that comes back from the response. If it is a successful promise, it will resolve and if it resolves we will use a `.then()` method.

The `.then()` method accepts a callback function as argument. The `response` param in the callback is where we have access to the object returned from the response.

The data sent back from the server will be stored on the `data` property of the response.

```
axios.get('http://www.example.com/people')
.then(response => {
  console.log(response.data)
})
```

If there is a failure somewhere along the way, we can handle it by using a `.catch()`. We chain this method onto the `.then()` so we can "catch" the error.

```
axios.get('http://www.example.com/people')
  .then(response => {
    console.log(response.data)
  })
  .catch(error => {
    console.log(error)
  })
```

Axios Methods

There are four methods that we will be covering from the `axios` library.

GET - This method will receive a url as an argument to make a `get request` to. It will return some data.

```
axios.get('http://website.com/api/')
  .then(response => {
    //code that does something with response.data
  })
  .catch(error => {
    console.log(error)
  })
```

POST - This method uses two arguments, the url to hit and an object for the body of the request. It creates data in the backend and returns a response.

```
axios.post('http://website.com/api/', { name: 'eric' })
  .then(response => {
    //code that does something with response.data
  })
  .catch(error => {
    console.log(error)
  })
```

PUT - This method uses two arguments, the url to hit and an object for the body of the request. It edits data in the backend and returns a response.

```
axios.put('http://website.com/api/3', { name: 'nitin' })
  .then(response => {
    //code that does something with response.data
  })
  .catch(error => {
    console.log(error)
  })
```

DELETE - This method will receive the url to hit as an argument. It deletes data in the backend and returns a response.

```
axios.delete('http://website.com/people/2')
  .then(response => {
    //code that does something with response.data
  })
  .catch(error => {
    console.log(error)
  })
```

Final Overview

Client-Server Model