Talking to a server

Long Course: Week 5

INSTRUCTOR
Daniel Omajali

1. WHAT IS HTTP?

HTTP stands for Hyper Text Transfer Protocol and it is responsible for communication between web servers & clients. Anytime you submit a form data you are using HTTP and you are going through the REQUEST & RESPONSE cycle (You make a request and you get a response back that has something called the header and the body).

Data sent can be stored in either a database, cookie, session, API etc

Data can be retrieved from databases, API, sessions etc

2. WHAT IS HTTPS?

HTTPS stands for Hyper Text Transfer Protocol Secure and it's basically where all the data sent back and forth is secured by something called SSL or TLS which stands for Secure Socket Layer or Transport Security Layer respectively.

Anytime you have users that are sending sensitive information, it should always be over HTTPS.

3. HTTP METHODS

When a request is made to a server, it has some kind of method attached to it which include

GET

This request is used when you want to get or fetch data from the server.

- Everytime you visit a webpage you are also making a GET request to the server via HTTP.

- Anytime an image or an asset loads, it is making a get request.

We can inspect the GET request method or any other methods from the Network tab of chrome dev tool which is accessed through inspecting element of any webpage.

3. HTTP METHODS

POST

This request is used when you are posting data such as when you submit a form, when you submit a blog post etc. You are sending data to the server which can be stored in the database, API or other components.

PUT

This request is used to update data that is already on the server.

DELETE

This request is used to delete data that is already on the server.

QUESTION

What are the various HTTP methods, what are their functions and where are they applicable?

ASSIGNMENT

LEARN ABOUT HTTP STATUS CODE

Possible Questions:
What does the following status code represent?

200, 201, 301, 304, 400, 401, 404, 500

4. ASYNCHRONOUS JAVASCRIPT

JavaScript is a single-threaded programming language which means only one thing can happen at a time. That is, the JavaScript engine can only process one statement at a time in a single thread.

While the single-threaded languages simplify writing code because you don’t have to worry about the concurrency issues, this also means you can’t perform long operations such as network access without blocking the main thread.

4. ASYNCHRONOUS JAVASCRIPT

Imagine requesting some data from an API. Depending upon the situation, the server might take some time to process the request while blocking the main thread making the web page unresponsive.

That's where **asynchronous** JavaScript comes into play. Using asynchronous JavaScript (such as callbacks, promises, and async/await), you can perform long network requests without blocking the main thread thus making web pages responsive.

5. ILLUSTRATION OF SYNCHRONOUS TASK EXECUTION

Let’s suppose we are doing an image processing or a network request in a synchronous way:

```
const processImage = (image) => {
  //doing some operations on image
  console.log('Image processed');
}
const networkRequest = (url) => {
  //requesting network resource
  return someData;
}
const greeting = () => {
  console.log('Hello World');
}
processImage(logo.jpg);
networkRequest('www.somerandomurl.com');
greeting();
```

5. ILLUSTRATION OF SYNCHRONOUS TASK EXECUTION

Doing image processing and network request takes time. So when processImage() function is called, it’s going to take some time depending on the size of the image.

When the processImage() function completes, it’s removed from the stack. After that the networkRequest() function is called and pushed to the stack. Again it’s also going to take some time to finish execution.

5. ILLUSTRATION OF SYNCHRONOUS TASK EXECUTION

At last when the networkRequest() function completes, greeting() function is called and since it contains only a console.log statement and console.log statements are generally fast, so the greeting() function is immediately executed and returned.

So you see, we have to wait until the function (such as processImage() or networkRequest()) has finished. This means these functions are blocking the call stack or main thread. So we can’t perform any other operation while the above code is executing which is not ideal.

6. SO WHAT'S THE SOLUTION?

The solution is introducing asynchronous way of doing task execution. And one of the way is PROMISES

QUESTION

What is the difference between SYNCHRONOUS task execution and ASYNCHRONOUS task execution?

7. PROMISES?

Promises is a way of doing asynchronous tasks (Tasks that can be executed on the thread while other process/tasks is running).

We use Promises when working with ajax requests, talking with servers or any async task that requires an action after it has been complete.

Examples of promised based HTTP client include Axios & fetch()

8. MAKING CALLS TO A SERVER/TALKING TO SERVERS (IN JAVASCRIPT)

We can make calls or make HTTP requests to a server to send or retrieve data using the following ways

AJAX

AXIOS

FETCH ()

JQUERY

FORM ACTION (HTML FORM TO SUBMIT)

For the purpose of this lesson, we will dwell on

FORM ACTION

AXIOS

FETCH ()

9. FORM ACTION

CREATING A FORM TO SUBMIT USING HTML

A form is a way of submitting data on a website. Forms tend to use POST Requests.

- Through form action, we can decide the action of our form and determine how our data is treated, sent or stored.

- Through form method, we can decide if it is a POST method.

ILLUSTRATION

Let's see how we can post data to a server using form action

Refer to code (form-action.html)

10. AXIOS

Axios is a promise based HTTP client for the browser and Node.js environment.

For any project where we need to make calls to a server or database to send data and retrieve data we can use Axios to make calls to a server.

11. SETTING UP AXIOS

We can set up axios in our project by either using the NPM or CDN. In this lesson we will be using the CDN to illustrate.

By using the CDN, we'll attach the following script file to our html page (Within the body or the head section):

```
<script
src="https://unpkg.com/axios/dist/axios.min.js"></script>
```

12. GET & POST REQUEST USING AXIOS

Refer to code (axios3.html)

13. FETCH () API REQUEST

Fetch API is also another way of making a call to a server

Refer to code

14. DISPLAYING DATA FROM SERVER IN OUR HTML ELEMENT

Refer to code (bonsai.html and bonsai-order.html)

15. ASYNC AWAIT

Async await is a simplified way of returning a promise either using axios or fetch ().

Async function always returns a promise.

Recall that when using Axios, when our code is successful it is logged into the **.then(res…** while when it is not successful it is logged into the **.then(err…**

However, when doing axios using async await, it is simplified to log the success in our **try** method and failure in the **catch**

16. GET REQUEST USING ASYNC AWAIT & AXIOS

Refer to code (bonsai.js)

ASSIGNMENT

USING AXIOS:

- Do a get request from the API route link (https://tbhpwebdevapi.azurewebsites.net/api/Bonsai)

- Do a post request and post a new item using same API link

- Do a delete request and delete an item with the id of 55 from the API route link

USING FORM ACTION:

- Do a post request and post a new to-do list to this API route link (https://jsonplaceholder.typicode.com/todos)

Tip: Visit link to see the various properties

DISPLAYING SERVER DATA:

- Do a get request and display a blog post with the index number of 30 from this API route (https://jsonplaceholder.typicode.com/posts)

(https://jsonplaceholder.typicode.com/users)