URL Parameters, Fetching Data, And Parsing JSON

Software Development Bootcamp

Topic

URL Parameters

What Are URL Parameters?

URL parameters, also known as query string parameters, are a way to pass data to a web server through the URL. They appear after a **question mark (?)** in a URL and are separated by **ampersands (&)**.

Structure Of URL Parameters

https://example.com/page?param1=value1&param2=value2

- The question mark (?) separates the URL from the parameters
- Each parameter is a key-value pair (e.g., **param1=value1**)
- Multiple parameters are separated by ampersands (&)

Why Use URL Parameters?

- Passing data between web pages
- Filtering or sorting content on a page
- Tracking information in analytics
- Customizing page content based on user input

Important Considerations

- URL parameters are visible in the address bar, so don't use them for sensitive information
- Some characters need to be encoded in URL parameters (e.g., spaces become %20)
- The order of parameters doesn't matter in most cases
- Parameter names are case-sensitive

Accessing URL Parameters

- Use `document.location.search` to get the URL's query string
- Parse the query string with `URLSearchParams`

```
// Example URL
"https://example.com/page?param1=value1&para
m2=value2"
const url = document.location.search
const searchParams = new
URLSearchParams(url);
const param2Value =
searchParams.get('param2')
console.log(param2Value)
```

Topic

Fetch API

Fetching Data

Fetch API is a way for your browser to get data from other places on the internet.

- Send a request: Ask for some information from another website
- Receive a response: That website sends you back the information you asked for

What Is An API?

API stands for Application Programming Interface.

- It's a set of rules and protocols that allow different software applications to communicate with each other.
- APIs define the methods and data structures that applications can use to request and exchange information.
- In the context of web development, APIs often refer to web services that allow you to retrieve or send data to a server.

What Is The Fetch API?

The Fetch API is a modern interface for making HTTP requests in web browsers.

- It provides a more powerful and flexible feature set than older HTTP request methods like XMLHttpRequest.
- The Fetch API uses Promises, which enables a simpler and cleaner API to compose asynchronous operations.
- It's built into modern browsers, so you don't need to load any external libraries to use it.
- The Fetch API can handle various types of data, including JSON, which is commonly used in web APIs.

Why is Fetch useful?

- **Dynamic updates**: Lets web pages update content without reloading (ex. updating a news feed)
- **Get and Send Data**: You can get data from other websites, and also send data to other websites (like submitting a form)
- **Improved User Experience**: By allowing asynchronous data fetching, it helps create smoother, more responsive web applications

Fetch and Async / Await

- Imagine sending a letter to a friend, you have to wait for their reply before you can continue the conversation.
- In programming we often have to wait for certain actions to finish before moving on to the next step.
- Fetch helps us request data from and send data to a server, but this can take time.
    - **async**: "Hey this function might need to wait for something, so let's prepare for that!"
    - **await**: "Hold on a sec, let's wait here until we get a reply"

Fetching Remote Data

```
// Making the fetch request
let response = await
fetch(`https://jsonplaceholder.typicode.com/posts/1`)
// Can be processed as text
let text = await response.text()
// OR can be processed as JSON (much more common)
let json = await response.json()
```

Fetching Local Data

```
// Making the fetch request using relative URL for data source
let response = await
fetch(`/some-blog-post.md`)
// Can be processed as text
let text = await response.text()
```

Fetch Example

Catching Errors

- Any time there is a chance of an error we want to catch it
- To do this we set up a **`try catch`** block

```
async function getPost() {
   // set up try catch block
   try {
       // fetch request
       const response = await
fetch(`https://jsonplaceholder.typicode.com/posts/NO-P
OST-AVAILABLE-HERE`)
       // wait for the response and convert it to JSON
       const data = await response.json()
       // show the data in the console
       console.log(data)
   } catch (error) {
       // if something goes wrong in the try block we
fall into the catch block and log the error to the
console
       console.log(error)
   }
}
```

Fetch and Form Submission

- Simple HTML form with Name, Email and a submit button

index.html

```
<body>
   <form id="myForm">
       <label for="name">Name:</label>
       <input type="text" id="name" name="name"
required>
       <br>
       <label for="email">Email:</label>
       <input type="email" id="email" name="email"
required>
       <br>
       <button type="submit">Submit</button>
   </form>

   <script src="index.js"></script>
</body>
```

```
// add the submit event listener to the form with
submitForm as the callback
document.getElementById("myForm").addEventListener("subm
it", submitForm);
// submitForm function defined as async
async function submitForm(event) {
 // prevent default submission behavior
 event.preventDefault();
 // access the form element
 const form = event.target;
 // collect the form data
 const formData = {
   name: form.name.value,
   email: form.email.value,
 };
```

```
 // set up try catch block
 try {
   // use fetch to send a POST request with the form data
   const response = await
fetch("https://example.com/submit-form", {
     method: "POST",
     headers: {"Content-Type": "application/json"},
     // convert the form data to JSON
     body: JSON.stringify(formData),
   });
   if (!response.ok) {
     throw new Error("Network response was not ok");
   }
   // wait for fetch request to complete and the response to
be converted to JSON
   const data = await response.json();
   // data is logged to the console if fetch is successful
   console.log("Success:", data);
   // catch block for any errors
 } catch (error) {
   console.error("Error:", error);
 }
}
```

Fetch and Form Submission

Topic

JSON

What Is JSON?

JSON stands for JavaScript Object Notation. It's a simple way to store and send data. Think of JSON like a container for information.

- It's easy for people to read and write
- Computers can understand it quickly
- Many different programming languages can use it

Structure Of JSON

- A collection of name/value pairs (similar to an object in JavaScript)
- Uses curly braces {} to define objects and square brackets [] to define arrays
- Name/value pairs are separated by commas
- Names are strings, values can be
    - String, Number, Boolean, Object, Array, Null

JSON Example

- This is a JSON object returned from a fetch request to the Rick and Morty API


```
{
 "id": 393,
 "name": "Roy",
 "status": "Alive",
 "species": "Human",
 "type": "Game",
 "gender": "Male",
 "origin": {
   "name": "Roy: A Life Well Lived",
   "url": "https://rickandmortyapi.com/api/location/32"
 },
 "location": {
   "name": "Roy: A Life Well Lived",
   "url": "https://rickandmortyapi.com/api/location/32"
 },
 "image":
"https://rickandmortyapi.com/api/character/avatar/393.jpeg"
,
 "episode": ["https://rickandmortyapi.com/api/episode/13"],
 "url": "https://rickandmortyapi.com/api/character/393",
 "created": "2018-01-20T19:15:27.239Z"
}
```

Why Use JSON?

- It doesn't take up much space, so it's fast to send
- People can read it easily
- Computers can work with it quickly
- Different systems can share data using JSON

JSON Limitations

- It can't include functions or complex data types
- You can't add comments to explain the data
- It doesn't have a special way to write dates (you use text instead)

Parsing JSON

- **Parsing JSON**: Converting a JSON string into a JavaScript object
    - Use `JSON.parse()` to convert a JSON string in a JavaScript object

```
// Parsing JSON
let jsonString = '{"data": "some data goes here", "number": 42}'
let parsedObject = JSON.parse(jsonString)
console.log(parsedObject.data)
// Output: some data goes here
console.log(parsedObject.number)
// Output: 42
```

Producing JSON

- **Producing JSON**: Converting a JavaScript object into a JSON string
    - Use `JSON.stringify()` to convert a JavaScript object into a JSON string

```
// Producing JSON
let jsObject = { name: "John", age: 30,
city: "New York" }
let jsonString = JSON.stringify(jsObject)
console.log(jsonString)
// Output:
{"name":"John","age":30,"city":"New York"}
```

Fetch API & JSON

- The Fetch API automatically parses JSON responses when you use `response.json()`

```
// Using with Fetch API (with async/await)
async function fetchData() {
 try {
   const response = await
fetch('https://api.example.com/data')
   const data = await response.json()
   console.log(data)
 } catch (error) {
   console.error('Error:', error)
 }
}
fetchData()
```

Accessing JSON Objects

- Once you have a JSON object in JavaScript (after parsing it from a string), you can easily access its data using dot notation or bracket notation

```
// Using with Fetch API (with async/await)
async function fetchData() {
 try {
   const response = await
fetch('https://rickandmortyapi.com/api/location/
32')
   // creating a variable 'data' to hold the
parsed response
   const data = await response.json()
   // access the data object
   console.log(data.name)
   console.log(data.residents)
 } catch (error) {
   console.error('Error:', error)
 }
}
fetchData()
```

Exercise

Rick And Morty