**Intro to Fetch API**

Hello, in this video I will be introducing the topic of

the Fetch API and

will be giving an overview on what will be covered in this module.

The Fetch API is an interface built into

the browser that allows users to make network requests.

Great, So why should you use it?

Well, it is a much needed improvement over the XML HTTP

request API, which was the previous method for making that work request.

The XML HTTP request API required a lot of boilerplate code,

which made it very hard to follow.

Developers often used third party libraries such as jQuery

to wrap the XML HTTP request API in a much more usable way.

However, with the fetch API,

developers now have an easy to use method for

making network requests that doesn't require any external libraries.

So how easy is it to use the Fetch API?

Well, in its simplest form, all you need to use is the Fetch method and

plug in the URL endpoint of your network request, and that's it.

The Fetch method will return a promise that will hold a fetch

response, and from that response you can obtain the data that you

are trying to receive.

See?

Easy to use.

In this module, I will be covering how to make a basic fetch call, how

to send data with your fetch call, and how to handle fetch responses.

### ****Introduction to the Fetch API****

#### What is the Fetch API?

The Fetch API is an interface that is used to make network requests.

#### Why is the Fetch API important?

The Fetch API is a much needed improvement over XMLHttpRequest, the old API for making network request. The Fetch API is built into most modern browsers and also returns Promises.

#### What are we going to learn?

* How to use the **fetch()** method to make network requests
* How to extract data from fetch responses
* How to customize network request settings
* How to use **Request**objects to have more control over the **fetch()** method.

**Basic Fetch Usage**

### ****Basic Fetch Usage****

Notice how a **fetch()** method is used to make a simple network request:

fetch("https://jsonplaceholder.typicode.com/todos/1")

.then(function(result){

return result.json()

})

.then(function(result){

console.log(result);

//logs Object {completed: false, id: 1, title: "delectus aut autem", userId: 1}

})

.catch(function(err){

console.log(err);

});

#### Fetch(url)

The **fetch()** method takes in an URL endpoint and is used to make a network request. The **fetch()** method returns a Promise that contains a Response object.

Notice how the **fetch()**method returns a Promise that contains a **Response** object:

fetch("https://jsonplaceholder.typicode.com/todos/1") //fetch() method used with an URL endpoint

.then(function(result){ //result contains a Response object

});

#### Extracting data from a Response object:

A Response object has several methods that are used to extract the fetched data.

Here are the common extraction methods:

* **json()** is used to extract a json object
* **text()** is used to extract a text string
* **blob()** is used to extract a file-like object

Notice how the **json()** method is used to extract a JSONobject:

fetch("https://jsonplaceholder.typicode.com/todos/1")

.then(function(result){

return result.json() //returns a promise containing the JSON data extracted from the Response object

})

.then(function(result){

console.log(result);

//logs Object {completed: false, id: 1, title: "delectus aut autem", userId: 1}

});

Notice how the **text()** method is used to extract a text string:

fetch("https://jsonplaceholder.typicode.com/todos/1")

.then(function(result){

return result.text() //returns a promise containing the text data extracted from the Response object

})

.then(function(result){

console.log(result);

//logs "{completed: false, id: 1, title: "delectus aut autem", userId: 1}"

});

### ****Handling Fetch Responses****

#### **Checking the Response Status**

It is important to check the status of the **Response** object that is fetched. A status between 200-299 means that the request was somewhat successful while statuses in the 400s or 500s mean that problems have occurred.

Notice how the status of a bad **Response**object is checked before handling the response:

//fetching a bad url

fetch("https://jsonplaceholder.typicode.com/bad\_url/1")

.then(function(result){ //contains a Response object

console.log(result);

if(result.ok){ //returns true if the Response status is within 200-299

return result.text();

}

else{ //if the fetch request had problems

console.log(result.status) //logs 404

return Promise.reject(result.status); //returns a rejected promise if the fetch request had problems

}

})

.then(function(result){

console.log(result); //doesn't occur since a rejected promise was returned earlier

})

.catch(function(err){

console.log("Error: " + err); //logs "Error: 404", handles the rejected promise

})

**Customizing Fetch Settings**

**Fetch Init Object**

The **fetch()** method can also take in an optional **init object*.***This object applies custom settings to the Fetch request.

Notice how the **fetch()** method is used with an URL endpoint and an **init object**:

//this init object specifies the method, headers, mode and body of the request

var initObject = {

method: 'POST',

headers: new Headers(),

mode: 'cors',

body: "{}"

}

//fetch() method used with an URL endpoint and an init object

fetch("https://jsonplaceholder.typicode.com/posts",initObject)

.then(function(result){ //result contains a Response object

return result.json() //returns a promise containing JSON data extracted from the Response object

})

.then(function(result){

console.log(result);

//logs Object {id: 101}

})

.catch(function(err){

console.log(err);

});

The following attributes of the **init object**will be covered in more detail in the next few sections:

* method
* body
* headers
* mode

**Method**

The *method* attribute is a string that is used to specify the HTTP request method type.

Here is a list of some commonly used method types:

* Get - used to retrieve an existing data resource
* Head - used to retrieve HTTP headers
* Post - used to create a new data resource
* Put - used to create a new data resource or modify an existing data resource
* Delete - used to delete a data resource

Notice how an **init object**with a *method* attribute of "Post" can be created:

var initObject = {

method: 'POST'

}

**Body**

The *body* attribute is a JSON string used to send data along with a fetch request. If the *body*value is an object, it is important to stringify the object that is being sent using **JSON.stringify()** or it will not process correctly. Get and Head HTTP requests can not have bodies.

Notice how an **init object**with a *body* attribute representing an object can be created:

var myBody = {

id: 12345,

name: 'abc',

age: 21

}

var initObject = {

body: JSON.stringify(myBody)

}

**Headers**

The*headers* attribute is used to add more information about the resource being fetched or the client doing the fetching. A **Headers** object can be created using the **new Headers()** constructor and individual headers can be added to the **Headers** object through the **append()** method.

Notice how a new **Headers** object is created and assigned to the *headers* attribute of the **init object**:

var myHeaders = new Headers();

myHeaders.append('Content-Type', 'application/json');

var initObject = {

headers: myHeaders

}

**Mode**

The *mode* attribute is a string that is used to determine whether or not the Fetch request can fetch resources from different servers.

In this course we will cover the following two *mode*types:

* same-origin - the Fetch request can only fetch resources from the same server
* cors (cross origin HTTP request) - the Fetch request can fetch resources from different servers

Notice how an **init object**is created with a *mode*attribute set to 'cors':

var initObject = {

mode: 'cors'

}

**Using Fetch with Requests**

### ****Using Fetch with Requests****

The **fetch()** method can take in a **Request** object instead of an URL and an **init object**. The **Request**constructor takes in the same parameters as the **fetch()**method, an URL and an optional **init object**. **Request**objects are used because they make Fetch requests a bit cleaner and also offer a bit more control.

Notice how a **Request**object is created and used with a **fetch()** method call:

//this init object specifies the method, headers, mode and body of the request

var initObject = {

method: 'POST',

headers: new Headers(),

mode: 'cors',

body: "{}"

}

//creates a new request object using an URL and an init object

var request = new Request("https://jsonplaceholder.typicode.com/posts",initObject)

//fetch() method used with a request

fetch(request).then(function(result){ //result contains a Response object

return result.json()

//returns a Promise containing JSON data extracted from the Response object

}).then(function(result){

console.log(result);

//logs Object {id: 101}

}).catch(function(err){

console.log(err);

});

### ****Reusing Request ObJects****

#### Requests with Bodies(POST, PUT)

If a **Request** object is used more than once in a Fetch request that involves bodies (POST, PUT) it will throw an error.

Notice how an error is thrown if a **Request**object is fetched again after being previously used in a POST request:

var initObject = {

method: 'POST',

headers: new Headers(),

mode: 'cors',

body: "{}"

}

var request = new Request("https://jsonplaceholder.typicode.com/posts",initObject)

//first time using Request object

fetch(request).then(function(result){

return result.json()

}).then(function(result){

console.log(result);

//logs Object {id: 101}

}).catch(function(err){

console.log(err);

});

//second time using Request object

fetch(request).then(function(result){

return result.json();

}).catch(function(err){

console.log(err.message)

// logs "Failed to execute 'fetch' on 'Window': Cannot construct

// a Request with a Request object that has already been used."

});;

#### Requests without Bodies (GET, HEAD)

However, **Request**objects can be used more than once in Fetch requests that don't involve bodies(Head,Get).

Notice how a **Request** object can be reused in multiple GET requests:

//makes a GET request

var request = new Request("https://jsonplaceholder.typicode.com/todos/1")

//first fetch request

fetch(request).then(function(result){

console.log(result.status) //logs 200, OK fetch response

})

//reusing request object

fetch(request).then(function(result){

console.log(result.status) //logs 200, OK fetch response after reusing request object

})