FETCH API TEACHING TIP

Purpose

The purpose of this document is to serve as a good resource and guide for CSCI E-31 students and web developers who are fairly new to Fetch API. This document attempts to break down the Fetch API structure and functionalities for web developers on-the-go. Common elements of fetch API are covered but detailed information about Fetch API are documented in the MDN web docs Fetch API and Using Fetch API webpages

Introduction

Fetch API is an Ajax web development technique that enables web applications to asynchronously send and receive data from a server without reloading the current page. Fetch API is relatively new but is more dynamic and easier that the older XMLHttpRequest method. It comprises an extensive library of constructors, properties and methods that simplify modern day ajax requests.

Basic Structure

The basic form of a Fetch API request is as follows,

```
fetch(URL, request init-object)
  .then(function(response object) {
    //code describing how the server response object is handled
  })
  .catch(function(error) {
    //code describing how the fetch errors is handled
  })
```

From the code snippet above,

1. The fetch() method sends the request init-object to the server listening at the “URL” address.
2. The server returns a promise that contains the response object. This response object is further accessed and processed by the “`.then()`” method via a callback.
3. If the fetch() URL request generates an error, the “`.catch()`” callback specifies how the error is handled.

Elements

The Fetch API primarily comprises three elements,

- URL of the resource
- Request init object
- Response object

The Fetch API request and response objects provide an easy and logical interface for executing dynamic ajax calls. The table below describes some Fetch API request/response object methods and properties that comprise the Fetch API interface. These are interfaces that enable the customization of common server requests and handling of common server responses.

CATEGORY	**NAME**	**DESCRIPTION**	**EXAMPLE**
URL	URL	This is a string that contains the path of the target resource.	`‘facebook.com/john’`
Request	body	This is the interface which describes that body/payload of the Fetch request. It can include strings, arrays, formData etc.	`body: ‘Nothing here’`
Request	cache	This is the interface which describes the cache mode of the request.	`cache: ‘no-cache’`
Request	method	This specifies the request’s method.	`method:’POST’`
Request	mode	This specifies the mode of the request. It determines cross-origin request validity and readable properties of the request object.	`mode: ‘cors’`
Request	redirect	This determines how redirects are handled.	`redirect: ‘follow’`
Request	headers	This describes the header details of the request object.	`headers: { 'content-type': 'application/json' }`
Request	credentials	This is the property that indicates whether the user agent should send cookies from the other domain in case or cross-origin request.	`credentials: 'same-origin'`

Response	headers	This contains a headers{} object that describes the header of the response object.	`console.log(response.headers);`
			`// returns a headers{} object`

Response	redirected	This is a Boolean value that indicates whether the response is a result of a request which you made which was redirected.	`if(response.redirected){`
			`   console.log("unwanted redirect")`
			`}`

Response	status	This contains the status code of the response.	`console.log(response.status);`
			`// returns "200" on success`

Response	body	This contains a readable stream of the response body contents .	`console.log(response.body);`
			`// returns "Just text content"`

Response	text()	This returns a promise that contains a plain text content of the response body.	`response.text().then(function(text) {`
			`  console.log(text);`
			`});`

Response	json()	This returns a promise that stores the response body text object as JSON.	`response.json().then(function(json) {`
			`  console.log(json.firstname);`
			`});`

Response	blob()	This returns a promise that the response body content as a file-like object of immutable, raw data.	`.then(function(Blob) {`
			`var imgURL = URL.createObjectURL(Blob);`
			`  myImage.src = objectURL;`
			`});`

Note: The table above describes the Fetch API interfaces used for common ajax and do not cover all elements. For more information visit the MDN web docs Fetch API and Using Fetch API pages.

<u>FETCH EXAMPLE: Supplying Several Request Object Options</u>

```
postData('http://wakanda.com/finance', {From_Klaw: 10,000,000})
  .then(amount => console.log(data)) // JSON dump from `response.json()` call
  .catch(error => console.error(error))

function postData(url, vibranium_payment) {
  // Default options are marked with *
  return fetch(url, {
    body: JSON.stringify(vibranium_payment),
    cache: 'no-cache',
    credentials: 'same-origin',
    headers: {
      'user-agent': 'Mozilla/4.0 MDN Example',
      'content-type': 'application/json'
    },
    method: 'POST',
    mode: 'cors',
    redirect: 'follow',
    referrer: 'no-referrer',
  })
  .then(response => response.json()) // parses response to JSON
```

FETCH EXAMPLE: Uploading JSON data to Wakanda.com

```
var url = 'https://wakanda.com/profile';
var name_data = {username: 'tchalla'};

fetch(url, {
  method: 'POST',
  body: JSON.stringify(name_data),
  headers: new Headers({
    'Content-Type': 'application/json'
  })
}).then(res => res.json())
.catch(error => console.error('Upload Error:', error))
.then(response => console.log('Upload Success:', response));
```

FETCH EXAMPLE: Uploading a File to Wakanda.com

```
var formData = new FormData();
var image_file = document.querySelector("input[type='file']");

formData.append('username', 'Eric Killmonger');
formData.append('avatar', image_file.files[0]);

fetch('https:// wakanda.com/profile_picture', {
  method: 'PUT',
  body: formData
})
.then(response => response.json())
.catch(error => console.error('Error:', error))
.then(response => console.log('Success:', response));
```