Fetching Data From Remote Servers

The Fetch API provides an interface for fetching resources, primarily across the network.

Using

The `Fetch API` is provided in the browser as a method named `fetch`

Takes at *least* one argument: a string representing the `URL` to access.

Make a sample TypeScript app for demonstration or use RunJS

Example

```
fetch('https://restcountries.eu/rest/v2/all')
```

This will access the *Countries* and fetch a list of all countries.

What do we receive?

If log this response we will see something that we cannot directly use.

```
let response = fetch('https://restcountries.eu/rest/v2/all')
console.log(response)
```

```
Promise {<pending>}
  __proto__: Promise
  [[PromiseStatus]]: "pending"
  [[PromiseValue]]: undefined
```

JavaScript `Promise` is like `Task` from `C#`

- We cannot use this `promise` directly, we must "resolve" the promise
- Think of a `promise` as an *IOU*
- A `promise` is an *asynchronous IOU* that will supply a function when the *IOU* is ready to redeem.
- To cash-in on our *IOU* we call the `then` method of the `promise` as such:

```
fetch('https://restcountries.eu/rest/v2/all').then(response => {
  console.log(response)
})
```

The response here is *still* not quite usable:

```
Response {type: "cors", url: "https://restcountries.eu/rest/v2/all/", redirected: true, status: 200, ok: true, …}
body: (...)
bodyUsed: false
headers: Headers {}
ok: true
redirected: true
status: 200
statusText: ""
type: "cors"
url: "https://restcountries.eu/rest/v2/all/"
```

Deserialize...

- This is because the response *body* itself must be *converted* into a form we can use.
- Fortunately, the `response` object gives us a method to gain access to the JSON:

```
fetch('https://restcountries.eu/rest/v2/all')
  .then(response => {
    return response.json()
  })
  .then(json => {
    console.log(json)
  })
```

This returns usable information!

```
[
  { name: 'Afghanistan', alpha2Code: 'AF', alpha3Code: 'AFG' },
  { name: 'Åland Islands', alpha2Code: 'AX', alpha3Code: 'ALA' },
  { name: 'Albania', alpha2Code: 'AL', alpha3Code: 'ALB' },
  { name: 'Algeria', alpha2Code: 'DZ', alpha3Code: 'DZA' },
  { name: 'American Samoa', alpha2Code: 'AS', alpha3Code: 'ASM' },
]
```

Improving the use of `fetch`

- Promises can often be challenging to use
- JavaScript has implemented a way to make asynchronous calls into synchronous calls.
- Similar to C#'s `async / await` system we can add `await` to a call that returns a promise.

Example

```
function countries() {
  const response = await fetch('https://restcountries.eu/rest/v2/all')

  // Check if the response is valid before decoding
  if (response.ok) {
    const json = await response.json()
    console.log(json)
  }
}
```

```
async function countries() {
  const response = await fetch('https://restcountries.eu/rest/v2/all')

  if (response.ok) {
    const json = await response.json()
    console.log(json)
  }
}
```

Making a POST request

- Supply a second argument, an object of options
- First property is required, and is the `method`
- May also need to provide an object containing `headers`
- If this is a `POST` we'll need a `body` in the correct format

```
async function createOneListItem() {
  const response = await fetch(
    // URL
    'https://one-list-api.herokuapp.com/items?access_token=illustriousvoyage',

    // Options
    {
      // This is a POST request
      method: 'POST',

      // We are sending JSON
      headers: { 'content-type': 'application/json' },

      // The body of the message is the object, but turned into a string in JSON format
      body: JSON.stringify({
        item: { text: 'Learn about Regular Expressions!' },
      }),
    }
  )

  if (response.ok) {
    // Process the response
    const json = await response.json()
    console.log(json)
  }
}
```

PUT, PATCH, DELETE, etc.

The HTTP verbs will work similarly to the `POST`

Fetch API is simple, powerful, but awkward to use

We'll be looking at some alternatives:

- axios
- react-query