Colton Kramer | Week 10 Reading | 03/08/2022

Before submitting info to the server, the client should do its own checks to make sure the data is correctly formatted. This is done with form validation. Validation done in the browser is called client-side validation and validation done in the server is called server-side validation. There are 3 main reasons we do this:

We want to get the right data

We want to protect users data

We want to protect ourselves

Using HTML5 built in controls such as listed below we can set up simple client side validation:

Required: cannot submit the form unless all requireds are filled in

Min/max length: Set min/max length of strings

Min/max: sets min/max values of numerical input types

Pattern: uses regular expression that defines a pattern the entered data needs to follow

Validating against a REGEX

Here are some of the patterns you may find:

a — Matches one character that is a (not b, not aa, and so on).  
abc — Matches a, followed by b, followed by c.  
ab?c—Matches a, optionally followed by a single b, followed by c. ( ac or abc)  
ab\*c—Matches a, optionally followed by any number of bs, followed by c. ( ac , abc, abbbbbc, and so on).  
a|b — Matches one character that is a or b.  
abc|xyz — Matches exactly abc or exactly xyz (but not abcxyz or a or y, and so on).

Most browsers support a Constraint Validation API which consists of methods and properties. With this you can return a validation message that returns a message telling you to whether your form have been validated or not.

The API above has many other methods used to help make a working form.

Using the fetch API

The most basic fetch looks like this

fetch('http://example.com/movies.json')

.then(response => response.json())

.then(data => console.log(data));

This grabs a JSON file across the network uses the .json() method to filter the http fluff and save it to an object that we can use in a project.

Checking if a fetch was successful is possible with by checking that the promise resolved then checking that the “Response.ok” property has a value of true.

The headers interface allows you to create your own headers via the “Headers()” constructor. Like this:

Const myHeaders = new Headers();

Guard

Since headers can be sent in requests received in responses, and have various limitations about what info can and should be mutable, headers’ objects have a guard property. Possible guard values are:

none: default.  
request: guard for a headers object obtained from a request ([Request.headers](https://developer.mozilla.org/en-US/docs/Web/API/Request/headers)).  
request-no-cors: guard for a headers object obtained from a request created with [Request.mode](https://developer.mozilla.org/en-US/docs/Web/API/Request/mode) no-cors.  
response: guard for a headers object obtained from a response ([Response.headers](https://developer.mozilla.org/en-US/docs/Web/API/Response/headers)).  
immutable: guard that renders a headers object read-only; mostly used for ServiceWorkers.

Body

Both requests and responses may contain body data. A body is an instance of any of the following tpyes:

ArrayBuffer  
ArrayBufferView  
Blob/File  
string  
URLSearchParams  
FormData