Dylan Warner

6/26/2021

WDD 330

W10 Readings

**Client-side Form Validation**

* Client-side form validation is an attempt to get the correct information from a user before it is sent to the backend of our applications.
* There are two types of client-side form validation: built-in form validation, JavaScript validation.
* There are alot of features built in for form validation that makes using javascript unnecessary in many cases. There are built in methods such as: required, minlength, maxlength, min, max, type, and pattern.

*Validating Forms Using JavaScript:*

* Most browsers support the [Constraint Validation API](https://developer.mozilla.org/en-US/docs/Web/API/Constraint_validation), which consists of a set of methods and properties available on the following form element DOM interfaces:

HTMLButtonElement (represents a <button> element)

HTMLFieldSetElement (represents a <fieldset> element)

HTMLInputElement (represents an <input> element)

HTMLOutputElement (represents an <output> element)

HTMLSelectElement (represents a <select> element)

HTMLTextAreaElement (represents a <textarea> element)

* The Constraint validation API makes the following properties available on the above elements.
* validationMessage: Returns a localized message describing the validation constraints that the control doesn't satisfy (if any). If the control is not a candidate for constraint validation (willValidate is false) or the element's value satisfies its constraints (is valid), this will return an empty string.
* validity: Returns a ValidityState object that contains several properties describing the validity state of the element. You can find full details of all the available properties in the [ValidityState](https://developer.mozilla.org/en-US/docs/Web/API/ValidityState) reference page; below is listed a few of the more common ones:
  + [patternMismatch](https://developer.mozilla.org/en-US/docs/Web/API/ValidityState/patternMismatch): Returns true if the value does not match the specified [pattern](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input#attr-pattern), and false if it does match. If true, the element matches the [:invalid](https://developer.mozilla.org/en-US/docs/Web/CSS/:invalid) CSS pseudo-class.
  + [tooLong](https://developer.mozilla.org/en-US/docs/Web/API/ValidityState/tooLong): Returns true if the value is longer than the maximum length specified by the [maxlength](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input#attr-maxlength) attribute, or false if it is shorter than or equal to the maximum. If true, the element matches the [:invalid](https://developer.mozilla.org/en-US/docs/Web/CSS/:invalid) CSS pseudo-class.
  + [tooShort](https://developer.mozilla.org/en-US/docs/Web/API/ValidityState/tooShort): Returns true if the value is shorter than the minimum length specified by the [minlength](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input#attr-minlength) attribute, or false if it is greater than or equal to the minimum. If true, the element matches the [:invalid](https://developer.mozilla.org/en-US/docs/Web/CSS/:invalid) CSS pseudo-class.
  + [rangeOverflow](https://developer.mozilla.org/en-US/docs/Web/API/ValidityState/rangeOverflow): Returns true if the value is greater than the maximum specified by the [max](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input#attr-max) attribute, or false if it is less than or equal to the maximum. If true, the element matches the [:invalid](https://developer.mozilla.org/en-US/docs/Web/CSS/:invalid) and [:out-of-range](https://developer.mozilla.org/en-US/docs/Web/CSS/:out-of-range) CSS pseudo-classes.
  + [rangeUnderflow](https://developer.mozilla.org/en-US/docs/Web/API/ValidityState/rangeUnderflow): Returns true if the value is less than the minimum specified by the [min](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input#attr-min) attribute, or false if it is greater than or equal to the minimum. If true, the element matches the [:invalid](https://developer.mozilla.org/en-US/docs/Web/CSS/:invalid) and [:out-of-range](https://developer.mozilla.org/en-US/docs/Web/CSS/:out-of-range) CSS pseudo-classes.
  + [typeMismatch](https://developer.mozilla.org/en-US/docs/Web/API/ValidityState/typeMismatch): Returns true if the value is not in the required syntax (when [type](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input#attr-type) is email or url), or false if the syntax is correct. If true, the element matches the [:invalid](https://developer.mozilla.org/en-US/docs/Web/CSS/:invalid) CSS pseudo-class.
  + valid: Returns true if the element meets all its validation constraints, and is therefore considered to be valid, or false if it fails any constraint. If true, the element matches the [:valid](https://developer.mozilla.org/en-US/docs/Web/CSS/:valid) CSS pseudo-class; the [:invalid](https://developer.mozilla.org/en-US/docs/Web/CSS/:invalid) CSS pseudo-class otherwise.
  + valueMissing: Returns true if the element has a [required](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input#attr-required) attribute, but no value, or false otherwise. If true, the element matches the [:invalid](https://developer.mozilla.org/en-US/docs/Web/CSS/:invalid) CSS pseudo-class.
* willValidate: Returns true if the element will be validated when the form is submitted; false otherwise.
* What kind of validation should I perform?
* What should I do if the form doesn't validate?
* How can I help the user to correct invalid data?
* These are the questions to ask when creating your own custom JavaScript form validation.

**Using Fetch**

* The Fetch API provides a JavaScript interface for accessing and manipulating parts of the HTTP pipeline, such as requests and responses. It also provides a global fetch() method that provides an easy, logical way to fetch resources asynchronously across the network.
* EX. simply Fetch request:

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

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

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

*Supplying Request Options:*

// Example POST method implementation:

async function postData(url = '', data = {}) {

// Default options are marked with \*

const response = await fetch(url, {

method: 'POST', // \*GET, POST, PUT, DELETE, etc.

mode: 'cors', // no-cors, \*cors, same-origin

cache: 'no-cache', // \*default, no-cache, reload, force-cache, only-if-cached

credentials: 'same-origin', // include, \*same-origin, omit

headers: {

'Content-Type': 'application/json'

// 'Content-Type': 'application/x-www-form-urlencoded',

},

redirect: 'follow', // manual, \*follow, error

referrerPolicy: 'no-referrer', // no-referrer, \*no-referrer-when-downgrade, origin, origin-when-cross-origin, same-origin, strict-origin, strict-origin-when-cross-origin, unsafe-url

body: JSON.stringify(data) // body data type must match "Content-Type" header

});

return response.json(); // parses JSON response into native JavaScript objects

}

postData('https://example.com/answer', { answer: 42 })

.then(data => {

console.log(data); // JSON data parsed by `data.json()` call

});

*Sending a Request With Credentials Included:*

* To cause browsers to send a request with credentials included on both same-origin and cross-origin calls, add credentials: 'include' to the init object you pass to the fetch() method.