Week 10

Monday, June 21, 2021 6:44 AM

Client-side form validation

When you enter data, the browser and/or the web server will check to see that the data is in the correct format and within the constraints set by the application. Validation done in the browser is called **client-side** validation, while validation done on the server is called **server-side** validation.

We want to make filling out web forms as easy as possible. So why do we insist on validating our forms? There are three main reasons:

- **We want to get the right data, in the right format.** Our applications won't work properly if our users' data is stored in the wrong format, is incorrect, or is omitted altogether.
- **We want to protect our users' data**. Forcing our users to enter secure passwords makes it easier to protect their account information.
- **We want to protect ourselves**. There are many ways that malicious users can misuse unprotected forms to damage the application.

HTML5 form controls include the following:

- required: Specifies whether a form field needs to be filled in before the form can be submitted.
- minlength and maxlength: Specifies the minimum and maximum length of textual data (strings)
- min and max: Specifies the minimum and maximum values of numerical input types
- type: Specifies whether the data needs to be a number, an email address, or some other specific preset type.
- pattern: Specifies a regular expression that defines a pattern the entered data needs to follow.

When an element is valid, the following things are true:

- The element matches the :valid CSS pseudo-class, which lets you apply a specific style to valid elements.
- If the user tries to send the data, the browser will submit the form, provided there is nothing else stopping it from doing so (e.g., JavaScript).

When an element is invalid, the following things are true:

- The element matches the :invalid CSS pseudo-class, and sometimes other UI pseudo-classes (e.g., :out-of-range) depending on the error, which lets you apply a specific style to invalid elements.
- If the user tries to send the data, the browser will block the form and display an error message.

The Constraint Validation API

Most browsers support the Constraint Validation API, 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 reference page; below is listed a few of the more common ones:
  - patternMismatch: Returns true if the value does not match the specified pattern, and false if it does match. If true, the element matches the :invalid CSS pseudo-class.
  - tooLong: Returns true if the value is longer than the maximum length specified by the maxlength attribute, or false if it is shorter than or equal to the maximum. If true, the element matches the :invalid CSS pseudo-class.
  - tooShort: Returns true if the value is shorter than the minimum length specified by the minlength attribute, or false if it is greater than or equal to the minimum. If true, the element matches the :invalid CSS pseudo-class.
  - rangeOverflow: Returns true if the value is greater than the maximum specified by the max attribute, or false if it is less than or equal to the maximum. If true, the element matches the :invalid and :out-of-range CSS pseudo-classes.
  - rangeUnderflow: Returns true if the value is less than the minimum specified by the min attribute, or false if it is greater than or equal to the minimum. If true, the element matches the :invalid and :out-of-range CSS pseudo-classes.
  - typeMismatch: Returns true if the value is not in the required syntax (when type is email or url), or false if the syntax is correct. If true, the element matches the :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 CSS pseudo-class; the :invalid CSS pseudo-class otherwise.
  - valueMissing: Returns true if the element has a required attribute, but no value, or false otherwise. If true, the element matches the :invalid CSS pseudo-class.
- willValidate: Returns true if the element will be validated when the form is submitted; false otherwise.

The Constraint Validation API also makes the following methods available on the above elements.

- checkValidity(): Returns true if the element's value has no validity problems; false otherwise. If the element is invalid, this method also fires an invalid event on the element.
- setCustomValidity(*message*): Adds a custom error message to the element; if you set a custom error message, the element is considered to be invalid, and the specified error is displayed. This lets you use JavaScript code to establish a validation failure other than those offered by the standard HTML5 validation constraints. The message is shown to the user when reporting the problem.

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.

Uploading JSON data

Use fetch() to POST JSON-encoded data.

```
const data = { username: 'example' };
fetch('https://example.com/profile', {
  method: 'POST', // or 'PUT'
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify(data) , // Notice how data is being sent
})
.then(response => response.json())
.then(data => {
  console.log('Success:', data);
})
.catch((error) => {
  console.error('Error:', error);
});
```