Forms and Validations

Before submitting data to the server, it is important to ensure all required form controls are filled out, in the correct format. This is called Client-Side Form Validation, and helps ensure data submitted matches the requirements set forth in the various form controls.

Client-Side validation is an initial check and an important feature of good user experience. By catching invalid data on the client-side, the user can fix it straight away. If it gets to the server and is then rejected, a noticeable delay is caused by a round trip to the server and then back to the client to tell the user to fix their data.

However, client-side validation should not be considered an exhaustive security measure! Your apps should always perform security checks on any form-submitted data on the server-side as well as the client-side, because client-side validation is too easy to bypass.

What is form validation?

When you go to any site with a registration form, you'll notice that they provide feedback when you don't enter your data in the format they are expecting. Below are some messages examples that you'll get:

- This field is required
- Please enter your phone number in the format xxxx-xxxx
- Please enter a valid email address
- Your password needs to be between 8 and 30 characters long and contain one uppercase letter, one symbol, and a number

This is called 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.

Different types of client-side validation

There are two different types of client-side validation that you’ll encounter on the web:

- **Built-in form validation -** Uses HTML form validation features. This validation generally doesn’t require much JavaScript. Built-in form validation has better performance than JavaScript, but it is not as customizable as JavaScript validation;
- **JavaScript Validation -** Is coded using JavaScript. This validation is completely customizable, but you need to create it all (or use a library);

Built-in form validation

One of the most significant features of modern form controls is the ability to validate most user data without relying on JavaScript. This is done by using validation attributes on form elements:

- **required -** Specifies whether a form field need to be filled in before the form can be submitted;
- **minlength & maxlength -** Specifies the minimum and maximum length of textual data;
- **min & max -** Specifies the minimum and maximum values of numerical input types;
- **type -** Specifies whether the data need 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 need to follow;

If the data entered in a form field follows all of the rules specified by the above attributes, it is considered valid. If not, it is considered invalid. 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.

Validation forms using JavaScript

The constraint Validation API 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 and the form element.

- **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.
- **reportValidity() -** Reports invalid field(s) using events. This method is useful in combination with preventDefault() in an onSubmit event handler.
- **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 HTML validation constraints. The message is shown to the user when reporting the problem.

References

Client-side form validation - Learn web development | MDN (mozilla.org)