## Validating Forms - MDN:

Client-side form 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 just tell the user to fix their data.

It should not, however, be your *only* security measure; your apps should be checked for security on **both** the server-side *and* the client-side. This is because malicious users can still send bad data to your servers, as client-side validations can be all too easily bypassed.

**Warning: Never trust data passed to your server from the client. Even if your form is validating correctly and preventing malformed input on the client-side, a malicious user can still alter the network request.**

### Built-in Form validation (HTML):

Not as customizable as JavaScript validation, but has generally better performance.

Things include:

* *required*: Specifies whether a form field needs to be filled before the form can be submitted.
* *minlength/maxlength*: Specifies the minimum and maximum lengths of **textual data (strings)**.
* *min/max*: Specifies the minimum and maximum values of **numerical** input types.
* *type*: Specifies whether the data needs to be a number, email address, or some other specific preset type.
* *pattern*: Specifies a [**regular expression**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions) that defines a pattern the entered data needs to follow.

### JavaScript Validation:

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

* [HTMLButtonElement](https://developer.mozilla.org/en-US/docs/Web/API/HTMLButtonElement) (represents a [button](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/button) element)
* [HTMLFieldSetElement](https://developer.mozilla.org/en-US/docs/Web/API/HTMLFieldSetElement) (represents a [fieldset](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/fieldset) element)
* [HTMLInputElement](https://developer.mozilla.org/en-US/docs/Web/API/HTMLInputElement) (represents an [input](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input) element)
* [HTMLOutputElement](https://developer.mozilla.org/en-US/docs/Web/API/HTMLOutputElement) (represents an [output](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/output) element)
* [HTMLSelectElement](https://developer.mozilla.org/en-US/docs/Web/API/HTMLSelectElement) (represents a [select](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/select) element)
* [HTMLTextAreaElement](https://developer.mozilla.org/en-US/docs/Web/API/HTMLTextAreaElement) (represents a [textarea](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/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**](https://developer.mozilla.org/en-US/docs/Web/API/ValidityState); 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. 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.

## Fetch API:

This has been reviewed in WDD 230 and on personal studies.