2020. 8. 9. WEB222 - Week 11 | web222

[**web222**](https://web222.ca/)



**WEB222 - Week 11**



**Suggested Readings**



[HTML Form Validation](https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms/Form_validation)



[Static Site Hosting](https://web222.ca/static-site-hosting.html)



**Client Side Form Validation**



When a user submits a form, we generally want to send the form’s data to a server. We use the form’s action to specify a server URL, and a method to indicate the HTTP request type to use when sending the data.

Before we can use this data in a meaningful way, we need to validate it. It’s easy for users to make typos, enter the right information but in the wrong field, or use a format we aren’t expecting. We need to be able to parse and understand the data using code. This means having data that follows some rules.

In order to be able to work with user data, we have to provide some mechanisms for enforcing these rules, and give users hints, guides, and safety checks as they are entering data and submitting forms.

We have two opportunities to validate form data:

1. Client-Side: before we submit the form to the server, we validate it in the browser using HTML5 and JavaScript.
2. Server-Side: after the data is submitted, the server must re-validate it.

We will be focusing on client-side validation in this course.

You might be wondering why we bother validating form data twice, if we’re just going to re-validate it no matter what on the server. There are a number of reasons:

1. Save bandwidth: don’t send data over the network if it’s incomplete or not in the correct form
2. Immediate feedback: users don’t have to wait for their data to travel all the way to the server, and the page to reload, before getting feedback that they need to correct something simple.
3. Contextual feedback: prompt users to correct mistakes as they are entering the data vs. at the end, after they’ve moved on from entering some piece of information (e.g. a credit card).

**HTML5 Validation Features**



We’ve already discussed a number of important <input> types that allow us to tell the browser about the type of data we expect, for example [<input type="tel">](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input/number) for telephone numbers or [<input type="email">](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input/email) for email addresses.

Each of these special purpose <input> types comes with its own set of built-in data validation:

**Email Address**

[<input type="email">](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input/email)

An email address must not be an empty string, and must be a valid (i.e., text is in valid email format vs. email address actually exists). If you include the multiple attribute, the control will allow a list of addresses, and validate each one.

**Telephone Number**

[<input type="tel">](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input/number)

Phone numbers are very difficult to validate, because they [differ so much around the world](https://github.com/googlei18n/libphonenumber/blob/master/FALSEHOODS.md). You might think you could just check for something like 555-555-5555 , but this would miss things like country codes, number patterns that use a different number of digits, short-codes for texting, 1-800 style numbers, etc.

As a result, there is no default validation applied to a tel type input.

**URL**

file:///C:/Users/k/Desktop/여름학기/WEB222\_NDD/Week11/WEB222-Lecture11 Client-side validation.html 1/6

2020. 8. 9. WEB222 - Week 11 | web222

[<input type="url">](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input/url)

Unlike telephone numbers, URLs *can* be validated. If you use a url type input, the browser will make sure it is not empty, and that the value is a valid URL.

**Dates and Times**

[<input type="date">](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input/date) , [<input type="time">](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input/time) , [<input type="week">](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input/week) , [<input type="month">](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input/month) , [<input type="date-local">](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input/date-local)

Dates and times are not validated by the browser. However, the user will usually be prompted to “pick” a date/time value visually instead of entering one as text. You can also further restrict the date/time by adding a min="..." or max="..." to the input , which specifies a date/time to use as a lower or upper range when validating.

**Colour**

[<input type="color">](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input/color)

* color ’s value is considered to be invalid if it can’t be converted (by the browser) into a seven-character lower-case hexadecimal value (e.g., #000000 ).

**Number**

[<input type="number">](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input/number) , [<input type="range">](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input/range)

* number must be a valid number, or the browser won’t allow it. You can also further restrict the number’s value by adding a min="..." or max="..." to the input , which specifies a lower or upper range when validating.

**Using Attributes to Prevent Invalid Data**



Beyond choosing trying to choose the most appropriate <input> type for your data, another layer of client-side validation comes from using attributes to indicate to both the user and browser what we expect to be entered.

**placeholder and title**

We’ve discussed [placeholder](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input#Labels_and_placeholders) previously as part of our forms and CSS discussion. It’s important to highlight it once again since it also plays an important role in helping the user understand how to enter data properly.

Together with <label> s and the title attribute (shown when you hover over an element in a tooltip), these extra bits of text provide important clues and instructions about how to use a given input control.

For example, if we are expecting the user to enter a list of email addresses, we could do the following:

<label for="address-list">Email Address List</label>

<input

id="address-list"

type="email"

multiple

placeholder="name1@example.com, name2@example.com, ..."

title="List of email addresses, separated by commas"

>

**disabled**

The [disabled attribute](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input#disabled) is a boolean (i.e., it is present or not present) that indicates that a field cannot be interacted with by the user. In the browser it will show up with a dimmer colour, and clicking it will have no effect.

We can use disabled to turn off certain controls in a form that don’t currently apply. Sometimes a form will have options with dependencies on other controls. For example, booking a flight that is one-way vs. two-way and whether or not you need a second date entered for the return trip.

<form action="/s" name="login">

<input type="text" name="flight">

<input type="date" name="date1">

<input type="checkbox" name="return-flight">

<input type="date" value="date2" disabled>

</form>

file:///C:/Users/k/Desktop/여름학기/WEB222\_NDD/Week11/WEB222-Lecture11 Client-side validation.html 2/6

2020. 8. 9. WEB222 - Week 11 | web222

Using disabled allows us to include and display optional input options in a form without polluting the data by accidentally allowing the user to enter information that isn’t appropriate.

**required**

The [required attribute](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input#required) is a boolean (i.e., it is present or not present) that indicates that a field **must** have a value before the user can submit the form. The browser will block attempts to submit until a value has been entered.

<form action="/s" name="login">

<input type="text" name="username" required>

<input type="password" name="password" required>

<input type="submit" value="Login">

</form>

In the form above, both the username and password fields are required, and must have a value before the form can be submitted (i..e, by clicking the Login button). Notice that the submit control does not have required attribute.

When a field has the required attribute, the browser automatically applies the :required pseudo-class. On the other hand, any field *without* therequiredattribute automatically gets the:optionalpseudo-class applied. This can be useful in CSS styling.

input**:required** {

*/\* styles for required input controls \*/*

}

input**:optional** {

*/\* styles for optional input controls \*/*

}

**pattern**

The [pattern attribute](https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms/Form_validation#Validating_against_a_regular_expression) allows us to include a regular expression for the browser to use when validating the value entered by a user for a given input control.

For example, imagine if we need the user to enter a file extension and want to support data of the following form .exe , .EXE , or exe .:

<input

name="file-extension"

type="text"

placeholder=".exe"

pattern="\.?[a-zA-Z]{3}"

>

Consider how you might write a regular expression for each of the following:

social security number (###-##-####)



phone number (555-555-5555 or 555-5555 or (555) 555-555)



ip address (127.0.0.1 or 255.255.255.255)



username (alpha, numbers underscore, dash, 8-16 long)



password (alpha, number, symbols, underscore, dash, up to 256 long)



postal code (m5w 1e6 or M5W 1E6 or M5W1E6)



price ($1.50 or 1.50 or 1)



seneca course code (WEB222SSA)



**JavaScript and Client-Side Validation**



All of the methods above are examples of static checks (i.e., they don’t change) that we’re adding to our form controls. They do a lot to help guard against invalid data; however, there are times that we need more flexible control over what happens.

In order to add dynamic checks (i.e., can be changed at runtime) we need to layer in use of JavaScript. By using JavaScript we have more freedom to create custom and complex validation rules beyond the set of static options provided by HTML and the browser. We can also use JavaScript in combination with CSS to provide a better user experience:

display more meaningful, context-aware error messages



show/hide error messages depending on the location of the cursor and where the user is focused



file:///C:/Users/k/Desktop/여름학기/WEB222\_NDD/Week11/WEB222-Lecture11 Client-side validation.html 3/6

2020. 8. 9. WEB222 - Week 11 | web222

place errors (or other information) anywhere in the DOM vs. being limited to labels, placeholder or tooltip text



**Accessing Form Fields**

When writing JavaScript to validate form fields, there are a number of ways to access the input controls and get their values. Consider the following form:

<form id="info-form" name="info" action="/i">

<input id="first-name" name="fname" type="text">

<input id="number-list" name="number-list" type="text">

</form>

Here are some of different ways we could access this form:

*// 1. Using its id and getElementById()*

**var** form **=** document.getElementById('info-form');

*// 2. Using its id and querySelector()*

**var** form **=** document.querySelector('#info-form');

* *3. Using document.forms and the id or name of the form* **var** form **=** document.forms["info-form"];

Once we have a reference to the <form> element in JavaScript, we can use the name of the form controls to get access to the individual fields and there .value :

* *Notice that we must wrap values with '-' in their names in ["..."] to access them.* **var** form **=** document.forms["info-form"];

**var** fname **=** form.fname.value;

**var** numberList **=** form["number-list"].value;

**Special Cases for Obtaining Form Values**

Some form controls need different approaches when you want to access their value in JavaScript:

1. <textarea> : use the .value property to access the text.
2. <input type="radio"> : use the name property (i.e., all radio buttons will use the same name in a group) to iterate over all possible radio controls, and then look at the .checked property, which will be true for the one checked.
3. <input type="checkbox"> : use the name property to iterate over all possible radio controls, and then look at the .checked property, which will be true for the one checked.
4. <select> : use the selectedIndex to determine which <option> index was selected (if any). A value of -1 means none are currently selected; a value greater than -1 indicates the index to use when accessing the options[n] array for the chosen option. If the <select> is defined to allow for multiple options, you can loop through the options and inspect the .selected property to determine if it’s true .

Consider the following form:

<form id="info-form" name="info" action="/i">

<label for="text">Enter some text</label>

<textarea id="text" name="text"></textarea>

<fieldset>

<legend>Pick a Colour</legend>

<label for="colour-red">Red</label>

<input type="radio" name="colour" id="colour-red" value="red" checked>

<label for="colour-green">Green</label>

<input type="radio" name="colour" id="colour-green" value="green">

<label for="colour-blue">Red</label>

<input type="radio" name="colour" id="colour-blue" value="blue">

</fieldset>

<label for="agree-disagree">I agree with the terms and conditions.</label> <input type="checkbox" name="agree" id="agree-disagree">

<label for="food">Favourite Food</label>

file:///C:/Users/k/Desktop/여름학기/WEB222\_NDD/Week11/WEB222-Lecture11 Client-side validation.html 4/6

2020. 8. 9. WEB222 - Week 11 | web222

<select id="food" name="food">

<option value="pizza">Pizza</option>

<option value="tacos">Tacos</option>

<option value="salad">Salad</option>

<option value="other">Other</option>

</select>

</form>

In order to access the form’s values in code, we could do the following:

**var** form **=** document.querySelector('#info-form');

* *Get the value form the <textarea>* **var** text **=** form.text.value.trim();
* *Get the chosen colour value from the radio button group* **var** colour;

**var** colourChoices **=** Array.**from**(form.colour);*// convert to array*colourChoices.forEach(**function**(option) {

**if**(option.checked) {colour **=** option.value;

}

});

* *Get the chosen food value form the <select>*

**var** food **=** "None";*// there may be nothing selected*

**var** foodChoices **=** Array.**from**(form.food);*// convert to array*

foodChoices.forEach(**function**(option) {

**if**(option.selected) {

food **=** option.value;

}

});

**Using the submit Event to Validate Forms with JavaScript**

There are a wide variety of custom validation tests we can write via JavaScript:

Check for the presence or absence of a field



Check the value of a field, and determine if it’s within an expected range, of a specific type, etc.



Confirm that some value is “real” vs. matching an expected format. (e.g., does a user id exist?)



Evaluate a group of input values together as a group. Do they make sense together?



An HTML <form> element exposes the submit event (and onsubmit event property), which we can use to add custom JavaScript code to handle the case that the user is trying to submit a form:

<form id="info-form" name="info" action="/i">

...

</form>

<script>

**var** infoForm **=** document.getElementById('info-form');

* *submit event fired when the user clicks "submit" button* infoForm.onsubmit **= function**() {
  + *Perform extra validation here. When finished validating, return*
  + *either `true` (form is valid) or `false` (form is invalid) to tell*
  + *the browser how to proceed.*

};

* *reset event fired when the user clicks a "reset" button* infoForm.onreset **= function**() {
  + *If you ever need to do extra work to clear a form, do it here.*

};

</script>

Consider the example of a form that asks the user to enter a list of 2-4 numbers. We’d like to allow the user as much freedom to enter this list as possible, and support any style of entry:

<div id="error-msg" class="error hidden"></div>

<form id="info-form" name="info" action="/i">

<label for="number-list">Number list</label>

file:///C:/Users/k/Desktop/여름학기/WEB222\_NDD/Week11/WEB222-Lecture11 Client-side validation.html 5/6

2020. 8. 9. WEB222 - Week 11 | web222

<input id="number-list" name="list" type="text">

</form>

<script>

**var** infoForm **=** document.getElementById('info-form');

infoForm.onsubmit **= function**() {

* *Check if number list is valid, and return true if it is* **if**(validateNumberList()) {
* *Hide the error message if it was displayed previously* hideErrorMessage();

**return true**;

}

* *Number list is invalid, so display error message and return false* showErrorMessage("Number list is invalid: expected 2 to 4 numbers in a list."); **return false**;

};

**function** showErrorMessage(msg) {

**var** errMessage **=** document.querySelector('#error-msg');

* *Remove the hidden class so the error message shows.* errMessage.classList.remove('hidden');
* *Set the error message text*

errMessage.innerHTML **=** msg;

}

**function** hideErrorMessage(msg) {

**var** errMessage **=** document.querySelector('#error-msg');

* *Add the hidden class so the error message goes away.* errMessage.classList.add('hidden');

}

**function** getNumberList() {

*// Get the number-list* **<**input**>**value

**var** list **=** document.querySelector('#number-list').value;

* *Get rid of leading/trailing spaces* list **=** list.trim();
* *Split the string into an array, separated by spaces, commas, or a combo of each* **return** list.split(/[,]+/);

}

**function** validateNumberList() {

**var** list **=** getNumberList();

* *Make sure we have between 2 and 4 elements. If not, return `false`*
* *to indicate this form is invalid (don't submit)*

**if**(**!**(list.length **>=**2 **&&** list.length **<=**4)) {

**return false**;

}

* *Make sure each element in the list is a number* **function** isNumber(n) {

**return !**isNaN(n);

}

* *Loop across every value in the array, and call isNumber() on the value.*
* *Return true if EVERY element passes isNumber, and false otherwise.* **return** list.every(isNumber);

}

</script>



This site is open source. [Improve this page](https://github.com/sictweb/web222/edit/master/weeks/week11/README.md).

file:///C:/Users/k/Desktop/여름학기/WEB222\_NDD/Week11/WEB222-Lecture11 Client-side validation.html 6/6