

INFO/CS 1300: Lab 11 (11/3)

By Grant Storey

Due at the end of your lab section. Demo your work in front of a TA to get your participation credit.

Overview

In this lab, we will be learning about client-side form validation using HTML5 and jQuery.

What You Need

Your laptop and a printed copy of this write-up (optional).

Part 1: Group Member Work Time

You have 10 minutes to check in with your group, look over the Final Project Milestone 2 (if you haven't already) and plan your next steps.

Part 2: HTML5 Forms

Download lab11-activity.zip from CMS and unpack it. Then, open the folder as a project in Atom, and start the PHP server.

You'll see a website for a fake Ithaca nightclub. You will also notice that the main page has a form in it. The form asks for three things; First Name, Last Name, and Email.

Fill out the entries in the form, then click the submit button. You will be directed to a response page, with text that says something like "Thanks, Grant Storey. You are now signed up to hear all the news about Niteclub." (if you put in a name other than "Grant" and "Storey," you'll get a personalized message.

Now, go back to the home page and click the submit button *without filling out any of the fields*. You'll see that you still get redirected to the response page, but it just says "Thanks, ." instead of providing a name. This doesn't seem very good, because the user was able to submit the form without filling anything out.

Form Validation

Luckily for us, HTML5 provides **form validation** (not to be confused with the W3C HTML & CSS validator), which allows us to prevent the user from submitting undesired values to the web server from the HTML form.

The simplest type of form validation is simply making an element *required*. To do this, you just add the "required" attribute to your form elements, like so:

```
<input id="firstName" name="firstName" placeholder="First name" required>
```

Note: You can also type `required="required"` if you prefer your attributes to have values.

Once you have put the **required** attribute on all of your form elements, try to submit the form without filling out anything. You can't! Instead, you get a little pop-over message that says "Please fill out this field." And until you do, you won't be able to submit the form.

Email Validation

However, you might notice another problem with this form. We ask for an email, but if you just type "No thanks" into the box the form will still accept it as valid. Try it out and see if you can submit an invalid email like "No thanks."

To fix this, we need to specify a type for our email input element. For input elements the type attribute can be used to validate the user's data. See https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input#Form_%3Cinput%3E_types for a reference of the different types your input element can be.

Now add `type="email"` to the input tag for the email address (with `id="userEmail"`). Then, refresh the page. If you try to type in a clearly invalid email address, like "No thanks" or "name@site@gmail.com", you will get warnings telling you the email address is invalid.

There are also other kinds of constraints you can put on a form, like minimum length, maximum length, etc. Check out the Mozilla website on form validation: https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms/Form_validation

HTML5 Form Validation: Problems

However, there are a few problems with this type of form validation. First of all, you cannot customize the pop-over at all; it will always have a white background (which doesn't match our color scheme), and you can't write your own error messages. Further, if you open your page in both Firefox and Chrome, you will notice that it renders *differently* in the two browsers. This might be okay or good enough for your Project 3, Milestone 2, but we can do better! Let's use jQuery to show and hide some custom designed error messages to our user.

Part 3: Validating Forms with jQuery.

First, we need to turn off those pop-over error messages so that we can add our own. Add the "novalidate" attribute to your form element, like so:

```
<form method="post" action="formSubmitted.php" id="mainForm" novalidate>
```

You'll notice that this seems to take us backwards a few steps: you can now submit the empty form again! But don't worry; the validation is still available behind the scenes, we just have to access it from jQuery.

Open up **clientValidation.js** in the scripts/ folder. Right now, this script waits for the document to be ready and then does... nothing. Let's start adding some form validation code.

Pseudocode

First, we're going to focus on validation of the first name only, to keep things simple. Before we start coding, we need to plan out what we want to tell the computer to do. So let's write some pseudocode! In a sentence, what we want to happen is "whenever the user tries to submit the form, if the first name field is empty, show the error message and don't allow them to submit." As pseudocode, that is something like:

```
Whenever the user tries to submit the form:
  Check if the first name field is empty.
  If the first name field is empty:
    Show the error message, and don't let the user submit the form.
  If the first name field has text in it:
    Don't show the error message.
```

You might wonder why we both show the error message when the first field is empty *and* hide the error message when the first field has text. But remember, your computer is dumb! Even if it's "obvious" that when we don't show the message, we should hide it, the computer won't know that if you don't tell it explicitly.

Real Code

First, we need to do something "Whenever the user tries to submit the form." To do that, copy the following code inside the `.ready` function:

```
// whenever the user tries to submit the form
$("#mainForm").on("submit", function() {

  // stop the form from submitting
  return false;
});
```

What's going on here? Again, don't worry if you don't understand all of the specifics, but in general what we are saying is "find the element with id '#mainForm' (that's our form), and when the user tries to **submit**, run some jQuery code that returns true or false based on whether we'll allow the form to be submitted.

If we return true, then the browser will submit the form to the server. If we return false, then the browser will not submit the form to the server.

Why do we return false now? This tells jQuery to stop the form from submitting. If you save this file and refresh the page, you'll see that you can't submit the form, no matter what you type! This is because we haven't written the jQuery validation yet.

Javascript Variables and Your First jQuery Validation

Now, let's take a moment to talk about how to create variables in Javascript. To initialize a variable named "minutesInAnHour" with the value 60, you would type:

```
var minutesInAnHour = 60;
```

var (short for “variable”) means that we are creating a new variable, and the = assigns the value 60 to that variable.

If you want to reference that variable at a later point, you just type the variable’s name, without the “var” part. For example:

```
var minutesInTwoHours = 2 * minutesInAnHour;
```

Would give us the number of minutes in two hours. Note that we don’t need “var” before minutesInAnHour this time, since we are only *accessing* an existing variable, not creating a new one. We *do* need the “var” before minutesInTwoHours, since that is a new variable.

Now, we’re going to take our form validation one step further. If we look at our pseudocode, the next thing we want to do is “Check if the first name field is empty.” Instead of our submit function simply saying return false;, lets get some actual validation. Replace return false; in your code with:

```
// check if the first name is empty
var firstNameIsValid = $("#firstName").prop("validity").valid;

return firstNameIsValid;
```

What’s going on here? In the first line, we use the guidelines from above to create a variable that is true if the input with ID “#firstName” is valid and false if is not. Don’t worry about the specifics of .prop("validity").valid. All you have to know is that this returns true if the selector is valid and false otherwise¹; in this case, that means it is *true* when the user has typed something as their first name, and *false* otherwise. In the second line, we return whether or not the input is valid; if it is not valid, the form will not submit.

If you refresh the page, you’ll find this is exactly what happens; if you don’t type anything for the first name, the form won’t submit, but if you do put a name, it will submit.

But this is still not as good as the HTML5 validation above! In the next step, we’ll show how this can be better.

Custom Error Messages

You might have noticed we already have the HTML for an error message for First Name in index.php with the id “#fnameError”; it’s just hidden with CSS using display: none;. In this next step we’ll use jQuery to show the error message whenever it is needed.

If we look at our next piece of pseudocode, it is: “If the first name field is empty, show the error message and don’t let the use submit the form.” However, we aren’t keeping track of whether the first name field is *empty*. We are keeping track of whether it is *valid*. In order to match what we have, we’ll have to adjust a pseudocode a little bit by flipping the order of our statements. So instead of:

¹ If you want to know a little bit more about what’s going on and have some CS background, we first use jQuery to get the “validity” object, which stores a bunch of information about whether the given input is valid. Then, we get the “valid” property from that object, which is a boolean that is true if the input is valid.

If the first name field is empty:
 Show the error message, and don't let the user submit the form.
If the first name field has text in it:
 Don't show the error message.

We have:

If the first name field is **valid** (has text in it):
 Don't show the error message.
Otherwise, if the first name field is empty:
 Show the error message, and don't let the user submit the form.

It's okay to modify your pseudocode like this. Sometimes you have to be flexible like this when writing code, and that is perfectly fine! You often start with your plan and realize you need to make changes to make it work. This is a normal part of coding.

Now that we have our pseudocode, let's turn it into real code. In between the two lines you added above, copy in the following code:

```
// if the first name field is valid (has text in it),
if(firstNameIsValid) {
  // hide the error message
  $("#fnameError").hide();
} else { // (otherwise, if the first name field is empty)
  // show the error message
  $("#fnameError").show();
}
```

This says that, if the first name is valid, we hide the error message. However, if it was *not* valid, we show the error message.

Refresh your page and try submitting without filling in the first name. You should see an error message show up. However, this error message has our own styling on it, and it looks the same in Firefox and Chrome.

Awesome!

Validating Multiple Inputs

However, this still only works for the First Name input. We want to validate *all* the inputs. First, let's change the code a little bit so that we separate out whether the *entire* form is valid versus whether the first name is valid. Your code should look like this inside of the `$("#mainForm").on("submit", function() { block:`

```
// check if the first name is empty
firstNameIsValid = $("#firstName").prop("validity").valid;

// if the first name field is valid (has text in it),
if(firstNameIsValid) {
  // hide the error message
  $("#fnameError").hide();
} else { // (otherwise, if the first name field is empty)
  // show the error message
  $("#fnameError").show();
}

// if the first name isn't filled out, stop the form from being submitted
return firstNameIsValid;
```

Replace that code with this:

```
// assume the form is valid, unless we find an invalid field
formValid = true;

// check if the first name is empty
firstNameIsValid = $("#firstName").prop("validity").valid;

// if the first name field is valid (has text in it),
if(firstNameIsValid) {
    // hide the error
    $("#fnameError").hide();
} else { // (otherwise, if the first name field is empty)
    // show the error
    $("#fnameError").show();
    // and don't let the user submit the form
    formValid = false;
}

// Validation for other fields here

// if the form is valid, let the user submit it; otherwise, block submission
return formValid;
```

If you refresh the page, this works exactly the same. But now, it will be easier to add in more validation, because we have a variable that stores where the entire form is valid as well as a variable that stores whether the first name is valid.

More Validation

Now, it's your turn to validate the last name. Start by writing out pseudocode, like we did above. You'll have to add a new error message to the HTML (with its own ID) and also add a section to your jQuery that validates it. Hint: just copying the code for the first name in both cases and replacing "first" with "last" is a pretty good starting point.

Once you've done this and refreshed the page, you should get error message for *both* the first and last name whenever both are empty. But if only one is empty, the error message should only show up for one.

Now, do the same thing for the email field. Again, you don't have to type everything from scratch; you just have to know which IDs to change.

Once you've got this working, you should only be able to submit if you include a first and last name as well as a valid email; if you try an email like "no email" or leave anything blank, the form won't submit and you will see one of our custom error messages.

Congratulations! You're done with the main part of the lab. Parts 4 and 5 are optional. Don't worry if you don't get to them during lab! You don't *need* to know the information to pass this class. But they are important for making a really good, professional website, so it's a good idea to try this out, or at least read it over and check out the solutions when they are posted.

Part 4: Styling Forms (Optional)

Instant Validation Feedback

jQuery isn't the only thing that has access to whether an input is valid or not; CSS does also. At the top of `all.css`, add the following code:

```
.centerPane input:invalid {  
  border-color: red;  
}
```

This tells the CSS to make the borders of any invalid input red. If you refresh the page and try this out, you'll see that the borders are red whenever the input is empty (or invalid for the email), and white otherwise. This provides your users with instant feedback on the validity of their input!

More generally, you can stick `:invalid` or `:valid` on a class (just like `:hover`) to style only invalid or valid form elements.

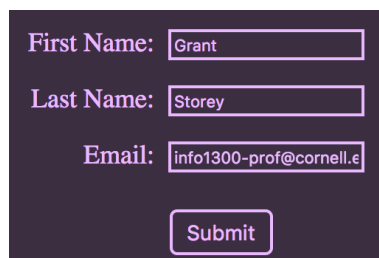
Styling Forms to Match the Page

You may have also noticed that the input and submit buttons, with their black text and white background, don't really match the rest of the page. Luckily, you can style forms with CSS as well!

Use CSS to change the color, background color, border style, and whatever else you want on the input and submit elements to make them match better with the page. It might be useful to know that the background color of the page is `#3A2E40` and the text color is `#EAB9FF`.

You might also want to know how to style the "placeholder text" in the inputs (the slightly gray text that says "First Name", etc). You can do this with the `input::placeholder` selector.

Hopefully, you can style the forms to fit the style of the page better. Mine ends up looking like this:



But really you can do as much/as little as feels right to you.

Part 5: Multiple Error Messages For One Input (Optional)

You might also notice that you currently only provide a single error message, even though the email can have two different problems: empty field or invalid email. In terms of user experience, it is better to be as specific as possible, so letting them know whether the problem is an empty field or an invalid email is a good idea.

This means you have to create two error messages, and activate them under different circumstances. But how do you know which to use? The secret is that the validity object (`$(selector).prop("validity")`) can tell you more than just whether the field is valid (via `.valid`). There is actually a whole range of things, which you can find on the Mozilla HTML5 Form Validation page at

https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms/Form_validation#Constraint_validation_API_properties.

This will show you a variety of properties beyond `.valid` and provide an explanation of what they do. You'll need to use two of them to which error message is appropriate to show for the email ("No email entered" vs "Invalid email entered").

And if this section makes no sense to you, don't worry; you don't need to know this to pass the class. Focus on parts 2 and 3. Also remember to create a plan before you start writing any code. Use pseudocode to help you do this. Don't be afraid to create a new plan if your current approach isn't working. You'll often discover when coding you have to try it several different ways before it works!

Credit

Once you have finished the lab, demo your work to a TA. The TA will check that you completed the lab and will ask you to sign the attendance sheet.

If you run out of time, show the TA what you were able to complete during lab. As long as you were actively working on the lab, during lab, you'll get credit.

If you were working on something else, goofing off, or showed up late, and didn't complete the lab, they won't give you credit. Lab credit is *participation* credit. Working on an assignment for another class, messing around on social media, or showing up late are not considered participation for this course.