META
+LAB

META+LAB PRESENTS:

Client Side Form Validation

A Presentation by Andrew Grano

What is form validation?

When a user fills out a form on your website, you must check that the data entered is correctly formatted before it is submitted to the server and saved in a database.

If the data that the user submitted is incorrectly formatted, the website will respond with a message:

- "This field is required"
- "Please enter your phone number in the format xxx-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"

META +LAB

Why validate forms?

- We need to get the correct data in the correct format
  - Otherwise, our application won't work as intended

- We need to keep our website secure
  - Hackers could attack the website by submitting malicious code through forms

META +LAB

Server-side form validation VS Client-side form validation

- **Server-side form validation**:
  - Occurs on the server AFTER the form has been submitted, but before it has been saved to the database.
  - If the form fails validation, error message(s) will be displayed to the user (after page reload)

- **Client-side form validation:**
  - Occurs in the browser BEFORE the form has been submitted.
  - If any individual field fails validation, an error message can be displayed immediately. We can also provide other visual feedback immediately, such as highlighting invalid form fields.

META +LAB

Should I use client-side form validation or server-side form validation?

Both!

Client-side validation should be implemented because it provides a better experience for your users, as it gives them **immediate feedback**.

Server-side validation is necessary as a **last-line of defense** against malicious data. (Hackers could circumvent client-side validation by disabling javascript)

META +LAB

If we are using client-side form validation to improve the user's experience, then…

What is the best experience we can offer?

Usability testing has found that **displaying an error when the user leaves a field**, and **keeping that error visible until the issue is fixed**, provides the best and fastest user experience.

Also helpful:

- Providing positive validation (feedback when a field is valid)
- Descriptive error messages

The biggest pitfall:

- Prematurely showing an error message before the user has finished interacting with the input.

META +LAB

This is starting to sound complicated…

Why is this important again?

Client-side form validation that is well implemented:

- Increases user satisfaction, as users find these forms easier to process visually
- Decreases the time it takes for a user to complete a form

For you or your company, this means:

- Users are more likely to return to your site or continue using your service
- If the form is a checkout form, reduced cart abandonment results in more $$$$$$$$

META +LAB

Types of Client-Side Form Validation

META
+LAB

HTML5 Built-In Form Validation

— Built into HTML5 is the ability to validate user data without relying on Javascript, instead using validation attributes on form elements...

- required
  - `<input type="text" required>`
- maxlength and minlength
  - `<input type="text" minlength="2" maxlength="5">`
- max and min
  - `<input type="number" min="1" max="10">`
- Pattern
  - accepts any regex
  - `<input type="text" pattern="foo|bar"> //only "foo" or "bar" are valid`
  - `<input type="text" pattern="^\d{5}-\d{4}|\d{5}|[A-Z]\d[A-Z] \d[A-Z]\d$"> //zip code regex`
- Type
  - If type is set to "email" or "URL", they will be validated without the need for pattern attribute
  - `<input type="email">`

META +LAB

HTML5 Built-In Form Validation

https://codepen.io/andrewMETALAB/pen/aRwVmo

Cons:

- Cannot style the look of the error message
- Cannot customize the content of the error message
- Inconsistent experience across browsers
- Breaks rule: “Display error when the user leaves the field”
- Breaks rule: “Keep the error visible until the issue is fixed”

CSS Pseudo Elements

CSS can handle form validation using the pseudo-classes **:invalid** and **:valid**

- These pseudo-classes are applied to inputs, based on the HTML5 validation attributes applied to the input

- Unfortunately, it doesn't work as expected
  - An empty input is set to :invalid on-page load
    - Breaks rule: "Don't prematurely show an error"
  - Validity is re-evaluated on every stroke
    - Breaks rule: "Display error when the user leaves the field"
  - Can't use it to control error messages, only for styling inputs

CSS Pseudo Elements

In the future, pure CSS validation may be possible.

Pseudo elements **:user-invalid** and **:user-error** are not currently supported, but would be applied if a form field is invalid after the user interacted with it.

For now, it's not worth using **:valid** or **:invalid**

The Constraint Validation API

- A javascript API built into modern browsers.
  - Can use a polyfill to support older browsers

- Consists of a set of methods and properties available on each form element, as well as the form itself
  - Must add an attribute of "**novalidate**" onto the <form> to prevent the browser from displaying error messages

The Constraint Validation API

- Call the **checkValidity()** method on a form element to test it against the HTML5 form attributes that you've specified
- You can also run **checkValidity()** on all form elements when the form is submitted

Each form element has a validity property that can tell us if a field is valid or invalid, and, if invalid, what the problem is. Examples:

- **formField.validity.patternMismatch**: will return false if the value does not conform to the pattern attribute specified
- **formField.validity.valueMissing**: will return true if the input has a required attribute does not have a value
- **formField.validity.valid**: will return true if the input is valid

The Constraint Validation API

The **validity** properties and the **checkValidity()** method essentially serve the same purpose.

There are also other methods available:

reportValidity() also checks a field's validity AND shows the native error message if invalid. This is useful, but **not fully supported**.

setCustomValidity('string') allows you to set a custom error message AND sets the form field's validity to true or false depending on if you pass an empty string as the argument or not. **It behaves strangely and forces us to write more complicated code than should be necessary.**

The Constraint Validation API

So, this API doesn't seem to be fully thought through. It is redundant, weird, and lacks consistent browsers support.

Still, we can leverage some of this built-in-browser-goodness to make a lightweight script suitable for basic validation purposes.

Chris Ferdinandi from css-tricks has done just that, by utilizing the validity properties:

https://codepen.io/cferdinandi/pen/eRppxO

This example satisfies our requirements of displaying an error when the user leaves a field, and keeping that error persistent until the issue is fixed,

Third Party JS Validation Libraries

"Form validation is literally the oldest trick in the JavaScript book: when JavaScript was introduced in Netscape 2 it could basically only do form validation. We've had twenty years to get it right, but we didn't."

-Peter-Paul Koch

If you need to implement more complex validation logic, then you will probably need a more powerful API.

Luckily, Javascript developers have been working on solutions to this issue for 20 years, and robust JS validation libraries exist.

The most notable JS Validation Libraries are:

- Validate.js: http://rickharrison.github.io/validate.js/
- jQuery validation plugin: https://jqueryvalidation.org/

Third Party JS Validation Libraries

These libraries **do not rely** on the HTML5 validation attributes. Instead, you set the rules that you want applied to each form field inside of a JSON object. The plugins offer more rules than the constraint Validation API offers, and allow you to create custom rules. The libraries also make it easier to display success messages.

These libraries have existed for a long time, are well supported, well documented, and you can find lots of info about them on stackoverflow.

The downside is that these libraries must be loaded by the client, resulting in longer load times for users.

Conclusion

- We **need** to implement server-side validation for security
- We **should** implement client-side validation to improve user's experience
  - Displaying the error when the user leaves a field
  - Keep the error visible until the issue is fixed
  - Use descriptive error messages
- A combination of the HTML5 form validation attributes along with the Constraint Validation API should suffice for basic forms
- More complex form validation still requires the use of a third-party JS library
- Browsers are inconsistent and slow to roll out new features. This is a constant battle for front-end developers.

META +LAB

Resources

General/Overview:

- https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms/Form_validation
- https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5/Constraint_validation
- https://www.smashingmagazine.com/2009/07/web-form-validation-best-practices-and-tutorials/
- https://css-tricks.com/form-validation-part-2-constraint-validation-api-javascript/

User testing:

- https://baymard.com/blog/inline-form-validation
- https://alistapart.com/article/inline-validation-in-web-forms

Down the Rabbit Hole:

- https://medium.com/samsung-internet-dev/native-form-validation-part-1-bf8e35099f1d

META +LAB

THAT'S A WRAP FOLKS,

THANK YOU!

META
+LAB