Client-side validation

REMEMBER: Client-side JS is NOT secure.

- Fully visible to the user
- Fully alterable by the user

Client-side JS provides **convenience**, not **security**

"Validation" is one such convenience.

What is validation?

- Prevent user from submitting invalid info
- Inform user of needed changes

There are MANY approaches

Does **not replace** server-side validation

But may be the "friendly" version

Client-side JS Validation is very common

Forms are vital part of user interactions

- **Register** new users
- **Login** for existing users
- **Payment Info** if buying
- **Posts** and **Comments**
- **Adding Details** to profile/articles

Front end Validation has advantages

- Fast
- Doesn't scroll page to top
- Doesn't have to fill-in form with existing answers

Standards-based validation

Some HTML standards to automatically validate

- `required` and `pattern` attributes
- `<input type="email">`
- These standards are pretty minimal
    - Very limited logic
    - Very limited UI controls
- Have some accessibility issues (?!)

Most validation is Client-side JS-based

- But server MUST also verify valid data

Simple Example: A required field

Front end validation can be **active** or **passive**

- **Active** - Informs the user of the problem
- **Passive** - User can't move forward until fixed

Active is the better UX

- We cover passive as well so you know how

Example Passive Validation of Required Field

Our chat application allows sending empty messages

- Perhaps you checked for this on server-side?

We can disable the submit button until they have text

Create some Client-side JS

Add to our HTML

```
  <script src="/chat.js"></script>
```

Create a chat.js file **in public/** (static asset)

```
console.log("Hello world");
```

- REMEMBER client-side JS just "text" to the server
- Client-side JS runs on the browser, not the server

Attempt a small change

```
const sendButton = document.querySelector(".send button");
const toSend = document.querySelector(".to-send");

sendButton.disabled = true;
```

If your `<script>` tag is before these elements

- Code will throw an error

`<script>` after `<body>` contents

How to load HTML before JS runs?

- JS could wait for an event that says page is loaded
- `<script>` can have a `defer` attribute (requires `src`)
- `<script>` can be the last element of the `<body>`

An early `<script>` element without `defer`

- "Blocks" the page
- Can't interact with elements not yet in the DOM

Most often: late `<script>` OR `defer`

Yay! Except...

You are polluting the global scope

Put your code in an IIFE:

```
"use strict";
(function () {
  // Your code here
})();
```

Add some complexity

```
"use strict";
( function() {
  const sendButton = document.querySelector(".send button");
  const toSend = document.querySelector(".to-send");

  sendButton.disabled = !toSend.value; // Before any typing

  toSend.addEventListener('input', (e) => {
    sendButton.disabled = !e.target.value;
  });

})();
```

Server Enforcement Required!

Remember a user can bypass JS or the browser

- Webdevs often do this with broken validation

If it is true requirement

- Server must enforce

Never assume front end validation works

Active validation

You should tell the user

- There is a problem
- How to fix the problem

Populate an error message

Example

On login form, username will be **allowlisted**

- Let's use `A-Z`, `a-z`, `0-9`, `_`
- "Allowlisted" = only certain characters allowed

If username does not pass check

- JS will populate an error message for user
- JS will prevent form submission

What Event?

Many options!

- `blur` event fires when field loses focus
- `input` event fires when value CHANGES
- `keydown` and `keyup` events fire on typing
    - down before character is added to value
        - can prevent add!
    - up after character is added to value
- `click` event on buttons
- `<form>` fires a `submit` event
    - Good for a final check of everything

"Best" UX still being decided

We've all had frustrations

- A field broken up to multiple parts
- Error messages after you leave the field
- Error messages before you even type
- Unclear if/where error is

Basic Example: on Submit

```
<form class="login" action="/login" method="POST">
<p class="error"></p>
<label>
  Username: <span class="required">*</span>
  <input class="username" name="username">
</label>
<button class="to-login">Login</button>
</form>
```

```
const formEl = document.querySelector('.login');
const usernameEl = document.querySelector('.username');
const errorEl = document.querySelector('.error');

formEl.addEventListener('submit', (e) => {
  const username = usernameEl.value;
  if( !username.match( /^[A-Za-z0-9_]+$/ ) ) {
    e.preventDefault();
    errorEl.innerText = 'A specific message goes here';
  }
});
```

A Lot of Notes!

- IIFE and 'use strict' skipped for space
- class names in real work probably more detailed
- `El` suffix
    - Usually "hungarian" notation undesirable
    - DOM nodes (elements) different than values
- Regex a whole thing (see `readings/js/regex.md`)
- Required vs Bad value?
- Good messages aren't easy!
- References to nodes break if DOM changed
- Soon use a different way to alter DOM!

This is to learn the syntax/options

We will soon learn a different approach

- Not changing specific elements
    - Instead recreating the HTML output
- More similar to what we do server-side
- More similar to what React does

Be prepared to handle that change in style

Are you requiring JS?

Always consider if you're **requiring** client-side JS

JS may or may not be a reasonable requirement

You should consider the cost/benefits

- You are not your audience - code for THEM

Progressive Enhancement

- It works without JS
- Or if some features not supported by browser
- Nicer experience if you have JS/those features