Centralized Validation







Centralized JavaScript Validation Explained Simply

Web developers often end up writing **form validation logic repeatedly** — one script per form. This approach is hard to maintain and error-prone. The concept below demonstrates how to use **one reusable JavaScript file** that automatically validates all form fields, based on small hints added to HTML inputs.

In this demo: The example is built inside an ASP.NET WebForms app to show that this idea works even in legacy frameworks. The same principle applies to any web framework — React, Angular, MVC, Razor, or plain HTML.

The Core Concept

Each input field carries a data-type attribute describing which rules to apply. The central JavaScript file reads these attributes dynamically, performs validation, and shows errors — no custom per-page JS required.

```
<input id="emailInput" data-type="email|required" />
<span id="emailInput-error" class="error"></span>
<input id="ageInput" data-type="number|min:18" />
<span id="ageInput-error" class="error"></span>
```

How It Flows Behind the Scenes

The Central Engine (Simplified)

```
// validation.js (simplified idea)
window.ValidationEngine = {
  rules: {},
  registerRule: function(name, fn) {
    this.rules[name] = fn;
  },
  validateField: function(el) {
    const rules = el.dataset.type.split("|');
    for (let r of rules) {
        let [rule, arg] = r.split(":');
        if (this.rules[rule] && !this.rules[rule](el.value, arg)) {
            document.getElementById(el.id + "-error").textContent = "Invalid " + rule;
            return false;
        }
    }
    document.getElementById(el.id + "-error").textContent = "";
    return true;
},
```

```
validateAll: function(formSelector) {
  let valid = true;
  document.querySelectorAll(formSelector + " [data-type]").forEach(el => {
    if (!this.validateField(el)) valid = false;
  });
  return valid;
};
```

Common Built-In Rules Example

- required field must not be empty
- email must be a valid email address
- number only digits allowed
- min:18 number must be ≥ 18

Using It in a Real ASP.NET Form

Here's how it connects to a Web Form, for example your **Employee Form**:

```
<asp:TextBox ID="txtName" runat="server" data-type="required" />
<span id="txtName-error" class="error"></span>
<asp:TextBox ID="txtEmail" runat="server" data-type="email|required" />
<span id="txtEmail-error" class="error"></span>
<asp:Button ID="btnSave" runat="server" Text="Save" OnClientClick="return Validation"</pre>
```

The button will only submit if all rules return **true**.

5 Why It's Framework-Agnostic

- Works in **ASP.NET Web Forms** (classic or modernized)
- Same logic applies in **React, Angular, Vue**, etc.
- No backend dependency purely **front-end logic**
- Easy to extend add new rules once, used everywhere

Typical Integration Script

```
$(function() {
$("[data-type]").on("blur input change", function() {
```

```
ValidationEngine.validateField(this);
});

$("#submitButton").on("click", function() {
  if (!ValidationEngine.validateAll("#myForm")) {
    alert("Please fix validation errors first.");
    return false;
  }
});
```

Educational Summary:

This demonstration shows how **centralized validation** lets you manage all input checks from a single JS file. The approach was showcased here in an **ASP.NET WebForms** app — proving that even old frameworks can benefit from modern, generic JS techniques.

The concept was proposed by <u>Rohit Ashok Yadav Linked In</u>and co-created with **ChatGPT (GPT-5)** to help developers modernize legacy systems while maintaining simplicity and reusability.

© 2025 My Company. All rights reserved.