Forms Handling (mini build)

A complete small form: read every value, validate empty fields and emails live, show errors, a success state, and the submitted data on the page.

Introduction

This is the **mini build**: the DOM skills from lesson 3.13 and the event skills from lesson 3.14, assembled into one complete form — the front-end half of almost every real app you will ever make. You will read every field, check the values, show clear errors, and finish with a success state and the submitted data rendered on the page. *(Roman Urdu: Ye chhota sa project pichli do lessons ko jod kar aik mukammal form banata hai.)*

Key points

- **The submit flow has a fixed order:** `e.preventDefault()` first, then **read** every value, then **validate**, then either show errors or show success. Keep this order in every form you build.
- **Read values by id:** `document.getElementById("name").value` gives you the text inside that field at the moment of submit. `.value` is for form fields; `.textContent` is for normal elements.
- **Validate in a planned order:** here, name first (empty means error), then email (empty, or no `@` inside, means error). Run **all** the checks and show **all** the errors together — never make the user fix one error, submit, and only then discover the next one.

  (Roman Urdu: Saari ghaltiyan aik saath dikhao, taakeh user ko baar baar submit na karna paray.)
- **Each error lives under its field:** a small empty element under every input is waiting to hold that field's message, so the user sees the problem exactly where the fix belongs.
- **Live errors clear while fixing:** an `input` listener on each field re-checks that one field as the user types, so a correct fix removes the error message immediately — no second submit needed.
- **Success is a state change:** when everything passes, hide (or reset) the form, show a success message, and **render the submitted data** on the page as a small summary card.
- **Honest note:** front-end validation is a **help for the user**, not real security. Anyone can bypass it, so a real backend must check every value again — that part comes later, in the real-apps lessons.

  (Roman Urdu: Front-end checking sirf user ki madad hai — asal hifazat hamesha backend par hoti hai.)

Code examples

Example 1 — Read every value on submit

```
<!DOCTYPE html>
<html>
<head>
  <title>Course Form - Step 1</title>
</head>
<body>
  <form id="courseForm">
    <label>Name: <input type="text" id="name"></label>
    <br><br>
    <label>Email: <input type="text" id="email"></label>
    <br><br>
    <label>Course:
      <select id="course">
        <option>JavaScript</option>
        <option>React</option>
        <option>Flutter</option>
      </select>
    </label>
    <br><br>
    <button type="submit">Submit</button>
  </form>
  <p id="output"></p>

  <script>
    const form = document.getElementById("courseForm");
    const output = document.getElementById("output");

    form.addEventListener("submit", function (e) {
      e.preventDefault();

      const name = document.getElementById("name").value;
      const email = document.getElementById("email").value;
      const course = document.getElementById("course").value;

      output.textContent = name + " | " + email + " | " + course;
    });
  </script>
</body>
</html>
```

What it does: On submit, the reload is stopped first. Then each field is read by its id with `.value` — text boxes give their typed text, and the dropdown gives the chosen course. The three values are joined and written onto the page, so you can see exactly what the form collected.

What you will see: type the name `Ahmed`, the email `ahmed@example.com`, choose `React`, and press Submit — the page does not reload, and the line `Ahmed | ahmed@example.com | React` appears under the form.

Example 2 — All errors together, clearing live as you type

```
<!DOCTYPE html>
<html>
<head>
  <title>Course Form - Step 2</title>
  <style>
    .error { color: red; }
  </style>
</head>
<body>
  <form id="courseForm">
    <label>Name: <input type="text" id="name"></label>
    <p class="error" id="nameError"></p>
    <label>Email: <input type="text" id="email"></label>
    <p class="error" id="emailError"></p>
    <button type="submit">Submit</button>
  </form>
  <p id="output"></p>

  <script>
    const form = document.getElementById("courseForm");
    const nameBox = document.getElementById("name");
    const emailBox = document.getElementById("email");
    const nameError = document.getElementById("nameError");
    const emailError = document.getElementById("emailError");
    const output = document.getElementById("output");

    function checkName() {
      if (nameBox.value.trim() === "") {
        nameError.textContent = "Name is required";
        return false;
      }
      nameError.textContent = "";
      return true;
    }

    function checkEmail() {
      if (emailBox.value.trim() === "") {
        emailError.textContent = "Email is required";
        return false;
      }
      if (!emailBox.value.includes("@")) {
        emailError.textContent = "Email must contain @";
        return false;
```

```
      }
      emailError.textContent = "";
      return true;
    }

    nameBox.addEventListener("input", checkName);
    emailBox.addEventListener("input", checkEmail);

    form.addEventListener("submit", function (e) {
      e.preventDefault();
      const nameOk = checkName();
      const emailOk = checkEmail();
      if (nameOk && emailOk) {
        output.textContent = "Form is valid: " + nameBox.value + " | " + emailBox.value;
      } else {
        output.textContent = "";
      }
    });
  </script>
</body>
</html>
```

What it does: Each check is its own small function that writes its message into the error element under its own field and returns `true` or `false`. On submit, **both** checks always run, so both errors can appear together. The same functions are also attached to the `input` event, so the moment a field becomes correct, its error disappears while the user is still typing.

What you will see: press Submit with both fields empty and two red messages appear at once: `Name is required` and `Email is required`. Type a name and its error clears live; type an email without `@` and the message changes to `Email must contain @`; add the `@` and it clears. Submit with both correct and the line `Form is valid:` appears with your name and email.

Example 3 — Success state and the summary card

```
<!DOCTYPE html>
<html>
<head>
  <title>Course Form - Complete</title>
  <style>
    .error { color: red; }
    .card { border: 1px solid #cccccc; padding: 12px; width: 260px; }
    #successBox { display: none; }
  </style>
</head>
<body>
  <form id="courseForm">
    <label>Name: <input type="text" id="name"></label>
    <p class="error" id="nameError"></p>
    <label>Email: <input type="text" id="email"></label>
    <p class="error" id="emailError"></p>
    <label>Course:
      <select id="course">
        <option>JavaScript</option>
        <option>React</option>
        <option>Flutter</option>
      </select>
    </label>
    <br><br>
    <button type="submit">Submit</button>
  </form>

  <div id="successBox">
    <h2>Form submitted successfully</h2>
    <div class="card">
      <p>Name: <span id="cardName"></span></p>
      <p>Email: <span id="cardEmail"></span></p>
      <p>Course: <span id="cardCourse"></span></p>
    </div>
  </div>

  <script>
    const form = document.getElementById("courseForm");

    form.addEventListener("submit", function (e) {
      e.preventDefault();

      const name = document.getElementById("name").value.trim();
```

```
      const email = document.getElementById("email").value.trim();
      const course = document.getElementById("course").value;
      const nameError = document.getElementById("nameError");
      const emailError = document.getElementById("emailError");

      let valid = true;
      if (name === "") {
        nameError.textContent = "Name is required";
        valid = false;
      } else {
        nameError.textContent = "";
      }
      if (email === "" || !email.includes("@")) {
        emailError.textContent = "Enter a valid email with @";
        valid = false;
      } else {
        emailError.textContent = "";
      }
      if (!valid) {
        return;
      }

      document.getElementById("cardName").textContent = name;
      document.getElementById("cardEmail").textContent = email;
      document.getElementById("cardCourse").textContent = course;

      form.style.display = "none";
      document.getElementById("successBox").style.display = "block";
    });
  </script>
</body>
</html>
```

What it does: This is the complete build. Submit runs the same read-and-validate flow: any problem writes its error under its field and stops the submit. If everything passes, the submitted values are copied — safely, with `textContent` — into a summary card, the form hides itself, and the hidden success box appears. The data the user sent is now rendered on the page as their receipt.

What you will see: submit with an empty name and you stay on the form with `Name is required` in red. Fix everything and submit again — the form disappears and a success box appears with the heading `Form submitted successfully` and a card showing your name, your email and your chosen course.

Quick recap

- Every form follows the same flow: `preventDefault` → read values → validate all → errors together, or success.
- Errors belong under their own field, and `input` listeners clear them live while the user fixes the field.
- A good finish hides or resets the form, shows success, and renders the submitted data — but the real check still belongs to the backend later.

Common mistakes

1. **Showing only one error at a time** — stopping at the first failed check forces the user into a submit-fix-submit loop for every single field. The fix: run every check, collect every message, and show them all together.
2. **Reading .textContent from an input** — form fields hold their text in `.value`; `textContent` on an input is empty, so your code "reads" nothing and validation fails on correct data. The fix: `.value` for inputs, selects and textareas; `textContent` for headings, paragraphs and spans.
3. **Trusting front-end validation as security** — it runs in the user's own browser, where anyone can change or bypass it, so bad data can still reach a server that does not check again. The fix: treat this validation as user help only, and repeat every check on the backend when you build one.

Practice

1. Add a third field to the complete form: a phone box. Validate it in the same style — empty means `Phone is required`, and fewer than 10 characters means `Phone looks too short`. Show its error under the field, clear it live on `input`, and add the phone to the summary card.
2. Add a `Reset` experience to the success state: a button inside the success box that hides the success box, shows the form again, and clears all three fields and all error messages, ready for a fresh entry.

Practice quiz

1. What is the fixed submit flow for every form?

- Validate, then preventDefault, then read
- preventDefault first, then read every value, then validate, then show errors or success
- Read, then preventDefault, then save
- Save first, validate later

Answer: preventDefault first, then read every value, then validate, then show errors or success

2. How do you read the text inside an input field?

- With .textContent, like a heading
- With .value, which is for form fields; .textContent on an input is empty

- With .innerHTML
- Inputs cannot be read, only written

Answer: With .value, which is for form fields; .textContent on an input is empty

3. Is front-end validation real security?

- Yes, nothing can bypass it
- No, it is help for the user only; anyone can bypass it, so a real backend must check every value again
- Yes, but only for email fields
- No, it does nothing at all

Answer: No, it is help for the user only; anyone can bypass it, so a real backend must check every value again

---

Watch the Code Scope channel: https://www.youtube.com/@CodeScopePK — this lesson's own video joins the channel when it is recorded.

Read online: https://scopetreasure.com/notes/js-forms-handling · More free tools and notes at https://scopetreasure.com