TDL SCHOOL

Web A11y for Developers

Day 3: Accessible Forms

TDL SCHOOL

Agenda

Day 1: Introduction

Day 2: Components

Day 3: Accessible Forms

Day 4: Best Practices

Accessible forms

Forms are one of the **most important and complicated** pieces of any website or application – from an accessibility standpoint.

Building accessible forms takes an entire team, a number of **design decisions** to make and **development hurdles** to cross.

There are **a lot of moving parts**. Let’s take a look at them.

Labeling

Every form has at least one input (text, checkbox, dropdown etc). One thing is true about all of them - **they must have a label.**

Labels are the primary names of the form inputs and let users know that **“This is an input and this is what it’s called”**.

The only element we need is the **<label>** element.

Labeling

Explicit labeling

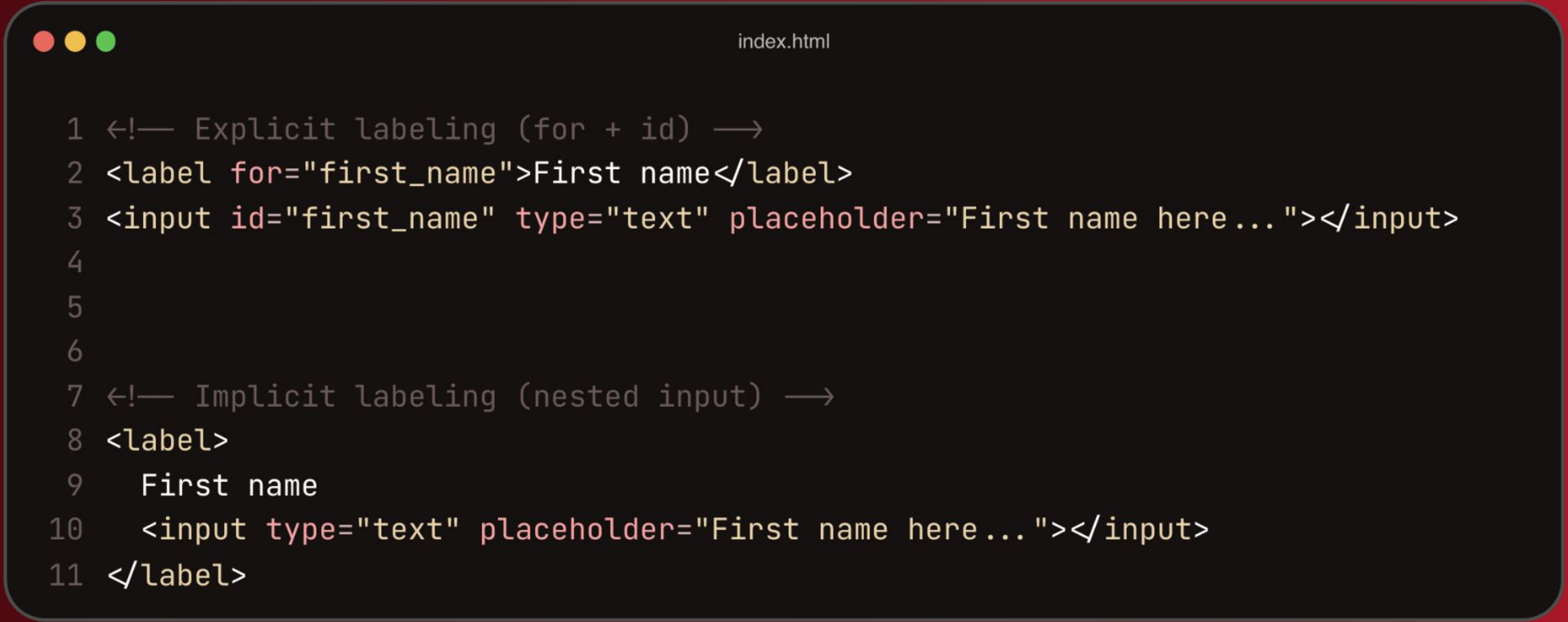
Using the **`for` attribute** on the label and connecting it with the **ID of the associated input**.

Example: When we cannot put them close together in the HTML.

Implicit labeling

Nesting the input inside the <label> element.

Example: When we need simple labeling, no need for IDs etc.



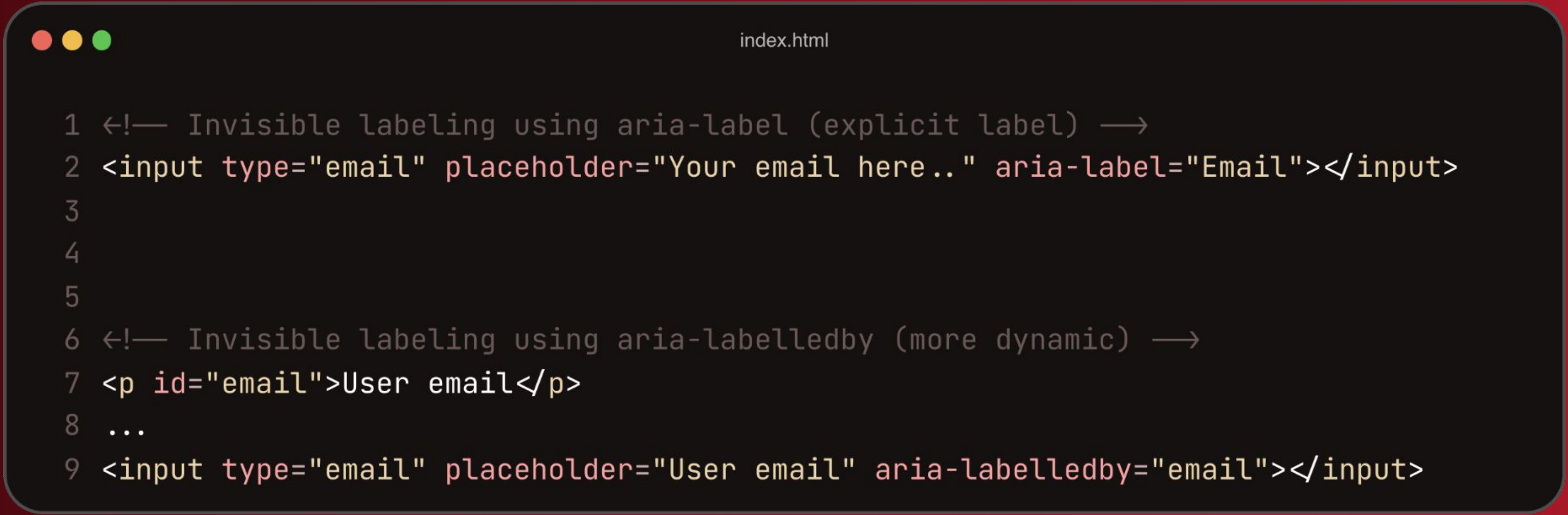
```html
<!-- Explicit labeling (for + id) -->
<label for="first_name">First name</label>
<input id="first_name" type="text" placeholder="First name here..."></input>



<!-- Implicit labeling (nested input) -->
<label>
  First name
  <input type="text" placeholder="First name here..."></input>
</label>
```

Invisible labels

We can add invisible labels using the `**aria-label**` and `**aria-labelledby**` attributes.

There are scenarios where it is not feasible or possible to show a visible label for a number of reasons:

- **Spreadsheet interfaces** (Google Sheets, Excel)
- **URL search bars** on Browsers
- **Quick filters** where the **context around them** explains their role



```html
<!-- Invisible labeling using aria-label (explicit label) -->
<input type="email" placeholder="Your email here.." aria-label="Email"></input>



<!-- Invisible labeling using aria-labelledby (more dynamic) -->
<p id="email">User email</p>
...
<input type="email" placeholder="User email" aria-labelledby="email"></input>
```

“Web accessibility requires a context-based approach to solving problems.”

“There is no silver bullet solution to everything.”

- Anyone that’s solved an a11y issue ever

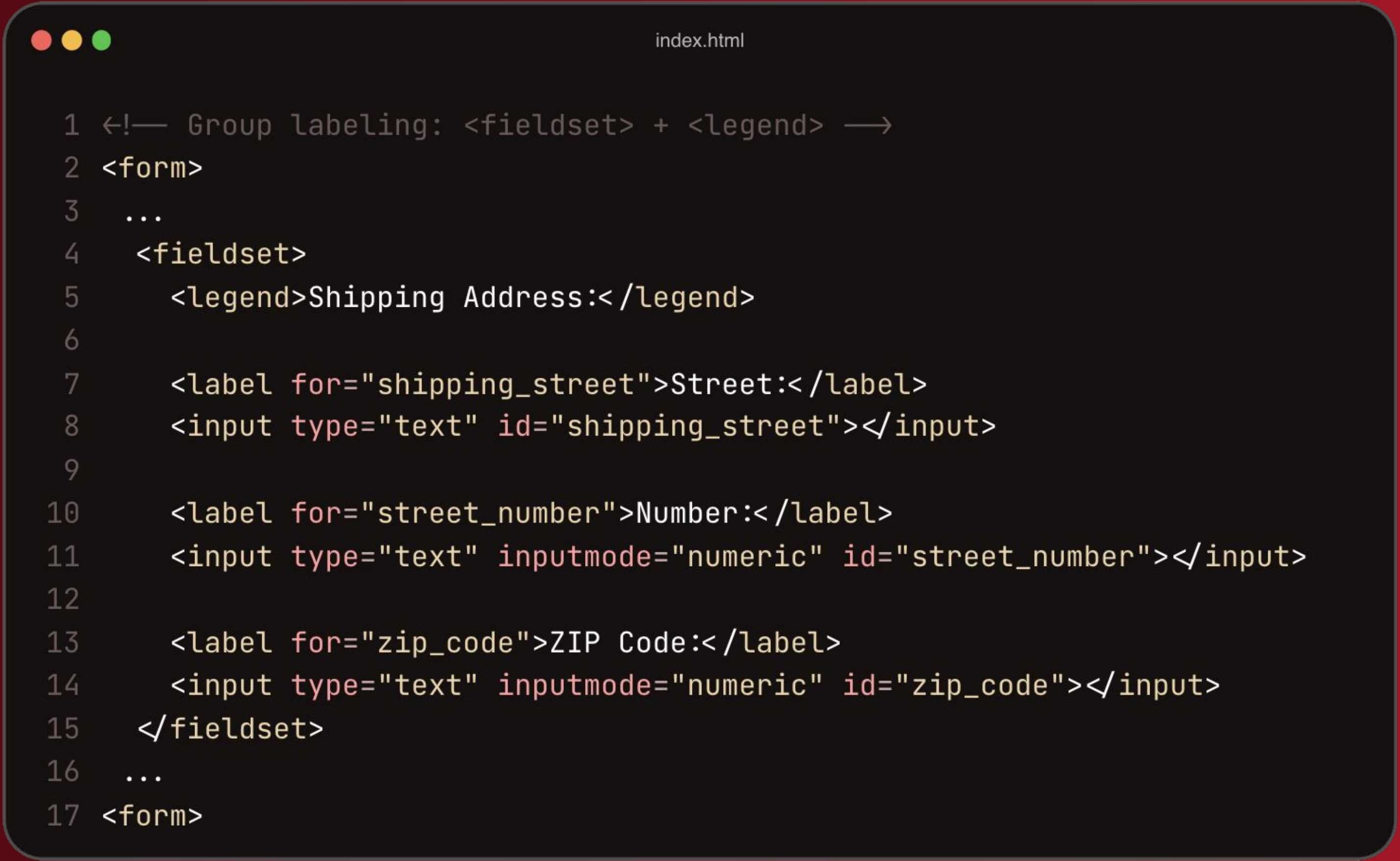
Group labeling

Sometimes multiple inputs are logically associated with each other and they may require a common label associated with that entire section of inputs.

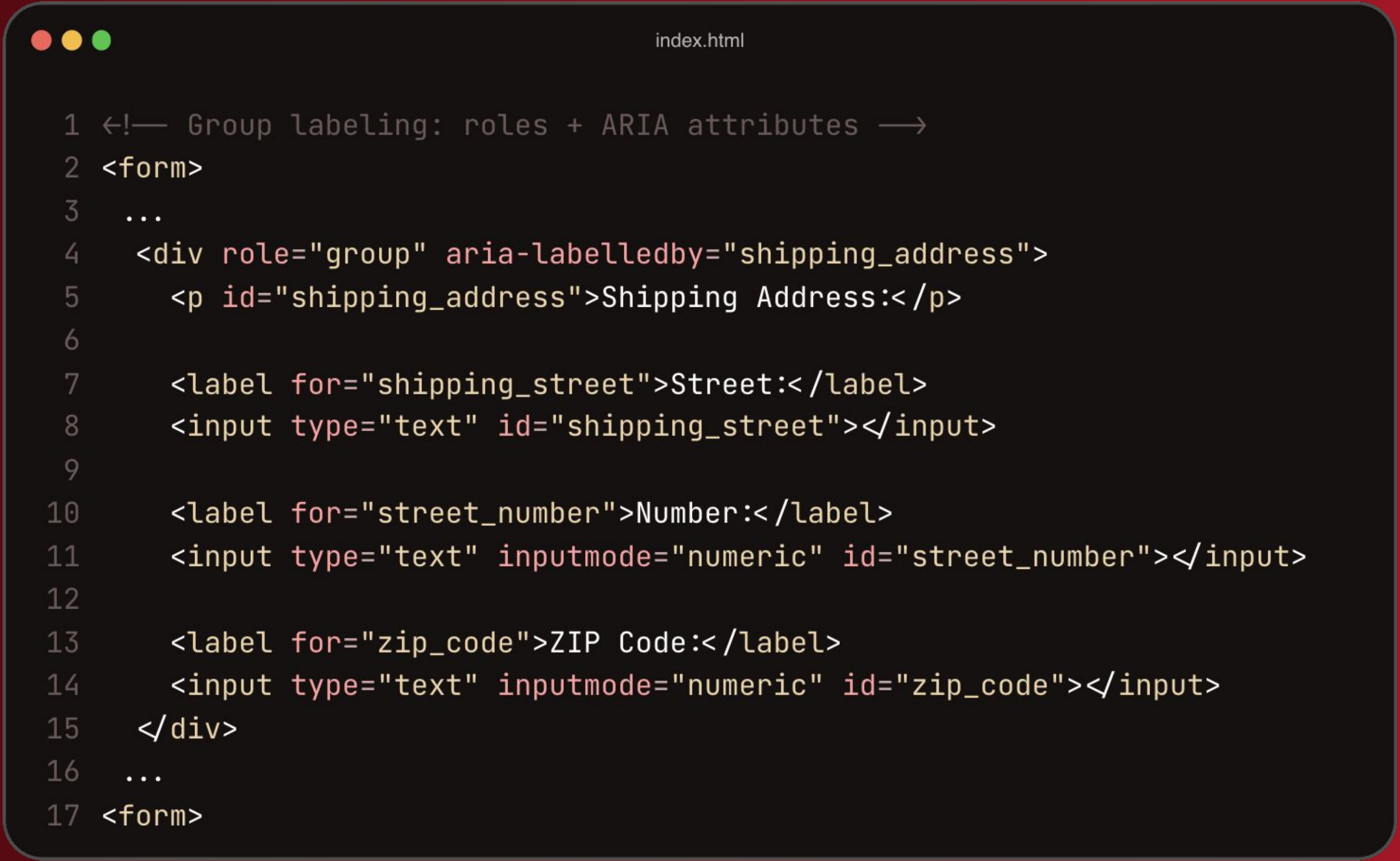
Example

Shipping address section that needs your **street name, number, city, zip code** etc.

We can achieve in 2 ways: **using the <fieldset> and <legend> elements** or **using roles and ARIA attributes**.


index.html

```html
<!-- Group labeling: <fieldset> + <legend> -->
<form>
 ...
  <fieldset>
    <legend>Shipping Address:</legend>

    <label for="shipping_street">Street:</label>
    <input type="text" id="shipping_street"></input>

    <label for="street_number">Number:</label>
    <input type="text" inputmode="numeric" id="street_number"></input>

    <label for="zip_code">ZIP Code:</label>
    <input type="text" inputmode="numeric" id="zip_code"></input>
  </fieldset>
 ...
<form>
```


index.html

```html
<!-- Group labeling: roles + ARIA attributes -->
<form>
 ...
  <div role="group" aria-labelledby="shipping_address">
    <p id="shipping_address">Shipping Address:</p>

    <label for="shipping_street">Street:</label>
    <input type="text" id="shipping_street"></input>

    <label for="street_number">Number:</label>
    <input type="text" inputmode="numeric" id="street_number"></input>

    <label for="zip_code">ZIP Code:</label>
    <input type="text" inputmode="numeric" id="zip_code"></input>
  </div>
 ...
<form>
```

Hints & Descriptions

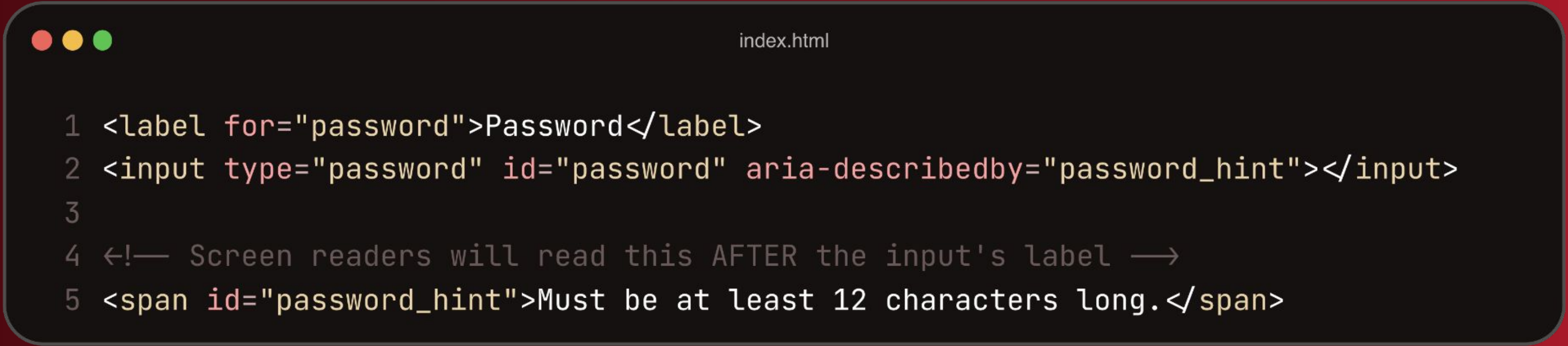
Some inputs require **additional information alongside them.**

We may need to give users more context on:

- How should they fill in the information?
- How should that information be formatted?
- What is the meaning of that information?

This is what's called **"hint messages"** or **"descriptions"**.

We can add them using the **`aria-describedby` attribute**.



```html
<label for="password">Password</label>
<input type="password" id="password" aria-describedby="password_hint"></input>

<!-- Screen readers will read this AFTER the input's label -->
<span id="password_hint">Must be at least 12 characters long.</span>
```

Errors & Live regions

In the event that **users enter invalid data**, or **the submission of our form fails** - we need to be able to tell them that there is a problem with the form.

Since **this information is not present on the form** when users start filling it in - **how do we notify them that an error happened**?

How do we tell screen readers that something’s wrong with the form?

Live regions

Mechanism that allows assistive technologies to **announce changes on the page**.

What should we do?

Mark a section of the page as a live region.

How do we do that?

Either explicitly set an **`aria-live` attribute** or **specific roles with implicit live region attributes**.

Live regions

aria-live

Adding this attribute to an element **marks it as a live region.** Any changes to that element's content will be announced by ATs.

There are 3 ways this change can be announced:

1. **"assertive"** – announced **immediately**, interrupting the current read out of content
2. **"polite"** – announced **after the screen reader is done reading** other content
3. **"off"** – announced **only if the user has already moved focus within the live region**

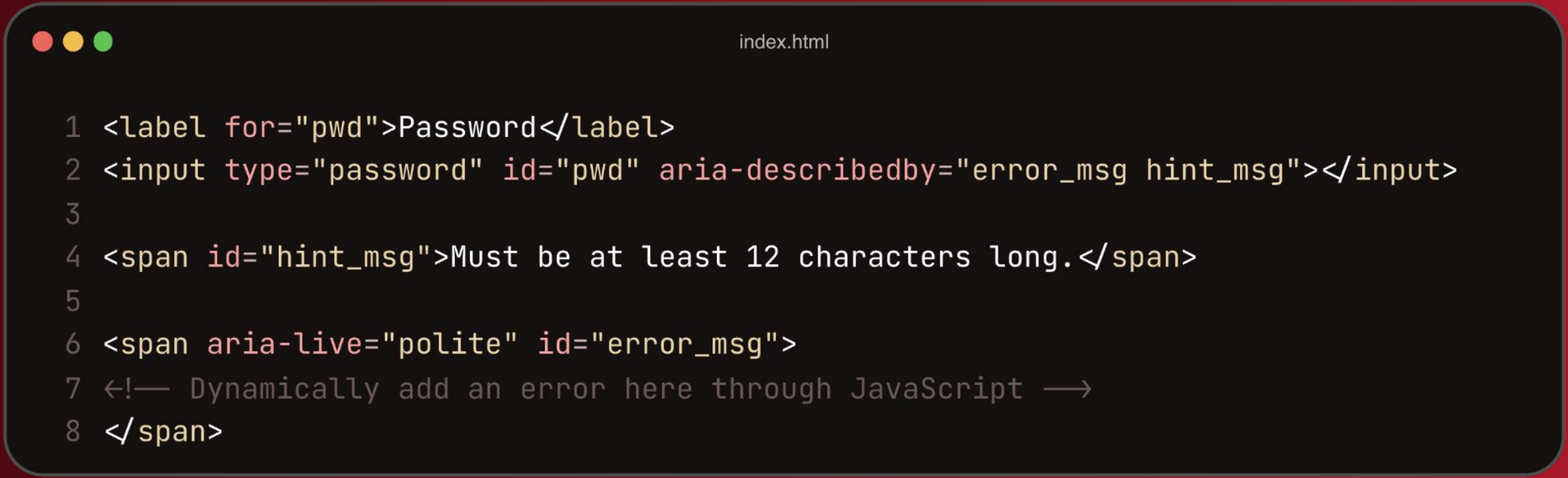
Live regions

Roles as implicit live regions

The role of **“alert”** automatically makes an element a live region with an **assertive** setting.

The role of **“status”** automatically makes an element a live region with a **polite** setting.

Check out some other roles that can do this too.



```html
<label for="pwd">Password</label>
<input type="password" id="pwd" aria-describedby="error_msg hint_msg"></input>

<span id="hint_msg">Must be at least 12 characters long.</span>

<span aria-live="polite" id="error_msg">
<!-- Dynamically add an error here through JavaScript -->
</span>
```

Error messages

Inline error messages

Appear **next to the input** that they are associated with.

Used for showing **descriptive errors**, often **offering a potential solution** within the message itself.

Can be used in different scenarios:

- As the user is **typing in the field** – can be distracting
- As soon as the user **leaves the field (on "blur")** – can be confusing and distracting
- **After the form has been submitted** – generally good UX

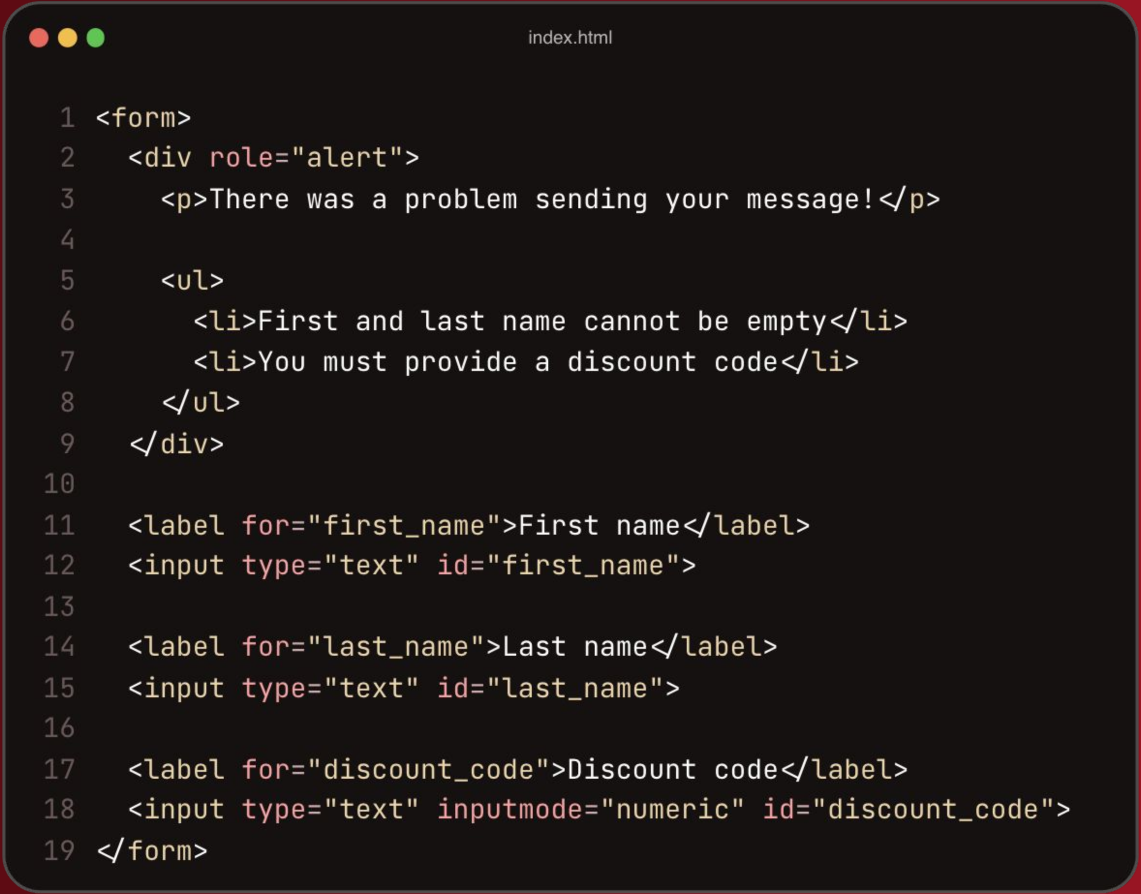
Error messages

Summary error messages

Appear at the **start of the form**, within a **global live region**.

Used for showing a list of more **general error messages**.

They always appear **after the form has been submitted**.


index.html

```html
<form>
  <div role="alert">
    <p>There was a problem sending your message!</p>

    <ul>
      <li>First and last name cannot be empty</li>
      <li>You must provide a discount code</li>
    </ul>
  </div>

  <label for="first_name">First name</label>
  <input type="text" id="first_name">

  <label for="last_name">Last name</label>
  <input type="text" id="last_name">

  <label for="discount_code">Discount code</label>
  <input type="text" inputmode="numeric" id="discount_code">
</form>
```

Error messages

Best Practices

We always want to make error messages **distinguishable from the rest of the form, inputs and descriptions**.

Never rely on color only!
Get into the habit of showing an **error icon next to the message**.

We also need to make the error message content **clear and understandable**. Ideally include a way to solve the problem.

Break time!

We’ll be right back.

Form validation

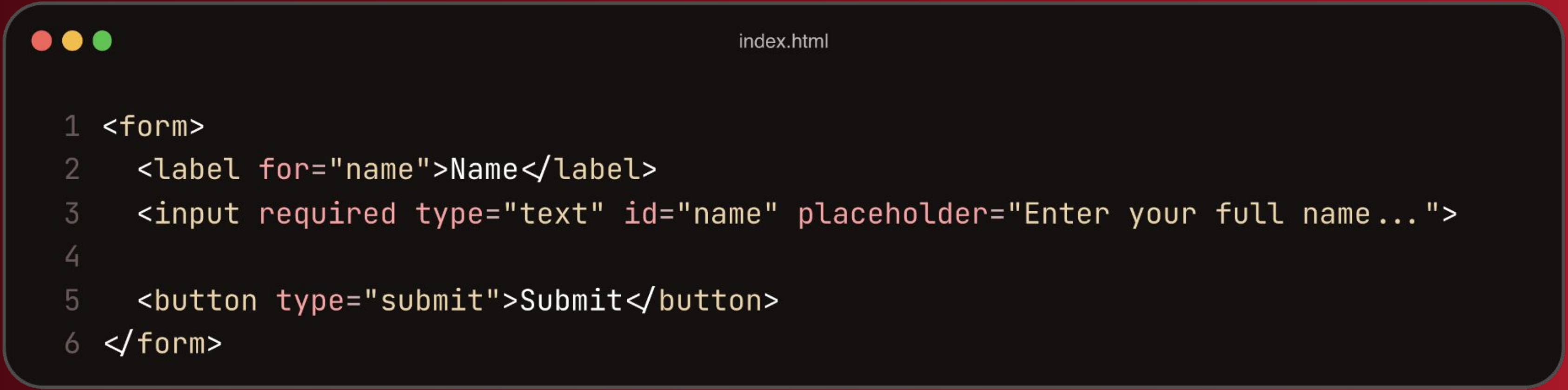
We can validate our forms either on the **client** or on the **server**.

For security reasons, it is always a good idea to **validate it in both places**.

Required fields

HTML's built-in validation system allows us to put different constraints on inputs (min/max length, pattern, required etc.).

The **`required` attribute** (constraint) makes sure **an input must be filled with valid information** before it can be submitted.


index.html

```html
<form>
  <label for="name">Name</label>
  <input required type="text" id="name" placeholder="Enter your full name...">

  <button type="submit">Submit</button>
</form>
```

Form validation

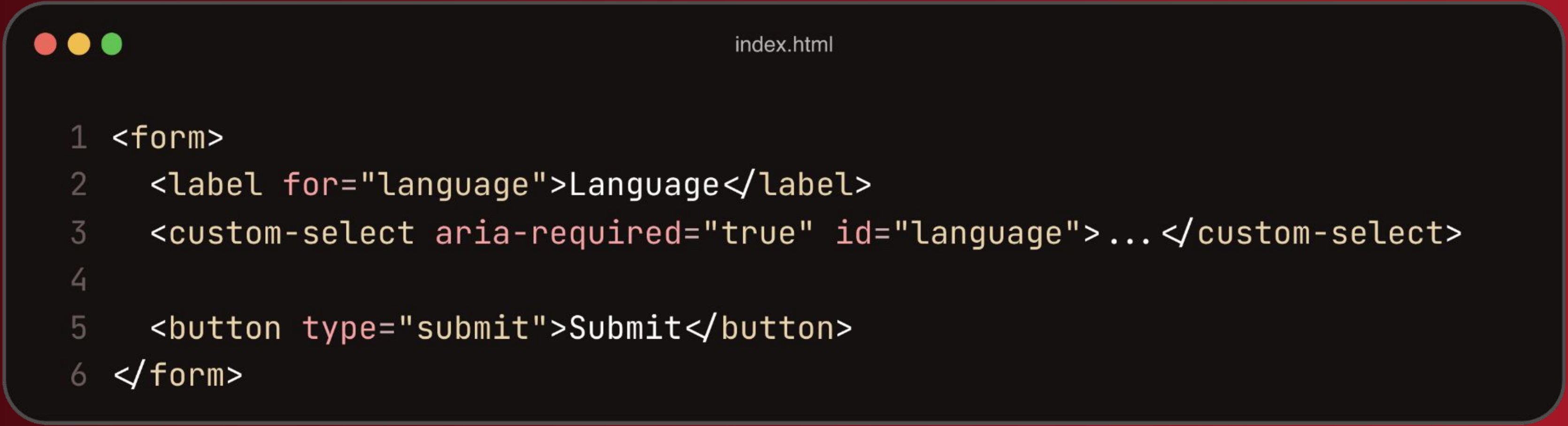
Required fields (contd.)

The native `required` attribute works great with **native inputs, textareas, select inputs, checkboxes** etc.

Screen readers will announce the fact that a field is required.

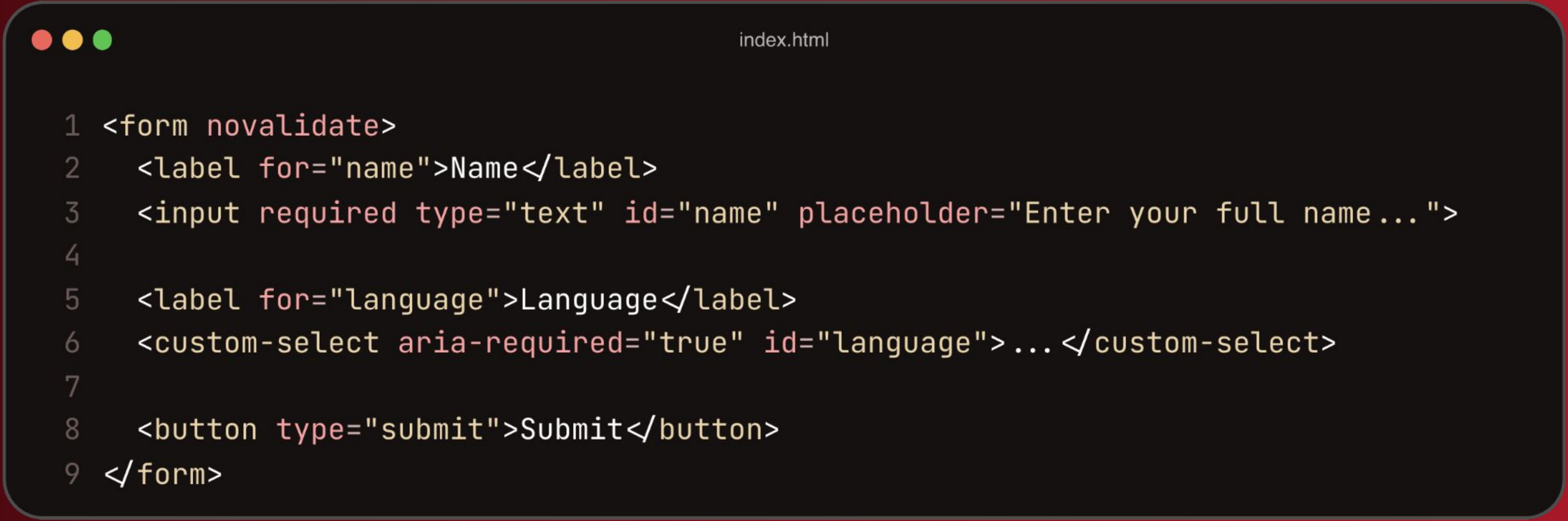
For **custom input components**, we should use **`aria-required`** instead.

This will NOT trigger the HTML’s built-in validation, but will make sure our component is announced as “required”.



index.html

```html
<form>
  <label for="language">Language</label>
  <custom-select aria-required="true" id="language">...</custom-select>

  <button type="submit">Submit</button>
</form>
```


index.html

```html
<form novalidate>
  <label for="name">Name</label>
  <input required type="text" id="name" placeholder="Enter your full name...">

  <label for="language">Language</label>
  <custom-select aria-required="true" id="language">...</custom-select>

  <button type="submit">Submit</button>
</form>
```

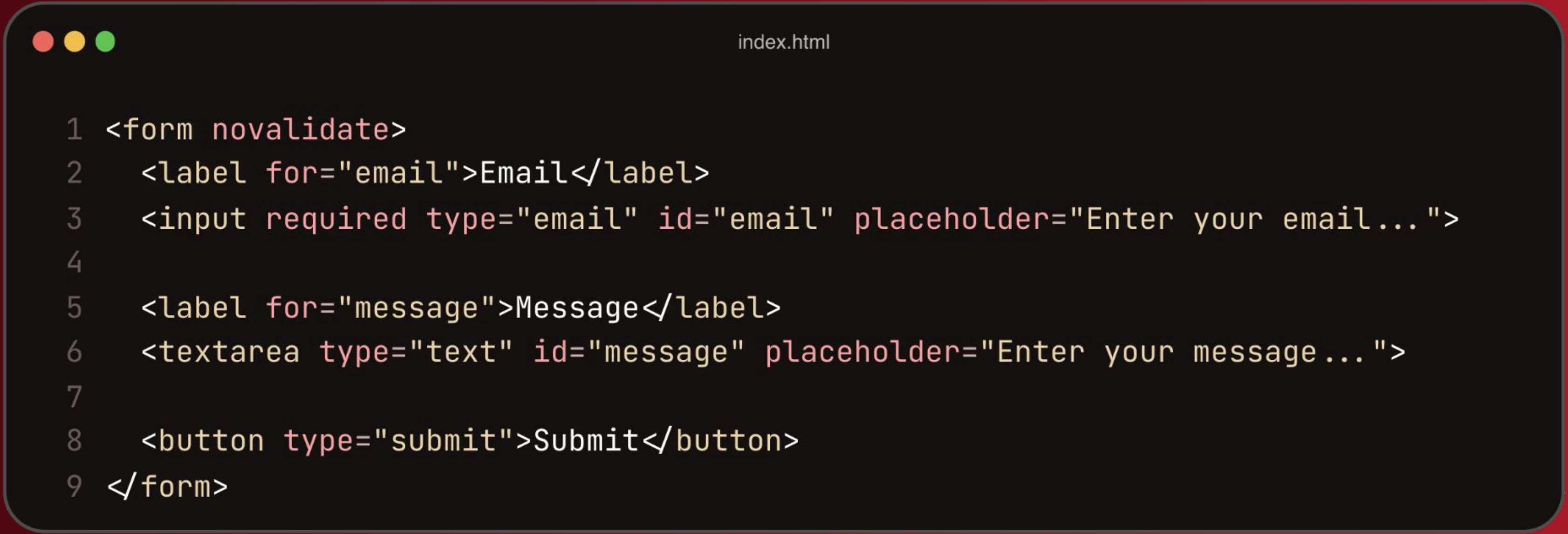
Form submission

We always want to have **multiple ways our form can be submitted.**

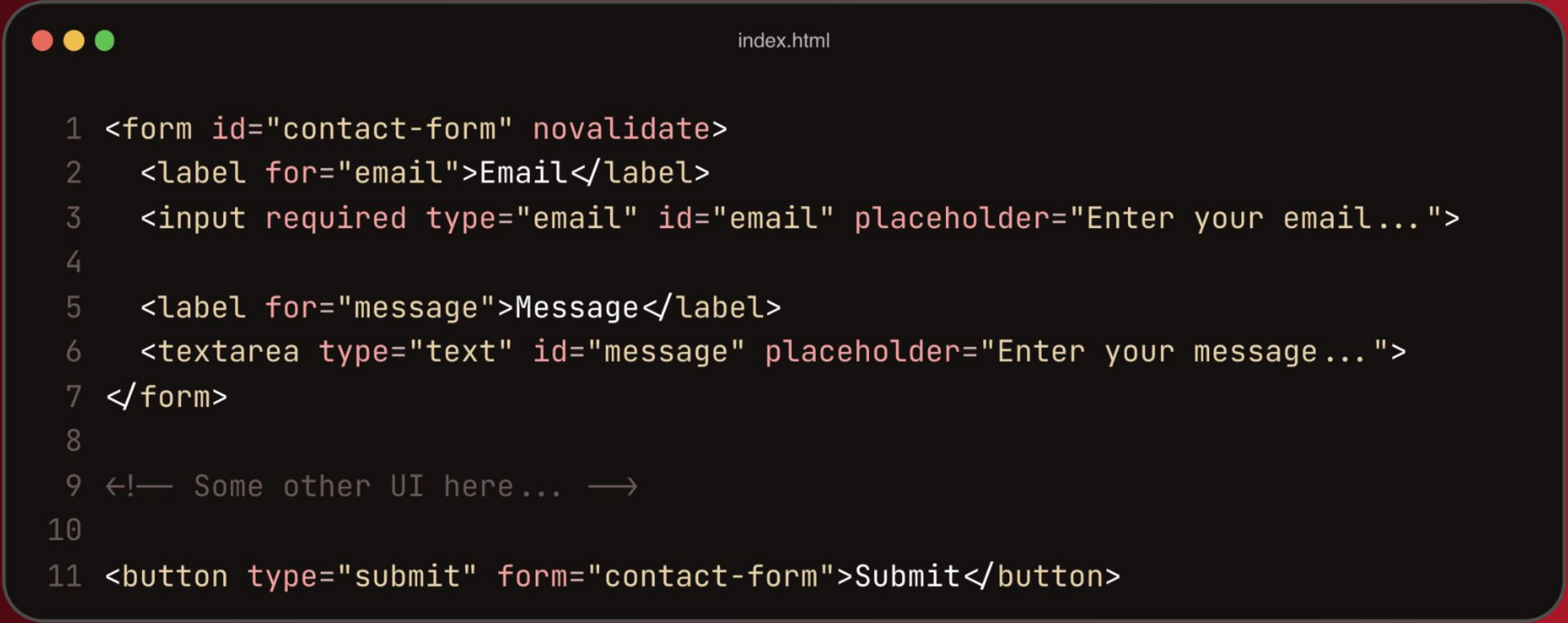
The two most common ways:

- Having a **clickable submit button** at the bottom of the form
- Ability to submit the form by **pressing the `Enter` key**

We can easily enable both scenarios at the same time by **having the clickable submit button** have a **`type` of “submit”**.


index.html

```html
<form novalidate>
  <label for="email">Email</label>
  <input required type="email" id="email" placeholder="Enter your email...">

  <label for="message">Message</label>
  <textarea type="text" id="message" placeholder="Enter your message...">

  <button type="submit">Submit</button>
</form>
```



```html
<form id="contact-form" novalidate>
  <label for="email">Email</label>
  <input required type="email" id="email" placeholder="Enter your email...">

  <label for="message">Message</label>
  <textarea type="text" id="message" placeholder="Enter your message...">
</form>

<!-- Some other UI here... -->

<button type="submit" form="contact-form">Submit</button>
```

Loading state

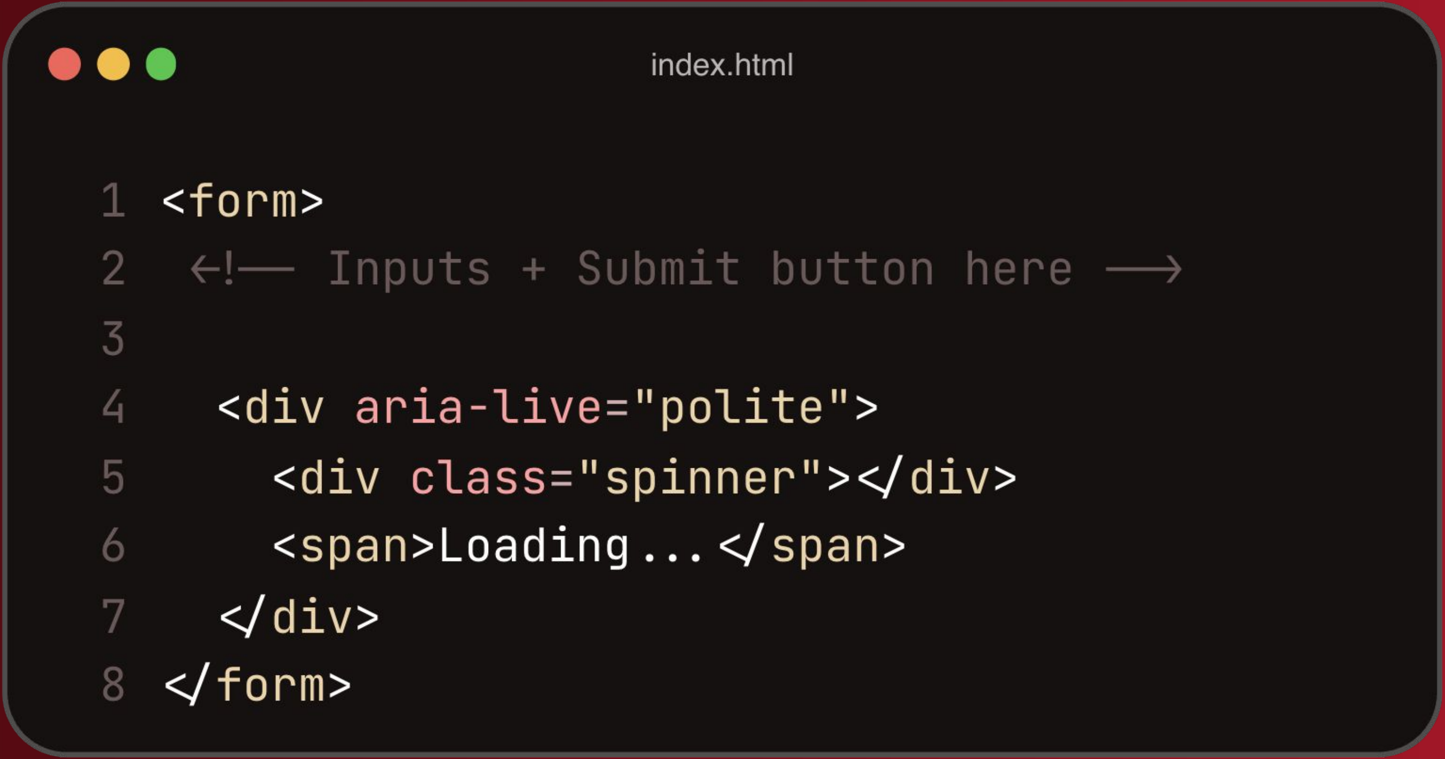
While our form is being submitted - we need to (at least) **disable the submit button** to avoid duplicated submissions.

Ideally we would also disable all of the inputs in the form.

Loading indicators

Generic spinners don’t help users that rely on assistive technologies - **the loading state is not being announced** to them.

We want to **wrap our spinner inside a live region** and also add a “Loading” message (visible or invisible label).



```html
<form>
 <!-- Inputs + Submit button here -->

  <div aria-live="polite">
    <div class="spinner"></div>
    <span>Loading...</span>
  </div>
</form>
```

Loading state

Skeleton loading indicators

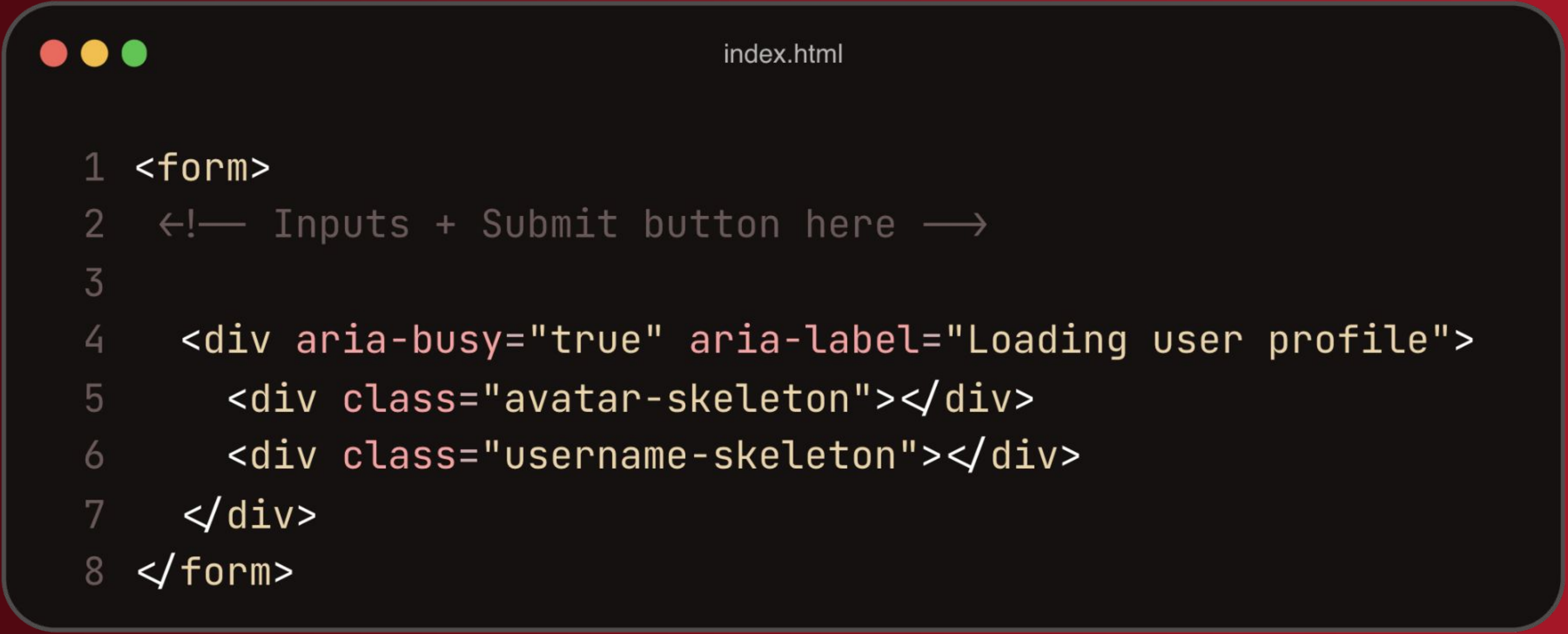
We need to mark these elements as **placeholder elements** that will tell assistive technologies that their content will change shortly.

How do we do that?

Set `**aria-busy=”true”**` on the skeleton element.
Add an `**aria-label**` **attribute** describing what is being waited on.

`aria-busy`

Used to indicate that an **element’s content is being modified**.
Content will be announced when its value becomes “false”.



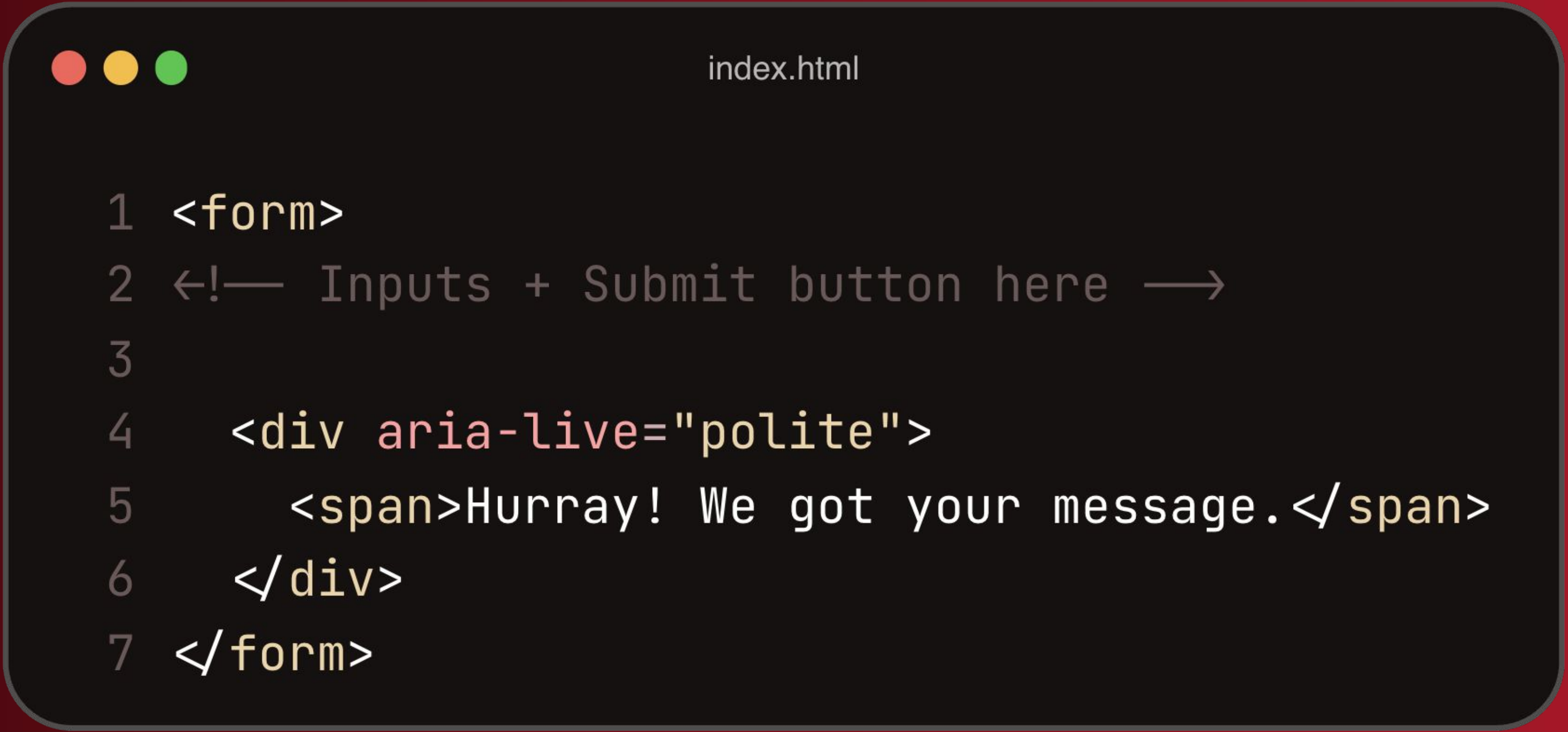
```html
<form>
 <!-- Inputs + Submit button here -->

  <div aria-busy="true" aria-label="Loading user profile">
    <div class="avatar-skeleton"></div>
    <div class="username-skeleton"></div>
  </div>
</form>
```

Success state

Probably **the most overlooked part** of accessible forms on the Web.

Remember, we want any any change in our form’s state to be announced by assistive technologies.

Success messages don’t need anything fancy, a **live region** will do just fine.



```html
<form>
<!-- Inputs + Submit button here -->

  <div aria-live="polite">
    <span>Hurray! We got your message.</span>
  </div>
</form>
```

Accessible forms

Recap

Make sure your inputs are **properly labeled**. Always try to have a visible label if possible.

Descriptions, hints and error messages need to be **programmatically connected** to their associated input.

Success/error messages and loading states need to be set up within live regions so that assistive technologies can announce them.

Always go with **native inputs, native attributes** if possible.

UX Improvements

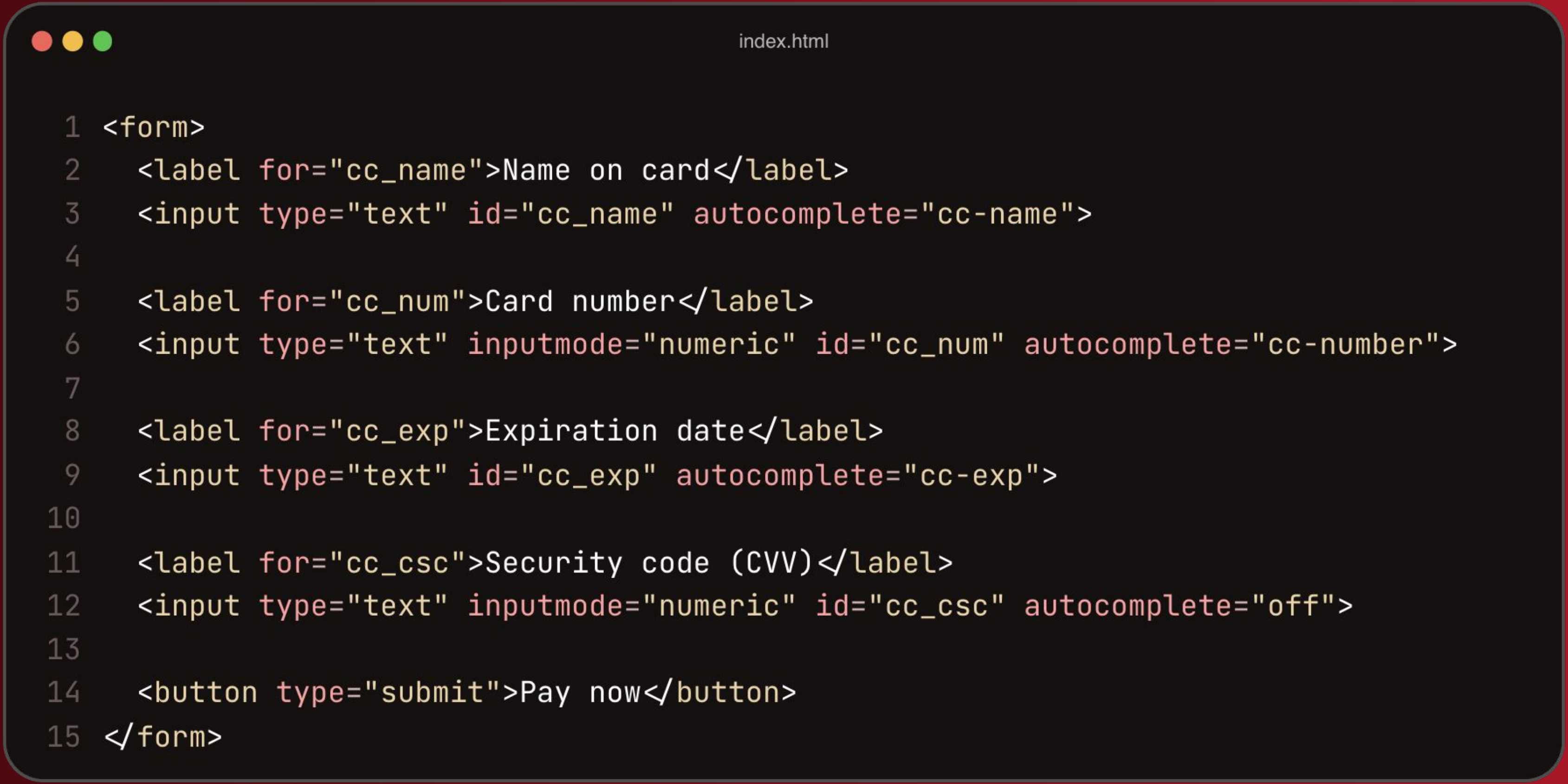
The `autocomplete` attribute

This attribute helps browsers **understand the meaning** of our inputs.

They can then use their **autofill feature** to fill out our forms on behalf of our users.

The `autocomplete` attribute can contain many values, each for a different kind of input.

Note: The **autofill feature** can also come from other software such as **password managers**. The more the merrier!


index.html

```html
<form>
  <label for="cc_name">Name on card</label>
  <input type="text" id="cc_name" autocomplete="cc-name">

  <label for="cc_num">Card number</label>
  <input type="text" inputmode="numeric" id="cc_num" autocomplete="cc-number">

  <label for="cc_exp">Expiration date</label>
  <input type="text" id="cc_exp" autocomplete="cc-exp">

  <label for="cc_csc">Security code (CVV)</label>
  <input type="text" inputmode="numeric" id="cc_csc" autocomplete="off">

  <button type="submit">Pay now</button>
</form>
```

UX Improvements

The `autocomplete` attribute

This attribute has many perks:

- Users will **commit fewer errors** while submitting our forms
- Users with **cognitive, motor and similar disabilities** can fill out our forms more easily
- Screen readers will have more information about the form and will announce it more clearly
- Everyone will generally spend much less time filling out forms

UX Improvements

The `autocomplete` attribute

This is **NOT a silver bullet solution** for every input ever.

Notice that the CVV input field had **`autocomplete=”off”`** - PCI DSS compliance prohibits storing users’ CVV number on devices.

When to use `autocomplete=”off”`

- **One-time tokens** and **password inputs** (sensitive information)
- **Discount/Promo code inputs** (expiring after usage)
- **Search inputs** where we list our own custom suggestions

UX Improvements

Input types & keyboard layouts

One of the most impactful ways to improve our forms - use the **appropriate input types and keyboard layouts**.

This helps browsers know **what kind of data is expected for a given input** and also provide the correct **virtual keyboard** (especially helpful for mobile devices).

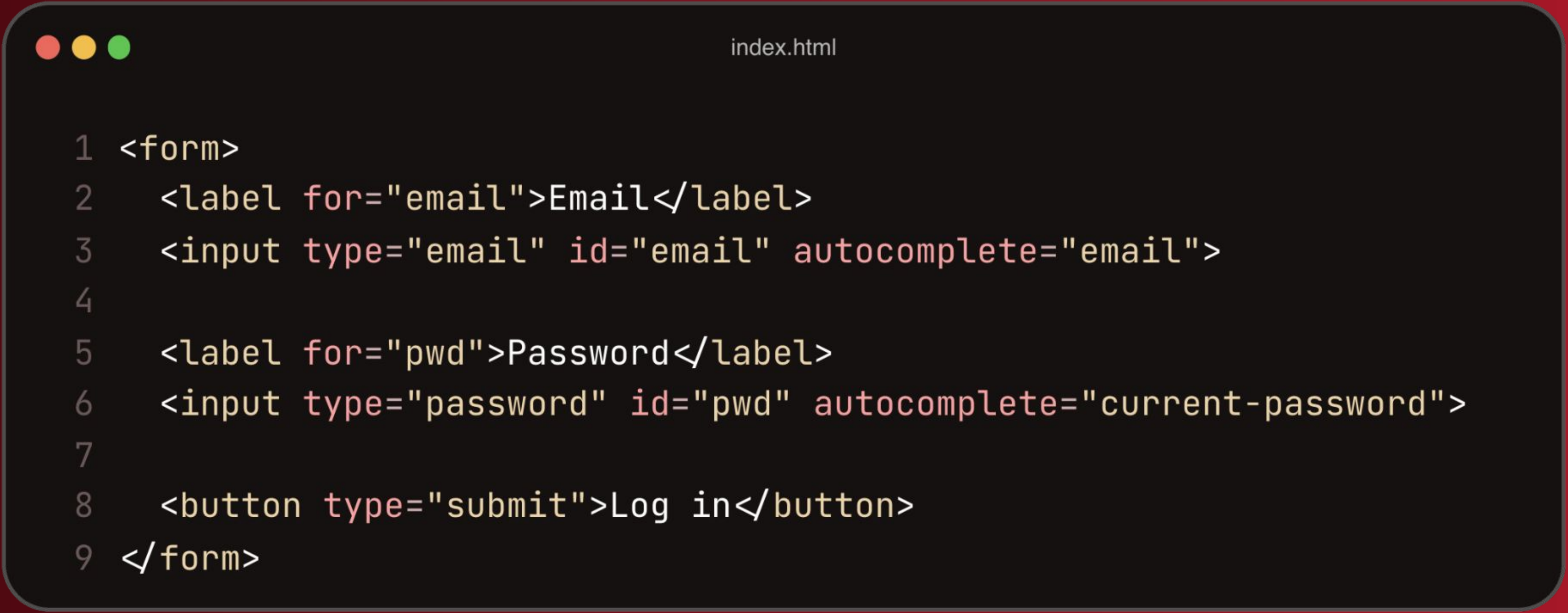
UX Improvements

The `type` attribute

Tells browsers and assistive technologies **what kind of data an input expects.**

Common values: **“text”**, **“email”**, **“password”**, **“tel”** and **“number”**.

Choosing the right `type` can enable **browser autocomplete and validation** features (ex. `type=”email”`).



index.html

```html
<form>
  <label for="email">Email</label>
  <input type="email" id="email" autocomplete="email">

  <label for="pwd">Password</label>
  <input type="password" id="pwd" autocomplete="current-password">

  <button type="submit">Log in</button>
</form>
```

UX Improvements

The `inputmode` attribute

Gives more fine-grained control over **which keyboard layout is displayed** and **which characters can be typed** into an input.

Common values:

- **`numeric`** - only digits (0-9)
- **`decimal`** - digits plus the decimal point
- **`tel`** - telephone keypad (0-9, *, #)
- **`email`** - keyboard layout optimized for emails (“@”, “.com”)
- **`url`** - keyboard layout optimized for URLs

UX Improvements

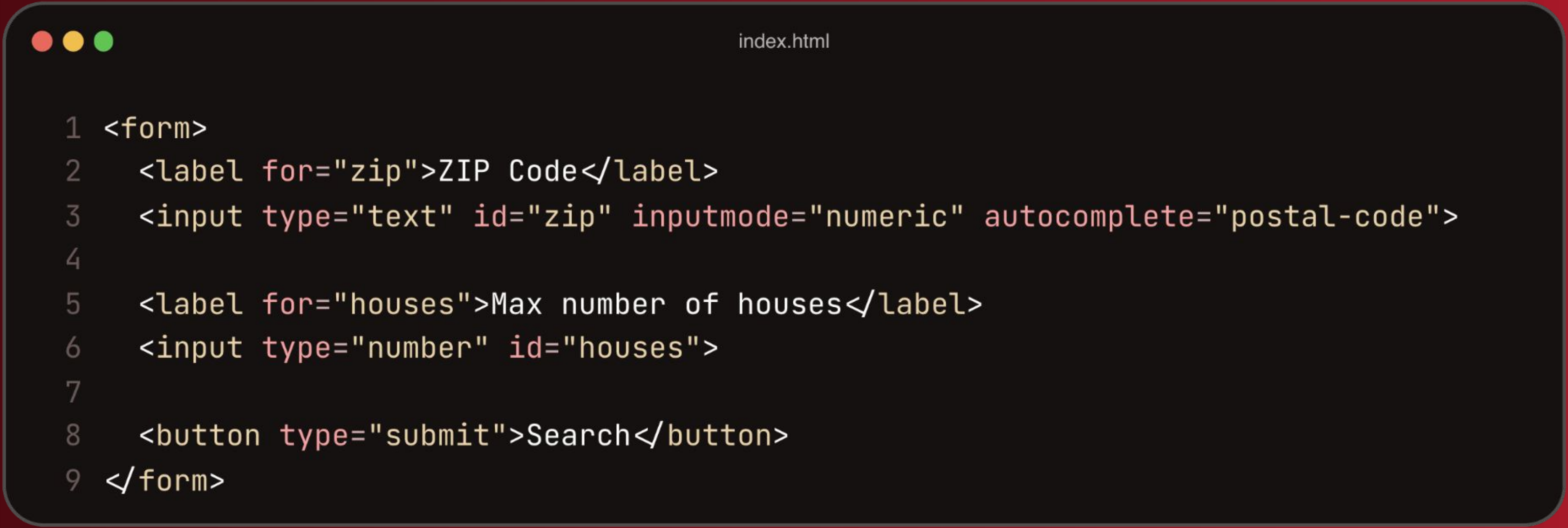
The `inputmode` attribute

This attribute can be particularly useful when the `type` attribute doesn’t quite match the keyboard layout we want to show.

Common gotcha for `type=”number”`

Numeric inputs can be made to work better by combining **`type=”text”`** and **`inputmode=”numeric”`**.

We want to use **`type=”number”`** for **quantity based inputs** (ex. Age, Price, Items count etc.).



```html
<form>
  <label for="zip">ZIP Code</label>
  <input type="text" id="zip" inputmode="numeric" autocomplete="postal-code">

  <label for="houses">Max number of houses</label>
  <input type="number" id="houses">

  <button type="submit">Search</button>
</form>
```

Conclusion

There are a lot of moving parts when it comes to building accessible forms on the Web.

You are now equipped with the knowledge to tackle these challenges and direct your worries to the business logic, animation and styling.

Try to take these concepts and **apply them to your previous and future projects** - you might be surprised how easy it is to miss the small details.

Thank you!

See ya next time!