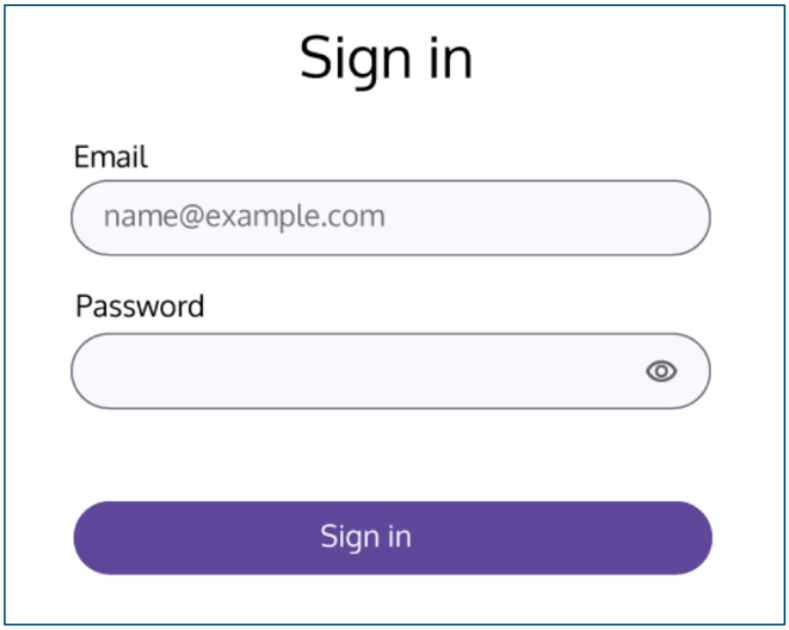
# Forms

Forms are one of the most important components of any web application, allowing users to input and submit data. HTML provides a variety of built-in form attributes that improve usability, structure, and accessibility.




## Semantic HTML for Forms

When creating forms, it's important to use **semantic HTML** to ensure proper structure and accessibility. Common elements include:

- <form> – the container that holds all form elements
- <label> – describes the purpose of an input field
- <input> – used for single-line user input
- <textarea> – used for multi-line text input
- <button> – used to submit or reset the form

## Example: Sign-In Form

```
<form action="/login" method="POST">

    <!-- Email Field -->
    <label for="email">Email</label>
    <input
      type="email"
      id="email"
      name="email"
      placeholder="name@example.com"
      required
    >

    <!-- Password Field -->
    <label for="password">Password</label>
    <input
      type="password"
      id="password"
      name="password"
      required
    >

    <!-- Submit Button -->
    <button type="submit">Sign in</button>

</form>
```

## Form Attributes

<form> element attributes:

- **action** – Specifies the URL where the form data will be sent upon submission
- **method** – Defines how the form data is sent:
  - GET: Appends data to the URL as query parameters (visible in the address bar)
  - POST: Sends data in the request body (more secure for sensitive data)

## **Common** <input> **Attributes**

| Attribute | Description |
|---|---|
| `required` | Ensures the field must be filled before submission |
| `placeholder` | Provides a hint or example inside the input field |
| `pattern` | Uses a regular expression to validate the input |
| `type` | Defines the kind of data expected (e.g., email, password, text) |
| `name` | Identifies the input for submission and server processing |

## Name/Value Pair in Form Submission

When a form is submitted, each input's data is sent as a **name/value pair**:

- The name attribute acts as the **key**
- The user's input becomes the **value**

For example, if a user enters "jane@example.com" in a field with name="email", the browser will send:

```
email=jane@example.com
```

This is how data is structured for processing on the server.

## Form Security and Validation

While forms are essential for interactivity, they also present **security risks**. It's critical to follow best practices:

- Sanitize and validate all input on the server side to prevent:
  - Injection attacks, such as Cross-Site Scripting (XSS) or SQL injection
- Use HTTPS to encrypt form data in transit, protecting it from interception
- Limit bot access and protect against brute-force attacks by:
  - Adding CAPTCHAs or honeypot fields

- Enforcing rate-limiting on submissions

Well-structured forms use semantic HTML, appropriate attributes, and input validation. By combining these techniques with secure handling practices, developers create accessible, functional, and secure form experiences for all users.