Add React to a Website

Use as little or as much React as you need.

React has been designed from the start for gradual adoption, and **you can use as little or as much React as you need**. Perhaps you only want to add some "sprinkles of interactivity" to an existing page. React components are a great way to do that.

The majority of websites aren't, and don't need to be, single-page apps. With **a few lines of code and no build tooling**, try React in a small part of your website. You can then either gradually expand its presence, or keep it contained to a few dynamic widgets.

Add React in One Minute

In this section, we will show how to add a React component to an existing HTML page. You can follow along with your own website, or create an empty HTML file to practice.

There will be no complicated tools or install requirements — **to complete this section, you only need an internet connection, and a minute of your time.**

Optional: Download the full example (2KB zipped)

Step 1: Add a DOM Container to the HTML

First, open the HTML page you want to edit. Add an empty `<div>` tag to mark the spot where you want to display something with React. For example:

```
<!-- ... existing HTML ... -->

<div id="like_button_container"></div>

<!-- ... existing HTML ... -->
```

We gave this `<div>` a unique `id` HTML attribute. This will allow us to find it from the JavaScript code later and display a React component inside of it.

> **Tip**
>
> You can place a "container" `<div>` like this **anywhere** inside the `<body>` tag. You may have as many independent DOM containers on one page as you need. They are usually empty — React will replace any existing content inside DOM containers.

Step 2: Add the Script Tags

Next, add three `<script>` tags to the HTML page right before the closing `</body>` tag:

```
  <!-- ... other HTML ... -->

  <!-- Load React. -->
  <!-- Note: when deploying, replace "development.js" with "production.min.js". -->
  <script src="https://unpkg.com/react@16/umd/react.development.js" crossorigin>
</script>
  <script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js" crossorigin>
</script>

  <!-- Load our React component. -->
  <script src="like_button.js"></script>

</body>
```

The first two tags load React. The third one will load your component code.

Step 3: Create a React Component

Create a file called `like_button.js` next to your HTML page.

Open **this starter code** and paste it into the file you created.

> **Tip**
> This code defines a React component called `LikeButton`. Don't worry if you don't understand it yet — we'll cover the building blocks of React later in our hands-on tutorial and main concepts guide. For now, let's just get it showing on the screen!

After **the starter code**, add two lines to the bottom of `like_button.js`:

```
// ... the starter code you pasted ...

const domContainer = document.querySelector('#like_button_container');
ReactDOM.render(e(LikeButton), domContainer);
```

These two lines of code find the `<div>` we added to our HTML in the first step, and then display our "Like" button React component inside of it.

That's It!

There is no step four. **You have just added the first React component to your website.**

Check out the next sections for more tips on integrating React.

View the full example source code

Download the full example (2KB zipped)

Tip: Reuse a Component

Commonly, you might want to display React components in multiple places on the HTML page. Here is an example that displays the "Like" button three times and passes some data to it:

View the full example source code

Download the full example (2KB zipped)

> **Note**
> This strategy is mostly useful while React-powered parts of the page are isolated from each other. Inside React code, it's easier to use component composition instead.

Tip: Minify JavaScript for Production

Before deploying your website to production, be mindful that unminifed JavaScript can significantly slow down the page for your users.

If you already minify the application scripts, **your site will be production-ready** if you ensure that the deployed HTML loads the versions of React ending in `production.min.js`:

```
<script src="https://unpkg.com/react@16/umd/react.production.min.js" crossorigin>
</script>
<script src="https://unpkg.com/react-dom@16/umd/react-dom.production.min.js"
crossorigin></script>
```

If you don't have a minification step for your scripts, here's one way to set it up.

Optional: Try React with JSX

In the examples above, we only relied on features that are natively supported by the browsers. This is why we used a JavaScript function call to tell React what to display:

```
const e = React.createElement;

// Display a "Like" <button>
return e(
  'button',
  { onClick: () => this.setState({ liked: true }) },
  'Like'
);
```

However, React also offers an option to use JSX instead:

```
// Display a "Like" <button>
return (
  <button onClick={() => this.setState({ liked: true })}>
    Like
  </button>
);
```

These two code snippets are equivalent. While **JSX is completely optional**, many people find it helpful for writing UI code — both with React and with other libraries.

You can play with JSX using this online converter.

Quickly Try JSX

The quickest way to try JSX in your project is to add this `<script>` tag to your page:

```
<script src="https://unpkg.com/babel-standalone@6/babel.min.js"></script>
```

Now you can use JSX in any `<script>` tag by adding `type="text/babel"` attribute to it. Here is an example HTML file with JSX that you can download and play with.

This approach is fine for learning and creating simple demos. However, it makes your website slow and **isn't suitable for production**. When you're ready to move forward, remove this new `<script>` tag and the `type="text/babel"` attributes you've added. Instead, in the next section you will set up a JSX preprocessor to convert all your `<script>` tags automatically.

Add JSX to a Project

Adding JSX to a project doesn't require complicated tools like a bundler or a development server. Essentially, adding JSX **is a lot like adding a CSS preprocessor.** The only requirement is to have Node.js installed on your computer.

Go to your project folder in the terminal, and paste these two commands:

1. **Step 1:** Run `npm init -y` (if it fails, here's a fix)
2. **Step 2:** Run `npm install babel-cli@6 babel-preset-react-app@3`

> **Tip**
> We're **using npm here only to install the JSX preprocessor;** you won't need it for anything else. Both React and the application code can stay as `<script>` tags with no changes.

Congratulations! You just added a **production-ready JSX setup** to your project.

Run JSX Preprocessor

Create a folder called `src` and run this terminal command:

```
npx babel --watch src --out-dir . --presets react-app/prod
```

> **Note**
> `npx` is not a typo — it's a package runner tool that comes with npm 5.2+.
>
> If you see an error message saying "You have mistakenly installed the `babel` package", you might have missed the previous step. Perform it in the same folder, and then try again.

Don't wait for it to finish — this command starts an automated watcher for JSX.

If you now create a file called `src/like_button.js` with this **JSX starter code**, the watcher will create a preprocessed `like_button.js` with the plain JavaScript code suitable for the browser. When you edit the source file with JSX, the transform will re-run automatically.

As a bonus, this also lets you use modern JavaScript syntax features like classes without worrying about breaking older browsers. The tool we just used is called Babel, and you can learn more about it from its documentation.

If you notice that you're getting comfortable with build tools and want them to do more for you, the next section describes some of the most popular and approachable toolchains. If not — those script tags will do just fine!

Previous article
Getting Started

Next article
Create a New React App