Add React to an Existing Project

If you want to add some interactivity to your existing project, you don't have to rewrite it in React. Add React to your existing stack, and render interactive React components anywhere.

> **Note**
>
> **You need to install Node.js for local development.** Although you can try React online or with a simple HTML page, realistically most JavaScript tooling you'll want to use for development requires Node.js.

Using React for an entire subroute of your existing website

Let's say you have an existing web app at `example.com` built with another server technology (like Rails), and you want to implement all routes starting with `example.com/some-app/` fully with React.

Here's how we recommend to set it up:

1. **Build the React part of your app** using one of the React-based frameworks.
2. **Specify `/some-app` as the *base path*** in your framework's configuration (here's how: Next.js, Gatsby).
3. **Configure your server or a proxy** so that all requests under `/some-app/` are handled by your React app.

This ensures the React part of your app can benefit from the best practices baked into those frameworks.

Many React-based frameworks are full-stack and let your React app take advantage of the server. However, you can use the same approach even if you can't or don't want to run JavaScript on the server. In that case, serve the HTML/CSS/JS export (`next export` output for Next.js, default for Gatsby) at `/some-app/` instead.

Using React for a part of your existing page

Let's say you have an existing page built with another technology (either a server one like Rails, or a client one like Backbone), and you want to render interactive React components somewhere on that page. That's a common way to integrate React—in fact, it's how most React usage looked at Meta for many years!

You can do this in two steps:

1. **Set up a JavaScript environment** that lets you use the JSX syntax, split your code into modules with the `import` / `export` syntax, and use packages (for example, React) from the npm package registry.
2. **Render your React components** where you want to see them on the page.

The exact approach depends on your existing page setup, so let's walk through some details.

Step 1: Set up a modular JavaScript environment

A modular JavaScript environment lets you write your React components in individual files, as opposed to writing all of your code in a single file. It also lets you use all the wonderful packages published by other developers on the npm registry—including React itself! How you do this depends on your existing setup:

- **If your app is already split into files that use `import` statements,** try to use the setup you already have. Check whether writing `<div />` in your JS code causes a syntax error. If it causes a syntax error, you might need to transform your JavaScript code with Babel, and enable the Babel React preset to use JSX.
- **If your app doesn't have an existing setup for compiling JavaScript modules,** set it up with Vite. The Vite community maintains many integrations with backend frameworks, including Rails, Django, and Laravel. If your backend framework is not listed, follow this guide to manually integrate Vite builds with your backend.

To check whether your setup works, run this command in your project folder:

Terminal Copy

```
npm install react react-dom
```

Then add these lines of code at the top of your main JavaScript file (it might be called `index.js` or `main.js`):

index.js Reset Fork

```
import { createRoot } from 'react-dom/client';

// Clear the existing HTML content
document.body.innerHTML = '<div id="app"></div>';

```

Hello, world

```
// Render your React component instead
const root = createRoot(document.getElementById('app'));
root.render(<h1>Hello, world</h1>);

```

If the entire content of your page was replaced by a “Hello, world!”, everything worked! Keep reading.

> **Note**
>
> Integrating a modular JavaScript environment into an existing project for the first time can feel intimidating, but it’s worth it! If you get stuck, try our community resources or the Vite Chat.

Step 2: Render React components anywhere on the page

In the previous step, you put this code at the top of your main file:

```
import { createRoot } from 'react-dom/client';

// Clear the existing HTML content
document.body.innerHTML = '<div id="app"></div>';

// Render your React component instead
const root = createRoot(document.getElementById('app'));
root.render(<h1>Hello, world</h1>);
```

Of course, you don’t actually want to clear the existing HTML content!

Delete this code.

Instead, you probably want to render your React components in specific places in your HTML. Open your HTML page (or the server templates that generate it) and add a unique `id` attribute to any tag, for example:

```
<!-- ... somewhere in your html ... -->
<nav id="navigation"></nav>
<!-- ... more html ... -->
```

This lets you find that HTML element with `document.getElementById` and pass it to `createRoot` so that you can render your own React component inside:

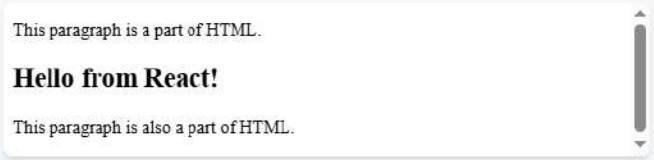
index.js index.html Reset Fork

```
import { createRoot } from 'react-dom/client';

function NavigationBar() {
  // TODO: Actually implement a navigation bar
  return <h1>Hello from React!</h1>;
}

const domNode = document.getElementById('navigation');
const root = createRoot(domNode);
root.render(<NavigationBar />);

```



Notice how the original HTML content from `index.html` is preserved, but your own `NavigationBar` React component now appears inside the `<nav id="navigation">` from your HTML. Read the `createRoot` usage documentation to learn more about rendering React components inside an existing HTML page.

When you adopt React in an existing project, it's common to start with small interactive components (like buttons), and then gradually keep "moving upwards" until eventually your entire page is built with React. If you ever reach that point, we recommend migrating to a React framework right after to get the most out of React.

Using React Native in an existing native mobile app

React Native can also be integrated into existing native apps incrementally. If you have an existing native app for Android (Java or Kotlin) or iOS (Objective-C or Swift), follow this guide to add a React Native screen to it.