

Overview

Production-grade React frameworks

Next.js

Remix

Gatsby

Expo (for native apps)

Bleeding-edge React frameworks

Next.js (App Router)

Start a New React Project

If you want to build a new app or a new website fully with React, we recommend picking one of the React-powered frameworks popular in the community.

You can use React without a framework, however we've found that most apps and sites eventually build solutions to common problems such as code-splitting, routing, data fetching, and generating HTML. These problems are common to all UI libraries, not just React.

By starting with a framework, you can get started with React quickly, and avoid essentially building your own framework later.

 DEEP DIVE

Can I use React without a framework?

^ Hide Details

You can definitely use React without a framework—that's how you'd [use React for a part of your page](#). **However, if you're building a new app or a site fully with React, we recommend using a framework.**

Here's why.

Even if you don't need routing or data fetching at first, you'll likely want to add some libraries for them. As your JavaScript bundle grows with every new feature, you might have to figure out how to split code for every route individually. As your data fetching needs get more complex, you are likely to encounter server-client network waterfalls that make your app feel very slow. As your audience includes more users with poor network conditions and low-end devices, you might need to generate HTML from your components to display content early—either on the server, or during the build time. Changing your setup to run some of your code on the server or during the build can be very tricky.

These problems are not React-specific. This is why Svelte has SvelteKit, Vue has Nuxt, and so on. To solve these problems on your own, you'll need to integrate your bundler with your router and with your data fetching library. It's not hard to get an initial setup working, but there are a lot of subtleties involved in making an app that loads quickly even as it grows over time. You'll want to send down the minimal amount of app code but do so in a single client-server roundtrip, in parallel with any data required for the page. You'll likely want the page to be interactive before your JavaScript code even runs, to support progressive enhancement. You may want to generate a folder of fully static HTML files for your marketing pages that can be hosted anywhere and still work with JavaScript disabled. Building these capabilities yourself takes real work.

React frameworks on this page solve problems like these by default, with no extra work from your side. They let you start very lean and then scale your app with your needs. Each React framework has a community, so finding answers to questions and upgrading tooling is easier. Frameworks also give structure to your code, helping you and others retain context and skills between different projects. Conversely, with a custom setup it's easier to get stuck on unsupported dependency versions, and you'll essentially end up creating your own framework—albeit one with no community or upgrade path (and if it's anything like the ones we've made in the past, more haphazardly designed).

If your app has unusual constraints not served well by these frameworks, or you prefer to solve these problems yourself, you can roll your own custom setup with React. Grab `react` and `react-dom` from npm, set up your custom build process with a bundler like [Vite](#) or [Parcel](#), and add other tools as you need them for routing, static generation or server-side rendering, and more.

Production-grade React frameworks

These frameworks support all the features you need to deploy and scale your app in production and are working towards supporting our [full-stack architecture vision](#). All of the frameworks we recommend are open source with active communities for support, and can be deployed to your own server or a hosting provider. If you're a framework author interested in being included on this list, [please let us know](#).

Next.js

Next.js' Pages Router is a **full-stack React framework**. It's versatile and lets you create React apps of any size—from a mostly static blog to a complex dynamic application. To create a new Next.js project, run in your terminal:

A screenshot of a terminal window with a dark background. The title bar says "Terminal" and there is a "Copy" button on the right. The command `npx create-next-app@latest` is entered in the terminal.

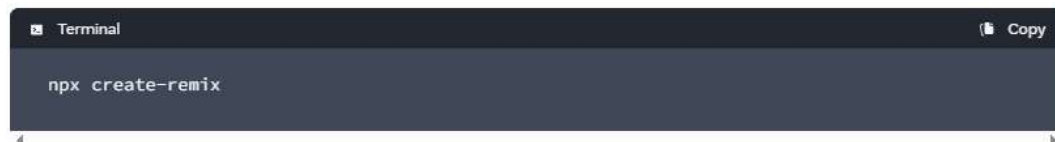
```
npx create-next-app@latest
```

If you're new to Next.js, check out the [learn Next.js course](#).

Next.js is maintained by [Vercel](#). You can [deploy a Next.js app](#) to any Node.js or serverless hosting, or to your own server. Next.js also supports a [static export](#) which doesn't require a server.

Remix

Remix is a **full-stack React framework with nested routing**. It lets you break your app into nested parts that can load data in parallel and refresh in response to the user actions. To create a new Remix project, run:

A screenshot of a terminal window with a dark background. The title bar says "Terminal" and there is a "Copy" button on the right. The command `npx create-remix` is entered in the terminal.

```
npx create-remix
```

If you're new to Remix, check out the Remix [blog tutorial](#) (short) and [app tutorial](#) (long).

Remix is maintained by [Shopify](#). When you create a Remix project, you need to [pick your deployment target](#). You can deploy a Remix app to any Node.js or serverless hosting by using or writing an [adapter](#).

Gatsby

Gatsby is a **React framework for fast CMS-backed websites**. Its rich plugin ecosystem and its GraphQL data layer simplify integrating content, APIs, and services into one website. To create a new Gatsby project, run:

A terminal window with a dark background. The title bar says "Terminal" and there is a "Copy" button in the top right corner. The command `npx create-gatsby` is entered in the terminal.

If you're new to Gatsby, check out the [Gatsby tutorial](#).

Gatsby is maintained by [Netlify](#). You can [deploy a fully static Gatsby site](#) to any static hosting. If you opt into using server-only features, make sure your hosting provider supports them for Gatsby.

Expo (for native apps)

[Expo](#) is a React framework that lets you create universal Android, iOS, and web apps with truly native UIs. It provides an SDK for [React Native](#) that makes the native parts easier to use. To create a new Expo project, run:

A terminal window with a dark background. The title bar says "Terminal" and there is a "Copy" button in the top right corner. The command `npx create-expo-app` is entered in the terminal.

If you're new to Expo, check out the [Expo tutorial](#).

Expo is maintained by [Expo \(the company\)](#). Building apps with Expo is free, and you can submit them to the Google and Apple app stores without restrictions. Expo additionally provides opt-in paid cloud services.

Bleeding-edge React frameworks

As we've explored how to continue improving React, we realized that integrating React more closely with frameworks (specifically, with routing, bundling, and server technologies) is our biggest opportunity to help React users build better apps. The Next.js team has agreed to collaborate with us in researching, developing, integrating, and testing framework-agnostic bleeding-edge React features like [React Server Components](#).

These features are getting closer to being production-ready every day, and we've been in talks with other bundler and framework developers about integrating them. Our hope is that in a year or two, all frameworks listed on this page will have full support for these features. (If you're a framework author interested in partnering with us to experiment with these features, please let us know!)

Next.js (App Router)

Next.js's App Router is a redesign of the Next.js APIs aiming to fulfill the React team's full-stack architecture vision. It lets you fetch data in asynchronous components that run on the server or even during the build.

Next.js is maintained by [Vercel](#). You can [deploy a Next.js app](#) to any Node.js or serverless hosting, or to your own server. Next.js also supports [static export](#) which doesn't require a server.

DEEP DIVE

Which features make up the React team's full-stack architecture vision?

 Hide Details

Next.js's App Router bundler fully implements the official [React Server Components specification](#). This lets you mix build-time, server-only, and interactive components in a single React tree.

For example, you can write a server-only React component as an `async` function that reads from a database or from a file. Then you can pass data down from it to your interactive components:

```
// This component runs *only* on the server (or during the build).
async function Talks({ confId }) {
  // 1. You're on the server, so you can talk to your data layer. API endpoint not required.
  const talks = await db.Talks.findAll({ confId });

  // 2. Add any amount of rendering logic. It won't make your JavaScript bundle larger.
  const videos = talks.map(talk => talk.video);

  // 3. Pass the data down to the components that will run in the browser.
  return <SearchableVideoList videos={videos} />;
}
```


Next.js's App Router also integrates [data fetching with Suspense](#). This lets you specify a loading state (like a skeleton placeholder) for different parts of your user interface directly in your React tree:

```
<Suspense fallback=<TalksLoading />>
  <Talks confId={conf.id} />
</Suspense>
```

Server Components and Suspense are React features rather than Next.js features. However, adopting them at the framework level requires buy-in and non-trivial implementation work. At the moment, the Next.js App Router is the most complete implementation. The React team is working with bundler developers to make these features easier to implement in the next generation of frameworks.