Focused on web standards and modern web app UX, you're simply going to build better websites

Remix is a full stack web framework that lets you focus on the user interface and work back through web standards to deliver a fast, slick, and resilient user experience. People are gonna love using your stuff.

Get Started

Read the Docs

```
export async function loader({ request }) {
  return getProjects();
}

export async function action({ request }) {
  const form = await request.formData();
  return createProject({ title: form.get("title") });
}

export default function Projects() {
  const projects = useLoaderData();
```

```
  const { state } = useNavigation();
  const busy = state === "submitting";

  return (
    <div>
      {projects.map((project) => (
        <Link to={project.slug}>{project.title}</Link>
      ))}

      <Form method="post">
        <input name="title" />
        <button type="submit" disabled={busy}>
          {busy ? "Creating..." : "Create New Project"}
        </button>
      </Form>
    </div>
  );
}
```

Jenna Smith

RADIX UI

I've been waiting for something to encourage progressive enhancement in the React space *forever* and Remix truly is so much more. Proving we don't need to sacrifice React or choose SSG for a lightning fast, accessible UI, and the DX makes it all too easy 😋

@jkup
Cloudflare

holy 💩 Remix is good

@
Dis

I just rewrot
Supabase ar

While you were waiting for your static site to build, distributed web infra-structure got really good. Break through the static.

Remix is a seamless server and browser runtime that provides snappy page loads and instant transitions by leveraging distributed systems and native browser features instead of clunky static builds. Built on the Web Fetch API (instead of Node) **it can run anywhere**. It already runs natively on Cloudflare Workers, and of course supports serverless and traditional Node.js environments, so you can come as you are.

Page speed is only one aspect of our true goal though. We're after **better user experiences**. As you've pushed the boundaries of the web, your tools haven't caught up to your appetite. **Remix is ready** to serve you from the initial request to the fanciest UX your designers can think up. Check it out 👀

Remix has a cheat code:
Nested Routes.
↑↑↓↓←→←→BA↵

Websites usually have levels of navigation that control child views.

Not only are these components pretty much always coupled to URL segments...

...they're also the semantic boundary of data loading and code splitting.

Hover or tap the buttons to see how they're all related

example.com/sales/invoices/102000

Fakebooks

Dashboard
Accounts
Sales
Expenses
Reports

Sales

Overview Subscriptions Invoices Customers Deposits

OVERDUE
$10,800

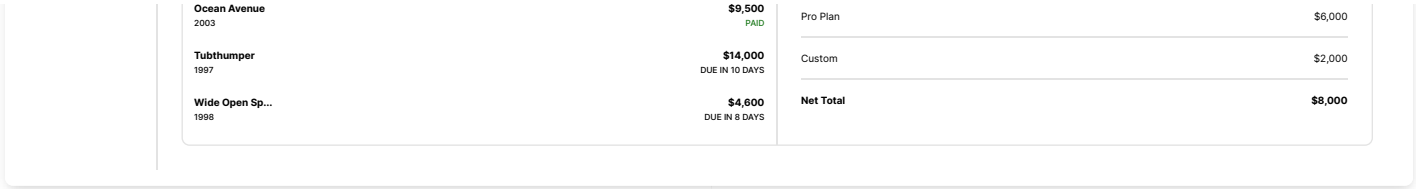
DUE SOON
$62,000

INVOICE LIST

Santa Monica
1995
$10,800
OVERDUE

Stankonia
2000
$8,000
DUE TODAY

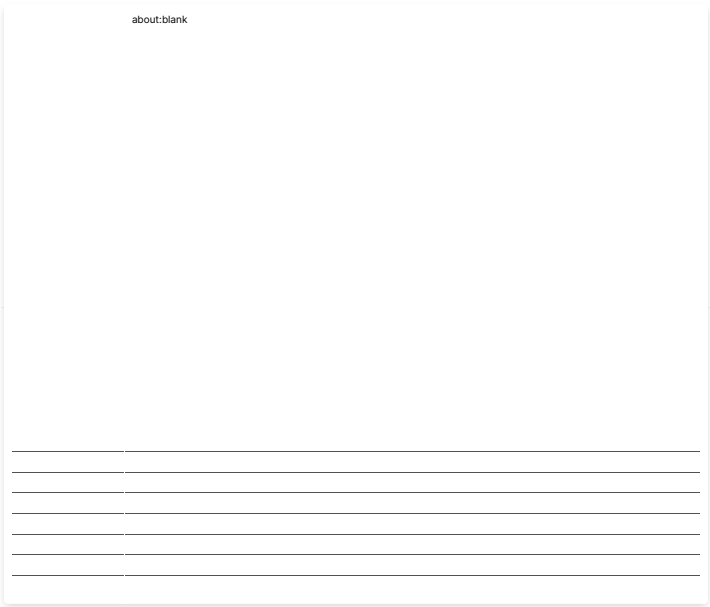
Stankonia
$8,000
DUE TODAY • INVOICED 10/31/2000



Through nested routes, Remix can eliminate nearly every loading state.

Most web apps fetch inside of components, creating request waterfalls, slower loads, and jank.

Remix loads data in parallel on the server and sends a fully formed HTML document. Way faster, jank free.



With Remix

(Keep scrolling to compare)

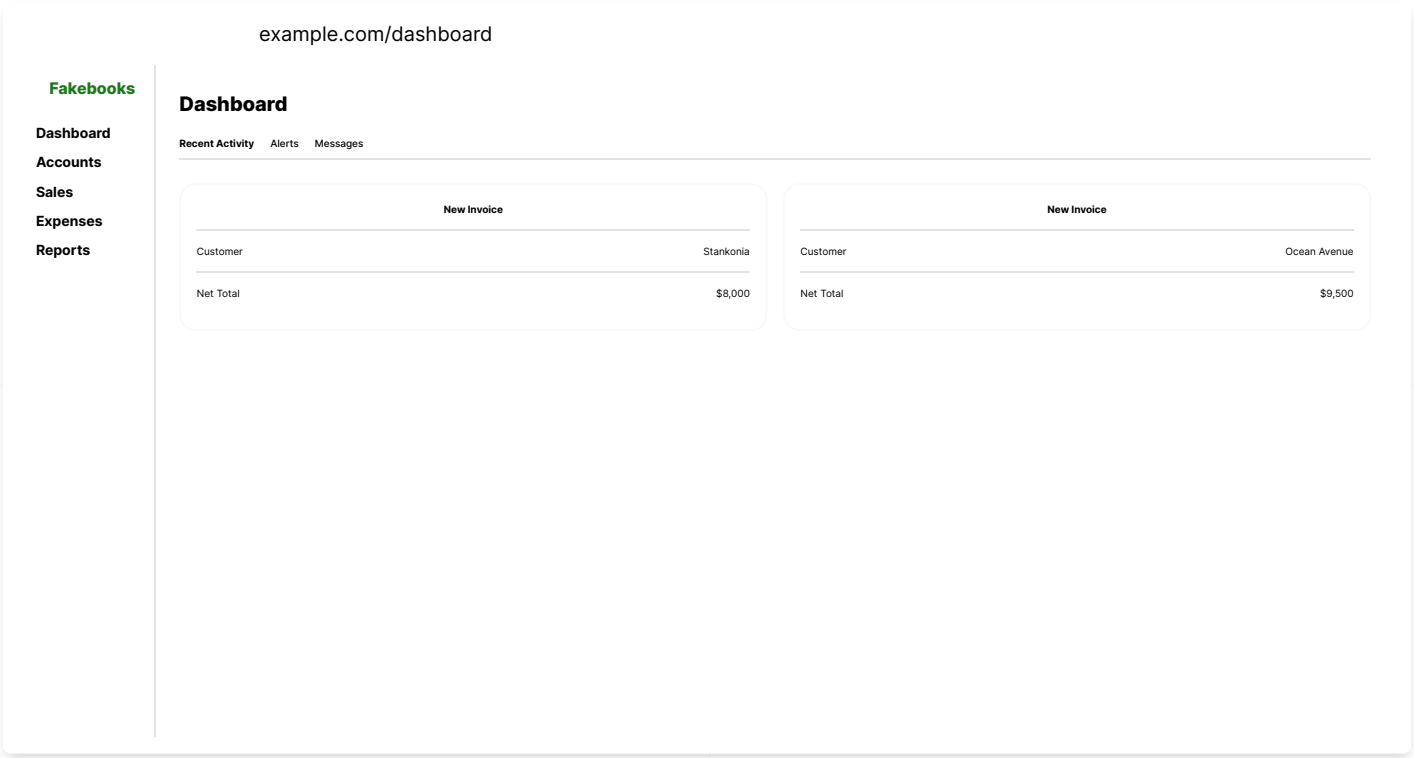
Nested routes allow Remix to make your app as fast as instant.

Remix can prefetch everything in parallel before the user clicks a link.

Public Data. User Data. Modules. Heck, even CSS.

Zero loading states. Zero skeleton UI. Zero jank.

Alright, you caught us, they're just prefetch link tags, #useThePlatform


example.com/dashboard
Fakebooks
Dashboard
Accounts
Sales
Expenses
Reports
Dashboard
Recent Activity
Alerts
Messages
New Invoice
Customer
Stankonia
Net Total
$8,000
New Invoice
Customer
Ocean Avenue
Net Total
$9,500

Data loading ... 🤦
You ever notice most of the code in your app is for changing data?

Imagine if React only had props and no way to set state. What's the point? If a web framework helps you load data but doesn't help you update it, what's the point? Remix doesn't drop you off at the `<form onSubmit>` cliff. (What the heck does `event.preventDefault` do anyway?)

Resilient, progressively enhanced data updates are built in.

It's so simple it's kind of silly. Just make a form...

...and an action on a route module. It looks like traditional HTML forms but enables fully dynamic web experiences you're after.

Remix runs the action server side, revalidates data client side, and even handles race conditions from resubmissions.

Get fancy with transition hooks and make some pending UI. Remix handles all the state, you simply ask for it.

Or get jiggy with some optimistic UI. Remix provides the data being sent to the server so you can skip the busy spinners for mutations, too.

HTML forms for mutations. Who knew?

```
export default function NewInvoice() {
  return (
    <Form method="post">
      <input type="text" name="company" />
      <input type="text" name="amount" />
      <button type="submit">Create</button>
    </Form>
  );
}
```

:)

Your websites run into problems, but with Remix they don’t need to be refreshed. Error handling is hard to remember and hard to do. That’s why it’s built in.

Remix handles errors while Server Rendering. Errors while Client Rendering. Even errors in your server side data handling.

Route Error Boundaries
keep the happy path happy.

Each route module can export an error boundary next to the default route component.

If an error is thrown, client or server side, users see the boundary instead of the default component.

Routes w/o trouble render normally, so users have more options than slamming refresh.

If a route has no boundary, errors bubble up. Just put one at the top and chill out about errors in code review, yeah?

```
export default function InvoiceRoute() {
  const invoice = useLoaderData();
  return <Invoice data={invoice} />;
}
```

That's probably enough for now. What are you waiting

now. What are you waiting for?

Go Play!