

react-router와 tanstack-router

Frontend의 router 생태계

React 사용시 선택 가능한 라이브러리들

1. Wouter (매우 가벼움)
2. React Router (근본)
3. Tanstack Router (후발주자)

더이상 route 기능만 제공하지 않는 router들

1. Code Split
 2. Loader
 3. Preload
 4. Scroll Restoration
 5. Server Side Rendering
- 등등

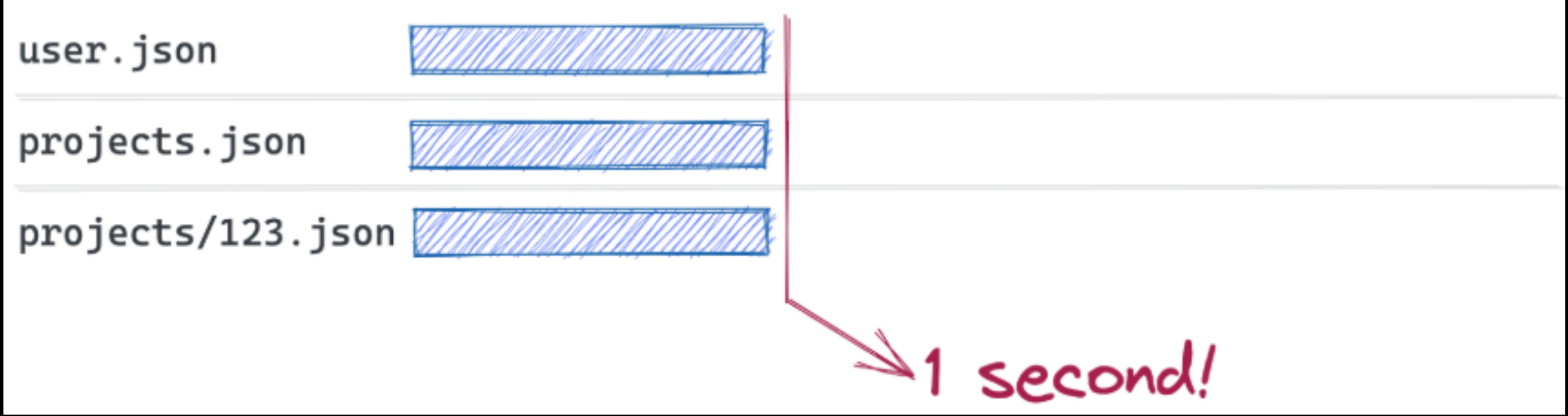
이러한 기능들이 없는, route기능만 제공하는 router를 선택하고싶다.

→ wouter

React Router의 Loader



Waterfall 문제가 존재

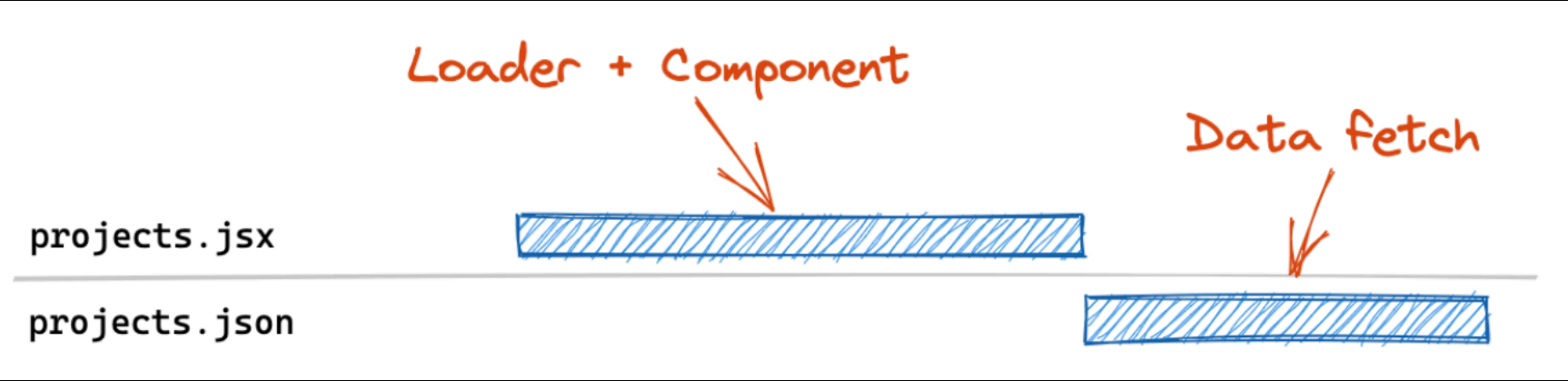


Waterfall 문제 해결

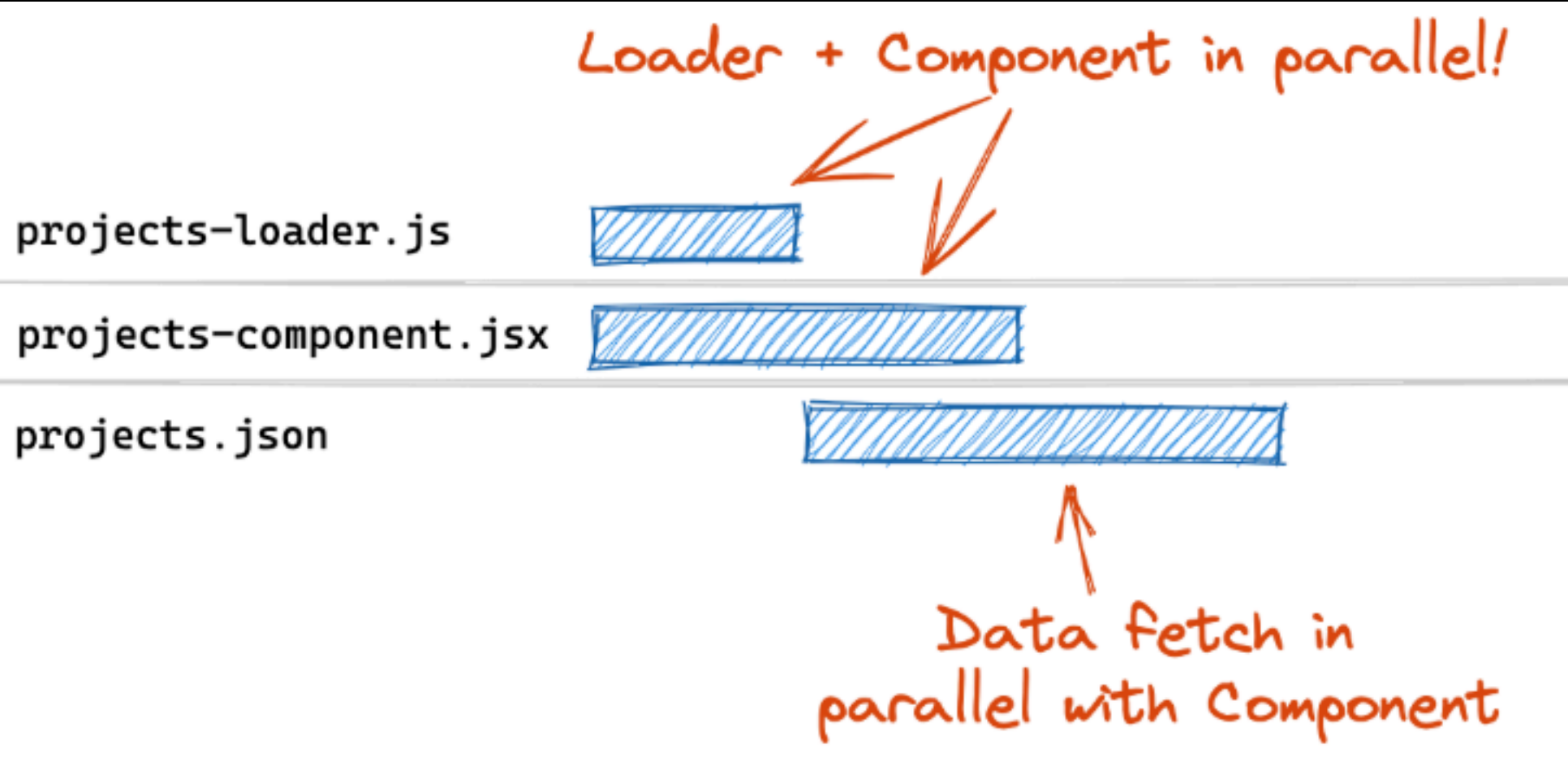
```
const routes = [{
  path: 'projects',
  lazy: () => import("./projects"), // 컴포넌트 + 로더 함께
}];

// projects.jsx
export function loader() { ... }
export function Component() { ... }
```

```
const routes = [
  {
    path: "projects",
    // 정적으로 로더만 분리
    async loader({ request, params }) {
      let { loader } = await import("./projects-loader");
      return loader({ request, params });
    },
    // 컴포넌트는 따로 lazy 로드
    lazy: () => import("./projects-component"),
  },
];
```



기존에는 페이지와 loader를 함께 배치하고 lazy를 사용하는 경우 Data Fetch가 더 느려짐



페이지와 loader를 함께 배치하지 않아 Data Fetch가 느리지 않음

Tanstack Router의 Loader


React Router 기능에 더해서 Caching 기능도 존재

TanStack Router Cache Pros:







- Built-in, easy to use, no extra dependencies
- Handles deduping, preloading, loading, stale-while-revalidate, background refetching on a per-route basis
- Coarse invalidation (invalidate all routes and cache at once)
- Automatic garbage collection
- Works great for apps that share little data between routes
- "Just works" for SSR


TanStack Router Cache Cons:

- No persistence adapters/model
- No shared caching/deduping between routes
- No built-in mutation APIs (a basic `useMutation` hook is provided in many examples that may be sufficient for many use cases)
- No built-in cache-level optimistic update APIs (you can still use ephemeral state from something like a `useMutation` hook to achieve this at the component level)







 **JayV30** · 5mo ago

Yeah, time for me to take a more serious look at tanstack-router. I'm through with all the react-router shenanigans


  201   Reply  Award  Share ...

 **kei_ichi** · 5mo ago

Believe me, you will not regret when move to TanStack Router.

  36   Reply  Award  Share ...


Well soon as I opened the docs and realized the "As a Library"/"As a Framework" pattern was going to stick around I was convinced there was no way this wasn't done to self-sabotage.

 **belousovnikita92** · 5mo ago

React router is one of the most annoying libraries there is, they've been breaking everything every major release I can remember.

Sole fact that the took down old versions documentation on v7 release is bad move in my book but well, at least they added it back

I agree with many people here, maybe it's actually time to move away from it






 **PistachioPlz** · 5mo ago


We swapped to Tanstack Router. We got really great help by Manuel from the router contributors team. The one pain point we had was TSR didn't support optional path parameters. (no `/foo?/bar`)

But that just made us rethink the structure, and since it was an admin panel type project, the change wasn't a big deal.

And yes it's definitely a new way of working and thinking about routes some times. Everything has to be type safe, so it can become quite complicated with the types when you need to re-use components that deal with routing. Take filters for example. We wanted to use a hook to handle filters, using query parameters as "state". But the way you validate search params and want to keep everything as type safe as possible, it can get quite... complicated when trying to use a `useFilters` hook

So if you don't have some talented typescript people on your team, you might have to let go of some of that type safety by setting `{ strict: false }` some times. :')

 5   Reply  Award  Share ...







 **Brahminmeat** · 5mo ago

Hi, I was at shopify during the Remix acquisition (though not directly involved) and it was like this from the beginning. Shopify doesn't just throw resources at something they could otherwise just throw money at as a "sponsor"









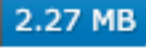
They acquired remix while the migration was already underway (or maybe because of it)

They have a pattern of eating up tech then doing not much with it and abandoning it outside of some very singular use cases. They like to chase new tech without actually wanting to integrate that new tech long term into their solutions (the main platform is still rails and react)

Now why did they buy up this particular framework? Your guess is as good as mine. It wouldn't surprise me if it was just to generate goodwill in the press and for shareholders, since that's all they really care about.

  75   Reply  Award  Share ...



			Stars	Issues	Version	Updated [?]	Created [?]	Size
	@tanstack/react-router	  	10,429	333	1.122.0	2 days ago	3 years ago	install size 
	react-router	  	55,121	183	7.6.3	2 days ago	11 years ago	install size 

100% Inferred TypeScript Support

Everything these days is written “in Typescript” or at the very least offers type definitions that are veneered over runtime functionality, but too few packages in the ecosystem actually design their APIs with TypeScript in mind. So while I’m pleased that your router is auto-completing your option fields and catching a few property/method typos here and there, there is much more to be had.

- TanStack Router is fully aware of all of your routes and their configuration at any given point in your code. This includes the path, path params, search params, context, and any other configuration you’ve provided. Ultimately this means that you can navigate to any route in your app with 100% type safety and confidence that your link or navigate call will succeed.
- TanStack Router provides lossless type-inference. It uses countless generic type parameters to enforce and propagate any type information you give it throughout the rest of its API and ultimately your app. No other router offers this level of type safety and developer confidence.

Summary

[Reference Documentation ↗](#)

Emulates the browser's scroll restoration on location changes. Apps should only render one of these, right before the `Scripts` component.

```
1 import { ScrollRestoration } from "react-router";
2
3 export default function Root() {
4   return (
5     <html>
6       <body>
7         <ScrollRestoration />
8         <Scripts />
9       </body>
10    </html>
11  );
12 }
```

This component renders an inline `<script>` to prevent scroll flashing. The `nonce` prop will be passed down to the script tag to allow CSP nonce usage.

```
1 <ScrollRestoration nonce={cspNonce} />
```

Props

ScriptsProps



A couple common attributes:

- `<Scripts crossOrigin>` for hosting your static assets on a different server than your app.
- `<Scripts nonce>` to support a content security policy for scripts with nonce-sources for your `<script>` tags.

You cannot pass through attributes such as `async`, `defer`, `src`, `type`, `noModule` because they are managed by React Router internally.

Scroll Restoration

Scroll restoration is the process of restoring the scroll position of a page when the user navigates back to it. This is normally a built-in feature for standard HTML based websites, but can be difficult to replicate for SPA applications because:

- SPAs typically use the `history.pushState` API for navigation, so the browser doesn't know to restore the scroll position natively
- SPAs sometimes render content asynchronously, so the browser doesn't know the height of the page until after it's rendered
- SPAs can sometimes use nested scrollable containers to force specific layouts and features.

Not only that, but it's very common for applications to have multiple scrollable areas within an app, not just the body. For example, a chat application might have a scrollable sidebar and a scrollable chat area. In this case, you would want to restore the scroll position of both areas independently.

To alleviate this problem, TanStack Router provides a scroll restoration component and hook that handle the process of monitoring, caching and restoring scroll positions for you.

It does this by:

- Monitoring the DOM for scroll events
- Registering scrollable areas with the scroll restoration cache
- Listening to the proper router events to know when to cache and restore scroll positions
- Storing scroll positions for each scrollable area in the cache (including `window` and `body`)
- Restoring scroll positions after successful navigations before DOM paint

”🧠 *Why is the loader not split?*”

- *“The loader is already an asynchronous boundary, so you pay double to both get the chunk and wait for the loader to execute.”*
- *“Categorically, it is less likely to contribute to a large bundle size than a component.”*
- *“The loader is one of the most important preloadable assets for a route, especially if you're using a default preload intent, like hovering over a link, so it's important for the loader to be available without any additional async overhead.*

Knowing the disadvantages of splitting the loader, if you still want to go ahead with it, head over to the [Data Loader Splitting](#) section.”

Code-Based Routing



Tip

Code-based routing is not recommended for most applications. It is recommended to use [File-Based Routing](#) instead.

⚠ Before You Start

- If you're using [File-Based Routing](#), skip this guide.
- If you still insist on using code-based routing, you must read the [Routing Concepts](#) guide first, as it also covers core concepts of the router.

app/routes/product.tsx

```
1 import type { Route } from "../types/product";
2 // types generated for this route 📁
3
4 export function loader({ params }: Route.LoaderArgs) {
5   // 📁 { id: string }
6   return { planet: `world #${params.id}` };
7 }
8
9 export default function Component({
10   loaderData, // 📁 { planet: string }
11 }: Route.ComponentProps) {
12   return <h1>Hello, {loaderData.planet}!</h1>;
13 }
```

```
<Link
  to=""
  clas
  para
  pr
  >
  <div className="text-4xl font-thin opacity-30 tabular-nums">
```

```
export const Route = createFileRoute("/products/$productId")({
  component: RouteComponent,
});

function R const productId: string
  const { productId } = Route.useParams();
  const { data, isLoading } = useGetProduct(+productId);
```

```

// route("/list", "./list.tsx")
import { Form } from "react-router";
import { TodoList } from "~/components/TodoList";

// this data will be loaded after the action completes...
export async function loader() {
  const items = await fakeDb.getItems();
  return { items };
}

// ...so that the list here is updated automatically
export default function Items({ loaderData }) {
  return (
    <div>
      <List items={loaderData.items} />
      <Form method="post" navigate={false} action="/list">
        <input type="text" name="title" />
        <button type="submit">Create Todo</button>
      </Form>
    </div>
  );
}






export async function action({ request }) {
  const data = await request.formData();
  const todo = await fakeDb.addItem({
    title: data.get("title"),
  });
  return { ok: true };
}

```

```

about copy.tsx U X
src > routes > about copy.tsx > ...
1  import { createFileRoute } from "@tanstack/react-router";
2
3  export const Route = createFileRoute("/about")({
4    |   component: Index,
5    | });
6
7  function Index() {
8    |   return (
9    |     <div className="p-2">
10    |       <h3>Welcome Home!</h3>
11    |     </div>
12    |   );
13  }

```

-  1st-class, built-in, and ready to use with no added configuration or code
-  Partial Support (on a scale of 5)
-  Supported via addon/community package
-  Possible, but requires custom code/implementation/casting
-  Not officially supported

Path Params		
Typesafe Path Params		
Typesafe Route Context		
Path Param Validation		
Custom Path Param Parsing/Serialization		
Ranked Routes		
Active Link Customization		
Optimistic UI		
Typesafe Absolute + Relative Navigation		 (1/5 via <code>buildHref</code> util)
Route Mount/Transition/Unmount Events		
Devtools		
Basic Search Params		
Search Param Hooks		
<code><Link/></code> / <code>useNavigate</code> Search Param API		 (search-string only via the <code>to</code> / <code>search</code> options)
JSON Search Params		
TypeSafe Search Params		
Search Param Schema Validation		
Search Param Immutability + Structural Sharing		
Custom Search Param parsing/serialization		
Search Param Middleware		

Route Pending Elements		
<code><Block></code> / <code>useBlocker</code>		 (no hard reloads or cross-origin navigation)
Deferred Primitives		
Navigation Scroll Restoration		
ElementScroll Restoration		
Async Scroll Restoration		
Router Invalidation		
Runtime Route Manipulation (Fog of War)		
Parallel Routes		
--	--	--
Full Stack	--	--
SSR		
Streaming SSR		
Generic RPCs		
Generic RPC Middleware		
React Server Functions		
React Server Function Middleware		
API Routes		
API Middleware		
React Server Components		 (Experimental)
<code><Form></code> API		

감사합니다.