

TanStack Start: A Fresh Take on React

Tal Moskovich
React IL 02/2025

Hello!

Tal Moskovich

Committing Code & Pushing Personal
Boundaries

💻 Frontend Engineer @ ImagenAI

🎧 Podcaster @ Lo-Techi

<https://lotechni.dev>

<https://www.linkedin.com/in/tmosko>



Meet TanStack

Open Source Libraries

TANSTACK START

*Full-stack React Framework
powered by TanStack Router*

Full-document SSR, Streaming, Server Functions, bundling and more, powered by TanStack Router, Nitro, Vite and ready to deploy to your favorite hosting provider.

TANSTACK ROUTER

Type-safe Routing for React applications.

A powerful React router for client-side and full-stack react applications. Fully type-safe APIs, first-class search-params for managing state in the URL and seamless integration with the existing React ecosystem.

TANSTACK QUERY

Powerful asynchronous state management, server-state utilities and data fetching

Powerful asynchronous state management, server-state utilities and data fetching. Fetch, cache, update, and wrangle all forms of async data in your TS/JS, React, Vue, Solid, Svelte & Angular applications all without touching any "global state".

TANSTACK TABLE

Headless UI for building powerful tables & datagrids

Supercharge your tables or build a datagrid from scratch for TS/JS, React, Vue, Solid, Svelte, Qwik, Angular, and Lit while retaining 100% control over markup and styles.

TANSTACK FORM

Headless UI for building performant and type-safe forms

Headless, performant, and type-safe form state management for TS/JS, React, Vue, Angular, Solid and Lit

TANSTACK VIRTUAL

Headless UI for Virtualizing Large Element Lists

Virtualize only the visible content for massive scrollable DOM nodes at 60FPS in TS/JS, React, Vue, Solid, Svelte, Lit & Angular while retaining 100% control over markup and styles.

TANSTACK RANGER

Headless range and multi-range slider utilities.

Headless, lightweight, and extensible primitives for building range and multi-range sliders.

TANSTACK STORE

Framework agnostic data store with reactive framework adapters

The immutable-reactive data store that powers the core of TanStack libraries and their framework adapters.

TANSTACK CONFIG

Configuration and tools for publishing and maintaining high-quality JavaScript packages

The build and publish utilities used by all of our projects. Use it if you dare!

The Magic Behind TanStack Start

The Magic Behind TanStack Start

- TanStack Router

The Magic Behind TanStack Start

- TanStack Router
- TanStack Query

The Magic Behind TanStack Start

- TanStack Router
- TanStack Query
- Vinxi -> Nitro -> Vite

The Magic Behind TanStack Start

- TanStack Router
- TanStack Query
- Vinxi -> Nitro -> Vite



Tanner Linsley

Why Do We Need Another Meta-Framework?

Why Do We Need Another Meta-Framework?

Client-First Philosophy

- Natural React development experience
- No "use client" directive chaos
- Opt-in to server features, not opt-out

Why Do We Need Another Meta-Framework?

Client-First Philosophy 🎯

- Natural React development experience
- No "use client" directive chaos
- Opt-in to server features, not opt-out

```
// Natural React code without directives
export function Component() {
  return <div>Client-side by default</div>
}
```

Why Do We Need Another Meta-Framework?

Cache = TanStack Query 

- No new caching system to learn
- Same patterns you already use in production

Why Do We Need Another Meta-Framework?

Cache = TanStack Query

- No new caching system to learn
- Same patterns you already use in production

```
// Works seamlessly with loaders
export const route = createFileRoute('/users')({
  loader: async () => {
    // Same patterns, same caching, zero learning curve
    return queryClient.ensureQueryData({
      queryKey: ['users'],
      queryFn: getUsers
    });
  }
});
```

Why Do We Need Another Meta-Framework?

Smart Data Handling - No More Full Page Refreshes

- No more "all or nothing" data fetching approach
- No more waiting for slow data to show fast data
- Only refetch what actually changed

Why Do We Need Another Meta-Framework?

Smart Data Handling - No More Full Page Refreshes

- No more "all or nothing" data fetching approach
- No more waiting for slow data to show fast data
- Only refetch what actually changed

```
// Update user profile without refreshing entire page
export const updateProfile = createServerFunction('post')(
  async (data) => {
    await saveProfile(data);
    // Only invalidate profile data, everything else stays cached
    queryClient.invalidateQueries({
      queryKey: ['profile', userId]
    });
});
```

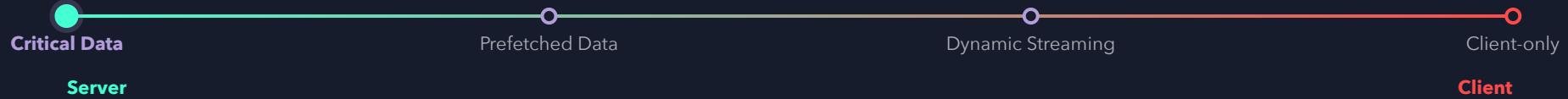
Why Do We Need Another Meta-Framework?

Incremental Adoption

- Easy migration path from existing apps
- Start with client-side features
- Add server capabilities gradually
- No need for full rewrites

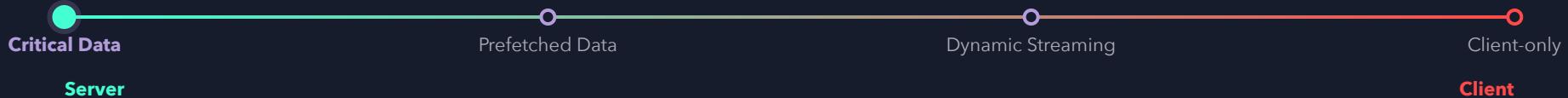
Demo

Data Fetching



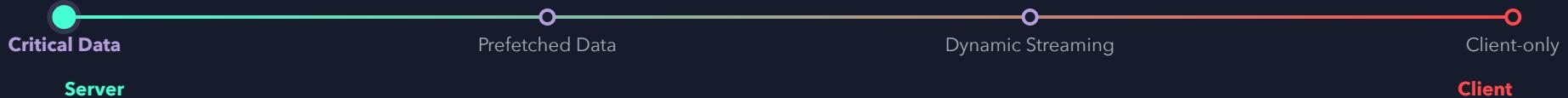
Data Fetching

```
1 // Fetching critical data on server-side
2 export const route = createRoute({
3   loader: async () => {
4     await queryClient.ensureQueryData(postsQuery)
5   },
6   component: () => {
7     const query = useQuery(postsQuery)
8     // Data is already available from loader
9     return <PostsList posts={query.data} />
10  }
11});
```



Data Fetching

```
1 // We can use `fetch` directly in the loader instead of `TanStack Query`  
2 export const route = createRoute({  
3   loader: async () => {  
4     posts: await fetchPosts()  
5   },  
6   component: () => {  
7     const { posts } = useLoaderData()  
8     return <PostsList posts={posts} />  
9   }  
10});
```



Data Fetching

```
1 // Prefetch data with deferred loading using `SuspenseQuery`  
2 export const route = createRoute({  
3   loader: async () => {  
4     queryClient.prefetchQuery(postsQuery)  
5   },  
6   component: () => {  
7     const { data } = useSuspenseQuery(postsQuery)  
8     return <PostsList posts={data} />  
9   }  
10});
```



Data Fetching

```
1 // Deferred loading with `Await` component
2 export const route = createRoute({
3   loader: async () => {
4     postsPromise: fetchPosts()
5   },
6   component: () => {
7     const { postsPromise } = useLoaderData()
8     return (
9       <Await
10         promise={postsPromise}
11         children={posts => {
12           return <PostsList posts={posts} />
13         }}
14       />
15     )
16   }
17});
```



Data Fetching

```
// Dynamic streaming  
  
const { data } = useSuspenseQuery(postsQuery)
```



Data Fetching

```
// Client-only data fetching  
  
const { data } = useQuery(postsQuery)
```



Demo + Slides



Thank You!



Tal Moskovich



lotechni.dev