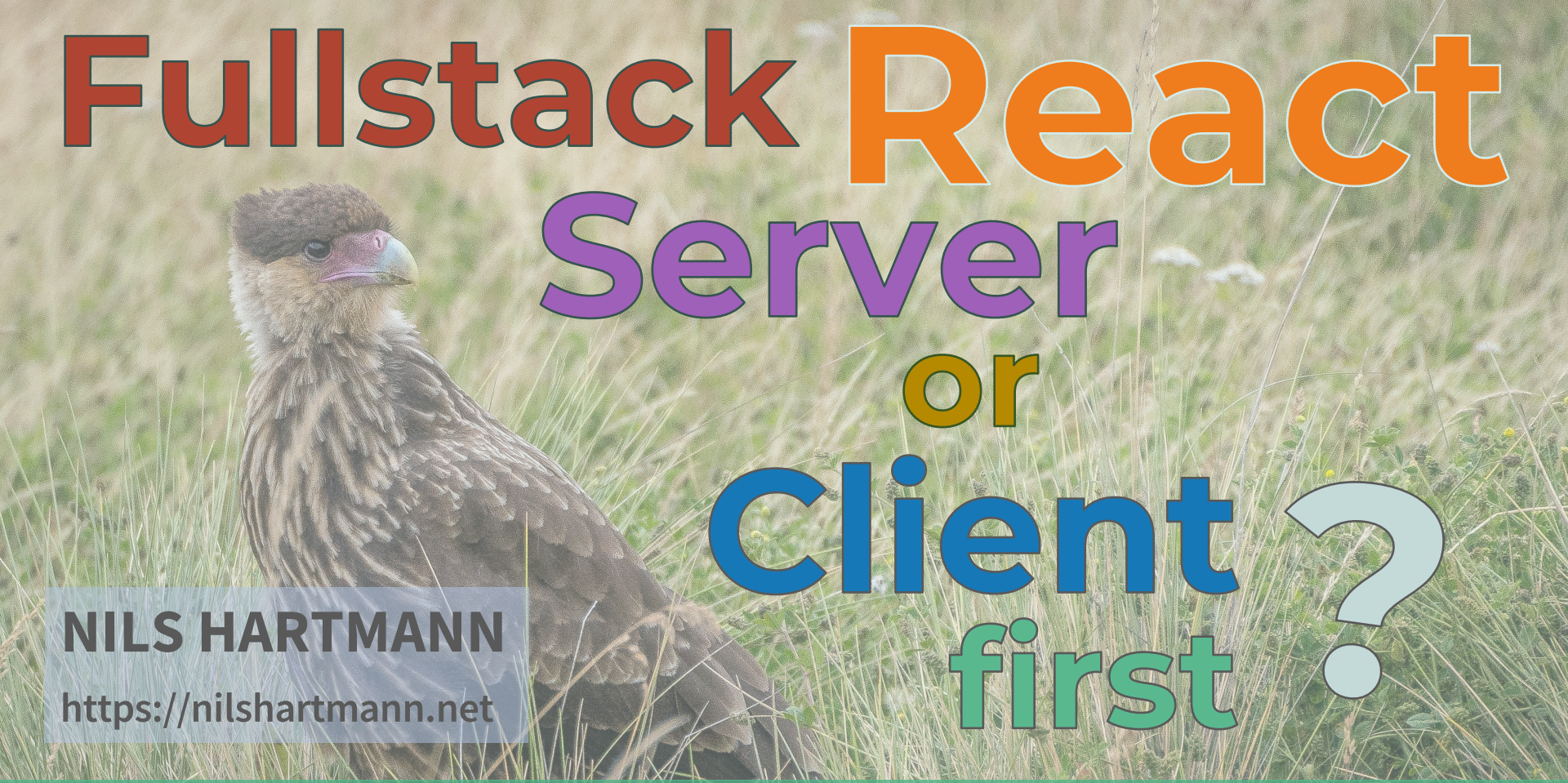
Fullstack React

Server or Client first?



NILS HARTMANN

https://nilshartmann.net

C‘T WEBDEV | COLOGNE | NOVEMBER 19, 2025

NILS HARTMANN

nils@nilshartmann.net

Freelance Software Developer

Java, Spring, GraphQL, React, TypeScript

https://graphql.schule/video-kurs

https://reactbuch.de

Our Stack

for today

KÜRZEN

TS Query ans Ende

Client-first

Data Fetching

TanStack Query

(aka React Query)

Client-first

Data Fetching

TanStack Query

(aka React Query)

optional, but typical

Client-first

Data Fetching

TanStack Query

(aka React Query)

Routing

TanStack Router

Client-first

Data Fetching

TanStack Query

(aka React Query)

Routing

TanStack Router

Fullstack

TanStack Start

Client-first

TanStack Start = Router + Server

https://tanstack.com/start

Currently RC

Vite Plug-in, built on top of TanStack Router

Adds: serverside rendering, streaming, server functions

No React Server Components (yet)


Server-first

Server-first

Next.js

https://nextjs.org

Currently Next.js 16, based on React 19.2

Support for all most all client and server React features

Including React Server Components and Server Functions

Side

by

Side

COMPARISON

Server-side rendering (SSR)

Client-first

Demo Überlegung

- Route anlegen
- Man sieht, wie der Router funktioniert
- TANSTACK
  - Dev mode
  - HTML beim ersten Request
  - TanStack Query Cache wird befüllt
  - Danach Client-Seite, Navigation hi– und her: Schnell wegen Cache
- NEXT
  - React Server components

Server-side rendering (SSR)

TanStack Start

Client-first

- „Classic“ SSR
- Only initial request is server-side rendered
- HTML is returned
- Complete JavaScript Code of the App is returned and hydrated

Server-side rendering (SSR)

Next.js

Server-first

- „Classic“ SSR plus React Server Components (RSC)
- First request returns HTML
- RSC Payload is returned on
- Any navigation (route chan server request!
- Only client components are hydrated

React Server components!!!!!!

Eigenes kapitel vergleich mit client Components
=> Build zeigen

Server-side rendering (SSR)

Next.js

Server-first

- „Classic“ SSR plus React Server Components (RSC)
- First request returns HTML
- RSC Payload is returned on all request
- Only Client Components are hydrated
- Any navigation (route change) is server request!
- State in Client Components survives

Routing

TanStack Start

Client-first

- Based on TanStack Router (replacement for React Router)
- Vite plug-in
- Filebased routing
- Highly typesafe for routes, params and search params

Routing

Next.js

Server-first

- Built-in file-based Routing
- Params and search params and are promises passed as props

Data fetching

TanStack Start

Client-first

- Use library of your choice (typically: TanStack Query)
- Data can be pre-loaded in loader functions of each Route
- Support of Suspense and Streaming

Data fetching

Next.js

Server-first

- Typically in async React Server Components
- When using in Client Components, it‘s up to you
- You can load data for Client Components in RSC and pass data as props
  - A few restrictions for properties apply (serializable!)
  - You can even pass promises from server to client

Server functions

TanStack Start

Client-first

- Defines explicit API (function) for defining server functions
  - You can specify http method (post or get)
  - Handler for validating input arguments
- Almost all argument types are supports
- When calling your server function you need to pass a single argument
- You can call your server function wherever you want

Server functions

Next.js

Server-first

- „Official“ React Server Function (aka Server Actions)
- Server functions can be inlined in Server Components or files
- Defined using a „use server“ directive
- Only for write operations (do not call while rendering)

SSG

TanStack Start

Client-first

- Possible but probably imho no first class support

SSG

Next.js

Server-first

- Can be used to fully generate static pages in build
- You can even partially generate pages
- Aggressive (and complex...) caching

Pre-rendering/pre-loading

TanStack Start

Client-first

- Loaders are executed before rendering
- <Link /> components determines pre-loading behaviour
- Default behaviour can also be set globally

Pre-rendering/pre-loading

Next.js

Server-first

- Per-default enabled for all internal links
- Is that a good choice?
- To change behaviour you have to set the property on every <Link />-component

Global state

TanStack Start

Client-first

- Works with any global state solution, as state is only on the client
- For populating global state during SSR, your library must support it

Global state

Next.js

Server-first

- You can use global state for client components
- Somehow complicated: we have server round trips on any navigation
- Client-side global state can differ from server
- Example:
  - On navigation on server side new data is read and rendered
  - Client-side global state doesn‘t know about that
  - UI is inconsistent

Summary

Summary

Next.js Server-first

- First: Do you really need server-side React at all?
- Next.js:
  - Probably nearest to React ideas and APIs
  - Shines if you need aggressive caching, prerendering etc.
  - Suitable for almost static apps (ecomerce, marketing)
- TanStack Start:
  - APIs are more explizit (createServerFn)
  - Concept easier (?) without RSC (not „two worlds“)
  - Suitable for interaktive apps that needs SSR (SEO etc.)

NILS HARTMANN
https://nilshartmann.net

Thanks a lot!

Code & Slides: https://react.schule/ijs-2025-tanstack

Questions and contact: nils@nilshartmann.net

My workshops (de): https://nilshartmann.net/workshops