A photograph of a bird of prey, likely a caracara, perched in a field of tall, golden-brown grass. The bird has dark brown feathers on its back and wings, with a white patch on its wing. Its head is light-colored with a pinkish-red cere and a hooked beak. The background is a soft-focus view of more grass and some low-lying plants.

# Fullstack React Server or Client first?

**NILS HARTMANN**

<https://nilshartmann.net>

# NILS HARTMANN

nils@nilshartmann.net

## Freelance Software Developer

Java, Spring, GraphQL, React, TypeScript



<https://graphql.schule/video-kurs>

<https://reactbuch.de>

[HTTPS://NILSHARTMANN.NET](https://nilshartmann.net)

# Fullstack React

Server

What is  
that? 😱

NILS HARTMANN

<https://nilshartmann.net>

# „Fullstack“ React

My definition

Client-first

# „Fullstack“ React

## My definition

Client-first

- An application contains UI/Rendering code on both **server** and **client** side

## My definition

Client-first

- An application contains UI/Rendering code on both **server** and **client** side
- For both sides **a** framework is used

## My definition

Client-first

- An application contains UI/Rendering code on both **server** and **client** side
- For both sides **a** framework is used
- Framework contains solutions for rendering, routing, data fetching, ...

## My definition

Client-first

- An application contains UI/Rendering code on both **server** and **client** side
- For both sides **a framework** is used
- Framework contains solutions for rendering, routing, data fetching, ...



- React itself provides **low-level APIs** for server-side features
- But: you need a **framework** to make use of it

# Our Candidates

CLIENT-FIRST

Client-first

Client-first

# TanStack Start

<https://tanstack.com/start>

Currently RC

Vite Plug-in

Supports: serverside rendering (SSR), streaming, server functions

No React Server Components (yet)



# TanStack Start = Router + Server

## Idea:

Build a „traditional“ SPA with TanStack Router

Move to serverside by adding TanStack Start

(ofc you can start with Start from the beginning)

Only parts of your logic run on serverside

# TanStack Router

<https://tanstack.com/router>

Alternative / replacement for React Router

Can be used for „traditional“ client-side SPAs

Vite Plug-in, file-based routing, exceptional TypeScript support

SERVER-FIRST

Server-first

Server-first

# Next.js

<https://nextjs.org>

Currently Next.js 16, based on newest React 19.2

Support for all most all client and server React features

Including React Server Components (RSC) and Server Functions

Server-first

# Next.js

## Idea

(Almost\*) all of your app runs on serverside

Only (small\*) parts run on the client

Aggressive caching and performance optimizations for fastest UI

\* My interpretation / opinion

# Demo

# Time



# Our app...

**http://localhost:3000**

The screenshot shows a web browser window titled "Donutgram" with the URL "http://localhost:3000/donuts/1". The page has a pink gradient background. At the top right is a "Donuts" logo. Below it is a large image of a cinnamon roll with pink frosting and colorful sprinkles. The title "Cinnamon Stack Overflow" is displayed above the image. A text box below the image contains the message: "Too many cinnamon layers, not enough memory - please reboot (and take a bite.)". To the right of the main image is a sidebar with a header "What the Snackers Say". It contains two text bubbles: one saying "The merge was sweeter than expected. Still caused a conflict in my stomach. (gitgourmet)" and another saying "It forked my heart. (issue\_lover)". A small "74" with a heart icon is at the bottom left of the sidebar.

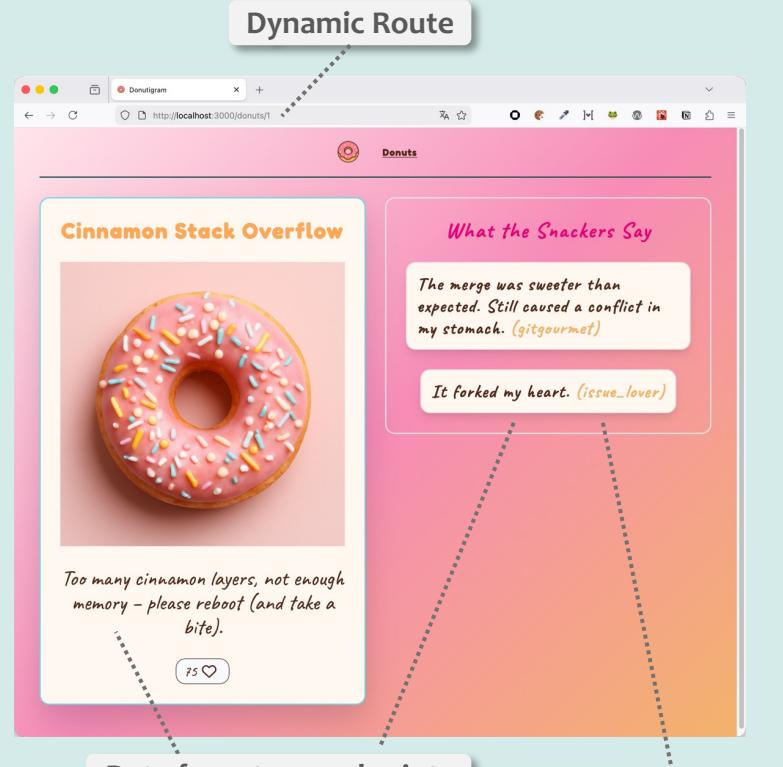
# Demo 1

The „important“ features  
in action

Routing

Data Fetching

Serverside rendering



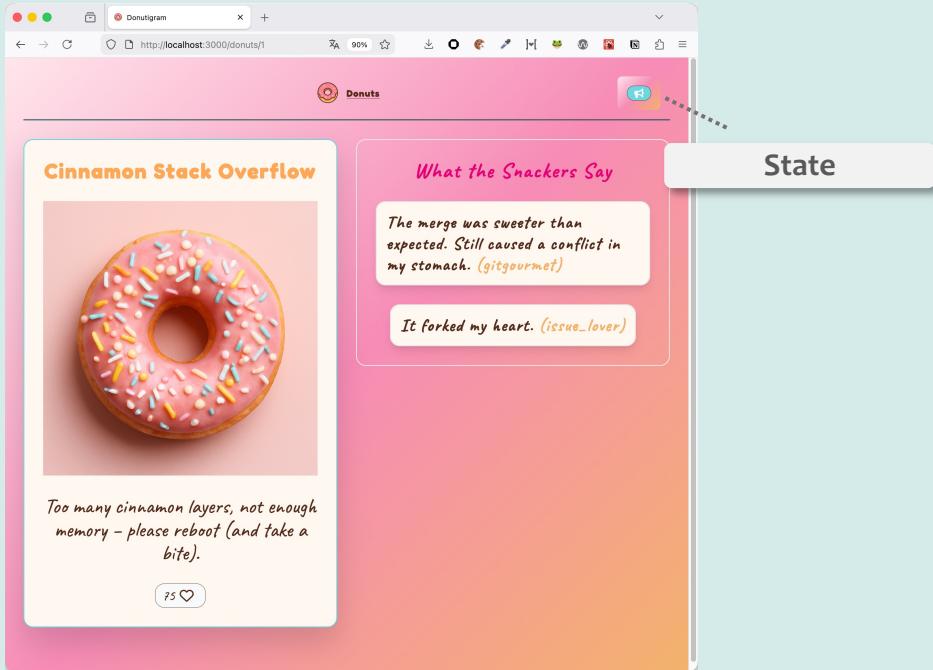
Data from two endpoints

Streaming

# Demo 2

## Interactivity

### Client-side state

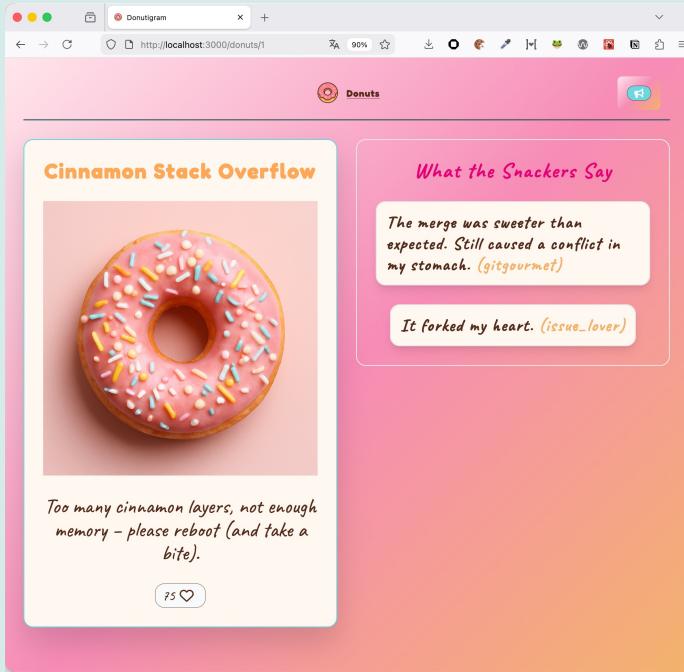


# Demo 3

## Server functions

aka

## Server actions



# Side

by

# Side

COMPARISON

# 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 all request
- Only Client Components are hydrated
- Any navigation (route change) is server request!
- State in Client Components survives

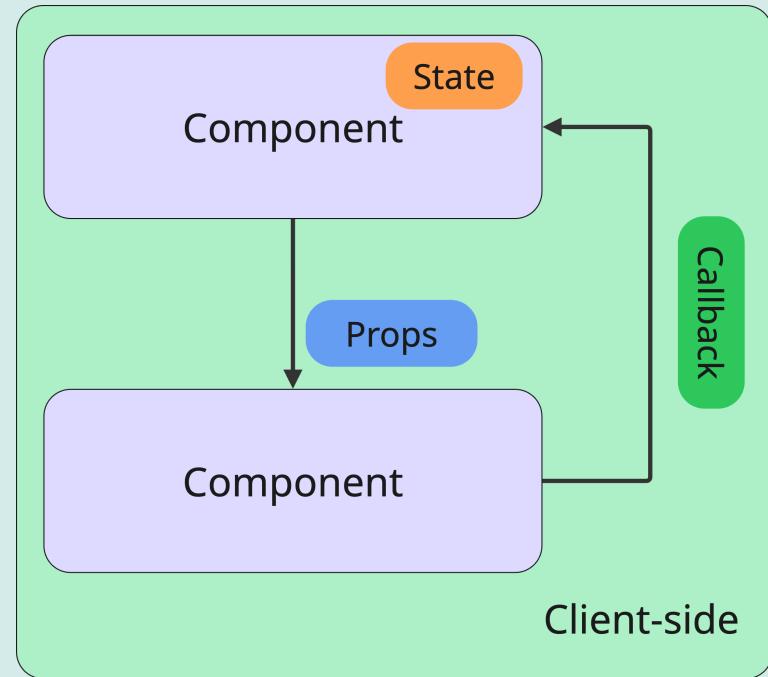


# Architecture

## TanStack Start

Client-first

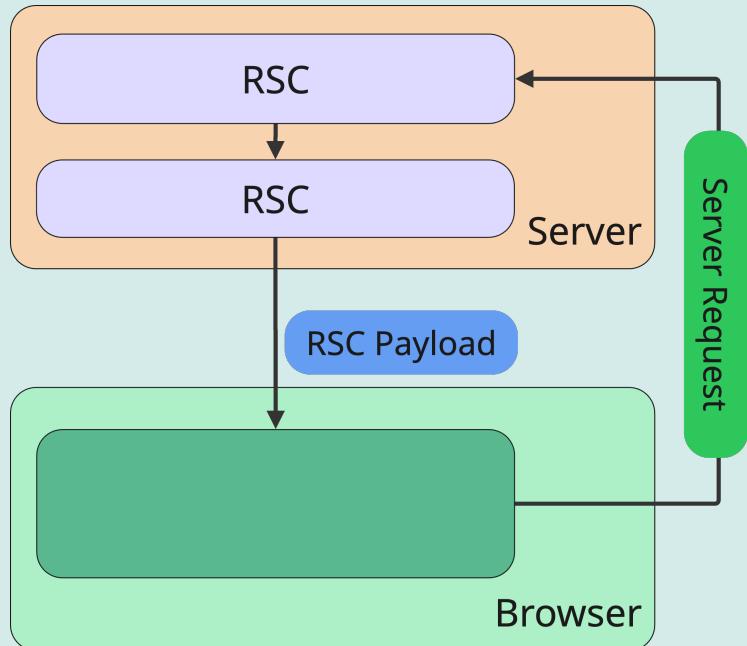
- You **data** is available in the client
- You code follows „traditional“ React architecture
- Typical render cycle



# Architecture

## Next.js Server-first

- Typically you have „rendered UI“ on client
- No data (exception: client components)
- To re-render, server roundtrip might be necessary
- Example:
  - Re-ordering a list



## 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

## 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

TanStack Start

Client-first

Server

```
const likeAction = createServerFn({ method: "POST" })  
  
.inputValidator((data) => {  
  if (typeof data === "string") {  
    return data;  
  }  
  
  throw new Error("Invalid data");  
})  
  
.handler(async ({ data: donutId }) => {  
  const response = await ky.put(`...`)  
  return DonutDto.parse(response);  
});
```

Client

```
function doLike(donutId: string) {  
  return saveLikeAction({  
    data: donutId,  
  });  
}
```

# 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)

## Static Side Generation (SSG)

### TanStack Start

Client-first

- Supported
- Does not seem to be a first-class use case
- For generation, your complete side is crawled

# Static Side Generation (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

# Summary

WHAT TO CHOOSE?

## Summary

### Summary

- First: Do you really need server-side React at all?

## Summary

### Summary

- 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.
  - Can be tricky with traditional React patterns, like often changing global state

# Summary

## Summary

- 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.
  - Can be tricky with traditional React patterns, like often changing global state
- TanStack Start:
  - APIs are more explicit (createServerFn)
  - Concept easier (?) without RSC (not „two worlds“)
  - Suitable for interactive apps that need SSR (SEO etc.)

# Thanks a lot!

Code & Slides: <https://react.schule/ctwebdev2025-fullstack>

Questions and contact: [nils@nilshartmann.net](mailto:nils@nilshartmann.net)

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

will publish the source code later or tomorrow (please be patient)

