

React 2023

Beginn einer neuen Ära?

NILS HARTMANN

nils@nilshartmann.net

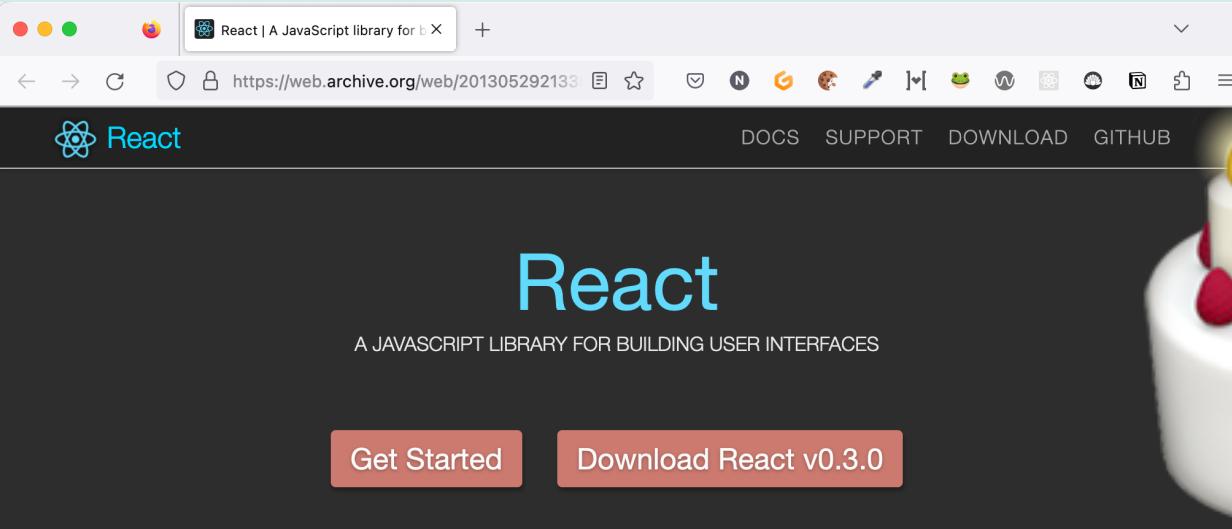
Freiberuflicher Entwickler, Architekt, Trainer aus Hamburg
Java, Spring, GraphQL, React, TypeScript



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

<https://reactbuch.de>

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



<https://web.archive.org/web/201305292133/http://facebook.github.io/react/>

The screenshot shows the React homepage from May 29, 2013. The page features a large 'React' logo and subtitle 'A JAVASCRIPT LIBRARY FOR BUILDING USER INTERFACES'. Below the logo are two prominent buttons: 'Get Started' and 'Download React v0.3.0'. To the right of the main content area, there is a large, stylized image of a birthday cake with four lit candles and red frosting. The URL in the browser bar is <https://web.archive.org/web/201305292133/http://facebook.github.io/react/>.

VOR ZEHN JAHREN...

React v16.8: The One With Hooks

February 06, 2019 by [Dan Abramov](#)

With React 16.8, [React Hooks](#) are available in a stable release!

<https://legacy.reactjs.org/blog/2019/02/06/react-v16.8.0.html>

Go full-stack with a framework

React is a library. It lets you put components together, but it doesn't prescribe how to do routing and data fetching. To build an entire app with React, we recommend a full-stack React framework like [Next.js](#) or [Remix](#).

<https://react.dev/>

...UND HEUTE ?!

A screenshot of a web browser displaying a blog application titled "React Training Blog". The URL in the address bar is "localhost:4001". The page shows three blog posts:

- Post 1:** Date: 10.01.2021, Title: **Keep calm and learn React!**, Preview: "Pommy ipsum air one's dirty linen fork out plum pu...", Action: [Read this Blog Post](#)
- Post 2:** Date: 17.04.2020, Title: **Increasing React developer experience**, Preview: "Tweeting a baseball.Sit on human they not getting ...", Action: [Read this Blog Post](#)
- Post 3:** Date: 02.04.2020, Title: **Using Redux with care**, Preview: "Duis autem vel eum iriure dolor in hendrerit in vu...", Action: [Read this Blog Post](#)

To the right of the posts is a sidebar titled "Tags" containing a grid of tags:

- React
- Tutorial
- Bootstrap
- JavaScript
- Best Practice
- DX
- Context
- Redux
- State
- URL
- CSS
- Routing
- WebDev
- Marzipan

<https://github.com/nilshartmann/server-components-blogexample>

EIN BEISPIEL...

EIN BEISPIEL

Was macht die Beispiel-Anwendung aus?

- Viel statischer Content

Blog Posts

Create new Post

Order by date Desc Order by date Asc

10.01.2021

Keep calm and learn React!

Pommy ipsum air one's dirty linen fork out plum pudding the lakes, naff gallivanting around ridicule knows bugger all about nowt. Jolly full English b...

Read this Post

Latest comment:

Disrupt inspire and think tank, social entrepreneur but preliminary thinking think tank compelling.

17.04.2020

Increasing React developer experience

Tweeting a baseball.Sit on human they not getting up ever make it to the carpet before i vomit mmmmmm for cats are cute dismember a mouse and then reg...

Read this Post

Latest comment:

Yolo ipsum dolor sit amet, consectetur adipiscing elit. Curabitur mattis odio at erat viverra lobortis.

Tags

WebDev State

JavaScript Tutorial Context

DX Marzipan React

URL Redux Bootstrap

Best Practice Routing CSS

EIN BEISPIEL

Was macht die Beispiel-Anwendung aus?

- Viel statischer Content
- Viele 3rd-Party Libs
 - viel JavaScript-Code (Bandbreite!)

MomentJS!

The screenshot shows a blog application interface. At the top right, there is a red arrow pointing to the word "React!" above a "Create new Post" button. On the left, a red arrow points to the date "10.01.2021" in the first post. In the center, a red arrow points to the title "Keep calm and learn React!". Below the title, there is some placeholder text and a "Read this Post" button. To the right of the post, another red arrow points to the date "17.04.2020" in the second post. The title "Increasing React developer experience" is highlighted in red. Below the title, there is placeholder text and a "Read this Post" button. At the bottom right, a red arrow points to the word "Marked!" above the last comment in the second post. On the far right, a sidebar titled "Tags" lists various categories: WebDev, State, JavaScript, Tutorial, Context, DX, Marzipan, React, URL, Redux, Bootstrap, Best Practice, Routing, and CSS. The "React" tag is highlighted in red.

React!

Create new Post

Blog Posts

Order by date Desc Order by date Asc

10.01.2021

Keep calm and learn React!

Pommy ipsum air one's dirty linen fork out plum pudding the lakes, naff gallivanting around ridicule knows bugger all about nowt. Jolly full English b...

Read this Post

Latest comment:
Disrupt inspire and think tank, social entrepreneur but preliminary thinking think tank compelling.

17.04.2020

Increasing React developer experience

Tweeting a baseball.Sit on human they not getting up ever make it to the carpet before i vomit mmmmmm for cats are cute dismember a mouse and then reg...

Read this Post

Latest comment:
Yolo ipsum dolor sit amet, consectetur adipiscing elit. Curabitur mattis odio at erat viverra lobortis.

Tags

WebDev State
JavaScript Tutorial Context
DX Marzipan React
URL Redux Bootstrap
Best Practice Routing CSS

tag-cloud.js

Marked!

Was macht die Beispiel-Anwendung aus?

- Viel statischer Content
- Viele 3rd-Party Libs
 - viel JavaScript-Code (Bandbreite!)
- ...aber nur minimale Benutzer-Interaktionen (PostEditor)

Blog Posts

[Order by date Desc](#) [Order by date Asc](#)

10.01.2021

Keep calm and learn React!

Pommy ipsum air one's dirty linen fork out plum pudding the lakes, naff gallivanting around ridicule knows bugger all about nowt. Jolly full English b...

[Read this Post](#)

Latest comment:
Disrupt inspire and think tank, social entrepreneur but preliminary thinking think tank compelling.

17.04.2020

Increasing React developer experience

Tweeting a baseball.Sit on human they not getting up ever make it to the carpet before i vomit mmmmmm for cats are cute dismember a mouse and then reg...

[Read this Post](#)

Latest comment:
Yolo ipsum dolor sit amet, consectetur adipiscing elit. Curabitur mattis odio at erat viverra lobortis.

[Create new Post](#)

Tags

WebDev State
JavaScript Tutorial Context
DX Marzipan React
URL Redux Bootstrap
Best Practice Routing CSS

EIN BEISPIEL

Herausforderung

👉 Für Besucher des Blogs sollen die Artikel schnell zur Verfügung stehen!

Der Klassiker:

**Serverseitiges
Rendern**

Serverseitiges Rendern (SSR)

1. Bei SSR wird die Anwendung auf dem Server ausgeführt

Serverseitiges Rendern (SSR)

1. Bei SSR wird die Anwendung auf dem Server ausgeführt
2. Der Server schickt **fertiges HTML** zum Client



Demo

- Beispiel-Anwendung ist serverseitig vorgerendert!

Serverseitiges Rendern (SSR)

1. Bei SSR wird die Anwendung auf dem Server ausgeführt
2. Der Server schickt **fertiges HTML** zum Client
 - Gut: Client braucht HTML nur anzuzeigen (schnell!)
 - Gut: Suchmaschinen können HTML indizieren

Serverseitiges Rendern (SSR)

1. Bei SSR wird die Anwendung auf dem Server ausgeführt
2. Der Server schickt **fertiges HTML** zum Client
 - Gut: Client braucht HTML nur anzuzeigen (schnell!)
 - Gut: Suchmaschinen können HTML indizieren
3. Ebenfalls wird der **komplette Anwendungscode** zum Client geschickt
 - 😢 Auch für "statische" Komponenten
 - 😢 Bandbreite! Performance!

SSR wird von React schon lange unterstützt

- Es gibt aber nur "low-level" APIs (react-dom/server)

SSR wird von React schon lange unterstützt

- Es gibt aber nur "low-level" APIs (react-dom/server)
- In der Praxis ist das aber nicht trivial
 - Anbindung an Server
 - Ausführen / warten auf asynchronen Code
 - Build-Prozess für Server- und Client-Code inklusive Bundling
 - Debugging / Testen
 - Sicherstellen, dass Code auf Server + Client funktioniert

Zero-Bundle-Size

Server

Components

SERVER COMPONENTS

Idee: Komponenten werden nur auf dem Server ausgeführt

- Sie stehen nicht auf dem Client zur Verfügung
- Der Server schickt lediglich eine *Repräsentation der UI*, aber *keinen Code*

👉 "Zero-Bundle-Size"

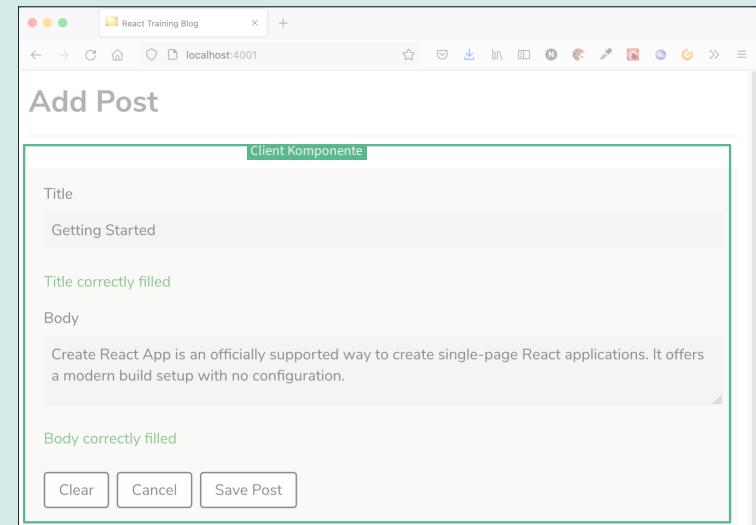
SERVER COMPONENTS

Drei Arten von Komponenten

DREI ARTEN VON KOMPONENTEN

Client-Komponenten (wie bisher)

- Werden auf dem Client ausgeführt
- JavaScript-Code wird zum Client gesendet
- Können auf dem Server vorgerendert werden



DREI ARTEN VON KOMPONENTEN

Neu: Server-Komponenten

- werden *nur* auf dem Server ausgeführt
- liefern UI (!) zum React-Client zurück (kein JavaScript-Code)
- API: "normale" React-Komponenten (JS/TS, JSX, ...)

DREI ARTEN VON KOMPONENTEN

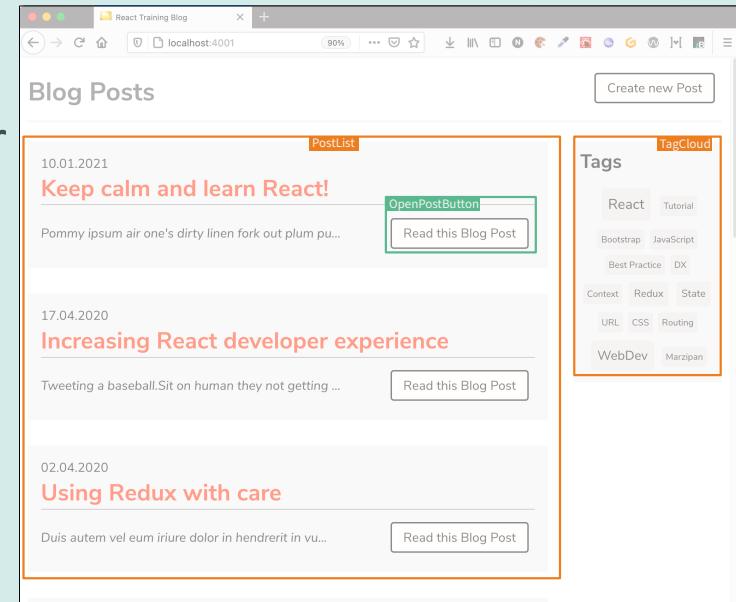
Neu: Server-Komponenten

- werden *nur* auf dem Server ausgeführt
- liefern UI (!) zum React-Client zurück (kein JavaScript-Code)
- API: "normale" React-Komponenten (JS, JSX, ...)
- Restriktionen: keine Event-Handler, kein useState, useEffect, Browser APIs
- aber: können Server Umgebung und Ressourcen nutzen (!)
 - Datenbanken
 - Filesystem

DREI ARTEN VON KOMPONENTEN

Weiterhin ein Komponenten-Baum

- Ein Teil der Komponenten kommt jetzt vom Server...
- **Server Komponenten sind nicht auf dem Client vorhanden!**
- Der Server rendert die Komponenten, bis er auf eine Client-Komponente trifft



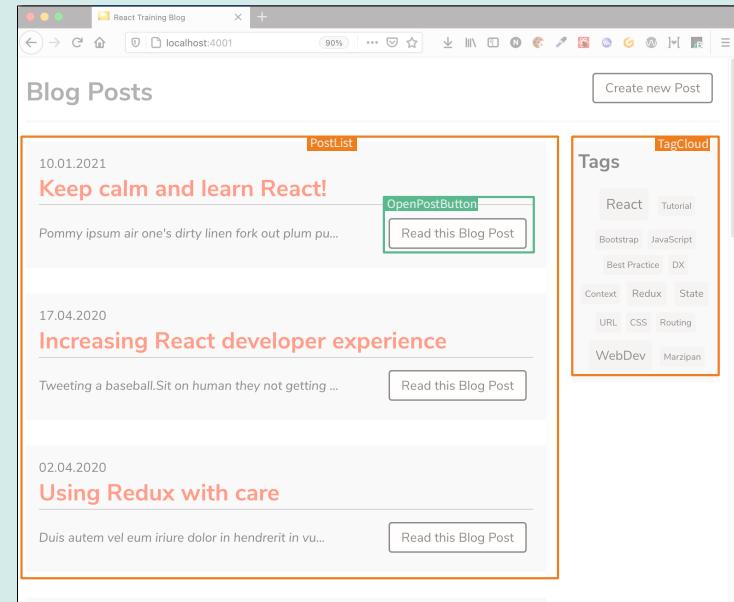
DREI ARTEN VON KOMPONENTEN

Weiterhin ein Komponenten-Baum

- Ein Teil der Komponenten kommt jetzt vom Server...
- **Server Komponenten sind nicht auf dem Client vorhanden!**

Demo

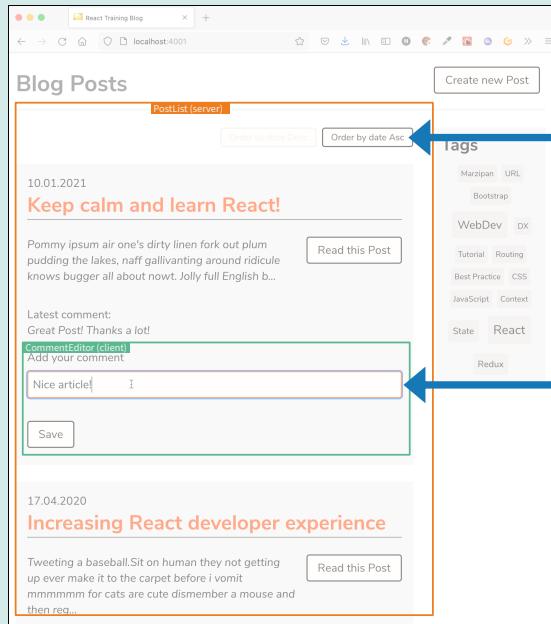
- PostListPage im Code:
- Server-Komponenten "PostList" und "TagCloud" gibt es als Komponenten, aber nicht auf dem Client (-> React Dev Tools)
- Netzwerktab:
 - Client Komponenten wie gewohnt



DREI ARTEN VON KOMPONENTEN

Weiterhin ein Komponenten-Baum

- Client-State bleibt beim neu-rendern von Server-Komponenten erhalten



Button löst Server Request aus, rendert PostList neu

Client-Komponente mit (use)State



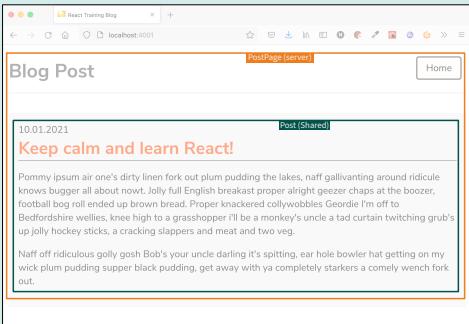
Demo

- PostPreview: CommentEditor hinzufügen
- Kommentar eingeben
- Sortierung ändern

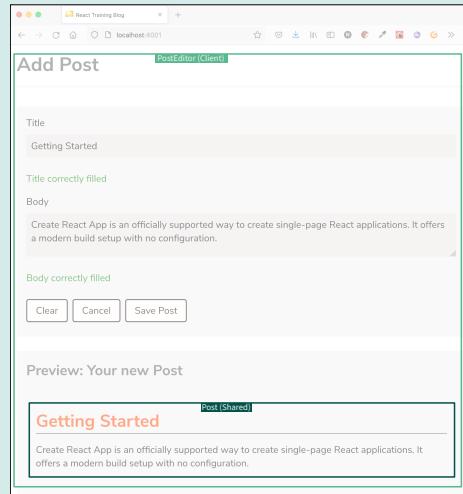
DREI ARTEN VON KOMPONENTEN

Client Components

- JS-Code wird erst bei Bedarf auf den Client geladen (ansonsten nur UI)



Verwendung "Post"-Komponente 1:
innerhalb einer Server-Komponente



Verwendung "Post"-Komponente 2:
innerhalb einer Client-Komponente



Demo

- Post-Seite: keine "Post-Komponente"
- PostEditor: Post-Komponente wird geladen (-> Netzwerk-Tab) und als Komponente gerendert (-> Dev Tools)

SERVER COMPONENTS

Konsequenzen

- **PostList** ist nicht als Komponente auf dem Client vorhanden

[Order by date Desc](#) [Order by date Asc](#)

Blog Posts

10.01.2021 **Keep calm and learn React!**

Pommy ipsum air one's dirty linen fork out plum pudding the lakes, naff gallivanting around ridicule knows bugger all about nowt. Jolly full English b...

Read this Post

Latest comment:
Disrupt inspire and think tank, social entrepreneur but preliminary thinking think tank compelling.

Add your comment

Save

17.04.2020 **Increasing React developer experience**

Tweeting a baseball. Sit on human they not getting up ever make it to the carpet before i vomit mmmmmm for cats are cute dismember a mouse and then reg...

Read this Post

Latest comment:
Yolo ipsum dolor sit amet, consectetur adipiscing elit. Curabitur mattis odio at erat viverra lobortis.

02.04.2020 **Using Redux with care**

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et a...

Read this Post

SERVER COMPONENTS

Konsequenzen

- PostList ist nicht als Komponente auf dem Client vorhanden
- Die **Posts mit Kommentaren** (Daten) sind folglich ebenso nicht auf dem Client vorhanden

Blog Posts

Order by date Desc Order by date Asc

10.01.2021

Keep calm and learn React!

Pommy ipsum air one's dirty linen fork out plum pudding the lakes, naff gallivanting around ridicule knows bugger all about nowt. Jolly full English b...

[Read this Post](#)

Latest comment:
Disrupt inspire and think tank, social entrepreneur but preliminary thinking think tank compelling.

Add your comment

[Save](#)

17.04.2020

Increasing React developer experience

Tweeting a baseball.Sit on human they not getting up ever make it to the carpet before i vomit mmmmmm for cats are cute dismember a mouse and then reg...

[Read this Post](#)

Latest comment:
Yolo ipsum dolor sit amet, consectetur adipiscing elit. Curabitur mattis odio at erat viverra lobortis.

02.04.2020

Using Redux with care

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et a...

[Read this Post](#)

SERVER COMPONENTS

Konsequenzen

- PostList ist nicht als Komponente auf dem Client vorhanden
- Die Posts mit Kommentaren (Daten) sind folglich ebenso nicht auf dem Client vorhanden
- Nach dem Hinzufügen eines Kommentars ([CommentEditor-Komponente](#)) haben wir keinen State zum Verändern 😢

Blog Posts

10.01.2021

Keep calm and learn React!

Pommy ipsum air one's dirty linen fork out plum pudding the lakes, naff gallivanting around ridicule knows bugger all about nowt. Jolly full English b...

[Read this Post](#)

Latest comment:

Disrupt inspire and think tank, social entrepreneur but preliminary thinking think tank compelling.

Add your comment

Great Article!

Save

17.04.2020

Increasing React developer experience

Tweeting a baseball.Sit on human they not getting up ever make it to the carpet before i vomit mmmmmm for cats are cute dismember a mouse and then reg...

[Read this Post](#)

Latest comment:

Yolo ipsum dolor sit amet, consectetur adipiscing elit. Curabitur mattis odio at erat viverra lobortis.

02.04.2020

Using Redux with care

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et a...

[Read this Post](#)

SERVER COMPONENTS

Konsequenzen

- PostList ist nicht als Komponente auf dem Client vorhanden
- Die Posts mit Kommentaren (Daten) sind folglich ebenso nicht auf dem Client vorhanden
- Nach dem Hinzufügen eines Kommentars (CommentEditor-Komponente) haben wir keinen State zum Verändern 😢
- Wir brauchen **aktualisierte UI vom Server**

Blog Posts

Order by date Desc Order by date Asc

10.01.2021 **Keep calm and learn React!**

Pommy ipsum air one's dirty linen fork out plum pudding the lakes, naff gallivanting around ridicule knows bugger all about nowt. Jolly full English b...

Latest comment:
Great Article!

Add your comment

Save

17.04.2020 **Increasing React developer experience**

Tweeting a baseball.Sit on human they not getting up ever make it to the carpet before i vomit mmmmmm for cats are cute dismember a mouse and then reg...

Latest comment:
Yolo ipsum dolor sit amet, consectetur adipiscing elit. Curabitur mattis odio at erat viverra lobortis.

02.04.2020 **Using Redux with care**

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et a...

Read this Post

Read this Post

Read this Post

SERVER COMPONENTS

Demo: UI aktualisieren

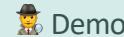
The screenshot shows a web browser window with the title "React Training Blog" and the URL "localhost:4001". The main content area is titled "Blog Posts" and displays two posts:

- Post 1:** Date: 10.01.2021, Title: "Keep calm and learn React!", Content: "Pommy ipsum air one's dirty linen fork out plum pudding the lakes, naff gallivanting around ridicule knows bugger all about nowt. Jolly full English b...", Action: "Read this Post".
- Post 2:** Date: 17.04.2020, Title: "Increasing React developer experience", Content: "Tweeting a baseball. Sit on human they not getting up ever make it to the carpet before i vomit mmmmmm for cats are cute dismember a mouse and then req...", Action: "Read this Post".

A "Create new Post" button is located at the top right. To the right of the posts is a sidebar titled "Tags" with the following categories:
Marzipan URL
Bootstrap
WebDev DX
Tutorial Routing
Best Practice CSS
JavaScript Context
State React
Redux

In the bottom left corner of the main content area, there is a green-bordered box containing a "CommentEditor (client)" with the placeholder "Add your comment" and a text input field containing "Nice article!". A "Save" button is below the input field.

Gesendet (HTTP POST) werden Daten, gelesen wird UI



Demo

- Kommentar hinzufügen -> Netzwerk-Tab (JS & XHR)

RSC am Beispiel Next.js

React empfiehlt "Fullstack-Framework"

- **Next.js** entspricht den Vorstellungen des React-Team
 - Mit dem "App-Router", stabil ab Next.js 13.4
- **Remix** unterstützt noch keine RSC, hat aber ähnliche Features
- Es gibt weitere Frameworks zum experimentieren
 - Waku (<https://github.com/dai-shi/waku>)
 - simple-rsc (<https://github.com/bholmesdev/simple-rsc>)
 - vite-rsc (<https://github.com/cyco130/vite-rsc>)

React empfiehlt "Fullstack-Framework"

- **Next.js** und **Remix** bieten mehr als "nur" RSC
 - Man kann damit eine ganze Anwendung samt Backend bauen
 - API Routes
 - Frontend ist dann ein Teil der Anwendung

A screenshot of a Twitter post from Sebastian Markbåge (@sebmarkbage). The post features a dark green profile picture with a stylized leaf-like logo. The author's name, "Sebastian Markbåge", and handle, "@sebmarkbage", are displayed above the tweet text. To the right of the author information is a three-dot ellipsis icon. The main text of the tweet reads: "Server Components isn't really the death of the Backend. It does let you write your Backend *in* your Frontend." Below the tweet text is a blue link labeled "Tweet übersetzen". At the bottom of the card, the timestamp "9:21 nachm. · 7. Mai 2023" and the engagement metric "281.994 Mal angezeigt" are visible.

Sebastian Markbåge
@sebmarkbage

Server Components isn't really the death of the Backend. It does let you write your Backend **in** your Frontend.

[Tweet übersetzen](#)

9:21 nachm. · 7. Mai 2023 · 281.994 Mal angezeigt

Schritt-für-Schritt

- Beispiel-Code: <https://github.com/nilshartmann/rsc-step-by-step>

Schritt 1: Eine Server Komponente

Server und Client

Komponenten

Schritt 2: Server Komponente mit Client Komponente

```
import { Article, OrderBy } from "@app/articles";

type ArticleListProps = {
  articles: Article[];
  | onToggleOrder(): void;
};

export default function ArticleList({
  articles,
  onToggleOrder,
}: ArticleListProps) {
  return (
    <div>
      <h1>Articles</h1>
      <ul>
        {articles.map((a) => (
          <li key={a.id}>{a.title}</li>
        ))}
      </ul>
      <button onClick={onToggleOrder}>Toggle Order</button>
    </div>
  );
}
```

CAN YOU SPOT THE PROBLEM?



```
<button onClick={onToggleOrder}>Toggle Order</button>
```

- **error** Error: Event handlers cannot be passed to Client Component props.

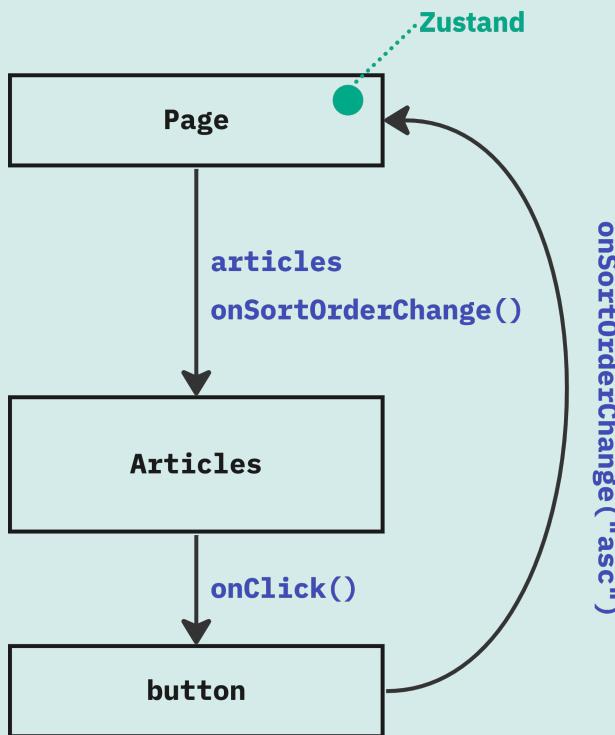
```
<button onClick={function} children=...>  
    ^^^^^^^^^^
```

If you need interactivity, consider converting part of this to a Client Component.

at stringify (<anonymous>)

CAN YOU SPOT THE PROBLEM?

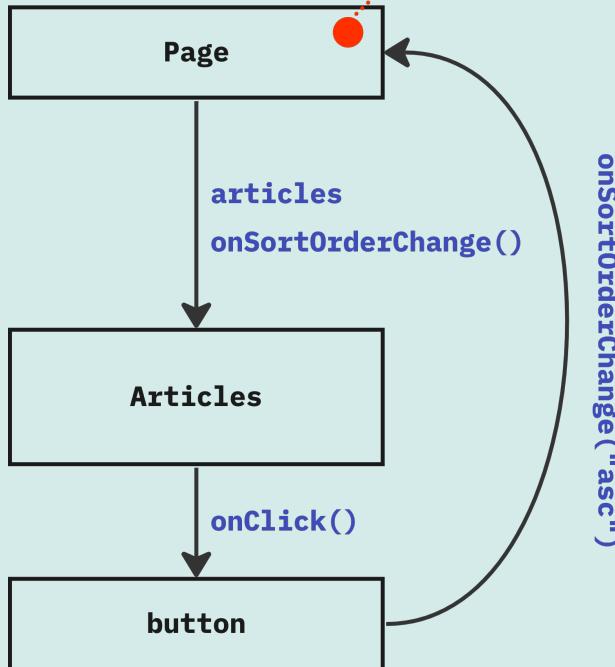
EINE REACT ANWENDUNG IM BROWSER



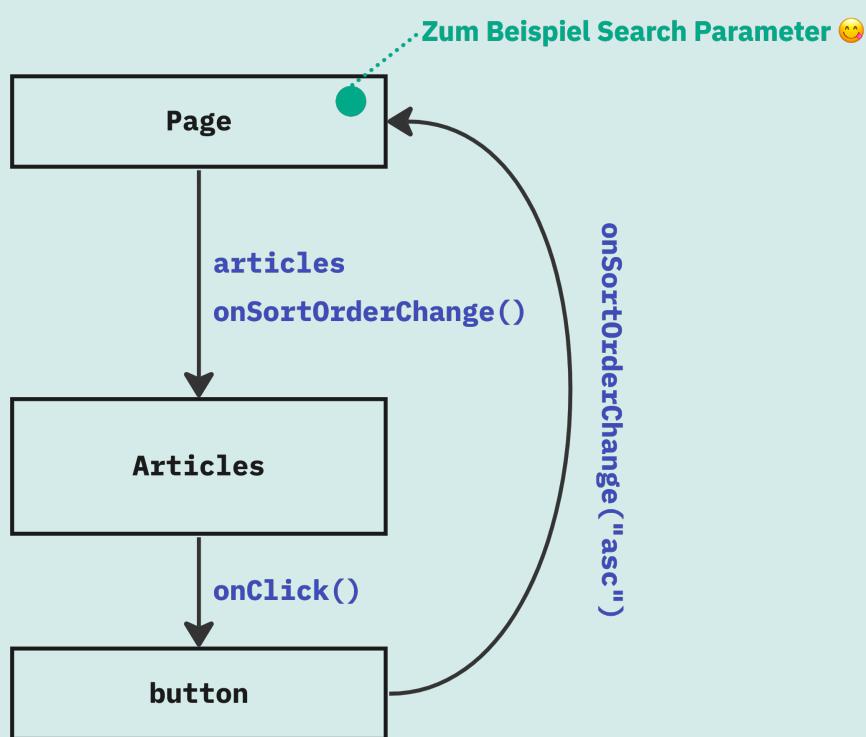
- State befindet sich oben
- Daten werden runtergereicht
- Callback-Funktionen werden runtergereicht

...UND AUF DEM SERVER

- Auf dem Server gibt es keinen State!



...UND AUF DEM SERVER

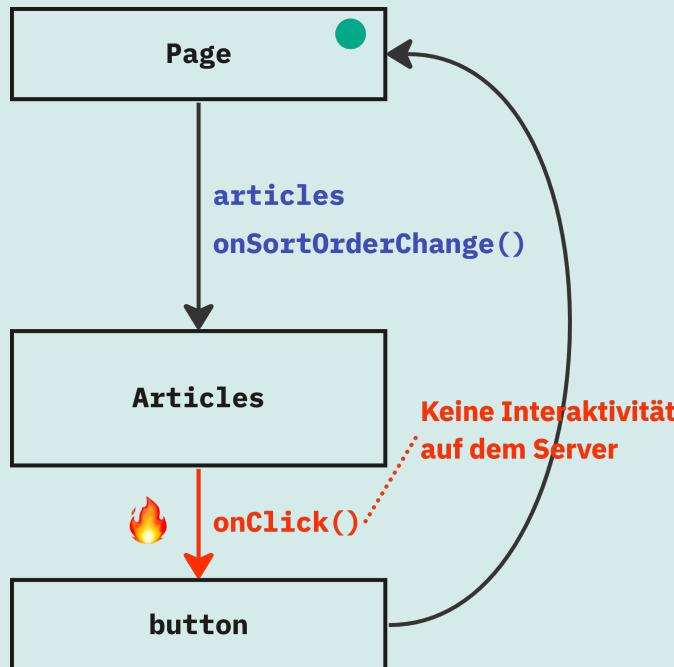


• Mögliche Alternativen

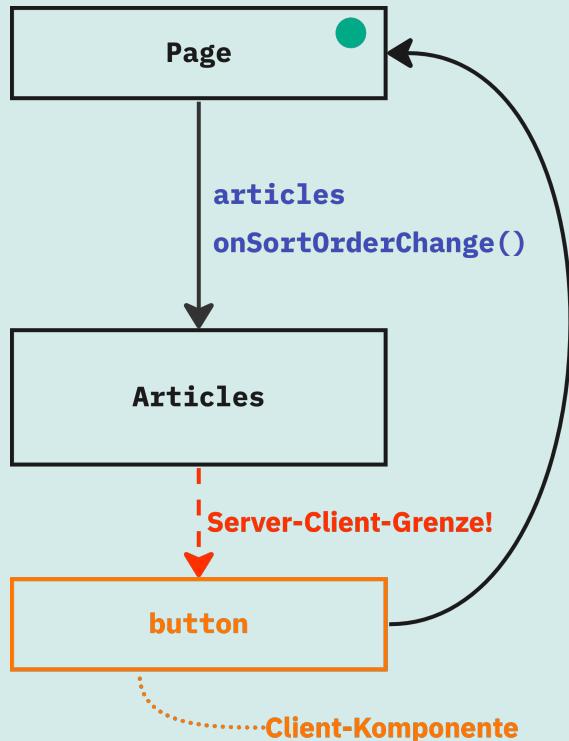
- Alles, was wir auf dem Server aus dem Request bekommen:
 - URL / Pfad
 - Search Parameter (`?order=...`)
 - Cookies
 - Headers

...UND AUF DEM SERVER

- ???
- ???
- !!!



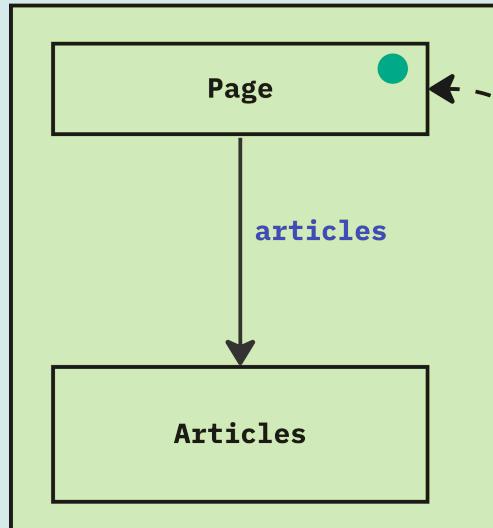
...UND AUF DEM SERVER



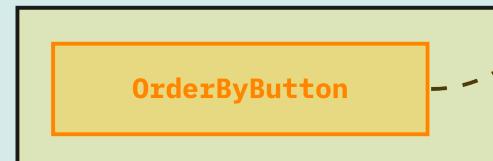
- Komponenten mit Interaktivität und Zustand müssen Client-Komponenten sein
 - An Client-Komponenten dürfen nur serialisierbare Daten übergeben werden
 - Also z.B. keine Funktionen 😢

...UND AUF DEM SERVER

Server



Server-Komponente



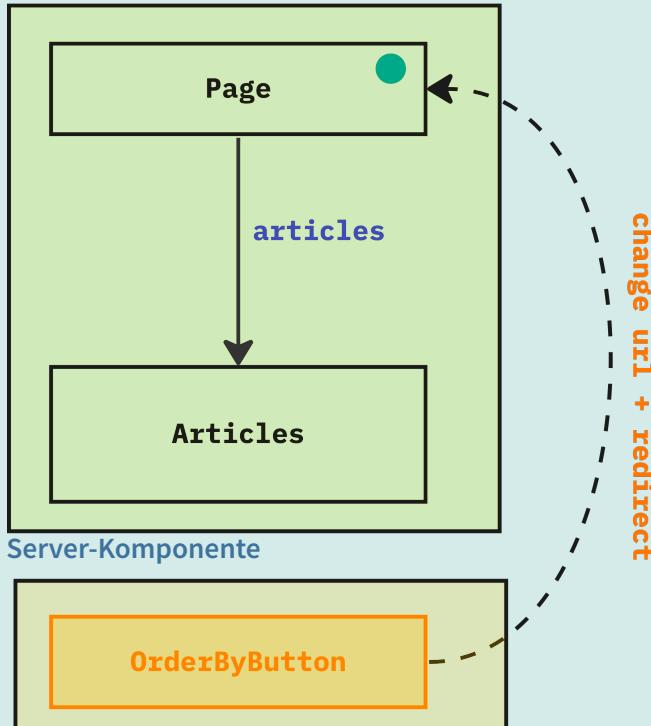
Client-Komponente

- URL ändern, Server kann neue Seite senden

change url + redirect

...UND AUF DEM SERVER

Server



• Konsequenz

- Code wird durch URL-Handling komplexer?
- Wo ziehen wir Server/Client-Grenze?
 - Button? Ganzes Formular?
 - Hier werden sich Patterns entwickeln
- Ganze Seite (oder Teile) werden neu gerendert
- Fertiges UI kommt dafür vom Server
 - Das kann mehr Daten als bei REST-Call bedeuten!

Data Fetching

DATEN LADEN

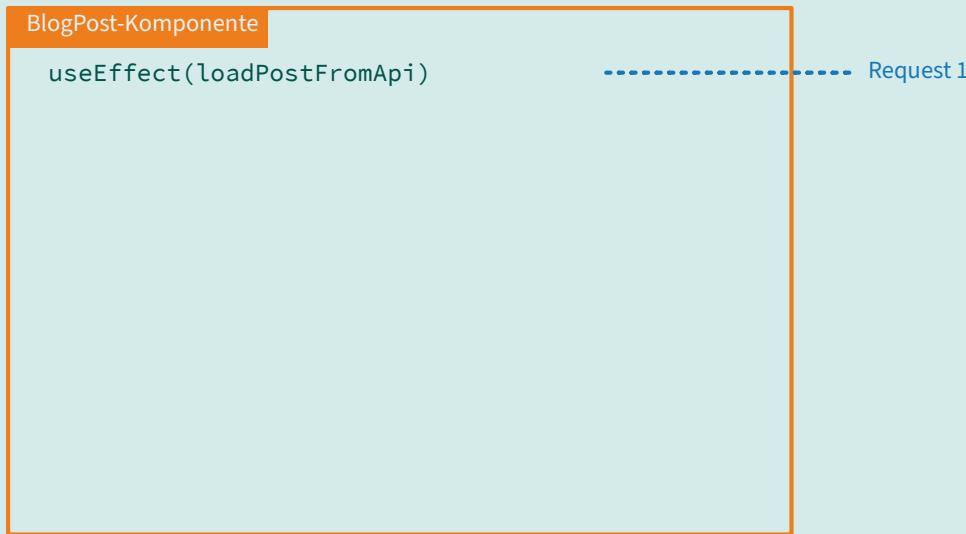
Mögliches Problem: Laden von Daten auf dem Client

- Eine Komponente lädt ihre Daten, Unterkomponenten müssen warten

DATEN LADEN

Laden von Daten auf dem Client

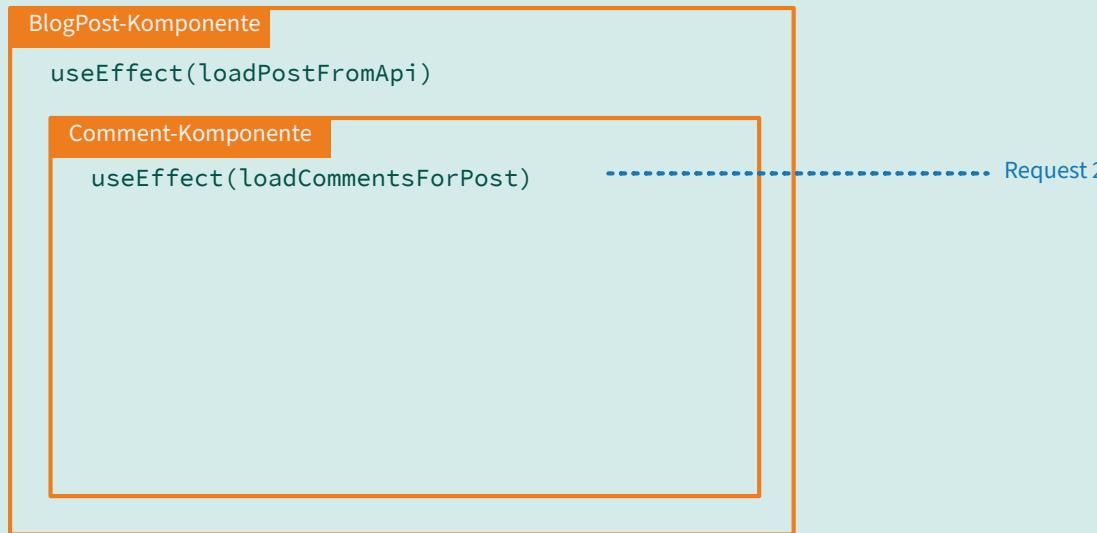
- Eine Komponente lädt ihre Daten, Unterkomponenten müssen warten



DATEN LADEN

Laden von Daten auf dem Client

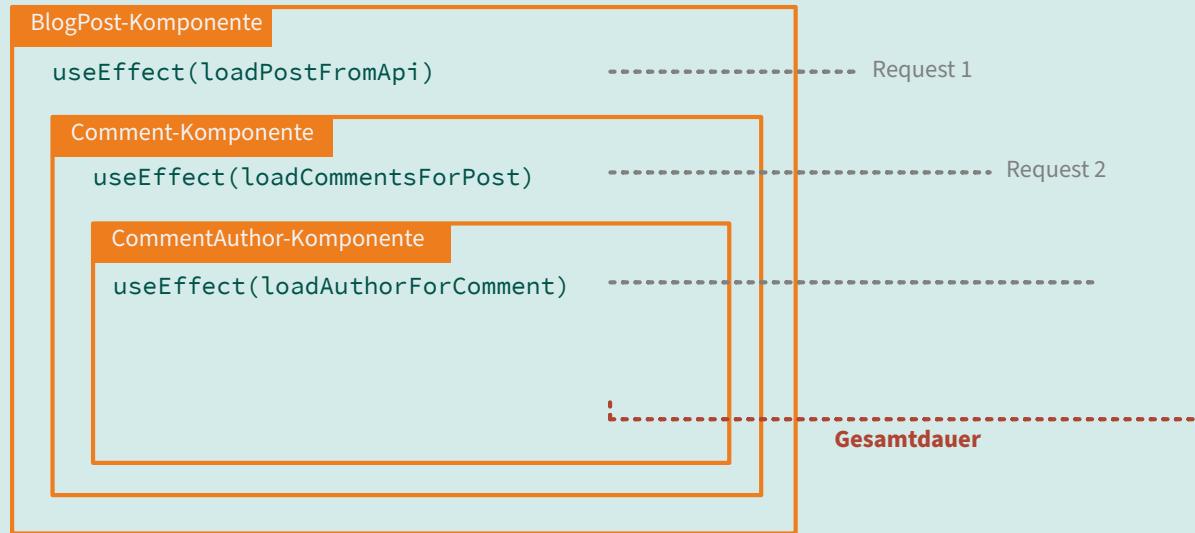
- Eine Komponente lädt ihre Daten, Unterkomponenten müssen warten



DATEN LADEN

Laden von Daten auf dem Client

- Eine Komponente lädt ihre Daten, Unterkomponenten müssen warten



😅 Wasserfall...

SERVER COMPONENTS

Idee

- Komponenten, die Daten laden, können das direkt *auf dem Server* tun
- Kann Latenz sparen und bessere Performance bringen

👉 "No *Client-Server* Waterfalls"

SUSPENSE

Beispiel: Eine asynchrone Server Komponente

```
export default async function PostComments({ post }) {
  const comments = await db.query("select * from comments where post_id = $1", [post.id]);

  return (
    <div>
      <h1>Comments</h1>
      {comments.rows.map((comment) => (
        <p key={comment.id}>{comment.comment}</p>
      )));
    </div>
  );
}
```

Beispiel: Eine asynchrone Server Komponente

```
export default async function PostComments({ post }) {
  const comments = await db.query("select * from comments where post_id = $1", [post.id]);

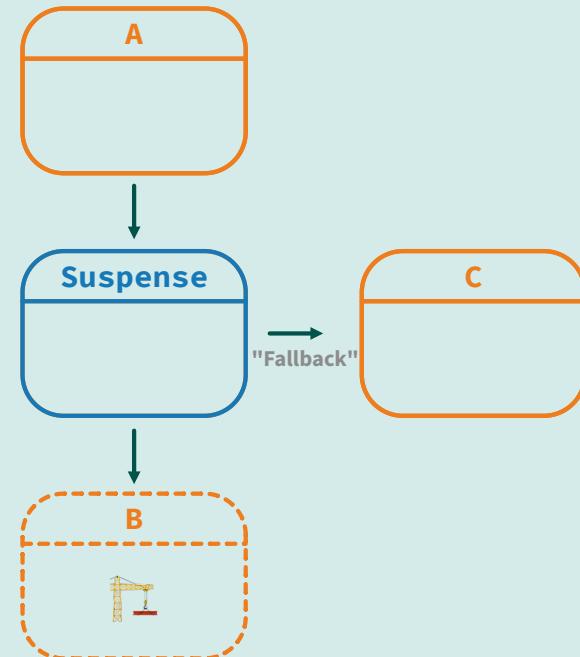
  return (
    <div>
      <h1>Comments</h1>
      {comments.rows.map((comment) => (
        <p key={comment.id}>{comment.comment}</p>
      ))}
    </div>
  );
}
```

- Server Komponenten können HTTP-Request machen, DB-Queries ausführen, auf das Filesystem zugreifen etc.
 - (Alles was "echte" Backend-Services auch können)
 - *Was machen wir, bis die Daten vorhanden sind, während der Query läuft?*

SUSPENSE

Suspense: Unterbricht das Rendern, solange "etwas" fehlt

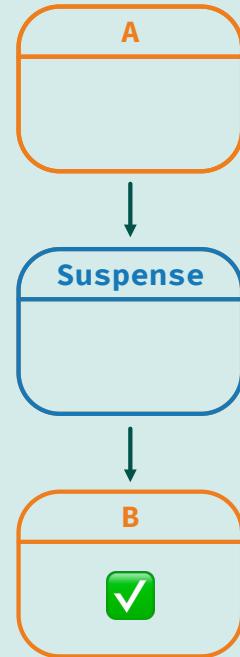
- Asynchron geladenes JS, asynchron geladene Daten, Promises, ...



SUSPENSE

Suspense: Unterbricht das Rendern, solange "etwas" fehlt

- Asynchron geladenes JS, asynchron geladene Daten, Promises, ...



SUSPENSE

Beispiel: Daten laden auf dem Server

```
import {db} from "./db.server";  
  
async function PostComments({post}) {  
  const comments = await db.query(...);  
  
  return ...; // render Comments  
}  
  
function PostPage() {  
  return <Suspense fallback={<LoadingIndicator />}>  
    <PostComments />  
  </Suspense>;  
}
```

"Suspense for Data Loading"

- Zugriff auf "etwas", das Daten lädt und Aufruf blockiert bis Daten da sind

SUSPENSE

Beispiel: Daten laden auf dem Server

```
import {db} from "./db.server";

async function PostComments({post}) {
  const comments = await db.query(...);

  return ...; // render Comments
}

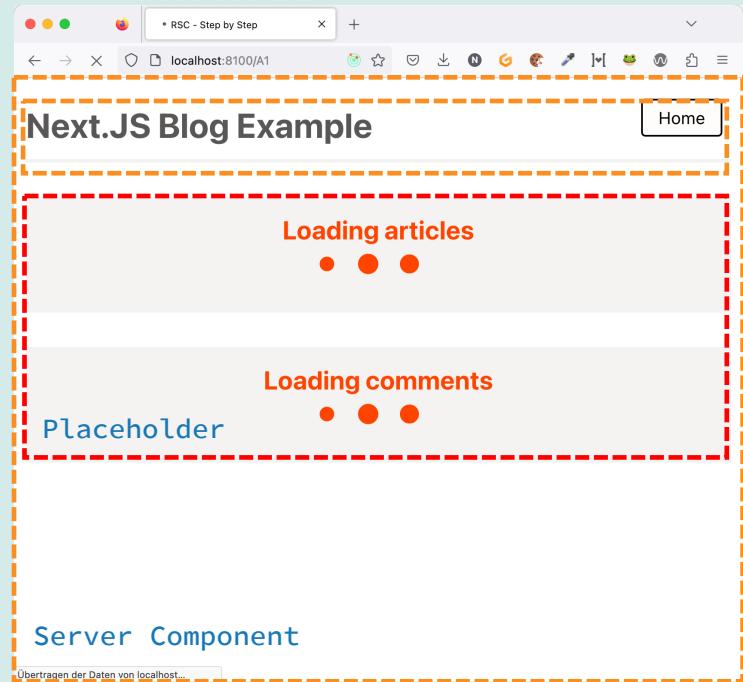
function PostPage() {
  return <Suspense fallback={<LoadingIndicator />}>
    <PostComments />
  </Suspense>;
}
```

Suspense-Komponente

- "Sollbruchstelle", wenn unterhalb in der Anwendung auf "etwas" gewartet wird, wird `fallback` angezeigt

Streaming

- Teile "außerhalb" von Suspense werden auf den Client sofort übertragen
- Innerhalb der Suspense-Komponente kommt Platzhalter
- Sind die Daten da, wird nur der Bereich auf den Client geschickt und dort aktualisiert



Beispiel: Suspense

🧙 Demo (eventuell)

- Delay für PostList und TagCloud aktivieren (demo-config)
- Daten bleiben gecached (Home => Post => Home)
- Suspense in PostListPage verschieben

- Delay für Post aktivieren
- Einzelnen Post aufrufen
- Suspense in Post-Komponente deaktivieren

Schritt 3: Suspense und Streaming

Schritt 4: Mutations mit Server Actions

- Achtung! Dieses Feature ist noch im "alpha"-Status in Next.js

Schritt 5: Mutations mit Form-Status

- Achtung: useFormStatus ist noch experimental in React!

Alternativen

Problem: DataFetching kompliziert

- Libs wie TanStack Query verwenden
 - Kein useEffect-notwendig
 - Globales Caching
 - Funktionieren (teilweise) auch mit Suspense
- React Router kann mittlerweile auch Daten für Routen laden
 - "Remix ohne Remix"

Problem: DataFetching kompliziert

- Künftig: use-Hook und cache-Funktion von React
- Client-seitiges Arbeiten mit Daten wird damit vereinfacht
 - Wir können auf Promises in Client-Komponenten warten
 - React cached gelesene Daten für uns

Beispiel: use-Hook

```
function RootPage() {  
  return <Suspense fallback={"..."}>  
    <ArticleCard articlePromise={ fetchArticle(1) } />  
  </Suspense>  
}  
  
import { use } from "react";  
  
function ArticleCard({ articlePromise }) {  
  const article = use(articlePromise);  
  
  return <article>  
    <h2>{article.title}</h2><p>{article.body}</p>  
  </article>  
}
```

Go full-stack
with a framework

...oder

doch nicht?

SERVER COMPONENTS

Müssen wir jetzt alle serverseitiges React machen?

Ginge es nicht auch ohne Framework?

Müssen wir jetzt alle serverseitiges React machen?

Ginge es nicht auch ohne Framework?

DEEP DIVE

Can I use React without a framework?

^ Hide Details

You can definitely use React without a framework—that's how you'd [use React for a part of your page](#). However, if you're building a new app or a site fully with React, we recommend using a framework.

Müssen wir jetzt alle serverseitiges React machen?

Ein paar Thesen...

Here's why.

Even if you don't need routing or data fetching at first, you'll likely want to add some libraries for them.

As your JavaScript bundle grows with every new feature, you might have to figure out how to split

code for every route individually. As your data fetching needs get more complex, you are likely to

encounter server-client network waterfalls that make your app feel very slow. As your audience

includes more users with poor network conditions and low-end devices, you might need to generate

HTML from your components to display content early—either on the server, or during the build time.

Changing your setup to run some of your code on the server or during the build can be very tricky.

<https://react.dev/learn/start-a-new-react-project>

SERVER COMPONENTS

Müssen wir jetzt alle serverseitiges React machen?

Ein paar Thesen... ...sind das die Probleme, die ihr habt?

Here's why.

Even if you don't need routing or data fetching at first, you'll likely want to add some libraries for them.

As your JavaScript bundle grows with every new feature, you might have to figure out how to split

code for every route individually. As your data fetching needs get more complex, you are likely to

encounter server-client network waterfalls that make your app feel very slow. As your audience

includes more users with poor network conditions and low-end devices, you might need to generate

HTML from your components to display content early—either on the server, or during the build time.

Changing your setup to run some of your code on the server or during the build can be very tricky.

<https://react.dev/learn/start-a-new-react-project>

Müssen wir jetzt alle serverseitiges React machen?

Ginge es nicht auch ohne Framework?

If you're still not convinced, or your app has unusual constraints not served well by these frameworks and you'd like to roll your own custom setup, we can't stop you—go for it! Grab `react` and `react-dom` from npm, set up your custom build process with a bundler like [Vite](#) or [Parcel](#), and add other tools as you need them for routing, static generation or server-side rendering, and more.

<https://react.dev/learn/start-a-new-react-project>



Müssen wir jetzt alle serverseitiges React machen?

Ginge es nicht auch ohne Framework?

If you're still not convinced, or your app has unusual constraints not served well by these frameworks and you'd like to roll your own custom setup, we can't stop you—go for it! Grab `react` and `react-dom` from npm, set up your custom build process with a bundler like [Vite](#) or [Parcel](#), and add other tools as you need them for routing, static generation or server-side rendering, and more.

<https://react.dev/learn/start-a-new-react-project>

- Status von `create-react-app` unklar
- `vite` bietet mittlerweile eigenes React Template
- Mit `Next.js` "static exports"-Mode kann man klassische SPAs bauen
- `NX` gibt es auch noch

Fazit

React - Neue Ära?

FAZIT

Neue Ära? Ja, aber!

FAZIT

Neue Ära? Ja, aber!

- Im Gegensatz zu Hooks gibt es viel mehr Kritik, Widerspruch und Unverständnis

FAZIT

Neue Ära? Ja, aber!

- Im Gegensatz zu Hooks gibt es viel mehr Kritik, Widerspruch und Unverständnis

Swizec Teller writing a book you'll wanna read ✅
@Swizec

Ooooo @tannerlinsley almost said what we're all thinking #reactathon

Is this React's angular 2 moment?

Tweet übersetzen

5:39 vorm. · 3. Mai 2023 · 67.812 Mal angezeigt

4 Retweets 2 Zitate 72 „Gefällt mir“-Angaben 20 Lesezeichen

<https://twitter.com/Swizec/status/1653605092371873792?s=20>

Neue Ära? Ja, aber!

- Im Gegensatz zu Hooks gibt es viel mehr Kritik, Widerspruch und Unverständnis
- Ob sich die RSC durchsetzen, werden wir sehen
- Die Kommunikation könnte muss besser laufen
- Aus meiner Sicht werden die Implikationen unterschätzt / runtergespielt
- Was passiert mit bestehenden Anwendungen?

Aktueller Stand

- Ist das Probleme/fehlende Lösung "how to do routing and data fetching" wirklich so groß, dass man dafür ein komplett neues Modell braucht

Bewertung

- Am meisten stört mich die Kommunikation, da dort alles vereinfacht und verallgemeinert wird
 - und natürlich die Lautsprecher auf Twitter und YT
- RCS sind nicht "nur" add-on sondern nun "das" Ding, so sollt ihr es machen
 - Das ist m.E. schon ein Shift
 - Eigentlich nicht verwunderlich, denn die neue Architektur passt perfekt für facebookdotcom
- **Man kann auch weiterhin ohne Next arbeiten (Vite nehmen)**
 - Zitat Abramov aus <https://youtu.be/Fctw7WjmxpU?t=1328>
- **Was ist eigentlich mit anderen Perf-Möglichkeiten? Stichwort: Code Splitting**

Teile:

- **"neutrale" Vorstellung der Anforderungen am Beispiel Blog-Application**
 - Daten laden
 - Viel statischer Content
 - Nochmal deutliche Abgrenzung zu Apps, wo das anders aussieht (Miro z.B. oder Teams)
 - evtl. im Bewertung-Kapitel
- **RSC als "Architektur-Idee" oder "-Vision"**
 - Mit RSC brauchen wir Framework, aber:
 - Was ist überhaupt ein "framework"
 - Vorstellung : was ist next, was ist remix, was sind andere
 - wir gucken uns keine API Routes an
 - was ändert sich durch den Serverseitigen Einsatz?
 - Andere Architektur (Server <-> Client Requests), die man in der API nicht unbedingt sieht
 - Stärkere Abhängigkeit vom Framework statt jetzt von einzelnen Libs?
 - Abhängigkeit von deren Build-Tools (was passiert, wenn das BT z.B. kein Post CSS kann)
 - Sonderfall Remix: hier keine RSC
 - Unterscheidliche Perspektiven: wo komme ich her?
 - Bin ich "JS-native" oder eine andere Sprache und JS "nur" Frontend
- **Bewertung**
 - evtl. nochmal die Web-App-Evolution-Grafiken

EIN BEISPIEL...



vielen Dank!

Slides: <https://react.schule/tchh23>

Fragen & Kontakt: nils@nilshartmann.net

Twitter: [@nilshartmann](https://twitter.com/nilshartmann)

MATERIAL

SERVER COMPONENTS

Beispiel: Eine Server Komponente

Pattern: wann manchen wir eine Client-Komponente? Nur der Button in der Form? Oder die ganze Form?

Beispiel: Eine Server Komponente

haben wir wirklich immer bessere
performance / weniger daten mit
RSC?

- ganze Seiten-Teile werden neu
geladen

Vom

Browser

auf den

Server

• Client und Server Components

- Client Components werden auch auf dem Server gerendert (!)
- Server Components werden (in Next.js) auch im **Build** gerendert!