

NILS HARTMANN

<https://nilshartmann.net>



React

2023

Beginn einer

.. neuen

Ära?

Slides: <https://react.schule/ejs23>

NILS HARTMANN

nils@nilshartmann.net

Freiberuflicher Entwickler, Architekt, Trainer aus Hamburg

Java, Spring, GraphQL, React, TypeScript

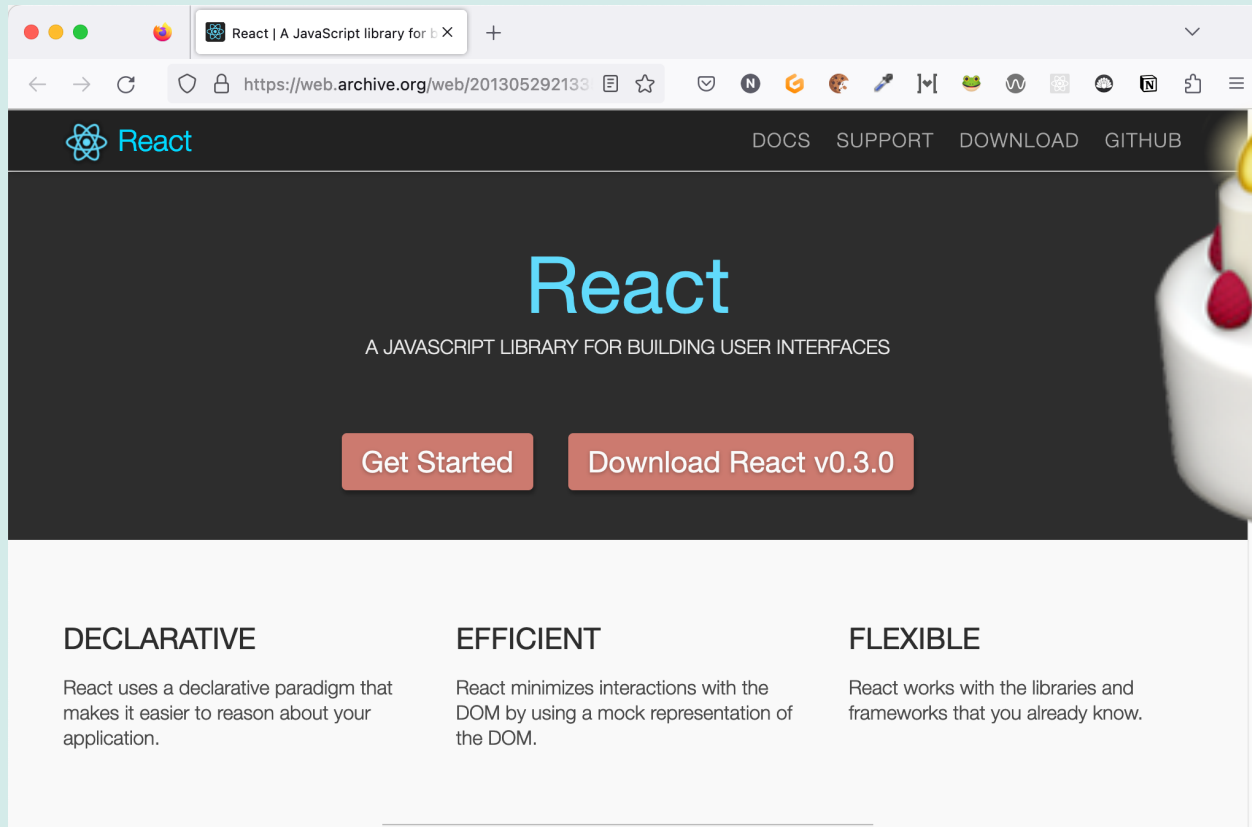


<https://graphql.schule/video-kurs>



<https://reactbuch.de>

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



<https://web.archive.org/web/20130529213355/http://facebook.github.io/react/>

VOR ZEHN JAHREN...

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 ?!



Swizec Teller writing a book you'll wanna read ✓

@Swizec

Oooooo @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>

REAKTIONEN



Ricky ✓

@rickhanlonii



React Server Components are just **PHP++**

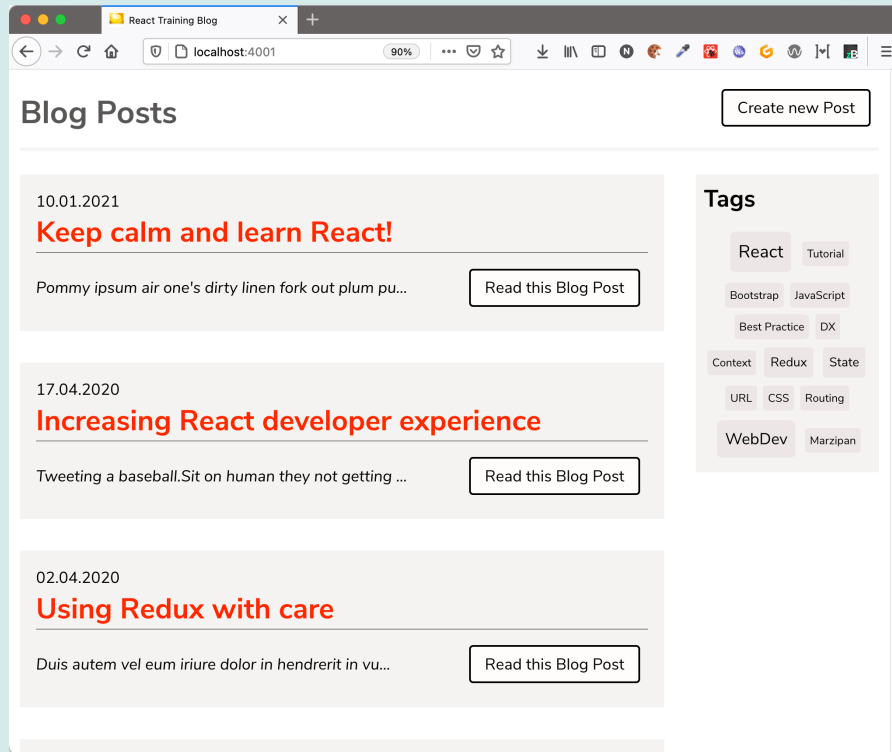
[Tweet übersetzen](#)

6:15 nachm. · 4. Mai 2023 · **134** Mal angezeigt



<https://twitter.com/rickhanlonii/status/1654157814779052034>

REAKTIONEN



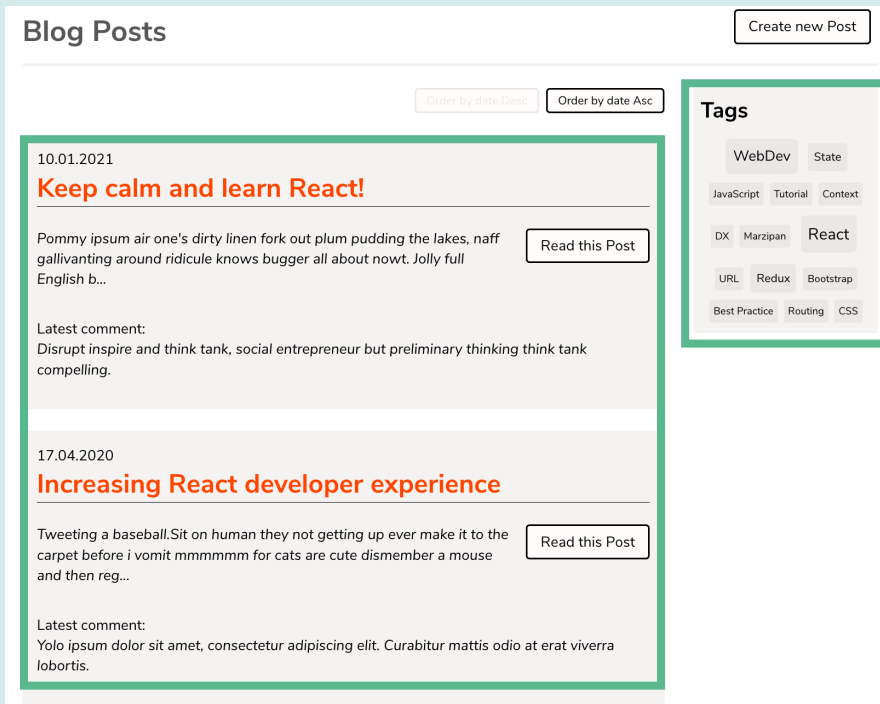
Code: <https://github.com/nilshartmann/fullstack-react-playground>

EIN BEISPIEL...

EIN BEISPIEL

Was macht die Beispiel-Anwendung aus?

- Viel statischer Content



EIN BEISPIEL

Was macht die Beispiel-Anwendung aus?

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

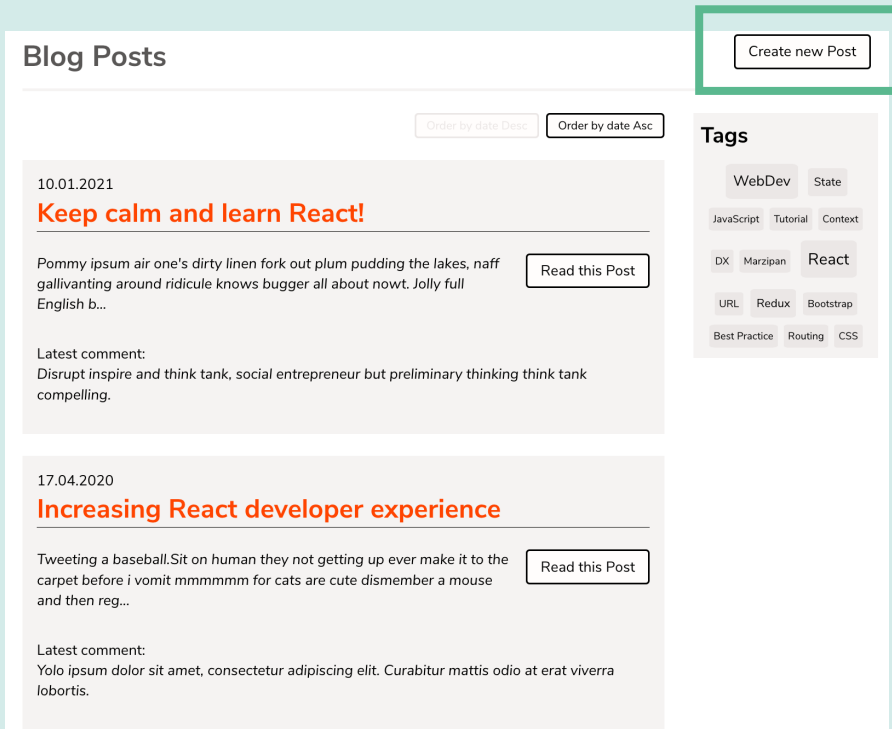
MomentJS!

The screenshot shows a web application titled 'Blog Posts'. It features a header with a 'Create new Post' button and two sorting options: 'Order by date Desc' and 'Order by date Asc'. The main content area displays two blog posts. The first post, dated '10.01.2021', has the title 'Keep calm and learn React!' and a 'Read this Post' button. The second post, dated '17.04.2020', has the title 'Increasing React developer experience' and a 'Read this Post' button. To the right of the posts is a 'Tags' section with a grid of tags including 'WebDev', 'State', 'JavaScript', 'Tutorial', 'Context', 'DX', 'Marzipan', 'React', 'URL', 'Redux', 'Bootstrap', 'Best Practice', 'Routing', and 'CSS'. Annotations with red arrows point to specific parts: 'React!' points to the title of the first post; 'tag-cloud.js' points to the 'React' tag in the tag cloud; 'Marked!' points to the text of the second post. A 'MomentJS!' label is also present at the bottom left of the slide.

EIN BEISPIEL

Was macht die Beispiel-Anwendung aus?

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



Probleme von
klassischen
Single-Page
Anwendungen
(lt. React-Team)

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>

Routing und Data Fetching benötigen Bibliotheken

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>

JavaScript-Code (im Browser) wächst mit jedem Feature

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>

Laden von Daten kann Eindruck langsamer App erzeugen

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>

Frühe Darstellung auch bei schlechtem Netzwerk/Hardware

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>

Ausführung von React-Code auf Server/im Build ist kompliziert

"Fullstack Architektur-Vision"

<https://react.dev/learn/start-a-new-react-project#which-features-make-up-the-react-teams-full-stack-architecture-vision>

"Fullstack Architektur-Vision"

<https://react.dev/learn/start-a-new-react-project#which-features-make-up-the-react-teams-full-stack-architecture-vision>

- **React Server Components (RSC):**
 - Komponenten, die auf dem Server, Client und im Build gerendert werden können
 - Data Fetching "integriert"

"Fullstack Architektur-Vision"

<https://react.dev/learn/start-a-new-react-project#which-features-make-up-the-react-teams-full-stack-architecture-vision>

- **React Server Components (RSC):**
 - Komponenten, die auf dem Server, Client und im Build gerendert werden können
 - Data Fetching "integriert"
- **Suspense:**
 - Loading Indikatoren deklarativ direkt im React-Tree, Server-Client übergreifend

"Fullstack Architektur-Vision"

<https://react.dev/learn/start-a-new-react-project#which-features-make-up-the-react-teams-full-stack-architecture-vision>

- **React Server Components (RSC):**
 - Komponenten, die auf dem Server, Client und im Build gerendert werden können
 - Data Fetching "integriert"
- **Suspense:**
 - Loading Indikatoren deklarativ direkt im React-Tree, Server-Client übergreifend
- **Problem:** Dafür braucht es Infrastruktur (Server, Build, Bundling, ...)

👉 **Deswegen "Fullstack-Framework"**

Zero-Bundle-Size

Server

Components

Introducing Zero-Bundle-Size React Server Components

December 21, 2020 by [Dan Abramov](#), [Lauren Tan](#), [Joseph Savona](#), and [Sebastian Markbåge](#)

2020 has been a long year. As it comes to an end we wanted to share a special Holiday Update on our research into **zero-bundle-size React Server Components**.

<https://legacy.reactjs.org/blog/2020/12/21/data-fetching-with-react-server-components.html>

21. 12. 2020

SERVER COMPONENTS

Idee: Komponenten können auf dem Server und im Build gerendert werden

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

👉 "Zero-Bundle-Size"

Arten von Komponenten

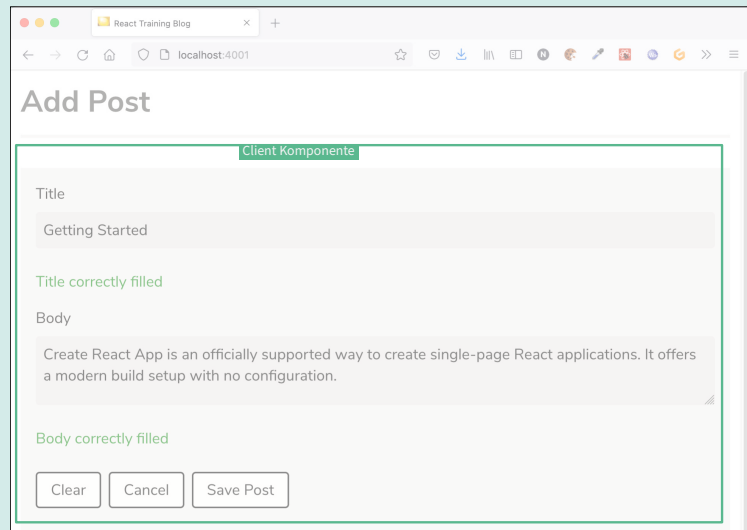
ARTEN VON KOMPONENTEN

Client-Komponenten (wie bisher)

- Werden auf dem Client ausgeführt, können auf dem Server gerendert werden
- JavaScript-Code wird zum Client gesendet
- Müssen (in Next.js) gekennzeichnet werden!

"use client"

```
export function Editor() {  
  const [title, setTitle] = useState("");  
  // ...  
}
```



Neu: Server-Komponenten

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

Neu: Server-Komponenten

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

```
import { marked } from "marked";
import { dateTimeString } from "../components/date-formatter";

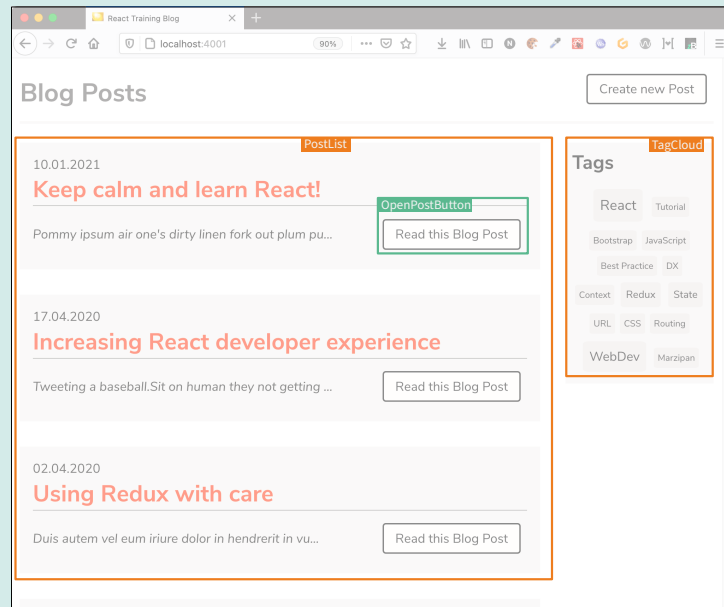
export default function Post(props) {
  const date = dateTimeString(props.post.date);
  const body = marked.parse(props.post.body);

  return (
    <article className="Container">
      <p className="Date">{date}</p>
      <h1>{post.title}</h1>
      <div dangerouslySetInnerHTML={{ __html: body }} />
    </article>
  );
}
```

ARTEN VON KOMPONENTEN

Weiterhin ein Komponenten-Baum

- Server Komponenten sind nicht auf dem Client vorhanden!



ARTEN VON KOMPONENTEN

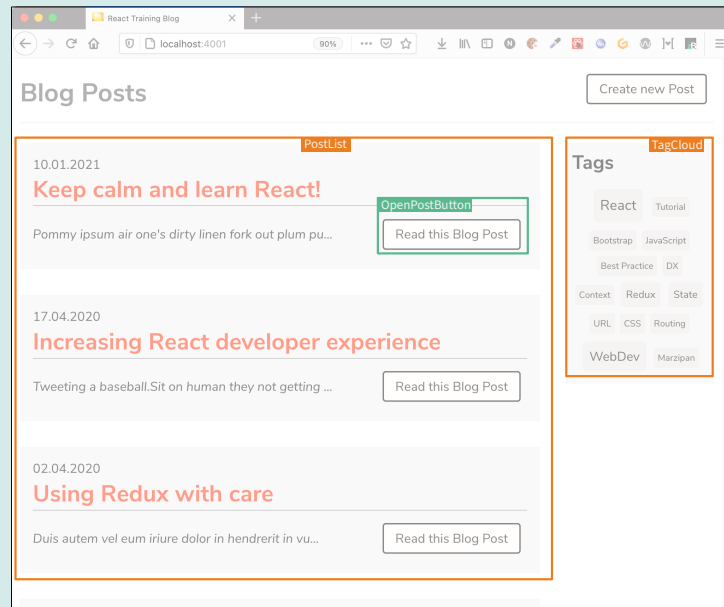
Weiterhin ein Komponenten-Baum

- Server Komponenten sind nicht auf dem Client vorhanden!



Demo

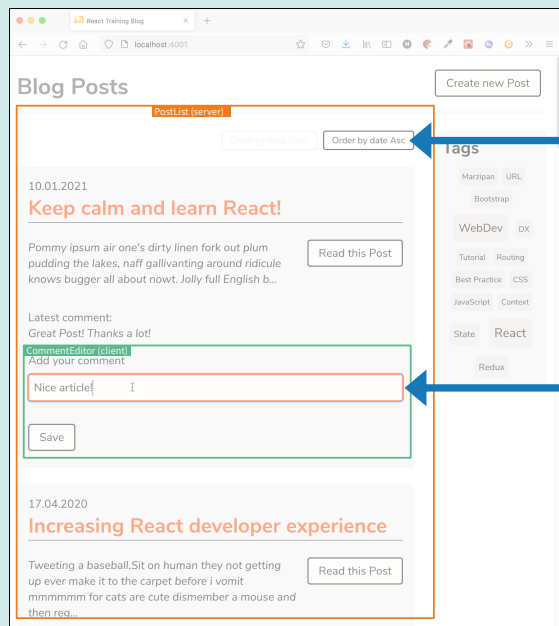
- PostListPage im Code:
- Server-Komponenten "PostList" und "TagCloud" gibt es als Komponenten, aber nicht auf dem Client (-> React Dev Tools)
- Netzwerktab:
 - Client Komponenten wie gewohnt



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



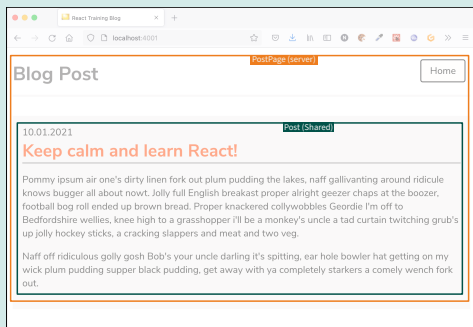
Demo

- **PostPreview:** CommentEditor hinzufügen
- Kommentar eingeben
- Sortierung ändern

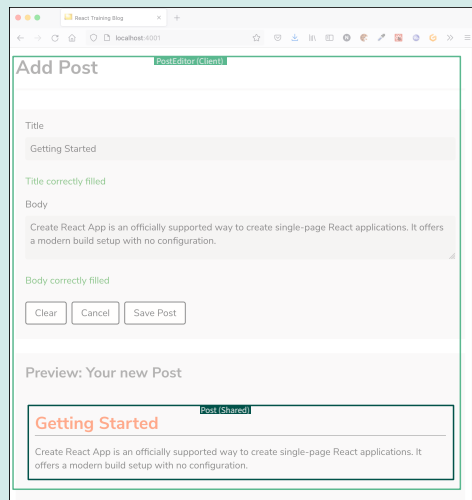
ARTEN VON KOMPONENTEN

"Mixed" Components

- Komponenten können sowohl auf dem Client als auch auf dem Server gerendert werden
- 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)

Data Fetching

Data Fetching in Server Components

- Komponenten, die Daten laden, können das direkt *auf dem Server* tun

SERVER COMPONENTS

Data Fetching in Server Components

- Komponenten, die Daten laden, können das direkt *auf dem Server* tun
- Kann Latenz sparen und bessere Performance bringen
- "No *Client-Server* Waterfalls"

SERVER COMPONENTS

Data Fetching in Server Components

- Komponenten, die Daten laden, können das direkt *auf dem Server* tun
- Kann Latenz sparen und bessere Performance bringen
- "No *Client-Server* Waterfalls"

👉 Server Components können **asynchron** sein 🤖

SERVER COMPONENTS

Beispiel: Eine asynchrone Server Komponente

```
async function fetchPost(postId) { ... }  
async function fetchComments(postId) { ... }
```

----- *Fachliche Funktionen, die Daten laden (z.B. mit "fetch")*

A dashed purple line originates from the closing curly brace of the first function, `fetchPost`, and points diagonally down and to the right towards the explanatory text.

Beispiel: Eine asynchrone Server Komponente

```
async function fetchPost(postId) { ... }
```

```
async function fetchComments(postId) { ... }
```

```
export async function PostPage(props) {  
  const post = await fetchPost(props.postId);  
  const commentsPromise = fetchComments(props.postId);  
  
  return (  
    <>  
      <Post post={post.data} />  
      <Comments comments={commentsPromise} />  
    </>  
  );  
}
```

SUSPENSE

Beispiel: Eine asynchrone Server Komponente

- Server Komponenten können *async functions* sein und auf Promises warten
 - HTTP-Requests, DB-Queries, Zugriff auf das Filesystem
 - (Alles was "echte" Backend-Services auch können)

```
export async function Comments(props) {  
  const comments = await fetchPost(props.commentsPromise);  
  
  return comments.map( comment => ( ... ) );  
}
```

SUSPENSE

Beispiel: Eine asynchrone Server Komponente

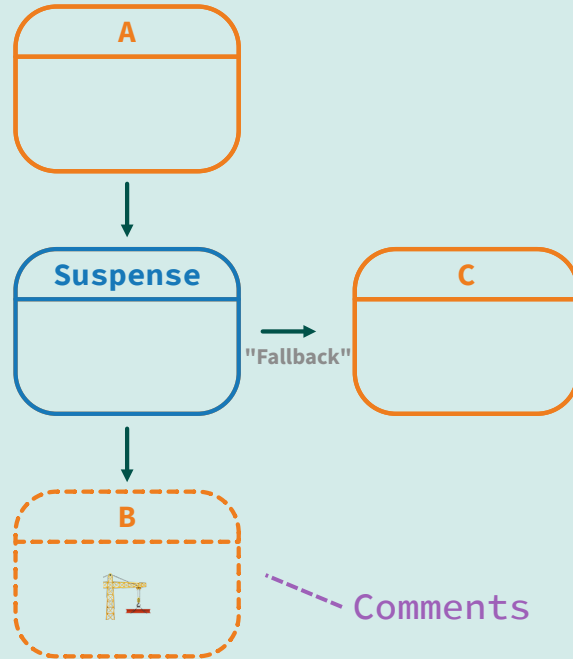
- Server Komponenten können *async functions* sein und auf Promises warten
 - HTTP-Requests, DB-Queries, Zugriff auf das Filesystem
 - (Alles was "echte" Backend-Services auch können)

```
export async function Comments(props) {  
  const comments = await fetchPost(props.commentsPromise);  
  
  return comments.map( comment => ( ... ) );  
}
```

Was machen wir, bis das Promise aufgelöst wird?

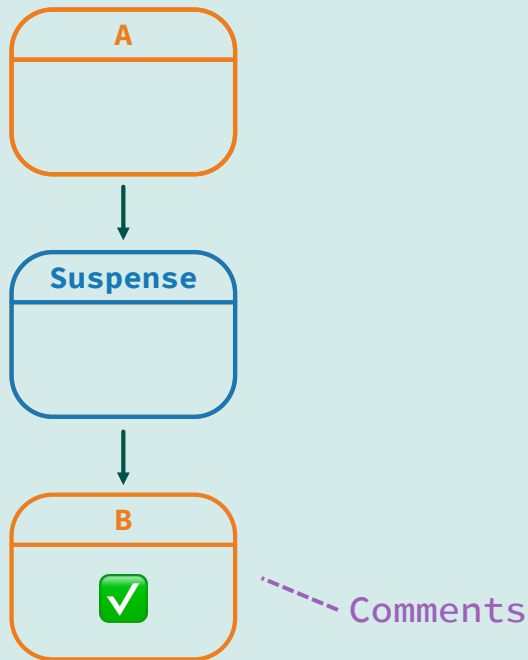
SUSPENSE

Suspense: Unterbricht das Rendern, solange "etwas" fehlt



SUSPENSE

Suspense: Unterbricht das Rendern, solange "etwas" fehlt



SUSPENSE

Beispiel: Data Fetching mit Suspense

```
export async function Comments(props) {  
  const comments = await fetchPost(props.commentsPromise);  
  
  return comments.map( comment => ( ... ) );  
}
```

SUSPENSE

Beispiel: Data Fetching mit Suspense

```
export async function Comments(props) {  
  const comments = await fetchPost(props.commentsPromise);  
  
  return comments.map( comment => ( ... ) );  
}  
  
export function PostPage(props) {  
  const commentsPromise = fetchComments(props.params.postId);  
  
  return (  
    <>  
      <Post post={post.data} />  
      <Suspense fallback={<LoadingIndicator />}>  
        <Comments />  
      </Suspense>  
    </>  
  );  
}
```

SUSPENSE

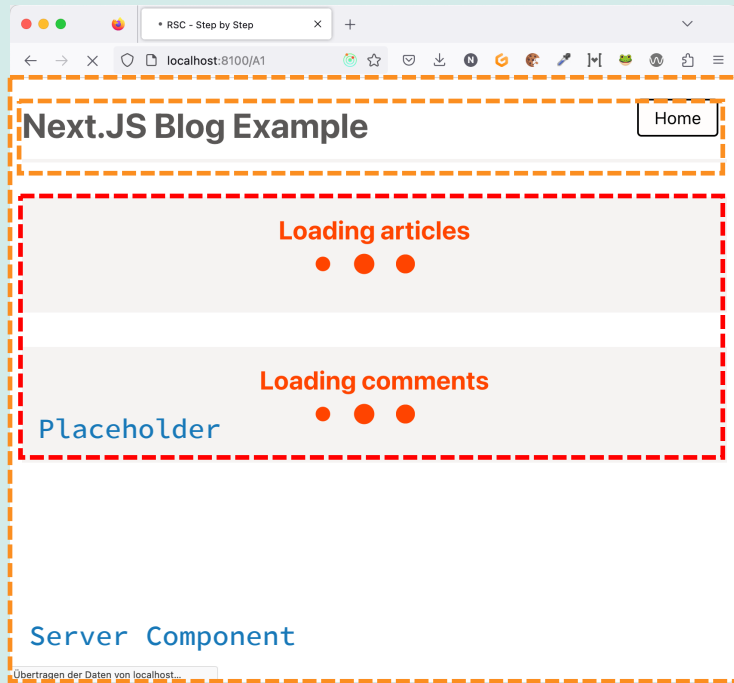
Streaming

- 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



Demo

- `post/postId/page.tsx`



Aufteilung

in

Server-Client:

Konsequenzen

```
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?

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

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

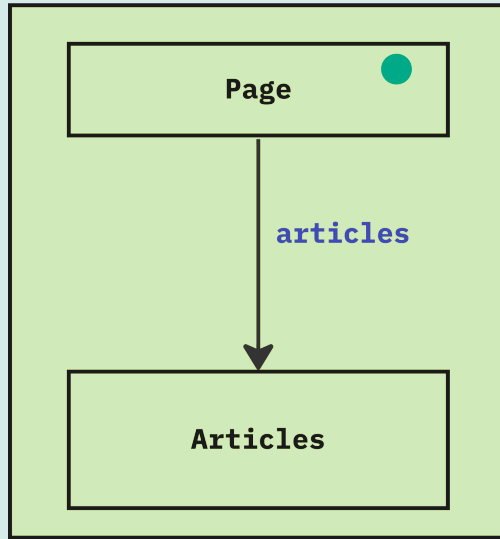
export default function ArticleList({
  articles,
  onToggleOrder,
}: ArticleListProps) {
  return (
    <div>
      <h1>Articles</h1>
      <ul>
        {articles.map((a) => (
          <li key={a.id}>{a.title}</li>
        ))}
      </ul>
      <button onClick={onToggleOrder}>Toggle Order</button>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

Keine Interaktion auf dem Server!

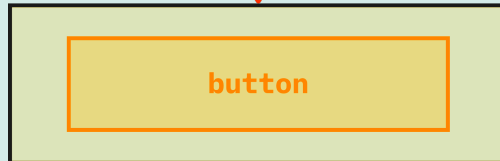
CAN YOU SPOT THE PROBLEM?

...UND AUF DEM SERVER

Server



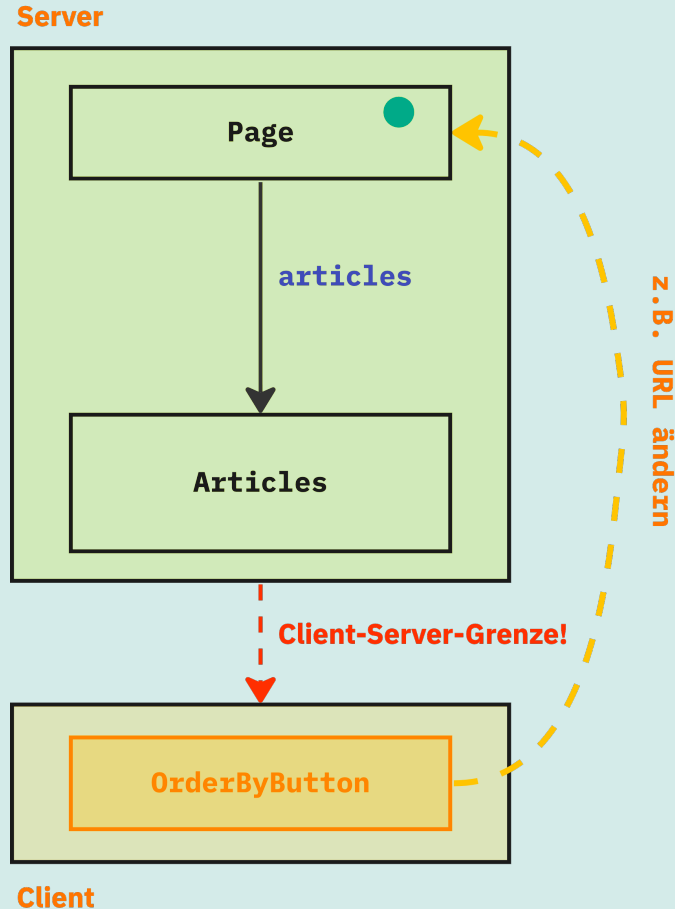
Client-Server-Grenze!



Client

- Properties müssen Client-Server-Grenze überwinden
- Müssen serialisierbare Daten sein
- **Keine (Callback-)Funktionen!**

...UND AUF DEM SERVER



- Properties müssen Client-Server-Grenze überwinden
- Müssen serialisierbare Daten sein
- Keine (Callback-)Funktionen!
- Zur Kommunikation müssen **Server-Requests** durchgeführt werden
 - z.B. URL ändern (Framework-spezifisch)

SERVER COMPONENTS

Mutations

- **PostList** ist nicht als Komponente 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

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 mmmmm 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

Mutations

- 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

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 mmmmm 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

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

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 mmmmm 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...

Read this Post

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 mmmmm 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

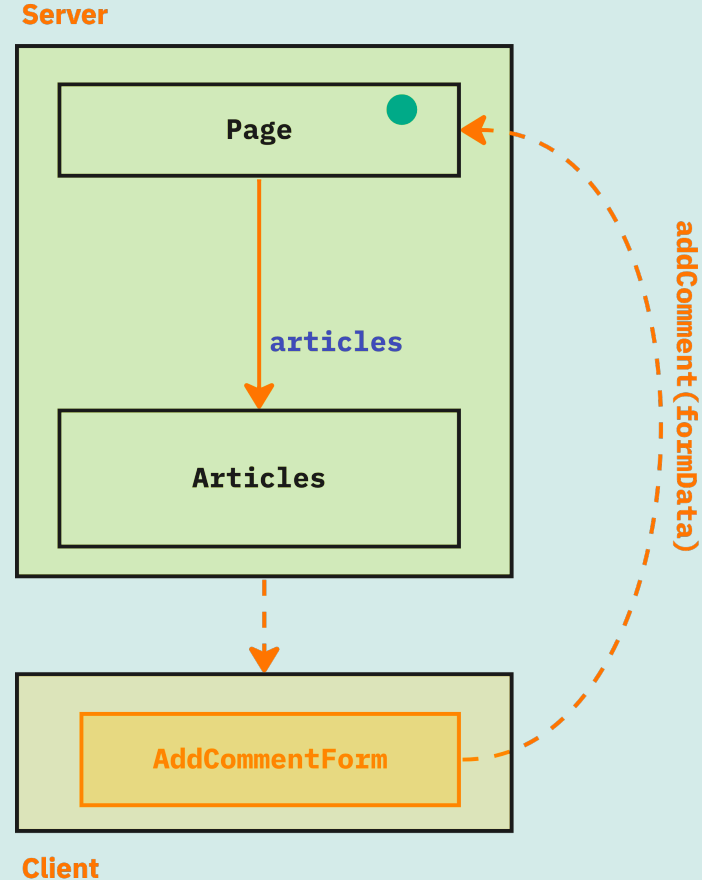
- PostList ist nicht als Komponente auf dem Client vorhanden
- Die Posts mit Kommentaren (Daten) sind folglich ebenso nicht auf dem Client vorhanden
- Nach dem Hinzufügen eines Kommentars (CommentEditor-Komponente) haben wir keinen State zum Verändern 🤔
- Wir brauchen **aktualisierte UI vom Server**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



MUTATIONS

Server Actions

- Das ist eine Art Remote Funktion, die UI (statt Daten) zurückliefert
- In React und Next.js noch Beta!



Server Actions

```
export function CommentEditor() {
```

```
  return <form>  
    <input type="text" name="comment" />  
    <button type="submit">Add</button>  
  </form>;
```

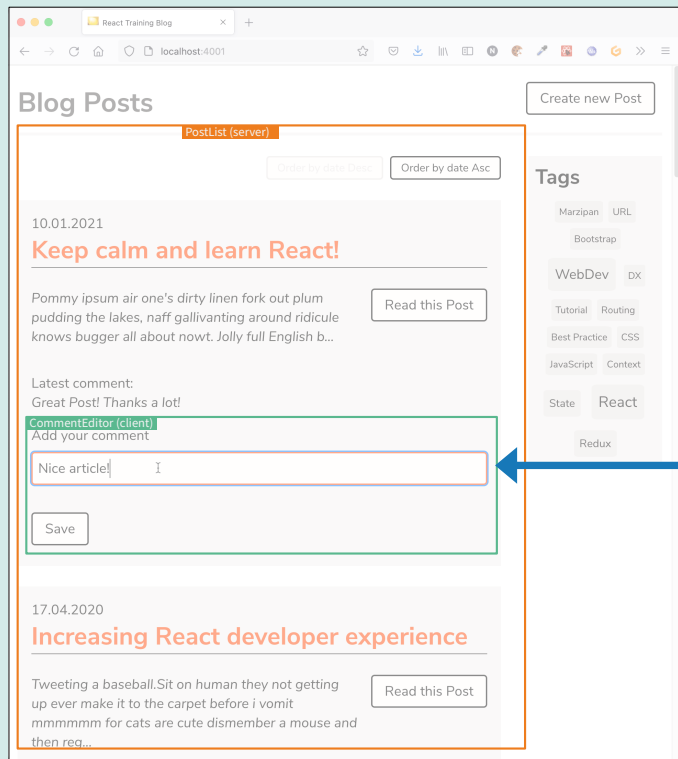
```
}
```


Server Actions

```
export function CommentEditor() {  
  
  async function addComment() {  
    "use server";  
    const comment = form.get("comment");  
  
    const response = await fetch(...);  
  
    revalidatePath("/"); // Next.js  
  
  }  
  
  return <form action={addComment}>  
    <input type="text" name="comment" />  
    <button type="submit">Add</button>  
  </form>;  
}
```

SERVER COMPONENTS

Demo: UI aktualisieren



Gesendet (HTTP POST) werden Daten, gelesen wird UI



Demo

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

Server Actions: Antwort

```
// Client-Komponente!
```

```
function CommentEditorForm({ onSave }) {  
  const [comment, setComment] = useState("");  
  const [result, setResult] = useState(null);
```

Server-Action (als Property)

```
  const submitForm = async (formData) => {  
    const result = await onSave(formData);  
    if (result?.error) {  
      setResult(result.error);  
    } else {  
      // ...  
    }  
  };  
  
  return <form action={submitForm}>...</form>  
}
```

Server Actions: Antwort

// Client-Komponente!

```
function CommentEditorForm({ onSave }) {  
  const [comment, setComment] = useState("");  
  const [result, setResult] = useState(null);  
  
  const submitForm = async (formData) => {  
    const result = await onSave(formData);  
    if (result?.error) {  
      setResult(result.error);  
    } else {  
      // ...  
    }  
  };  
  
  return <form action={submitForm}>...</form>  
}
```

Server-Action (als Property)

 Beispiel

- **CommentEditor.tsx**
- **CommentEditorForm.tsx**

useTransition und useFormStatus

- Mit **useTransition** kann der Aufruf der Server-Action gewrappt werden
- **useFormStatus** (experimental!) liefert weitere Informationen über den Request (läuft, Antwort,)

 Beispiel

- `PostEditor.tsx`, Delay einschalten!

Go full-stack
with a framework

...aber

welches? 🤔

Fullstack-Frameworks

- **Remix & Next.js**
 - für "vollständige" React-Anwendungen
 - Remix (noch) kein Support für RSC
- **Gatsby etc.**
 - eher für statische Websites

Next.js und Remix

- Bringen kompletten Stack mit
 - Server
 - Build-System
- Serverseitiger Router über Filesystem / Konventionen
- Unterstützung für SSR und Streaming

⚡ Kein Client-seitiges Routing

⚡ Kein Migrationspfad (von "klassischem" React)

Next.js

- Entspricht der "Fullstack Architektur-Vision" des React Teams
- Unterstützt React Server Components
- Statische Komponenten können im Build (!) gerendert werden

Remix

- Noch keine RSC
- Daten laden und schreiben nur in Routen
 - etwas weniger flexibel als RSC
- "Nur" SSR
- Verbreitung nicht so hoch wie Next.js

Go full-stack
with a framework

...oder

doch nicht?! 🤔

SERVER COMPONENTS

Müssen wir jetzt alle serverseitiges React machen?

Ginge es nicht auch ohne Framework?

SERVER COMPONENTS

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.

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

Client-seitiges React (SPA) wird bleiben

- Tooling-Frage offen
 - 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

"Moderne" React-Architekturen lösen Probleme

- Data Fetching Libs vereinfachen Data Fetching
- Suspense wird auf dem Client funktionieren
- **use-Hook** in Client-Komponenten statt async-Komponenten (Server)

Fazit

React – Neue Ära?

Neue Ära? Ja, aber!

Neue Ära? Ja, aber!

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



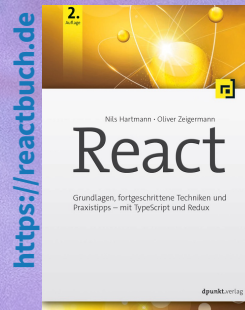
<https://twitter.com/Swizec/status/1653605092371873792?s=20>

Neue Ära? Ja, aber!

- Im Gegensatz zu Hooks gibt es viel mehr Kritik, Widerspruch und Unverständnis
- Ob sich die "Fullstack Architektur-Vision" durchsetzt, werden wir sehen
- Die Kommunikation ~~könnte~~ muss besser laufen

NILS HARTMANN

<https://nilshartmann.net>



Vielen Dank!

Slides: <https://react.schule/ejs23>

Code: <https://github.com/nilshartmann/fullstack-react-playground>

Fragen & Kontakt: nils@nilshartmann.net

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