

NILS HARTMANN

<https://nilshartmann.net>

Serverside Rendering 2.0?

React Server Components

Slides: <https://react.schule/codetalks-react>

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)

Single-Page-Applikationen

- "Echte" Anwendungen mit viel Interaktion

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 the following categories:

- 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](#)

Tags

WebDev State

JavaScript Tutorial Context

DX Marzipan React

URL Redux Bootstrap

Best Practice Routing CSS

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.

EIN BEISPIEL

Was macht die Beispiel-Anwendung aus?

- Viel statischer Content
- Rendern des statischen Contents benötigt 3rd-Party Libs
 - viel JavaScript-Code (Bandbreite!)

MomentJS!

The screenshot shows a blog application interface. At the top, there's a header with "Blog Posts" and a "Create new Post" button. Below the header, there are two blog post cards.

- Post 1:** Date: 10.01.2021, Title: **Keep calm and learn React!**, Preview: 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**, Preview: 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..., Action: **Read this Post**.

To the right of the posts is a sidebar titled "Tags" containing a tag cloud. Red arrows point from the text labels on the left to specific elements in the sidebar:

- A red arrow points from "React!" to the "React" tag in the tag cloud.
- A red arrow points from "tag-cloud.js" to the entire tag cloud area.
- A red arrow points from "Marked!" to the "Curabitur mattis odio at erat viverra lobortis." preview text.

Was macht die Beispiel-Anwendung aus?

- Viel statischer Content
- Rendern des statischen Contents benötigt 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
 - Gut: Client braucht HTML nur anzuzeigen (schnell!)
 - Gut: Suchmaschinen können HTML indizieren

Abgrenzung: 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 **Anwendungscode** zum Client geschickt
 - Wenn vom Browser geladen, ist die Anwendung interaktiv
 - Danach in der Regel keine Server Round-trips mehr

Serverseitiges Rendern - Zusammenfassung

-  Schnelle erste Darstellung
-  Kein Gewinn, bis Anwendung im Client auch *interaktiv* ist
-  Kompletter Anwendungscode muss auf den Client (Bandbreite! Performance!)
-  Anwendungscode muss auf Client *und* Server funktionieren

Zero-Bundle-Size

Server

Components

"-experimental-

```
"dependencies": {  
  "react": "0.0.0-experimental-7ec4c5597",  
  "react-dom": "0.0.0-experimental-7ec4c5597",  
  "react-fetch": "0.0.0-experimental-7ec4c5597",  
  "react-fs": "0.0.0-experimental-7ec4c5597",  
  "react-pg": "0.0.0-experimental-7ec4c5597",  
  "react-server-dom-webpack": "0.0.0-experimental-7ec4c5597",
```



CURRENT STATE

SERVER COMPONENTS

Idee

- Server Components 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

SERVER COMPONENTS

Drei Arten von Komponenten

- Wie bisher: Client-Komponenten

- werden *nur* auf dem Client ausgeführt
- können keine Server-Komponenten verwenden

The screenshot shows a web browser window titled "React Training Blog" at "localhost:4001". The main content is an "Add Post" form. The "Title" input field contains the text "Getting Started" and is highlighted with a green border, indicating it is correctly filled. Below it, a message says "Title correctly filled". The "Body" input field contains placeholder text: "Create React App is an officially supported way to create single-page React applications. It offers a modern build setup with no configuration." This field is also highlighted with a green border, indicating it is correctly filled. Below it, a message says "Body correctly filled". At the bottom of the form are three buttons: "Clear", "Cancel", and "Save Post".

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, ...)

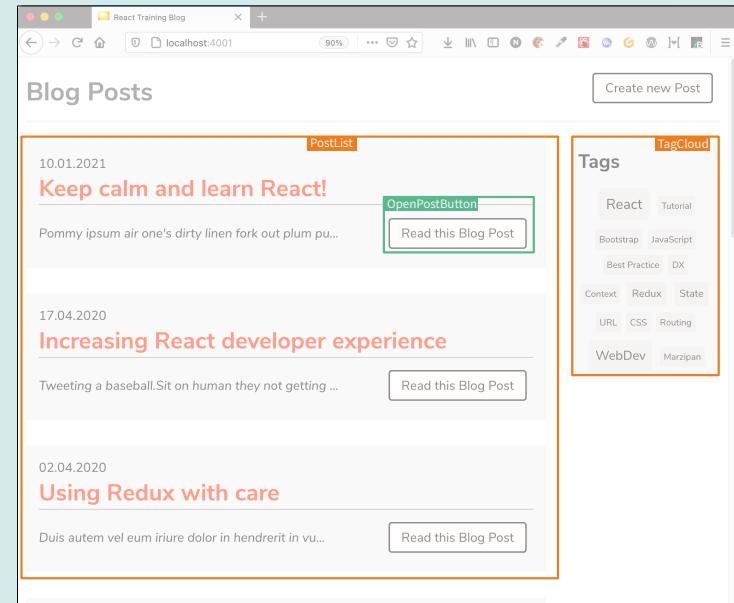
Drei Arten von Komponenten

- 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: kein useState, useEffect, Browser APIs
- aber: können Server Umgebung und Ressourcen nutzen (!)
 - Datenbanken
 - Filesystem

Weiterhin ein Komponenten-Baum

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



SERVER COMPONENTS

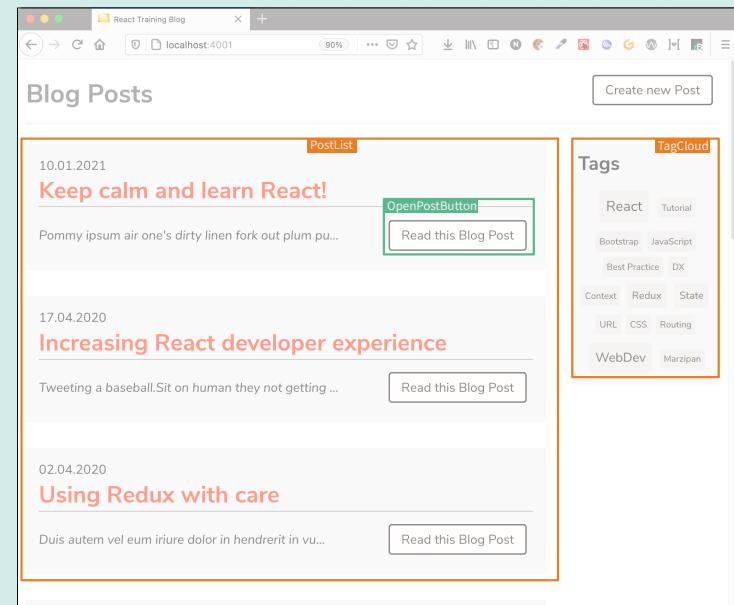
Weiterhin ein Komponenten-Baum

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



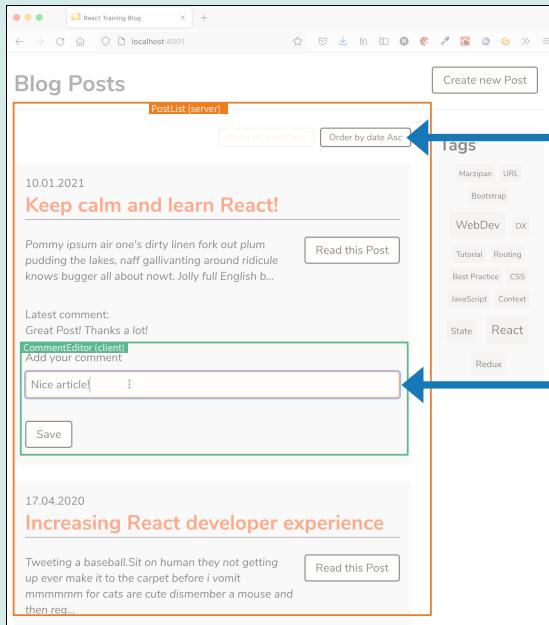
Demo

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



SERVER COMPONENTS

Weiterhin ein Komponenten-Baum



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

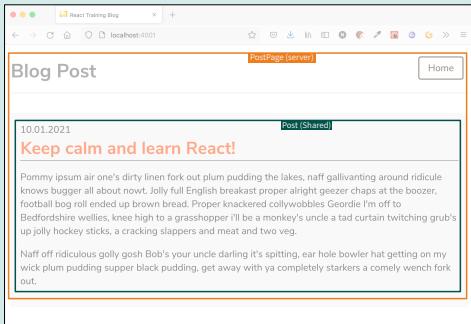
- Neu: Shared Komponenten

- werden auf dem Server und dem Client ausgeführt
 - es gelten also die Restriktionen von Server und Client-Komponenten
 - können von Server- und Client-Komponenten verwendet werden
-
- der entsprechende JavaScript-Code wird erst auf den Client übertragen, wenn er wirklich benötigt wird

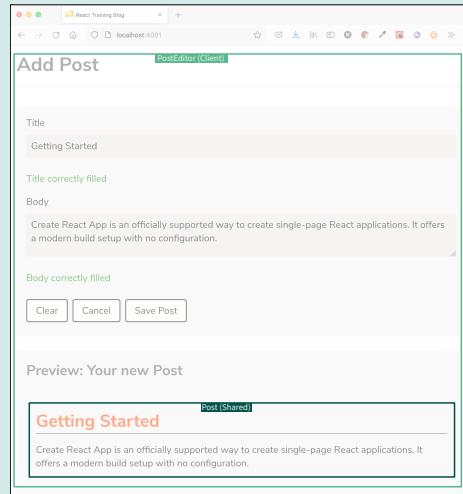
SERVER COMPONENTS

Shared 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)

Konsequenzen

- PostList ist nicht als Komponente auf dem Client vorhanden
- Die Posts 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

Konsequenzen

- PostList ist nicht als Komponente auf dem Client vorhanden
 - Die Posts 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**
-
- Es gibt einen globalen Cache (unstable API!)
 - Der Cache hält die UI-Fragmente und kann mit neuen UI Fragmenten (zzt. manuell) aktualisiert werden

SERVER COMPONENTS

Demo: UI aktualisieren

The screenshot shows a web browser window with the title "React Training Blog" at "localhost:4001". The main content area is titled "Blog Posts". It displays two blog 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...", Buttons: "Read this Post", "CommentEditor (client)" (with placeholder "Add your comment"), and a text input field containing "Nice article!" with a "Save" button.
- 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...", Buttons: "Read this Post".

To the right of the posts is a sidebar titled "Tags" with a list of tags: Marzipan, URL, Bootstrap, WebDev (selected), DX, Tutorial, Routing, Best Practice, CSS, JavaScript, Context, State, React, and Redux.

Gesendet (HTTP POST) werden Daten, gelesen wird UI



Demo

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

Data Fetching

DATEN LADEN

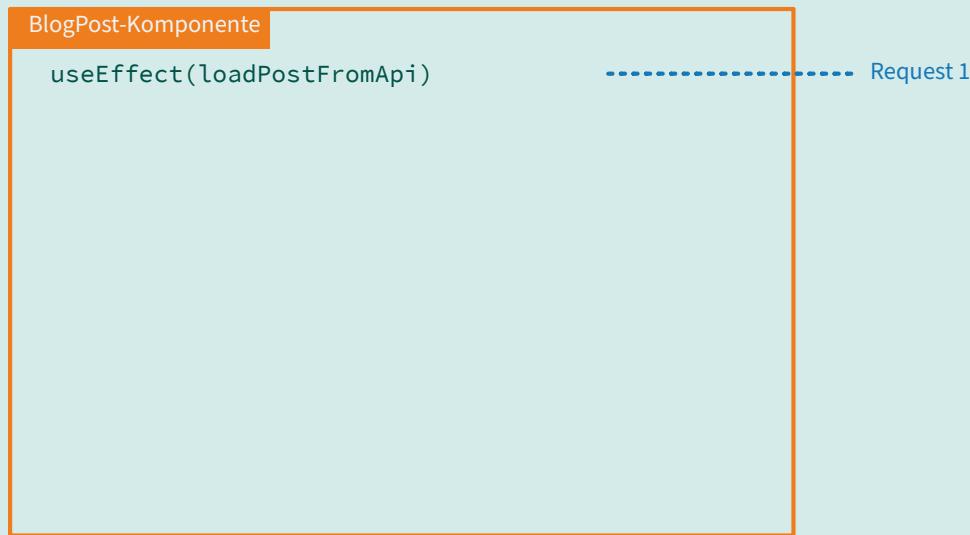
Mögliches Problem: Laden von Daten auf dem Client

- Eine Komponente lädt ihre Daten, Unterkomponenten müssen warten

DATEN LADEN

Laden von Daten auf dem Client

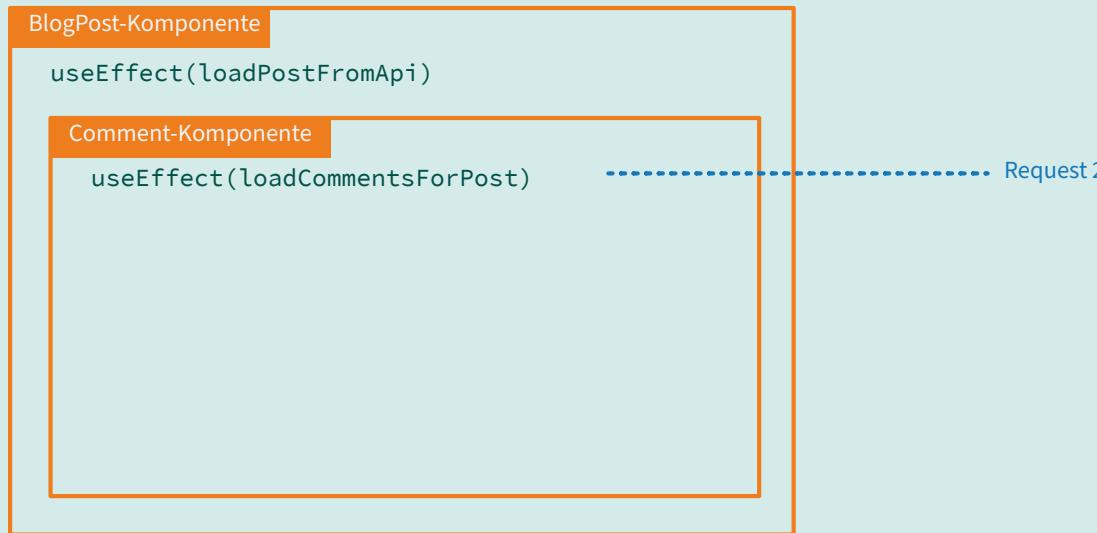
- Eine Komponente lädt ihre Daten, Unterkomponenten müssen warten



DATEN LADEN

Laden von Daten auf dem Client

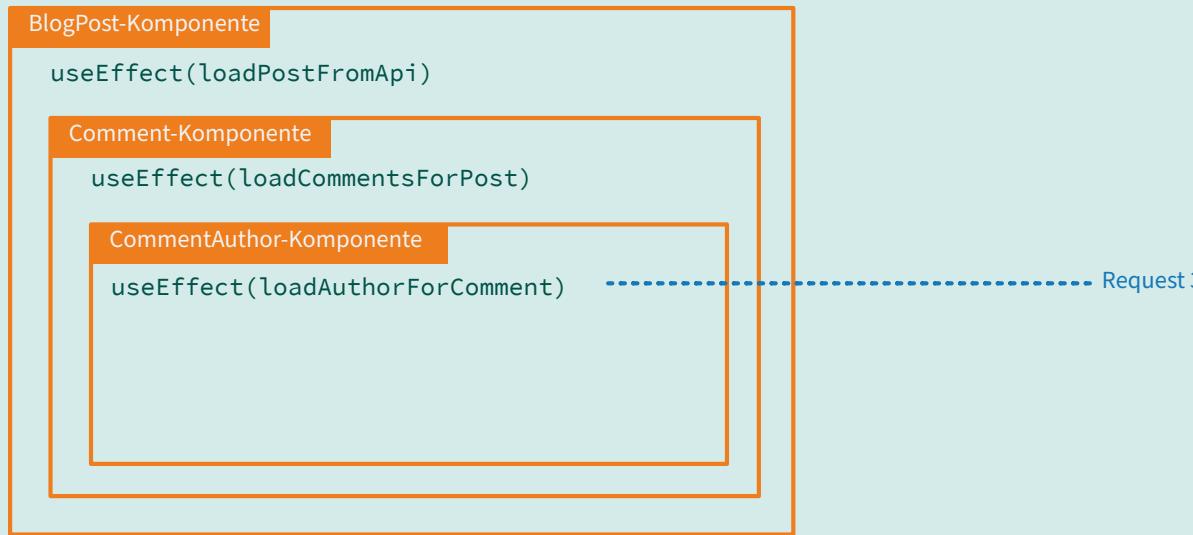
- Eine Komponente lädt ihre Daten, Unterkomponenten müssen warten



DATEN LADEN

Laden von Daten auf dem Client

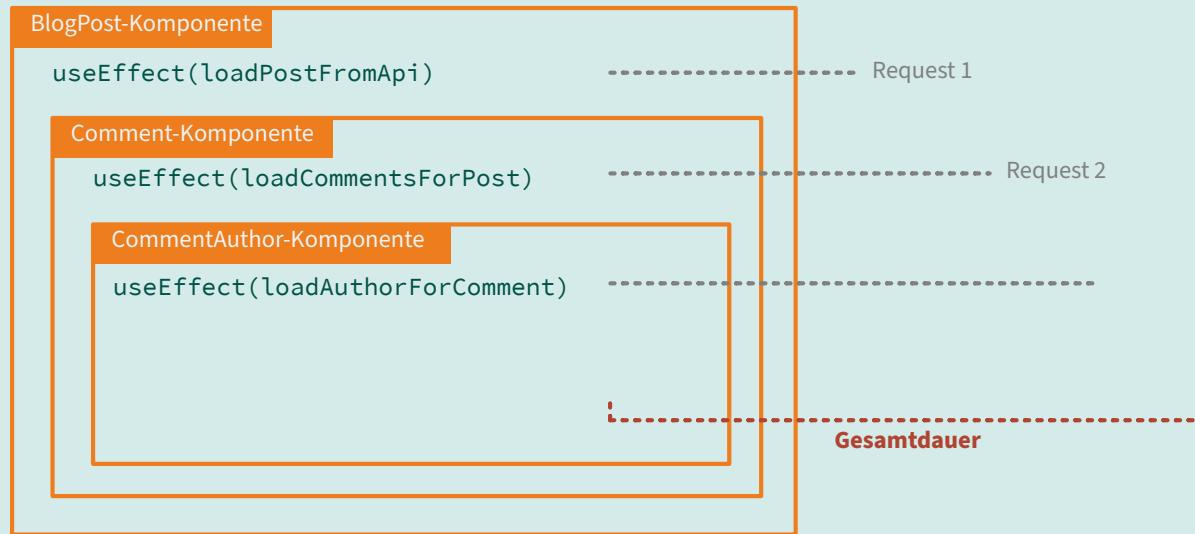
- Eine Komponente lädt ihre Daten, Unterkomponenten müssen warten



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"

SERVER COMPONENTS

Beispiel: Eine Server Komponente

SUSPENSE

Beispiel: Eine Server Komponente

experimental!

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

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

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

Beispiel: Eine Server Komponente

experimental!

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

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

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

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

SUSPENSE

Suspense: React kann das Rendern von Komponenten unterbrechen, während (asynchron) Daten geladen werden

- Funktioniert aktuell (React 18) für **Code Splitting**
 - Code Splitting in Server-Komponenten eingebaut
- **In der Zukunft** auch zum **Laden von beliebigen Daten** (Client und Server)
 - "That will likely come after the 18.0 release, but we're hoping that to have something during the next 18.x minor releases." (<https://github.com/reactwg/react-18/discussions/47#discussioncomment-847004>)

Hintergrund: Suspense for Data Fetching

- Eine Komponente kann auf "etwas" warten
- React weiß, dass die Komponente auf etwas wartet
- Solange gewartet wird, wird eine Fallback-Komponente gerendert
- Die Fallback-Komponente wird oberhalb mit Suspense festgelegt
 - Wie ein try-catch-Handler für ausstehende Daten

SERVER COMPONENTS

Beispiel: Daten laden auf dem Server

```
import db from "./blog-db";  
  
function PostList() {  
  const posts = db.readPosts(); -----  
  
  return ...; // render Posts  
}  
  
function PostListPage() {  
  return <Suspense fallback={<LoadingIndicator />}>  
    <PostList />  
  </Suspense>;  
}
```

"Suspense for Data Loading"

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

SERVER COMPONENTS

Beispiel: Daten laden auf dem Server

```
import db from "./blog-db";

function PostList() {
  const posts = db.readPosts();

  return ...; // render Posts
}

function PostListPage() {
  return <Suspense fallback={<LoadingIndicator />}>
    <PostList />
  </Suspense>;
}
```

Suspense-Komponente

- "Sollbruchstelle", wenn unterhalb in der Anwendung auf "etwas" gewartet wird, wird fallback angezeigt
- Eine Art try-cache für ausstehende Daten
- Wird es wohl so auch auf dem Client geben

SERVER COMPONENTS

Beispiel: Suspense



Demo (falls noch Zeit ist)

- Delay für PostList und TagCloud aktivieren (delay.server.js)
- Daten bleiben gecached (Home => Post => Home)
- Suspense in PostListPage verschieben
- Delay für Post aktivieren
- Post aufrufen

Server Components

SERVER COMPONENTS

Aktueller Stand

- Nicht in React 18 enthalten

Server Components is Still in Development

Server Components is an upcoming feature that allows developers to build apps that span the server and client, combining the rich interactivity of client-side apps with the improved performance of traditional server rendering. Server Components is not inherently coupled to Concurrent React, but it's designed to work best with concurrent features like Suspense and streaming server rendering.

<https://reactjs.org/blog/2022/03/29/react-v18.html>

Aktueller Stand

- Nicht in React 18 enthalten
- Beta-Support u.a. in Next.js
- Vieles noch offen, u.a. Support für die üblichen Bundler

Server Components is Still in Development

Server Components is an upcoming feature that allows developers to build apps that span the server and client, combining the rich interactivity of client-side apps with the improved performance of traditional server rendering. Server Components is not inherently coupled to Concurrent React, but it's designed to work best with concurrent features like Suspense and streaming server rendering.

Server Components is still experimental, but we expect to release an initial version in a minor 18.x release. In the meantime, we're working with frameworks like Next.js, Hydrogen, and Remix to advance the proposal and get it ready for broad adoption.

<https://reactjs.org/blog/2022/03/29/react-v18.html>



vielen Dank!

Slides: <https://react.schule/codetalks-react>

Fragen & Kontakt: nils@nilshartmann.net

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