

**NILS HARTMANN**

<https://nilshartmann.net>

Slides: <https://react.schule/entwicklerde-nextjs>

# Fullstack React

**Schritt-für-Schritt am Beispiel Next.js**

# NILS HARTMANN

nils@nilshartmann.net

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



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

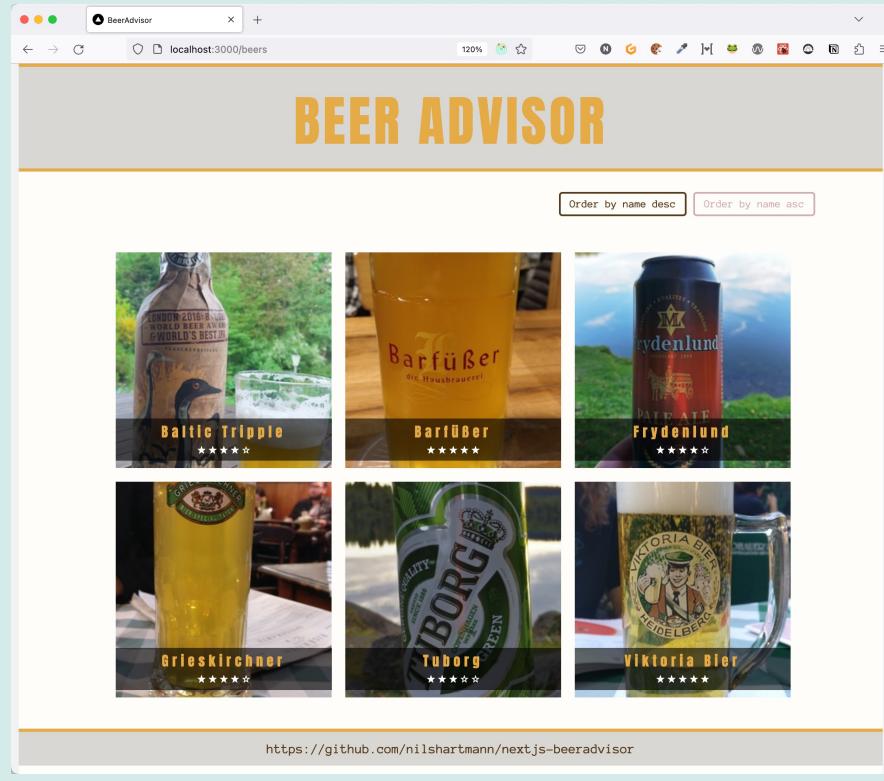
<https://reactbuch.de>

**HTTPS://NILSHARTMANN.NET**

# 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/>



Beispiel-Code: <https://github.com/nilshartmann/nextjs-beeradvisor>

EIN BEISPIEL...

# EIN BEISPIEL

## Was macht die Beispiel-Anwendung aus?

- Viel statischer Content

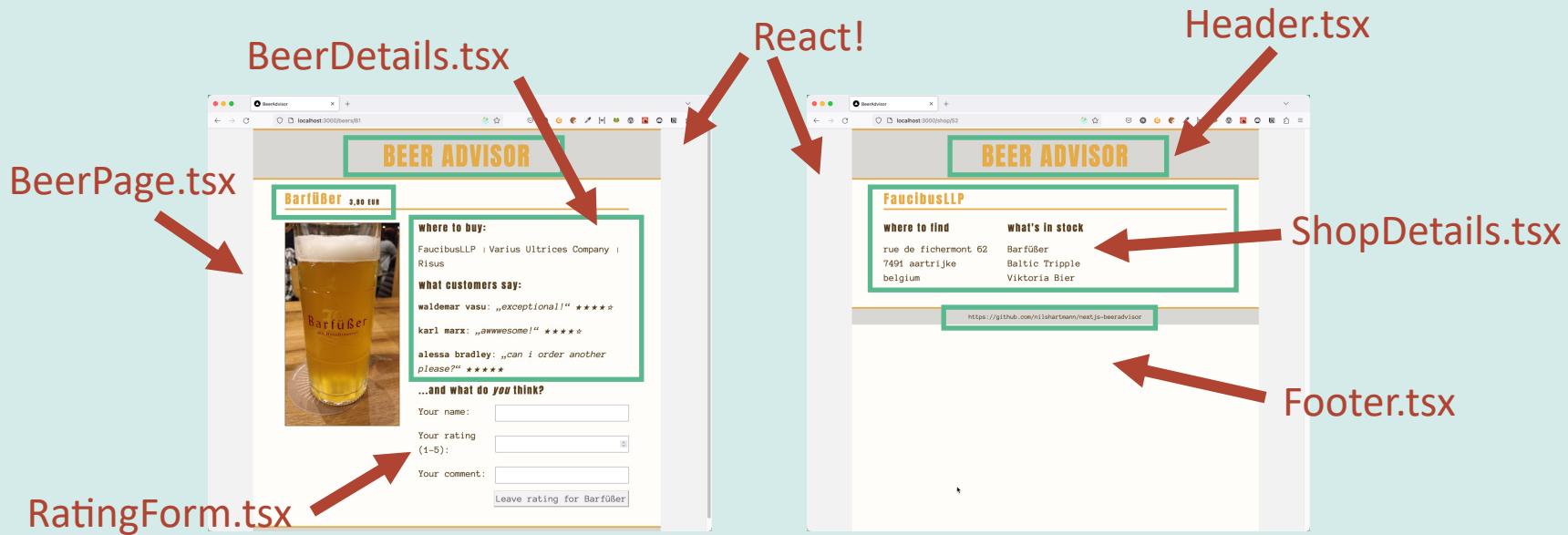
The screenshot shows a web browser window titled "Beer Advisor" at the URL "localhost:3000/beer/81". The main heading is "BEER ADVISOR". Below it, a green box highlights the beer name "Barfüßer" and its price "3.80 EUR". To the left is a photograph of a glass of light beer with a white head. To the right, under the heading "Where to buy:", is the text "FaucibusLLP | Varius Ultrices Company | Risus". Under "what customers say:" are reviews from "waldemar vasu", "karl marx", and "alessa bradley", each accompanied by a five-star rating icon. At the bottom, there's a form for users to leave their own rating and comment.

The screenshot shows the same "Beer Advisor" application at the URL "localhost:3000/beer/82". The main heading is "BEER ADVISOR". Below it, a green box highlights the beer name "FaucibusLLP". To the left is a photograph of a glass of beer. To the right, under the heading "where to find:", is the text "rue de fichermon 62 7491 aartrijke belgium". Under "what's in stock:" are the names "Barfüßer", "Baltic Triple", and "Viktoria Bier". At the bottom, there's a link "https://github.com/nishhartmann/nextjs-beeradvisor".

# EIN BEISPIEL

## Was macht die Beispiel-Anwendung aus?

- Viel statischer Content
- Viel JavaScript 😱



# EIN BEISPIEL

## Was macht die Beispiel-Anwendung aus?

- Viel statischer Content
- Viel JavaScript 😱
- ...gleichzeitig wenig Interaktion 😔

The screenshot shows a web browser window titled "BEER ADVISOR". The main content is for a product named "Barfüßer 3,80 EUR". It includes a photograph of a glass of beer, a section titled "where to buy:" listing "FaucibusLLP | Varius Ultrices Company | Risus", and a section titled "what customers say:" with reviews from "waldemar vasu", "karl marx", and "alesse bradley". At the bottom, there is a form with a red border containing fields for "Your name:", "Your rating (1-5):", and "Your comment:". A button at the bottom right says "Leave rating for Barfüßer".

The screenshot shows a web browser window titled "BEER ADVISOR". The main content is for a product named "FaucibusLLP". It includes a section titled "where to find" with the address "rue de fichermon 62 7491 aartrijke belgium" and a section titled "what's in stock" listing "Barfüßer Baltic Triple Viktoria Bier". At the bottom, there is a link "https://github.com/nishartmann/reactjs-beeradvisor".

## EIN BEISPIEL

### Anforderung

👉 Die Seiten sollen möglichst schnell für den Benutzer **sichtbar** und **bedienbar** sein

## EIN BEISPIEL

### Mögliche Probleme

- (Viel) JavaScript-Code, der...

## EIN BEISPIEL

### Mögliche Probleme

- (Viel) JavaScript-Code, der...
  - ... vom Browser geladen werden muss

### Mögliche Probleme

- (Viel) JavaScript-Code, der...
  - ... vom Browser geladen werden muss
  - ... interpretiert und ausgeführt werden muss

### Mögliche Probleme

- (Viel) JavaScript-Code, der...
  - ... vom Browser geladen werden muss
  - ... interpretiert und ausgeführt werden muss
- ...und mit jeder neuen Komponente mehr wird

**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

## 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: Kein JavaScript für die Darstellung notwendig

## 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: Kein JavaScript für die Darstellung notwendig
3. Ebenfalls wird der **komplette Anwendungscode** zum Client geschickt
  - 😢 Auch für "statische" Komponenten
  - 😢 Bandbreite! Performance!

# 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:**

- Platzhalter für "langsame" Teile einer Seite
- Mit Streaming können diese Teile einer Seite "nachgeliefert" werden, sobald sie gerendert sind

# "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:**

- Platzhalter für "langsame" Teile einer Seite
- Mit Streaming können diese Teile einer Seite "nachgeliefert" werden, sobald sie gerendert sind

- **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>

## SERVER COMPONENTS

**Idee:** Komponenten werden nicht im Client ausgeführt

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

👉 "Zero-Bundle-Size"

## SERVER COMPONENTS

### Arten von Komponenten

# ARTEN VON KOMPONENTEN

## Client-Komponenten (wie bisher)

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

**BEER ADVISOR**

**Barfüßer 3,80 EUR**

**where to buy:**  
FaucibusLLP | Varius Ultrices Company | Risus

**what customers say:**  
**waldemar vasu:** „exceptional!“ ★★★★☆  
**karl marx:** „awwwesome!“ ★★★★☆  
**alessa bradley:** „can i order another please?“ ★★★★☆

**...and what do you think?**

Your name:

Your rating (1-5):

Your comment:

Leave rating for Barfüßer



### Neu: Server-Komponenten

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

## DREI ARTEN VON KOMPONENTEN

### Neu: Server-Komponenten

- werden auf dem *Server* oder im *Build-Prozess* gerendert
- 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

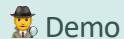
- Ein Teil der Komponenten kommt jetzt vom Server...
- **Server Komponenten sind nicht auf dem Client vorhanden!**
- Der Server rendert die Komponenten, bis er auf eine Client-Komponente trifft



# DREI ARTEN VON KOMPONENTEN

## Weiterhin ein Komponenten-Baum

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



### Demo

- beers/beerId/page im Code:
- Server-Komponenten "Page" und "RatingList" gibt es im Source als Komponenten, aber nicht auf dem Client (-> React Dev Tools)
- RatingForm ist aber vorhanden



# RSC am Beispiel Next.js

## React empfiehlt "Fullstack-Framework"

- **Server Components** erfordern Rendern auf dem Server oder im Build
- Dazu braucht man ein "**Fullstack-Framework**"

## React empfiehlt "Fullstack-Framework"

- **Server Components** erfordern Rendern auf dem Server oder im Build
- Dazu braucht man ein "**Fullstack-Framework**"
- "**Framework**" ist verharmlosend, weil es sich in der Regel um einen kompletten Stack samt Build-Tools und Laufzeitumgebung handelt

## React empfiehlt "Fullstack-Framework"

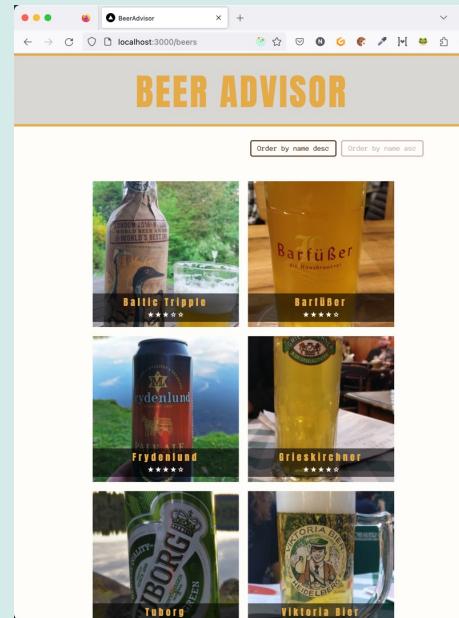
- **Server Components** erfordern Rendern auf dem Server oder im Build
- Dazu braucht man ein "**Fullstack-Framework**"
- "**Framework**" ist verharmlosend, weil es sich in der Regel um einen kompletten Stack samt Build-Tools und Laufzeitumgebung handelt
- Deswegen werden solche Frameworks auch als "**Meta-Frameworks**" bezeichnet (=> Sammlung von Frameworks)

## React empfiehlt "Fullstack-Framework"

- **Next.js** entspricht den Vorstellungen des React-Team
  - Mit dem "App-Router", stabil ab Next.js 13.4

## Schritt-für-Schritt

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



## Schritt 1: Eine Server Komponente



Demo

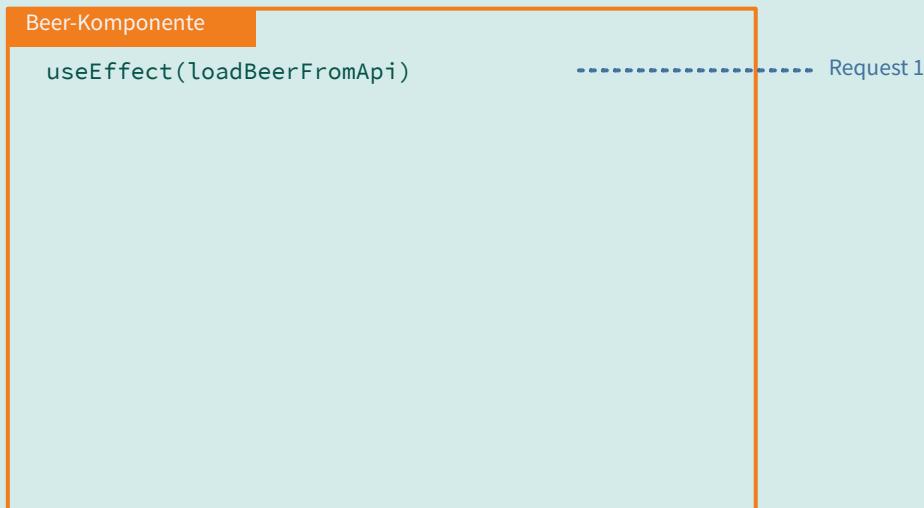
- Landing-Page mit Link auf /beers
- Children in Layout
- console.log in Page-Komponente

# Data Fetching

# DATEN LADEN

## Laden von Daten auf dem Client

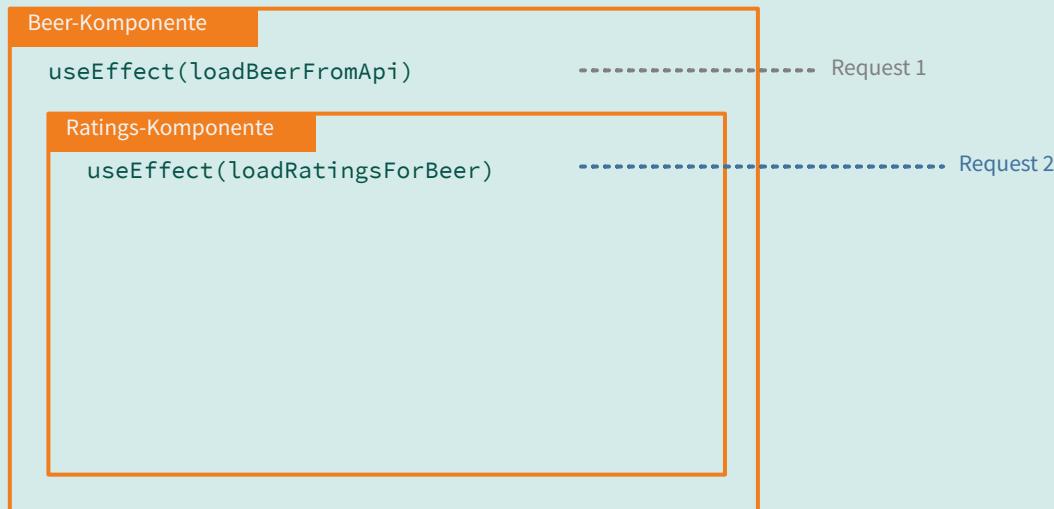
- Eine Komponente lädt ihre Daten, Unterkomponenten müssen warten



# DATEN LADEN

## Laden von Daten auf dem Client

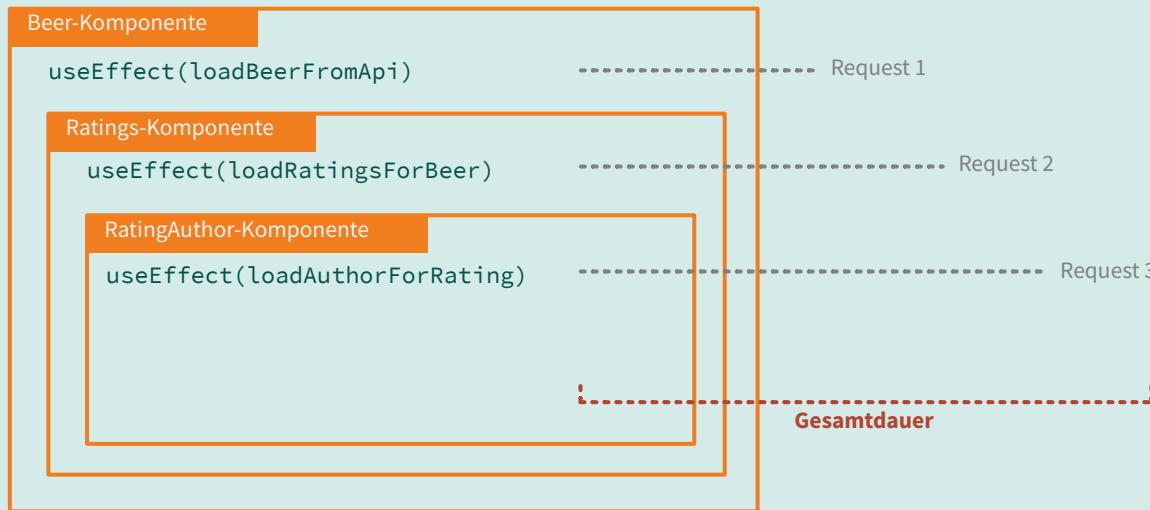
- Eine Komponente lädt ihre Daten, Unterkomponenten müssen warten



# DATEN LADEN

## Laden von Daten auf dem Client

- Eine Komponente lädt ihre Daten, Unterkomponenten müssen warten



😓 Wasserfall...

### Idee

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

## 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-Komponenten können asynchron sein

## Schritt 2: Eine asynchrone Server-Komponente



Demo

- BeerListPage anlegen
- DB-Zugriff mit loadBeers
  - loadBeers zeigen
- BeerImageList verwenden, um Beers anzuzeigen
- 🔎 **statische Komponenten bislang! (Build!)**

## Schritt 3: Eine asynchrone Server-Komponente, träge ist



Demo

- beers/[beerId] Beer-Page mit DB (loadBeer)
- `type BeerPageProps = { params: { beerId: string } };`
- Fertige Komponente aus beer-details-page-fragment.tsx kopieren
- Aufruf künstlich verzögern (sleep in loadBeer)
- loading.tsx

## Schritt 3: Eine asynchrone Server-Komponente, die zwei Daten Quellen benötigt

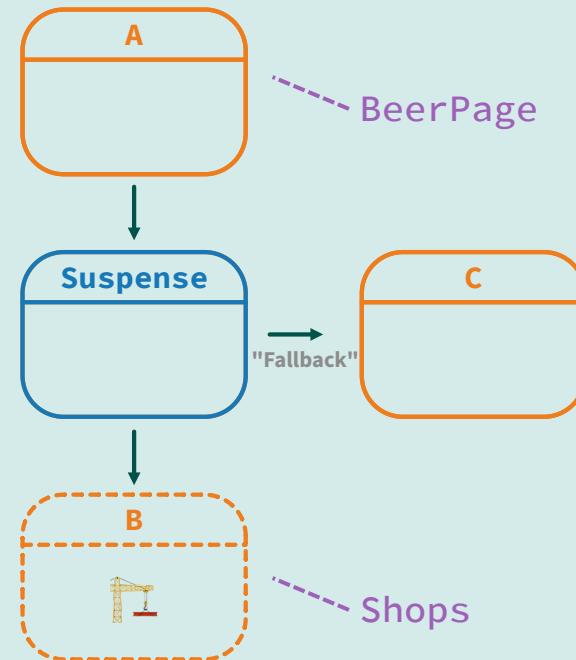


Demo

- beers/[beerId] Beer-Page wieder schnell machen (slow entfernen)
- beers/[beerId] Beer-Page shops erweitern (fertiges fetch in db-queries)
- Zeigen: Promise an Unterkomponente (Shops)
  - -> Parallel fetching!
- Aufruf künstlich verzögern (slow=2400)
- 😞 Jetzt wartet die ganze Seite auf die Shops...
- -> Suspense vorstellen (Slides)

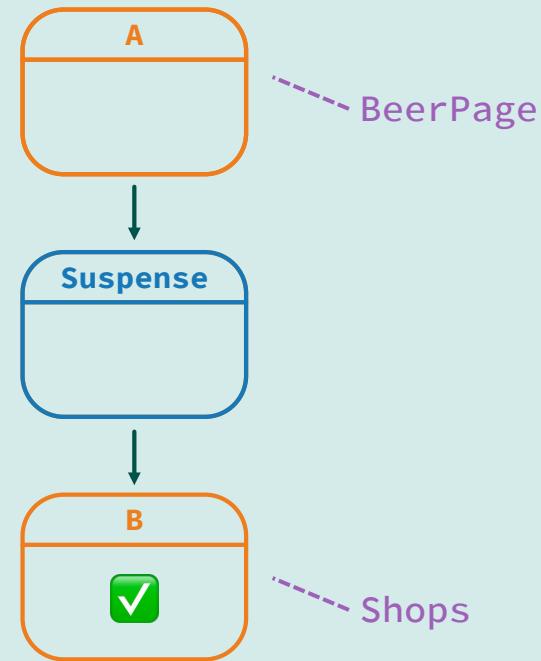
## SUSPENSE

**Suspense:** Unterbricht das Rendern, solange "etwas" fehlt



## SUSPENSE

**Suspense:** Unterbricht das Rendern, solange "etwas" fehlt



## Schritt 4: Suspense



Demo

- Aufruf von `loadShops`  
verlangsamen (`[beerId]/page.tsx`)
- Suspense in `BeerDetails`  
einführen

# Aufteilung in Server-Client: Konsequenzen

```
type BeerListProps = {
  beers: SingleBeer[];
  onToggleOrder(): void;
};

export default function BeerList({ beers, onToggleOrder }: BeerListProps) {
  return (
    <div>
      <h1>Beers</h1>

      <ul>
        {beers.map((b) => (
          <li key={b.id}>{b.name}</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.

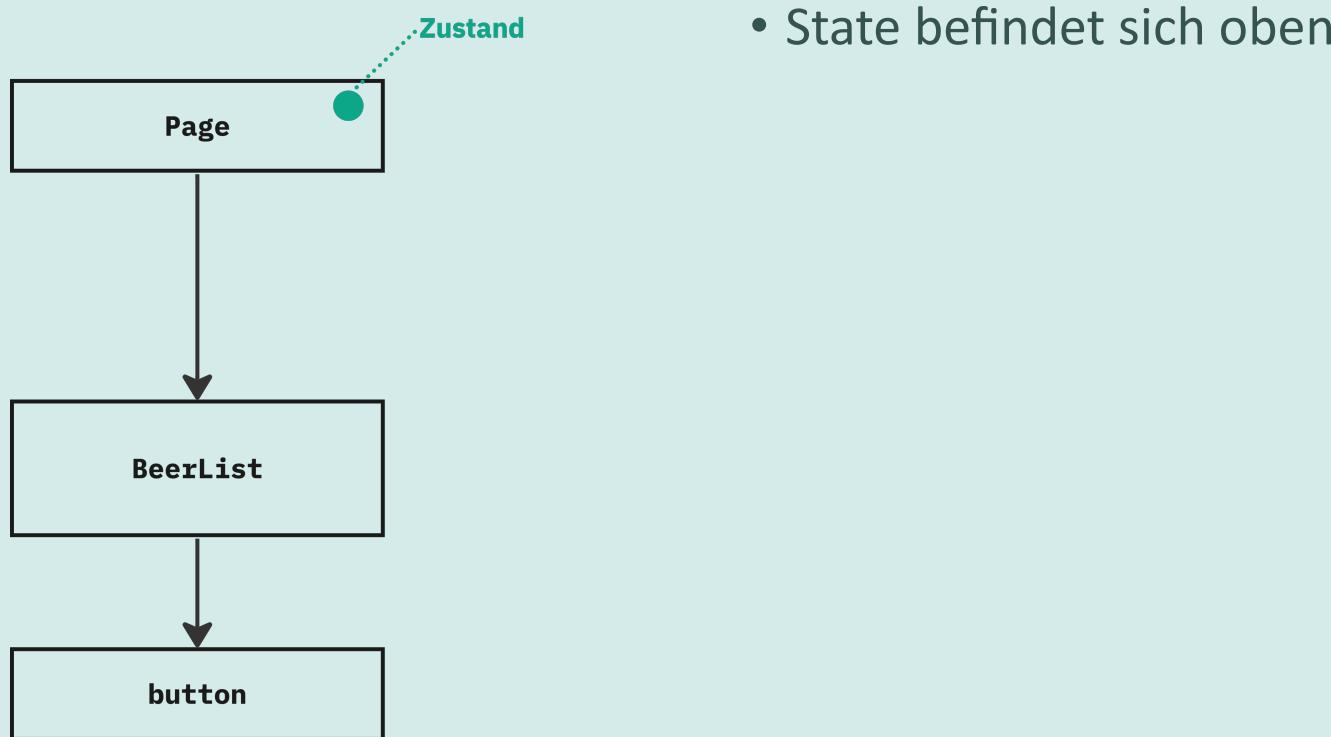
```
<button onClick={function} children=...>  
    ^^^^^^^^^^
```

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

at stringify (<anonymous>)

CAN YOU SPOT THE PROBLEM?

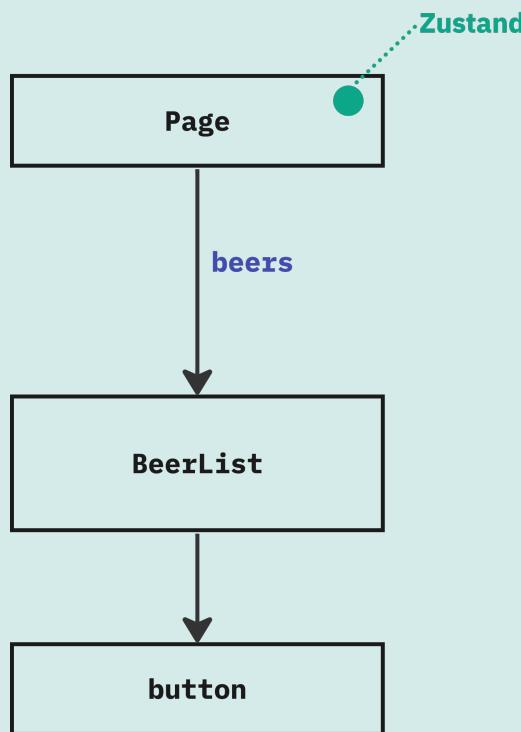
# EINE REACT ANWENDUNG IM BROWSER



- State befindet sich oben

Eine "normale" React-Anwendung...

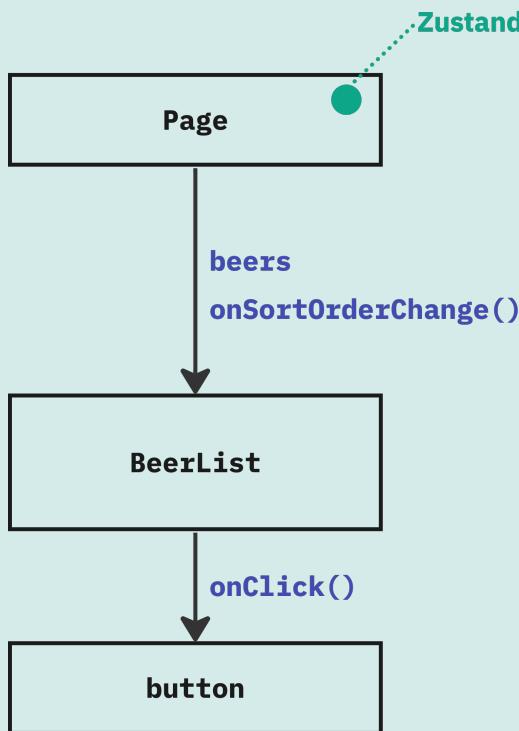
# EINE REACT ANWENDUNG IM BROWSER



- State befindet sich oben
- Daten werden runtergereicht ("props")

Eine "normale" React-Anwendung...

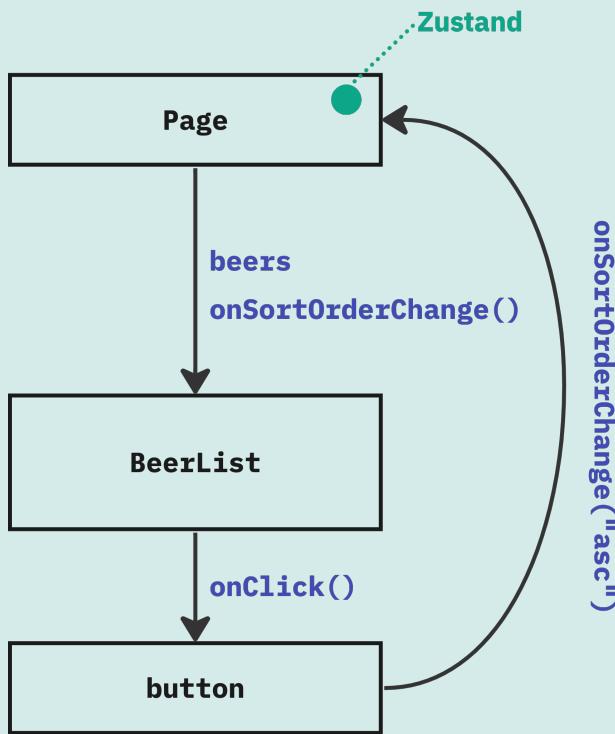
# EINE REACT ANWENDUNG IM BROWSER



- State befindet sich oben
- Daten werden runtergereicht ("props")
- Callbacks werden runtergereicht

Eine "normale" React-Anwendung...

# EINE REACT ANWENDUNG IM BROWSER

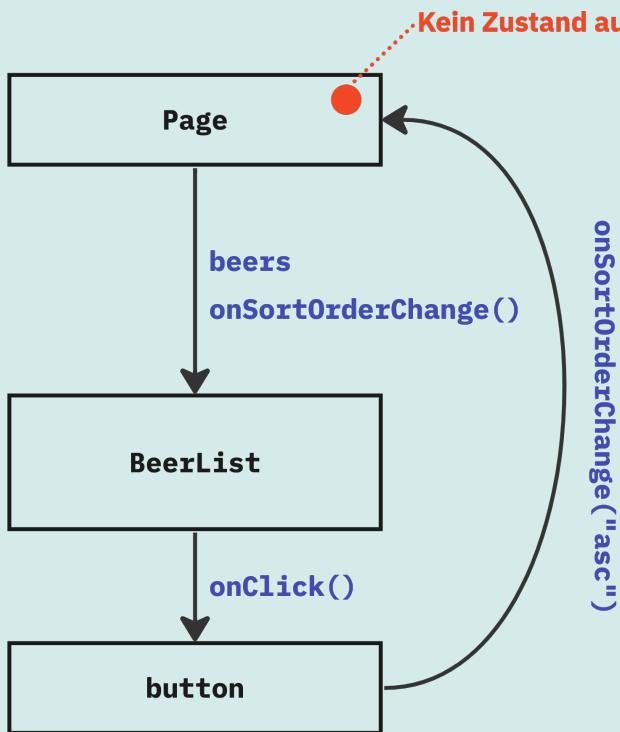


- State befindet sich oben
- Daten werden runtergereicht ("props")
- Callbacks werden runtergereicht
- Über Callbacks kann State-Veränderung ausgelöst werden

Eine "normale" React-Anwendung...

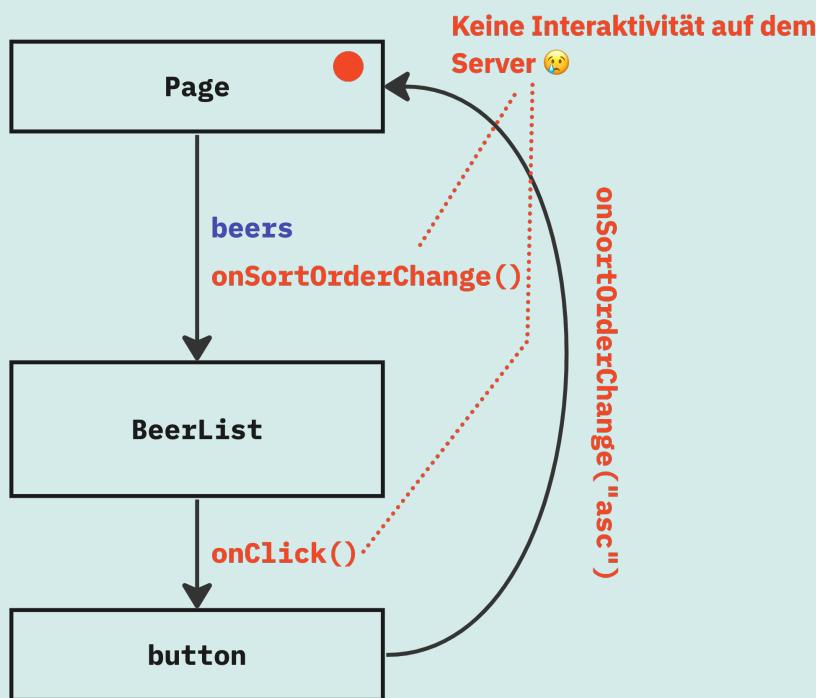
## ...UND AUF DEM SERVER

- Auf dem Server gibt es keinen State!



Mit Next.js sind wir aber auf dem Server (by Default)

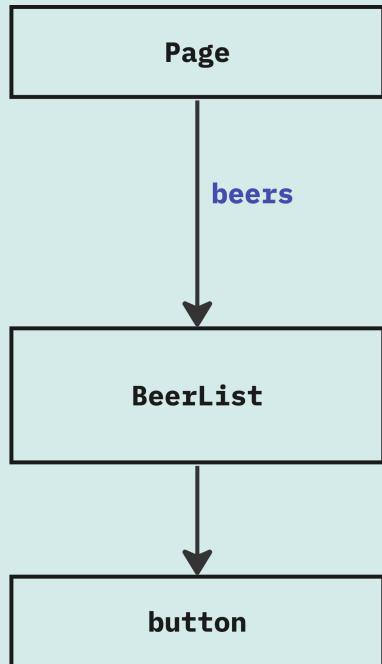
## ...UND AUF DEM SERVER



- Auf dem Server gibt es keinen State!
- ...und keine Interaktion

Mit Next.js sind wir aber auf dem Server (by Default)

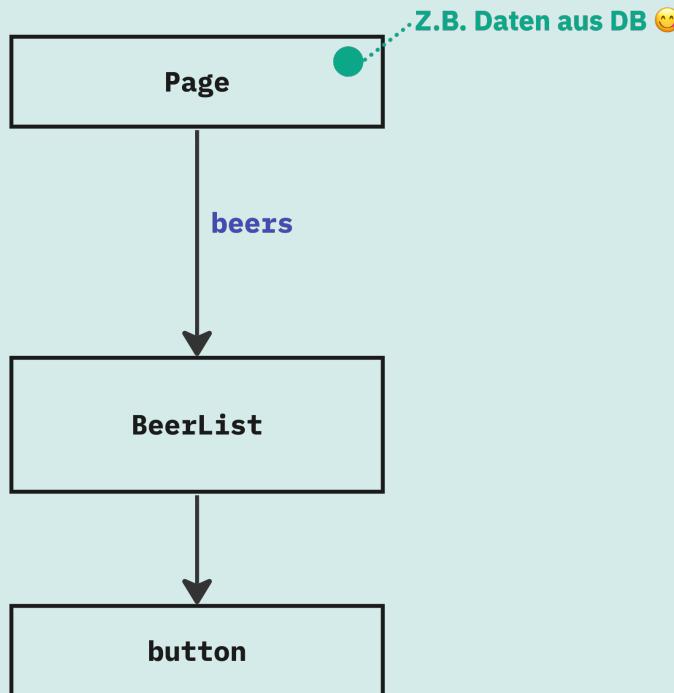
## ...UND AUF DEM SERVER



- Auf dem Server gibt es keinen State!
- ...und keine Interaktion
- Wir haben nur statischen Content

Mit Next.js sind wir aber auf dem Server (by Default)

## ...UND AUF DEM SERVER

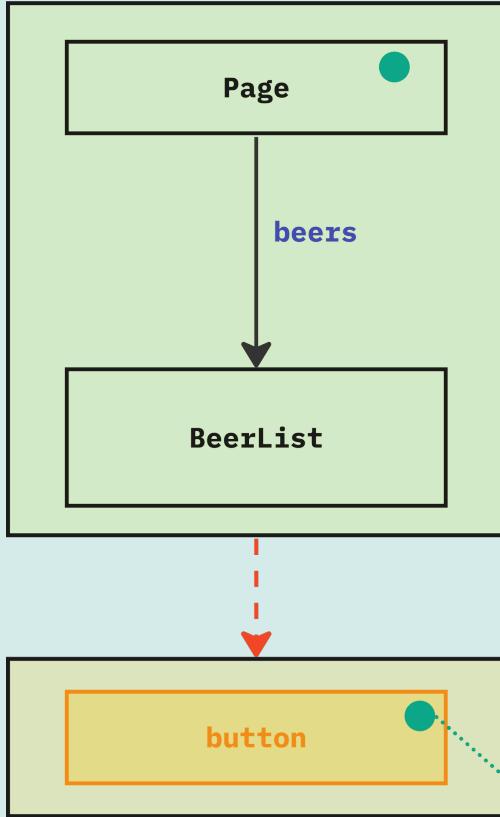


- Auf dem Server gibt es keinen State!
- ...und keine Interaktion
- Wir haben nur statischen Content
- Wir haben aber **Daten**  
z.B. aus DB, Microservice, Filesystem...

Mit Next.js sind wir aber auf dem Server (by Default)

## ...UND AUF DEM SERVER

Server



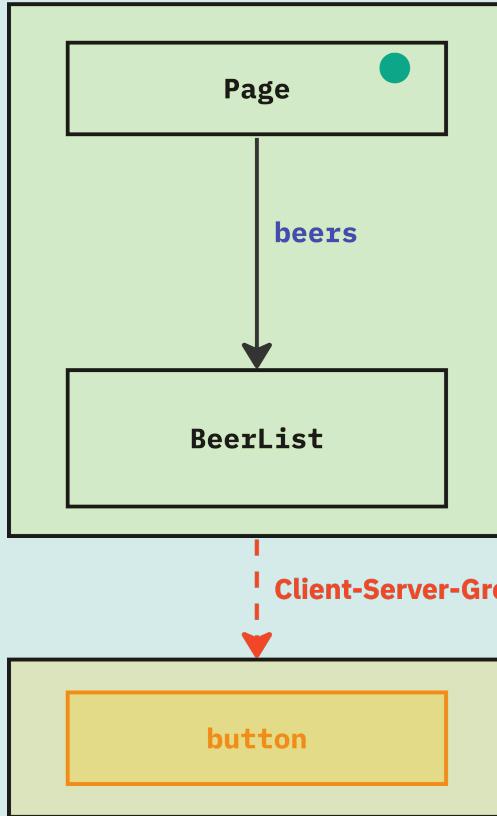
- Bestimmte Teile müssen auf den Client
  - Event-Handler
  - Lokaler State
  - Effekte

Client

Interaktives muss auf den Client 😎

## ...UND AUF DEM SERVER

Server

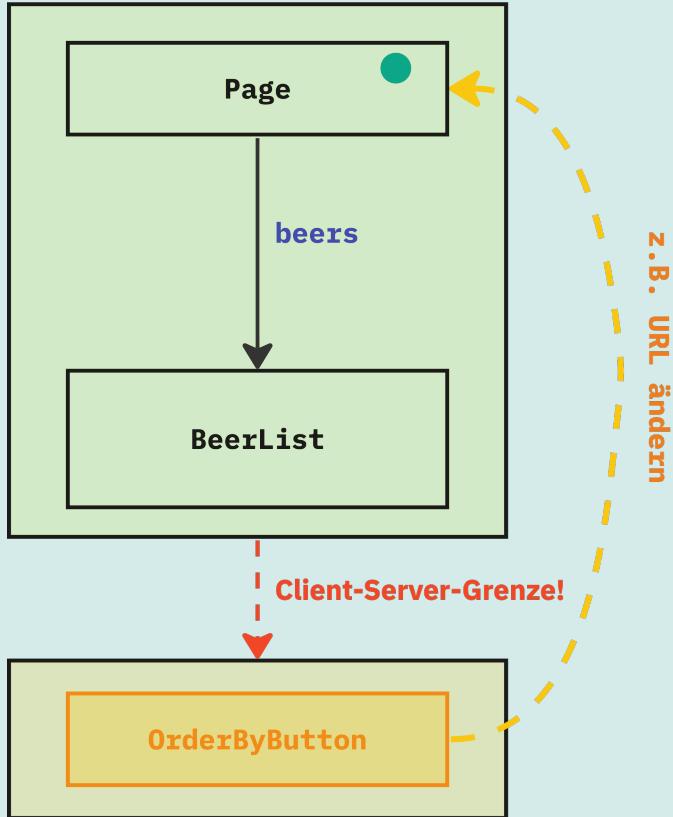


- Properties müssen hier Client-Server-Grenze überwinden
- Müssen serialisierbare Daten sein
- **Keine (Callback-)Funktionen!**

Client

## ...UND AUF DEM SERVER

Server

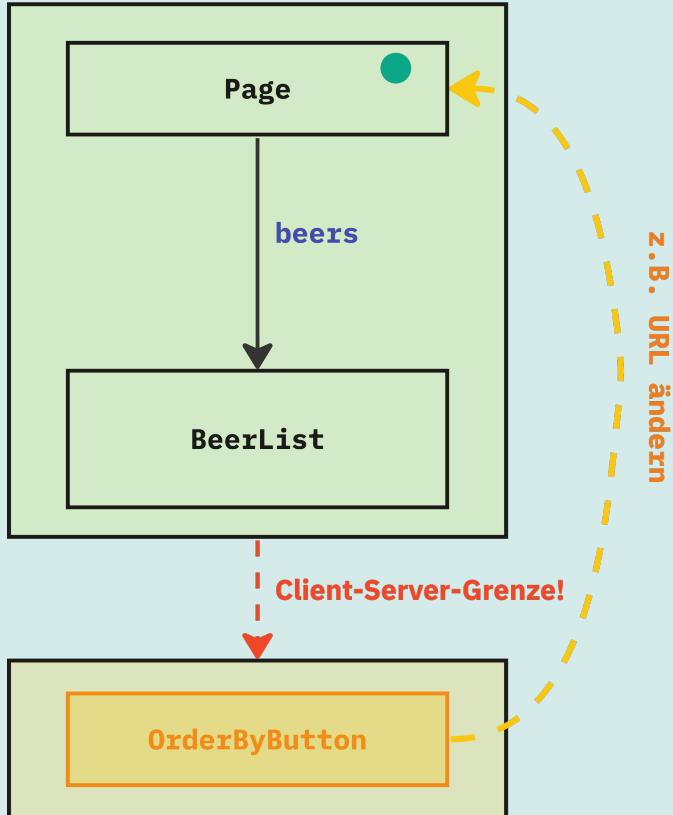


- Properties müssen Client-Server-Grenze überwinden
- Müssen serialisierbare Daten sein
- Keine (Callback-)Funktionen!
- Zur Kommunikation müssen **Server-Requests** durchgeführt werden
  - z.B. URL ändern

Client

## ...UND AUF DEM SERVER

Server

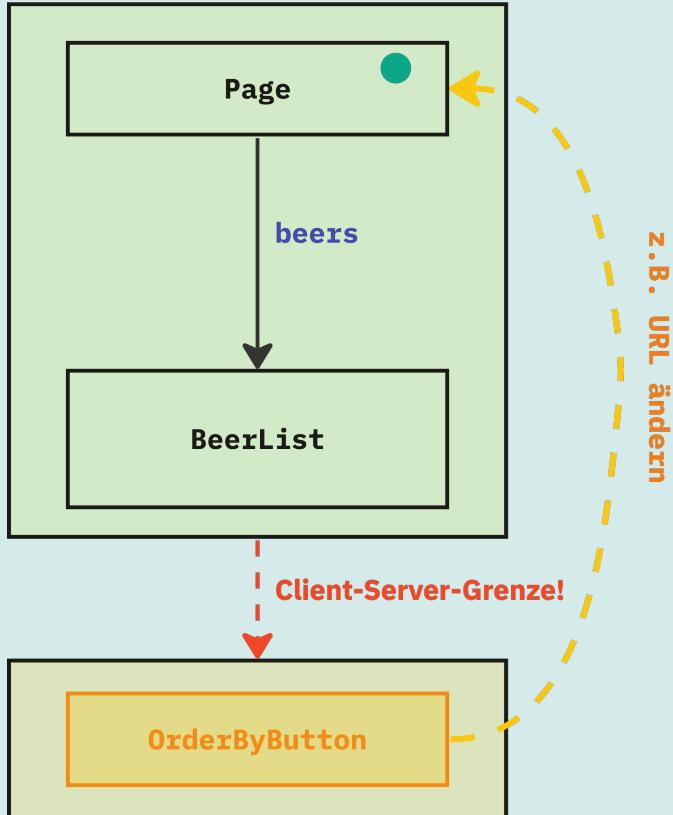


- Properties müssen Client-Server-Grenze überwinden
- Müssen serialisierbare Daten sein
- Keine (Callback-)Funktionen!
- Zur Kommunikation müssen Server-Requests durchgeführt werden
  - z.B. URL ändern
- **Server-Komponente hat Zugriff auf Request Informationen**
  - URL mit Search Params
  - Cookies
  - Headers

Client

## ...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 generiert
- Fertiges UI kommt dafür vom Server
  - Das kann mehr Daten als bei (REST-)API-Call bedeuten!

Client

## Schritt 5: Eine Client-Komponente



### Demo

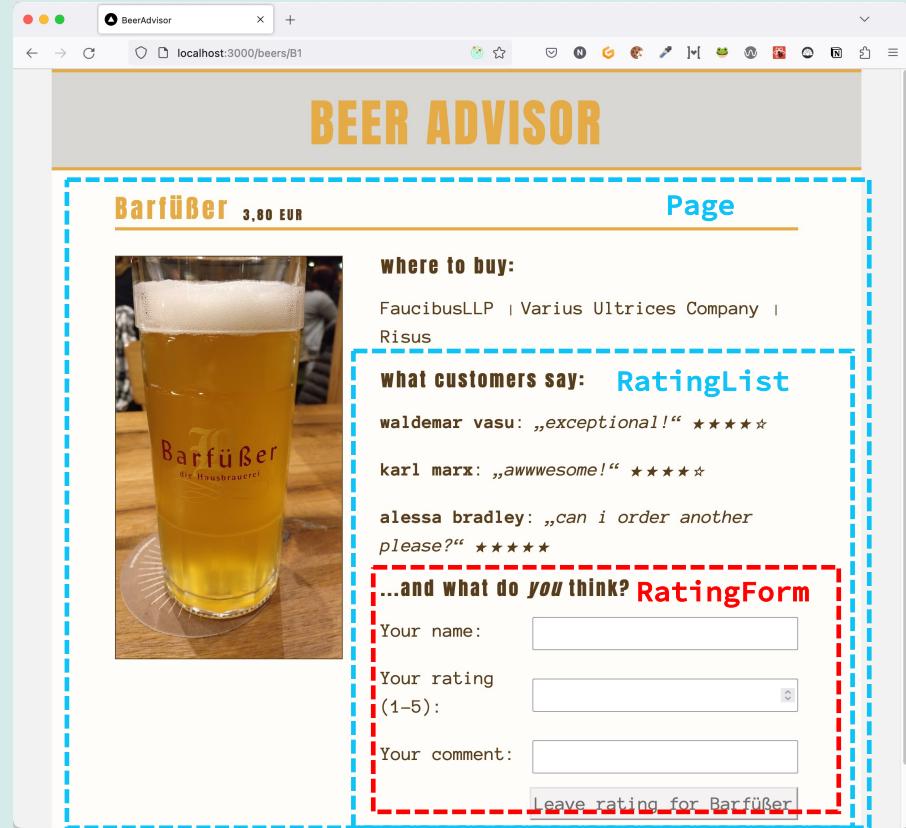
- OrderByButton Komponente bauen
  - OrderBy-Type als Property ("name\_asc" , "name\_desc")
  - Client-Komponente: Hooks möglich, useBeerAdvisorSearchParams
    - URL als "globaler Zustand"
- In BeerListPage einbauen
- In BeerListPage abhängig von SearchParams sortieren
  - An dieser Stelle Server Komponente, d.h. Hook ist hier nicht verwendbar
- ```
type BeerListPageProps = {
  searchParams?: { [key: string]: string };
};

const orderBy: OrderBy = (searchParams?.order_by || "name_asc") as OrderBy;
```

# MUTATIONS

## Verändern von Daten: Hinzufügen einer Bewertung

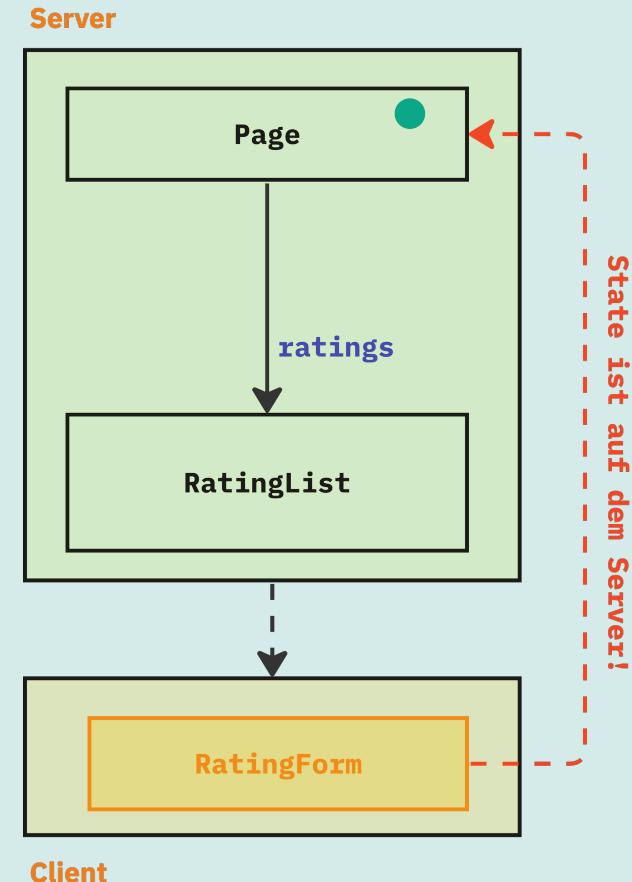
Server-Komponente  
Client-Komponente



## MUTATIONS

### Verändern von Daten

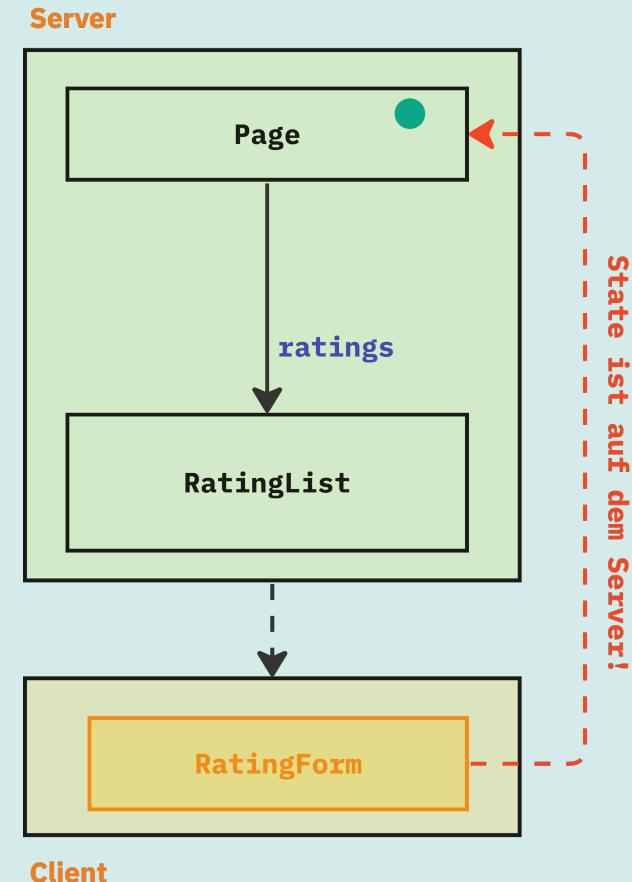
- Nach dem Verändern von Daten muss die UI aktualisiert werden
- Mangels State auf dem Client geht das aber nicht wie bislang



## MUTATIONS

### Verändern von Daten

- Nach dem Verändern von Daten muss die UI aktualisiert werden
- Mangels State auf dem Client geht das aber nicht wie bislang
- Der **Server** muss nach Datenänderungen **aktualisierte UI** liefern



## Schritt 6a: Exkurs API Endpunkte mit Next.js



Demo

- **API Route Handler**
  - Beispiel: GET Endpunkt implementieren !!
  - api/beers/route.ts
  - GET mit simplem NextResponse.json

## Schritt 6: Ein Formular

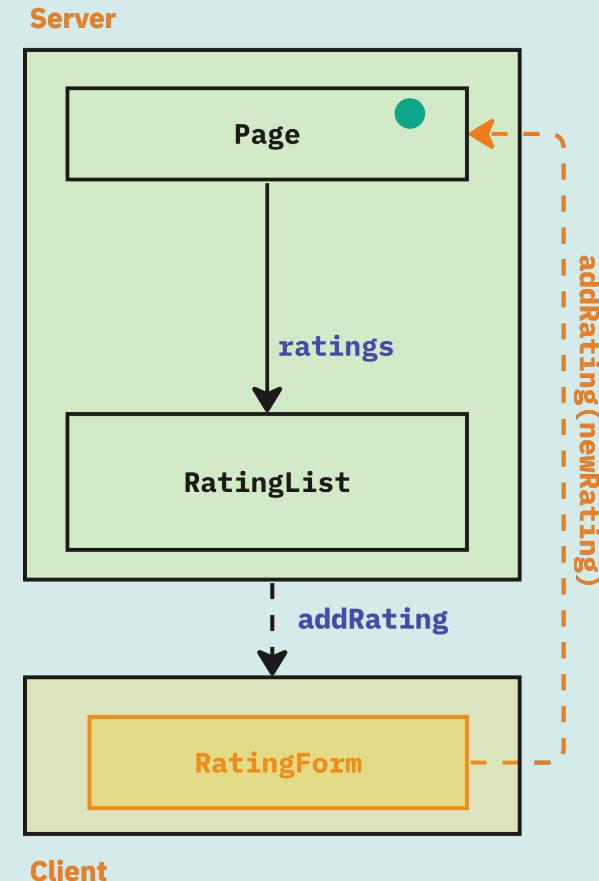


- API Route Handler
  - Fertigen POST Handler kopieren! (api/beers/[beerId])
- In AddRatingForm onSave implementieren
- mit fetch:
  - POST `/api/beers/\${beerId}/rating`
  - Content-Type-Header
  - Body: AddRatingRequestBody)
- Was passiert mit der Darstellung => nix
  - Warum?
- router.refresh()

## MUTATIONS

### Alternative: Server Actions

- **Experimentelles** React-Feature
- Das ist eine Art Remote Funktion, die aus einer Server- oder Client-Komponente aufgerufen werden kann



## Ausblick: Server Action



- Fertige Action in src-Verzeichnis kopieren (aus 99\_fertig)
  - Zeigen
- AddRatingForm anpassen
  - useTransition
  - wenn response.status === "created" dann ist alles gut, State zurücksetzen
  - Aktualisierung der UI: revalidatePath hinzufügen

# Fazit

### Server Components

- Eröffnen neue Use-Cases für React
  - Aber: man muss die Probleme haben, die Server Components lösen!
  - Bitte auf eigene Anforderungen schauen!

### Server Components

- Eröffnen neue Use-Cases für React
  - Aber: man muss die Probleme haben, die Server Components lösen!
  - Bitte auf eigene Anforderungen schauen!
- API ist sehr nah an der heutigen React API
  - aber mitunter große Architektur-Unterschiede!

### Next.js ist "battery-included" Framework

- "Battery-included": Routing, Build, Data Fetching
- Eigene Cache-Lösung ist sehr "speziell"
- Offen:
  - was ist mit Testen?
  - wie sieht ein Migrationspfad aus?
- Man muss die Probleme haben, die das Framework löst!

### Alternative: Klassische Single-Page-Anwendung

- Klassische SPAs wird es weiter geben
  - auch mit React

### Alternative: Klassische Single-Page-Anwendung

- **Klassische SPAs wird es weiter geben**
  - auch mit React
- **React wird neue APIs auch für die Client-Seite anbieten**
  - Suspense wird auf dem Client funktionieren
  - use-Hook in Client-Komponenten oder async-Komponenten

### Alternative: Klassische Single-Page-Anwendung

- Klassische SPAs wird es weiter geben
  - auch mit React
- React wird neue APIs auch für die Client-Seite anbieten
  - Suspense wird auf dem Client funktionieren
  - use-Hook in Client-Komponenten oder async-Komponenten
- Es gibt viele Bibliotheken, die typische Probleme lösen
  - Router
  - Data Fetching

**NILS HARTMANN**  
<https://nilshartmann.net>



# vielen Dank!

Slides: <https://react.schule/entwicklerde-nextjs>

Fragen & Kontakt: [nils@nilshartmann.net](mailto:nils@nilshartmann.net)

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