

**NILS HARTMANN**

<https://nilshartmann.net>

Slides: <https://react.schule/gedoplan-2024>

**Von oben bis unten JavaScript?**

# **Fullstack- Anwendungen mit React und Next.js**

GEDOPLAN EXPERTENKREIS | ONLINE, 18. APRIL 2024 | @NILSHARTMANN

# 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)

## "React fortgeschrittenen Workshop"

<https://gedoplan.de/it-schulungen/react-fortgeschrittenen-workshop/>

## "Fullstack React-Anwendungen mit Next.js"

<https://gedoplan.de/it-schulungen/workshop-fullstack-react-anwendungen-mit-next-js/>

# Fullstack- Anwendungen

...ein Blick zurück...

## Historie von Webanwendungen

## Der Anfang...



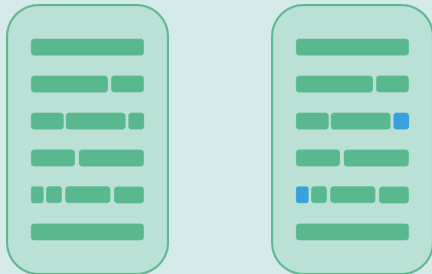
- Nur der Server ist für die UI (HTML/CSS) verantwortlich
- Der Browser ist nur für die Darstellung zuständig (keine Logik)
- Klare Trennung der Verantwortlichkeit

## Der Anfang...



- Problem: nur mit HTML / CSS (fast) keine Interaktivität möglich 😞
- Für **Interaktivität** wird zwingend **JavaScript** benötigt

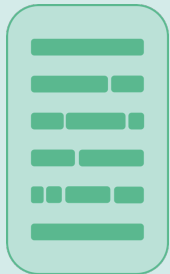
## "Internet 2.0"



- JavaScript-Schnipsel werden eingestreut
  - jquery etc.
- Kleinere Interaktivität jetzt möglich
- Verantwortlichkeiten jetzt zwischen Server und Browser aufgeteilt

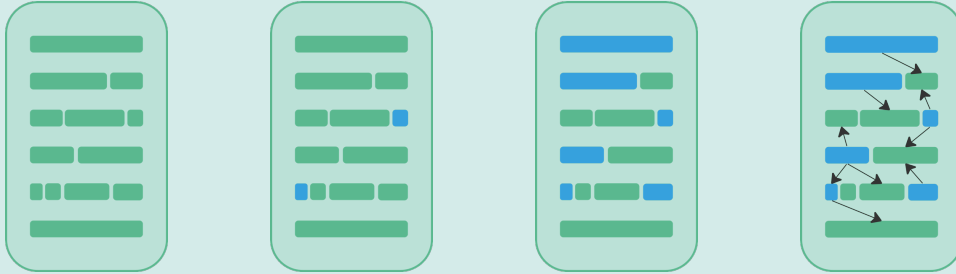


## "Internet 2.0"



- Immer mehr Interaktivität: immer **mehr JavaScript**

## "Internet 2.0"



- Problem: Code der Anwendung wird schwer verständlich und fehlerhaft 😞
- Bunter Strauß an Server- und Client-Technologien
- Verantwortlichkeit willkürlich auf Frontend und Backend aufgeteilt

## Single-Page-Anwendungen (seit ca. 2010)



- Single-Page-Anwendungen: nur noch JavaScript

## Single-Page-Anwendungen (seit ca. 2010)



- Single-Page-Anwendungen: nur noch JavaScript
- Websites werden jetzt zu Anwendungen

## Single-Page-Anwendungen (seit ca. 2010)



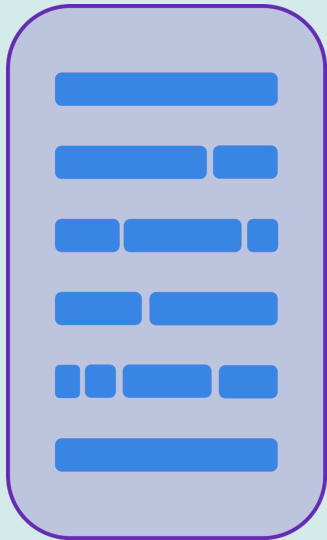
- **Single-Page-Anwendungen:** nur noch JavaScript
- Websites werden jetzt zu Anwendungen
- Klare Verantwortlichkeit: Server für Daten, Client für UI

## Single-Page-Anwendungen (seit ca. 2010)



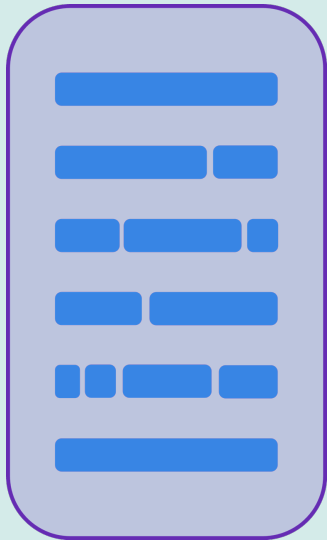
- **Single-Page-Anwendungen:** nur noch JavaScript
- Websites werden jetzt zu Anwendungen
- Klare Verantwortlichkeit: Server für Daten, Client für UI
- Modernes Tooling erlaubt UI-Entwicklung wie vom Backend gewohnt

## Single-Page-Anwendungen



- Problem: nun viel JavaScript im Browser
- Für **statische Inhalte** wird aber kein **JavaScript** benötigt

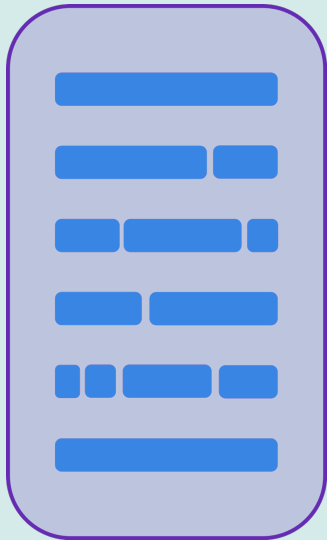
## Single-Page-Anwendungen



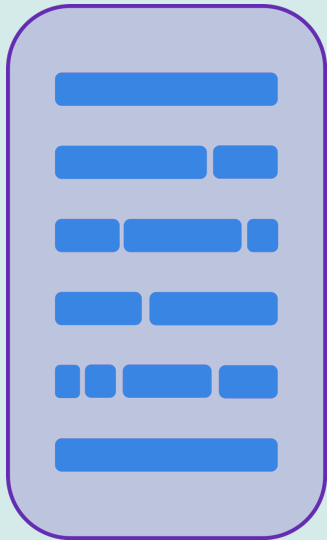
- JavaScript-Code:
  1. muss zum Browser gesendet werden
  2. muss vom Browser ausgeführt werden
  3. kann dann die darzustellenden Daten lesen
  4. kann dann erst die Daten anzeigen
  5. erst dann ist die Anwendung einsatzbereit
  6. Mit jedem Feature wird es mehr



## Fullstack-Anwendungen

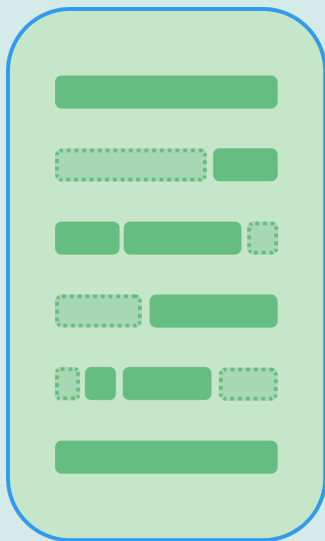


## Fullstack-Anwendungen



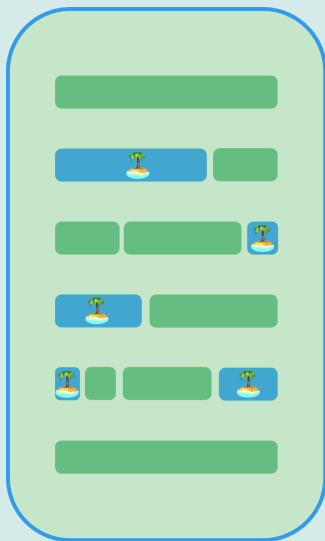
- Ebenfalls vollständig in **JavaScript** geschrieben, aber:

## Fullstack-Anwendungen



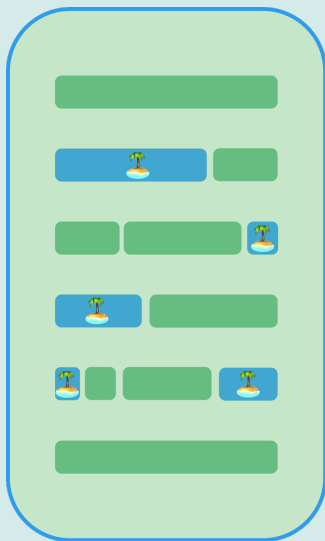
- Vollständig in **JavaScript** geschrieben, aber:
  1. **UI-Code** wird serverseitig vorgerendert
  2. **UI-Code** wird zum Browser gesendet und angezeigt

## Fullstack-Anwendungen



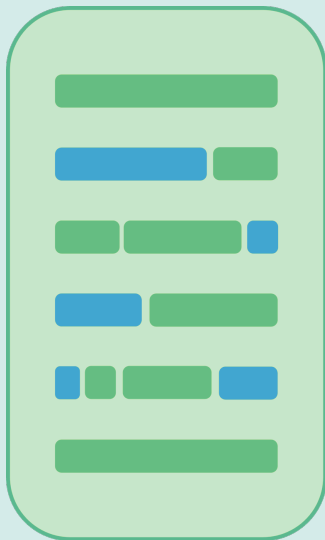
- Vollständig in **JavaScript** geschrieben, aber:
  1. **UI-Code** wird serverseitig vorgerendert
  2. **UI-Code** wird zum Browser gesendet und angezeigt
  3. Nur der JavaScript-Code ("Islands") **für Interaktionen** wird zum Browser geschickt

## Fullstack-Anwendungen



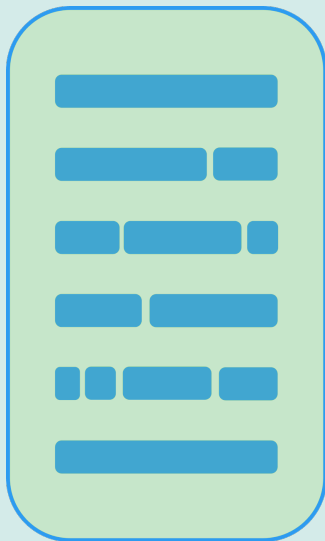
- Anwendung startet schneller:
  1. Browser bekommt **UI-Code** zur Darstellung
  2. Der **notwendige JavaScript-Code** wird nachgeladen
  3. Anwendung jetzt **interaktiv**

## Fullstack-Anwendungen



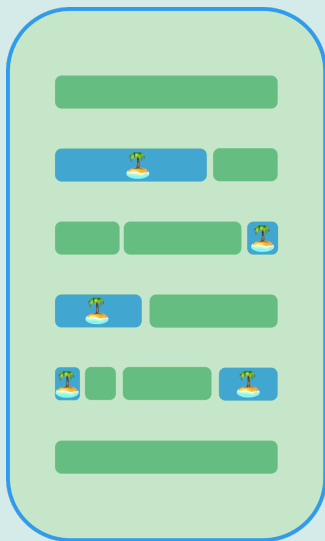
- Wir sind zurück zur **JavaScript-Schnipsel**-Architektur
  - aber: die Schnipsel werden automatisch vom Framework erzeugt
  - die Schnipsel existieren nur zur **Laufzeit**

## Fullstack-Anwendungen



- Wir sind zurück zur **JavaScript-Schnipsel**-Architektur
  - aber: die Schnipsel werden automatisch vom Framework erzeugt
  - die Schnipsel existieren nur zur Laufzeit
  - Bei der **Entwicklung** ist "unser" Code ist aus "einem Guß"

## Fullstack-Anwendungen

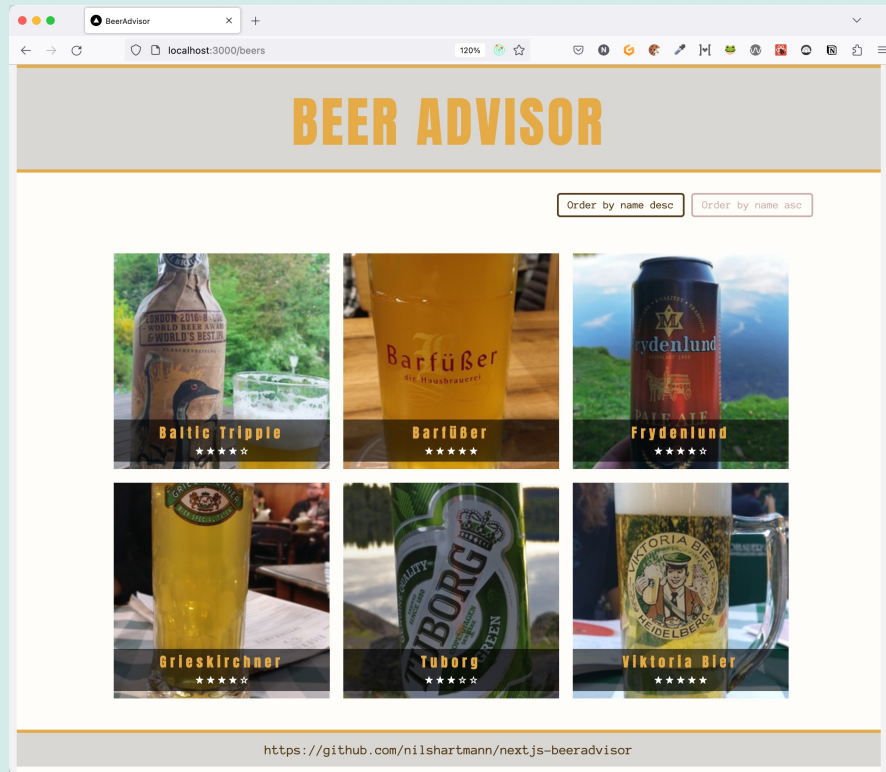


- Mehrere Ansätze und Frameworks
- Funktionalität und Herangehensweise unterschiedlich
  1. Qwik (eigenes Framework)
  2. Astro (eigenes Framework + Support für alle SPAs)
  3. SvelteKit (Svelte)
  4. Next.js (React)



# Fullstack- Anwendungen

mit React und Next.js



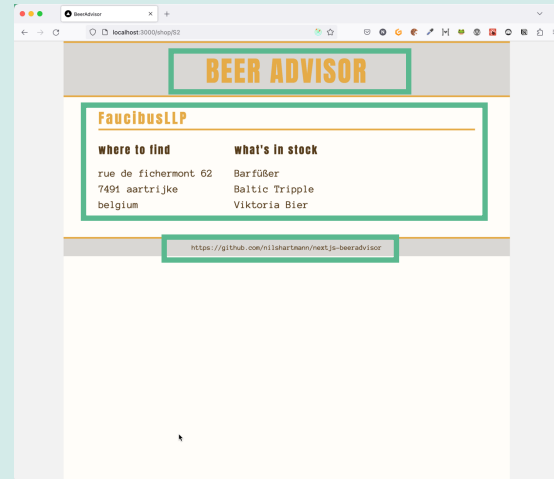
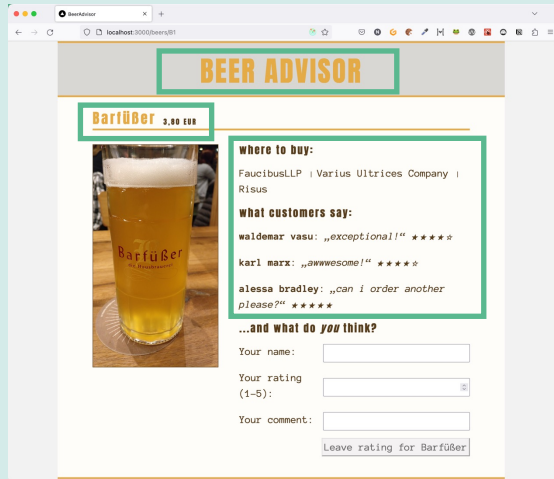
Beispiel-Code: <https://github.com/nilshartmann/nextjs-beeradvisor>

EIN BEISPIEL...

# EIN BEISPIEL

## Was macht die Beispiel-Anwendung aus?

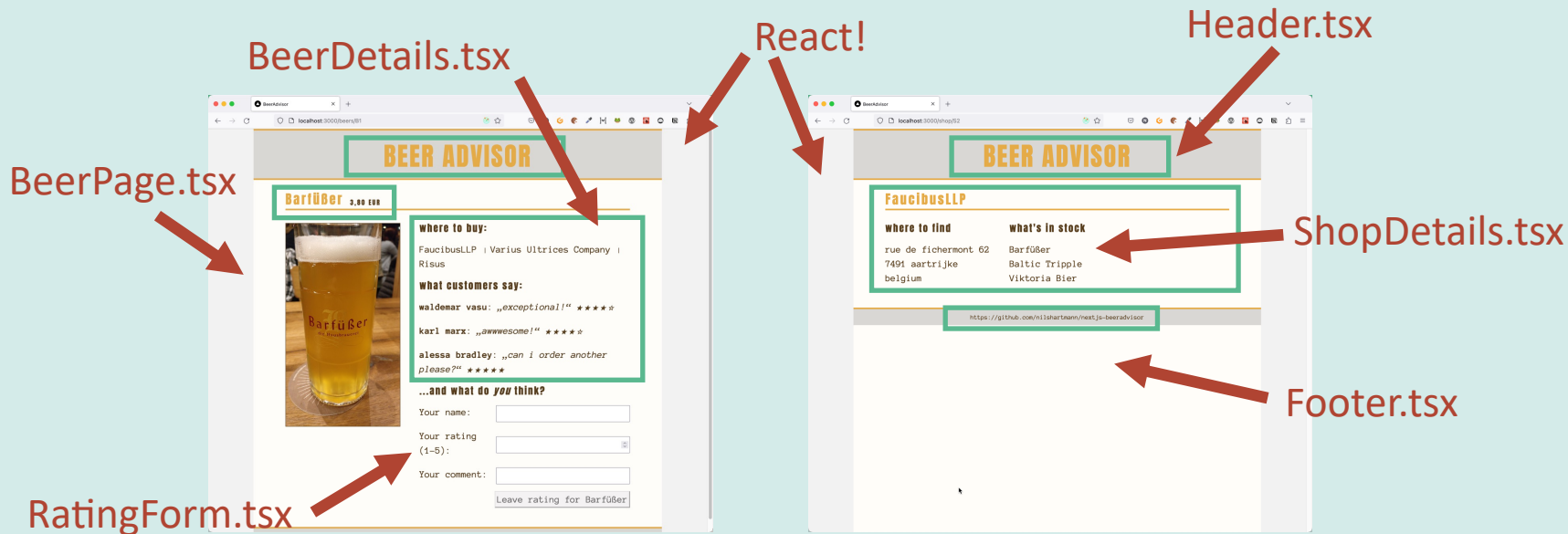
- Viel statischer Content 😊



# EIN BEISPIEL

## Was macht die Beispiel-Anwendung aus?

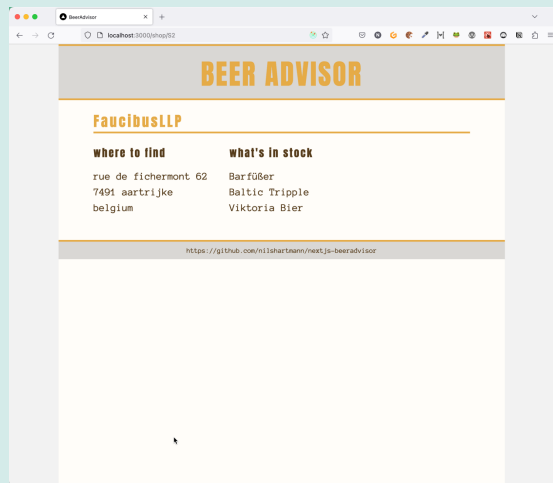
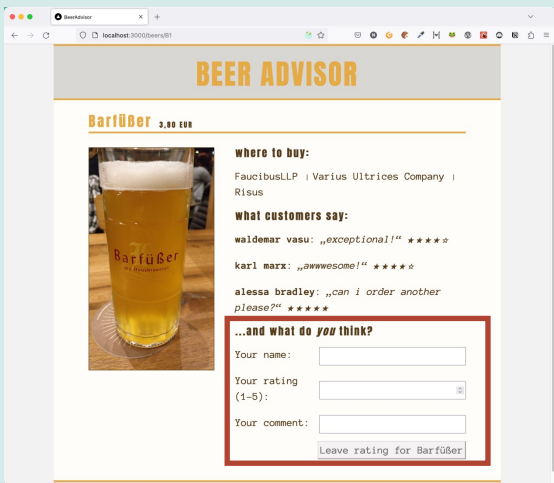
- Viel statischer Content 😊
- Viel JavaScript 😱



# EIN BEISPIEL

## Was macht die Beispiel-Anwendung aus?

- Viel statischer Content 😊
- Viel JavaScript 😱
- ...gleichzeitig wenig Interaktion 😞



## "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



### 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-Teams

Zero-Bundle-Size  
**Server**  
Components

## 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*

## Arten von Komponenten

# ARTEN VON KOMPONENTEN

## Client-Komponenten (wie bisher)

- Werden auf dem Client gerendert

## BEER ADVISOR

---

**Barfüßer** 3.80 EUR



**where to buy:**

FaucibusLLP | Varius Ultrices Company | Risus

**what customers say:**

waldemar vasu: „exceptional!“ ★★★★★

karl marx: „awesome!“ ★★★★★

alessa bradley: „can i order another please?“ ★★★★★

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

Your name:

Your rating (1-5):

Your comment:



# ARTEN VON KOMPONENTEN


## Client-Komponenten (wie bisher)

- Werden auf dem Client gerendert
- oder auf dem Server 🤨

## BEER ADVISOR

---

**Barfüßer** 3.80 EUR



**where to buy:**

FaucibusLLP | Varius Ultrices Company | Risus

**what customers say:**

waldemar vasu: „exceptional!“ ★★★★★

karl marx: „awesome!“ ★★★★★

alessa bradley: „can i order another please?“ ★★★★★

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

Your name:

Your rating (1-5):

Your comment:

# ARTEN VON KOMPONENTEN

## Client-Komponenten (wie bisher)

- Werden auf dem Client gerendert
- oder auf dem Server 🤔


Wie bisher:

- JavaScript-Code immer zum Client gesendet
- Können deshalb interaktiv sein

## 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 *you* think?**

Your name:

Your rating (1-5):

Your comment:

## Neu: Server-Komponenten

- werden auf dem Server gerendert

## Neu: Server-Komponenten

- werden auf dem Server gerendert
- oder im Build 🤔

### Neu: Server-Komponenten

- werden auf dem Server gerendert
- oder im Build 🤔
- liefern UI (!) zum React-Client zurück (kein JavaScript-Code)

# ARTEN VON KOMPONENTEN

Komponenten können gemischt werden

## BEER ADVISOR

### Barfüßer 3.80 EUR



**where to buy:**

FaucibusLLP | Varius Ultrices Company |  
Risus

**what customers**

waldemar: „exceptional!“ ★★★★★  
marx: „awwwesome!“ ★★★★★  
alessa bradley: „can i order another  
please?“ ★★★★★

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

Your name:

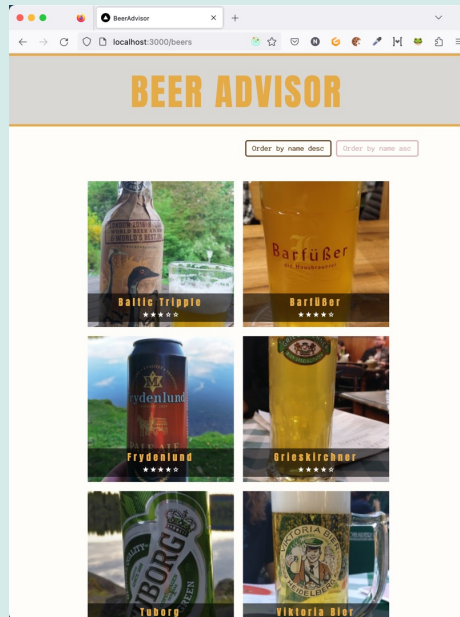
Your rating  
(1-5):

Your comment:

# RSC am Beispiel Next.js

## Schritt-für-Schritt

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





## Schritt 1: Eine Server Komponente



Demo: Landing Page

- `/page.tsx` zeigen
- `console.log` in Page-Komponente
  - auf dem Server
  - im Browser

# Data Fetching

### Idee

- Komponenten, die Daten laden, können das direkt *in der Komponente* tun

## SERVER COMPONENTS

### Idee

- Komponenten, die Daten laden, können das direkt *in der Komponente* tun
- Server Components können die Server-Infrastruktur nutzen (DB, Filesystem)

## SERVER COMPONENTS

### Idee

- Komponenten, die Daten laden, können das direkt *in der Komponente* tun
- Server Components können die Server-Infrastruktur nutzen (DB, Filesystem)

👉 Server-Komponenten können dazu 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: Komponente, die interaktiv ist



### Demo

- `beers/[beerId]` Beer-Page aus `material/beer-details-page.txt` einfügen
- Zeigen: hier haben wir auch eine Client-Komponente
  - die ist aber serverseitig vorgerendert

## Schritt 4: Eine asynchrone Server-Komponente, die träge ist



### Demo

- BeerDetailPage:
  - Aufruf künstlich verzögern (sleep in db-queries/loadBeer)
- loading.tsx



## Schritt 5: 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 um WhereToBuy

## Schritt 6: Eine interaktive Komponente, die Daten verändern will



### Demo

- **AddRatinForm**
  - `saveNewRating` zeigen und hinzufügen
  - `runSave` hinzufügen, um Antwort zu verarbeiten

**Aufteilung**

**in**

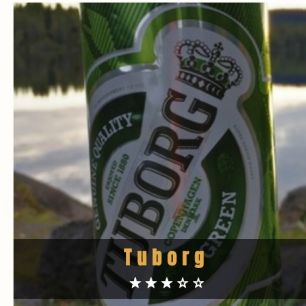
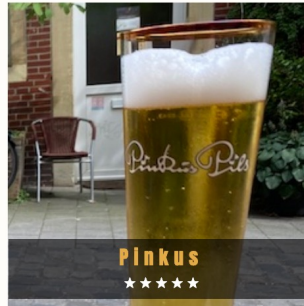
**Server-Client:**

**Konsequenzen**

# BEER ADVISOR

Order by name desc

Order by name asc



BEISPIEL: ÄNDERN DER SORTIERUNG

```
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.

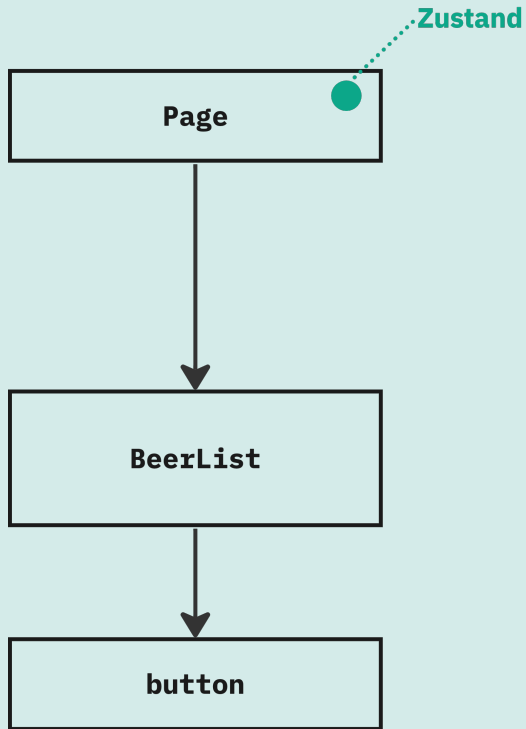
```
<button onClick={function} children=...>  
      ^^^^^^^^^^^
```

If you need interactivity, consider converting part of this to a Client Component.  
at stringify (<anonymous>)

**CAN YOU SPOT THE PROBLEM?**

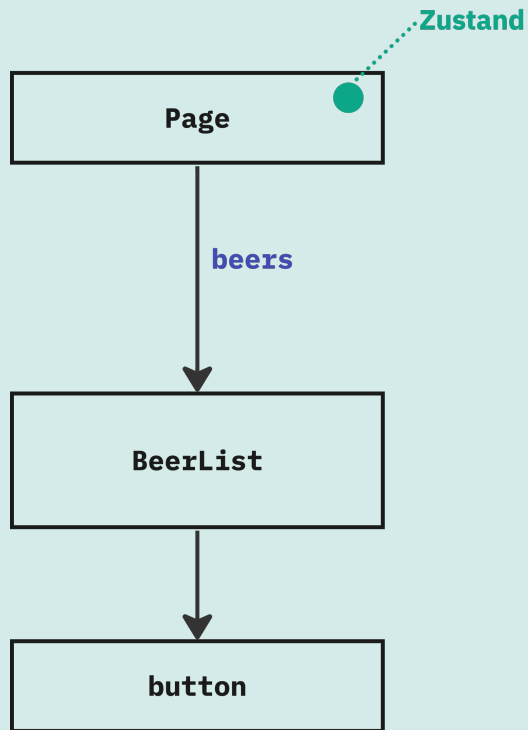
# EINE REACT ANWENDUNG IM BROWSER

- State befindet sich oben



Eine "normale" React-Anwendung...

# EINE REACT ANWENDUNG IM BROWSER

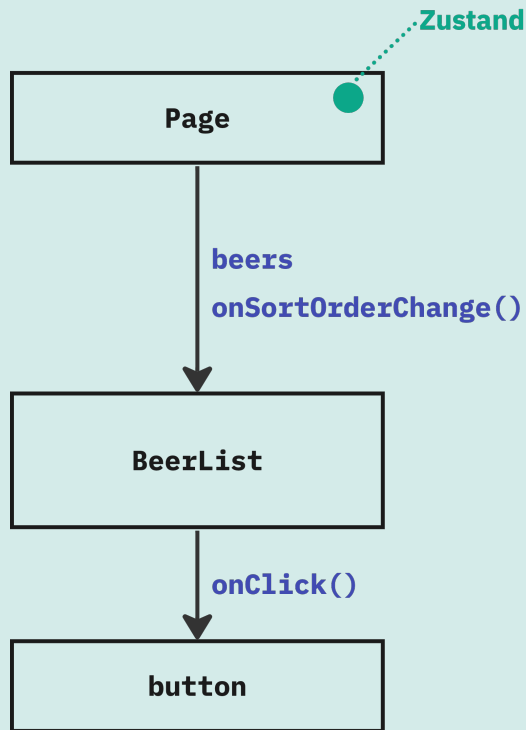


- State befindet sich oben
- Daten werden runtergereicht ("props")

Eine "normale" React-Anwendung...



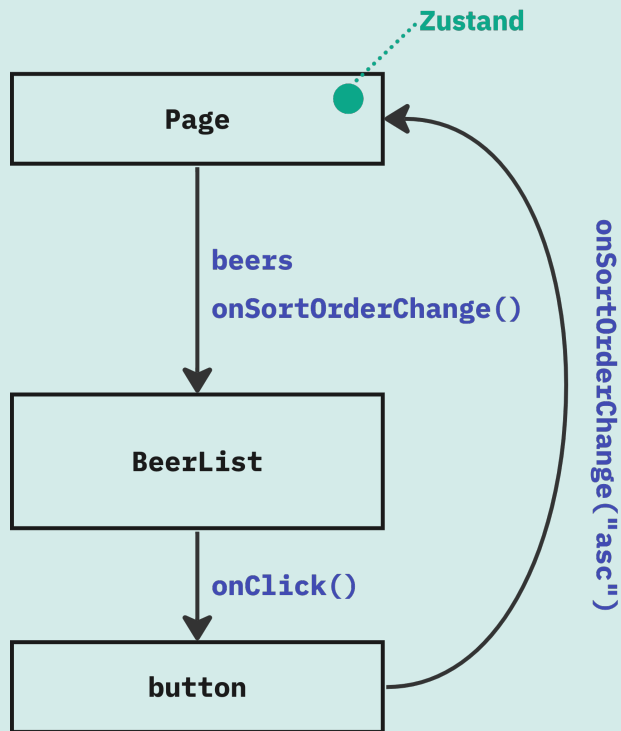
# EINE REACT ANWENDUNG IM BROWSER



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

Eine "normale" React-Anwendung...

# EINE REACT ANWENDUNG IM BROWSER



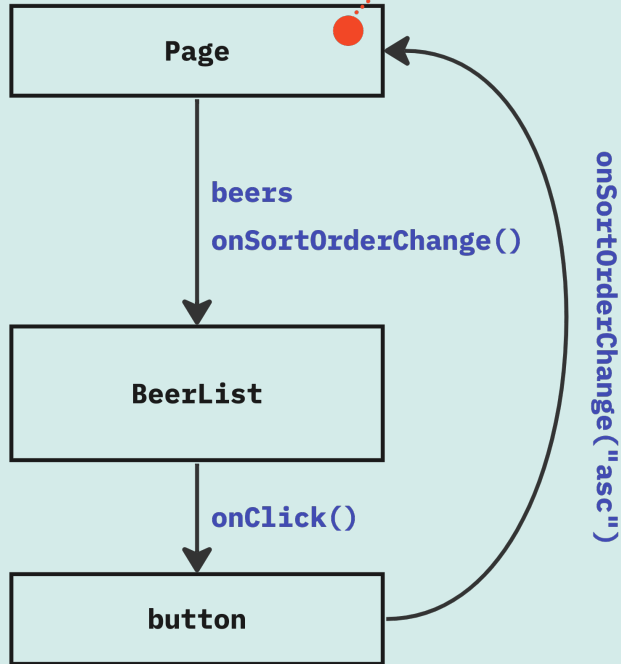
- State befindet sich oben
- Daten werden runtergereicht ("props")
- Callbacks werden runtergereicht
- Über Callbacks kann State-Veränderung ausgelöst werden

Eine "normale" React-Anwendung...

## ...UND AUF DEM SERVER

Kein Zustand auf dem Server 🤔

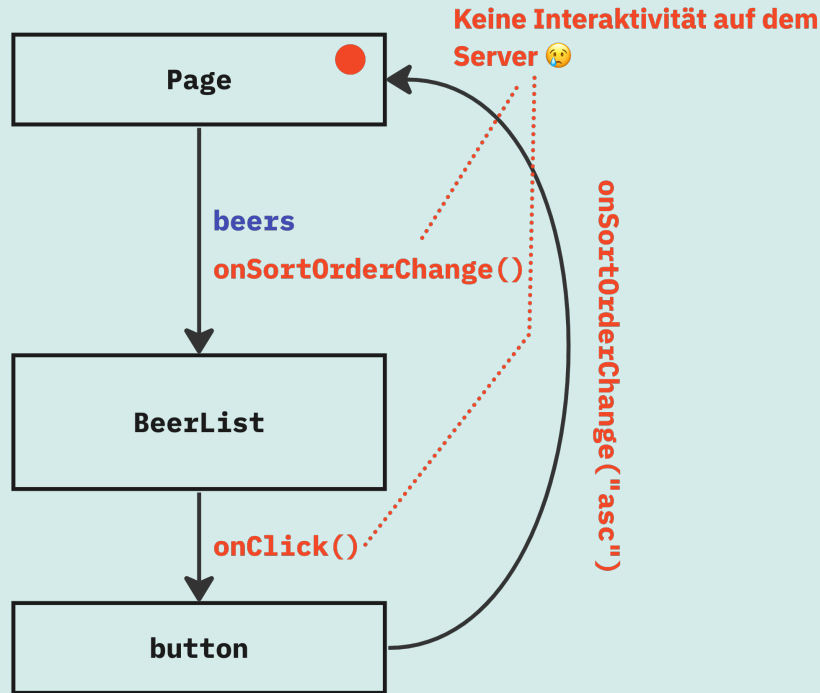
- Auf dem Server gibt es keinen State!



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

## ...UND AUF DEM SERVER

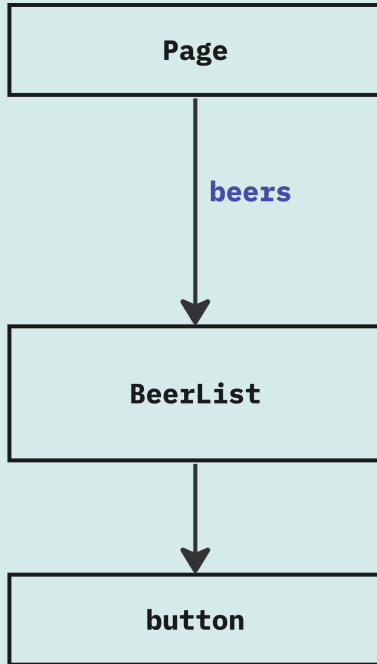
- Auf dem Server gibt es keinen State!
- ...und keine Interaktion



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

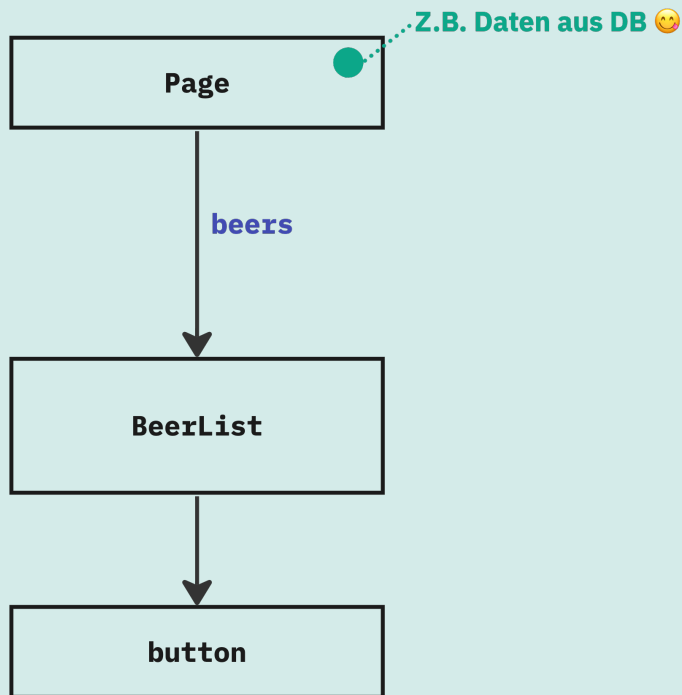
## ...UND AUF DEM SERVER

- 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

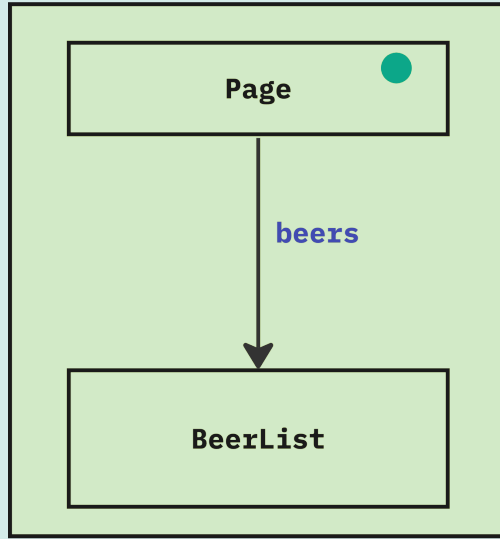


- Auf dem Server gibt es keinen State!
- ...und keine Interaktion
- Wir haben nur statischen Content
- Wir haben aber **Daten**  
z.B. aus DB, Microservice, Filesystem...

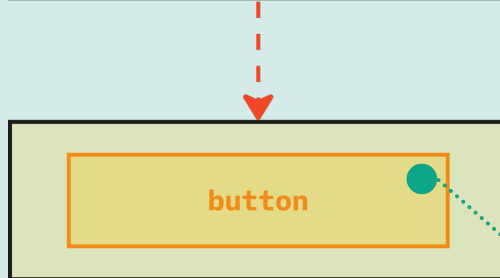
Mit Next.js sind wir aber auf dem Server (by Default)

## ...UND AUF DEM SERVER

Server



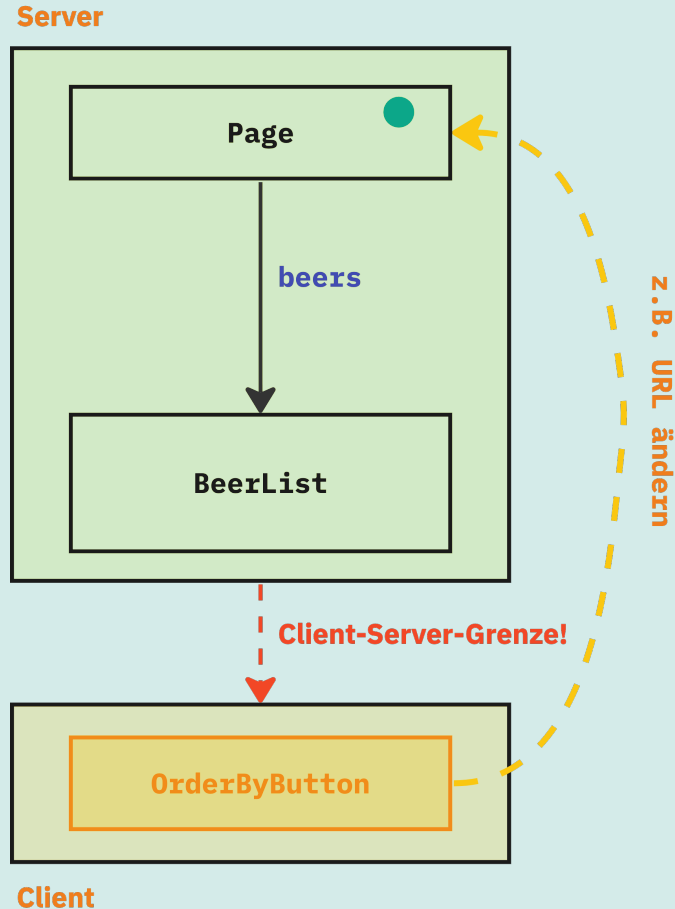
- Bestimmte Teile **müssen** auf den Client
  - z.B. Event-Handler



Client

Interaktives muss auf den Client 🤖

## ...UND AUF DEM SERVER

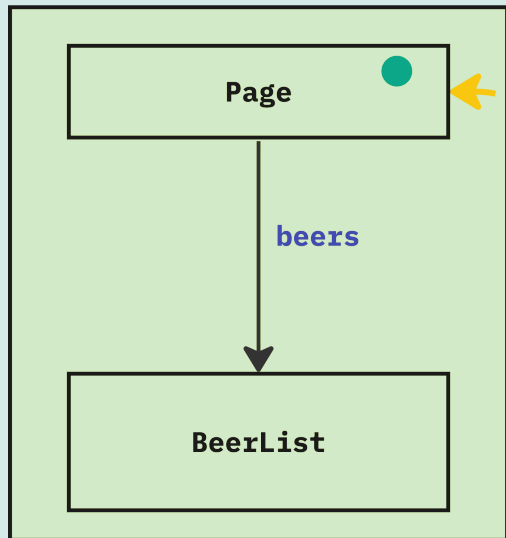


- Properties müssen Client-Server-Grenze überwinden
- Müssen serialisierbare Daten sein
- Keine (Callback-)Funktionen!
- Zur Kommunikation müssen **Server-Requests** durchgeführt werden
  - z.B. URL ändern

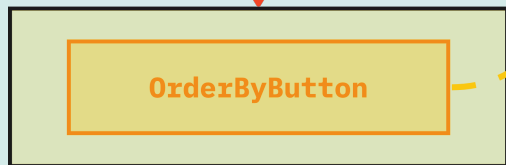


## ...UND AUF DEM SERVER

Server



Client-Server-Grenze!



Client

z.B. URL ändern

```
"use client";
```

```
function OrderByButton({orderBy}) {
```

```
  const updateUrl = () => { ... }
```

```
  return (
```

```
    <Button
```

```
      onClick={updateUrl}
```

```
    >
```

```
      Order by name {orderBy}
```

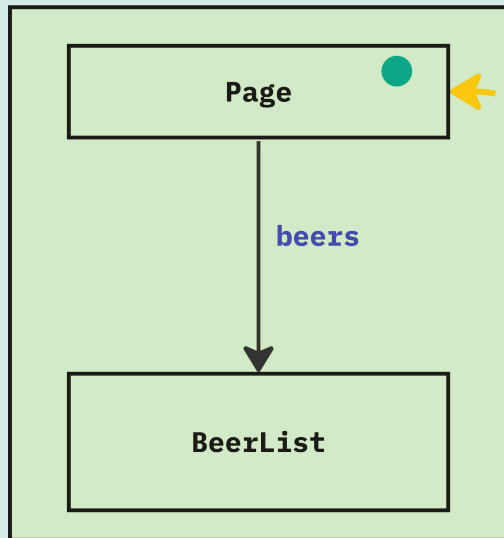
```
    </Button>
```

```
  )
```

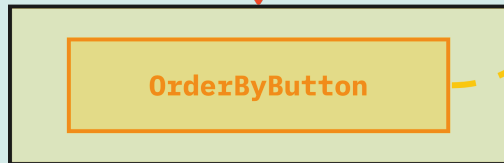
```
}
```

## ...UND AUF DEM SERVER

Server



Client-Server-Grenze!

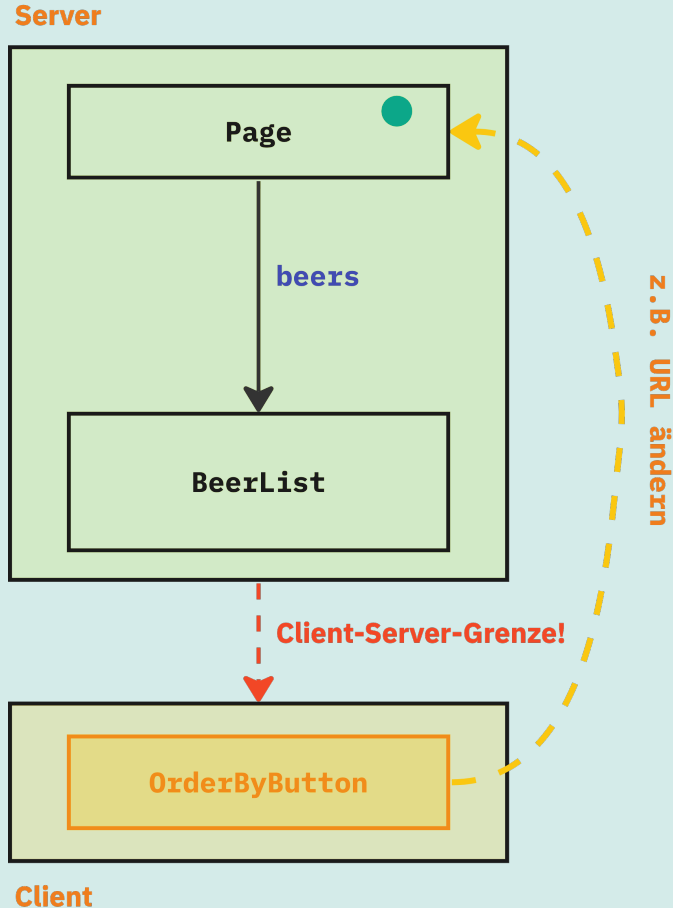


Client

z.B. URL ändern

```
export function BeerListPage() {  
  
  const beers = loadBeers();  
  // ...  
  
  return (  
    <>  
    <OrderByButton orderBy="asc" />  
    <OrderByButton orderBy="desc" />  
  
    <BeerList beers={beers} />  
    </>  
  );  
}
```

## ...UND AUF DEM 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-)API-Call bedeuten!

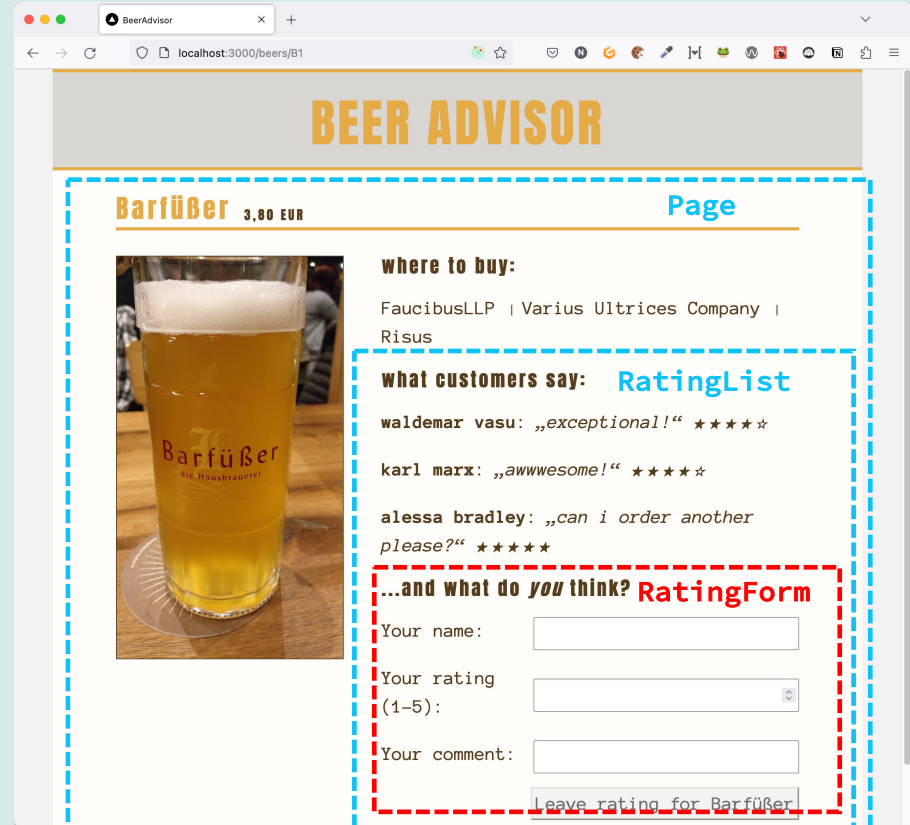
# Mutations

# MUTATIONS

## Verändern von Daten: Hinzufügen einer Bewertung

Server-Komponente

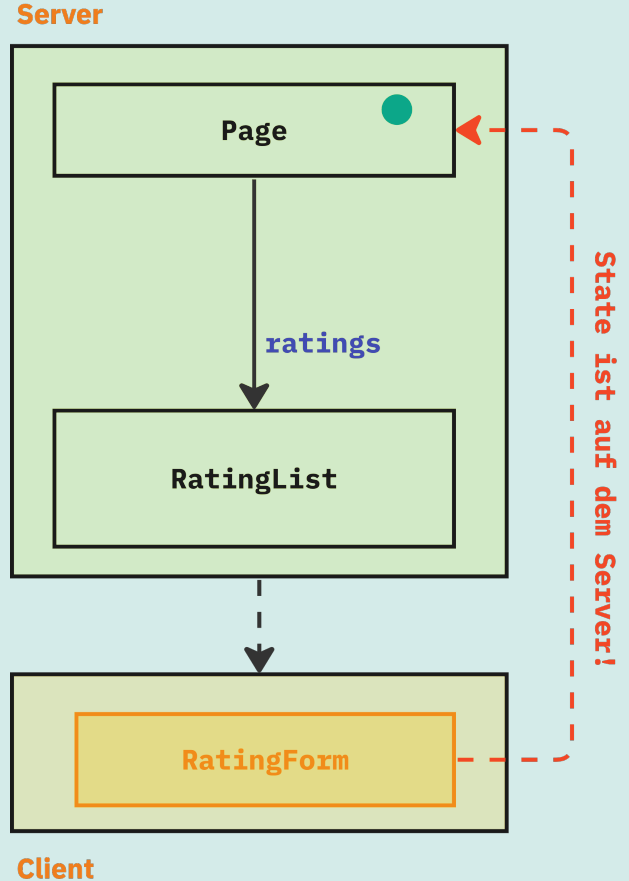
Client-Komponente



# MUTATIONS

## Verändern von Daten

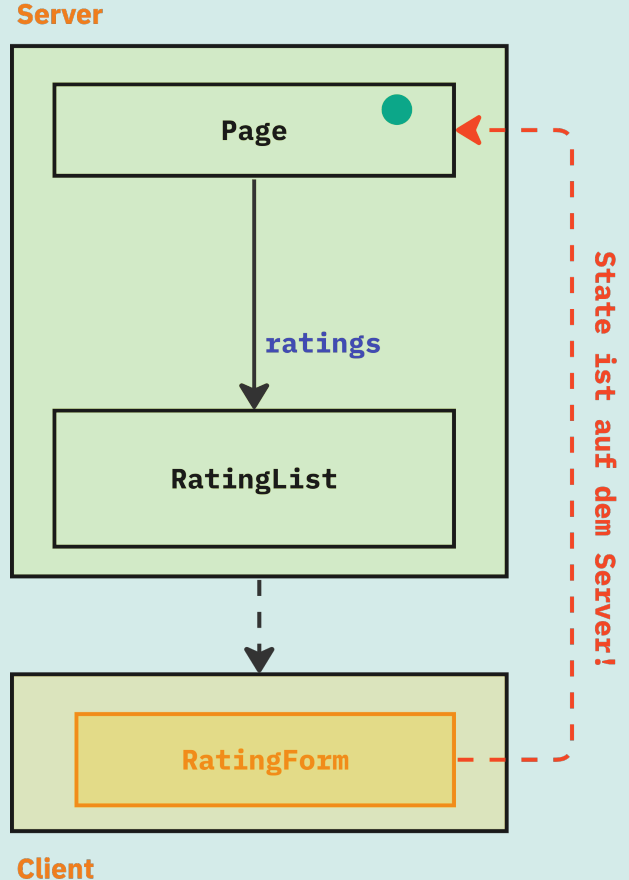
- Nach dem Verändern von Daten muss die UI aktualisiert werden
- Mangels State auf dem Client geht das aber nicht wie bislang



# 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



## MUTATIONS

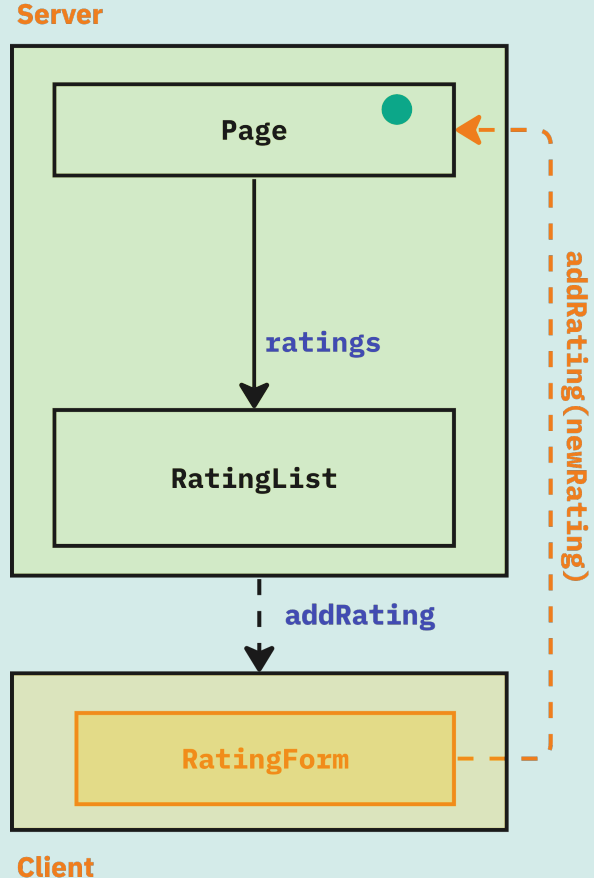
### Server Actions



# MUTATIONS

## Server Actions

- Neues "canary" React-Feature
- Remote Funktion, die aus einer Komponente aufgerufen werden kann



## Server Actions

### Demo

- AddRatingForm in BeerDetailPage einfügen
- AddRatingForm:
  - - action `saveNewRating` hinzufügen
- Netzwerkverkehr: Antwort vom Server, keine UI
- in `saveNewRating` `invalidateRoute` machen
- **?** Menge an Daten?

**NILS HARTMANN**

<https://nilshartmann.net>

<https://reactbuch.de>



# Vielen Dank!

Slides: <https://react.schule/gedoplan-2024>

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

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

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