NILS HARTMANN

https://nilshartmann.net

Slides: https://react.schule/oose24

React

Abschied von der Single-Page-Anwendung?

NILS HARTMANN

nils@nilshartmann.net

Freiberuflicher Entwickler, Architekt, Trainer aus Hamburg

Java, Spring, GraphQL, React, TypeScript

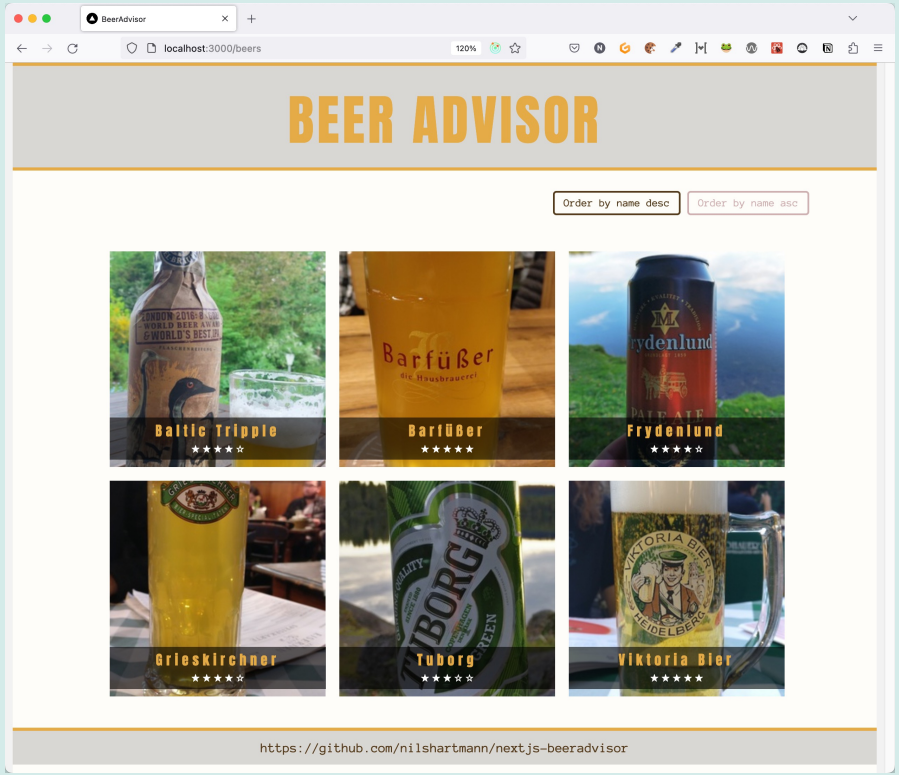
https://graphql.schule/video-kurs

https://reactbuch.de

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/



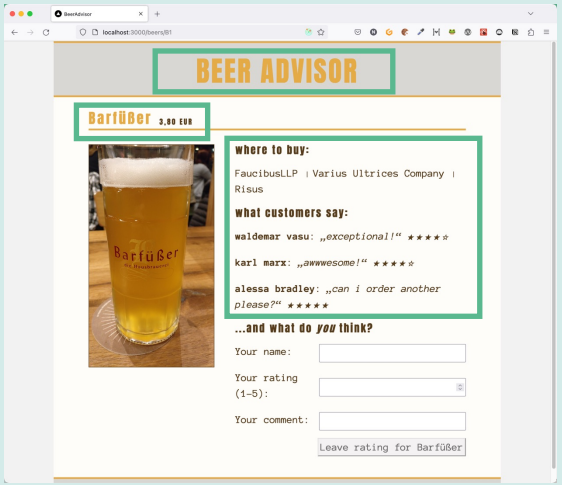
Beispiel-Code: https://github.com/nilshartmann/nextjs-beeradvisor

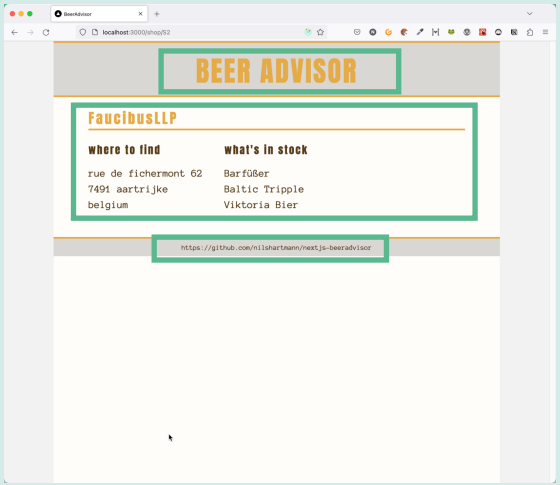
EIN BEISPIEL...

EIN BEISPIEL

Was macht die Beispiel-Anwendung aus?

- Viel Bier 😋

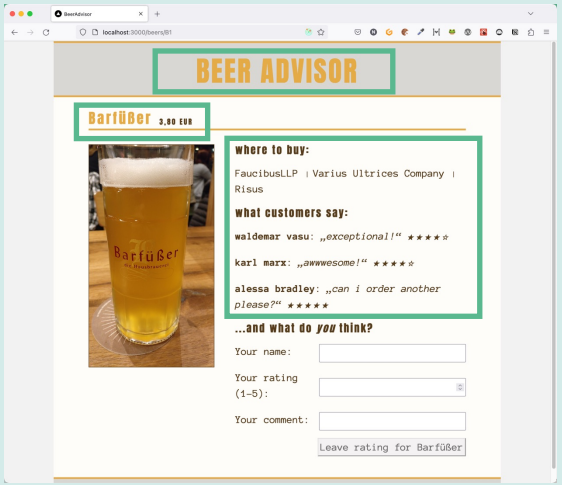


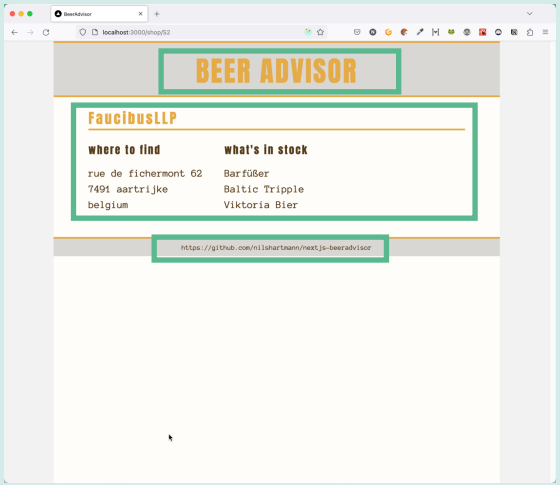


EIN BEISPIEL

Was macht die Beispiel-Anwendung aus?

- Viel statischer Content ☺️

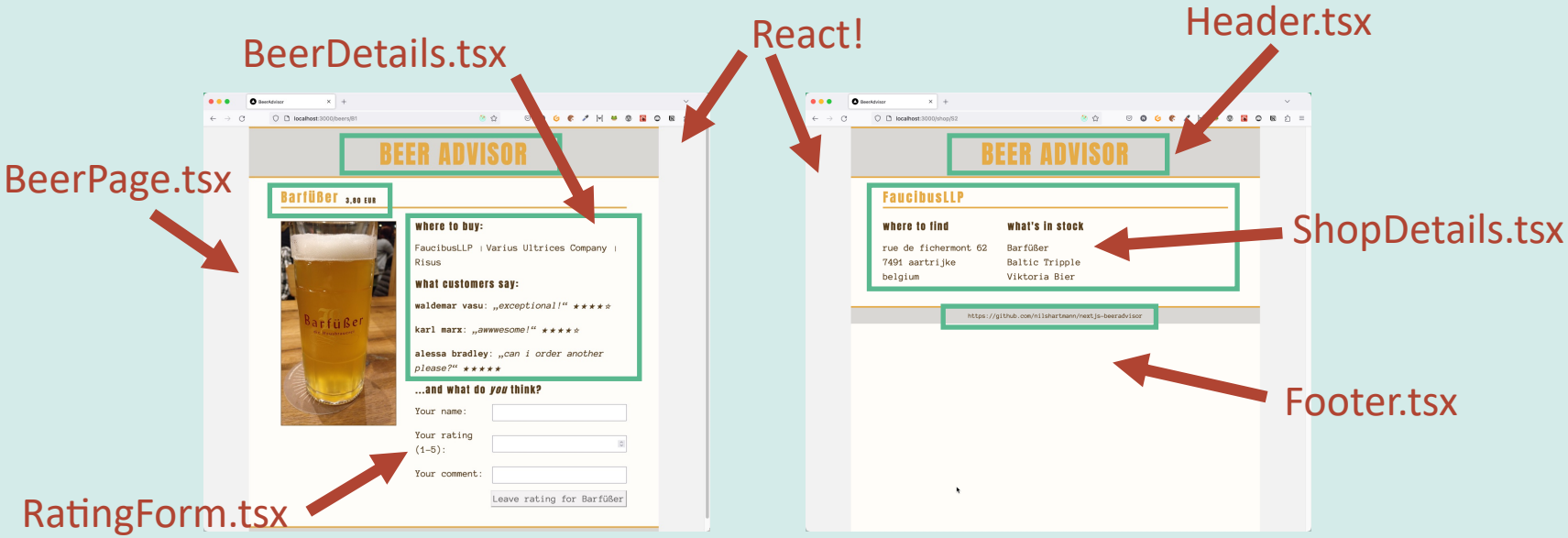




EIN BEISPIEL

Was macht die Beispiel-Anwendung aus?

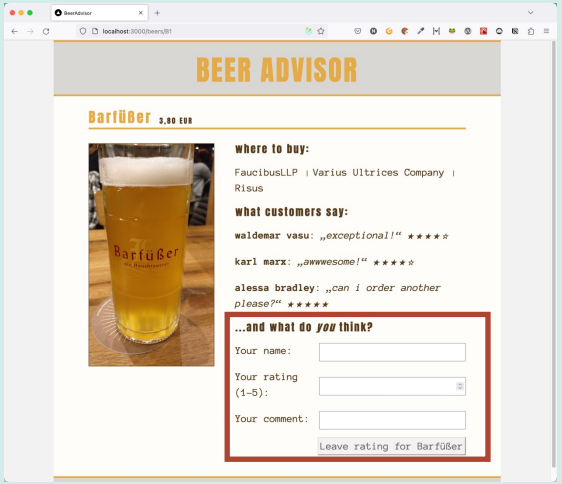
- Viel statischer Content ☺️
- Viel JavaScript 😱

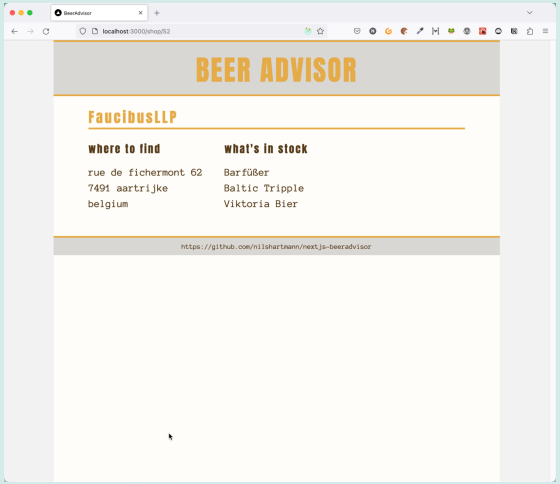


EIN BEISPIEL

Was macht die Beispiel-Anwendung aus?

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





Anforderung

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

Mögliche Probleme

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

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

Demo

- Beispiel-Anwendung ist serverseitig vorgerendert!

Serverseitiges Rendern (SSR)

1. Bei SSR wird die Anwendung auf dem Server ausgeführt

2. Der Server schickt **fertiges HTML** zum Client
   - Gut: Client braucht HTML nur anzuzeigen (schnell!)
   - Gut: Suchmaschinen können HTML indizieren

Serverseitiges Rendern (SSR)

1. Bei SSR wird die Anwendung auf dem Server ausgeführt

2. Der Server schickt **fertiges HTML** zum Client
   - Gut: Client braucht HTML nur anzuzeigen (schnell!)
   - Gut: Suchmaschinen können HTML indizieren

3. Ebenfalls wird der **komplette Anwendungscode** zum Client geschickt
   - 😢 Auch für "statische" Komponenten
   - 😢 Bandbreite! Performance!

SSR wird von React schon lange unterstützt

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

SSR wird von React schon lange unterstützt

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

- In der Praxis ist das nicht trivial
  - Anbindung an Server
  - Ausführen / warten auf asynchronen Code
  - Build-Prozess für Server- und Client-Code inklusive Bundling
  - Debugging / Testen
  - Sicherstellen, dass Code auf Server + Client funktioniert

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 das Gefühl einer langsamen 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

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

React empfiehlt "Fullstack-Framework"

- **Next.js** entspricht den Vorstellungen des React-Teams

- **Remix** unterstützt noch keine RSC, hat aber ähnliche Features

Zero-Bundle-Size

Server Components

SERVER COMPONENTS

Idee: Komponenten werden nicht im Client ausgeführt

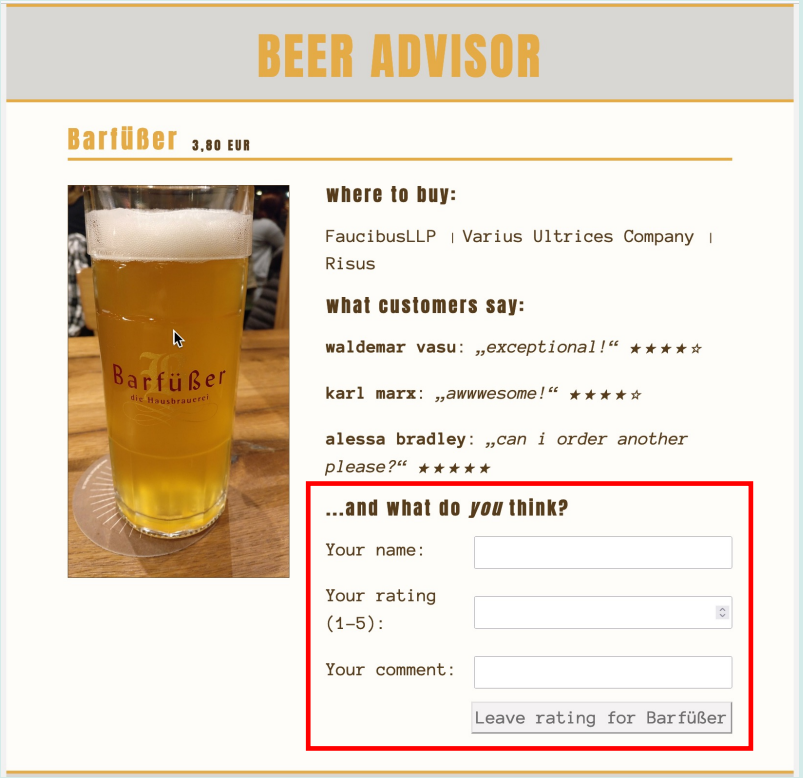
- Sie stehen auf dem Client nur fertig gerendert zur Verfügung
- Der Server schickt lediglich eine *Repräsentation der UI,* aber *keinen Code*

Arten von Komponenten

ARTEN VON KOMPONENTEN

Client-Komponenten (wie bisher)

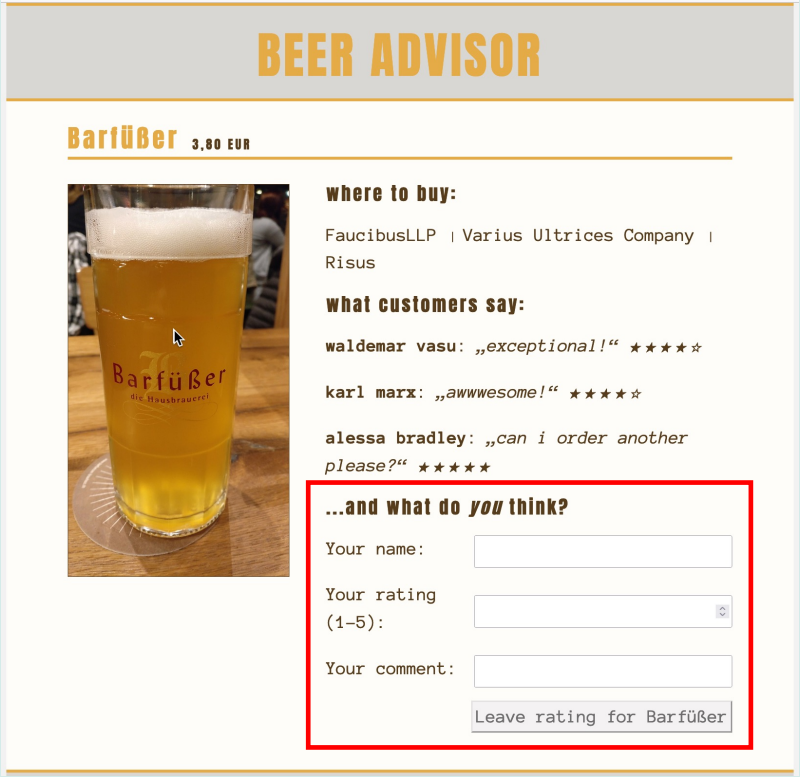
- Werden auf dem Client gerendert



ARTEN VON KOMPONENTEN

Client-Komponenten (wie bisher)

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



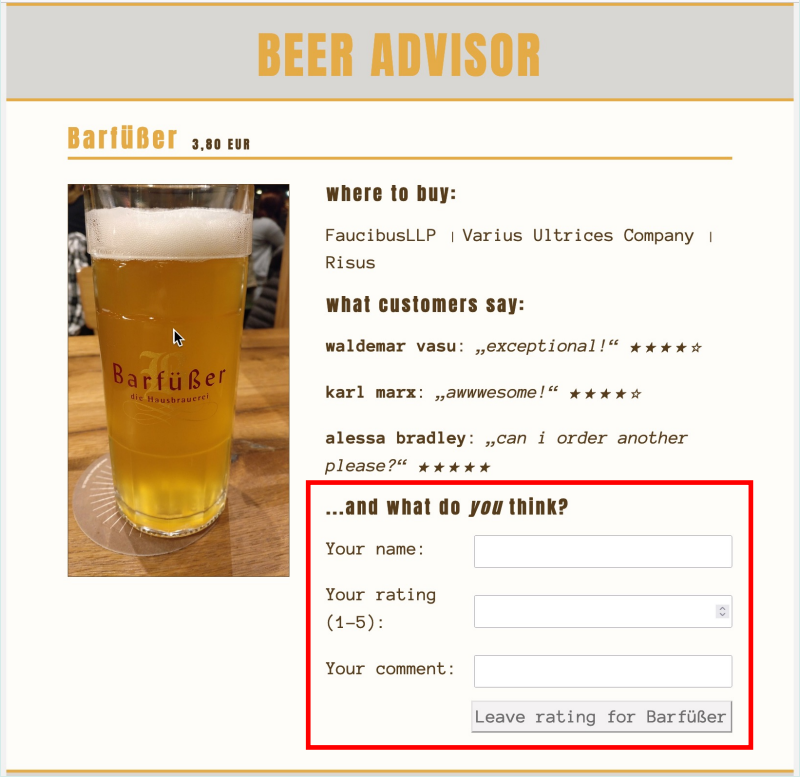
ARTEN VON KOMPONENTEN

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



Neu: Server-Komponenten

- werden auf dem Server gerendert

ARTEN VON KOMPONENTEN

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

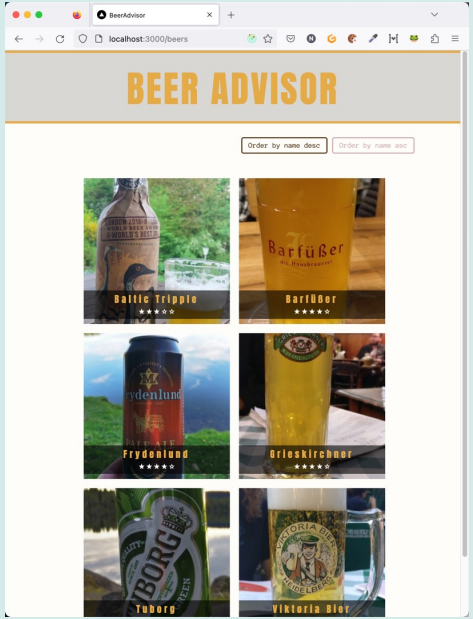
Komponenten können gemischt werden

RSC am Beispiel

Next.js

Schritt-für-Schritt

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



Schritt 1: Eine Server Komponente

🕵️ Demo

- "LandingPage" /page.tsx anlegen
- console.log in Page-Komponente
  - auf dem Server
  - im Browser

Data Fetching

Idee

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

Idee

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

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: Eine asynchrone Server-Komponente, die träge ist

Demo

- **beers/[beerId] Beer-Page aus material/beer-details-page.txt einfügen**
- **Aufruf künstlich verzögern (sleep in loadBeer)**
- **loading.tsx**

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

👨‍🚒 Demo

- **beers/[beerId] Beer-Page wieder schnell machen (slow entfernen)**
- **BeerDetails-Komponente erweitern s. Todos)**
- **Zeigen: Promise an Unterkomponente (Shops)**
  - **-> Parallel fetching!**
- **Aufruf künstlich verzögern (slow=2400)**
- 😔 **Jetzt wartet die ganze Seite auf die Shops...**
- **Suspense um WhereToBuy**

Aufteilung

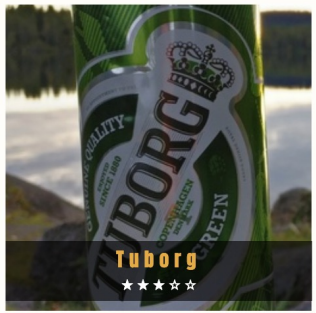
in

Server-Client:

Konsequenzen

BEER ADVISOR

Order by name desc | Order by name asc



BEISPIEL: ÄNDERN DER SORTIERUNG

```tsx
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?

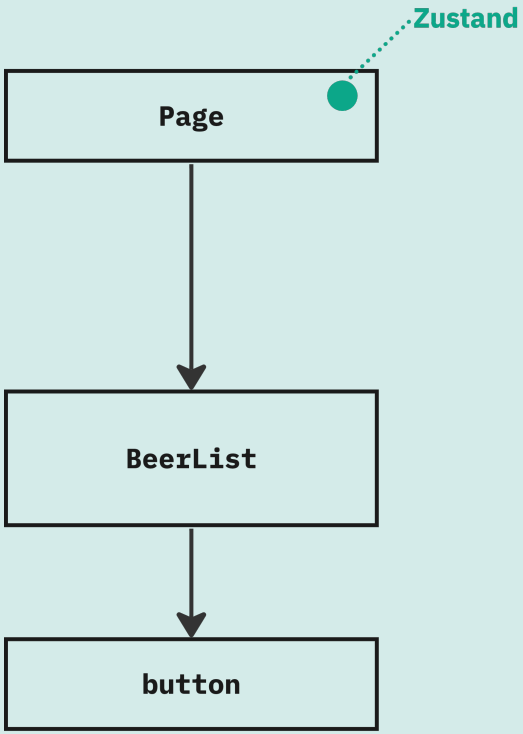
```
<button onClick={onToggleOrder}>Toggle Order</button>
```

```
- 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?

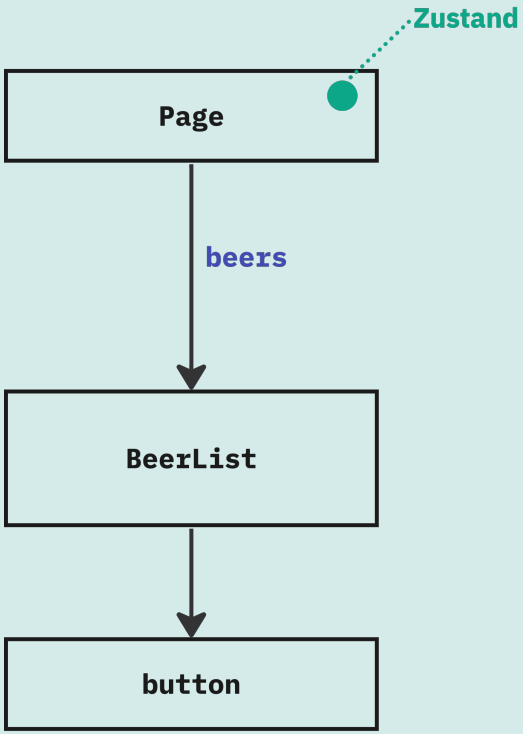
EINE REACT ANWENDUNG IM BROWSER



- State befindet sich oben

Eine "normale" React-Anwendung...

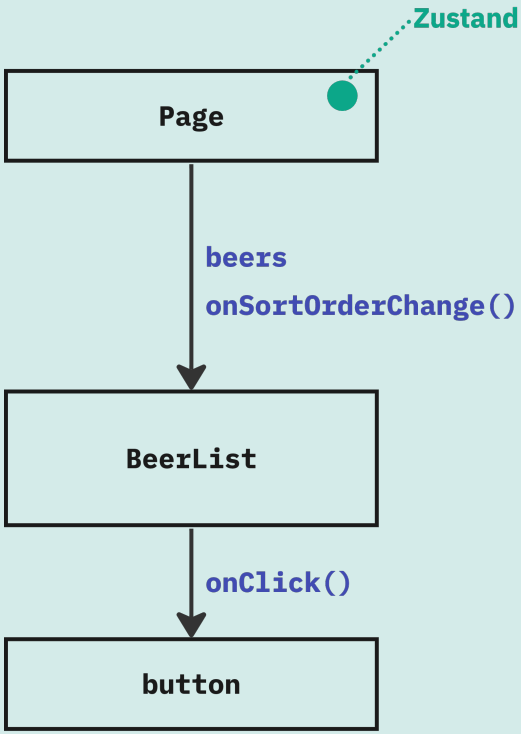
EINE REACT ANWENDUNG IM BROWSER



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

Eine "normale" React-Anwendung...

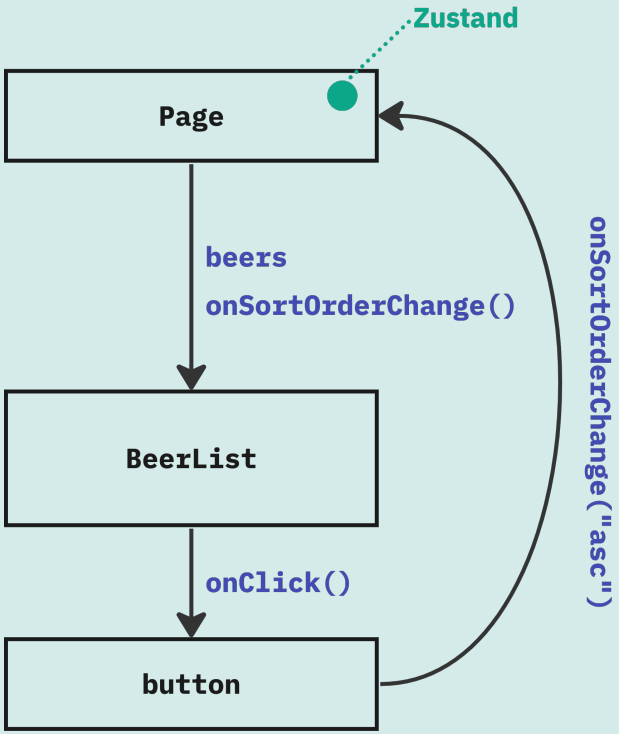
EINE REACT ANWENDUNG IM BROWSER



Eine "normale" React-Anwendung...

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

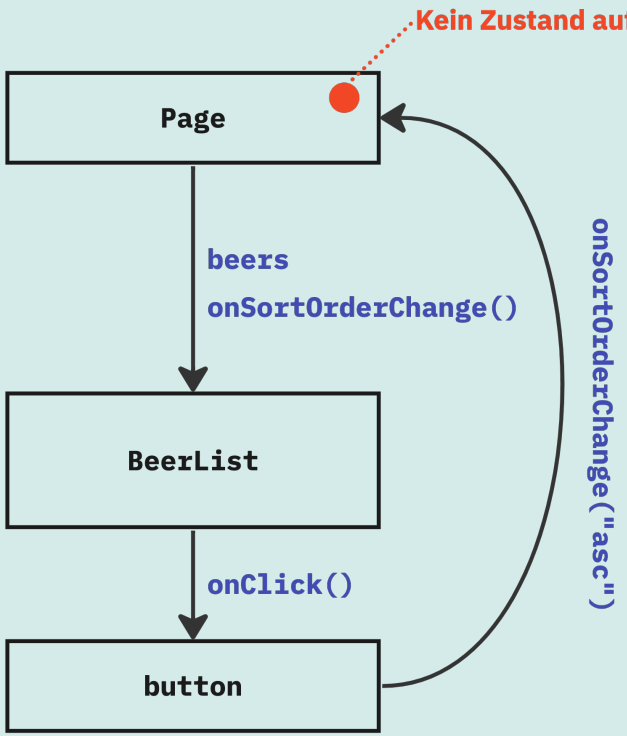
EINE REACT ANWENDUNG IM BROWSER



Eine "normale" React-Anwendung...

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

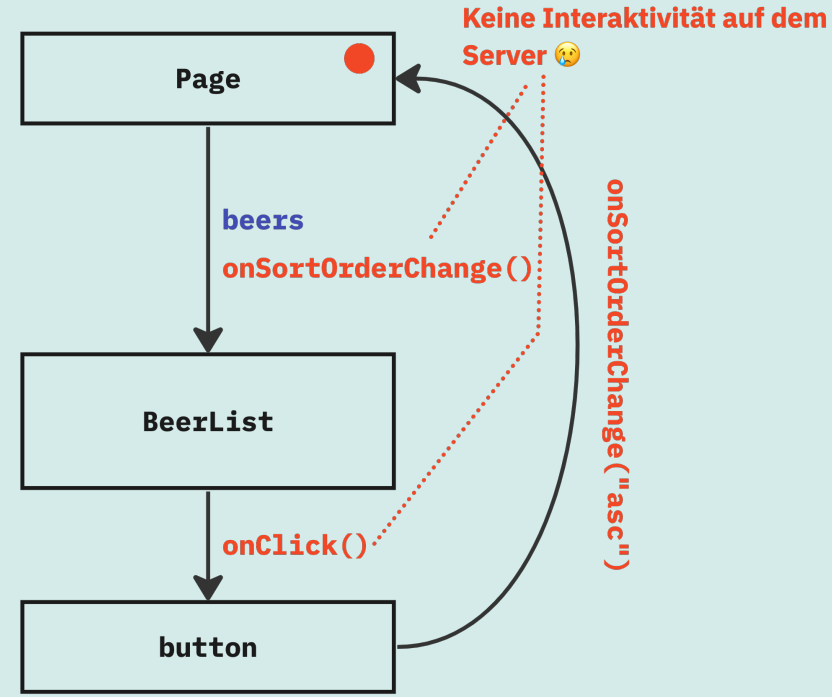
...UND AUF DEM SERVER



- Auf dem Server gibt es keinen State!

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

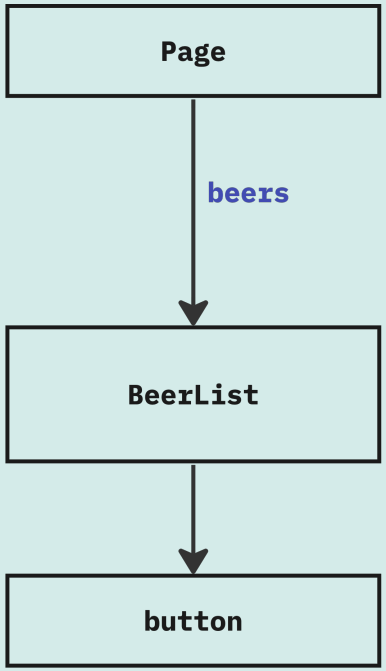
...UND AUF DEM SERVER



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

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

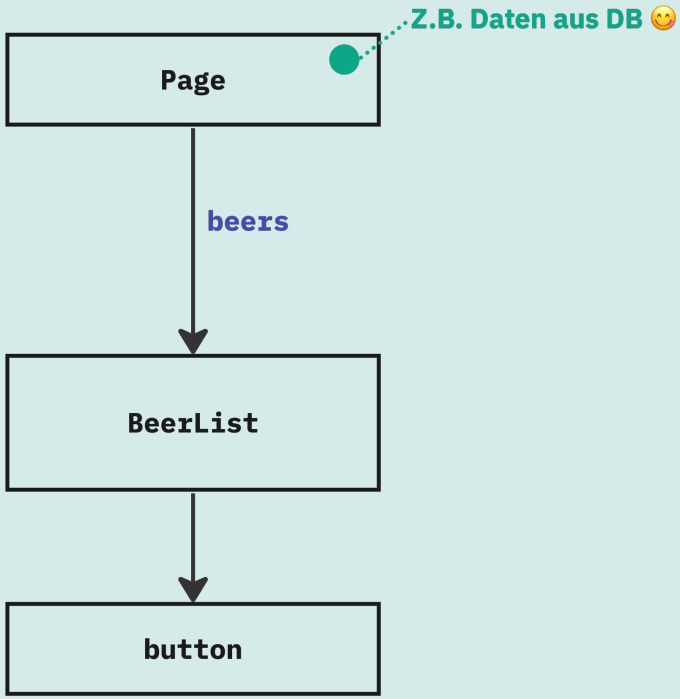
...UND AUF DEM SERVER



- 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

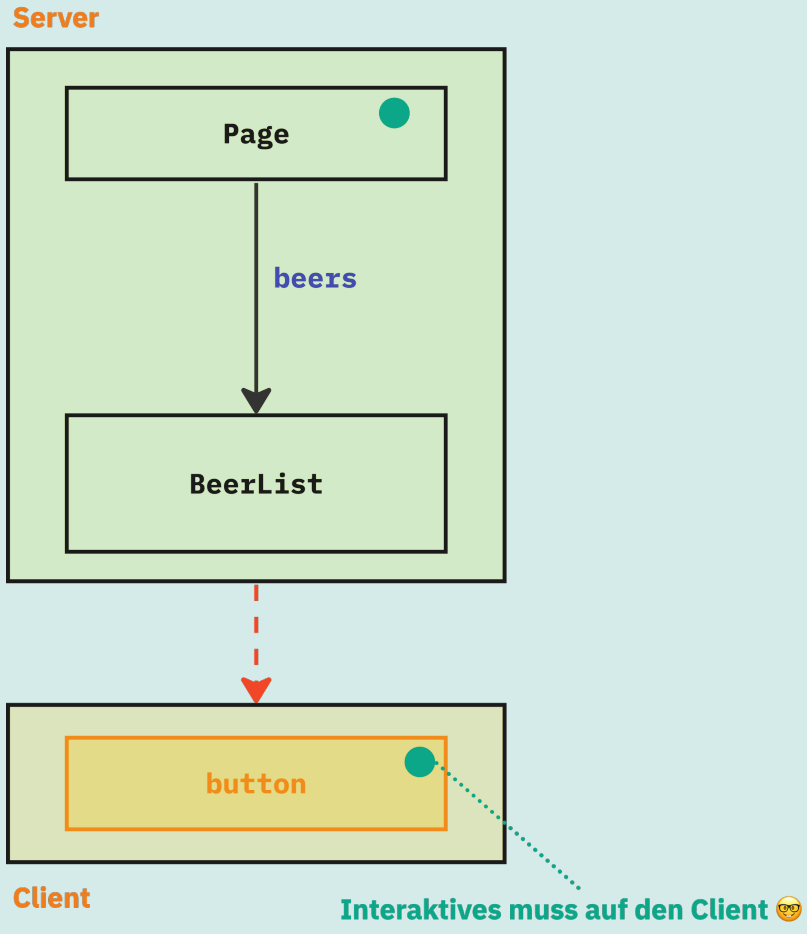


- Auf dem Server gibt es keinen State!
- ...und keine Interaktion
- Wir haben nur statischen Content
- Wir haben aber **Daten**

  z.B. aus DB, Microservice, Filesystem...

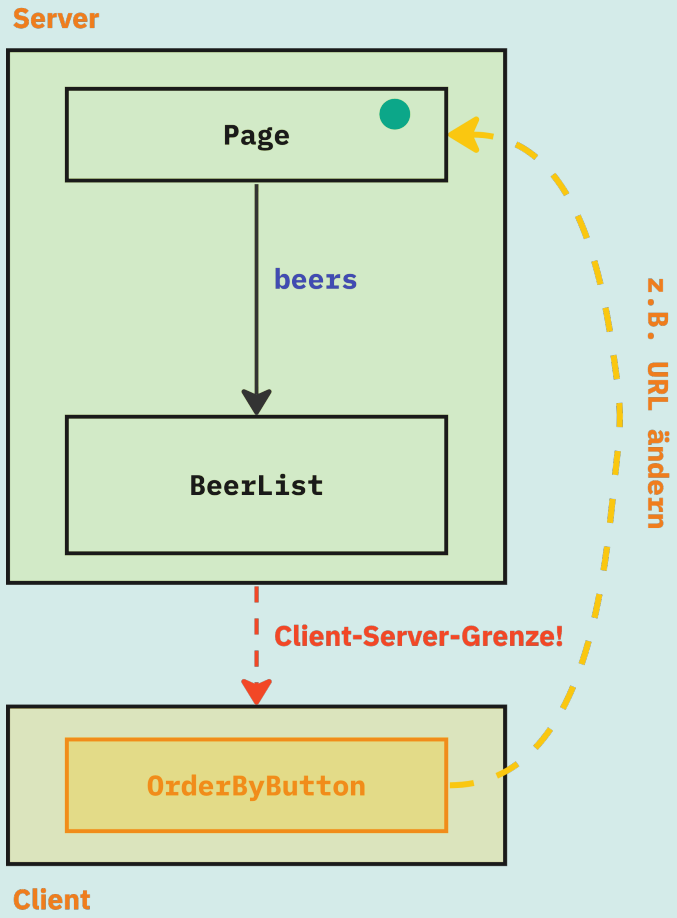
Mit Next.js sind wir aber auf dem Server (by Default)

...UND AUF DEM SERVER



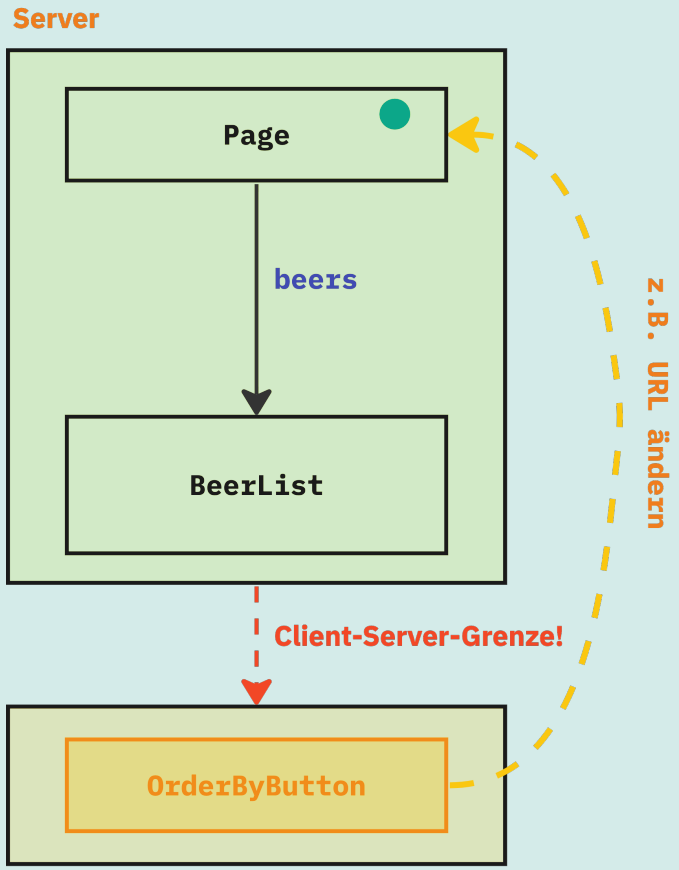
- Bestimmte Teile **müssen** auf den Client
  - z.B. Event-Handler

...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



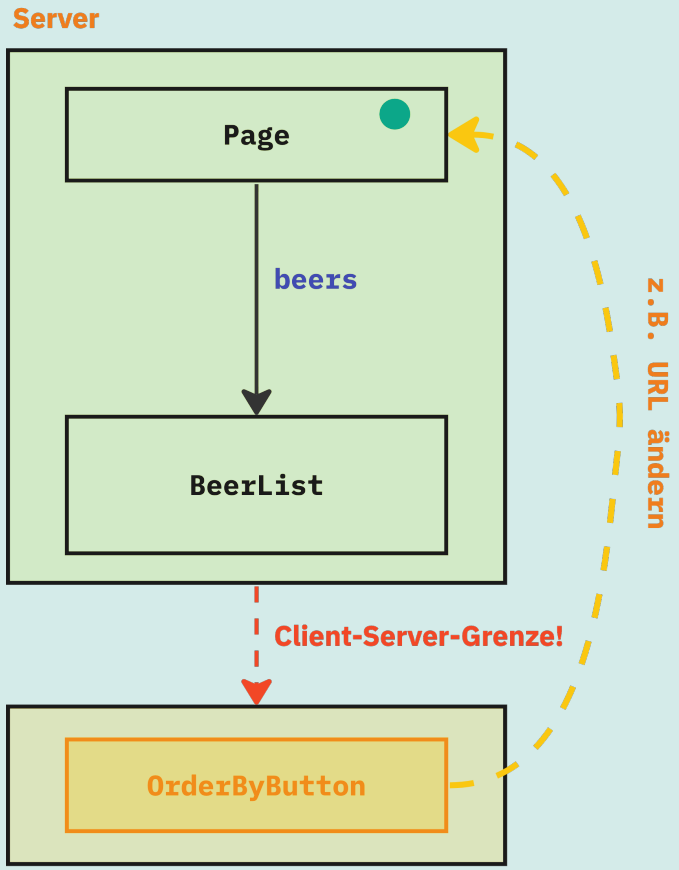

```
"use client";

function OrderByButton({orderBy}) {

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

  return (
    <Button
     onClick={updateUrl}
    >
      Order by name {orderBy}
    </Button>
  )
}
```

...UND AUF DEM SERVER



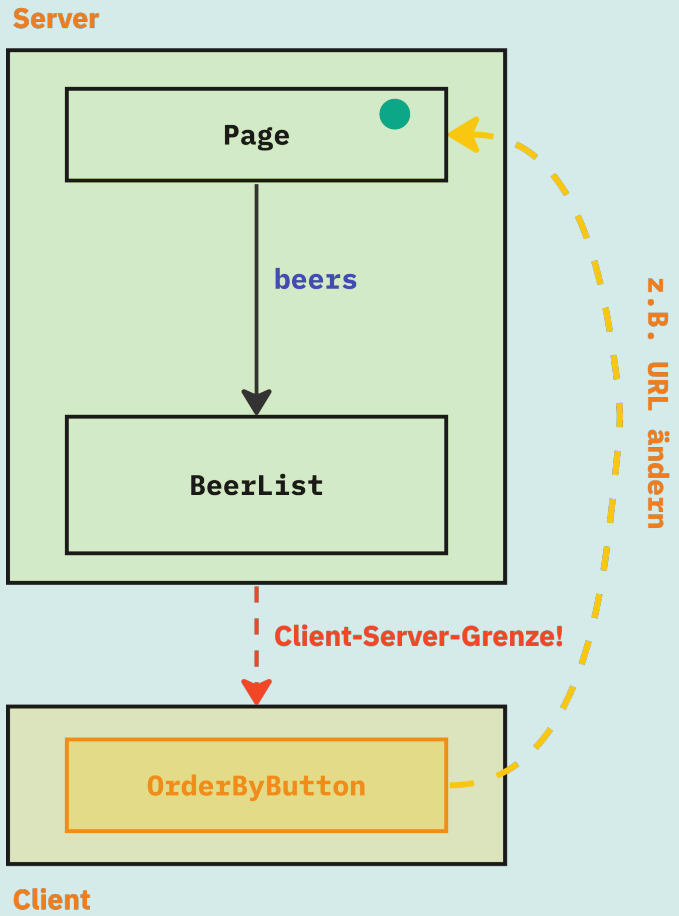

```
export function BeerListPage() {

  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!

Schritt 5: Eine Client-Komponente

🕵️ Demo

- button mit onClick in BeerListPage einbauen

- **In BeerListPage einbauen**
  - **was passiert?**

- **"use client"-Direktive**

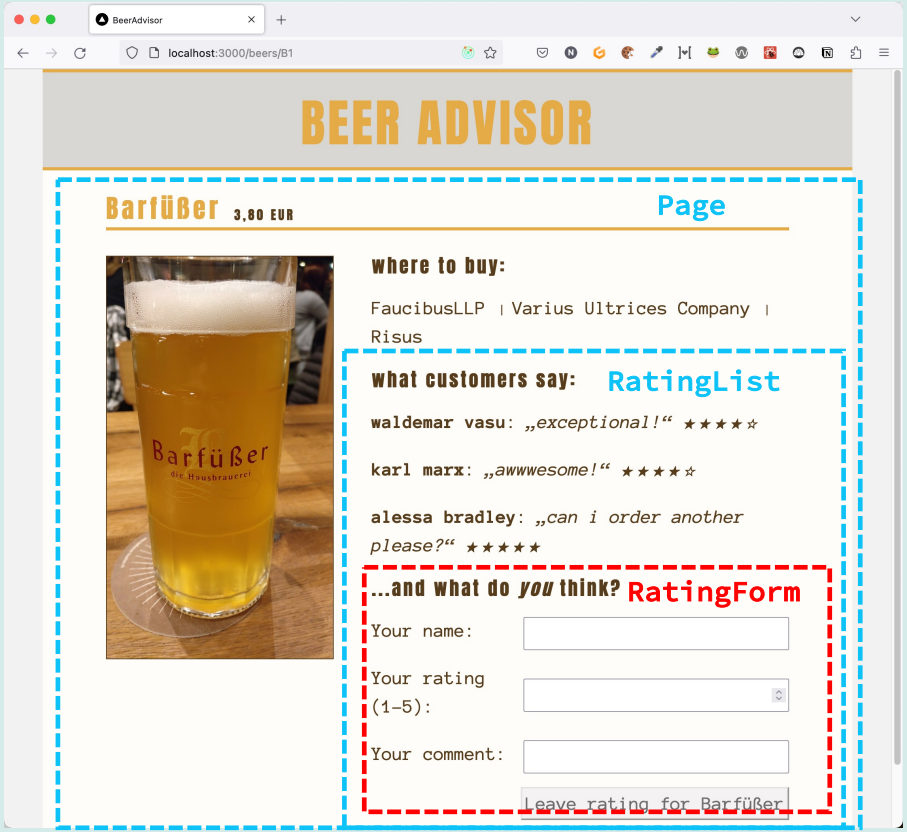
- **In BeerListPage abhängig von SearchParams sortieren**
  - **An dieser Stelle Server Komponente, d.h. Hook ist hier nicht verwendbar**
  - ```
    type BeerListPageProps = {
      searchParams?: { [key: string]: string };
    };
    ```
  - ```
    const orderBy: OrderBy = (searchParams?.order_by || "name_asc") as OrderBy;
    ```

Mutations

MUTATIONS

Verändern von Daten:

Hinzufügen einer Bewertung

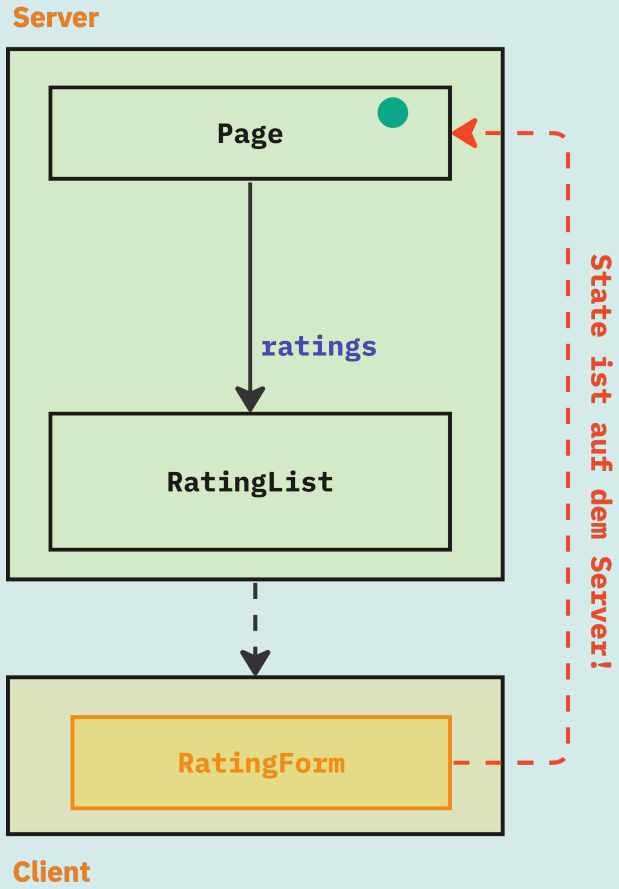


Server-Komponente

Client-Komponente

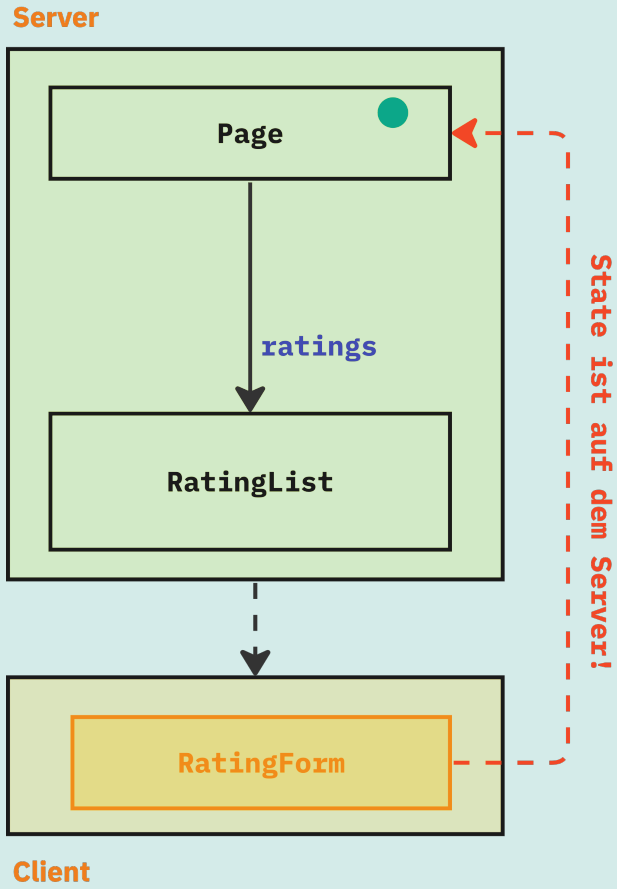
Verändern von Daten

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



Verändern von Daten

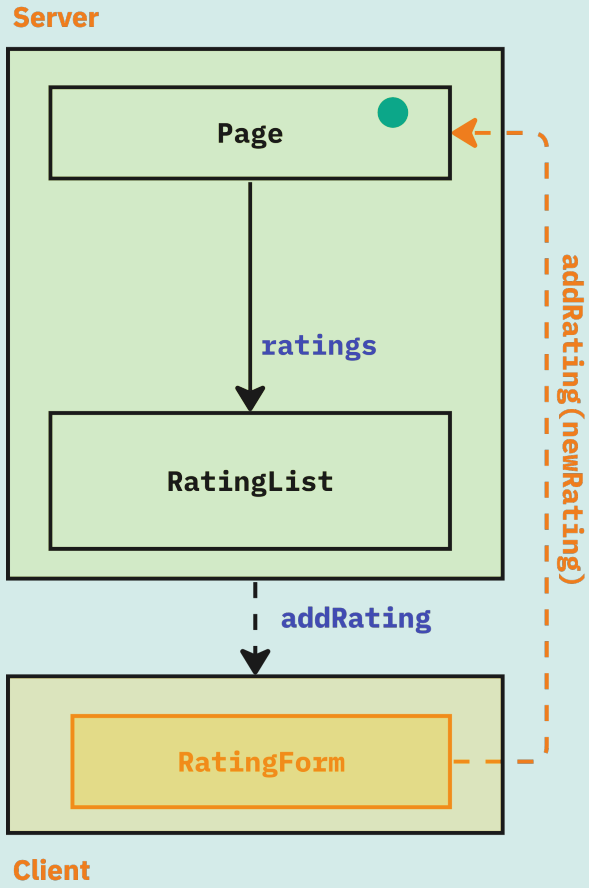
- Nach dem Verändern von Daten muss die UI aktualisiert werden
- Mangels State auf dem Client geht das aber nicht wie bislang
- Der Server muss nach Datenänderungen aktualisierte UI liefern



Server Actions

Server Actions

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



Server Actions


Demo

- **Einfache "helloWorld" async server action bauen**
- **Per Button - z.B. im AddRatingForm – aufrufen**
    - **Netzwerkverkehr angucken**

- **AddRatingForm in BeerDetailPage einfügen**

- **AddRatingForm:**
- **- action** saveNewRating **hinzufügen**

- **Netzwerkverkehr: Antwort vom Server, keine UI**

- **in** saveNewRating invalidateRoute machen

- ? **Menge an Daten?**

Go full-stack
with a framework

...oder

doch nicht?! 😤

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

Müssen wir jetzt alle serverseitiges React machen?

Ginge es nicht auch ohne Framework?

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

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

Müssen wir jetzt alle serverseitiges React machen?

Ginge es nicht auch ohne Framework?

Client-seitiges React (SPA) wird bleiben

- Tooling-Frage offen
  - Status von create-react-app unklar, tendenziell ☠️
  - vite bietet mittlerweile eigenes React Template
  - Mit Next.js "static exports"-Mode kann man klassische SPAs bauen
  - Remix bietet neuerdings auch SPA-Mode an
  - NX gibt es auch noch

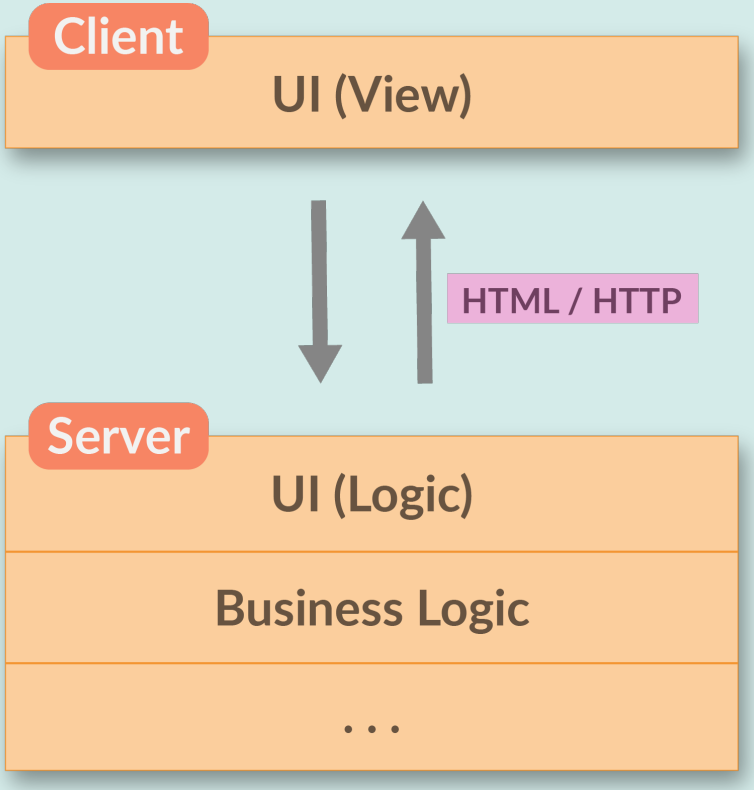
"Moderne" React-Architekturen lösen Probleme

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

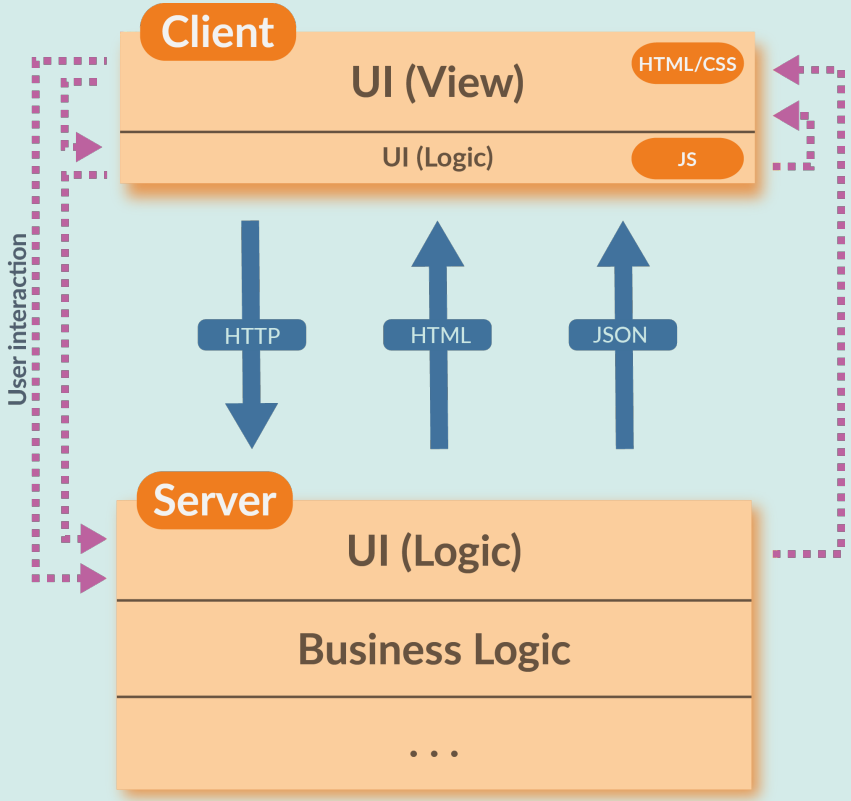
Fazit

Fullstack – Zukunft der Frontend-Entwicklung?

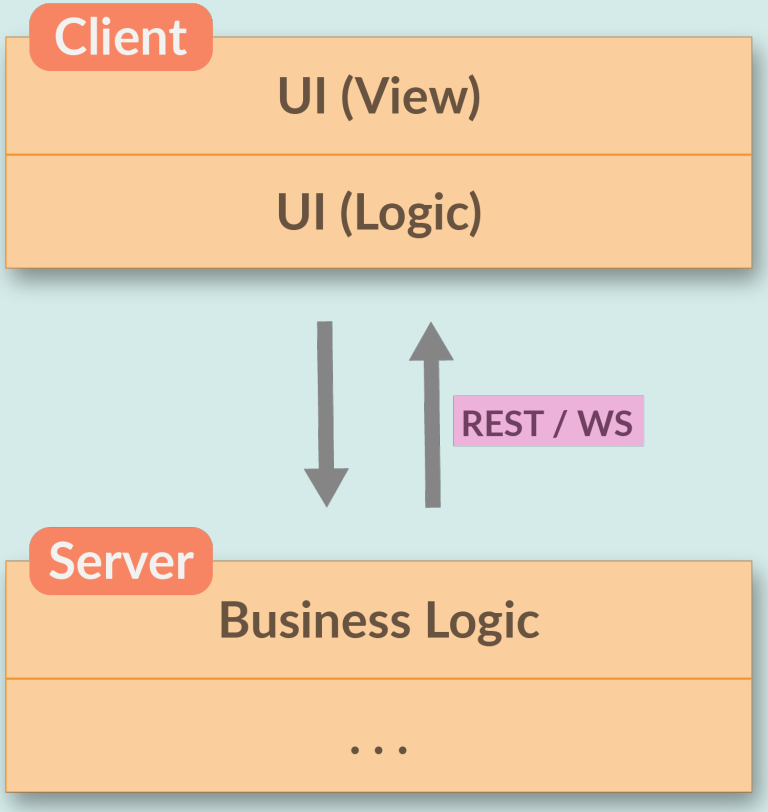
- Serverseitige App (Java, C#, ...)



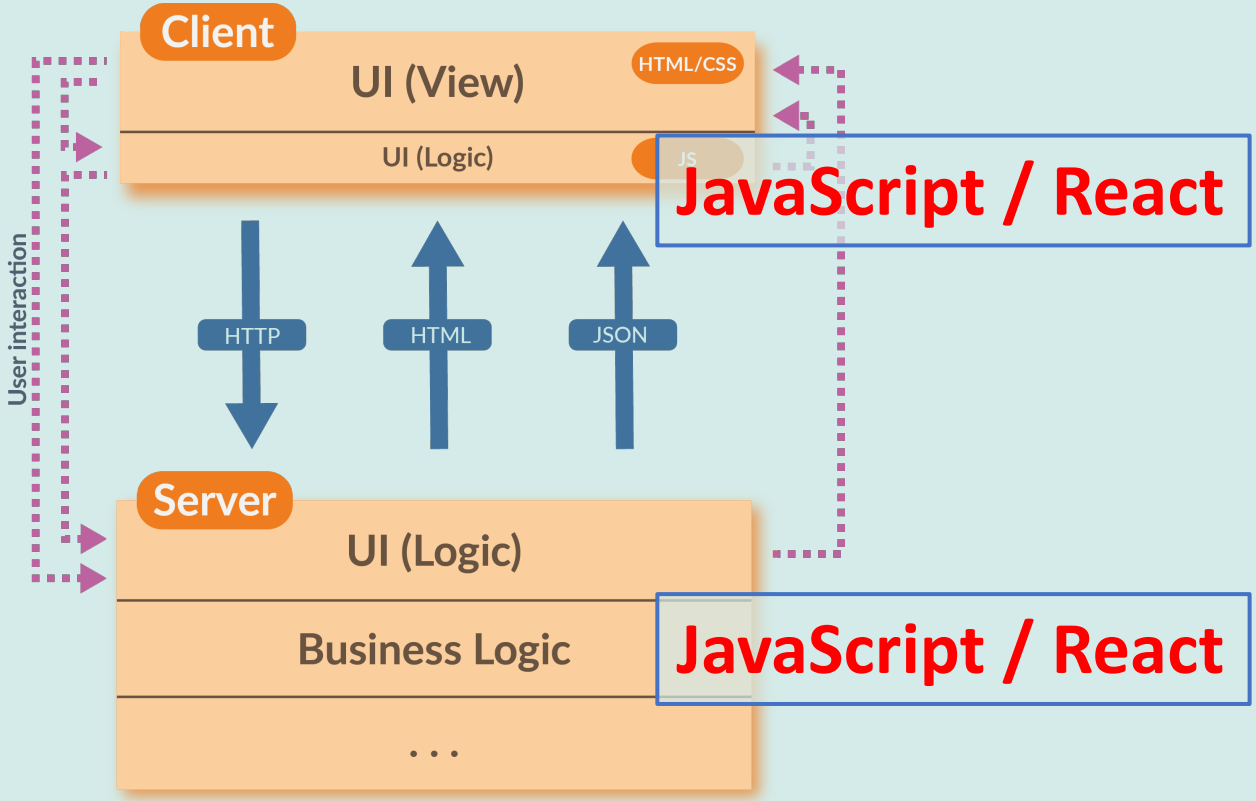
- ...wie es weiter ging (Serverseite + jQuery)



- ...bis zur Single-Page-Anwendung...



- ...und nun?


Client
UI (View)
HTML/CSS
UI (Logic)
JS
JavaScript / React
User interaction
HTTP
HTML
JSON
Server
UI (Logic)
Business Logic
...
JavaScript / React

FAZIT

Zukunft?

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

FAZIT

Zukunft?

- Im Gegensatz zu Hooks gibt es viel mehr Kritik, Widerspruch und Unverständnis
- Ob sich die RSC durchsetzen, werden wir sehen
- Die Kommunikation ~~könnte~~ muss besser laufen

FAZIT

Zukunft?

- Aus meiner Sicht werden die Implikationen unterschätzt
  - Man kann mit Next.js "statische" SPAs bauen, aber das ist sicher ein Sonderfall
  - Betrieb eines JavaScript-Servers bringt neue Herausforderungen
  - Vergangenheit hat gezeigt, dass "transparente" Aufteilung von Server/Client-Teilen nicht trivial ist

Zukunft?

- RSC gehen wieder Richtung "multi-page-application"
  - Mischung von Server-Client-Code kann verwirrend sein
  - Ob und welche Features der Fullstack-Frameworks verwendet werden können, hängt von der eigenen Umgebung ab
  - Ob man von RSC-Versprechen profitiert hängt auch von eigenem Setup, Infrastruktur ab

Zukunft?

- Was passiert mit bestehenden Anwendungen?
  - Je nach eingesetztem und gewünschtem Tech-Stack sehr aufwendig (aka "rewrite")

Zukunft?

- Noch viele Fragen offen
  - (unter anderem der Akzeptanz)

- In jedem Fall wird "klassisches" React auch weiterhin funktionieren

NILS HARTMANN
https://nilshartmann.net

https://reactbuch.de

Vielen Dank!

Slides: https://react.schule/oose24

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

Fragen & Kontakt: nils@nilshartmann.net

Twitter: @nilshartmann