NILS HARTMANN
https://nilshartmann.net

Die Qual der Wahl

Die passende

Frontend-Technologie

Slides: https://react.schule/jax-2024

NILS HARTMANN

nils@nilshartmann.net

Freiberuflicher Entwickler, Architekt, Trainer aus Hamburg

Java, Spring, GraphQL, React, TypeScript

https://graphql.schule/video-kurs

https://reactbuch.de

BEISPIEL-CODE

Ihr findet das "Recipify"-Beispiel in unterschiedlichen Implementierungen (und leicht unterschiedlichem Feature-Set) hier:

HTMX und React SPA

https://github.com/nilshartmann/jughh-spa-oder-htmx-backend

Next.JS

https://github.com/nilshartmann/nextjs-workshop

Frontend-Entwicklung ...

Frontend-Entwicklung ...

Frontend-Entwicklung ...

Was soll der ganze
(JavaScript-)
Quatsch?

Frontend-Entwicklung ...

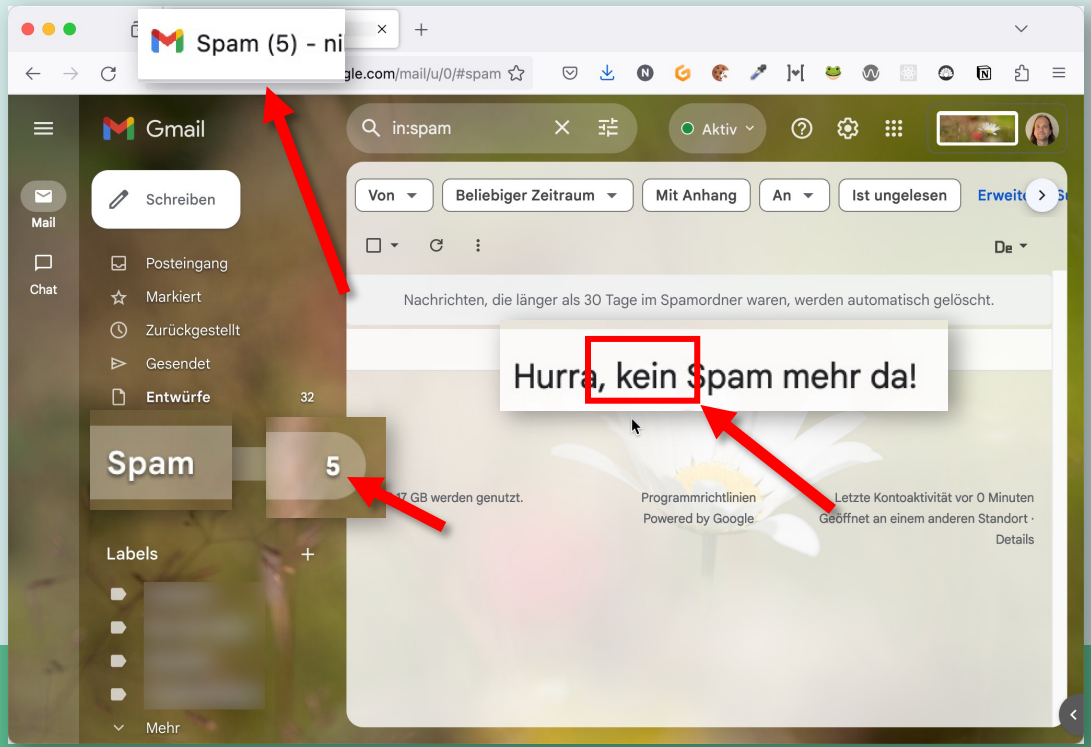
Frontend?
Das ist doch kein Problem,
bisschen HTML+CSS = fertig

Frontend-Entwicklung ...

Können wir das nicht nebenbei machen?
Was soll da schon schiefgehen?

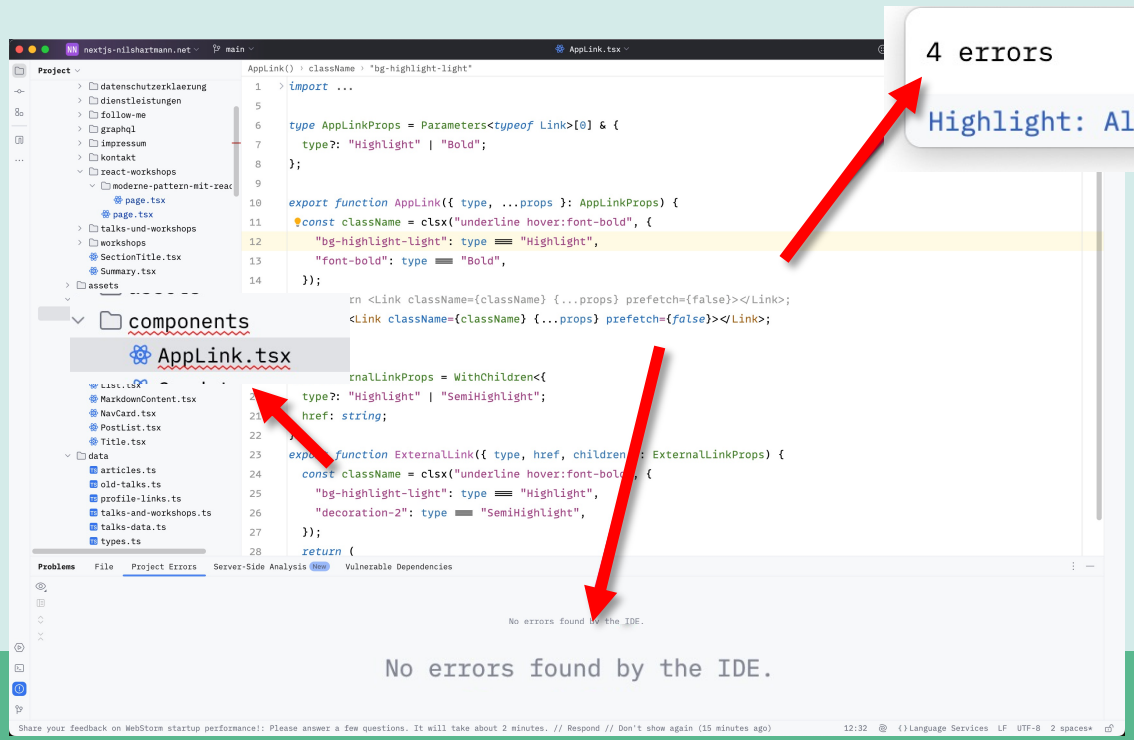
Frontend-Entwicklung:

Was soll schon schief gehen?

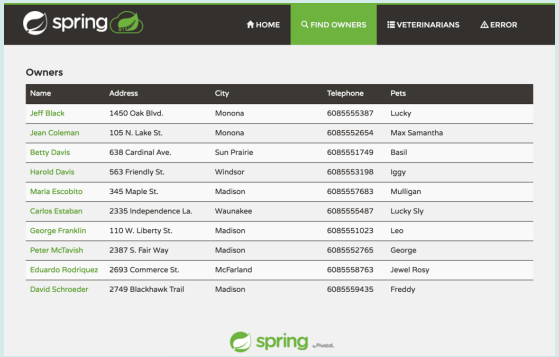


Frontend-Entwicklung:

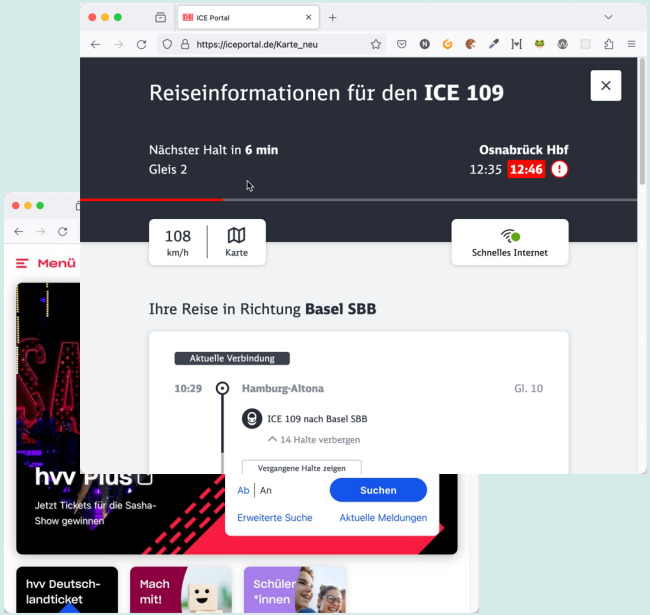
Was soll schon schief gehen?

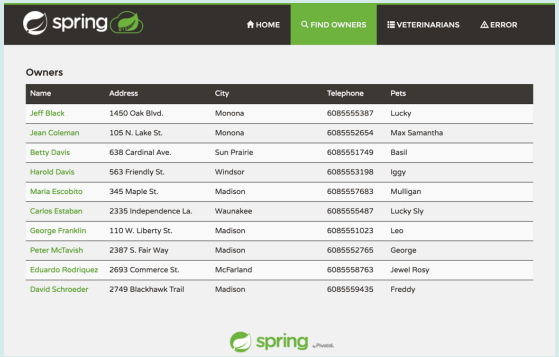


WEBANWENDUNGEN

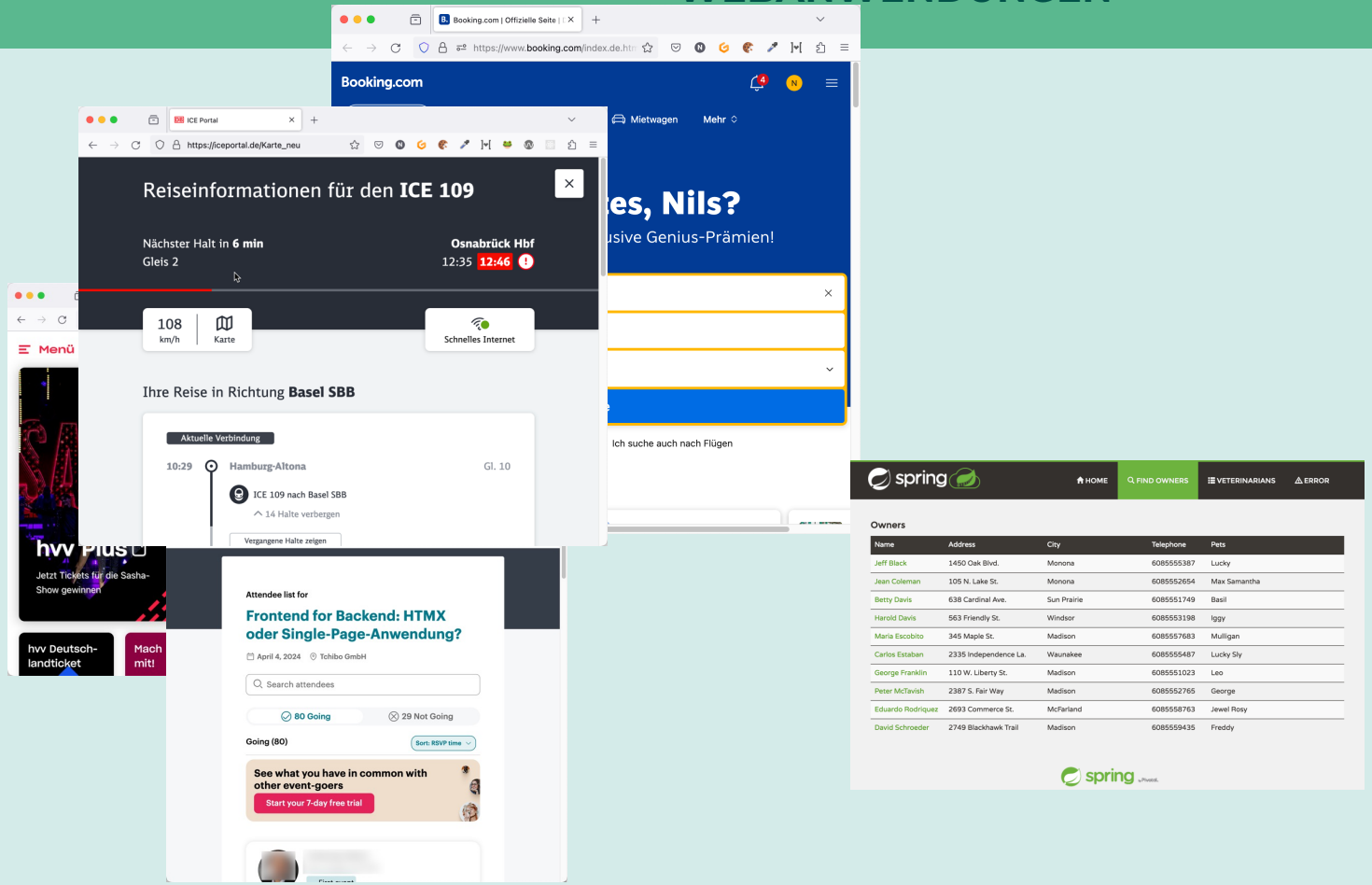


WEBANWENDUNGEN

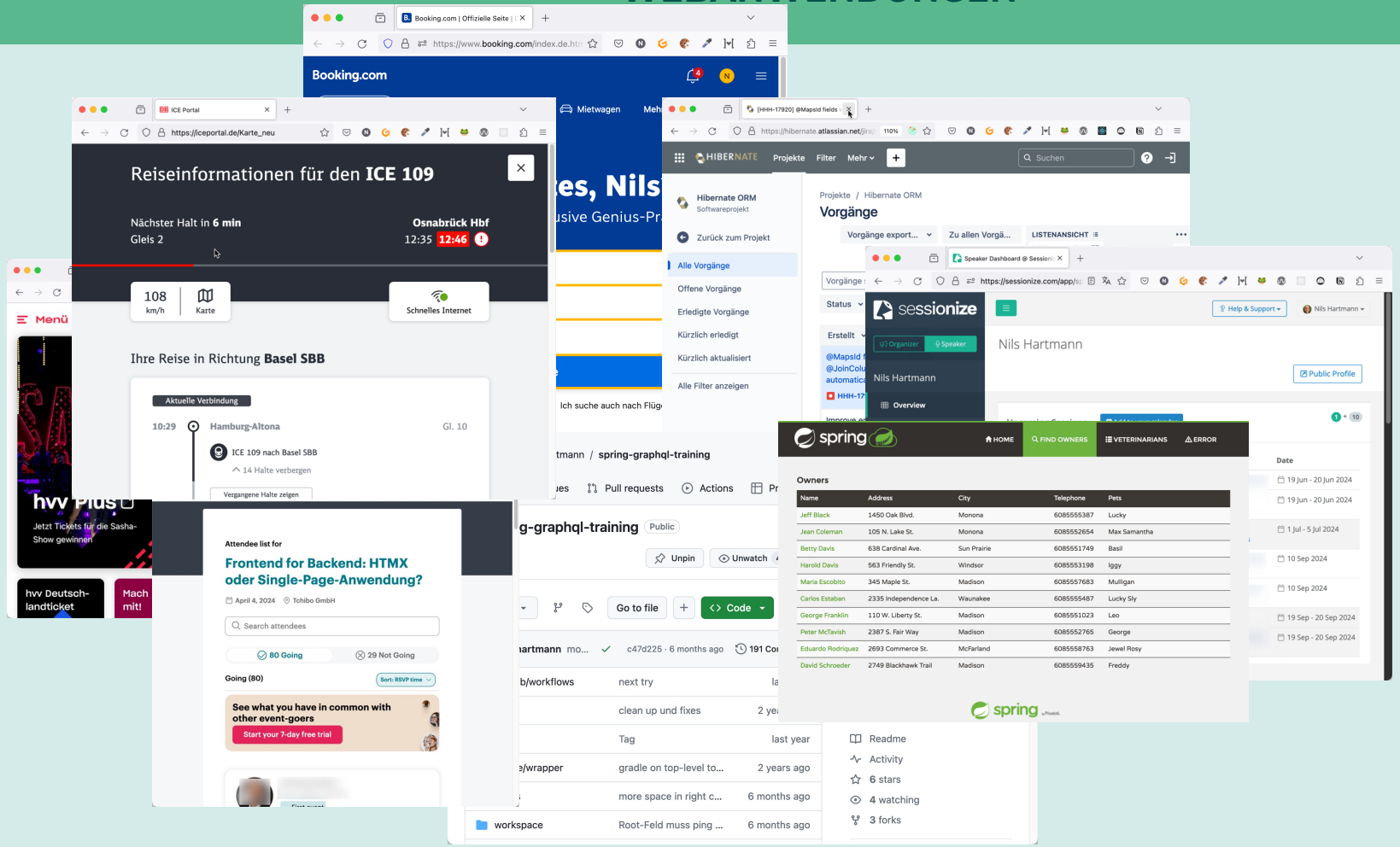




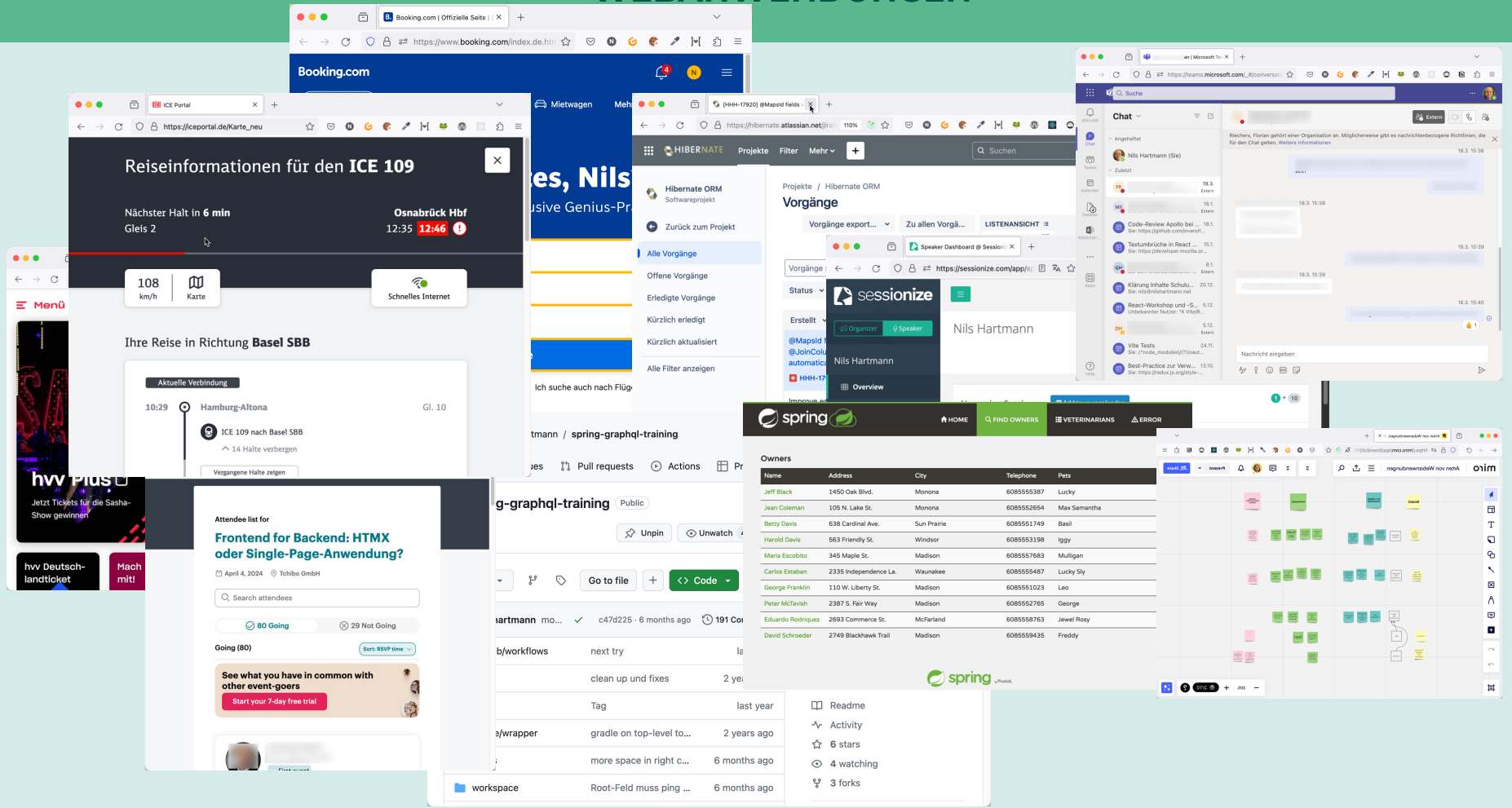
WEBANWENDUNGEN



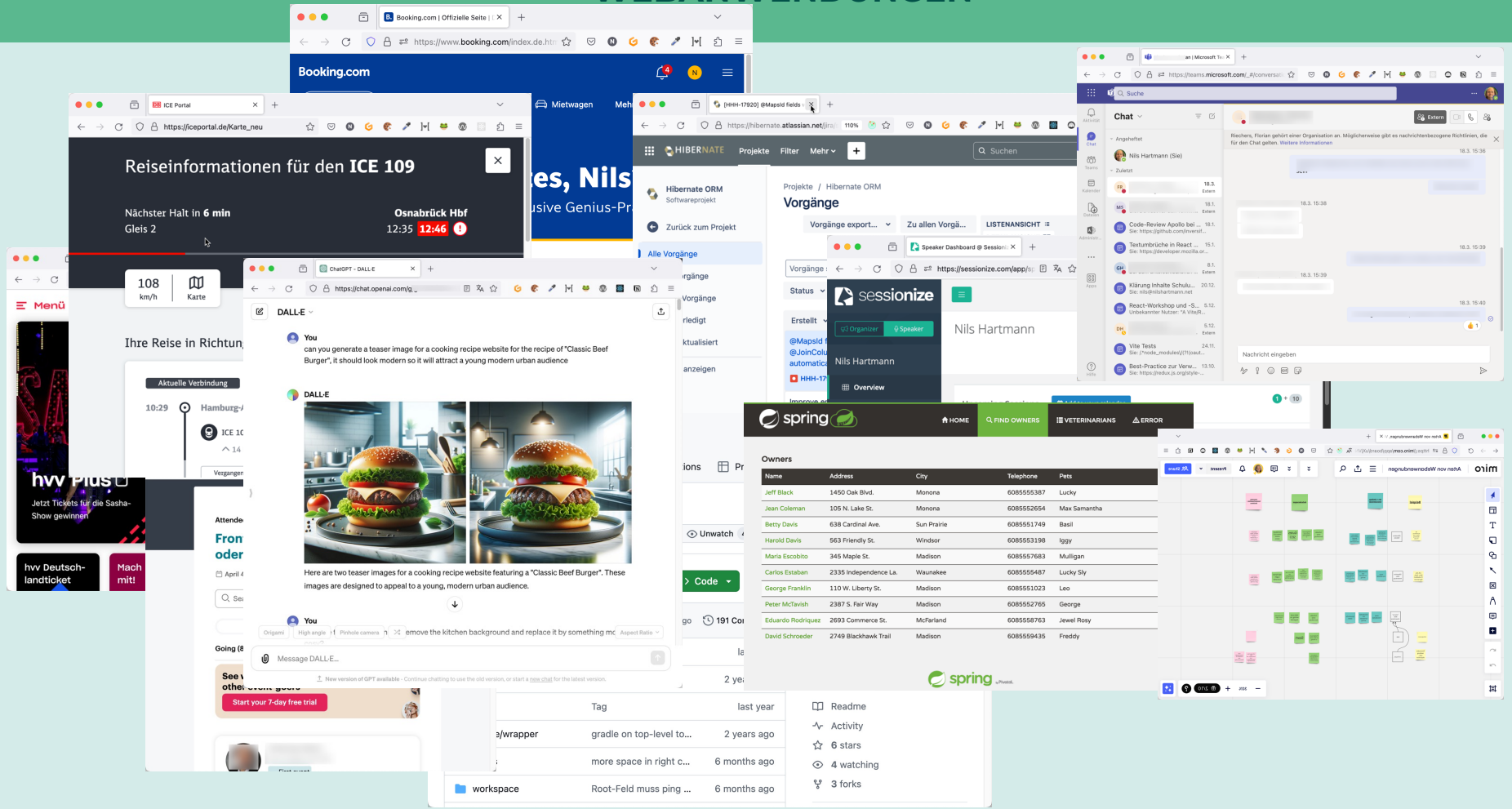
WEBANWENDUNGEN



WEBANWENDUNGEN

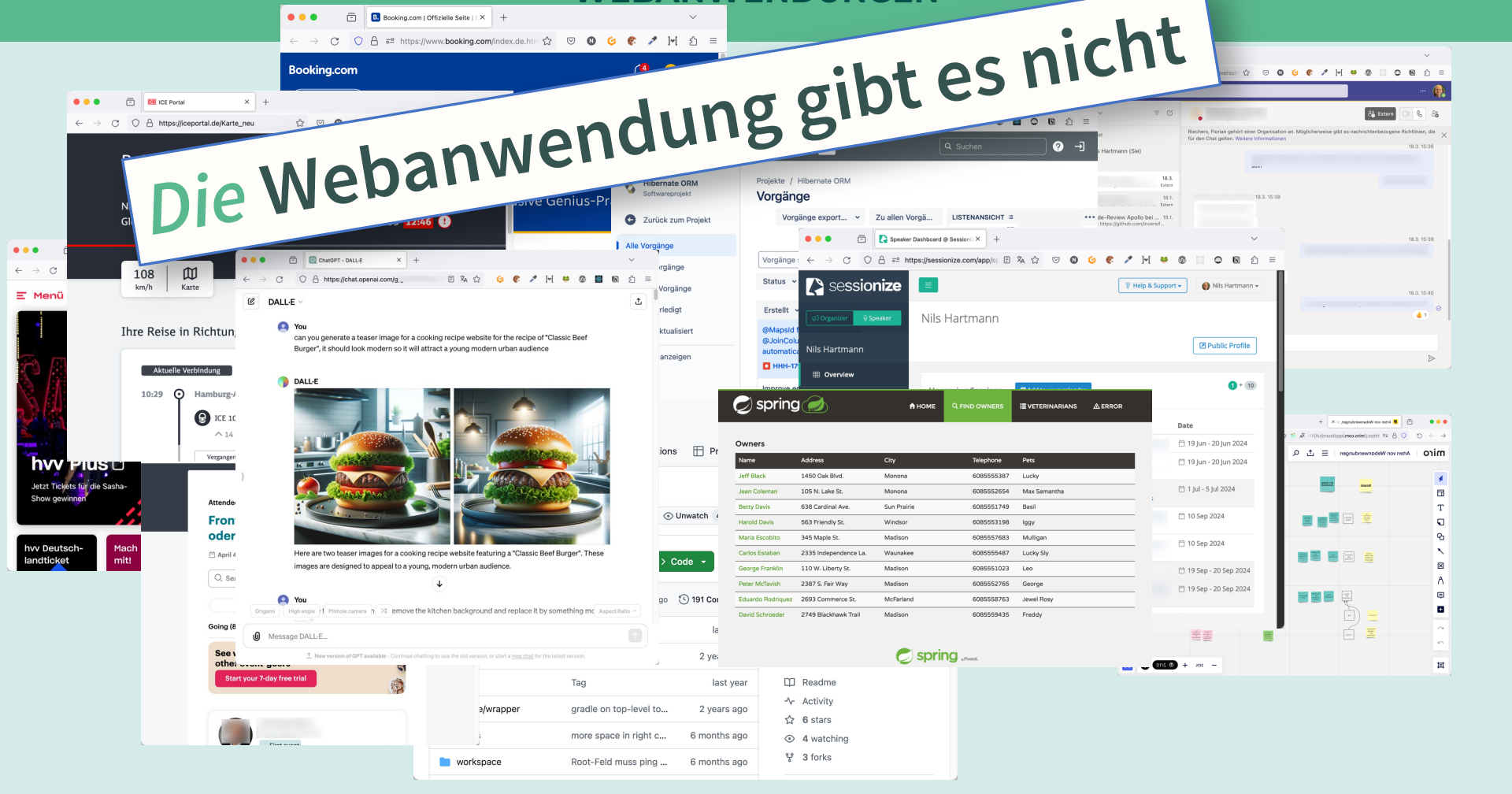


WEBANWENDUNGEN



WEBANWENDUNGEN

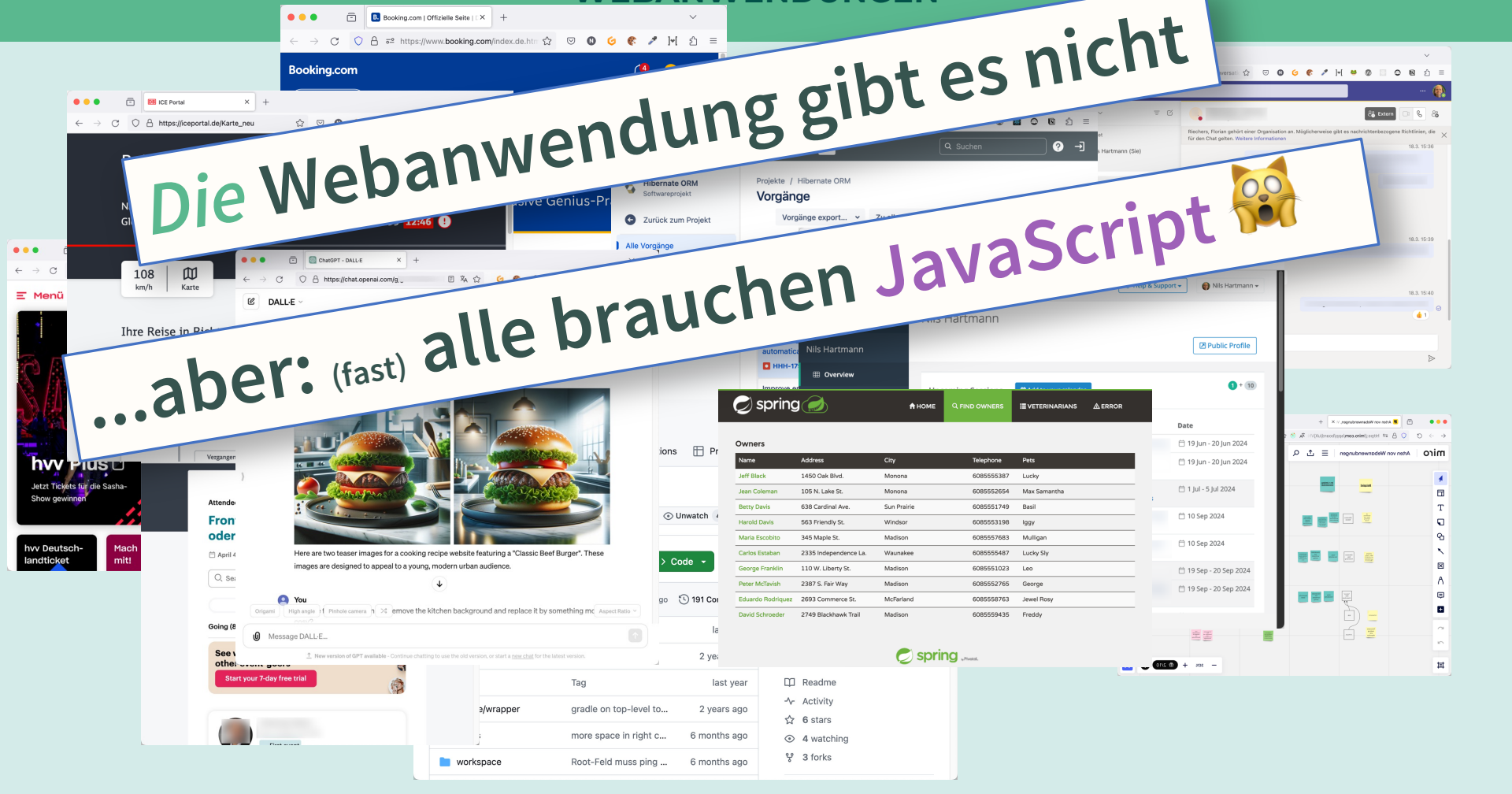
Die Webanwendung gibt es nicht



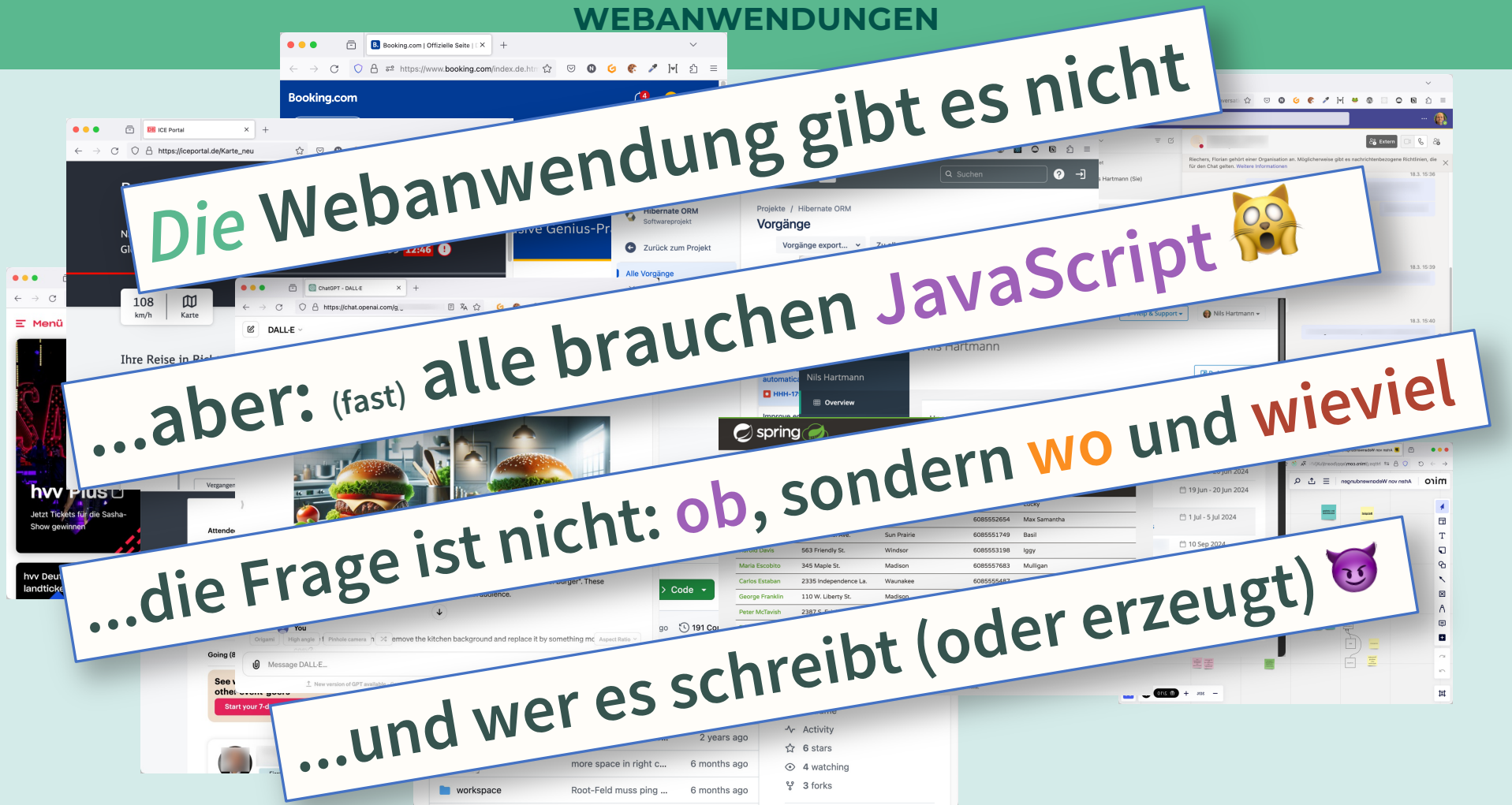
WEBANWENDUNGEN

Die Webanwendung gibt es nicht

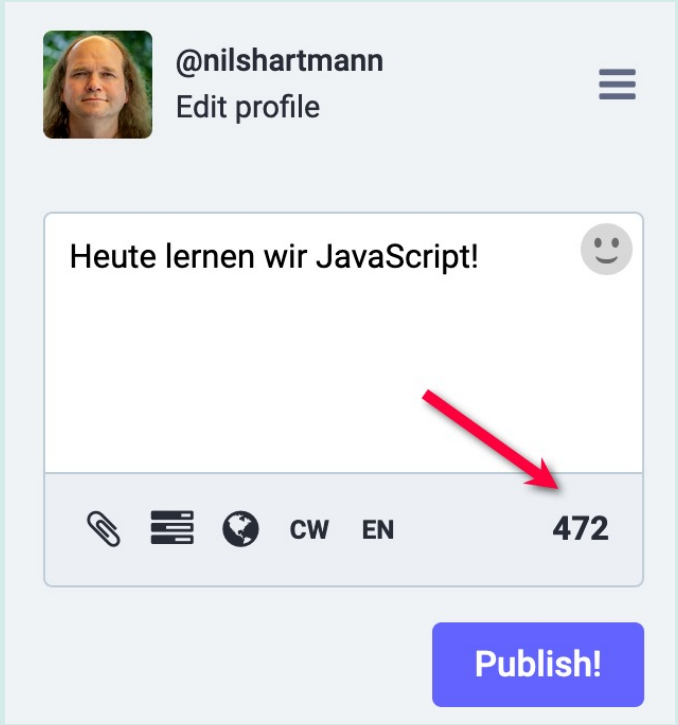
...aber: (fast) alle brauchen JavaScript




Die Webanwendung gibt es nicht
...aber: (fast) alle brauchen JavaScript
...die Frage ist nicht: ob, sondern wo und wieviel

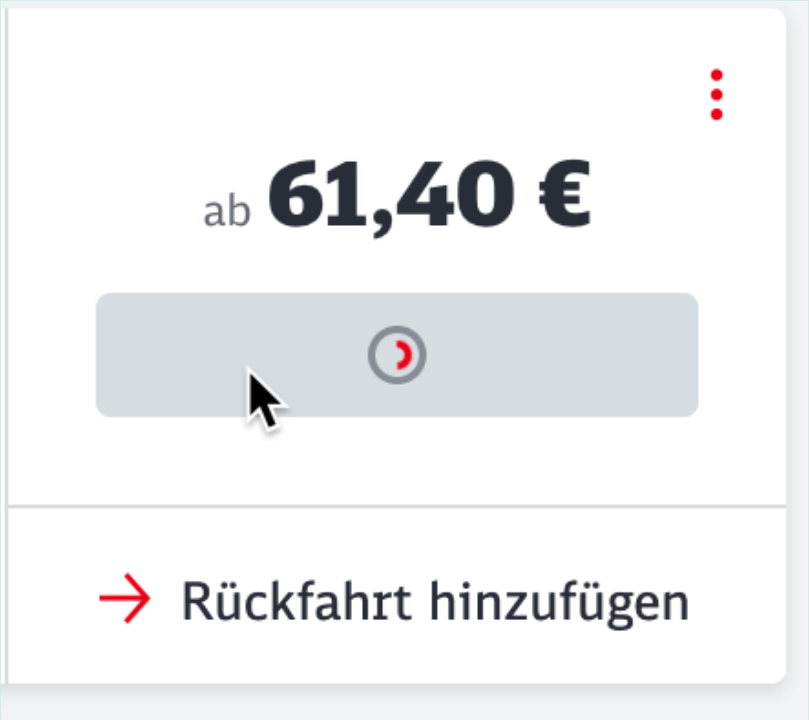

Die Webanwendung gibt es nicht
...aber: (fast) alle brauchen JavaScript
...die Frage ist nicht: ob, sondern wo und wieviel
...und wer es schreibt (oder erzeugt)

Ein Beispiel...



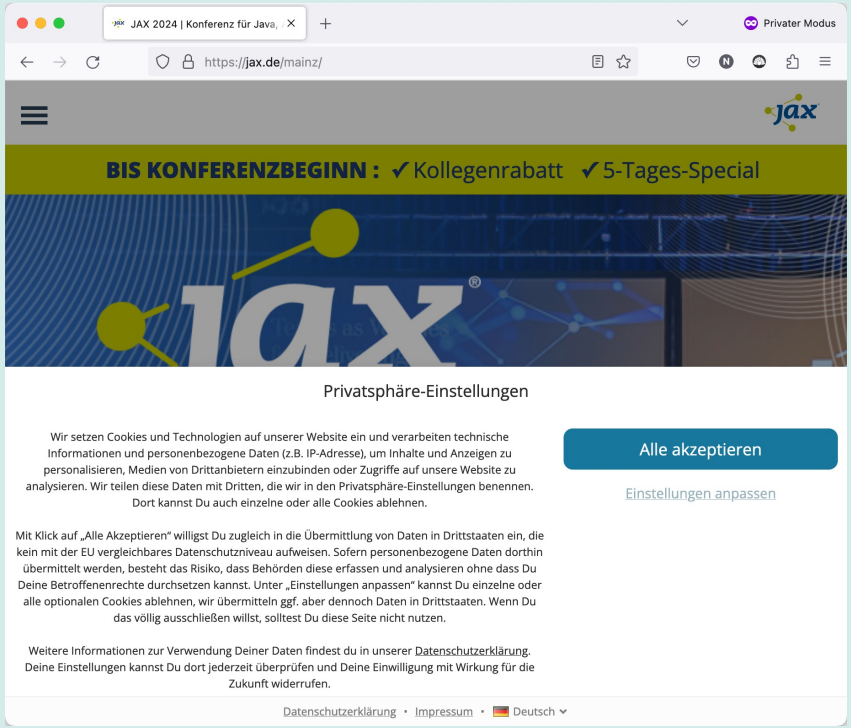
Interaktiver Zeichenzähler

Ein Beispiel...



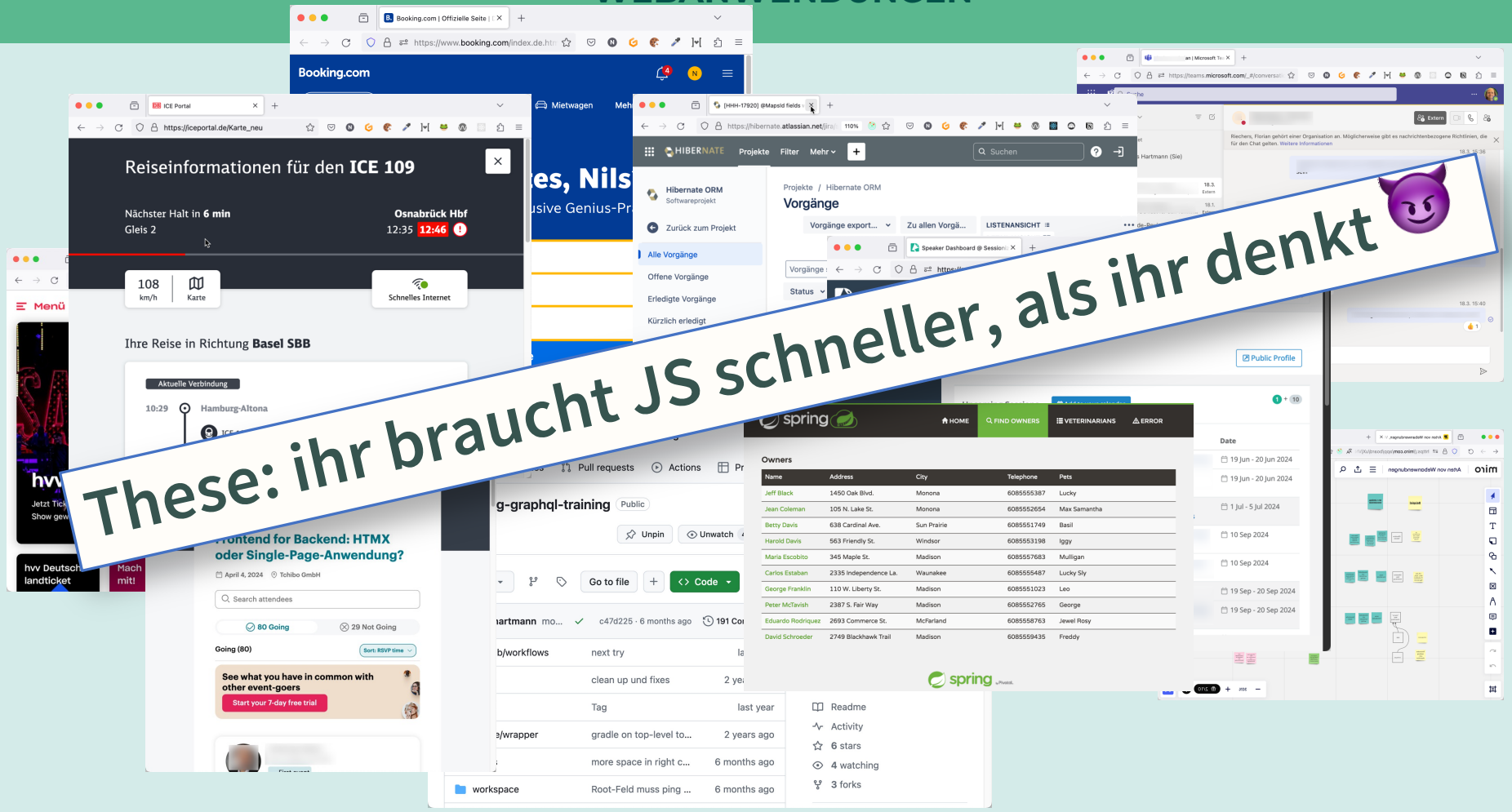
Ladeanimation

Ein Beispiel...



Cookie-Banner wegklicken 🥳

WEBANWENDUNGEN



Es geht also "nur" darum, wie wir mit JavaScript umgehen:

Es geht also "nur" darum, wie wir mit JavaScript umgehen:

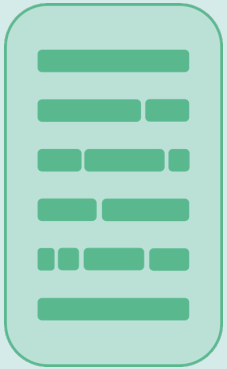
Welche Konsequenz hat JavaScript zur Laufzeit

Es geht also "nur" darum, wie wir mit JavaScript umgehen:

Welche Konsequenz hat JavaScript zur Laufzeit

...und bereits während der Entwicklung?

"Können wir nicht hier und da, ad-hoc JavaScript hinzufügen?"



HTML-Seite

Ja, das geht!

HTML-Seite

- Typische Vertreter dieser Architektur zum Beispiel Spring WebMVC
- Templatesprache (z.B. Thymeleaf)
- JavaScript Schnipsel ("vanilla" oder zum Beispiel jQuery)

Ja, das geht!

HTML-Seite

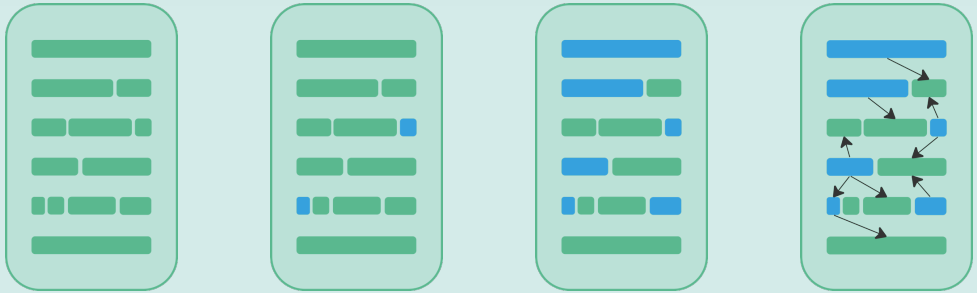
- Eigentlich optimal:
  - wir haben JavaScript nur da, wo wir es wirklich brauchen, für Interaktivität
  - alles andere kann statisches HTML und CSS sein ❤️

"...hier und da müsste auch noch schnell was interaktives hin..."

HTML-Seite

- Wir schreiben also noch etwas mehr JavaScript Schnipsel

"...und hier... und hier ... und hier... und ..." au weia!



HTML-Seite

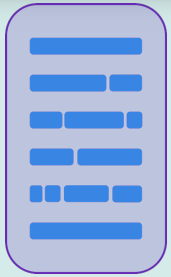
Das Problem von Schnipsel-Architektur

- Bunter Strauß an Server- und Client-Technologien (Backend-Sprache, Template-Sprache, JavaScript)
- Verantwortlichkeit willkürlich auf Frontend und Backend aufgeteilt

Das Problem von Schnipsel-Architektur

- Das Problem "schleicht sich ein"
- Es gilt das "Gesetz des Umschlagens von Quantität in Qualität" (F. Engels):
- Plötzlich hat unser Code nicht "ein paar Probleme" sondern ist kaputt

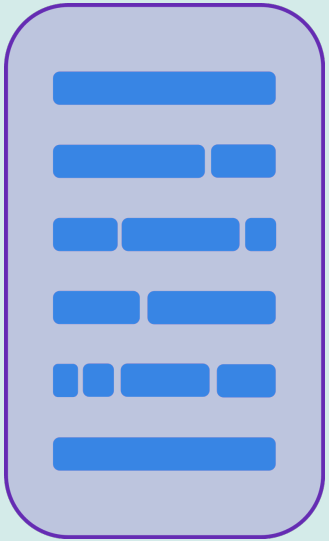
Dann besser alles in JavaScript? Single-Page-Anwendungen



JavaScript-Code

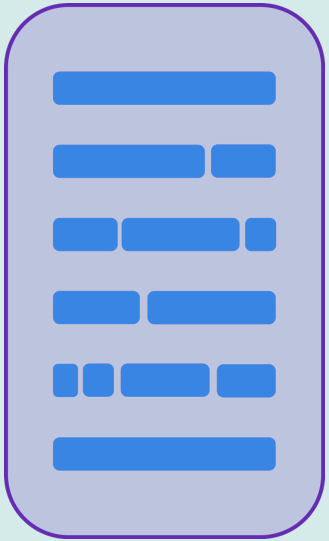
- ab ca. 2010
- Aus "Seiten" werden jetzt "Anwendungen"
- Klare Verantwortlichkeit: Server für Logik und Daten, Browser für UI
- Es gibt stabile und verbreitete Frameworks für jeden Geschmack

Single-Page-Anwendungen



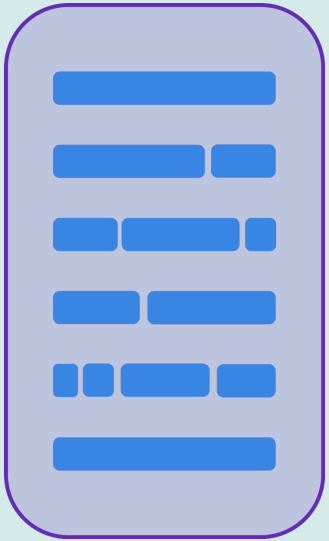
- Darstellung erfolgt vollständig mit JavaScript
- Statisches HTML spielt (fast) keine Rolle
- Die Anwendung kommuniziert mit dem Backend über API
- Ausgetauscht werden Daten, aber keine UI
- Vertreter: Angular, React, Svelte, Vue

Single-Page-Anwendungen



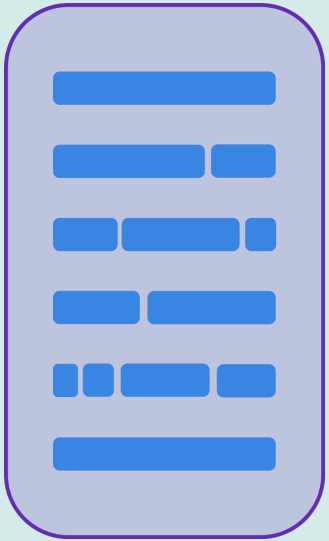
- Konsequenz #1: (viel) JavaScript zur Entwicklungszeit

Single-Page-Anwendungen



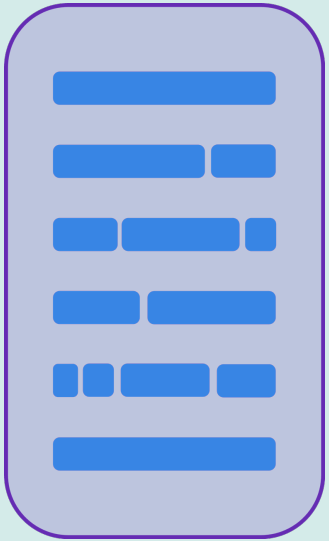
- Konsequenz #1: (viel) JavaScript zur Entwicklungszeit
- Konsequenz #2: (viel) JavaScript zur Laufzeit

Single-Page-Anwendungen



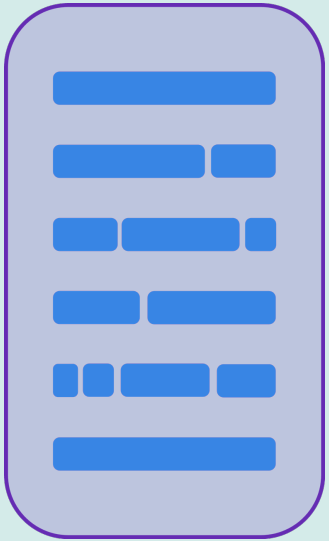
- Konsequenz #1: viel JavaScript zur Entwicklungszeit

Single-Page-Anwendungen



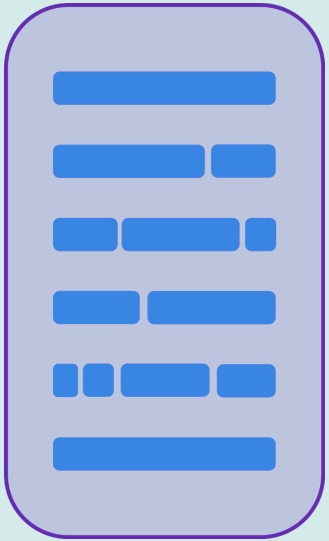
- **Konsequenz #1: viel JavaScript zur Entwicklungszeit**
- Es gibt modernes Tooling für die Entwicklung
  - Wie aus der Enterprise Backend-Entwicklung mit Java gewohnt

Single-Page-Anwendungen



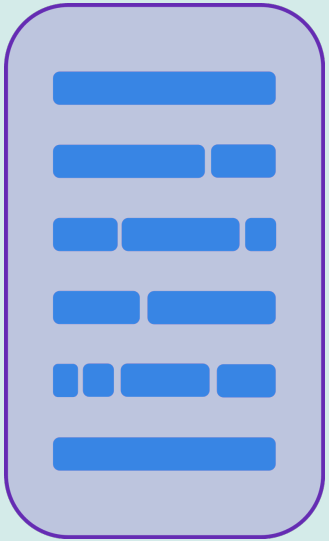
- **Konsequenz #1: viel JavaScript zur Entwicklungszeit**
- Es gibt modernes Tooling für die Entwicklung
  - Wie aus der Enterprise Backend-Entwicklung mit Java gewohnt
  - IDEs und Debugger, Typsicherheit, Unit- und Integrationstest, ...

Single-Page-Anwendungen



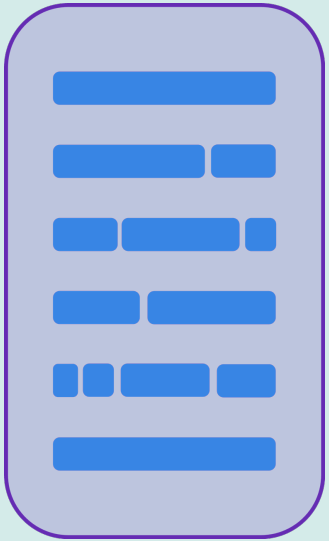
- **Konsequenz #1: viel JavaScript zur Entwicklungszeit**
- Es gibt modernes Tooling für die Entwicklung
  - Wie aus der Enterprise Backend-Entwicklung mit Java gewohnt
  - IDEs und Debugger, Typsicherheit, Unit- und Integrationstest, ...
- Aber: wenn man JavaScript doof findet, falscher Ansatz

Single-Page-Anwendungen



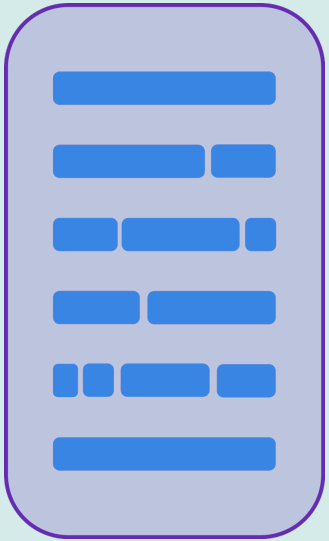
- **Konsequenz #2: viel JavaScript zur Laufzeit**
- Auch für statische Inhalte

Single-Page-Anwendungen



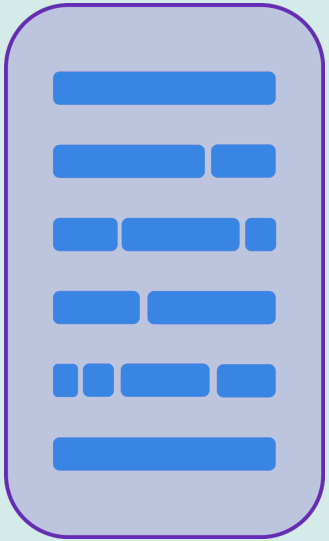
- JavaScript-Code:
    1. muss zum Browser gesendet werden
    2. muss vom Browser ausgeführt werden
    3. kann dann die darzustellenden Daten lesen
    4. kann dann erst die Daten anzeigen
    5. erst dann ist die Anwendung einsatzbereit
    6. Mit jedem Feature wird es mehr

Single-Page-Anwendungen



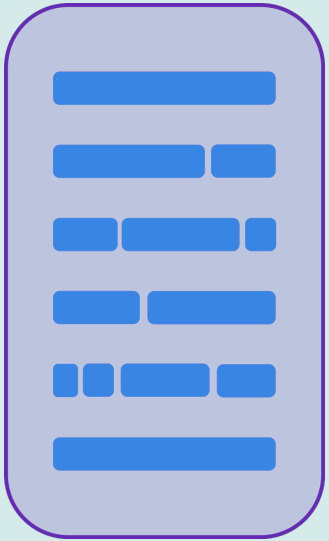
- Das hat Auswirkungen auf die **Laufzeit**-Performance
- Insbesondere beim Starten

Single-Page-Anwendungen

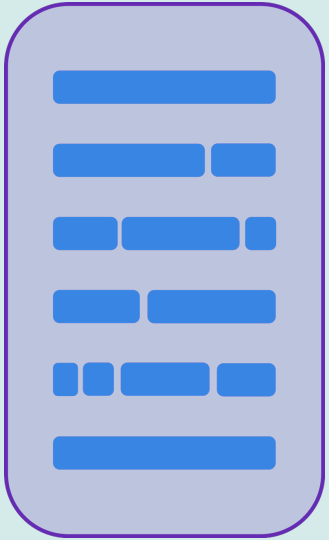


- Das hat Auswirkungen auf die **Laufzeit**-Performance
- Insbesondere beim Starten
- Ob das ein Problem für die eigene Anwendung ist, muss man von Fall zu Fall entscheiden
  - In-House- oder B2B-Anwendungen haben andere Anforderungen als ein Online-Shop

Fullstack-Anwendungen (JavaScript)

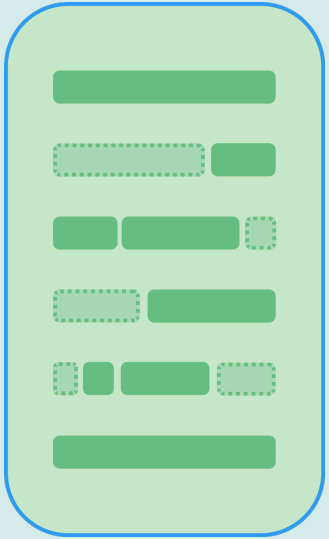


Fullstack-Anwendungen (JavaScript)



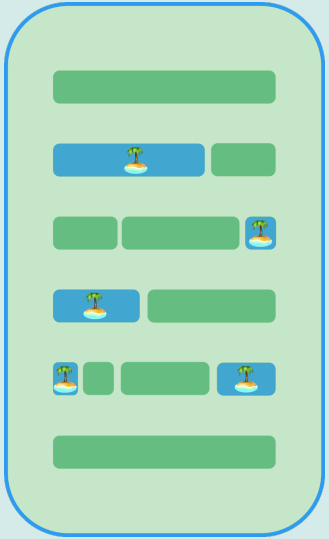
- Ebenfalls vollständig in JavaScript geschrieben

Fullstack-Anwendungen (JavaScript)



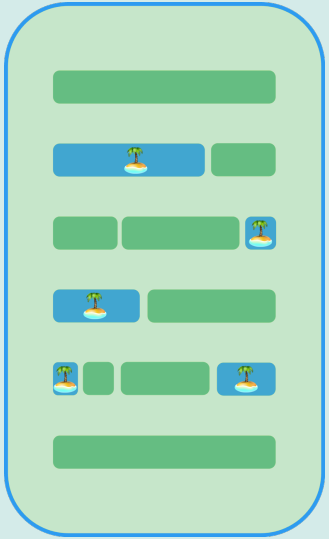
- Grundsätzliche Idee:
  1. **UI-Code** wird serverseitig vorgerendert
  2. **UI-Code** wird zum Browser gesendet und angezeigt

Fullstack-Anwendungen (JavaScript)



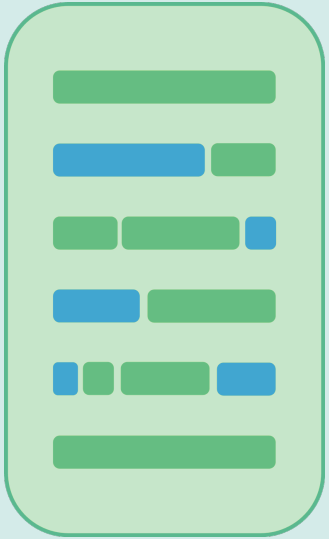
- Grundsätzliche Idee:
  1. **UI-Code** wird serverseitig vorgerendert
  2. **UI-Code** wird zum Browser gesendet und angezeigt
  3. Nur der JavaScript-Code ("Islands") für Interaktionen wird zum Browser geschickt

Fullstack-Anwendungen (JavaScript)



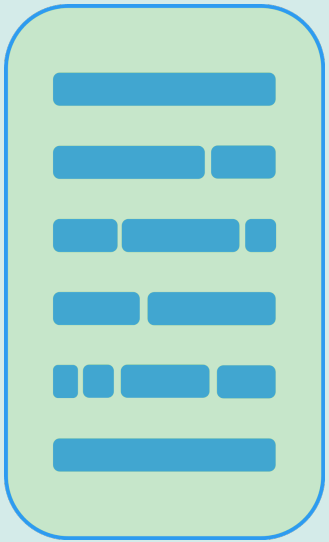
- Anwendung startet schneller:
  1. Browser bekommt UI-Code zur Darstellung
  2. Der notwendige JavaScript-Code wird nachgeladen
  3. Anwendung jetzt interaktiv

Fullstack-Anwendungen (JavaScript)



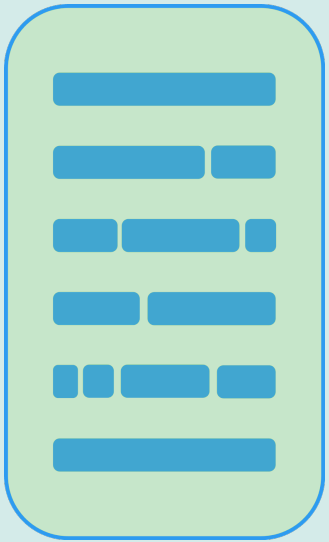
- Wir sind zurück zur JavaScript-Schnipsel-Architektur
  - aber: die Schnipsel werden automatisch vom Framework erzeugt
  - die Schnipsel existieren nur zur Laufzeit

Fullstack-Anwendungen (JavaScript)



- Wir sind zurück zur JavaScript-Schnipsel-Architektur
  - aber: die Schnipsel werden automatisch vom Framework erzeugt
  - die Schnipsel existieren nur zur Laufzeit
  - In der Entwicklung ist "unser" Code aus "einem Guss"
  - aber weiterhin in JavaScript

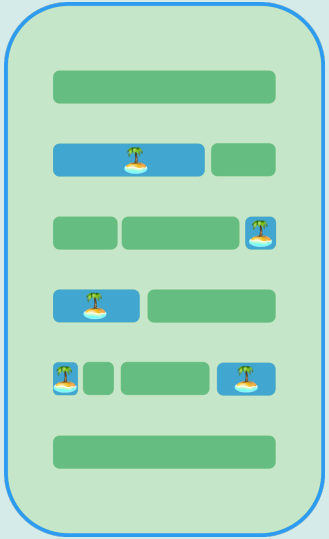
Fullstack-Anwendungen (JavaScript)



- Beispiel: Next.js
- Likes-Komponente ist interaktive Client-Komponente ("Island")

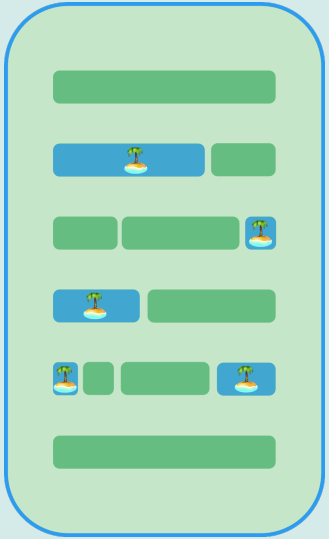
```
function RecipeCard({ recipe }: { recipe: RecipeDto }) {
  return (
    <section>
      <H1>{recipe.title}</H1>
      <RecipeCategories>
        <Likes />
        <CookingTime />
      </RecipeCategories>
    </section>
  );
}
```

Fullstack-Anwendungen (JavaScript)



- Bekannte Vertreter:
  1. Next.js (React)
  2. SvelteKit (Svelte)
  3. Nuxt.js (Vue)
  4. Astro (eigenes Framework + Support für alle SPAs)
  5. Qwik (eigenes Framework)

Fullstack-Anwendungen (JavaScript)



- Bekannte Vertreter:
  1. Next.js (React)
  2. SvelteKit (Svelte)
  3. Nuxt.js (Vue)
  4. Astro (eigenes Framework + Support für alle SPAs)
  5. Qwik (eigenes Framework)
- Funktionalität und Herangehensweise unterschiedlich

HTMX

introduction

htmx gives you access to AJAX, CSS Transitions, WebSockets and Server Sent Events directly in HTML, using attributes, so you can build modern user interfaces with the simplicity and power of hypertext

htmx is small (~14k min.gz'd), dependency-free, extendable, IE11 compatible & has **reduced** code base sizes by 67% when compared with react

https://htmx.org/

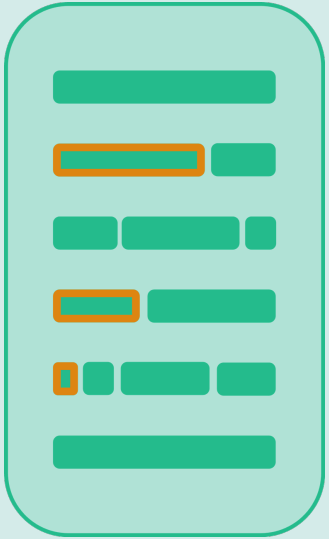
HTMX - Grundlagen

- HTML-Elemente werden mit HTMX-Attributen ergänzt
- Damit wird beschrieben, welche Server Requests bei einem "Trigger" ausgeführt werden sollen
- HTMX kümmert sich um die Ausführung des Requests und die Verarbeitung der Antwort
- Der Server muss HTML-Schnipsel liefern ("Hypermedia")

```
<html lang="en">
  <body>
    <div hx-get="/hello-world"
         hx-trigger="click"
         hx-target="#result">
      Get Greeting
    </div>

    <div id="result"></div>

    <script
      type="text/javascript"
      th:src="@{/htmx/htmx-1.9.10.min.js}"
    ></script>
  </body>
</html>
```

Demo: localhost:8080/hello

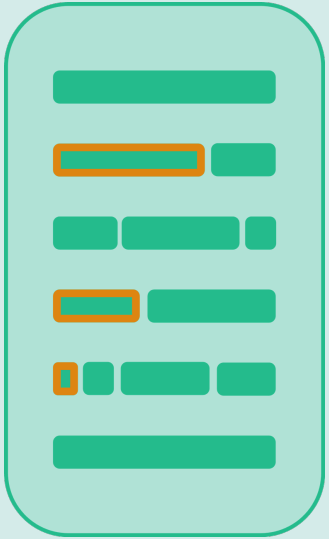
HTMX



- Bei "Triggern" Server Requests für neue **UI**
- Man muss eine Art eigene DSL beherrschen

```
<input
  type="search"
  hx-get="/search"
  hx-trigger="input[target.value.length > 2] changed delay:200ms"
  hx-target="#searchResult"
  hx-swap="outerHTML"
  hx-indicator="#searchIndicator"
/>
```

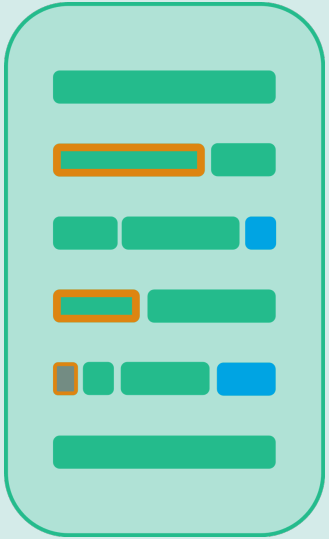
HTMX



- Bei "Triggern" Server Requests für neue **UI**
- Man muss eine Art eigene DSL beherrschen
- ...und eine Template-Sprache im Backend

```
<a
  th:href="@{/search(page=${nextPage},search=${search})}"
  th:if="${hasMore == true}"
  th:hx-get="@{/search(page=${nextPage})}"
  th:classappend="|
    ${!active && !disabled ? 'text-gray hover-underline' : ''}
    ${active ? 'text-blue underline' : ''}
    ${disabled ? 'text-gray' : ''}
    |"
  hx-target="#searchResult" hx-select="#searchResult > *"
>Find more...</a>
```

HTMX: Grenzen



- Für alles andere braucht man… JavaScript!

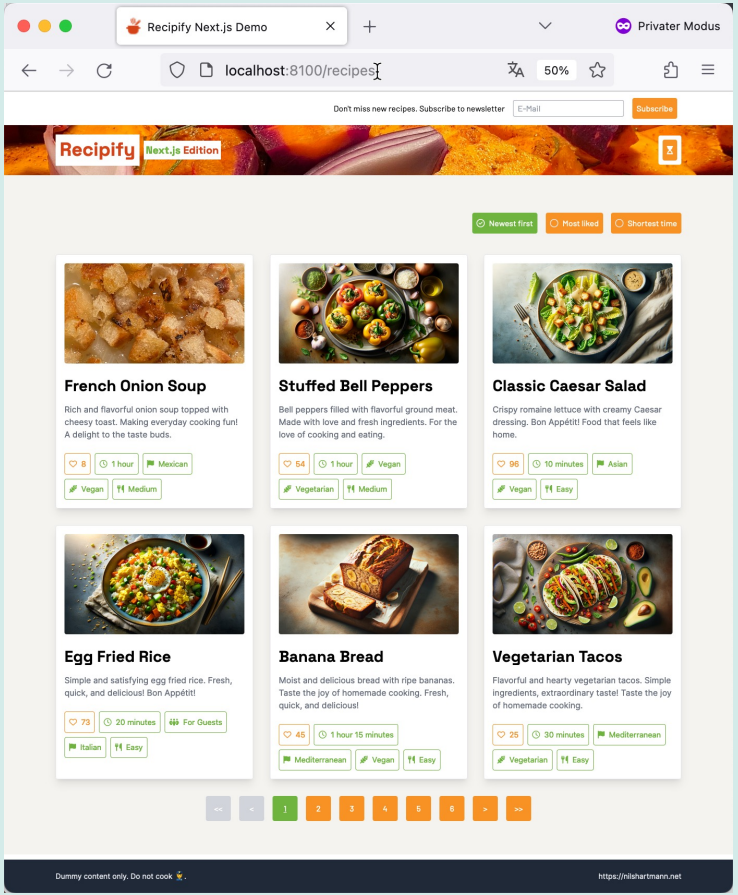
```
<input
    type="search"
    th:value="${search}"
    th:hx-get="@{/search}"
    hx-trigger="input[target.value.length > 2] changed delay:200ms"
    hx-target="#searchResult"
    hx-swap="outerHTML"
    hx-indicator="#searchIndicator"
    hx-on:input="handleInputChange(event)"
/>

<script>
  function updateErrorMessage(e) {...}
</script>
```

Die Ansätze unter der Lupe

EIN PAAR BEISPIELE...

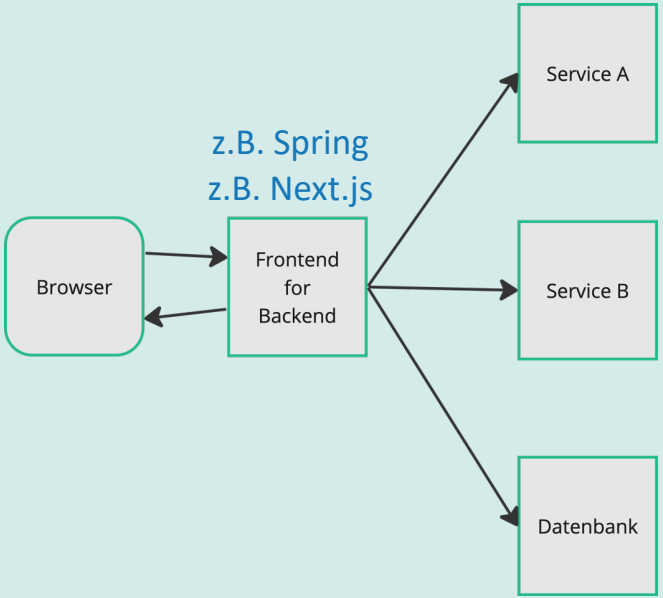
BEISPIEL: INITIALER SEITENAUFRUF

- **Beispiel: initialer Seitenaufruf**
- Demo: localhost:8100



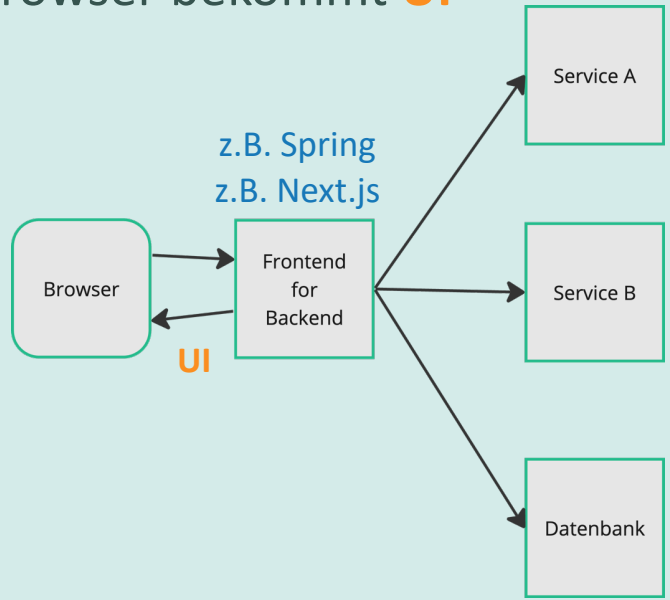
ERMITTLUNG VON DATEN

- **Ermittlung von Daten**
- Fullstack / HTMX: In-Process, DB, API, ...



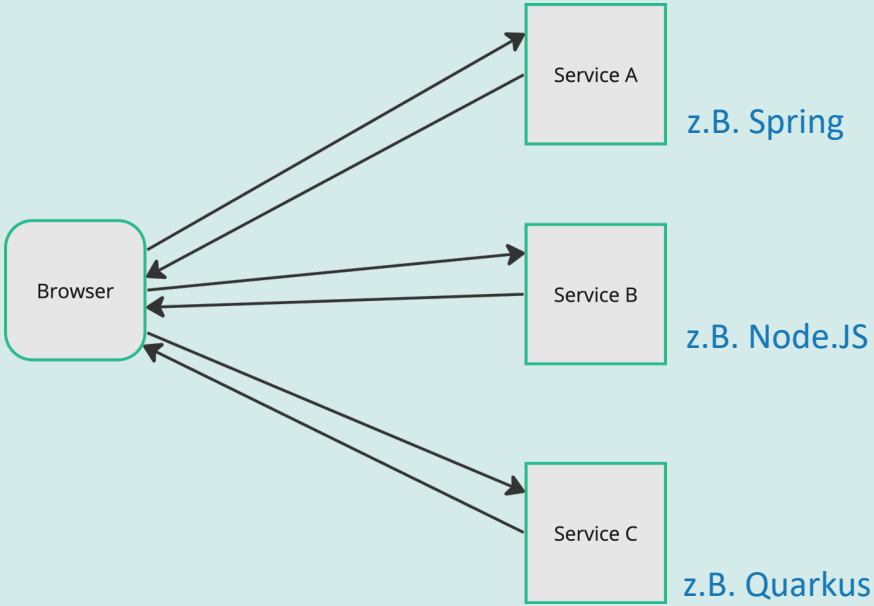
BEISPIEL: INITIALER SEITENAUFRUF

- **Ermittlung von Daten**
- Fullstack / HTMX: In-Process, DB, API, ...
- Backend muss **UI** rendern (bei Fullstack ist das Implementierungsdetail)
- Browser bekommt **UI**



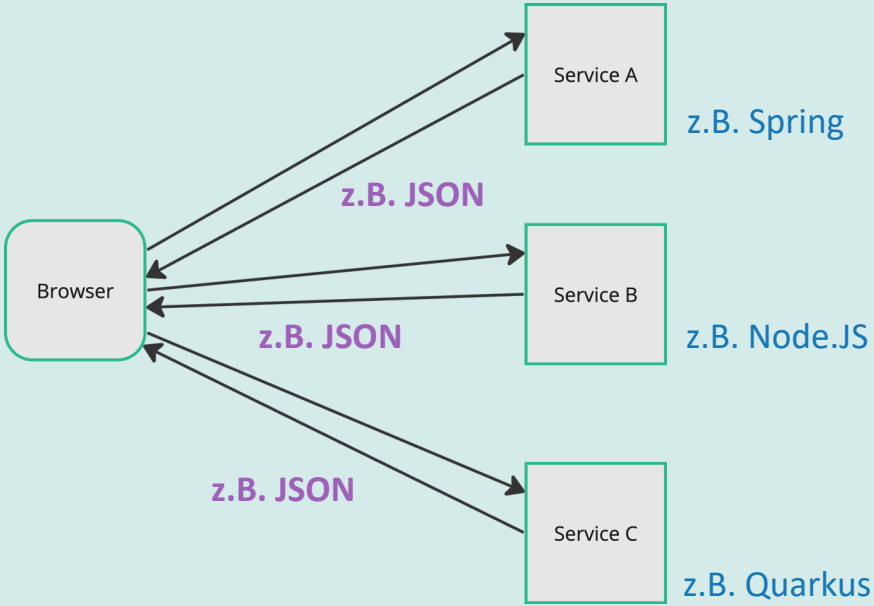
BEISPIEL: INITIALER SEITENAUFRUF

- **Ermittlung von Daten**
- SPA: HTTP / REST / GraphQL API



BEISPIEL: INITIALER SEITENAUFRUF

- **Ermittlung von Daten**
- SPA: HTTP / REST / GraphQL API
- Frontend-Anwendung bekommt **Daten**

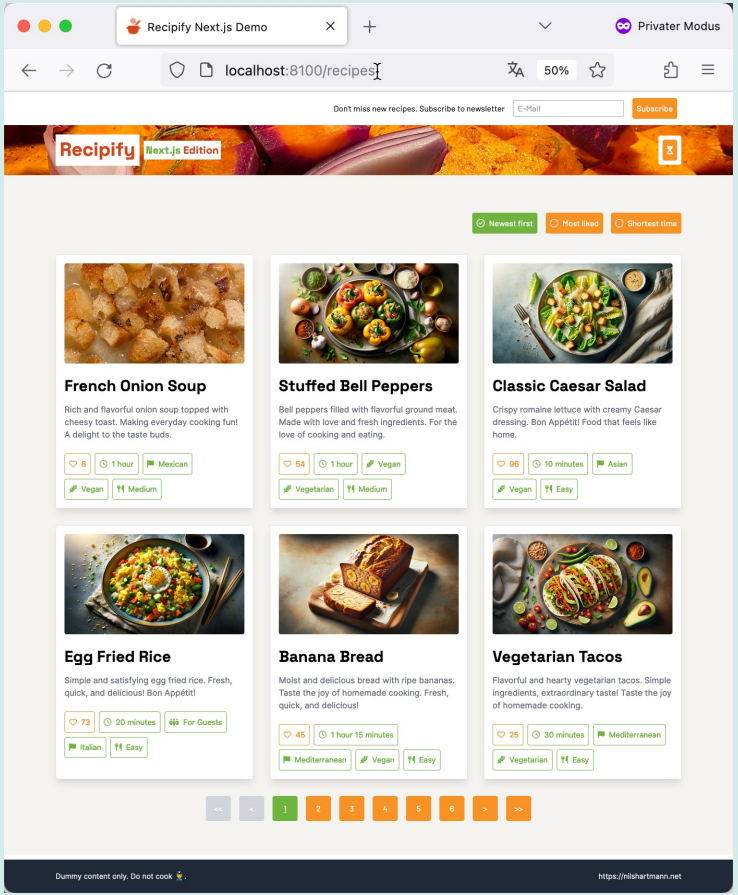


BEISPIEL: INITIALER SEITENAUFRUF

- **Beispiel: initialer Seitenaufruf**

Code

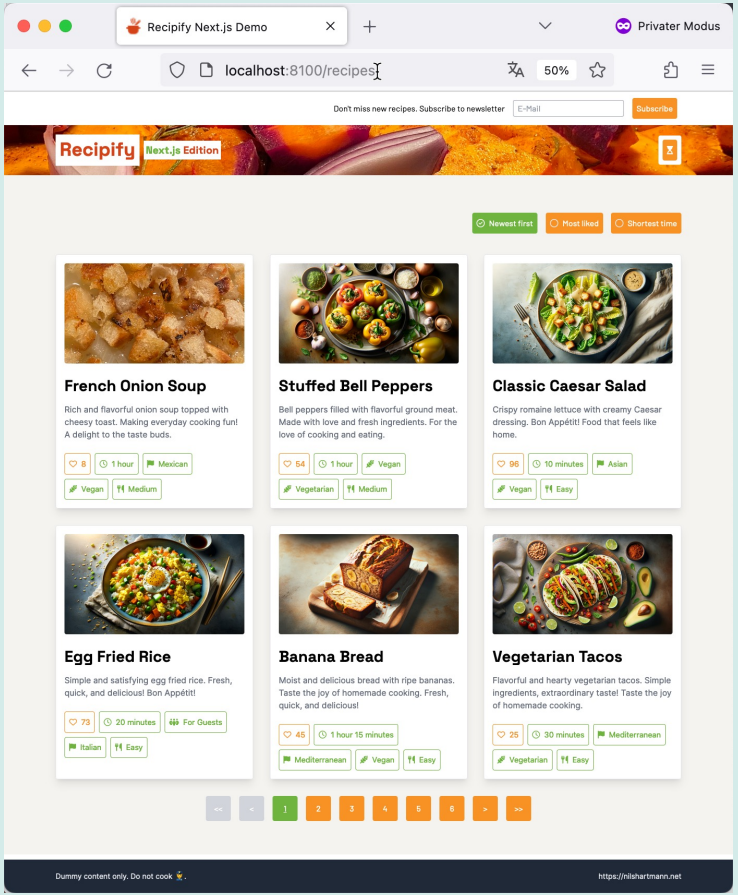
- SPA und Fullstack: JavaScript
- HTMX: Template-Sprache im Backend
  - Beispiel: Spring WebMVC + Thymeleaf



BEISPIEL: SEITE AKTUALISIEREN

- Beispiel: Seite aktualisieren (Sortierung)

Demo: Next.js

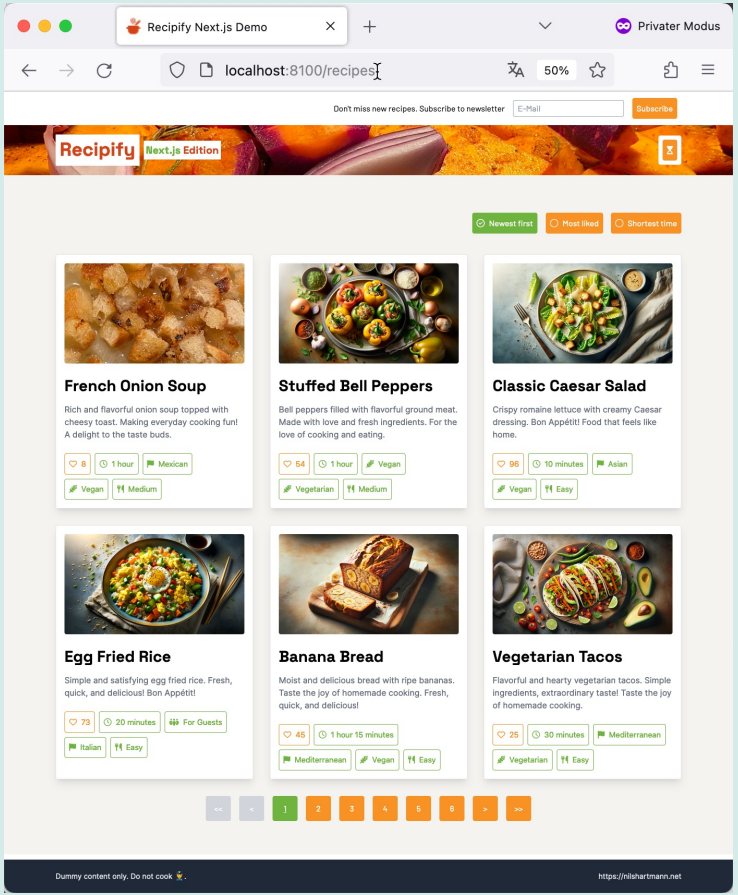


BEISPIEL: SEITE AKTUALISIEREN

- **Beispiel: Seite aktualisieren (Sortierung)**

Zur Laufzeit

- SPA: Daten
- Fullstack: HTML / UI für Ausschnitt
- HTMX: HTML für Ausschnitt



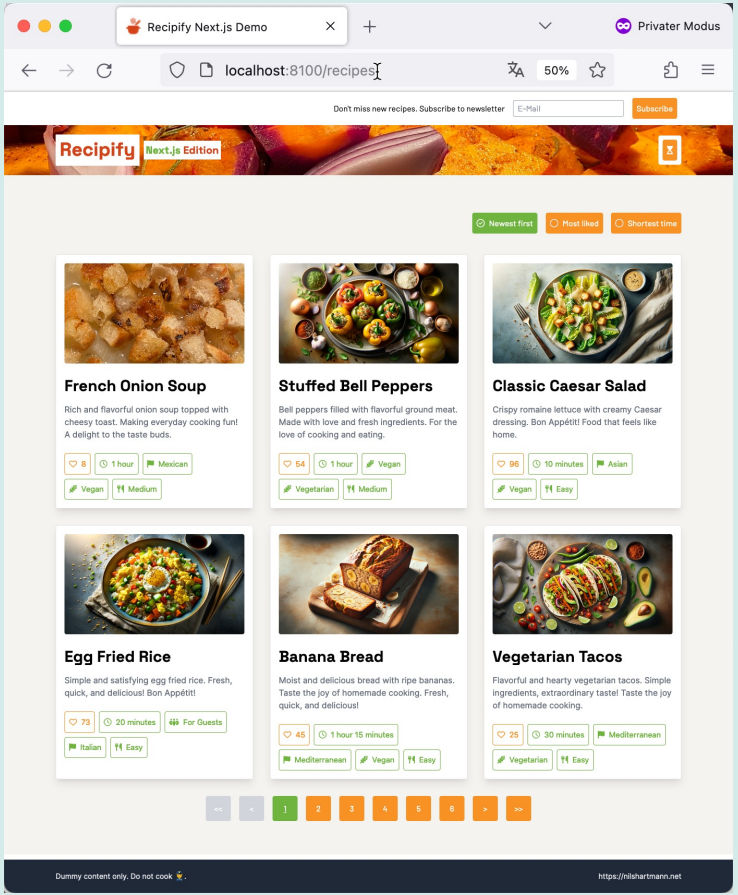
BEISPIEL: SEITE AKTUALISIEREN

- **Beispiel: Seite aktualisieren (Sortierung)**

HTMX

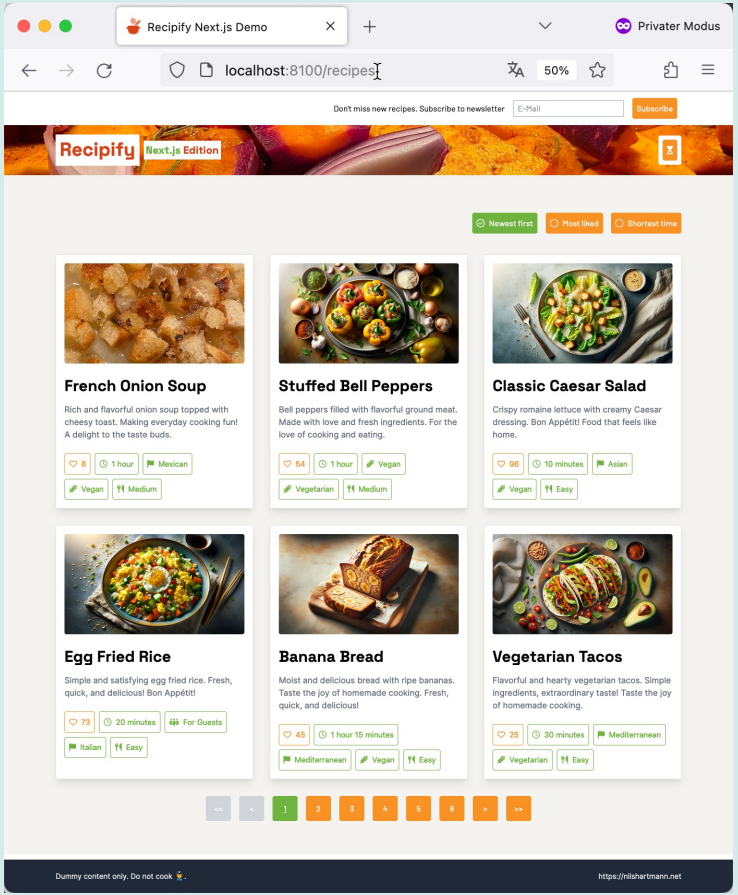
- Trigger auf (a)-Element
- Verhindert das "klassische" Navigieren
- Lädt neues HTML, ersetzt "main"

```
<a
   th:hx-get="@{/recipes(orderBy=time)}"
   hx-swap="outerHTML"
   hx-target="main"
>
   Time
</a>
```



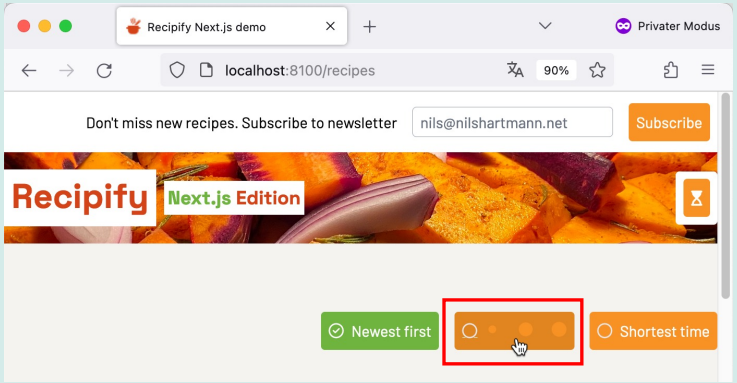
BEISPIEL: SEITE AKTUALISIEREN

- **Beispiel: Seite aktualisieren (Sortierung)**
- Warum eigentlich kein Full-Page-Reload? 🤔
- Demo: React SPA Newsletter



BEISPIEL: SEITE AKTUALISIEREN

- **Beispiel: Seite aktualisieren**
- Warum eigentlich kein Full-Page-Reload? 🤔
- Demo: NextJS Loading Indikator



BEISPIEL: SEITE AKTUALISIEREN

- **Beispiel: Loading Indikator**
- Next.JS/React: Transition

```
export function OrderButton({ orderBy }) {
  const [pending, startTransition] = useTransition();
  const router = useRouter();
  const href = "/recipes?orderBy=" + orderBy;

  const handleClick = (e) => {
    startTransition(() => {
      router.push(href);
    });
  };

  return (
      pending ? <LoadingIndicator /> :
        <Link href={href} onClick={handleClick}>
          {orderBy}
        </Link>

  );
}
```

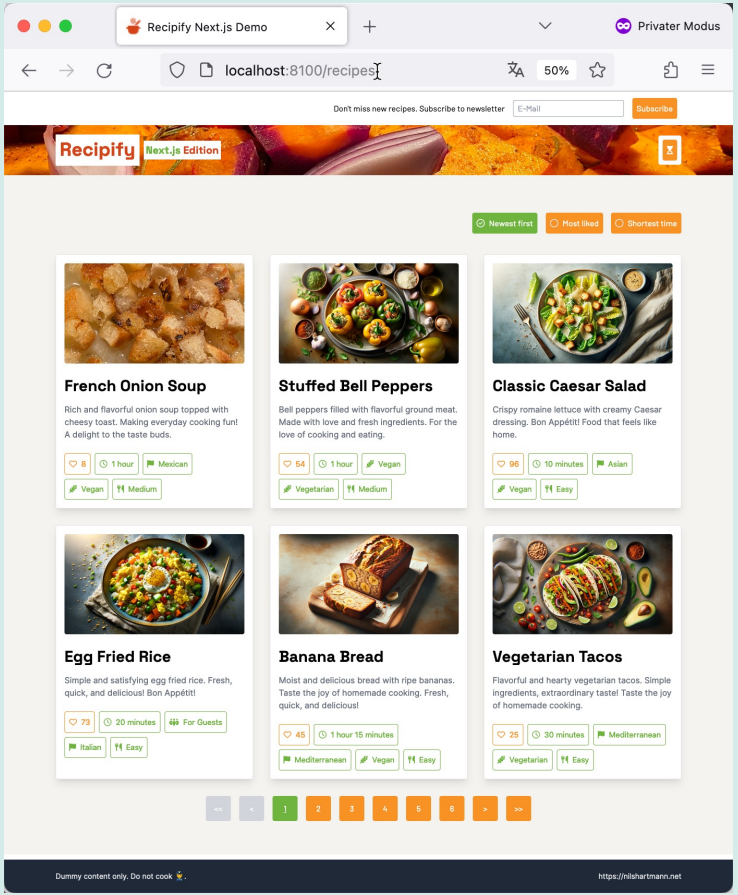
BEISPIEL: SEITE AKTUALISIEREN

- **Beispiel: Loading Indikator**
- HTMX: CSS-Klasse wird während Request an markiertes Element gehängt
- Flexibilität?

```
<a
  th:hx-get="@{/recipes(orderBy=time)}"
  hx-swap="outerHTML"
  hx-target="main"
  hx-indicator="#searchIndicator"
>
  Time
</a>

<div id="searchIndicator" />
```

BEISPIEL: VERLINKBARE SEITEN

- **Beispiel: Verlinkbare Seiten**
- Sortierung in der URL
- Demo: React SPA



BEISPIEL: VERLINKBARE SEITEN

- **Beispiel: Verlinkbare Seiten**
- SPA (und Fullstack) Code arbeitet mit URL
- **Identischer** Code für initialen Seitenaufbau und für Neusortierung nach Klick

```
export default function RecipeListPage() {
  const { orderBy } = recipeListRoute.useSearch();
  const recipes = useGetAllRecipesQuery(0, orderBy);

  return (
    <main>
      <RecipeListNavBar />
      <RecipeList recipes={recipes} />
    </main>
  );
}
```

BEISPIEL: VERLINKBARE SEITEN

- **Beispiel: Verlinkbare Seiten**
- HTMX: Trigger definieren und URL umschreiben, um "SPA Verhalten" zu bekommen

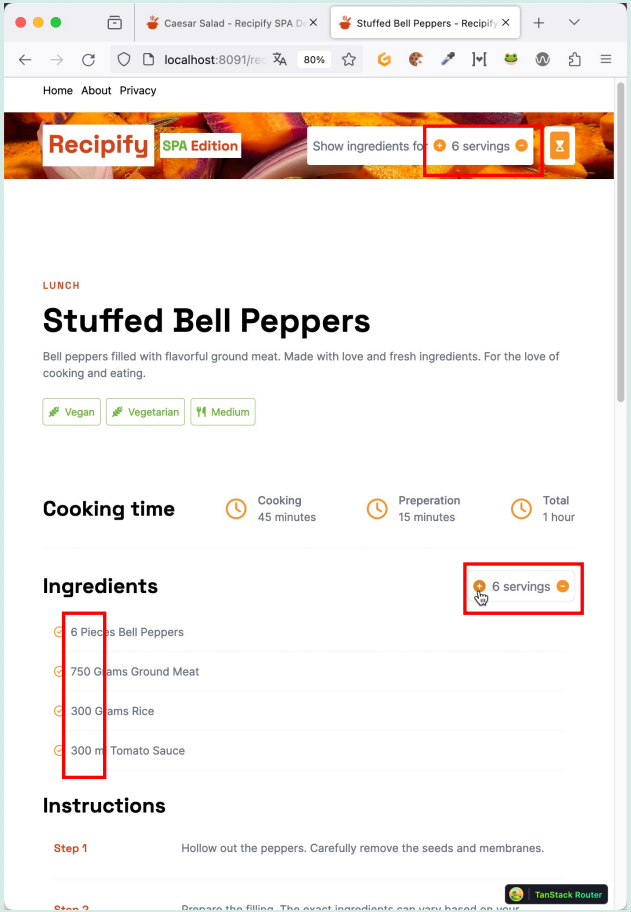
```
<a
   th:hx-get="@{/recipes(orderBy=time)}"
   th:hx-push-url="@{/recipes(orderBy=time)}"
   hx-swap="outerHTML"
   hx-target="main"
>
   Time
</a>
```

- **Beispiel: Verlinkbare Seiten**
- HTMX: **Zwei** Endpunkte erforderlich
  - Endpunkt **eins** für initiale, vollständige Seite (mit oder ohne Sortierung)

- **Beispiel: Verlinkbare Seiten**
- HTMX: **Zwei** Endpunkte erforderlich
  - Endpunkt **eins** für initiale, vollständige Seite (mit oder ohne Sortierung)
  - Endpunkt **zwei** für den Trigger zum Sortieren auf dem Link (nur Austausch des main-Elementes)

BEISPIEL: CLIENT-ZUSTAND

- **Beispiel: Client-Zustand**
- Demo: React SPA "Servings"

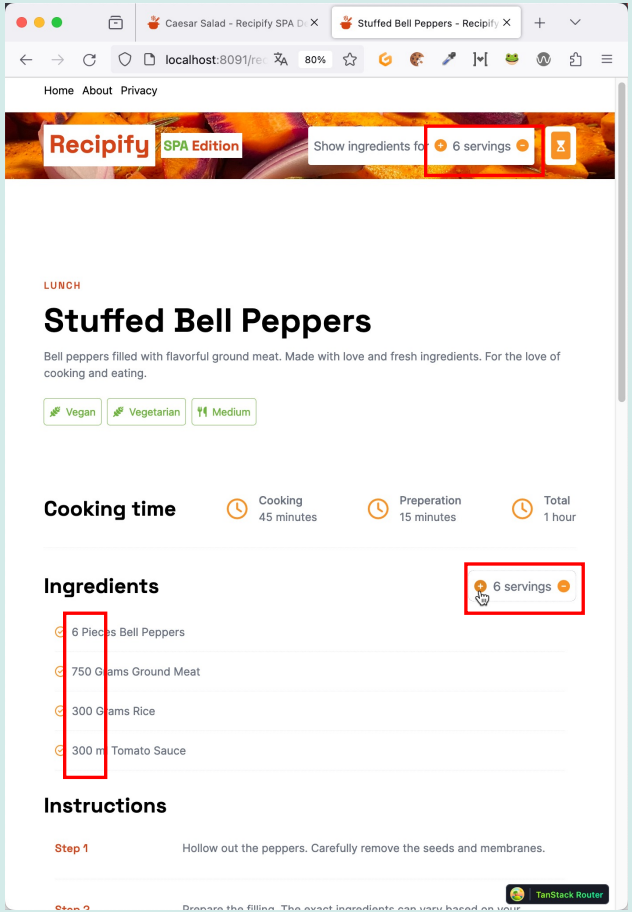


BEISPIEL: CLIENT-ZUSTAND

- **Beispiel: Client-Zustand**

Zur Laufzeit

- SPA: Nur neu rendern auf dem Client
- Fullstack: Nur Client und/oder Server
- HTMX: Nur Client und/oder Server



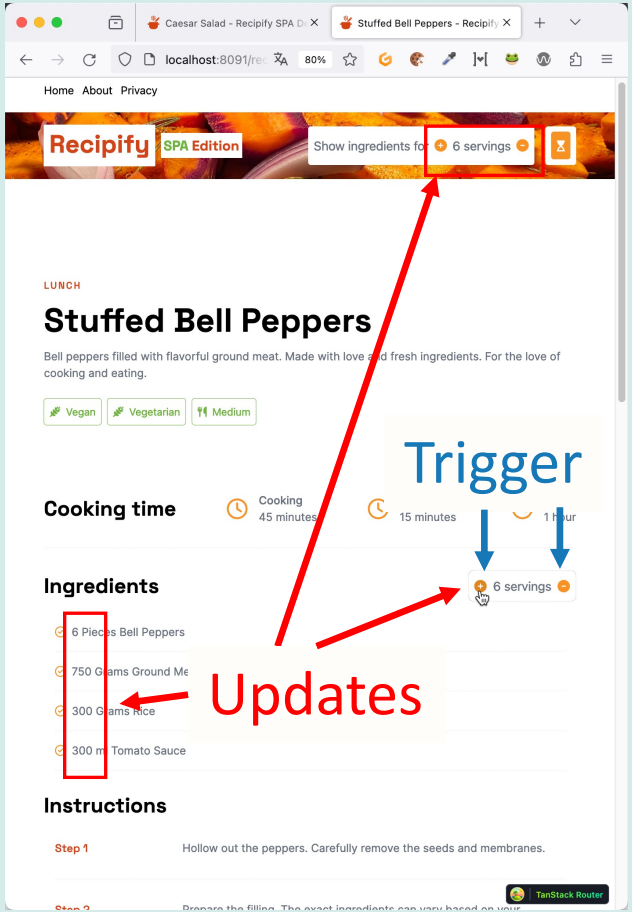
BEISPIEL: CLIENT-ZUSTAND

- **Klassisches Beispiel für globalen, nur Client-seitigen State**
- Daten werden global im Client gehalten
- Jede Komponente darf die Daten ändern und lesen
  - vergleichbar mit UI-Service-Layer-Zugriff
- Komponenten können wiederverwendet werden

```
export default function ServingsWidget() {
  const servingsStore = useServingsStore();
  return (
    <div>
      <i
        onClick={() => servingsStore.increaseServings()}
      />
        {servingsStore.servings} servings
      <i
        className={servingsStore.servings < 2 ? "Disabled": ""}
        onClick={() => servingsStore.decreaseServings()}
      />
    </div>
  );
}
```

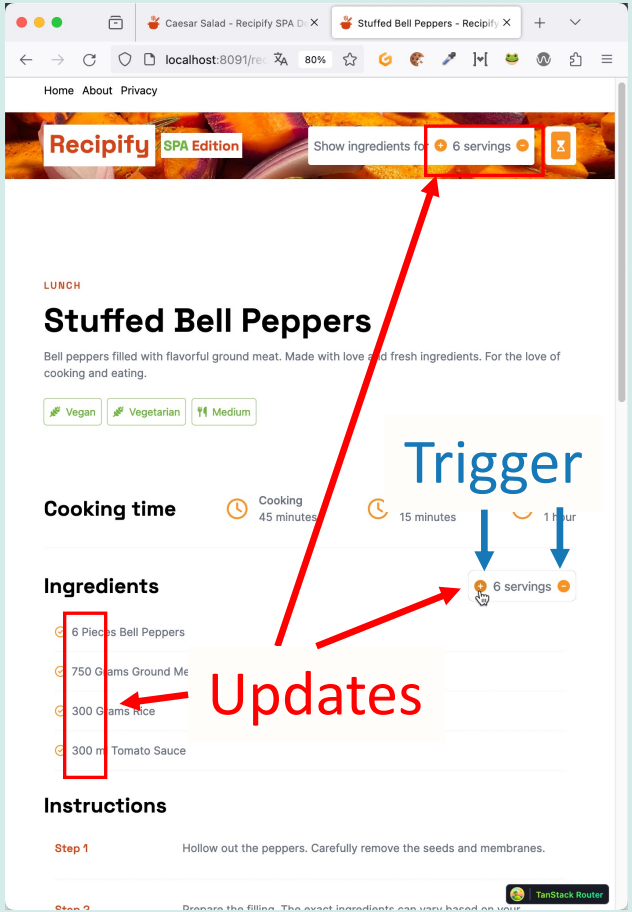
BEISPIEL: CLIENT ZUSTAND

- **HTMX Option 1:**
- Entweder: Client-seitig per JavaScript
  - Ohne HTMX
  - JavaScript-Schnipsel einstreuen
  - Code-Probleme wie vor zehn Jahren



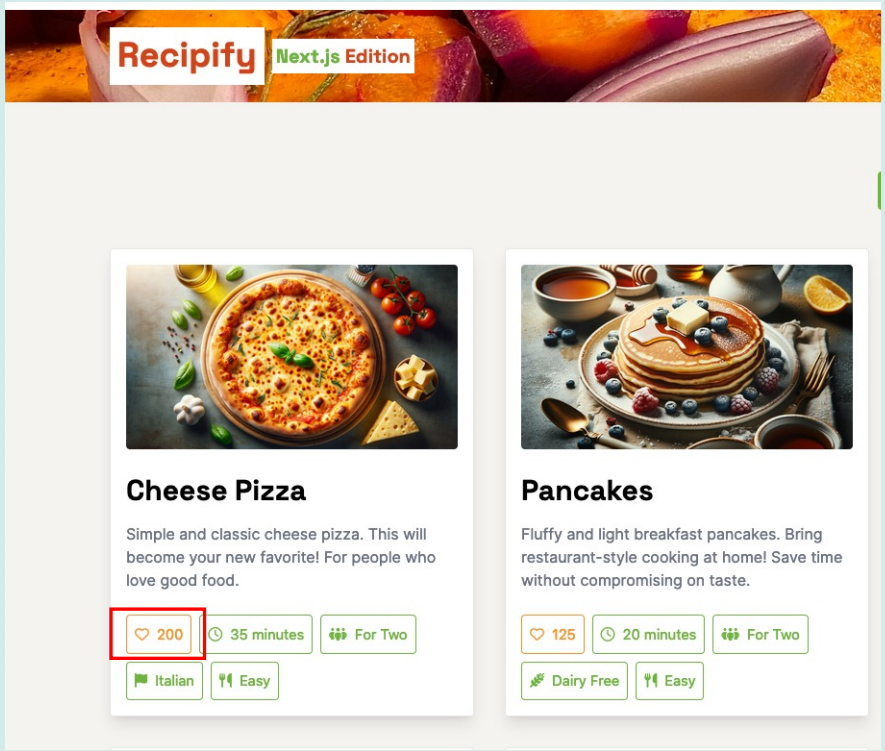
BEISPIEL: CLIENT ZUSTAND

- **HTMX Option 2: Serverseitig**
  - Klick auf +/-Button ist ein "Trigger"
  - Lädt neue UI
- Problem:
  - Aktualisierung an mehreren Stellen
  - Das ginge mit "Out-of-Band"
  - Sogar über "Seiten"-Wechsel hinaus
  - Wo merken wir uns den Zähler?
  - Wird schnell unübersichtlich



BEISPIEL: DATEN ÄNDERUNGEN

- **Beispiel: Daten-Änderungen**
- Demo: Next.JS Like Button

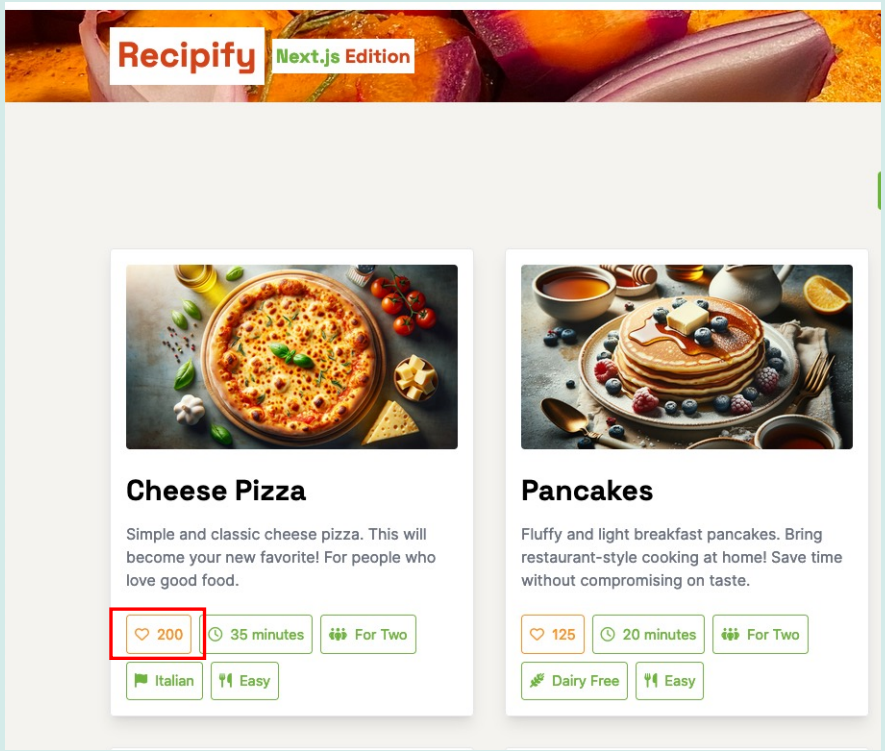


BEISPIEL: DATEN ÄNDERUNGEN

- **Beispiel: Daten-Änderungen**

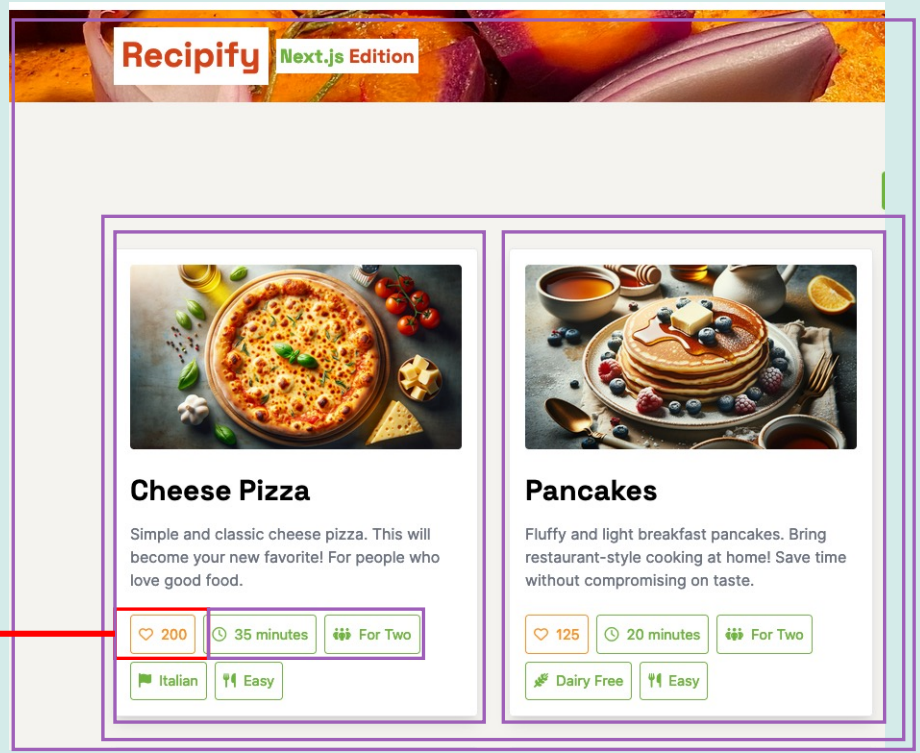
Zur Laufzeit

- SPA: HTTP Request, aktualisierter Like kommt zurück
- Fullstack: HTTP Request (ggf. proprietäres Protokoll, z.B. React Server Action)
- HTMX: Server Request (UI kommt zurück)



BEISPIEL: DATEN ÄNDERUNGEN

- **Beispiel: Daten-Änderungen**
- Like-Button ist "Client-Komponente", da sie interaktiv ist (anklickbar)
- Kann aber transparent mit Server-Komponenten gemischt werden



BEISPIEL: DATEN ÄNDERUNGEN

- **Beispiel: Daten-Änderungen**
- Like-Button ist "Client-Komponente", da sie interaktiv ist (anklickbar)
- Kann aber transparent mit Server-Komponenten gemischt werden
- Nur der JS-Code der Client-Komponente kommt in den Browser!

```
function RecipeCard({ recipe }: { recipe: RecipeDto }) {
  return (
    <section>
      <H1>{recipe.title}</H1>
      <RecipeCategories>
        <Likes />
        <CookingTime />
      </RecipeCategories>
    </section>
  );
}
```

Client Komponente

Server Komponenten

Auswahl-

Kriterien

AUSWAHLKRITERIEN

- **Wie interaktiv ist meine Anwendung eigentlich?**
  - Tipp: nicht unterschätzen!

AUSWAHLKRITERIEN

- **Habe ich mit (viel) JavaScript zur Laufzeit ein Problem?**
  - Das ist nicht für jede Anwendung der Fall!

AUSWAHLKRITERIEN

- **Habe ich mit (viel) JavaScript in der Entwicklung ein Problem?**
  - Dann eher kein JavaScript-basierter Ansatz

AUSWAHLKRITERIEN

- **Wie gefällt mir die Entwicklung**
  - Sprache
  - Tooling, Entwicklungsumgebung etc.
  - Bei HTMX beachten: wie sieht die Template-Sprache auch? Gefällt mir das?

AUSWAHLKRITERIEN

- **Wie gefällt mir die Entwicklung**
  - Sprache
  - Tooling, Entwicklungsumgebung etc.
  - Bei HTMX beachten: wie sieht die Template-Sprache auch? Gefällt mir das?
- **Ist Laufzeit nicht das wichtigere Kriterium?**
  - Ja aber!
  - Wenn wir kein Bock haben, die Anwendung zu bauen, wird das ja auch nichts
  - Hier muss man vielleicht eine Balance finden

AUSWAHLKRITERIEN

- **Komponentenmodell**
  - Anwendungen sind komplex
  - Mit Komponenten kann Komplexität aufgeteilt werden (wie im Backend)
  - Unterstützt meine gewünschte Technologie das?
    - SPA und Fullstack (JavaScript): Ja
    - HTMX: kommt auf das Backend an

Die Qual der Wahl

Fazit

FAZIT

- **Laufzeit-Sicht: Fullstack + HTMX vorsichtig vergleichbar**

FAZIT

- **Entwicklung: Fullstack + SPA vorsichtig vergleichbar**
- HTMX fällt hier deutlich raus
- Insbesondere React-Ansätze (alles JavaScript, selbst "Templates") unterscheidet sich deutlich von HTMX (Backend-Sprache, Template-Sprache, JavaScript)

FAZIT

- **Meine persönliche Präferenz**
- Wenn es geht, SPA machen: guter Kompromiss sowohl zur Laufzeit + Entwicklung

FAZIT

- **Meine persönliche Präferenz**
- Wenn es geht, SPA machen: guter Kompromiss sowohl zur Laufzeit + Entwicklung
- Dann Fullstack, weil der Ansatz konsequente Weiterentwicklung ist

FAZIT

- **Meine persönliche Präferenz**
- Wenn es geht, SPA machen: guter Kompromiss sowohl zur Laufzeit + Entwicklung
- Dann Fullstack, weil der Ansatz konsequente Weiterentwicklung ist

FAZIT

- **HTMX wirkt auf mich wie Technik von vor zehn Jahren**
  - wir bauen aber heute ganz andere Anwendungen als damals:
    - ganz andere fachlicher Komplexität
    - viel höhere Ansprüche

Quelle: "Modern frontends with htmx", Wim Deblauwe, 2023 (Hervorherbungen von mir)

FAZIT

Wenn ihr nur eins aus diesem Vortrag mitnehmen wollt:

FAZIT

Wenn ihr nur eins aus diesem Vortrag mitnehmen wollt:

Frontend-Entwicklung bitte, bitte ernstnehmen!

Das ist nicht "weniger" als Backend-Entwicklung

(Wenn ihr kein Bock auf Frontend habt, dann lasst das doch einfach die Kolleg:innen machen 😇)

Frontend-Entwicklung ...

Frontend?

Vielleicht doch nicht so trivial

NILS HARTMANN
https://nilshartmann.net

Vielen Dank!

Slides: https://react.schule/jax-2024

Fragen & Kontakt: nils@nilshartmann.net

Twitter: @nilshartmann