

Web Dev NextJs



Michael

Fullstack



Joy

Frontend/Design



Max

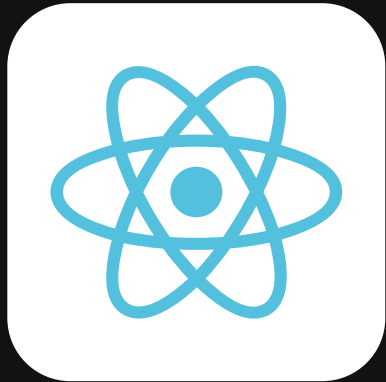
Backend

Was ist NextJs

React is a library. [...] it doesn't prescribe how to do routing and data fetching. [...] we recommend a full-stack React framework like Next.js ...

99

Was ist NextJs



Full-stack

Viele Features
Flexibel

SEO & Images
Optimiert

Was bietet NextJs



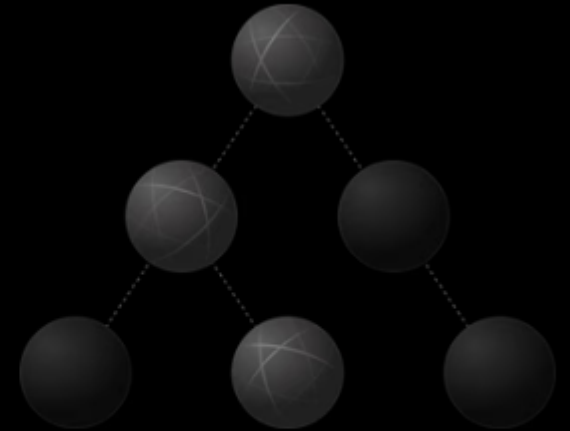
Built-in Optimizations

Automatic Image, Font, and Script Optimizations for improved UX and Core Web Vitals.



Dynamic HTML Streaming

Instantly stream UI from the server, integrated with the App Router and React Suspense.



React Server Components

Add components without sending additional client-side JavaScript. Built on the latest React features.

Was bietet NextJs

Data Fetching

Make your React component async and await your data. Next.js supports both server and client data fetching.

CSS Support

Style your application with your favorite tools, including support for CSS Modules, Tailwind CSS, and popular community libraries.

Client and Server Rendering

Flexible rendering and caching options, including Incremental Static Regeneration (ISR), on a per-page level.

Server Actions

Run server code by calling a function. Skip the API. Then, easily revalidate cached data and update your UI in one network roundtrip.

Route Handlers

Build API endpoints to securely connect with third-party services for handling auth or listening for webhooks.

Advanced Routing & Nested Layouts

Create routes using the file system, including support for more advanced routing patterns and UI layouts.

Middleware

Take control of the incoming request. Use code to define routing and access rules for authentication, experimentation, and internationalization.

Next.js 15

The power of full-stack to the frontend. Read the release notes.



Geschichte NextJs



2016



2019



2020



2023

Einordnung NextJs



★ 132k

NextJs



★ 57k

NuxtJs



★ 51k

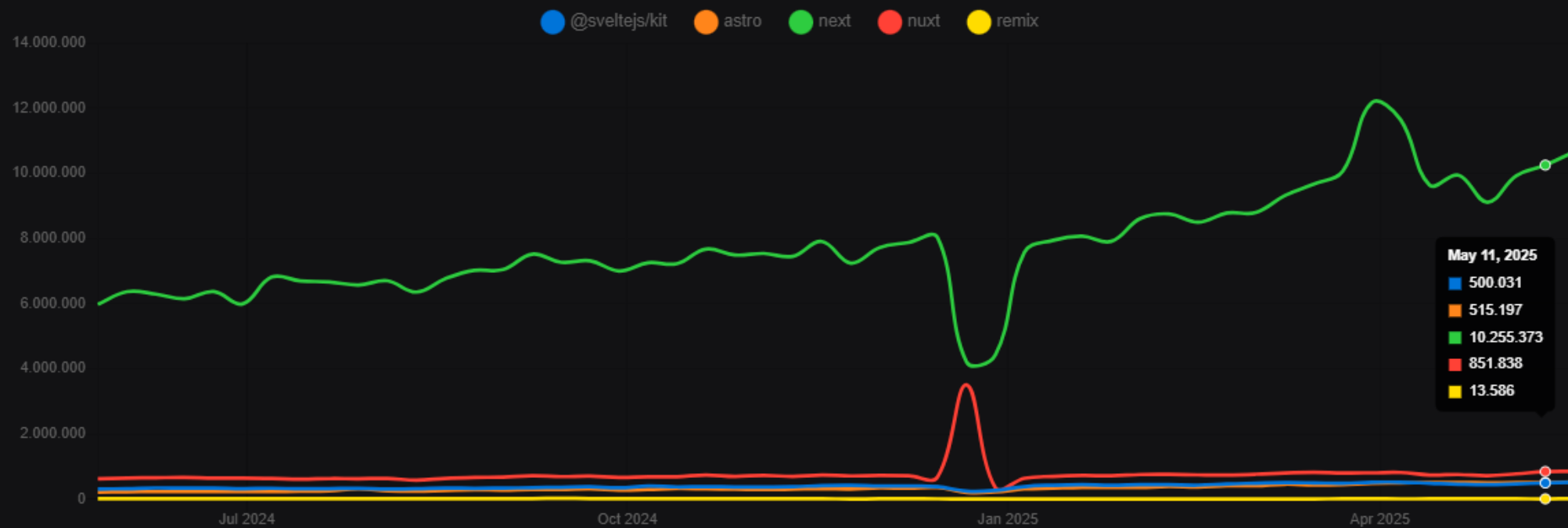
Astro



★ 19k

SvelteKit

Einordnung NextJs



(Markt) Nutzung

<https://nextjs.org/showcase>

<https://vercel.com/templates/next.js>



Setup

Node 18+

```
npx create-next-app@latest
```

Kein pnpm



Fundamentals

Live

Fragen?

Unser Beispiel

Ressourcen für Übungen



git clone <https://github.com/dopeshot/web-dev.git>

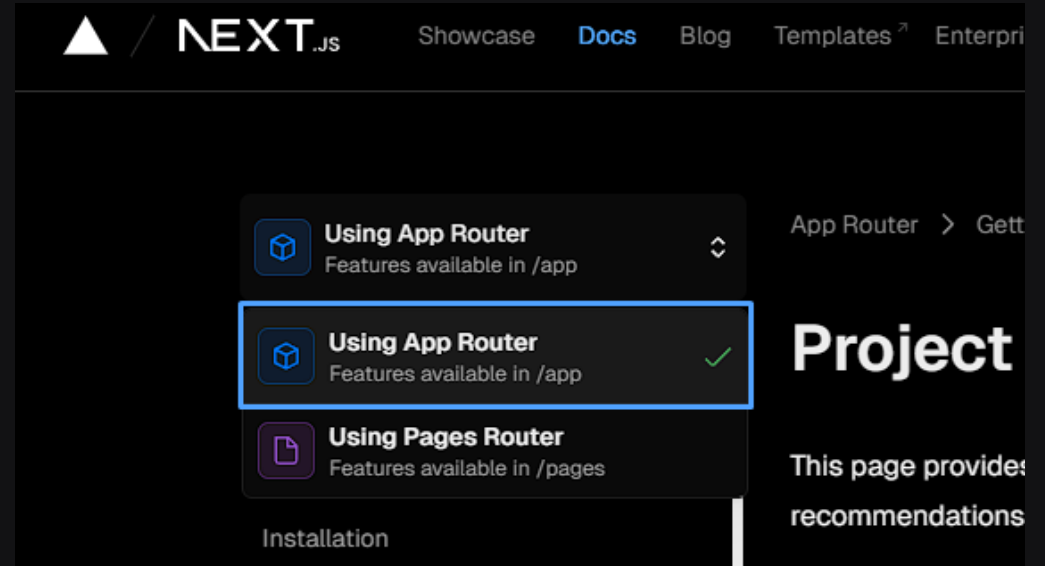
git clone <git@github.com:dopeshot/web-dev.git>



nextjs.org

Routing

mit dem App Router!



Wie funktioniert **File-Based Routing**?



app/



app/faq/



<http://localhost:3000/faq>



app/faq/page.tsx

Wie sieht eine `page.tsx` aus?

```
// page.tsx

export default function Beispielseite() {
  return (
    <main className="container">
      <h1>Beispielseite</h1>
    </main>
  )
}
```

`http://localhost:3000/vorlesung/informatik`



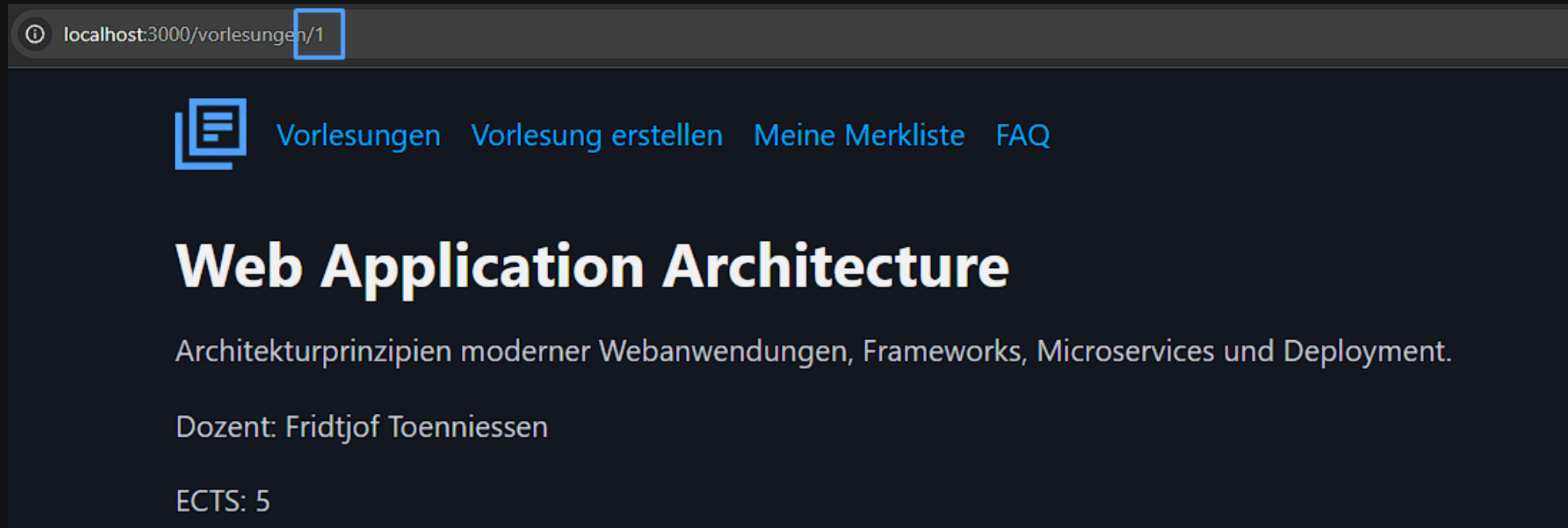
?

`http://localhost:3000/vorlesung/informatik`



`app/vorlesung/informatik/page.tsx`

Was tun, wenn wir die **Route nicht kennen?**



Dynamische Routen



/vorlesungen/[id]/page.tsx



/vorlesungen/123

```
// /vorlesungen/[id]/page.tsx

export default async function VorlesungDetailPage({
  params,
}: {
  params: Promise<{ id: string }>
}) {
  // id aus den URL Parametern lesen
  const id = (await params).id

  ... // mit id Inhalte über eine API fetchen und anzeigen
}
```

Fragen?

Linking

Seiten verlinken mit next/link.

Navigation mit `<Link>`

```
import Link from 'next/link'

<Link href="/faq"/>Faq</Link>
<Link href={` /vorlesungen/${id}`}>Zur Vorlesung</Link>
```

Vorteile <Link>

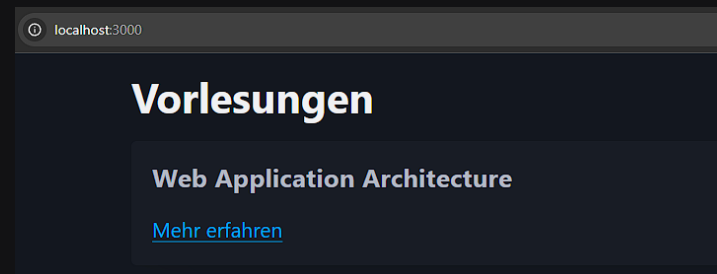
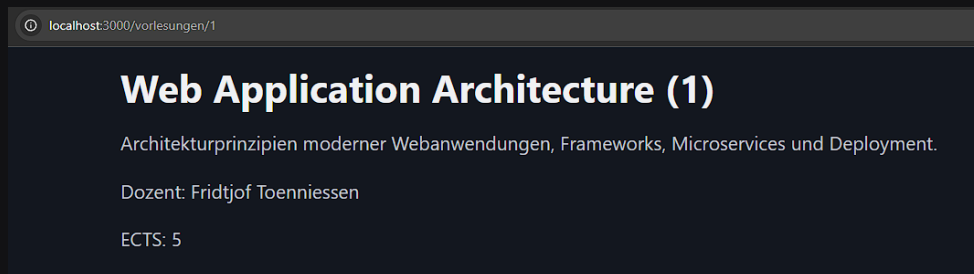
- ✓ Kein vollständiger Reload der Seite
- ✓ Schnelleres Laden durch Prefetching
- ✓ Bessere UX

Fragen?

Übungsaufgabe

Routing & Linking

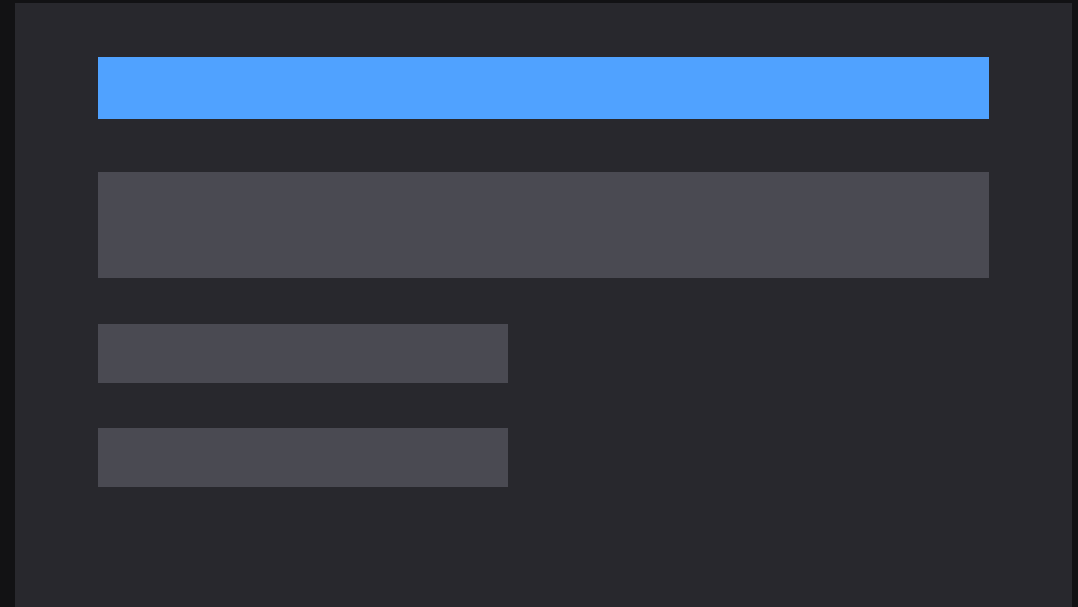
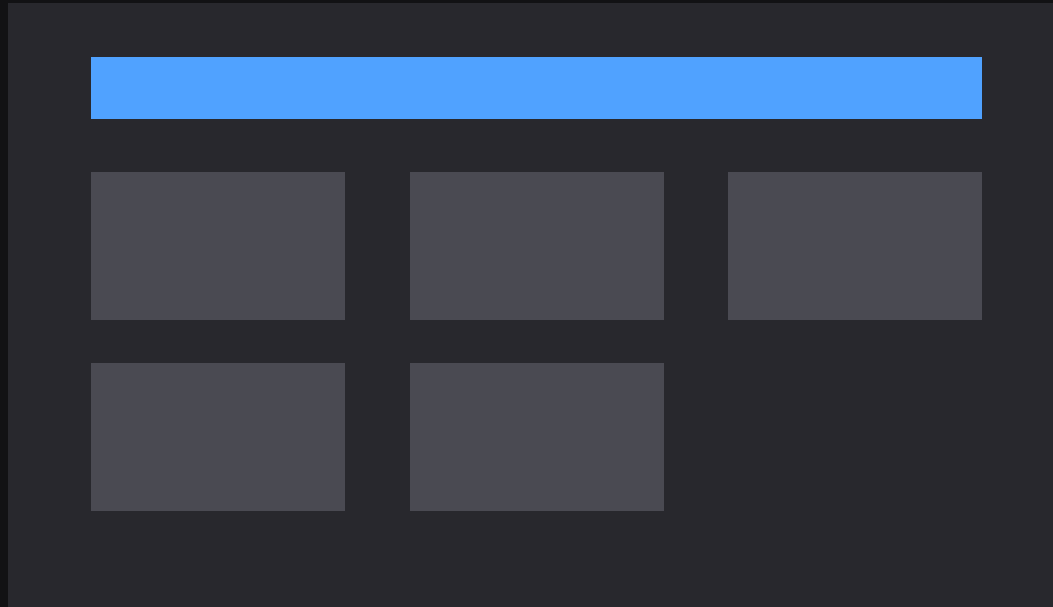
1. Wechsle den branch: `git checkout übung-1-routing-und-linking`
2. Erstelle die Detailseite für eine Vorlesung in `src/app/vorlesungen/[id]/page.tsx`. Nutze eine lokale Variable mit Beispieldaten, um die Detailansicht einer Vorlesung darzustellen.
3. Verlinke die Hauptseite `/src/app/vorlesungen/page.tsx` mit der Detailseite über einen Link.
4. 💡 Tipp: Sieh dir die Übersichtsseite an, um zu sehen, welche Felder angezeigt werden sollen.



Layout

Gemeinsame UI-Struktur.

Was ist ein **Layout**?



File-Based Struktur **Layout**



/page.tsx



/layout.tsx



vorlesungen/



vorlesungen/layout.tsx

Aufbau Layout



/



/page.tsx



/layout.tsx



vorlesungen/



vorlesungen/layout.tsx

```
export default function RootLayout({
  children,
}: {
  children: React.ReactNode
}) {
  return (
    <html>
      <body>
        <Navbar />
        <main>{children}</main>
      </body>
    </html>
  )
}
```


Vorteile **Layout**

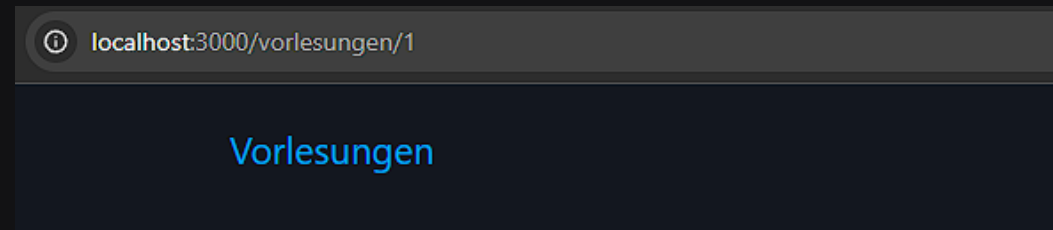
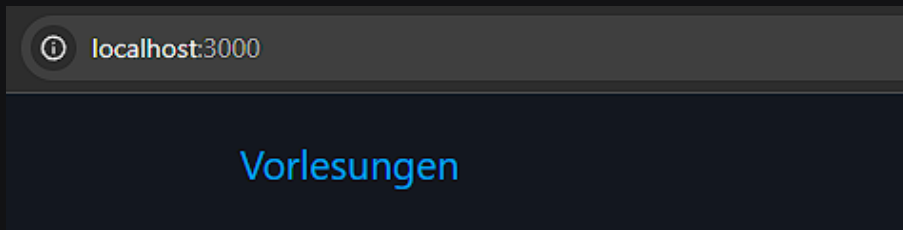
- ✓ Wiederverwendbar -> weniger Code Duplicate
- ✓ Persistenz → schneller navigieren, kein flackern beim animieren
- ✓ Zentrale Steuerung von Logik & UI-Elementen

Fragen?

Übungsaufgabe

Layout

1. Wechsle den branch: `git checkout übung-2-layout`
2. Erstelle eine neue Komponente für die Navbar `/src/app/components/Navbar.tsx`
Die Navbar soll einen Link zur Vorlesungs-Übersichtsseite (/) enthalten.
3. Füge die Navbar in das RootLayout ein `/src/app/layout.tsx`



React Server Components

Wie **rendert** NextJS?

Client Side Rendering

CSR

Server Side Rendering (React)

SSR

Static Site Generation

SSG

React Server Components

RSC

Client Side Rendering

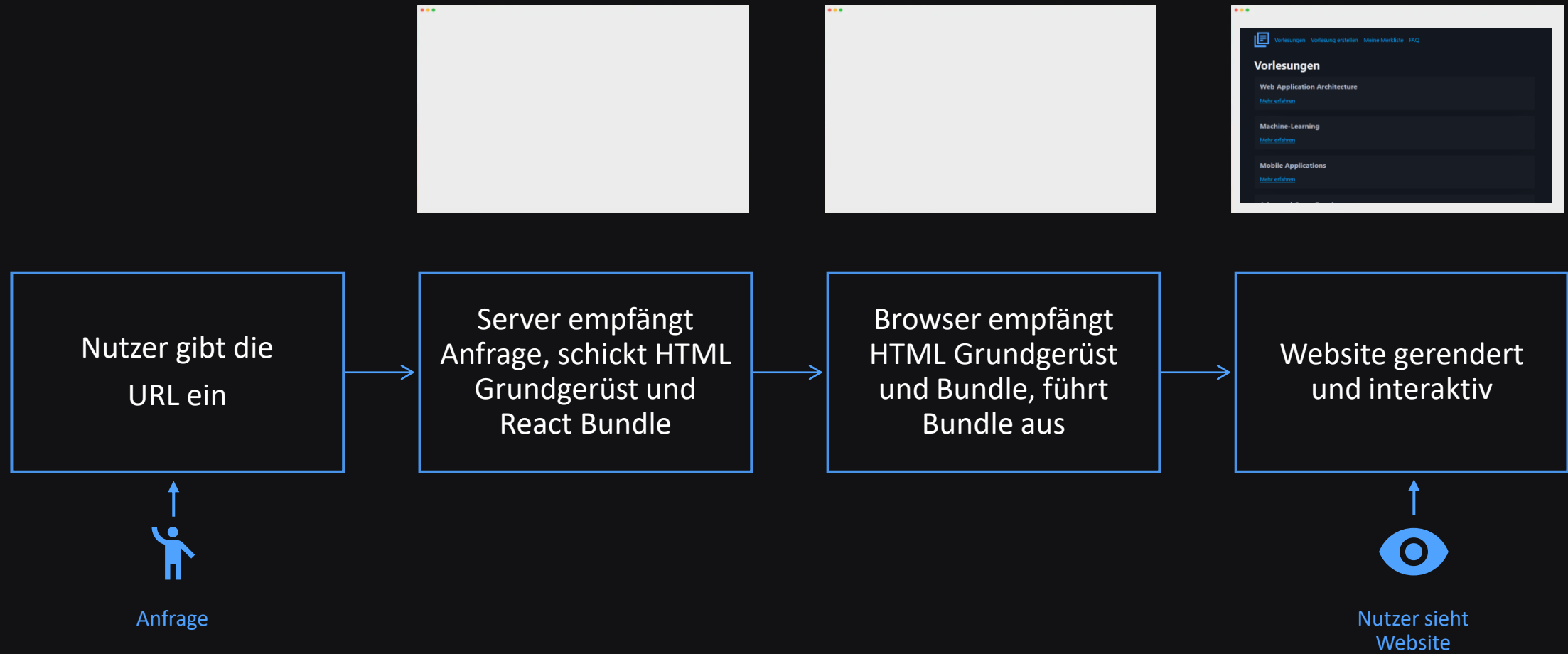
Die Webseite wird erst im Browser mit JavaScript aufgebaut.

JavaScript
Bundle



Server

Browser



Live

An echten Beispielen erklärt!

<https://wc-react-todo-app.netlify.app/>

Vorteile und Nachteile

Client Side Rendering

TODO LIST



Add Task

Add TODO

Title

Status

Incomplete



Add Task

Cancel



Client



Server

Geringe Serverlast

Einfaches Deployment

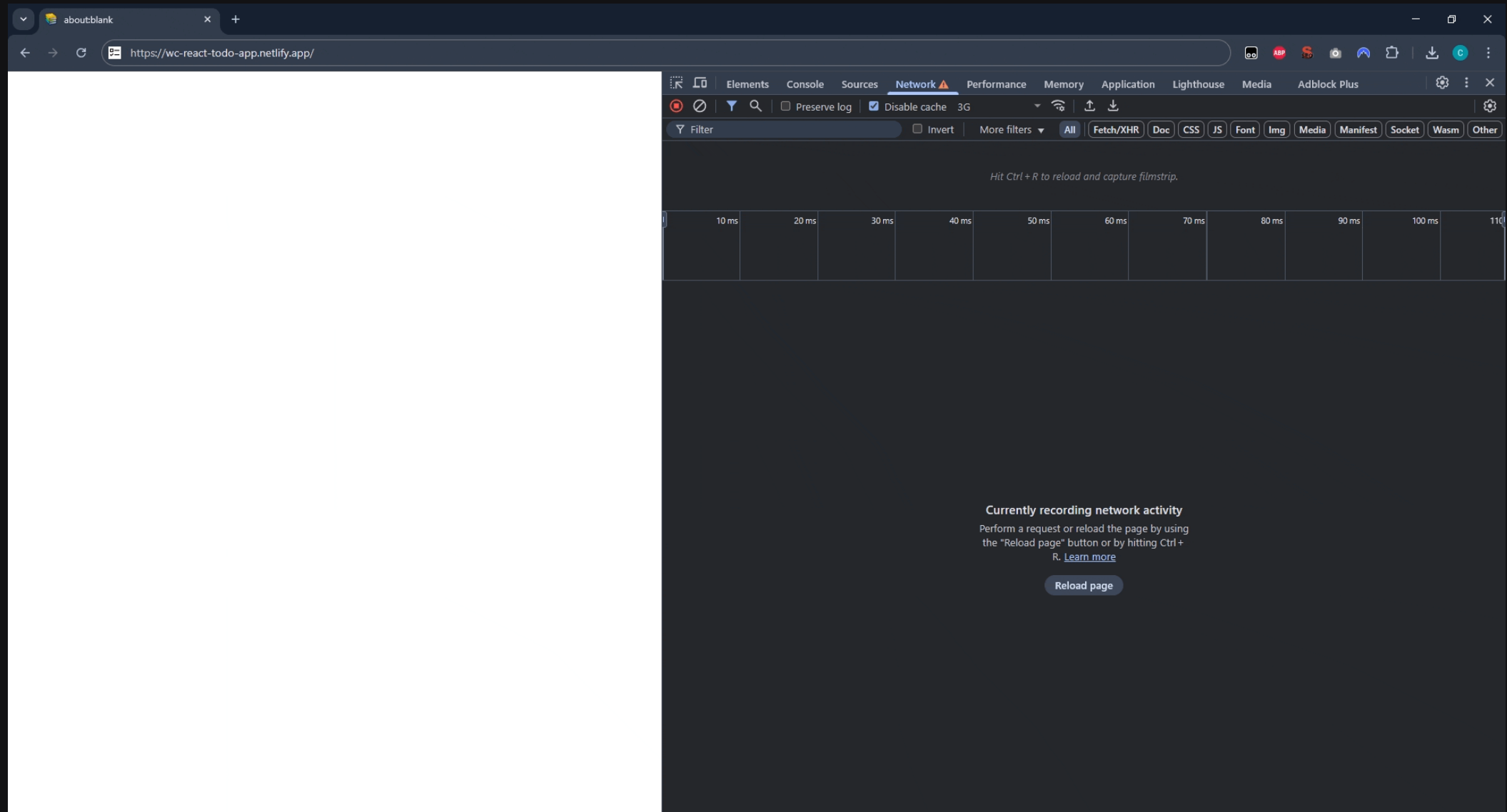
Kosteneffizient

Entwicklerfreundlich und leicht umzusetzen

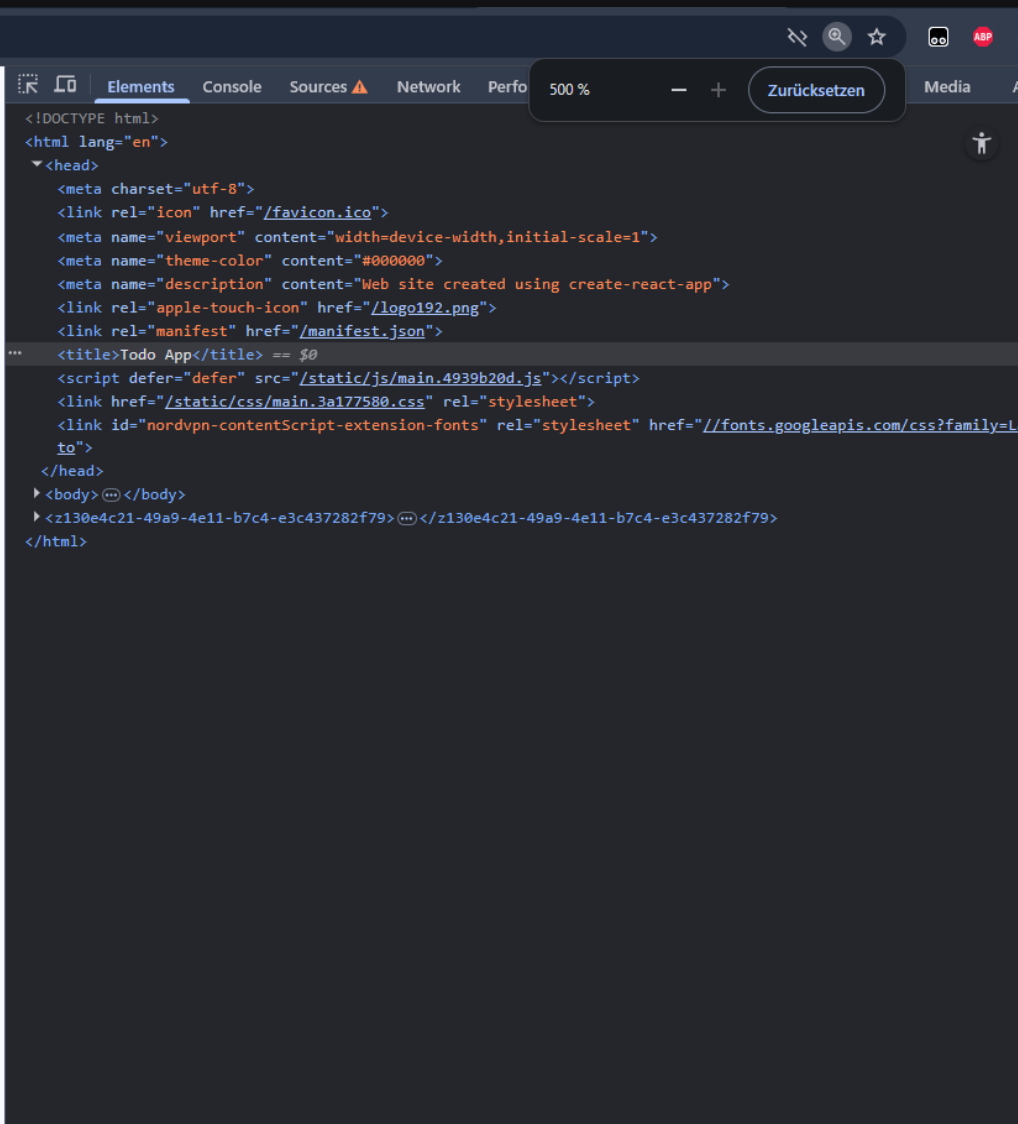
Was könnten **Nachteile** sein?

Client Side Rendering

Client Side Rendering Nachteile



You need to enable JavaScript to run this app.



Server Side Rendering

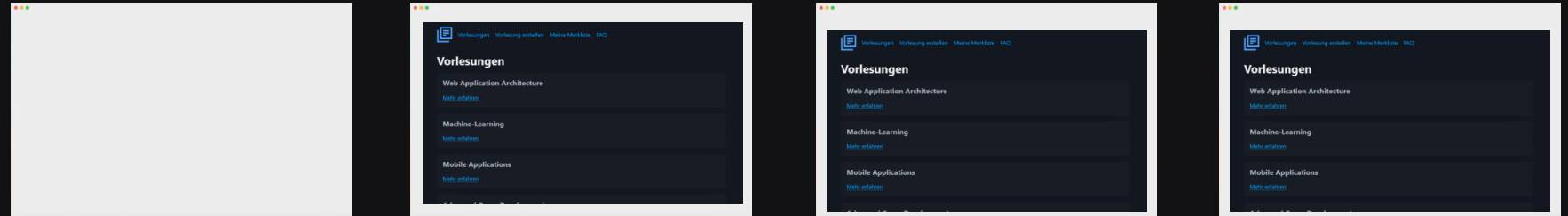
Die erste Darstellung der Seite wird auf dem Server gerendert – der Rest passiert im Browser.

Probleme Client Side Rendering

Langsamer First Contentful Paint

Schlechte SEO

Server Side Rendering



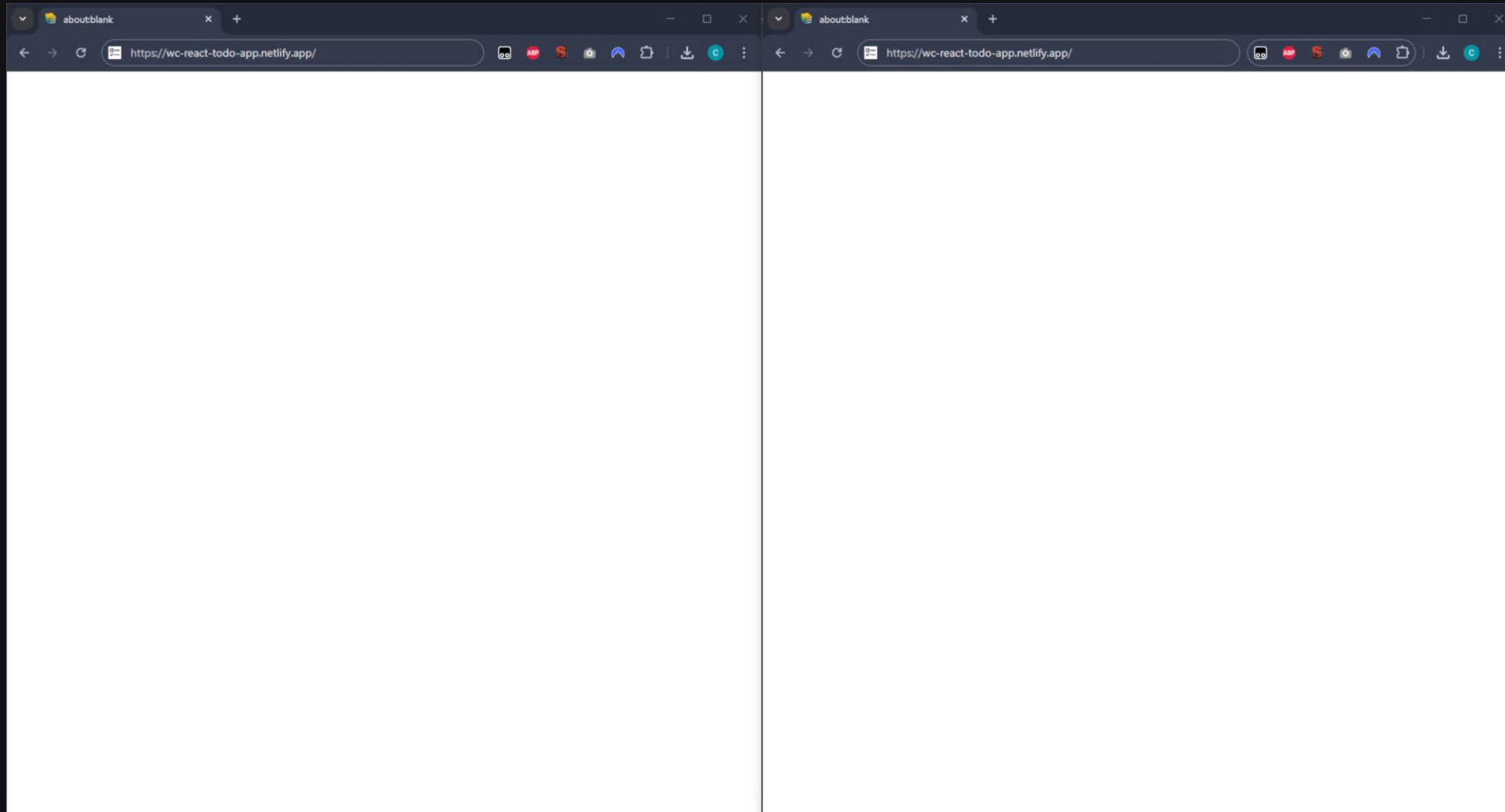
Live

An echten Beispielen erklärt!

Vorteile und Nachteile

Server Side Rendering

Server Side Rendering Vorteile



Server Side Rendering

Client Side Rendering

Server Side Rendering Vorteile

The screenshot displays a web browser window with the address bar showing `https://wc-react-todo-app.netlify.app`. The page content is a simple TODO application interface. At the top, the text "TODO LIST" is centered. Below it, there is a blue button labeled "Add Task" on the left and a dropdown menu labeled "All" on the right. A large light gray box contains the text "No Todos". The browser's developer tools are open on the right side, showing the "Elements" tab. The DOM tree reveals the following structure:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <link rel="icon" href="/favicon.ico">
    <meta name="viewport" content="width=device-width,initial-scale=1">
    <meta name="theme-color" content="#000000">
    <meta name="description" content="Web site created using create-react-app">
    <link rel="apple-touch-icon" href="/logo192.png">
    <link rel="manifest" href="/manifest.json">
    <title>Todo App</title>
    <script defer="defer" src="/static/js/main.4939b20d.js"></script>
    <link href="/static/css/main.3a177580.css" rel="stylesheet">
    <style id="_goober">
  </head>
  <body>
    <div id="root">
      <div class="container">
        <p class="title_title_mJ80Q">TODO List</p>
        <div class="app_app_wrapper__+aeJE">
          <div class="app_appHeader_N7YR4">
        </div>
          <div class="app_content_wrapper_Mm7EF" style="opacity: 1; transform: none;">
        </div>
        <div style="position: fixed; z-index: 9999; inset: 16px; pointer-events: none;">
      </div>
    </body>
  </html>
```

Was könnten **Nachteile** sein?

Server Side Rendering

Server Side Rendering Nachteile

Browser APIs werden nicht unterstützt

Doppelte Ausführung (Code läuft auf Server + Client)

Immer noch viel Code im Client

React Server Components

Komponenten die nur auf dem Server gerendert werden, nicht im Browser.

Probleme Server Side Rendering

Interaktivität kommt verzögert



Doppelte Ausführung

Immer noch viel Code im Client


Medium

Search


Write





JAMstack at Scale: 3 Powerful Approaches to Handle Thousands of Pages Efficiently


 Leon Neumann · 9 min read · Feb 6, 2025


3

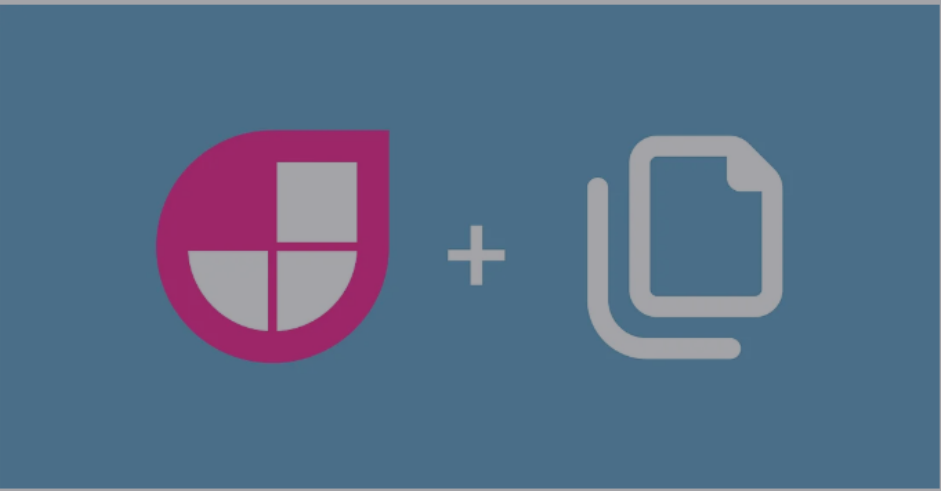








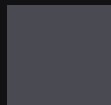




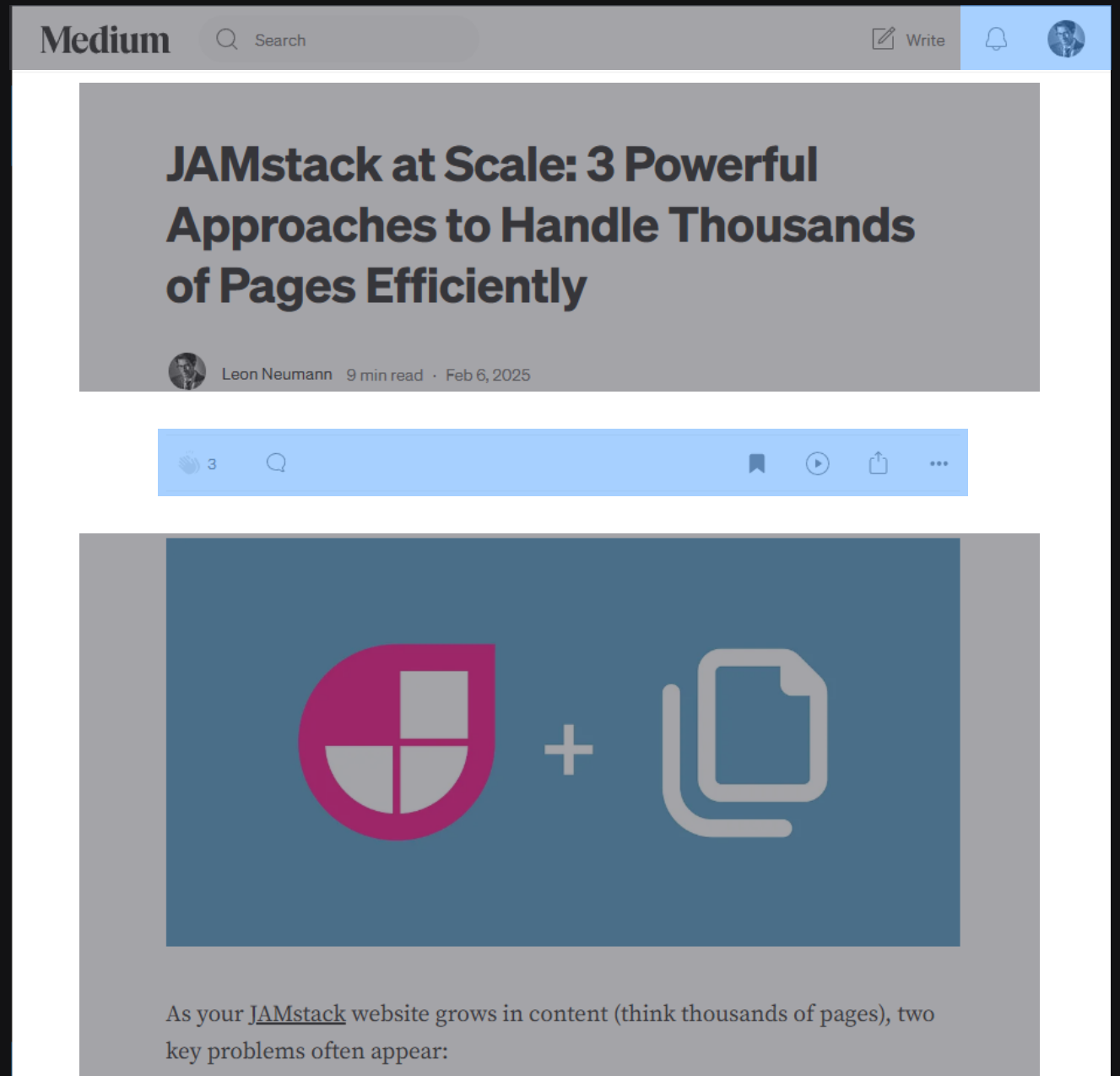
As your JAMstack website grows in content (think thousands of pages), two key problems often appear:



Client Component



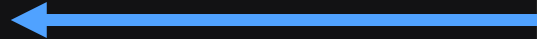
Server Component



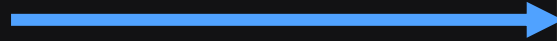


Server

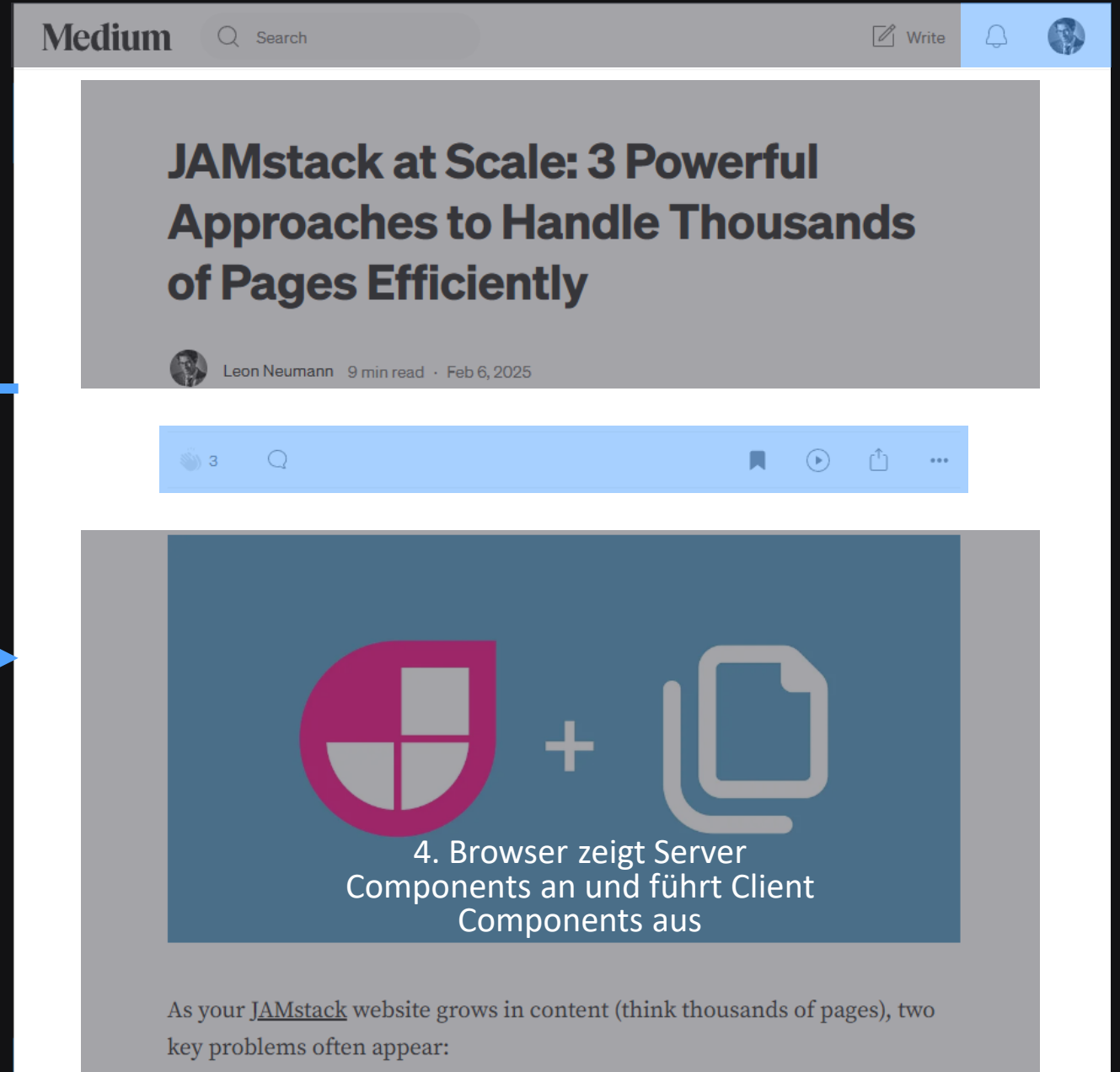
1. Neue Seite
anfragen



2. Server Component
rendert Seite



3. Seite mit restlichen
Client Components
schicken



React Server Components

Kein JavaScript wird an den Client geschickt

Sie sind **nur für die Darstellung**, interaktive Logik bleibt bei **Client Components**

Sie können direkt auf Datenbanken, APIs etc. zugreifen

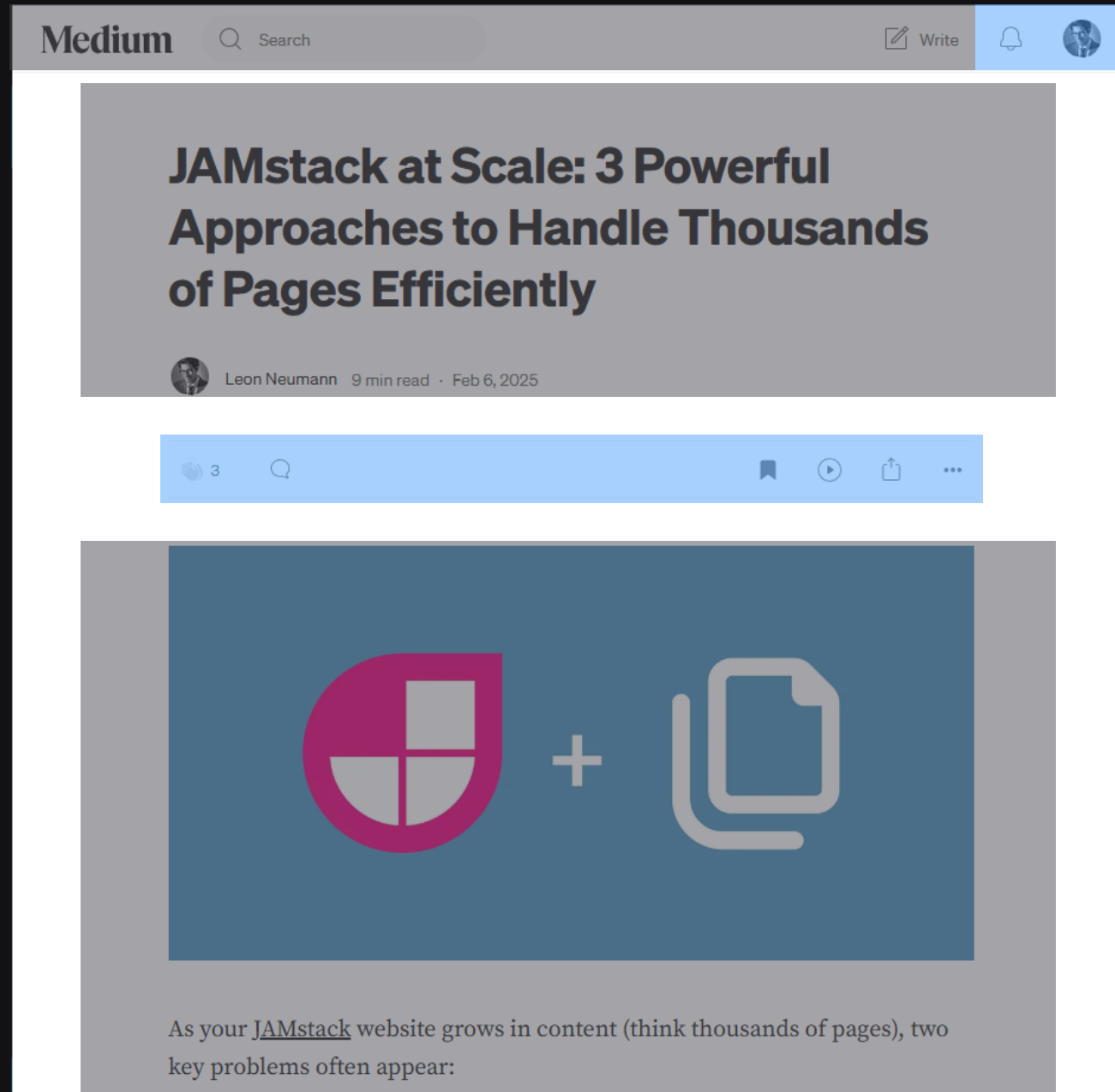
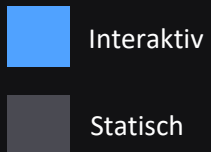
Live

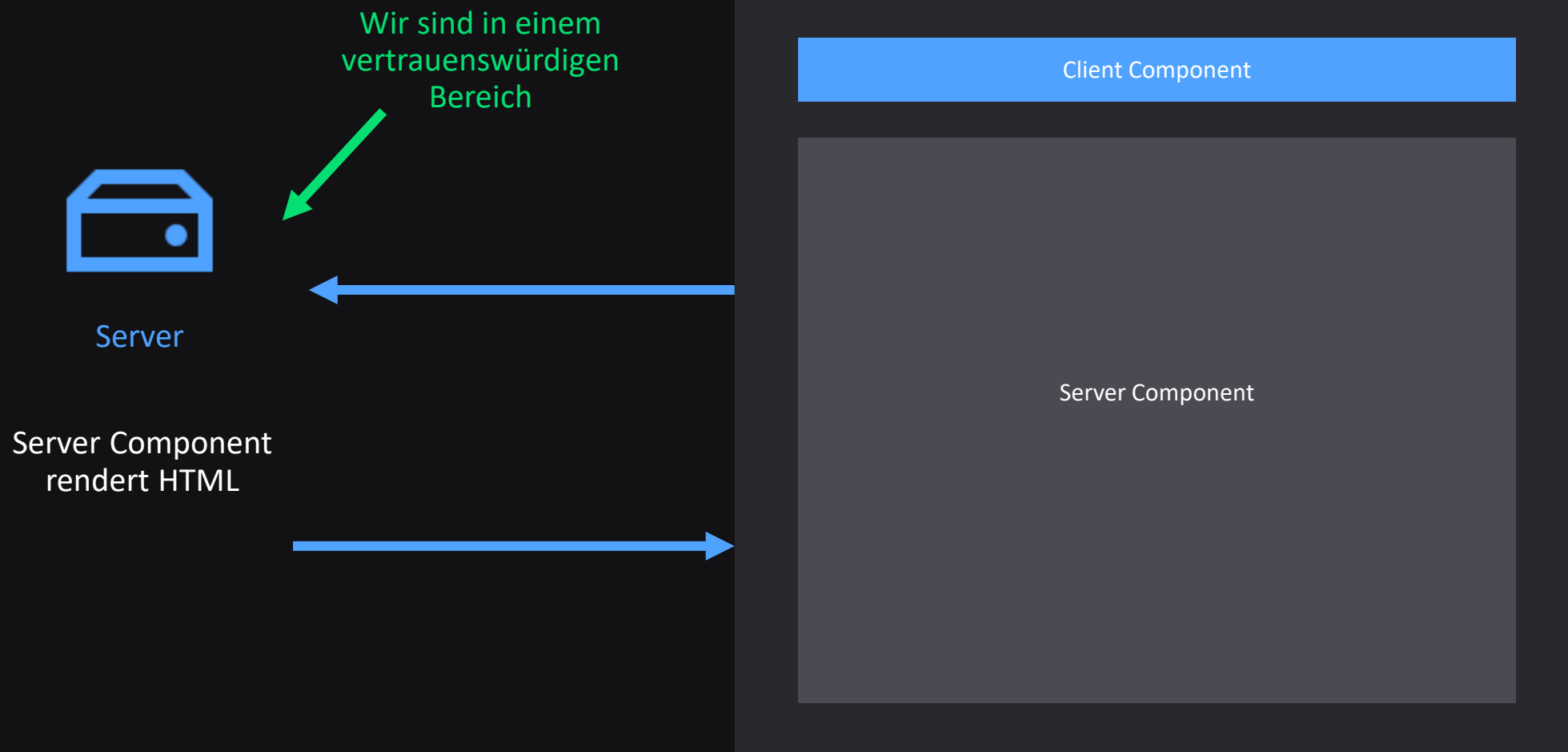
An echten Beispielen erklärt!

Vorteile und Nachteile

React Server Components

- Performanter
- Kleineres Bundle

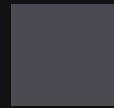




React Server Components Nachteile



Client Code



Server Code

Keine Interaktivität

(z.B. useState, useEffect)

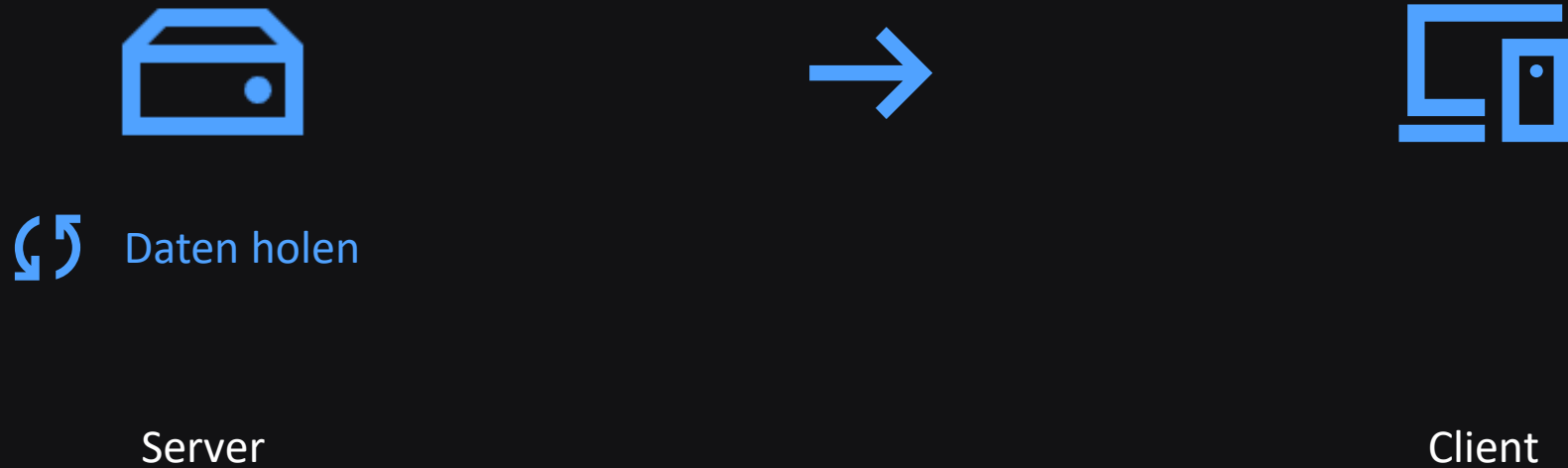


Fragen?

Server Data Fetching

Wie holen wir Daten vom Server?

Was ist Server Data Fetching?



Server Data Fetching mit `fetch()`

```
export default async function VorlesungsPage()
  const result = await fetch('https://api.example.com/vorlesungen')
  const vorlesungen = await result.json()

  return <Vorlesungsliste data={vorlesungen} />
}
```



Next.js erweitert `fetch()` mit serverseitigem Caching

Server Data Fetching mit **ORM** oder Datenbank

```
export default async function VorlesungenOverviewPage() {  
  const vorlesungen = await getDatabase().all('SELECT * FROM vorlesungen')  
  
  return <Vorlesungsliste data={vorlesungen} />  
}
```

Fragen?

Übungsaufgabe

Server Data Fetching

1. Wechsle den branch: `git checkout übung-3-server-data-fetching`
2. Erstelle einen serverseitigen Daten-Fetch auf der Detailseite `src/app/vorlesungen/[id]/page.tsx`. Lese die `id` aus den URL-Parametern aus und hole die Daten der entsprechenden Vorlesung aus der Datenbank.
3. 💡 Tipp: Nutze das sql statement und zeige einen Fehler an wenn es keine Vorlesung gibt (wegen Typescript).

```
const database = await getDatabase()  
const vorlesung = await database.get<Vorlesung>('SELECT * FROM vorlesungen WHERE id = ?', [id])  
  
// Wenn die Vorlesung nicht gefunden wurde, Fehler anzeigen  
if (!vorlesung) {  
  return <p>Vorlesung nicht gefunden!</p>  
}
```

Server Actions

Wie schicken wir Daten zum Server?

Was sind **Server Actions**?

Server Actions sind asynchrone Funktionen, die direkt auf dem Server ausgeführt werden und ohne separate API-Route auskommen.



Server



Formular

```
export default function Page() {  
  
  async function save(formData: FormData) {  
    'use server'  
    ...  
  }  
  
  return <form action={save}>  
    <label>Name der Vorlesung</label>  
    <input type="text" name="name" />  
  </form>  
}
```

Übungsaufgabe

Server Actions

1. Wechsle den branch: `git checkout übung-4-server-actions`
2. Füge die Server Action für das Speichern einer Vorlesung hinzu.
`/src/app/vorlesungen/erstellen/page.tsx`
3. Erstelle eine `<form>` Komponente und füge die Felder `name`, `beschreibung`, `dozent`, `ects` zum Formular hinzu.
4. Bearbeite die `save` Funktion das sie die Felder speichert.

```
const database = await getDatabase()  
await database.run(`INSERT INTO vorlesungen (name, beschreibung, dozent, ects) VALUES (?, ?, ?, ?)`,  
[name, beschreibung, dozent, ects]) // In die Datenbank eintragen  
  
revalidatePath('/') // Cache für die Startseite revalidieren  
redirect('/') // Weiterleiten nach dem Speichern
```

Ende Vorlesung #1

Danke!