

**Web Dev NextJs**



Michael

Fullstack



Joy

Frontend/Design



Max

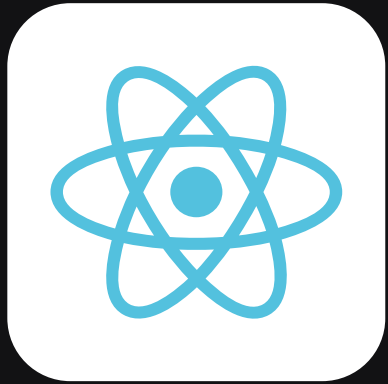
Backend

# Was ist NextJs

99

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 ...

# Was ist NextJs



Full-stack

Viele Features  
Flexibel

SEO & Images  
Optimiert

# Was bietet NextJs



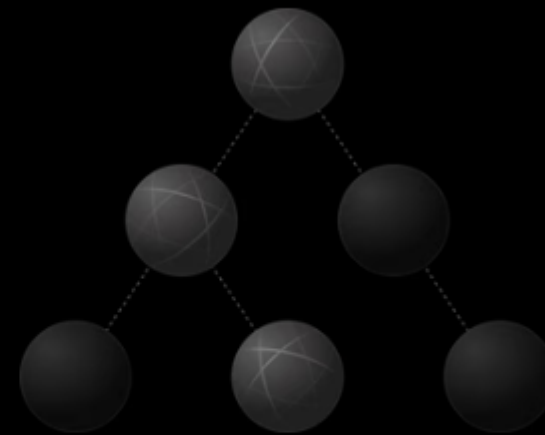
## Built-in Optimizations

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



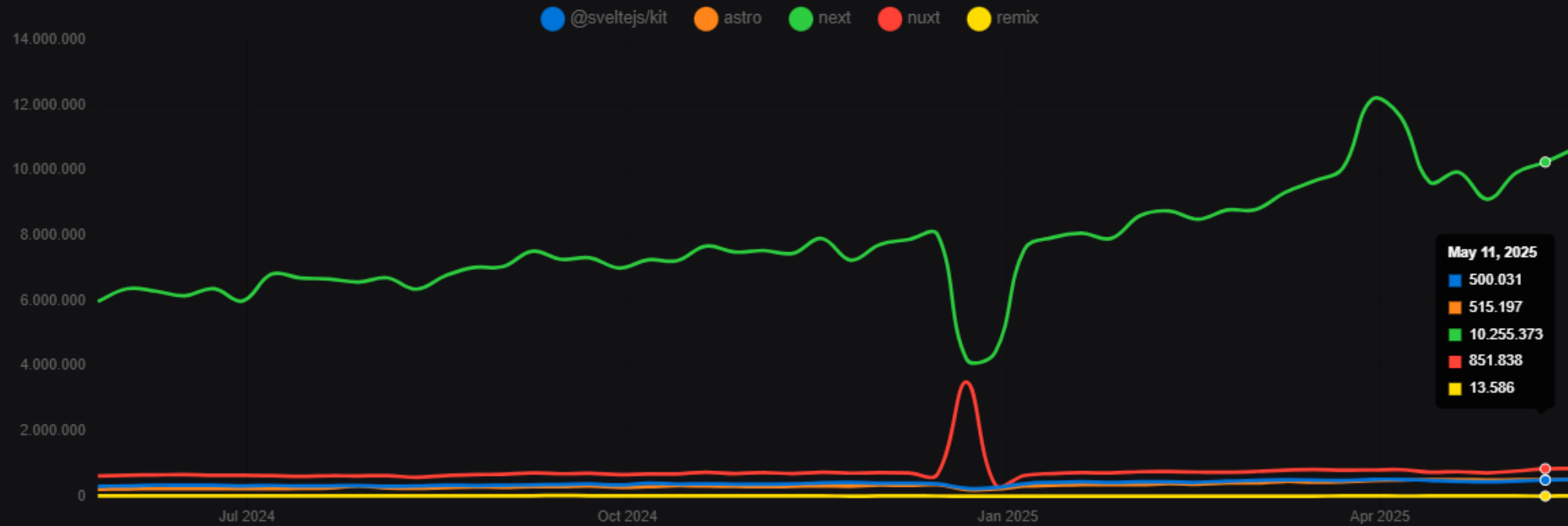
★ 51k  
Astro



★ 19k  
SvelteKit



# Einordnung NextJs



# (Markt) Nutzung

<https://nextjs.org/showcase>

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

# Setup

Node 18+

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

Fundamentals

# Fundamentals

Live

# Fragen?

# Unser Beispiel

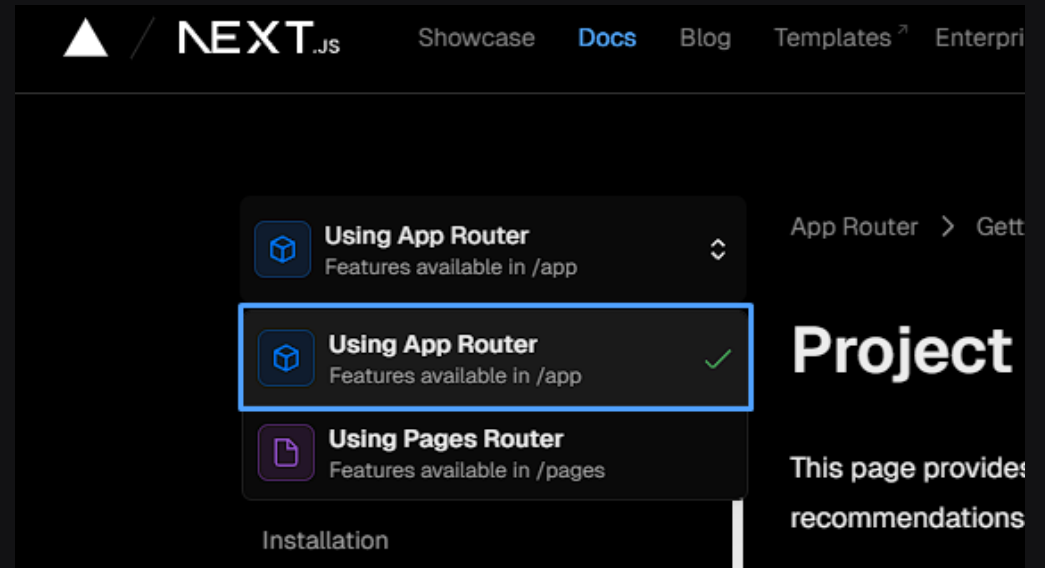
# Ressourcen für Übungen

 `git clone https://github.com/dopeshot/web-dev.git`  
`git clone git@github.com:dopeshot/web-dev.git`

 [nextjs.org](https://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>



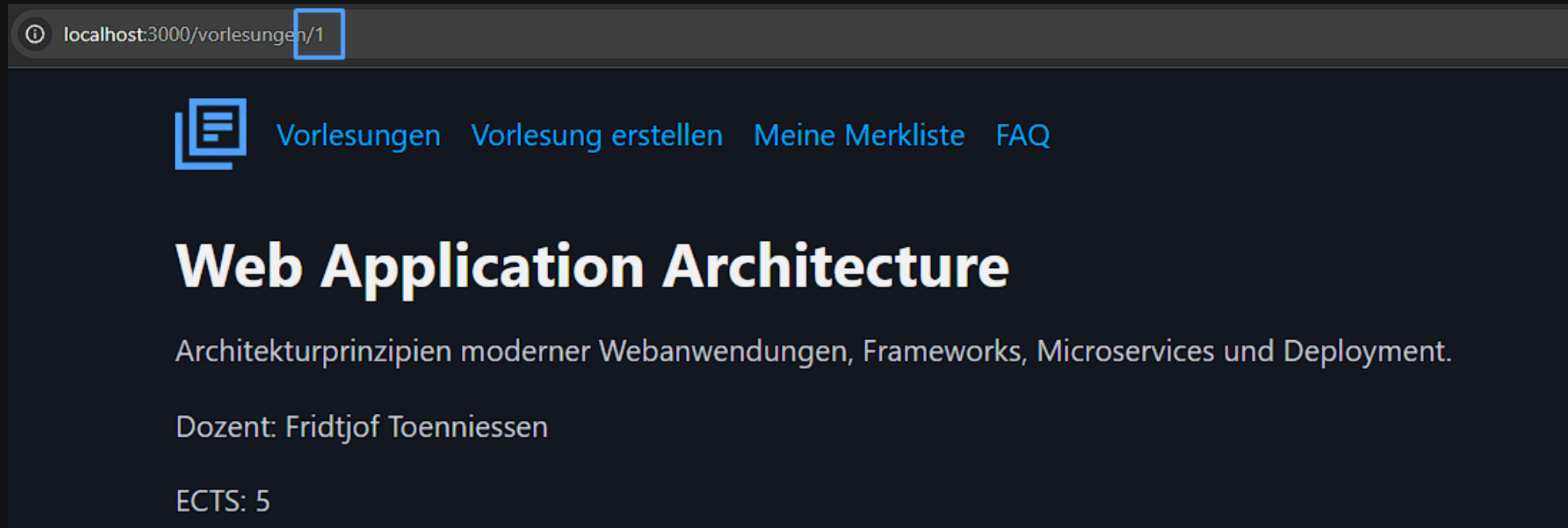
?

`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>

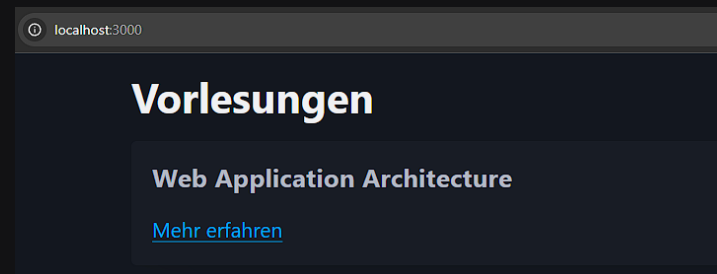
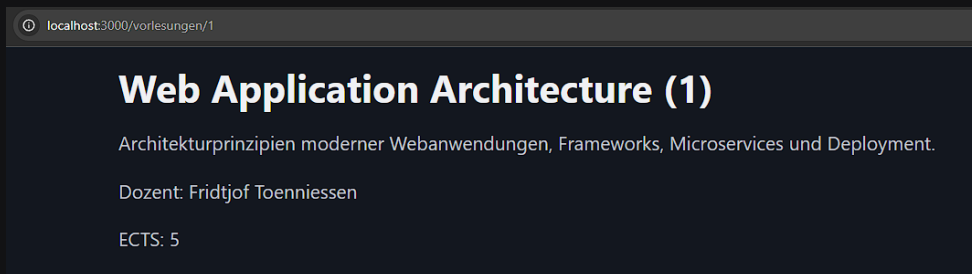
- ✓ Kein vollständiger Reload der Seite
- ✓ Schnelleres Laden durch Prefetching
- ✓ Bessere UX

# Fragen?

## Übungsaufgabe

# Routing & Linking

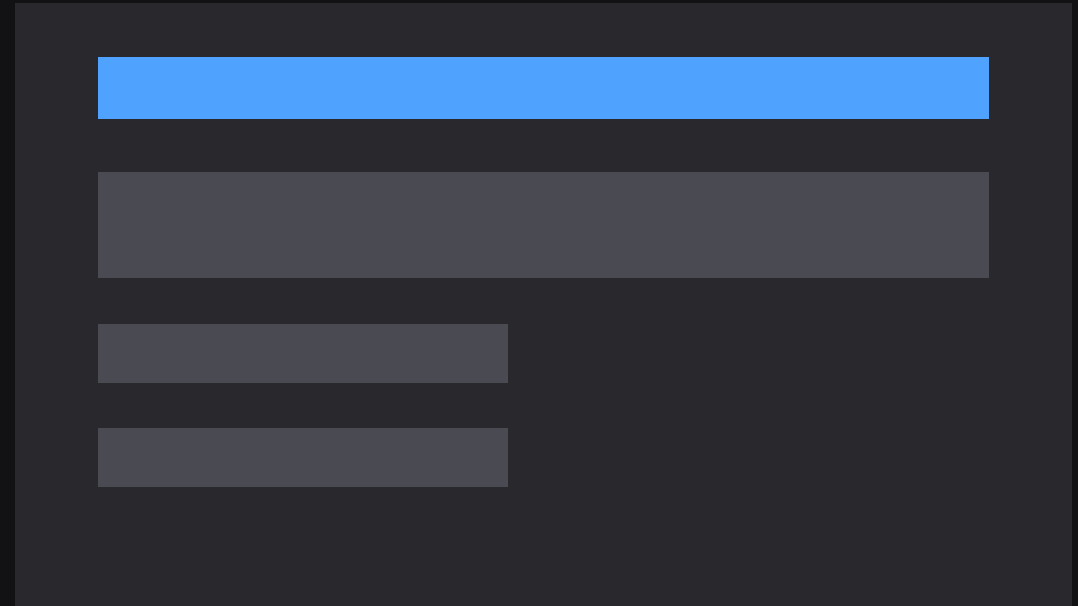
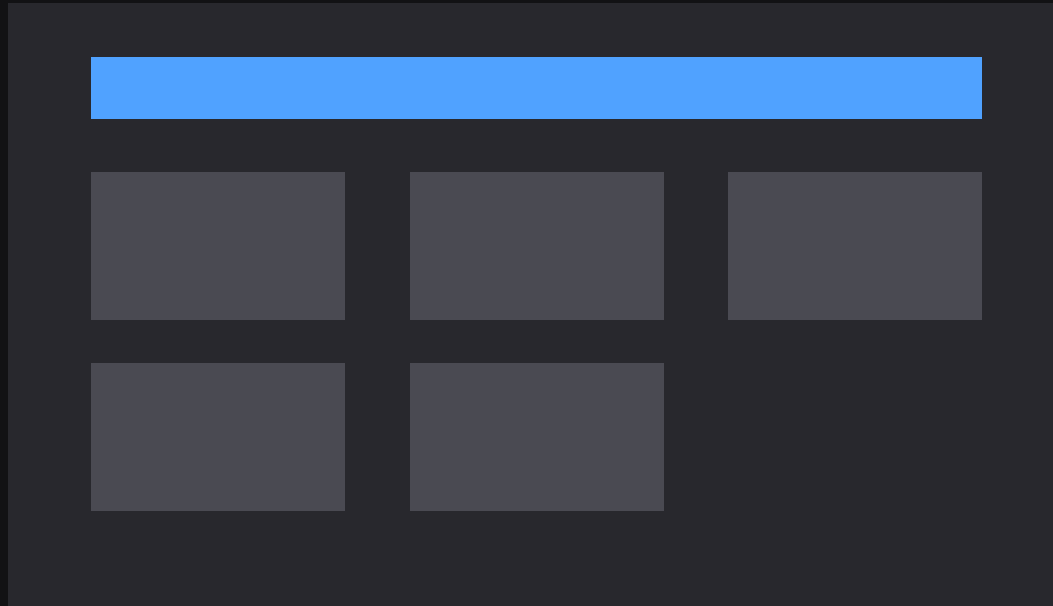
1. Wechsle den branch: `git checkout übung-1-routing-und-linking`
2. Erstelle die Detailseite für eine Vorlesung in `src/app/vorlesungen/[id]/page.tsx`. Nutze eine lokale Variable mit Beispieldaten, um die Detailansicht einer Vorlesung darzustellen.
3. Verlinke die Hauptseite `/src/app/page.tsx` mit 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

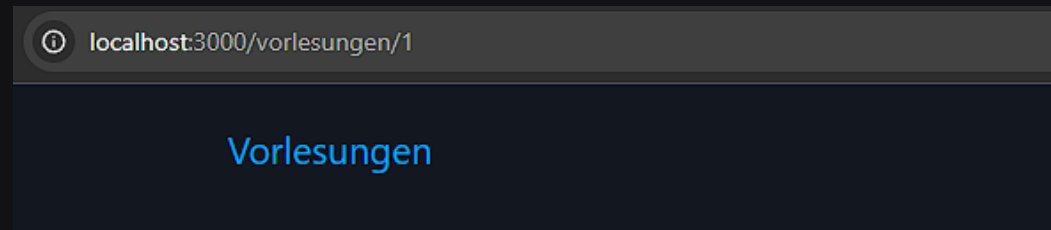
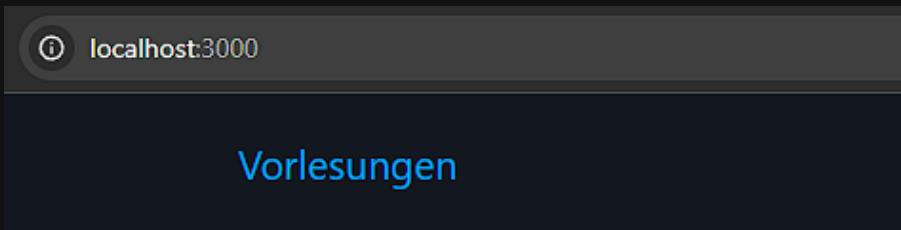
- ✓ Wiederverwendbar -> weniger Code Duplicate
- ✓ Persistenz → schneller navigieren, kein flackern beim animieren
- ✓ Zentrale Steuerung von Logik & UI-Elementen

# 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 **render**t 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

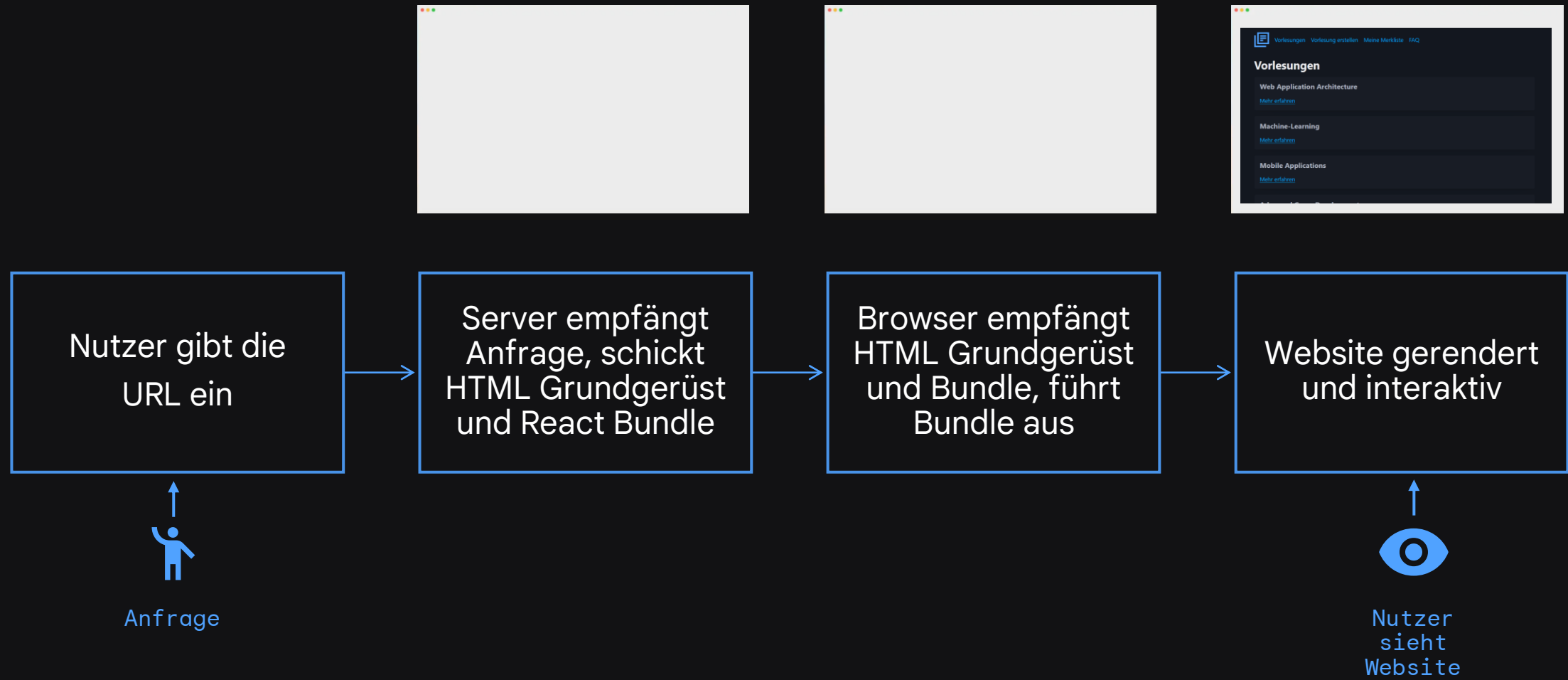


Server

Browser



## Client Side Rendering



# 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

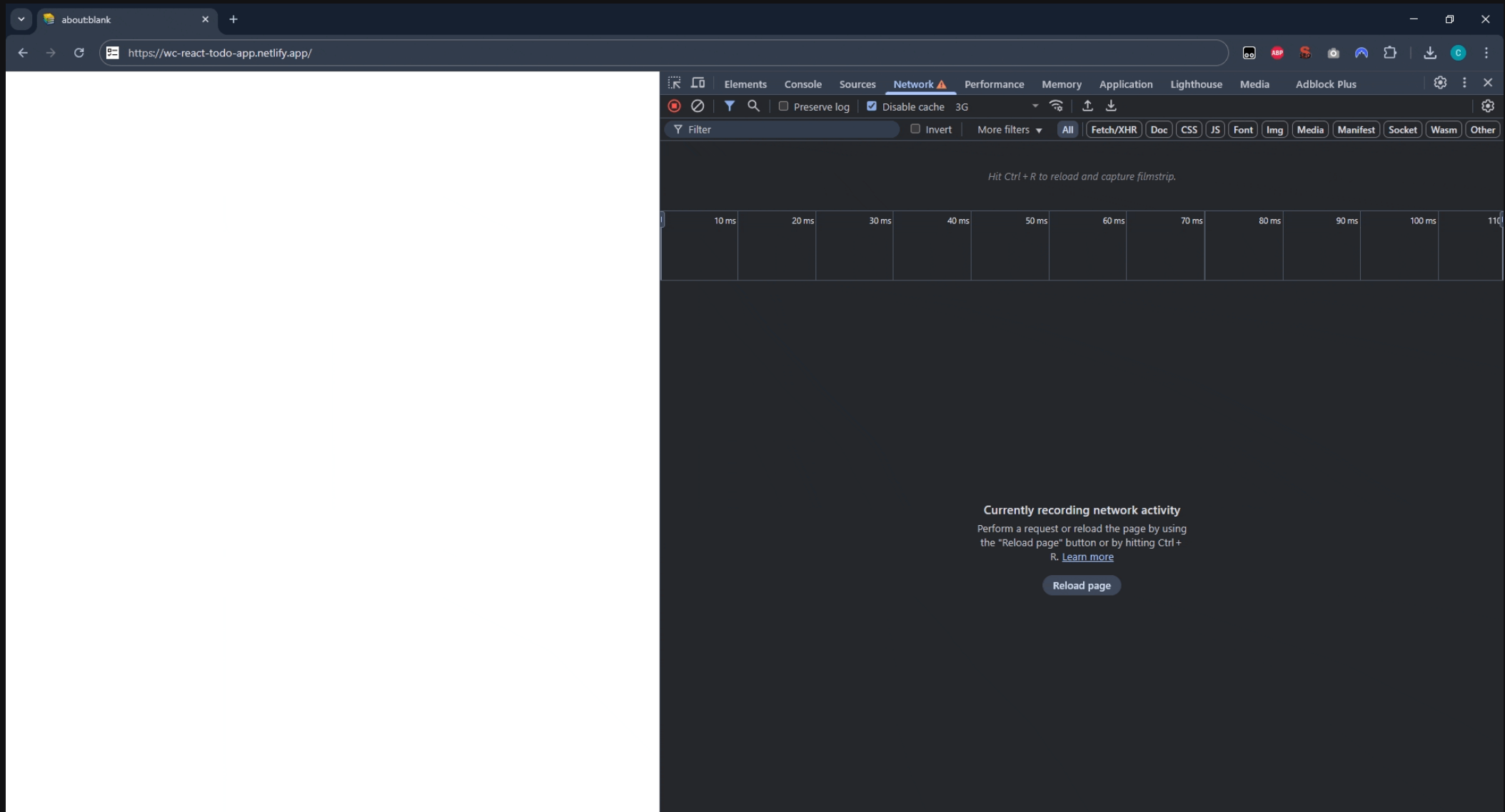
Kosteneffizient

Entwicklerfreundlich und leicht umzusetzen

# Was könnten **Nachteile** sein?

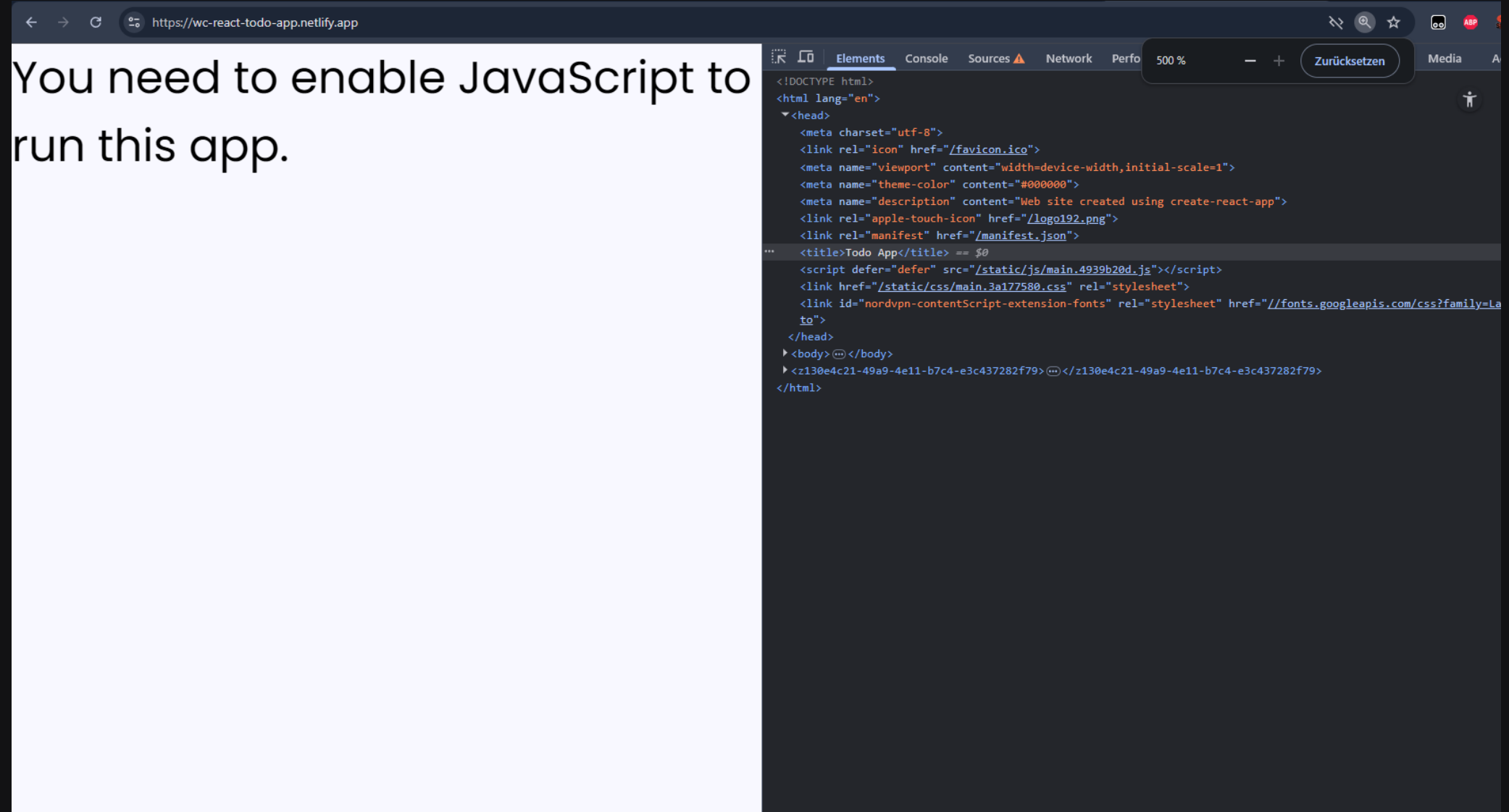
Client Side Rendering

# Client Side Rendering Nachteile



## Client Side Rendering Nachteile

You need to enable JavaScript to run this app.



The screenshot shows a web browser window with the URL `https://wc-react-todo-app.netlify.app`. The page content is a large white rectangle with the text "You need to enable JavaScript to run this app." The browser's developer tools are open, showing the 'Elements' panel. The DOM tree displays the following HTML 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">
    <link id="nordvpn-contentScript-extension-fonts" rel="stylesheet" href="//fonts.googleapis.com/css?family=La
to">
  </head>
  <body>
    <z130e4c21-49a9-4e11-b7c4-e3c437282f79>
  </body>
</html>
```

# Server Side Rendering

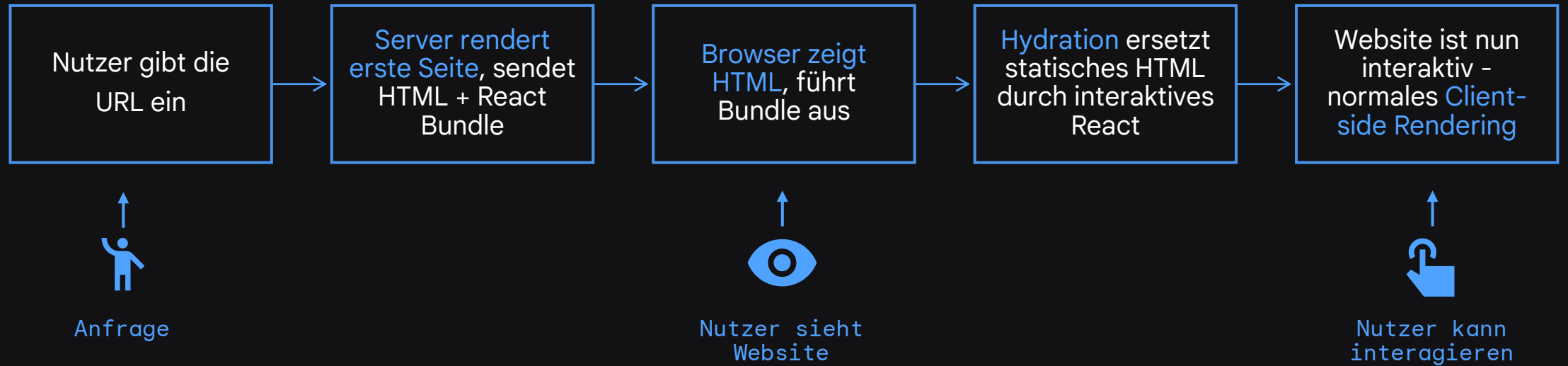
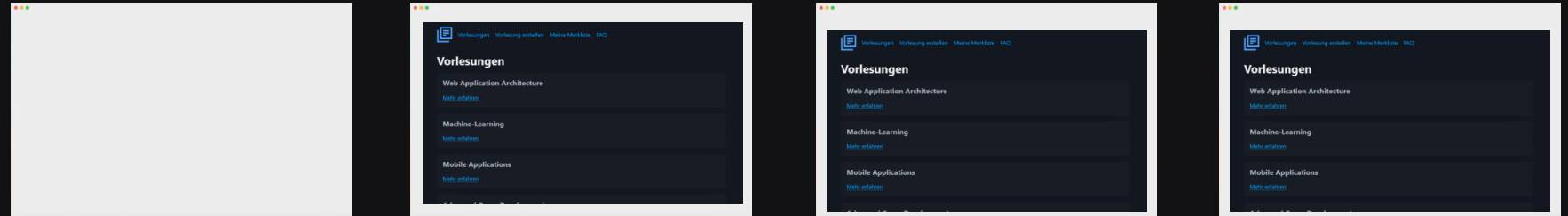
Die erste Darstellung der Seite wird auf dem Server gerendert – der Rest passiert im Browser.

Probleme Client Side Rendering

Langsamer First Contentful Paint

Schlechte SEO

# Server Side Rendering





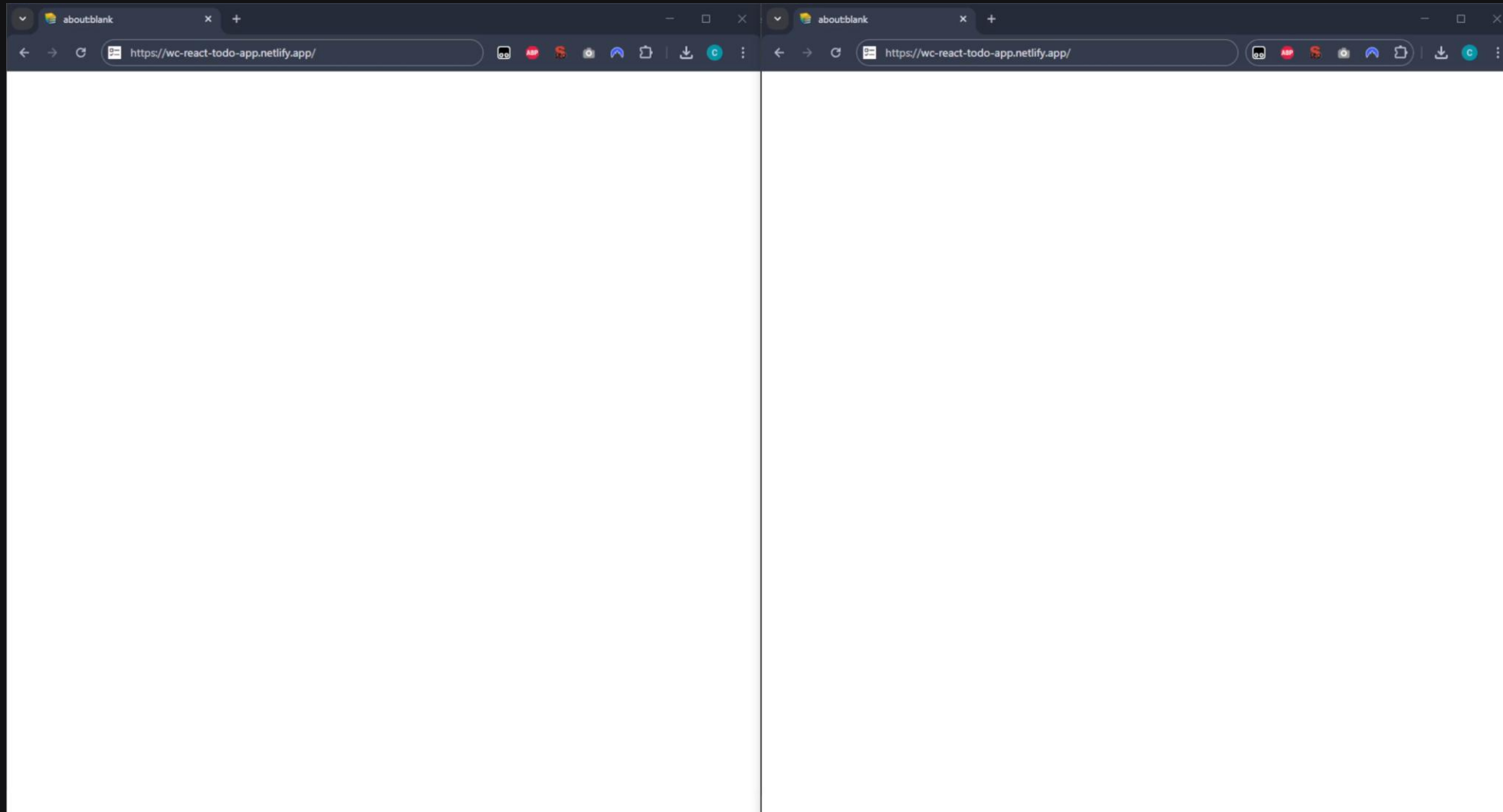
# Live

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 URL `https://wc-react-todo-app.netlify.app`. The application is a "TODO LIST" with a blue "Add Task" button, a filter dropdown set to "All", and a "No Todos" message. The browser's developer tools are open to the "Elements" tab, showing the following HTML 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>
      </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

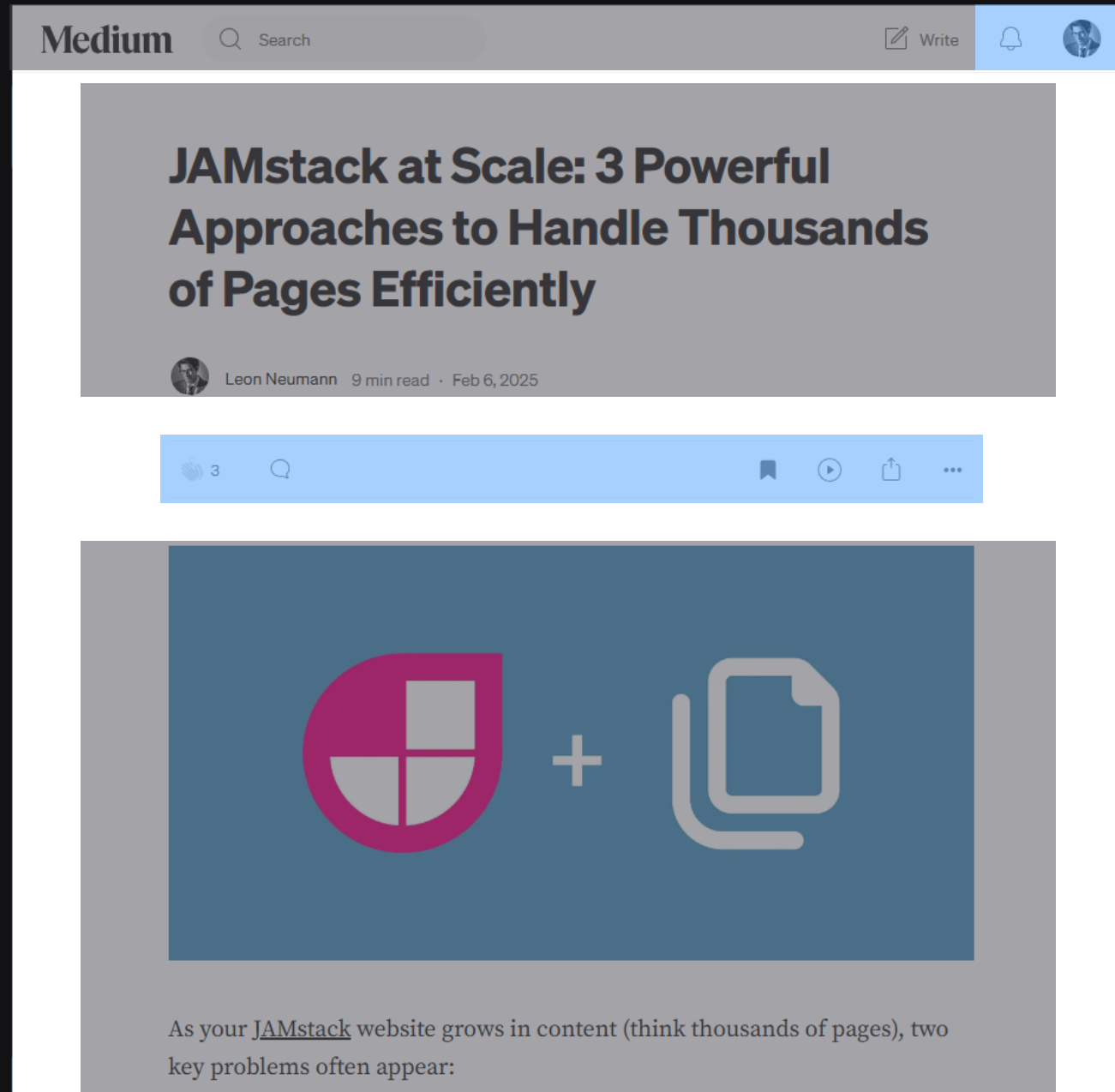
Komponenten die nur auf dem Server gerendert werden, nicht im Browser.

Probleme Server Side Rendering

Interaktivität kommt verzögert

Doppelte Ausführung

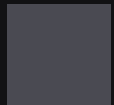
Immer noch viel Code im Client



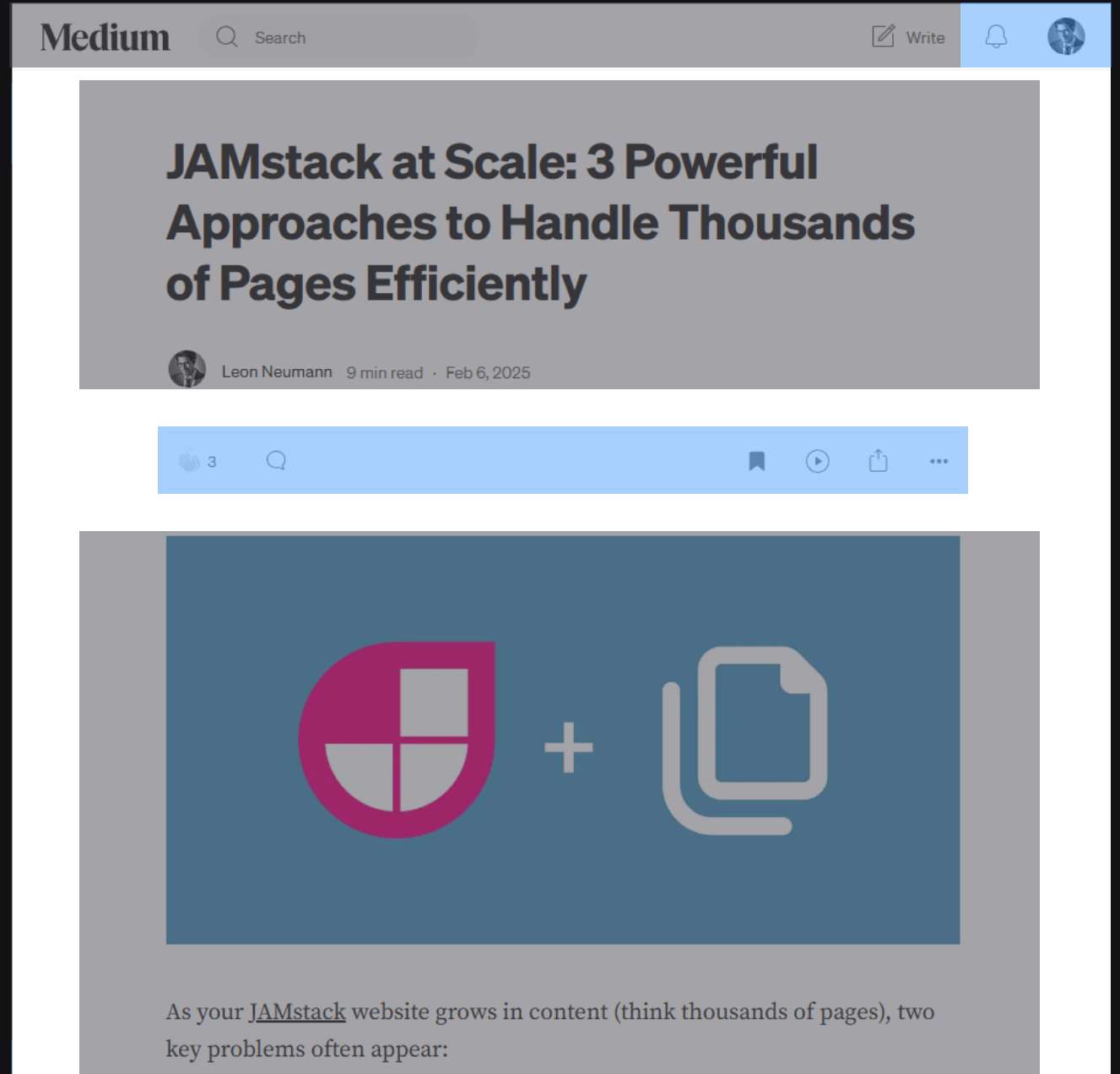


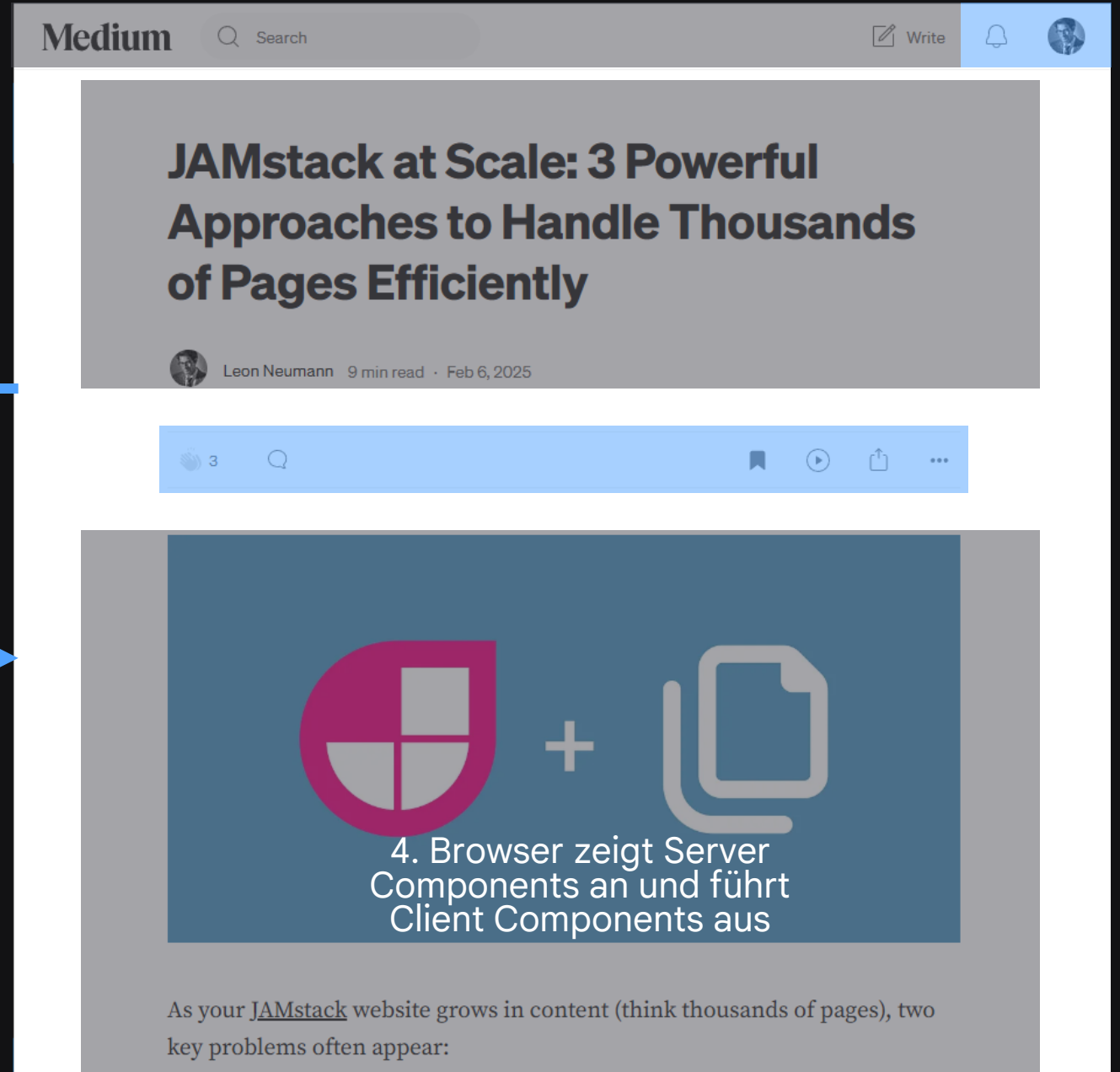
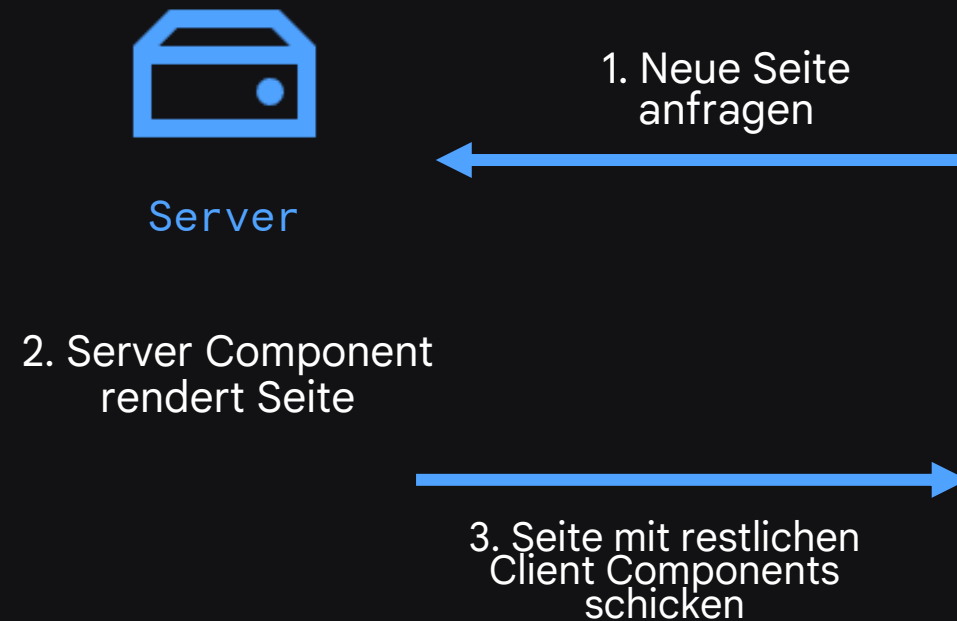


Client Component



Server Component





# 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

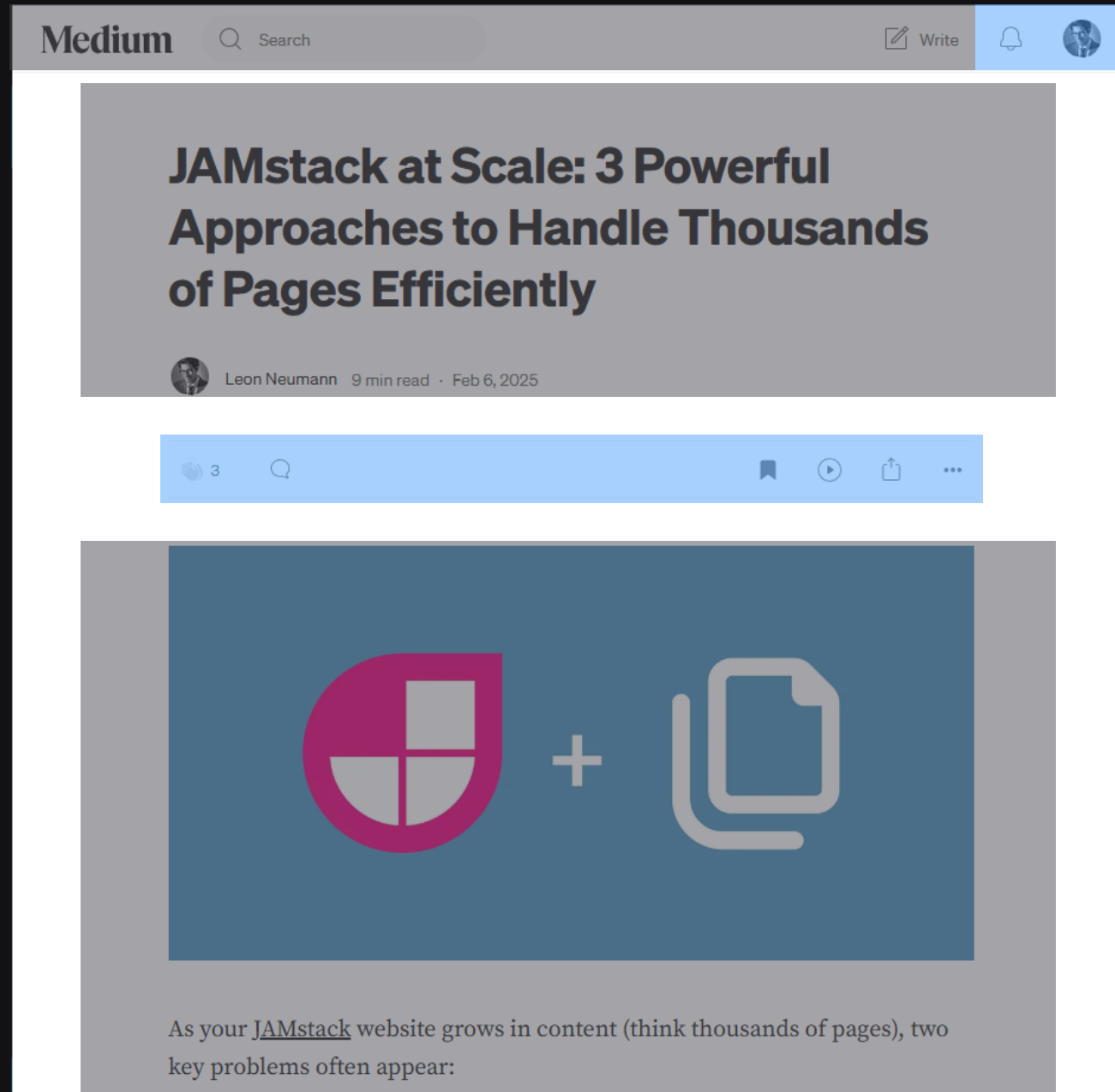
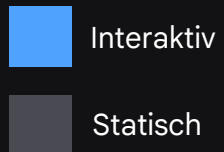
# Live

An echten Beispielen erklärt!

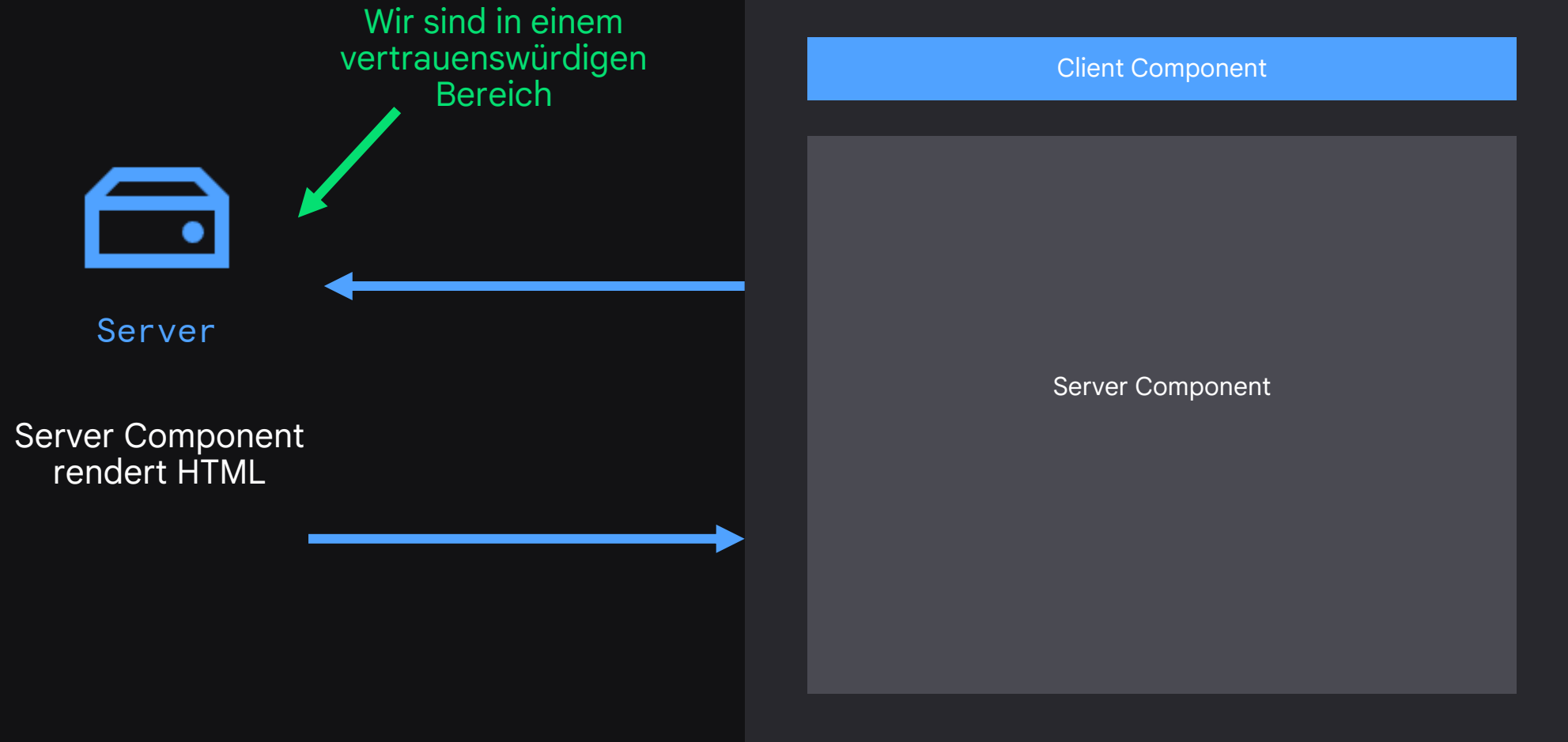
# Vorteile und Nachteile

React Server Components

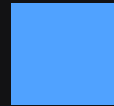
- Performanter
- Kleineres Bundle



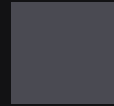
## React Server Components Vorteile



## React Server Components Nachteile



Client Code



Server Code

Keine  
Interaktivität  
(z.B. `useState`,  
`useEffect`)



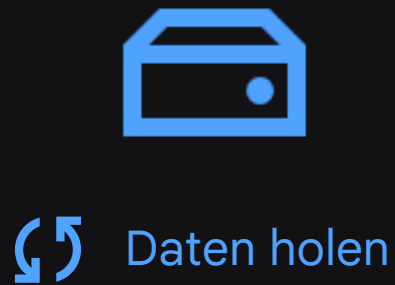


# Fragen?

# Server Data Fetching

Wie holen wir Daten vom Server?

# Was ist Server Data Fetching?



Server



Client

# 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>
}
```

# Ende Vorlesung #1

Danke!

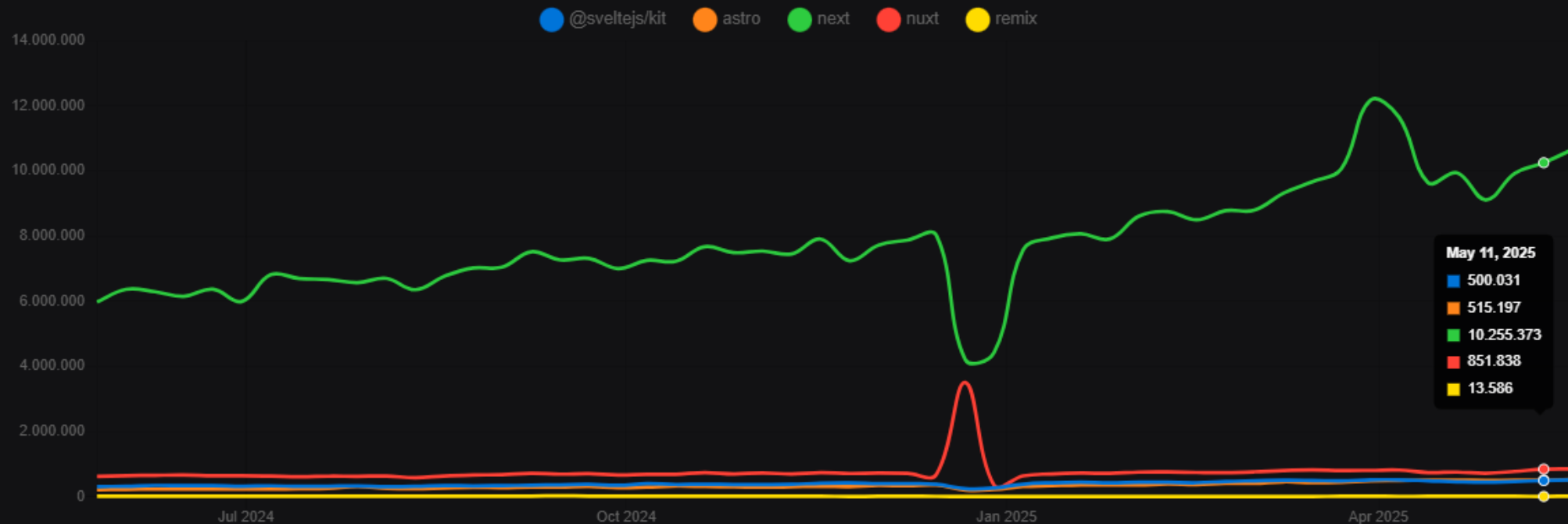


# NextJS

## Vorlesung 2

# Was ist NextJS?

Fullstack Web Framework - Entwickelt von [Vercel](#)



NETFLIXJOBS

CAREERSSTANDORTEONLY AT NETFLIX

BE PART OF WHAT'S NEXT

Search jobs by keyword

Stadt, Bundesland, Postleitzahl oder „Telearbeit“

Los

Erweiterte Optionen (1)

Filter zurücksetzen

Sortieren nach: Relevanz

25 offene Stellen. Verwende deinen Lebenslauf, um die passende Stelle zu finden. Laden Sie Ihren Lebenslauf hoch

Neues und HighlightsHerrenD:

Alle Neuerscheinungen anzeigen

Hoch bewertet

KOSTENLOSER STANDARDVERSAND IM ADICLUB

hilfebestellungen und rückgabegeschenkkartenwerde mitglied

SCHUHEMÄNNERFRAUENKINDERSPORTLIFESTYLEOUTLET

Suchen

1

FILTERN & SORTIEREN

Adicolor Classic Firebird Loose Trainingshose

Frauen Originals

17 Farben

Firebird Shorts

Frauen Originals

7 Farben

adidas x Jeremy Scott Pride Mesh-T-Shirt

Originals

Neu

Adicolor Classic Firebird Loose Trainingshose

Frauen Originals

17 Farben

OpenAI

Hauptmenü

Forschungsindex

Forschungsüberblick

Forschungsresidenz

Aktuelle Fortschritte

OpenAI o3 und o4-mini

GPT-4.5

OpenAI o1

GPT-4o

Sora

Forschung

AllePublikationFazitMeilensteinVeröffentlichung

Safety

23. Mai 2025

Addendum to OpenAI o3 and o4-mini system card: OpenAI o3 Operator

We are replacing the existing GPT-4o-based model for Operator with a version based on OpenAI o3. The API version will remain based on 4o.

Veröffentlichung

16. Mai 2025

Introducing Codex

Introducing Codex: a cloud-based software engineering agent that can work on many tasks in parallel, powered by codex-1. With Codex, developers can simultaneously deploy multiple agents t...

Safety

16. Mai 2025

Addendum to o3 and o4-mini system card: Codex

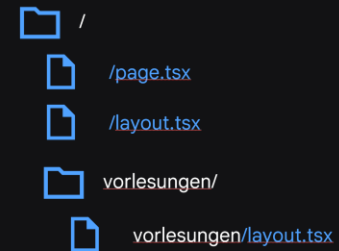
Codex is a cloud-based coding agent. Codex is powered by codex-1, a version of OpenAI o3 optimized for software engineering. codex-1 was trained using reinforcement learning on real-wor...

## Setup CLI

`npx create-next-app@latest`

## Routing & Navigation

### File-Based Struktur Layout



## Data Fetching

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

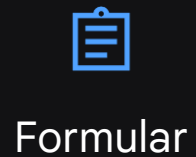
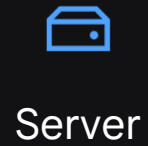
	Client Side Rendering (CSR)	Server Side Rendering (SSR)	React Server Components (RSC)
Ausführungsort	Nur im <b>Browser</b>	Initial auf dem <b>Server</b> , dann Browser	Nur auf dem <b>Server</b>
Was wird an Client gesendet?	JavaScript	HTML und JavaScript	HTML*
Produktqualität für Nutzer	◆ Mittel	✓ Gut	🏆 Exzellent
Komplexität	Einfach	Mittel	Hoch (Fullstack-Denken nötig)

# 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.



```
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. Erweitere die `save` funktion zu einer `Server Action` unter `/src/app/vorlesungen/erstellen/page.tsx`
3. Vervollständige die `<form>` Komponente und füge die Felder `beschreibung`, `dozent`, `ects` zum Formular hinzu.
4. Vervollständige die `save` Funktion.

```
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
```

# Client Components



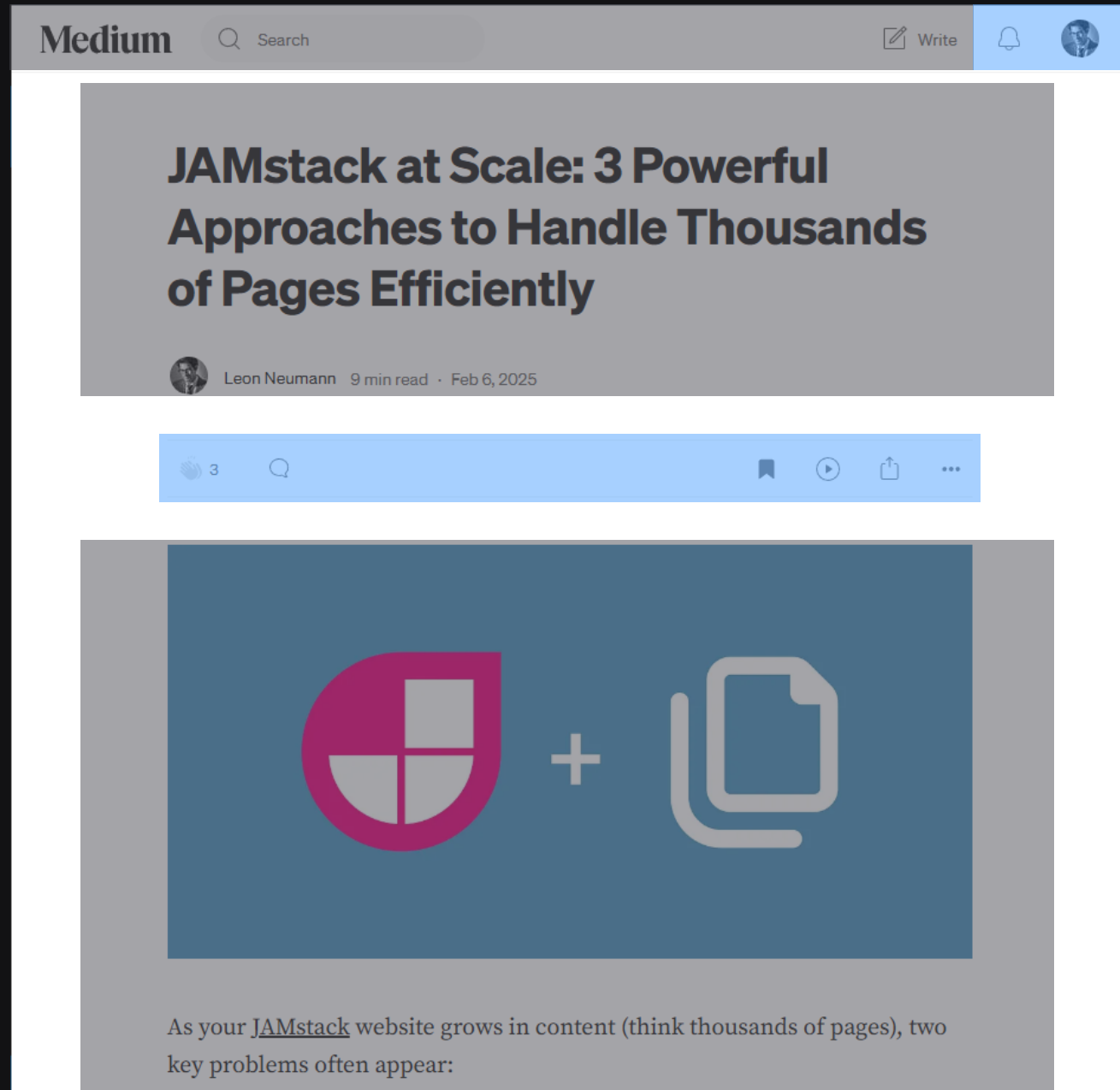
Client Component

**Client Components**  
laufen im Browser

**Interaktivität**

**State & Lifecycle**

**Browser APIs**

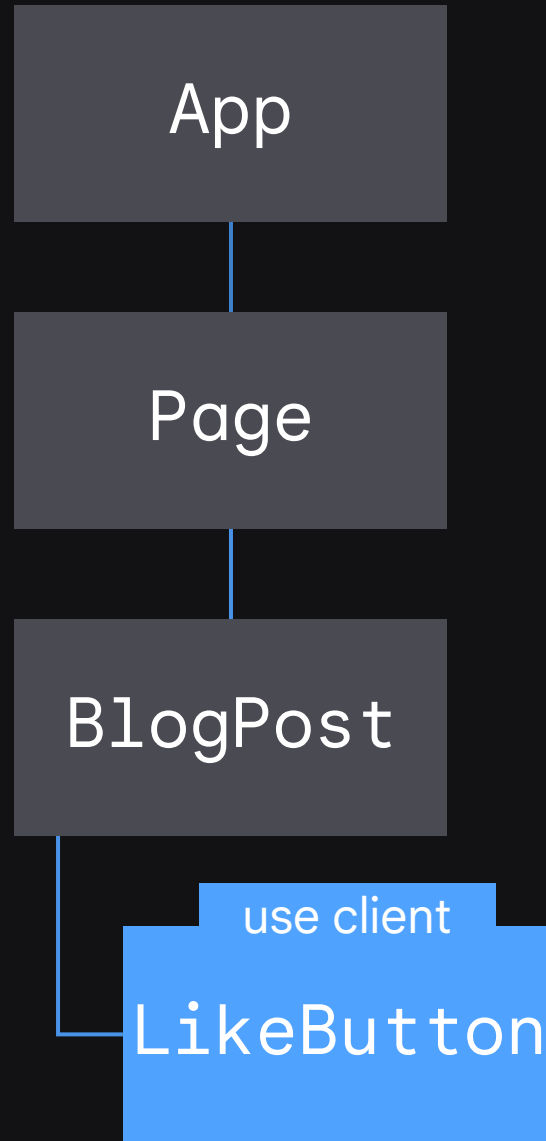
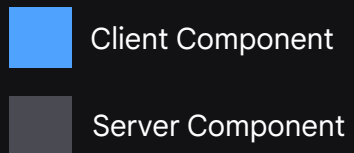


```
'use client';
```

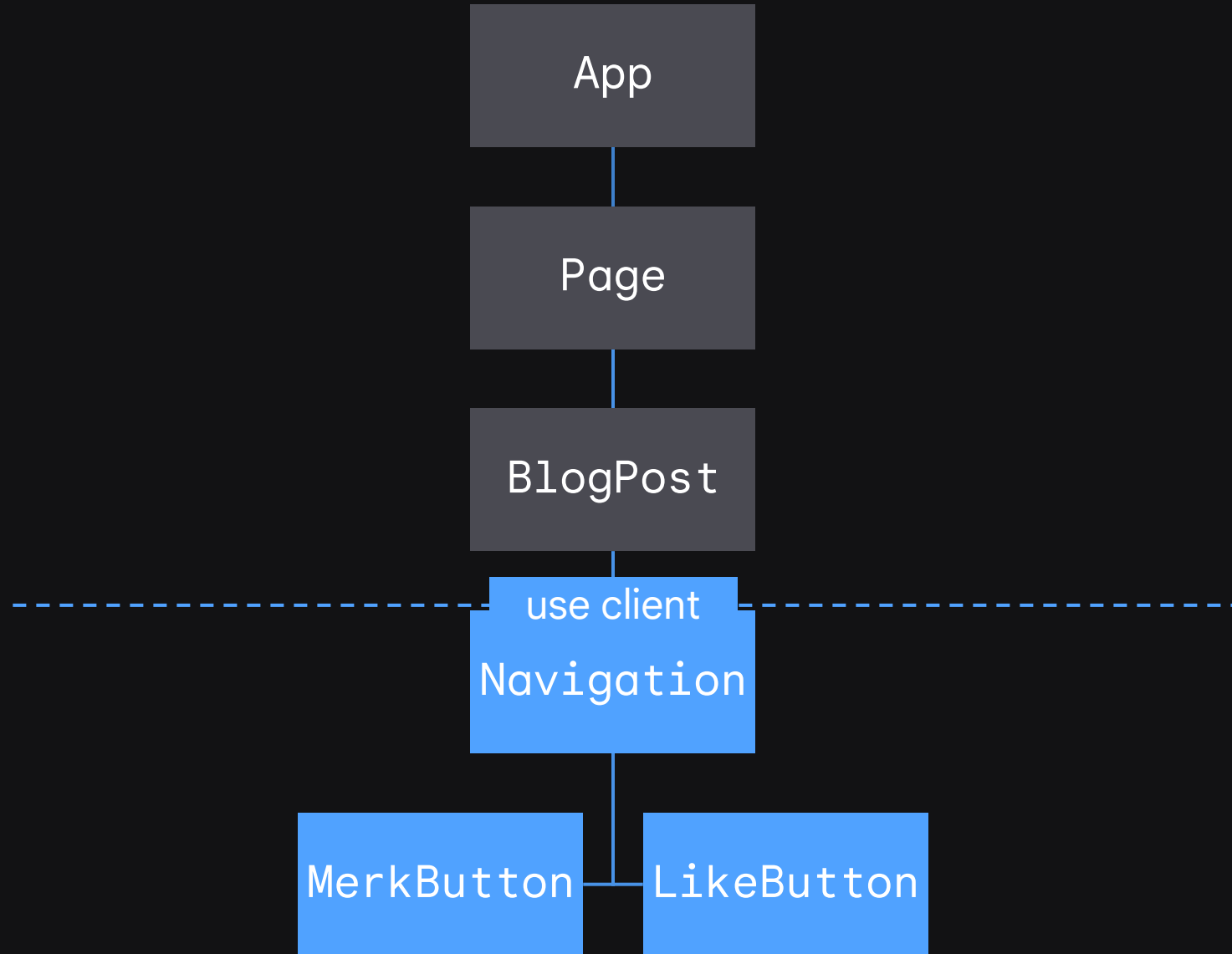
```
import { useState } from 'react';  
import { formatDate } from '../formatters';  
import Button from '../button';
```

```
export default function RichTextEditor({ timestamp, text }) {  
  const date = formatDate(timestamp);  
  // ...  
  const editButton = <Button />;  
  // ...  
}
```

## Client Components



## Client Components



# Fragen?

## Übungsaufgabe

# Merkbutton mit useState

1. Wechsle den branch: `git checkout übung-5-merkebutton`
2. Erstelle eine **Client Component** mit "use client"-Direktive (`/src/components/MerkButton.tsx`).
3. Die Komponente zeigt einen `<button>`:  
**Gemerkt oder Merken**
4. Beim Klick auf den Button soll der Zustand umgeschaltet werden.
5. Die Komponente soll auf der `/app/vorlesungen/[id]/page.tsx` eingebaut werden.

Tipps:

Verwende `useState()`, um den Zustand zu speichern.

Der Status muss **nicht gespeichert bleiben**, nur im aktuellen Session-State.



# Live

An echten Beispielen erklärt!

Erster Seitenaufruf



Server  
prerendert  
Client Component

SSR



Hydratation und  
Initialisierung im  
Browser

CSR

Folgenavigation



Normales  
Rendering im  
Browser

CSR

# Gemerkt Zustand im Browser speichern!

[Vorlesungen](#) [Vorlesung erstellen](#)

## Web Application Architecture

Architekturprinzipien moderner Webanwendungen, Frameworks, Microservices und Deployment.

Dozent: Fridtjof Toenniessen

ECTS: 5

Vorlesung merken ☒



localStorage

Im Browser

merkliste=[1,3,5]

# Live

An echten Beispielen erklärt!

# Was ist Client-only Rendering?

Komponenten werden **nur im Browser** gerendert

Kein Server-Side Rendering (SSR)

## MerkButtonClientOnly.tsx

```
'use client'

import dynamic from 'next/dynamic'

export const MerkButtonDynamic = dynamic(
  () => import('./MerkButton').then((module) => module.MerkButton),
  {
    ssr: false,
    loading: () => <span aria-busy="true">Laden...</span>,
  },
)
```

[Vorlesungen](#) [Vorlesung erstellen](#)

# Web Application Architecture

Architekturprinzipien moderner Webanwendungen, Frameworks, Microservices und Deployment.

Dozent: Fridtjof Toenniessen

ECTS: 5

Vorlesung merken

# Fragen?



## Übungsaufgabe

# Client-only Komponente

1. Wechsle den branch: `git checkout übung-6-client-only`
2. Erstelle eine Datei `MerkButtonClientOnly.tsx`
3. Importiere und binde `MerkButton` mit `dynamic` aus `next/dynamic` ein  
`() => import('./MerkButton').then((module) => module.MerkButton)`
4. Setze `ssr: false`
5. Erstelle eine Lade-Animation (z.B. `<span aria-busy="true">Laden...</span>`)
6. Ersetze in `/app/vorlesungen/[id]/page.tsx` den bisherigen `MerkButton` durch `MerkButtonClientOnly`

<https://nextjs.org/docs/app/guides/lazy-loading#skipping-ssr>

# Route Handlers

API-Endpunkte

# Was sind Route Handlers?



Web-API Standard (Request, Response)

# File-Based Struktur Route Handlers



/app



/page.tsx



/route.ts ✗



api/



api/route.ts



## Route Handlers im Code

```
// /app/api/vorlesungen/route.ts

export async function GET() {
  const database = await getDatabase()
  const vorlesungen = await database.all('SELECT * FROM vorlesungen')

  return Response.json(vorlesungen)
}
```

# Route Handlers Params übergeben

```
// app/api/vorlesungen/route.ts

export async function GET(request: NextRequest) {
  const semester = request.nextUrl.searchParams.get('semester')
  return Response.json({ message: `Semester: ${semester}` })
}
```



```
http://localhost:3000/api/vorlesungen?semester=SS2025

{ message: `Semester: SS2025` }
```

## Route Handlers Error Handling

```
return NextResponse.json({ success: true }, { status: 201 });
```

```
return NextResponse.json({ error: 'Bad request' }, { status: 400 });
```

# Fragen?



Wann sollte man **Route Handlers** nutzen?

→ Client Components, Externe Tools

## Wann nutzt man was?

Neue Vorlesung über ein Formular anlegen



**Server Action**

Vorlesungsliste im Server Component anzeigen



**Fetch/Datenbankzugriff  
im Server Component**

Client Site Data Fetching



**Route Handler**

# Fragen?

# Client Data Fetching

Wie wird clientseitig auf Daten zugegriffen?

## Wann machen wir Client Data Fetching?

- ✓ Wenn Daten nur im Browser verfügbar sind
- ✓ Live-Daten
- ✓ SEO oder initiale Ladezeit egal

# Beispiel Merkliste

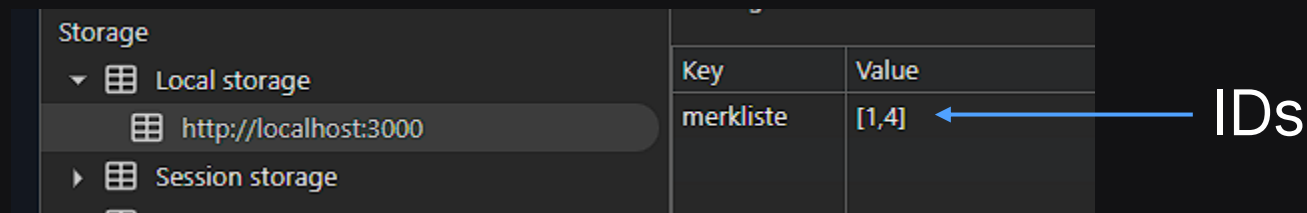


Namen

Ladezustand

Error Handling

Empty State



IDs

# Client Data Fetching mit `useEffect()`

```
'use client'

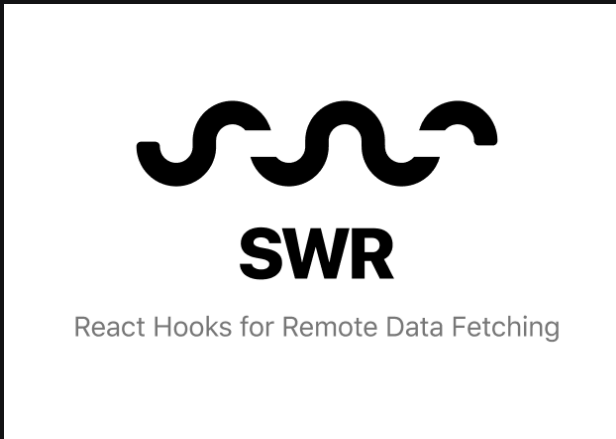
export default function Merkliste() {
  const [vorlesungen, setVorlesungen] = useState([])

  useEffect(() => {
    const merklisteIds = getMerklisteFromLocalStorage()

    async function fetchMerkliste() {
      const response = await fetch(`/api/vorlesungen?ids=${merklisteIds}`)
      const vorlesungen = await response.json()
      setVorlesungen(vorlesungen)
    }
    fetchMerkliste()
  }, [])

  // Render Liste mit Namen
  return <Merkliste vorlesungen={vorlesungen}> />
}
```

# Client Data Fetching mit Libraries



**TanStack Query** v4



# Client Data Fetching mit SWR

```
// /app/Merkliste.tsx

'use client'

const fetcher = (url: string) => fetch(url).then((res) => res.json())

export function Merkliste() {
  const { data: vorlesungen } = useSWR('/api/vorlesungen', fetcher)

  return <Merkliste vorlesungen={vorlesungen} />
}
```

# Client Data Fetching mit SWR

```
const fetcher = (url: string) => fetch(url).then((res) => res.json())  
  
const { data: vorlesungen } = useSWR('/api/vorlesungen', fetcher)
```



Key -> wenn gleich  
cached

# Client Data Fetching mit SWR

```
const { data } = useSWR(shouldFetch ? '/api/vorlesungen' : null, fetcher)
```

# Client Data Fetching mit SWR

```
const { data: vorlesungen, error, isLoading } = useSWR('/api/vorlesungen', fetcher)

if (isLoading) {
  return <Loading />
}

if (error) {
  return <Error />
}

if (!vorlesungen || vorlesungen.length === 0) {
  return <EmptyState />
}

return <Merkliste vorlesungen={vorlesungen} />
```

## Übungsaufgabe

# Merkliste Seite

1. Wechsle den branch: `git checkout übung-7-merkliste-seite`
2. Vervollständige den `get` Route Handler in `/src/app/api/vorlesungen/route.ts`, sodass man per `searchParam` Vorlesungs-IDs übergeben kann und als Antwort die kompletten Vorlesungsobjekte mit Namen erhält.
3. Ergänze dann im Merkliste-Client-Component in `/src/app/merkliste/Merkliste.tsx` den Fetch mit `useSWR` an den Route Handler. Mache den Fetch nur wenn wirklich IDs aus dem `LocalStorage` zurückkommen. Füge noch eine Lade-UI, eine Error-UI und eine UI für eine leere Merkliste ein.

```
const { data: vorlesungen, error, isLoading } = useSWR(key, fetcher)
```

Static

# Static

Static

# Wie **render**t NextJS

Client Side Rendering

CSR

Server Side Rendering (React)

SSR

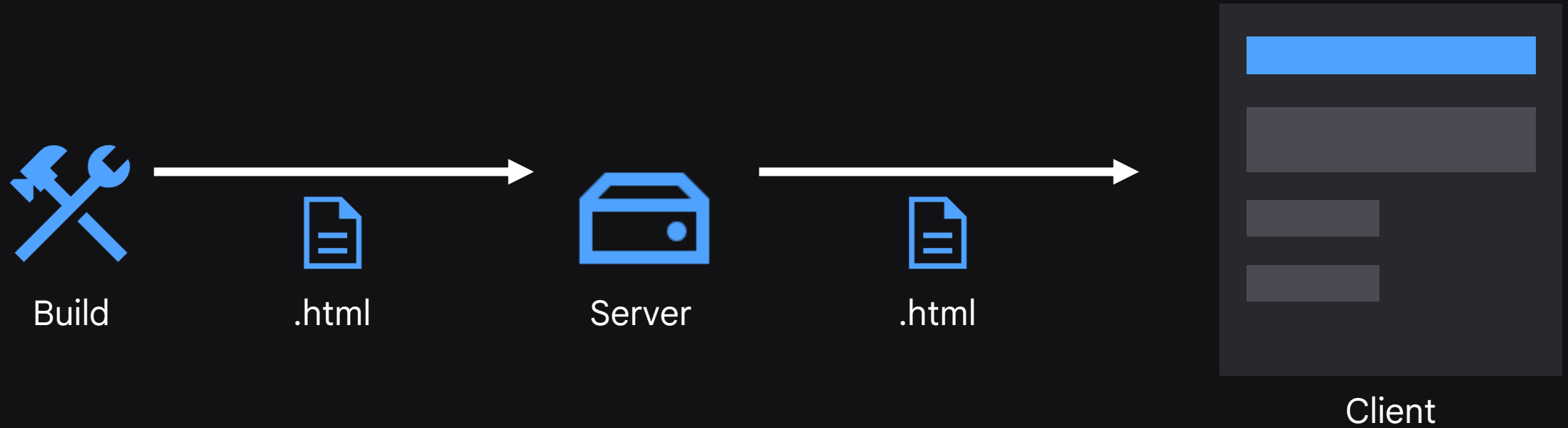
Static Site Generation

SSG

React Server Components

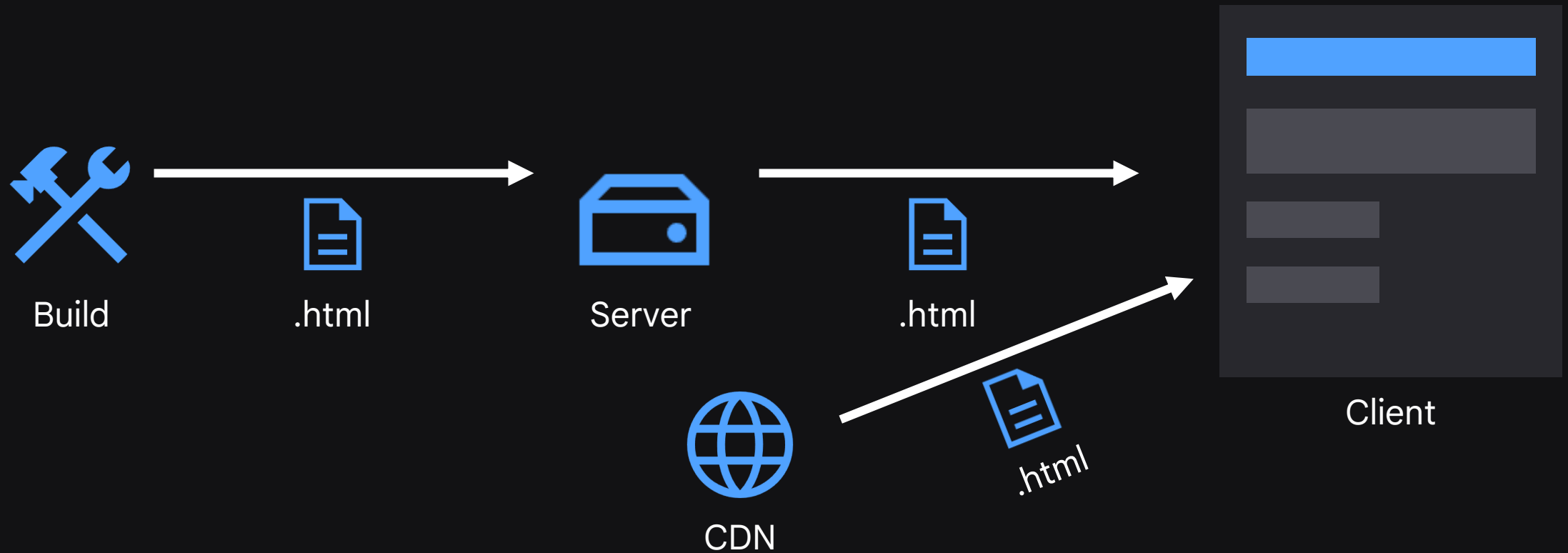
RSC

## Was bedeutet Static





## Was bedeutet Static



## Warum Static?



Schnell



Sicher



Skalierbar

## Wann Static?

- Alles bekannt bei Build
- Geringe Datenmengen
- Änderungsarme Daten
- Public

## Wann Static?

- Alles bekannt bei Build
- Geringe Datenmengen
- Änderungsarme Daten
- Public

## Wann kein Static?

- Unbekanntes beim Build

## Use Cases von Static?

- Blogs
- Landing Pages
- Dokumentation
- Portfolio

Static

# Umgang mit Static

Static Site  
Generation



## How To SSG

```
const nextConfig = {  
  output: 'export',  
}  
  
module.exports = nextConfig
```

## How To SSG

```
/out/index.html  
/out/404.html  
/out/blog/post-1.html  
/out/blog/post-2.html
```



# Supported Features

## Server Components

```
export default async function Page() {  
  // This fetch will run on the server during `next build`  
  const res = await fetch('https://api.example.com/...')  
  const data = await res.json()  
  
  return <main>...</main>  
}
```

## Supported Features

Route Handlers

GET only

```
export async function GET() {  
  return Response.json({name: 'Lee'})  
}
```



data.json

## Unsupported Features

- Dynamic Routes
- Cookies
- ...
- Middleware
- Server Actions
- ...

## Unsupported Features

- Dynamic Routes
- Cookies
- ...
- Middleware
- Server Actions
- ...

## Dynamic Routes - Static

```
export async function generateStaticParams() {  
  const posts = await fetch(...)  
  return posts.map((post) => ({id: post.id})) as Params  
}  
  
export default async function Page(props: Params) {  
  return (...)  
}
```

Static

## Live

Branch: example-static-site-generation

Dynamic Route mit generateStaticParams

Npm run build

./out folder

Static

# Fragen?

Static

**Aber . . .**

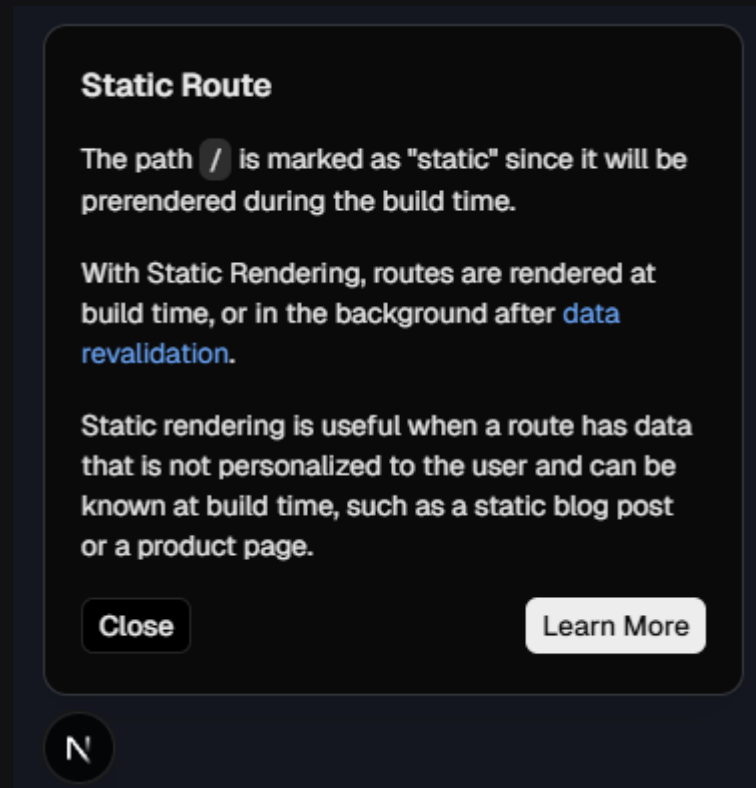
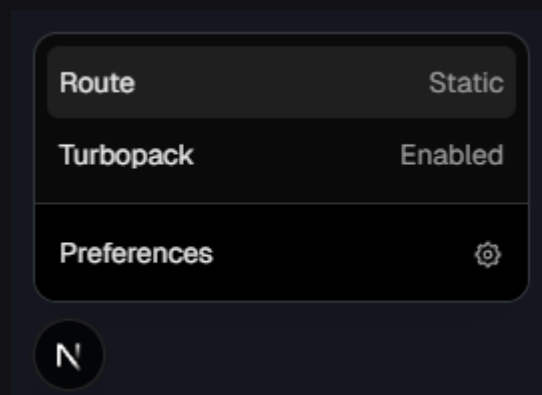
Branch: main

npm run dev

Symbol unten links betrachten



## Was ist dann das?



Static

# Umgang mit Static

Static Site  
Generation

Automatic Static  
optimization

# Automatic Static optimization

```
> next build
```

▲ Next.js 15.3.3

Creating an optimized production build ...

- ✓ Compiled successfully in 1000ms
- ✓ Linting and checking validity of types
- ✓ Collecting page data
- ✓ Generating static pages (10/10)
- ✓ Collecting build traces
- ✓ Finalizing page optimization

Route (app)	Size	First Load JS
o /	175 B	105 kB
o /_not-found	145 B	102 kB
f /api/vorlesungen	145 B	102 kB
o /faq	175 B	105 kB
o /merkliste	5.27 kB	110 kB
o /test-error	326 B	102 kB
f /vorlesungen/[id]	515 B	102 kB
o /vorlesungen/erstellen	145 B	102 kB
+ First Load JS shared by all	102 kB	
chunks/20f27031-3b4977d31a17978d.js	53.2 kB	
chunks/691-d71a779478597b53.js	46.4 kB	
other shared chunks (total)	1.99 kB	

- o (Static) prerendered as static content
- f (Dynamic) server-rendered on demand

# Automatic Static optimization

Problem:

Static wird gecached

./ wird nicht neu validiert

## Automatic Static optimization

### Problem:

Static wird gecached  
./ wird nicht neu validiert

### Lösung:

Incremental Static Regeneration **ISR**  
`revalidatePath("/")`  
`export const revalidate = 600`

## Automatic Static optimization

<u>Route (app)</u>	<u>Size</u>	<u>First Load JS</u>	<u>Revalidate</u>
o /	175 B	105 kB	10m
o /_not-found	145 B	102 kB	
f /api/vorlesungen	145 B	102 kB	
o /faq	175 B	105 kB	
o /merkliste	5.27 kB	110 kB	
o /test-error	326 B	102 kB	
f /vorlesungen/[id]	515 B	102 kB	
o /vorlesungen/erstellen	145 B	102 kB	
+ First Load JS shared by all	102 kB		
chunks/20f27031-3b4977d31a17978d.js	53.2 kB		
chunks/691-d71a779478597b53.js	46.4 kB		
other shared chunks (total)	1.99 kB		

Static

# Fragen?

# Built-in Optimizations

Image, Font, Script und mehr...



# Optimierungen in Next.js

- ✓ Schriftoptimierung
- ✓ Skriptoptimierung
- ✓ Code-Splitting & Lazy Loading
- ✓ Automatische statische Optimierung

## Image Optimization mit next/image

```
import Image from 'next/image'  
import logo from '../assets/logo.png'  
  
<Image src={logo} alt="Logo" />
```

## Was wird optimiert?

- ✓ Größenanpassung & Formatkonvertierung
- ✓ Responsive Bilder
- ✓ Lazy Loading
- ✓ Vermeidung CLS

# Fragen?

Quiz

# Wer Wird Next Millionär?

[quiz.hdmcsm.dev](http://quiz.hdmcsm.dev)

**Viel Spaß mit NextJS!**

Danke!