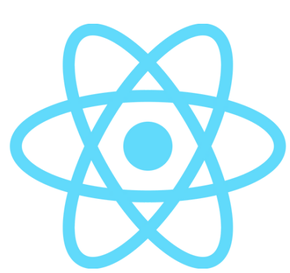
Bastian Jakobi, Marcel Willie, Marvin Pfau

React & Next.js



Wer wir sind

Marcel Willie

2 Jahre mit React
2 Jahre mit Next.js

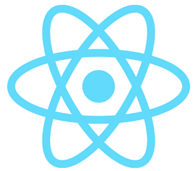
Bastian Jakobi

3 Jahre mit React
1 Jahr mit Next.js

Marvin Pfau

2 Jahre mit React
Neuling in Next.js

Was ist React?



JavaScript-Bibliothek für den Aufbau von Benutzeroberflächen

Entwickelt von Facebook, veröffentlicht 2013

Komponenten basierte Architektur

Vorteile von React

- Wiederverwendbare Komponenten
- Virtuelles DOM für bessere Leistung und effiziente Updates
- Gleichzeitige Entwicklung von Web-Apps und mobilen Anwendungen

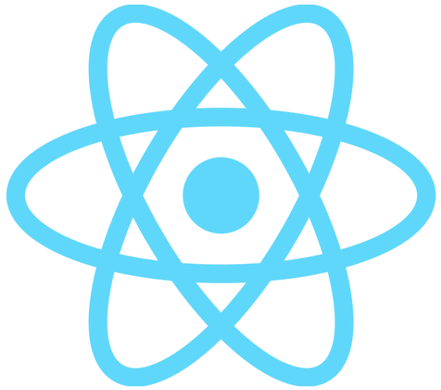
Und was ist Next.js?

Entwickelt von Vercel
(ehemals ZEIT) im Jahr 2016

Erleichtert das Erstellen
von React-basierten
Anwendungen im
serverseitigem Rendering

Wer benutzt Next.js?

Zeitplan



Vorstellung React & Typescript

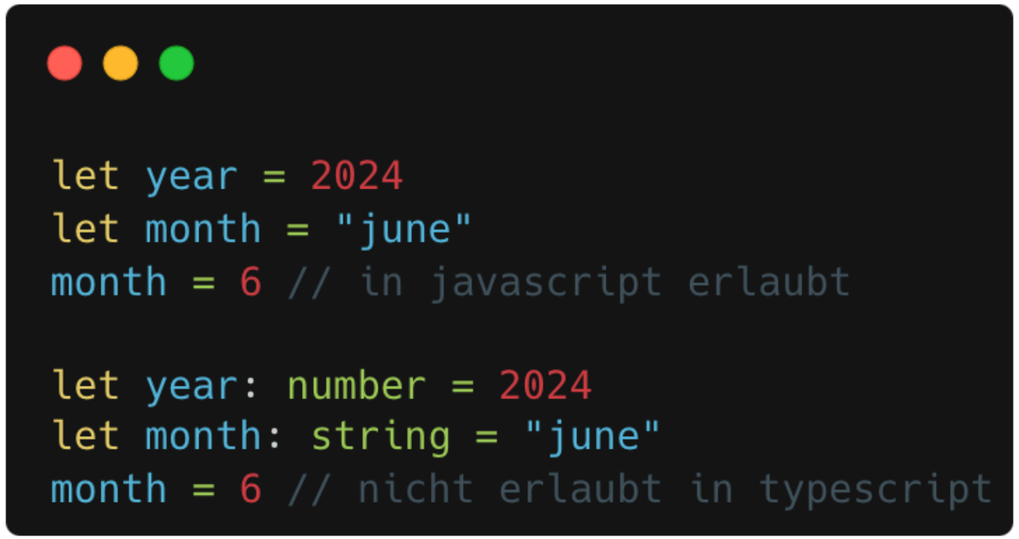
Jetzt

Vorstellung Next.js

Nächste zwei Stunden

Typescript vs. Javascript

Statisch typisiert
vs.
Dynamisch typisiert



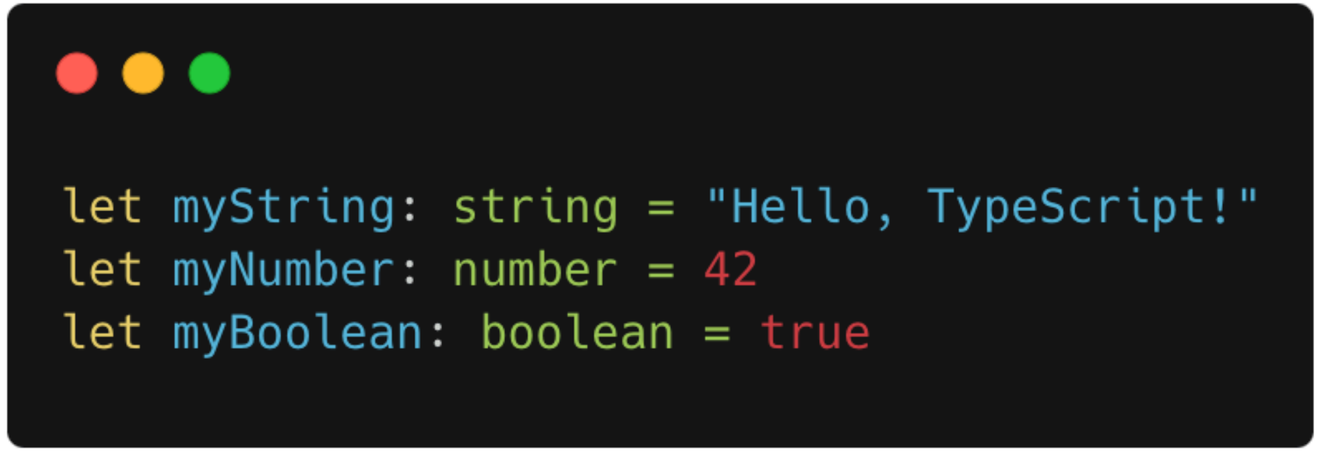
```
let year = 2024
let month = "june"
month = 6 // in javascript erlaubt

let year: number = 2024
let month: string = "june"
month = 6 // nicht erlaubt in typescript
```

Typescript vs. Javascript

- Bessere IDE-Unterstützung
- Erhöhte Code Qualität und Wartbarkeit

Types: Primitive Types

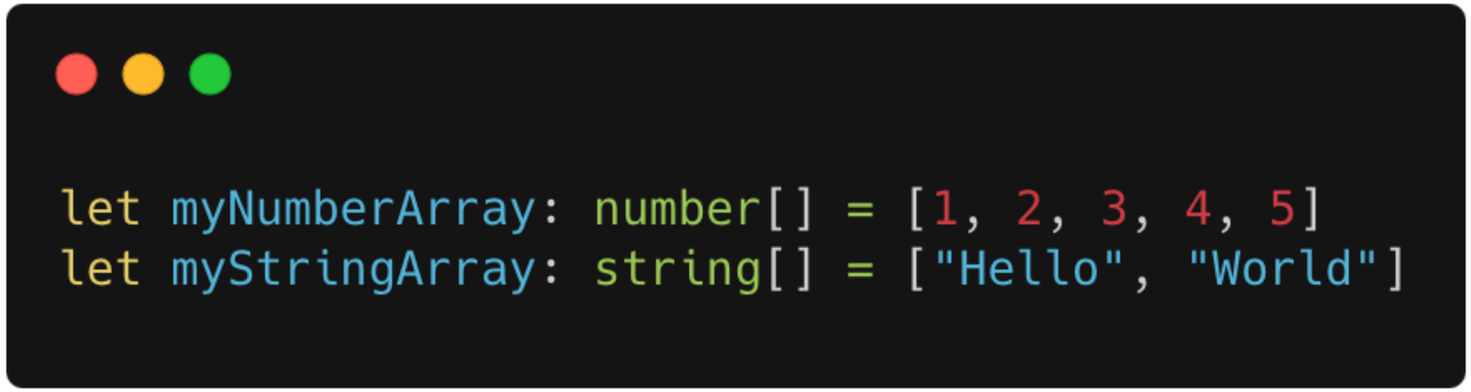
- string
- number
- boolean



```
let myString: string = "Hello, TypeScript!"
let myNumber: number = 42
let myBoolean: boolean = true
```

Types: Arrays

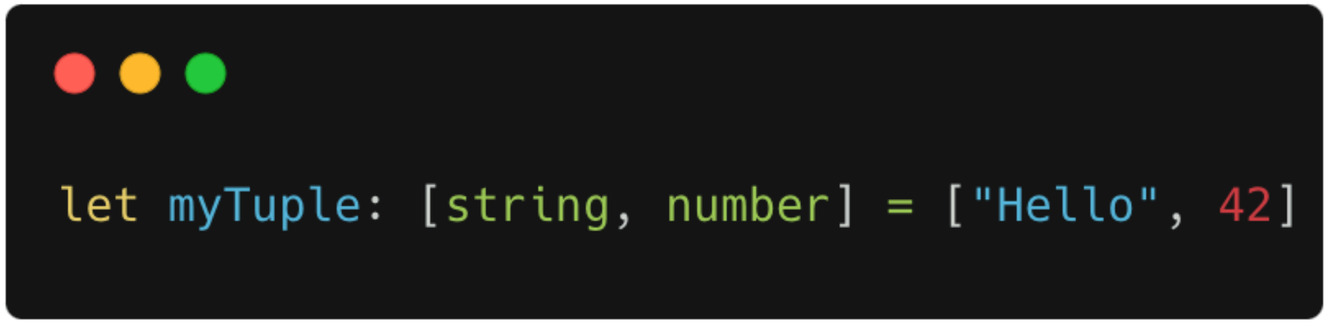
- number[]
- string[]



```
let myNumberArray: number[] = [1, 2, 3, 4, 5]
let myStringArray: string[] = ["Hello", "World"]
```

Types: Tuples

- [string, number]



Types: Enums

- enum Color{Red, Green, Blue}

```
enum Color {
  Red,
  Green,
  Blue
}

let myColor: Color = Color.Green
```

Types: Weitere

- Any
- null und undefined
- object
- Custom Types bei Props

Aufgabe

00

5 min

Wie ist ein Projekt strukturiert?

Aufbau einer Basic UI

```
import Blog from './components/Blog/Blog'

function App() {
  return (
    <>
      <h1 className='header'>My Cool React Blog</h1>
      <Blog />
    </>
  )
}

export default App
```

Komponenten mit Props

```tsx
import './PostCard.css'

interface Post {
  id: number;
  title: string;
  body: string;
  imagePath: string;
}

interface PostCardProps {
    post: Post
}

function PostCard(props: PostCardProps) {
    const { post } = props
    return (
        <div className="card">
            <img src={'/' + post.imagePath} alt={post.title} className="card-image"/>
            <h2>
                {post.title}
            </h2>
            <p>
                {post.body.substring(0, 100)}...
            </p>
        </div>
    )
}

export default PostCard
```

Komponenten mit Props

```
import PostCard from
'../Post/PostCard'
export default function Post() {

  /* ... load post obj */

    return (
        <PostCard post={post} />
    )
}
```

Conditinal-Rendering

```
import PostCard from '../Post/PostCard'

export default function Post() {

  /* ...  */

    if(post == undefined){
        return(<p>Post not found</p>)
    }
    return (
        <>
            <h2>post.title</h2>
            <PostCard post={post} />
        </>
    )
}
```

Conditinal-Rendering

```
import PostCard from "../Post/PostCard";

export default function Post() {
    /* ...  */

    return (
        <>
            {post != undefined ? (
                <>
                    <h2>post.title</h2>
                    <PostCard post={post} />
                </>
            ) : (
                <p>Post not found</p>
            )}
        </>
    );
}
```

Conditinal-Rendering

```
import PostCard from "../Post/PostCard";

export default function Post() {
    /* ...  */

    return (
        <>
            {post && (
                <>
                    <h2>post.title</h2>
                    <PostCard post={post} />
                </>
            )}
        </>
    );
}
```

Aufgabe

01

5 min

Listen rendern in React – map

- ‘map’ ist eine Methode, die auf Arrays angewendet wird.
- Sie erstellt ein neues Array, indem sie eine Funktion auf jedes Element des ursprünglichen Arrays anwendet.

Listen rendern in React – map

```jsx
import React from 'react';

const names = ['Alice', 'Bob', 'Charlie'];

function NameList() {
  return (
    <ul>
      {names.map((name, index) => (
        <li key={index}>{name}</li>
      ))}
    </ul>
  );
}

export default NameList;
```

React Hooks

- **Hooks sind Funktionen, die innerhalb einer funktionalen Komponente aufgerufen werden können**
- **Erlauben Zugriff auf State, Side Effects und andere Lebenszyklus Events einer React Komponente**

*“Hooks are functions that let you **“hook into”** React state and lifecycle features[...]”*

React Hook: useState

- ‘useState’ ermöglicht, State in funktionalen Komponenten zu verwalten.
- Gibt nach dem Aufruf zwei Werte zurück: den aktuellen Zustand und eine Funktion, um diesen Zustand zu aktualisieren.

React Hook: useState

```tsx
import React, { useState } from 'react';

function Counter() {
  // Deklariere eine neue Zustandsvariable namens
"cconst [count, setCount] = useState<number>(0);

  return (
    <div>
      <p>Du hast {count} Mal geklickt</p>
      <button onClick={() => setCount(count + 1)}>
        Klicke mich
      </button>
    </div>
  );
}

export default Counter;
```

React Hook: useEffect

- ‘useEffect’ erlaubt das ausführen von Seiteneffekten in funktionalen Komponenten.
- Typische Anwendungsfälle: Datenabruf, manuelle DOM-Manipulation, Abonnements.

React Hook: useEffect

```tsx
import React, { useState, useEffect } from 'react';

function DataFetchingComponent() {
  const [data, setData] = useState<any[]>([]);
  const [loading, setLoading] = useState<boolean>(true);

  useEffect(() => {
    // Fetch-Daten, wenn die Komponente gemountet wird
    fetch('https://api.example.com/data')
      .then(response => response.json())
      .then(data => {
        setData(data);
        setLoading(false);
      })
      .catch(error => {
        console.error('Fehler beim Abrufen der Daten:', error);
        setLoading(false);
      });
  }, []); // Leeres Array als Abhängigkeit, damit der Effekt nur einmal ausgeführt wird

  if (loading) {
    return <p>Loading...</p>;
  }

  return (
    <ul>
      {data.map(item => (
        <li key={item.id}>{item.name}</li>
      ))}
    </ul>
  );
}

export default DataFetchingComponent;
```

Aufgabe

02

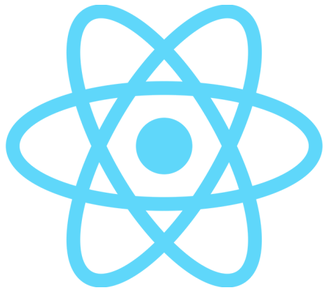
30 min

Recap

- Typscript-Basics
- React-Projekt Aufbau
- React-Komponenten inkl. Props
- Rendern von Listen
- Hooks für Data-Fetching und useState

Unterschiede

zwischen Next.js und React



×

React

- Bibliothek
- Erstellung von Benutzeroberflächen
- Fokus auf Komponentenlogik und UI-Rendering
- Rendering: Client
- Routing: extra React Router

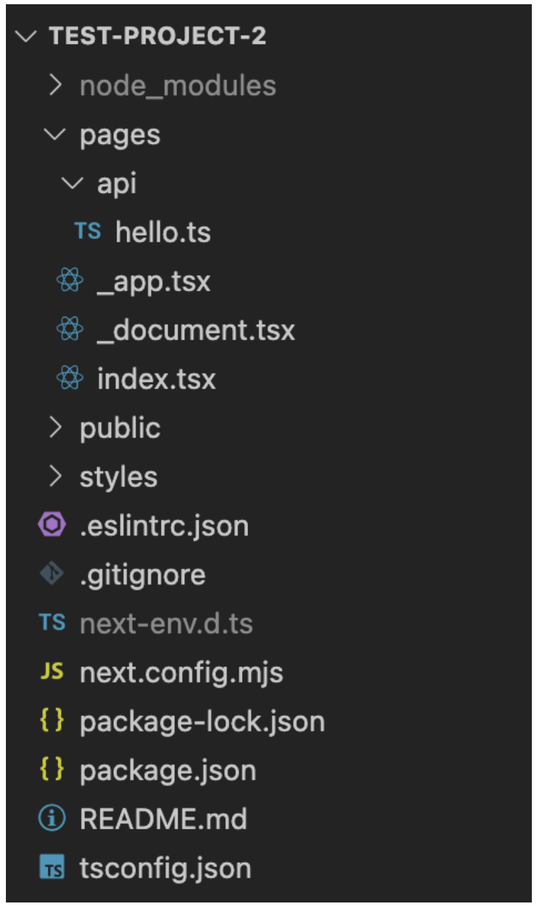
Next.js

- Framework
- Baut auf React auf
- Serverseitige Funktionen (bspw. API)
- Rendering: Client, Server, Static
- Routing: file-based System
- Code-Splitting

Projektstruktur

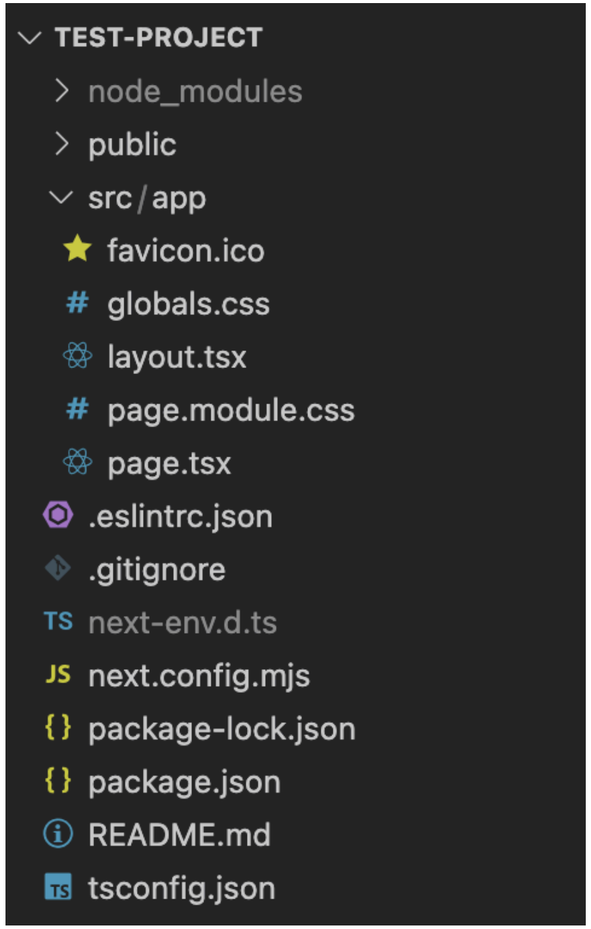
Next.js

V1



/pages

V2



/src/app/

Grundlegende Verzeichnisse

/public/ → statische Assets

/src/ → Hauptcontainer der Anwendung

/src/app/ → enthält alle Komponenten, separate Seiten der Anwendung

Optionale Verzeichnisse

/src/app/api/ → Ordner für API-Routen

/src/components/ → Ordner für wiederverwendbare Komponenten

/src/styles/ → Ordner für wiederverwendbare CSS-Styles

/src/utils/ → Ordner für wiederverwendbare Hilfsfunktionen

/src/models/ → Ordner für eigene Typendefinitionen und Interfaces

Konfigurationsdateien

- **next.config.js**: Konfiguration Next.js
- **babel.config.js**: Hinzufügen und Konfiguration von Plugins
- **tsconfig.json**: TypeScript-Konfigurationen
- **.env.local**: Lokale Environment Variablen

Aufgabe

03

5 min

File-based Routing

- Routing durch Dateistruktur definiert
- Jedem Ordner innerhalb des /src/app/-Verzeichnisses wird einer URL-Route zugeordnet
- Die **page.tsx** ist die Hauptdatei eines Ordners/Route
- Ordner können beliebig verschachtelt werden
- Kein zusätzliches Routing-Setup erforderlich
- Direkt ersichtlich, welche Datei zu welcher Route gehört

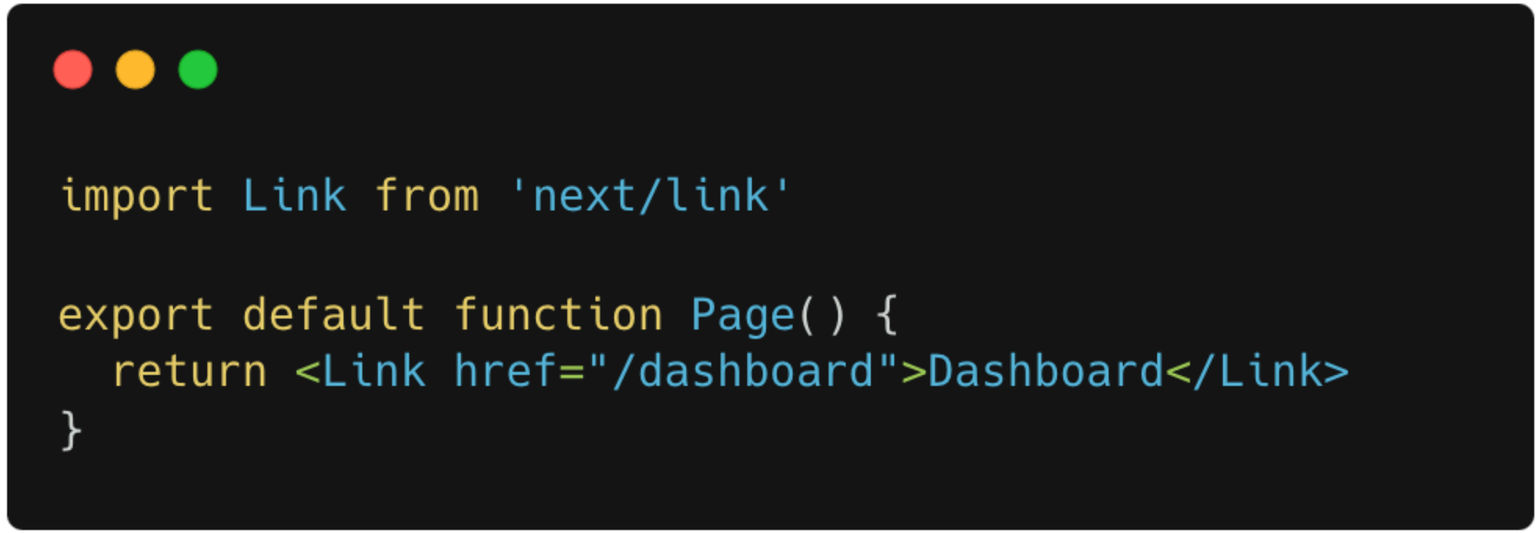
Beispiele

- **/src/app/page.tsx** → Haupt-URL **/**
- **/src/app/about/page.tsx** → Route **/about**
- **/src/app/products/item/page.tsx** → Route **/products/item**

<Link/> Element

- <Link> ist eine React-Komponente, die das HTML-Element <a> erweitert
- Nutzt Prefetching für bessere Performance
- Primäre Möglichkeit, zwischen Routen in Next.js zu navigieren

<Link> Element



```
import Link from 'next/link'

export default function Page() {
  return <Link href="/dashboard">Dashboard</Link>
}
```

Dynamisches Routing

- Ein Ordner der Struktur [...] ist dynamisch
- Die Datei **/src/app/blog/[slug]/page.tsx** kann über **/blog/1**, **/blog/2**, etc. aufgerufen werden
- **slug** ist im Komponent über **params** zugänglich

Dynamisches Routing

```
export default function Page({ params }: { params: { slug: string } })
{ return <div>My Post: {params.slug}</div>
}
```

Erweiterte Routing-Optionen

- **Fallback-Routen:** error.ts, global-error.ts, not-found.ts (404-Seite)
- **Middleware:** middleware.ts - Logik bevor eine Seite gerendert wird (Authentifizierung, Redirects, etc.)
- **Shared-Layout:** layout.ts - Übergeordnetes Layout
- **Loading-Screen:** loading.ts - Vorgeschaltete Ladeanimation
- **API-Routen:** route.ts - serverseitige Routen, um API-Anfragen zu verarbeiten

error.ts

```tsx
'use client' // Error components must be Client Components

import { useEffect } from 'react'

export default function Error({
  error,
  reset,
}: {
  error: Error & { digest?: string }
  reset: () => void
}) {
  useEffect(() => {
    // Log the error to an error reporting service
    console.error(error)
  }, [error])

  return (
    <div>
      <h2>Something went wrong!</h2>
      <button
        onClick={
          // Attempt to recover by trying to re-render the segment
          () => reset()
        }
      >
        Try again
      </button>
    </div>
  )
}
```

middleware.ts

```ts
import { NextResponse, NextRequest } from 'next/server'

// This function can be marked `async` if using `await` inside
export function middleware(request: NextRequest) {
  return NextResponse.redirect(new URL('/home', request.url))
}

export const config = {
  matcher: '/about/:path*',
}
```

layout.ts

```tsx
export default function Layout({
  children,
}: {
  children: React.ReactNode
}) {
  return <section>{children}</section>
}
```

loading.ts

```tsx
export default function Loading() {
  // Or a custom loading skeleton
  return <p>Loading...</p>
}
```

route.ts

Aufgabe

04

30 min

API Route

- Ordner unter **/src/app/api/** werden automatisch als API-Endpunkte behandelt
- Der Pfad bestimmt die URL des Endpunkts
- Die **route.ts** Datei definiert die Logik des Endpunkts
- Beispiel: **/src/app/api/users/route.ts** ist über die URL **/api/user** erreichbar
- Der Name der export Funktion definiert die Funktionalität des Endpunkts (GET, POST, etc.)

route.ts

```ts
export async function GET(request: Request) {}

export async function HEAD(request: Request) {}

export async function POST(request: Request) {}

export async function PUT(request: Request) {}

export async function DELETE(request: Request) {}

export async function PATCH(request: Request) {}

// If `OPTIONS` is not defined, Next.js will automatically implement `OPTIONS` and  set the appropriate Response `Allow` header depending on the other methods defined in the route handler.
export async function OPTIONS(request: Request) {}
```

API Route

```
export async function GET(req: NextApiRequest, res: NextApiResponse)
{   try {
        res.json({status: 200, message:"response message"})
    } catch (e) {
        console.log(e);
        res.json({status: 500,  message:"error message"})
    }
}
```

API Route

```
export async function POST(req: Request, res: NextApiResponse)
{   try {
        const data: any = await req.json()
        const articleBody: string = data.articleBody
        const articleTitle: string = data.articleTitle
        await addArticleToDB(articleTitle,articleBody)
        res.json({status: 200, message:"response message"})
    } catch (e) {
        console.log(e);
        res.json({status: 500,  message:"error message"})
    }
}
```

Dynamische API Route

- Wie bei den Seiten können API-Routen dynamische Pfade verwenden
- z.B. **/src/app/api/posts/[id]/route.ts**, um Operationen auf einen spezifischen Beitrag durchzuführen
- die id kann über **params** abgerufen werden

```
export async function GET(req: NextApiRequest, { params }: { params: { id: string } }, res:
NextApiResponse) {
    try {
        const { id } = params;
        const article = await getArticle(id);
        res.json({status: 200, data: article})
    } catch (e) {
        console.log(e);
        res.json({status: 500, data: e.message})
    }
}
```

Aufgabe

05

15 min