

# Next.js

Denis Uspenskii TINF24B4

# Agenda

1. Next.js / node.js / React
2. Installation
3. Projektstruktur
4. Dateibasiertes Routing
5. Link
6. Image
7. Server and Client components
8. Not found page
9. Loading state
10. Error page
11. API

# 1. Next.js / node.js / React

**React** = UI-Bibliothek für Komponenten im Browser

**Next.js** = React-Framework (läuft auf Node)

**Node.js** = JavaScript-Runtime auf dem Server

## 2.1. Installation von Node (Mac OS)

# Download and install fnm:

```
curl -o- https://fnm.vercel.app/install | bash
```

# Download and install Node.js:

```
fnm install 25
```

# Verify the Node.js version:

```
node -v # Should print "v25.0.0"
```

# Verify npm version:

```
npm -v # Should print "11.6.2"
```

## 2.2. Installation von Node (Linux)

# Download and install fnm:

```
curl -o- https://fnm.vercel.app/install | bash
```

# Download and install Node.js:

```
fnm install 25
```

# Verify the Node.js version:

```
node -v # Should print "v25.0.0".
```

# Verify npm version:

```
npm -v # Should print "11.6.2".
```

## 2. Next.js starten

**npx create-next-app@latest my\_app** (lowercase)

ODER

**npx create-next-app@latest .**

UND DANN

**npm run dev**

(<http://localhost:3000/>)

✓  **test** ~/Documents/DHBW/3 Semester/Software Engineering/test


>  .idea

>  .next

>  app


>  node\_modules library root


>  public

 .gitignore

 eslint.config.mjs

 next.config.ts


 next-env.d.ts

 package.json

 package-lock.json

 postcss.config.mjs

 README.md

 tsconfig.json

## 3.1. Projektstruktur

## 3.2. Projektstruktur

`tsconfig.json` == TypeScript-Konfiguration

`postcss.config.mjs` == Konfigurationsdatei für PostCSS

`package-lock.json` == Versionen aller npm-Pakete

`package.json` == Projekt-Definition

`next-env.d.ts` == TS-Deklarationsdatei für Next.js.

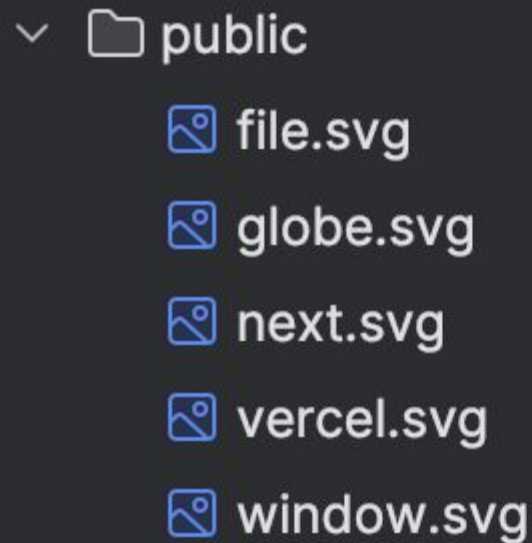
`next.config.ts` == Projektweite Next-Konfiguration

`eslint.config.mjs` == ESLint-Konfiguration



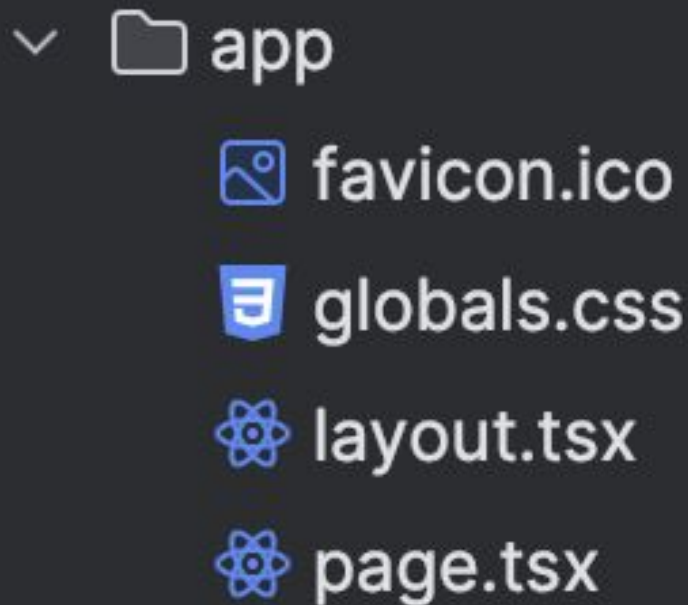
## 3.3. public folder

- Ordner für statische Dateien
- öffentlich erreichbar über eine URL wie  
`localhost:3000/file.svg`

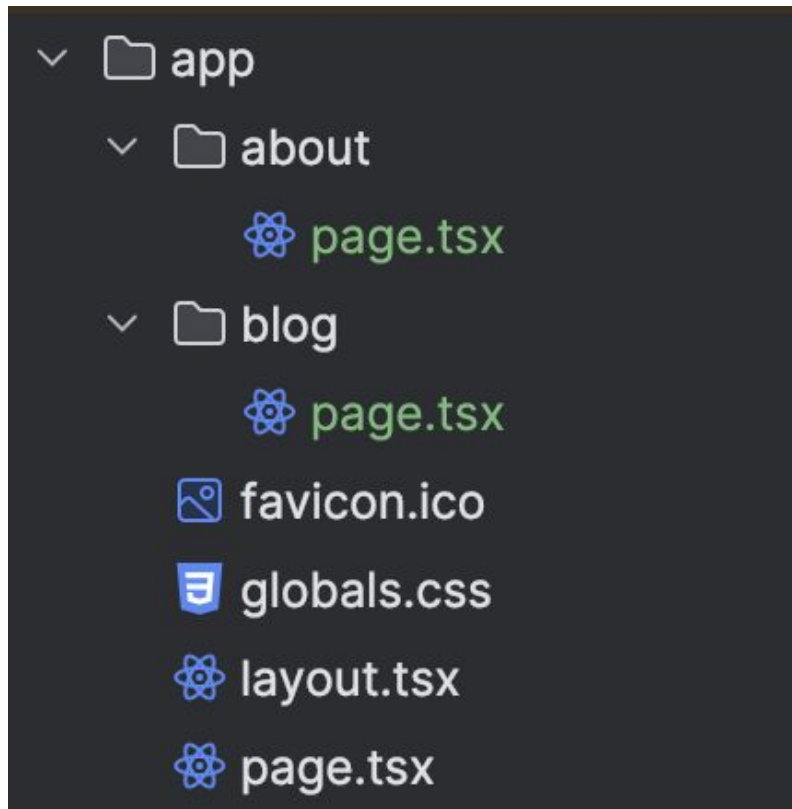
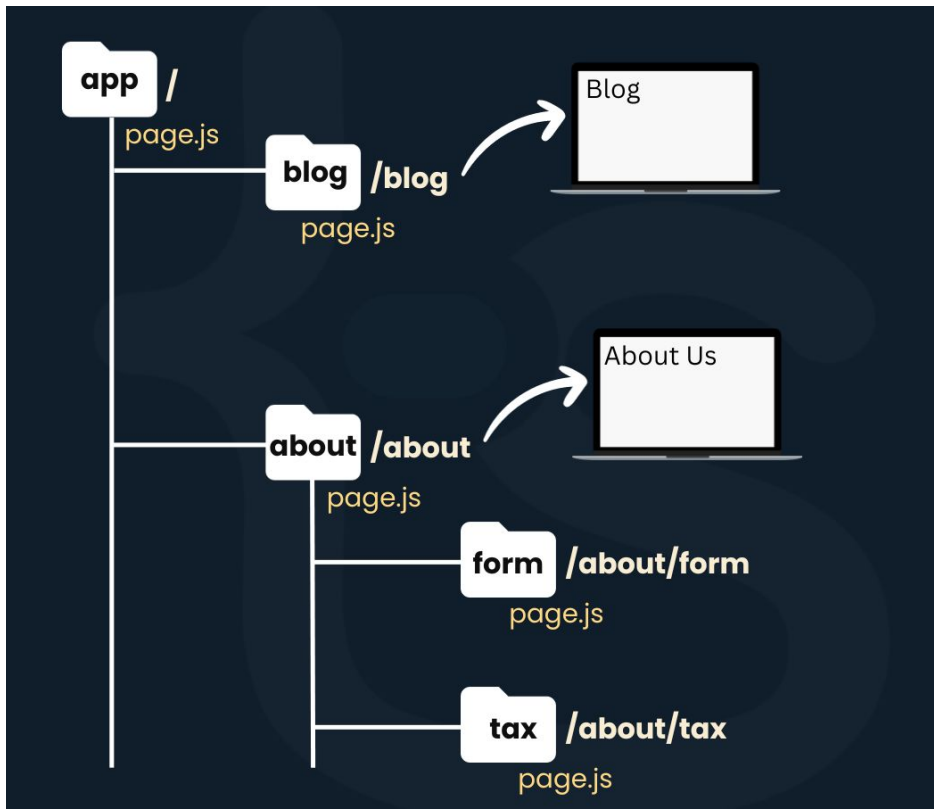


## 3.4. app folder

- favicon.ico ersetzen
- Globale CSS
- Metadata in Layout
- Start Page



## 4. Dateibasiertes Routing



## 5. Link

```
import Link from 'next/link';
```

```
<Link> About </Link>
```

```
<Link href="/about" className="nav-link">About</Link>
```

## 6.1. Image

```
import Image from "next/image";
```

```
<Image
```

```
  className="dark:invert"
```

```
  src="/next.svg" // aus public
```

```
  alt="Next.js logo"
```

```
  width={100}
```

```
  height={20}
```

```
  priority
```

```
/>
```

## 6.2. Image

Um Bilder aus externen Sources zu benutzen, muss man die Sources in **next.config.ts** bestimmen.

```
images: {  
  remotePatterns: [  
    {hostname: "..."}  
  ]  
}
```

## 7.1. Server and Client components

Jede erstellte Funktionskomponente ist standardmäßig eine Serverkomponente

„**use client**“; oben wandelt eine Serverkomponente in eine Client-Komponente.

## 7.2. Server and Client components

HTML wird auf dem Server gerendert und dann an einen Client gesendet.

```
<button onClick={() => alert(„Hello!“)}> Click me  
</button>
```

ohne „**use client**“ funktioniert nicht



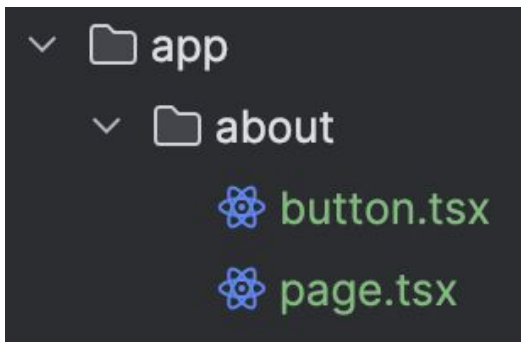
## 7.3. Vorteile der Serverkomponente:

- Das Laden der ersten Seite erfolgt wesentlich schneller.
- Besser für SEO
- Serverkomponenten können in asynchrone Komponenten umgewandelt werden

```
export default function async About()
```

## 7.4. button onClick und async Funktion?

Separate Komponente!



```
return (  
  <div>  
    <h1>About</h1>  
    <p>This is the about page.</p>  
    <ButtonComponent />  
  </div>  
);
```

```
"use client";
```

```
export default function ButtonComponent() : Element { Show usages new *  
  return <button onClick={() : void => alert( message: "Hello!")}> Klick me </button>  
}
```

## 8. Not found page

**not-found.tsx** kreieren und einstellen:

```
export default function NotFoundPage() {  
  return (  
    <h1>NOT FOUND </h1>  
  );  
}
```

## 9. Loading state

**loading.tsx** kreieren und einstellen:

```
export default function LoadingState() {  
  return (  
    <h1>Loading...</h1>  
  );  
}
```

## 10. Error page

**error.tsx** kreieren und einstellen:

```
"use client";
```

```
export default function Error() {  
  return <div>Error!!</div>  
}
```

# 11. API

- API Ordner kreieren
- Endpoint bestimmen
- route.ts einstellen:

```
import {NextResponse} from "next/server";

export async function GET() : Promise<NextResponse<{ message: string }>... { Show usages new *
  return NextResponse.json({message: "Hello from api!"});
}

export async function POST(req: Request) : Promise<NextResponse<{ message: string }>... { Show usages new *
  const data : any = await req.json();
  const {name} = data;
  return NextResponse.json({message: `Hello ${name}, this was sent from the api!`});
}
```

Git zur Repo:

<https://github.com/Pienotdie/learnandshare.git>