

Next.js Grundlagen und Routing

Lernziele

- ☐ Den Unterschied zwischen einer Library und einem Framework kennen
 - ☐ Verstehen, warum Next.js ein so beliebtes Framework ist
 - ☐ Wissen, welche Features im Bootcamp verwendet werden
 - ☐ Die grundlegenden Konzepte von Next.js verstehen:
 - ☐ Client-side Routing
 - ☐ Seitennavigation mit `next/link`
 - ☐ Bildoptimierung mit `next/image`
-

Unterschied zwischen einer Library und einem Framework

Next.js ist ein React-Framework, das auf der React-Library aufbaut. Sowohl eine Library als auch ein Framework sind wiederverwendbarer Code, der von jemand anderem geschrieben wurde. Ihr Zweck ist es, dir zu helfen, häufige Probleme auf einfachere Weise zu lösen.

Wenn du eine Library verwendest, hast du die Kontrolle darüber, welche Teile des Codes du wann verwendest. Im Gegensatz dazu ist ein Framework wie Next.js strenger: Du hast nur begrenzte Wahlmöglichkeiten, wann und wie du den gegebenen Code verwendest. Wenn du diese Einschränkungen akzeptierst, nimmt dir ein Framework viel Arbeit im Hintergrund ab.

Was ist Next.js?


Next.js ist ein React-Framework, das dir Bausteine zur Erstellung schneller Webanwendungen bietet. Diese Bausteine liefern vorgefertigte Lösungen für die Hauptkonzepte, auf die du beim Erstellen moderner Anwendungen stößt, wie Benutzeroberfläche, Routing, Datenabruf, Infrastruktur usw.

 Lies mehr über [Next.js auf der Next.js Homepage](#).

Welche Features von Next.js werden wir im Bootcamp verwenden?

Next.js wird uns bei folgenden Themen helfen:

- Eine Vorlage als Ausgangspunkt
- Ein Bundler, Transpiler und Entwicklungsserver
- Routing: Navigation zwischen Seiten, dynamisches Routing
- Automatisch optimierte Bilder
- API-Routen

 Next.js hat noch viel mehr zu bieten, was es zu einem so beliebten Framework macht. Um einen Eindruck von allen Features zu bekommen, [wirf einen kurzen Blick in die Dokumentation](#).

How to Next.js: Grundlagen

Unterschiede zu Create React App

Hier siehst du einen Vergleich einiger relevanter Unterschiede zwischen Next.js und Create React App (CRA):

	Next.js (neu)	Create React App (alt)
Start lokaler Dev-Servers	<code>npm run dev</code>	<code>npm run start</code>
Root-Komponente	<code>_app.js</code>	<code>App.js</code>
Dokument	<code>_document.js</code>	<code>public/index.html</code>
Standard-Styling	CSS Modules*	CSS*
Rendering	Server- und Client-seitig	Client-seitig
Routen-Definition	Dateistruktur im <code>pages</code> -Ordner	n/a
Client-seitige Links	<code><Link></code> -Komponente	n/a
Bildoptimierung	<code><Image></code> -Komponente	n/a
Modifikation von <code><head></code>	<code><Head></code> -Komponente	n/a
Schriftarten laden	<code>@next/font</code> -Paket	n/a
API-Routen	<code>pages/api</code> -Ordner	n/a
ESLint	Next.js-spezifische Regeln	CRA-spezifische Regeln
Bundler + Transpiler	Webpack/Turbopack + SWC	Webpack + Babel

* Sowohl Next.js als auch CRA unterstützen alle modernen Styling-Lösungen.

Server-Side Rendering

Mit CRA lädt der Browser ein fast leeres HTML-Dokument (`public/index.html`). Dein React-Code wird nur im Browser ausgeführt.

Next.js kommt mit einer Funktion namens "Server-side Rendering". Diese Funktion führt deine React-Komponenten auf dem Server aus, um ein vollständiges HTML-Dokument an den Client (den Browser) zu senden. Auf dem Client wird dein React-Code dann erneut ausgeführt.

Dies ermöglicht viele Optimierungstechniken, die wir hier nicht besprechen werden. Es gibt jedoch eine wichtige Implikation, die du kennen musst:

Da dein React-Code in einer Serverumgebung und nicht nur in einer Browserumgebung ausgeführt wird, musst du vorsichtig sein, wenn du Browser-APIs (wie `window` oder `document`) verwendest. Diese sind nur im Browser verfügbar und werden die App auf dem Server zum Absturz bringen. Wenn du eine Browser-API verwendest, musst du sicherstellen, dass dein Code nur auf dem Client ausgeführt wird. Zum Beispiel wird Code innerhalb eines `useEffect`-Hooks nur auf dem Client ausgeführt, da Effekte nicht auf dem Server, sondern nur auf dem Client ausgeführt werden. Event-Handler wie `onClick` werden ebenfalls nur auf dem Client ausgeführt.

```
useEffect(() => {
  console.log(window.innerWidth);
}, []);

return <button onClick={() => console.log(window.innerWidth)}>Click
me</button>;
```

Routing

Bisher haben unsere React-Anwendungen nur eine einzelne Seite angezeigt. Der Prozess des bedingten Renderns verschiedener Seiten basierend auf der URL (Pfadname) und der Navigation zwischen diesen Seiten wird als Routing bezeichnet.

Da eine gute Routing-Lösung nicht einfach zu erstellen ist, verlassen sich fast alle React-Entwickler auf eine externe Routing-Library. Zum Beispiel ist `react-router` eine sehr beliebte Lösung.

Next.js bietet Routing als integriertes Feature, sodass du keine weitere Library benötigst.

Das Routing in Next.js basiert auf dem Dateisystem im `pages`-Ordner:

- `pages/index.js` → `/` (`index.js` impliziert immer die Root-Route eines Ordners)
- `pages/about.js` → `/about`

Um komplexere Routen zu unterstützen, kannst du die entsprechende verschachtelte Ordnerstruktur erstellen:

- `pages/about/me.js` → `/about/me`
- `pages/about/all-others.js` → `/about/all-others`
- `pages/about/some/long/route.js` → `/about/some/long/route`

💡 Du kannst auch dynamische Routen definieren (Routen, die dynamische Parameter haben). Dies wird ein Thema einer zukünftigen Sitzung sein.

💡 Dateibasiertes Routing kann uneindeutig sein. Die Dateien `pages/about/index.js` und `pages/about.js` sind beide mit `/about` verknüpft. In der Praxis ist dies selten ein Problem. Trotzdem solltest du dir dessen bewusst sein.

📖 Lies mehr über [Routing in den Next.js-Dokumenten](#).

<Link>-Komponente

Für client-seitige Übergänge zwischen Routen verwende die `<Link>`-Komponente, die von `next/link` bereitgestellt wird. Angenommen, ein `pages`-Verzeichnis enthält

- `pages/index.js`
- `pages/about.js`
- `pages/about/me.js`,

Du kannst zu jeder dieser Seiten auf folgende Weise verlinken:

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

export default function Navigation() {
  return (
    <ul>
      <li>
        <Link href="/">Home</Link>
      </li>
      <li>
        <Link href="/about">About</Link>
      </li>
      <li>
        <Link href="/about/me">Me</Link>
      </li>
    </ul>
  );
}
```

 Lies mehr über [next/link](#) in den Next.js-Dokumenten.

Bildoptimierung mit `next/image`

Next.js kommt mit einer automatischen Bildoptimierung – der `next/image`-Komponente. Diese Funktion vermeidet zum Beispiel das Ausliefern großer Bilder an Geräte mit einem kleineren Viewport und Bilder werden standardmäßig lazy-loaded. Beachte, dass die Verwendung von `<Image>` mit ein wenig Standard-Styling verbunden ist.

Hier ist eine Beispielimplementierung. Beachte, dass die `height`- und `width`-Props die gewünschte Rendergröße sein sollten, mit einem Seitenverhältnis, das mit dem Quellbild identisch ist.

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

export default function AnimalImage() {
  <Image
    src="/images/a_small_dog.jpg"
    height={144}
    width={144}
    alt="Ein Bild eines kleinen Hundes"
  />;
}
```

Wenn du Bilder von externen Domains verwendest, musst du die erlaubten Domains in der `next.config.js`-Datei konfigurieren:

```
const nextConfig = {
  reactStrictMode: true,
  images: {
    domains: ["images.unsplash.com"],
  },
}
```

```
  },  
};
```

 Lies mehr über die [Konfiguration von Domains](#) für `next/image` und `next/image` im Allgemeinen in den [Next.js-Dokumenten](#).

Resources

- [Next.js Homepage](#)
- [Next.js Docs](#)
- [Routing in the Next.js Docs](#)
- [next/link in the Next.js Docs](#)
- [next/image in the Next.js Docs](#)
- [Difference between a Framework and a Library on freecodecamp](#)