Next.js Dynamische Routen

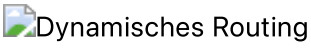
Lernziele

- [ ] Das Konzept von dynamischen Pfaden verstehen
- [ ] Wissen, wie man dynamische Pfade implementiert
- [ ] Wissen, wie man Links dynamisch generiert
- [ ] Wissen, wie man imperative Routing verwendet
- [ ] Das `next/head`-Komponente verstehen

Konzept von Dynamischen Routen

Wenn deine App viele mögliche Routen mit wiederkehrenden Mustern hat, wäre es schwierig bis unmöglich, für jede Route eine JavaScript-Datei zu erstellen.

Mit dynamischen Routen kannst du (Teile) des Pfads in dynamische Parameter umwandeln, indem du eckige Klammern verwendest. Wenn die URL mit dem Pfad übereinstimmt, macht Next.js die dynamischen Abfrageparameter mit dem `useRouter`-Hook verfügbar.

 Dynamisches Routing

Implementierung von Dynamischen Routen

Um eine dynamische Route zu erstellen, kannst du eckige Klammern um Datei- oder Ordnernamen im `pages`-Verzeichnis setzen: `pages/movies/[categorySlug]/page/[pageNumber].js`.

Die folgenden Pfade entsprechen diesem Beispiel:

- `movies/romance/page/12`
  - `categorySlug` ist `"romance"`
  - `pageNumber` ist `"12"`
- `movies/action/page/1`
  - `categorySlug` ist `"action"`
  - `pageNumber` ist `"1"`
- `movies/comedy/page/3`
  - `categorySlug` ist `"comedy"`
  - `pageNumber` ist `"3"`

Um auf die Abfrageparameter in einer Komponente zuzugreifen, verwende den `useRouter`-Hook, der aus `next/router` importiert wird:

```
// pages/movies/categories/[categorySlug]/page/[pageNumber].js
import { useRouter } from "next/router";

export default function CategoryPage() {
  const router = useRouter();
  const { categorySlug, pageNumber } = router.query;
```

```
  return (
    <div>
      <p>Category Slug: {categorySlug}</p>
      <p>Page Number: {pageNumber}</p>
    </div>
  );
}
```

Dies gilt natürlich auch für ein einfacheres Beispiel mit einem einzelnen dynamischen Abfrageparameter.

```
// pages/movies/[slug].js
import { useRouter } from "next/router";

export default function MoviePage() {
  const router = useRouter();
  const { slug } = router.query;

  return (
    <div>
      <p>Slug: {slug}</p>
    </div>
  );
}
```

> 💡 Der Name des Abfrageparameters entspricht immer dem Namen der Datei/des Verzeichnisses: `[slug].js` → `const { slug } = router.query;`

> 📙 Lies mehr über **Dynamische Routen** in der Next.js-Dokumentation.

Verlinkung zu Dynamischen Routen

Du kannst die `Link`-Komponente verwenden, um zu dynamischen Pfaden zu verlinken. Verwende String-Interpolation mit einem Template-String, um die dynamischen Abfrageparameter in den Pfad einzufügen.

Betrachte eine `Movies`-Komponente, die eine Liste von Links zu allen Filmen rendert. Interpoliere den `slug` in den Pfad als Abfrageparameter:

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

export default function Movies({ movies }) {
  return (
    <ul>
      {movies.map(({ id, slug, title }) => (
        <li key={id}>
          <Link href={`/movies/${slug}`}>{title}</Link>
        </li>
      ))}
    </ul>
```

```
  );
}
```

> 📙 Lies mehr über **Verlinkung zu dynamischen Pfaden** in der Next.js-Dokumentation.

Imperatives Routing

Die Verwendung von `<Link>` ist die beste Option, wann immer der Benutzer eigenständig durch die App navigiert. Es gibt jedoch Situationen, in denen du keine `Link`-Komponente verwenden kannst, weil du programmatisch zu einer anderen Seite navigieren möchtest. Ein Beispiel für eine **indirekte** Benutzerinteraktion ist das Ändern der Seite nach dem Absenden eines Formulars:

```
import { useRouter } from "next/router";

export default function Form() {
  const router = useRouter();

  function handleSubmit(event) {
    // some data handling … ✨

    router.push("/home");
  }

  return <form onSubmit={handleSubmit}>…</form>;
}
```

> 📙 Lies mehr über **Routing Imperativ** in der Next.js-Dokumentation.

next/head-Komponente

Next.js enthält eine eingebaute `<Head>`-Komponente, um Elemente in den Kopfbereich der Seite einzufügen. Auf diese Weise kannst du ganz einfach die Metadaten der Seite wie den `<title>`-HTML-Tag ändern:

```
import Head from "next/head";

export default function Movies() {
  return (
    <>
      <Head>
        <title>Movies</title>
        <meta name="viewport" content="initial-scale=1.0, width=device-
width" />
      </Head>
      <ul>…</ul>
    </>
  );
}
```

> 📙 Lies mehr über **next/head** in der Next.js-Dokumentation.

Resources

- Dynamic Routes in the Next.js Docs
- Linking to dynamic paths in the Next.js Docs
- Routing Imperatively in the Next.js Docs
- next/head in the Next.js Docs