Create React App erklärt

Coding-Assistent

Das ist das **ideale Szenario** für eine Migration! Da ihr Next.js eigentlich "nur" als überdimensionierten CRA-Ersatz nutzt (ohne SSR und API-Routen), wird der Umstieg auf Vite euer Projekt deutlich vereinfachen und den Entwicklungsprozess beschleunigen.

Ihr befreit euch quasi von dem serverseitigen Ballast, den ihr gar nicht nutzt.

Hier ist der konkrete Schlachtplan für den Wechsel von **Next.js (SPA Modus)** zu **Vite + React Router**, unter Berücksichtigung eures Java Backends.

---

1. Routing ersetzen (Der größte Schritt)

Next.js nutzt ein **dateibasiertes Routing** (Dateien im Ordner `pages` werden automatisch zu Routen). Vite hat kein Routing eingebaut. Ihr müsst also eine Bibliothek installieren. Der Standard ist `react-router-dom`.

Vorgehensweise:

1. **Installieren:** `npm install react-router-dom`
2. `pages` **Ordner auflösen:** Ihr könnt den Ordner `pages` behalten, aber er hat keine magische Funktion mehr. Es sind jetzt normale Komponenten.
3. **Router konfigurieren:** Ihr müsst eine zentrale Datei (meist `App.js` oder `App.tsx`) erstellen, die eure Routen definiert.

Vergleich:

Next.js (Struktur)	Vite + React Router (Code in `App.js`)
`pages/index.js`	`<Route path="/" element={<Home />} />`
`pages/dashboard.js`	`<Route path="/dashboard" element={<Dashboard />} />`
`pages/users/[id].js`	`<Route path="/users/:id" element={<UserDetail />} />`

Code-Beispiel für `src/App.js`:

JavaScript

```
import { BrowserRouter, Routes, Route } from 'react-router-dom';
import Home from './pages/index';         // Pfade anpassen
import Dashboard from './pages/dashboard';

function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/dashboard" element={<Dashboard />} />
        {/* Wildcard für 404 Fehler, falls gewünscht */}
        <Route path="*" element={<div>Seite nicht gefunden</div>} />
      </Routes>
    </BrowserRouter>
  );
}

export default App;
```

---

2. Komponenten anpassen (Suchen & Ersetzen)

Ihr müsst drei spezifische Next.js-Komponenten austauschen, die in Vite nicht existieren.

A. Links (`next/link`)

Next.js nutzt `<Link href="...">`. React Router nutzt `<Link to="...">`.

- **Suche nach:** `import Link from 'next/link'`
- **Ersetze mit:** `import { Link } from 'react-router-dom'`
- **Code ändern:** `<Link href="/about">` wird zu `<Link to="/about">`.

B. Bilder (`next/image`)

Die `<Image />` Komponente von Next.js optimiert Bilder serverseitig. Da ihr keinen Next.js Server mehr habt, fällt das weg.

- **Lösung:** Ersetzt `<Image src="..." />` durch das standardmäßige HTML-Tag `<img src="..." />`.
- **Hinweis:** Die Bilder müssen nun im `public` Ordner liegen oder direkt importiert werden (`import myImg from './assets/bild.png'`).

C. Head Tags (`next/head`)

Falls ihr den Seitentitel dynamisch ändert (z.B. `<title>Dashboard</title>`), braucht ihr einen Ersatz.

- **Lösung:** Nutzt `react-helmet-async` (ähnlich wie `next/head`).

---

3. Verbindung zum Java Backend (Proxy)

In der Entwicklung läuft euer Frontend auf `localhost:5173` (Vite Standard) und euer Java Backend vermutlich auf `localhost:8080`. Um CORS-Fehler zu vermeiden, habt ihr in Next.js wahrscheinlich `rewrites` in der `next.config.js` genutzt.

In Vite macht man das in der `vite.config.js` über `server.proxy`.

Beispiel `vite.config.js`:

JavaScript

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'

export default defineConfig({
  plugins: [react()],
  server: {
    proxy: {
      // Alle Anfragen, die mit /api beginnen, gehen an das Java Backend
      '/api': {
        target: 'http://localhost:8080', // Eure Java Backend URL
        changeOrigin: true,
        secure: false,
      }
    }
  }
})
```

Das bedeutet: Ein Fetch auf `/api/users` im Frontend wird von Vite intern an `http://localhost:8080/api/users` weitergeleitet.

---

4. Umgebungsvariablen

Next.js zwingt euch dazu, Variablen mit `NEXT_PUBLIC_` zu präfixen. Vite nutzt `VITE_`.

1. Benennt in eurer `.env` Datei alle `NEXT_PUBLIC_API_URL` um in `VITE_API_URL`.
2. Ersetzt im Code `process.env.NEXT_PUBLIC_API_URL` durch `import.meta.env.VITE_API_URL`.

---

5. Der Build-Prozess (Deployment)

Da ihr eine SPA baut, wollt ihr am Ende statische Dateien (HTML, CSS, JS), die ihr irgendwo hochladen könnt (z.B. Nginx, Apache oder direkt in den `resources/static` Ordner eures Java Backends).

- **Befehl:** `npm run build`
- **Ergebnis:** Ein Ordner `dist` (bei CRA hieß er `build`).

- **Deployment:** Den Inhalt dieses Ordners könnt ihr 1:1 auf jeden Webserver legen. Es ist kein Node.js Server mehr nötig, um das Frontend auszuliefern.

Zusammenfassung des Plans

1. Neues Vite Projekt erstellen (`npm create vite@latest`).
2. Euren `src` Ordner (Components, Styles, Utils) hineinkopieren.
3. `react-router-dom` installieren und Routing in `App.js` aufbauen.
4. `next/link` und `next/image` entfernen/ersetzen.
5. Proxy für das Java Backend in `vite.config.js` einrichten.

Soll ich dir ein Beispiel zeigen, wie man die Routen für eine typische "Login -> Dashboard -> Detailseite" Struktur mit React Router aufbaut?