

# Recoil in Next.js-Projekt

Recoil State Management in Next.js-Projekt

---

## Wiki für Recoil State Management in Next.js-Projekt

### Überblick

Diese Wiki-Seite dokumentiert den Einsatz und die Konfiguration von Recoil für das State Management in unserem Next.js-Projekt. Recoil wird verwendet, um den Zustand unserer Anwendung auf einfache und effektive Weise zwischen den Komponenten zu teilen.

### Einrichtung

#### Installation

Um Recoil in Ihrem Projekt zu verwenden, installieren Sie es zuerst über npm:

```
1 | npm install recoil
```

#### Einrichten des RecoilRoot

Um Recoil in Ihrer Anwendung zu nutzen, müssen Sie die gesamte Anwendung (oder zumindest den Teil, der den Zustand verwaltet) mit einem `<RecoilRoot>`-Provider umschließen. Dies geschieht typischerweise in der `_app.js`-Datei von Next.js:

```
1 // pages/_app.js
2 import { RecoilRoot } from 'recoil';
3
4 function MyApp({ Component, pageProps }) {
5     return (
6         <RecoilRoot>
7             <Component {...pageProps} />
8         </RecoilRoot>
9     );
10 }
11
12 export default MyApp;
```

## Verwendung von Atomen

Atome sind die grundlegenden Einheiten des Zustands in Recoil. Jedes Atom hat einen eindeutigen Schlüssel und einen Standardwert.

### Definieren eines Atoms

Definieren Sie Atome in einer zentralen Datei, um sie leicht in verschiedenen Komponenten wiederzuverwenden:

```

1 // features/gisStationState.js
2 import { atom } from 'recoil';
3
4 export const gisStationsStaticDistrictState = atom({
5   key: 'gisStationsStaticDistrict',
6   default: [],
7 });

```

### Zustand in Komponenten verwenden

Recoil bietet verschiedene Hooks, um den Zustand in Ihren Komponenten zu verwalten:

- ▶ **useRecoilState** : Gibt ein Zustandspaar zurück (ähnlich wie **useState** ), das den Zustandswert und eine Setter-Funktion enthält.
- ▶ **useRecoilValue** : Gibt nur den aktuellen Zustandswert zurück, ohne die Möglichkeit zur Aktualisierung.
- ▶ **useSetRecoilState** : Gibt nur eine Funktion zum Aktualisieren des Zustands zurück, was die Komponente effizienter macht, da sie nicht bei jeder Zustandsänderung neu gerendert wird.

## Beispiele

### MapComponent.js

In **MapComponent.js** verwenden Sie **useRecoilState** , um den Zustand sowohl zu lesen als auch zu aktualisieren:

```

// components/MapComponent.js
import { useRecoilState } from 'recoil';
import { gisStationsStaticDistrictState } from '../features/gisStationState';

const MapComponent = () => {
  const [GisStationsStaticDistrict, setGisStationsStaticDistrict] = useRecoilS...

```

```
8  const updateDistricts = (districts) => {
9    setGisStationsStaticDistrict(districts);
10   };
11
12   return (
13     // Komponenten-Rendering
14   );
15 }
```

## DataSheet.js

In [DataSheet.js](#) verwenden Sie `useRecoilValue`, um den Zustand nur zu lesen, da diese Komponente den Zustand nicht ändern muss:

```
1 // components/DataSheet.js
2 import React from "react";
3 import { useRecoilValue } from 'recoil';
4 import { gisStationsStaticDistrictState } from '../features/gisStationState';
5
6 function DataSheet() {
7   const GisStationsStaticDistrict = useRecoilValue(gisStationsStaticDistrictSta
8
9   return (
10     <div id="mainDataSheet" className="absolute top-3 right-3 w-1/6 min-w-[300px]
11       <ul>
12         {GisStationsStaticDistrict.map(station => (
13           <li key={station.id}>{station.name}</li>
14         )));
15       </ul>
16     </div>
17   );
18 }
19
20 export default DataSheet;
```

## Debugging

Nutzen Sie Recoil DevTools für eine effektive Zustandsverwaltung und Debugging. Diese Tools bieten Einblicke in den Zustand Ihrer Anwendung und helfen bei der Fehlersuche.