

redux-saga example

Usually splitting your app state into `pages` feels natural, but sometimes you'll want to have global state for your app. This is an example using `redux` and `redux-saga` that works with universal rendering. This is just one way it can be done. If you have any suggestions or feedback please submit an issue or PR.

This example and documentation is based on the [with-redux](#) example.

Deploy your own

Deploy the example using [Vercel](#) or preview live with [StackBlitz](#)



How to use

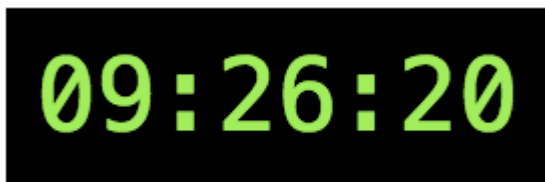
Execute [create-next-app](#) with [npm](#) or [Yarn](#) to bootstrap the example:

```
npx create-next-app --example with-redux-saga with-redux-saga-app
# or
yarn create next-app --example with-redux-saga with-redux-saga-app
# or
pnpm create next-app -- --example with-redux-saga with-redux-saga-app
```

Deploy it to the cloud with [Vercel](#) ([Documentation](#)).

Notes

In the first example we are going to display a digital clock that updates every second. The first render is happening in the server and then the browser will take over. To illustrate this, the server rendered clock will have a different background color than the client one.



Our page is located at `pages/index.js` so it will map the route `/`. To get the initial data for rendering we are implementing the static method `getInitialProps`, initializing the redux store and dispatching the required actions until we are ready to return the initial state to be rendered. Since the component is wrapped with `next-redux-wrapper`, the component is automatically connected to Redux and wrapped with `react-redux Provider`, that allows us to access redux state immediately and send the store down to children components so they can access to the state when required.

For safety it is recommended to wrap all pages, no matter if they use Redux or not, so that you should not care about it anymore in all child components.

`withRedux` function accepts `makeStore` as first argument, all other arguments are internally passed to `react-redux connect()` function. `makeStore` should return a new instance of redux store each time when called, no memoization needed here. See the [full example](#) in the Next Redux Wrapper repository.

The trick here for supporting universal redux is to separate the cases for the client and the server. When we are on the server we want to create a new store every time, otherwise different users data will be mixed up. If we are in the client we want to use always the same store and hydrate it when needed. That's what we accomplish in `store.js`

The clock, under `components/clock.js`, has access to the state using the `connect` function from `react-redux`. In this case Clock is a direct child from the page but it could be deep down the render tree.

The second example, under `components/counter.js`, shows a simple add counter function with a class component implementing a common redux pattern of mapping state and props. Again, the first render is happening in the server and instead of starting the count at 0, it will dispatch an action in redux that starts the count at 1. This continues to highlight how each navigation triggers a server render first and then a client render second, when you navigate between pages.