Redux Persist example

This example shows how to integrate Redux with the power of Redux Persist in Next.js.

With the advantage of having a global state for your app using `redux`. You'll also require some of your state values to be available offline. There comes `redux-persist` using which you can persist your states in browser's local storage. While there are various ways of persisting your states which you can always find in there documentation. This is an example of how you can integrate `redux-persist` with redux along with Next.js's universal rendering approach.

Deploy your own

Deploy the example using Vercel or preview live with StackBlitz

Deploy

How to use

Execute `create-next-app` with npm or Yarn to bootstrap the example:

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

Deploy it to the cloud with Vercel (Documentation).

Notes

In this example, we are going to use the Next.js example with-redux to see how you can add a layer of persistence for one of the state from global redux state. To know more about how to create a Next.js project with Redux, you can browse the example project with-redux to know more about its implementation.

The Redux Persist has been initialized in `store.js`. You can modify the `redux-persist` configuration (In this example, we are persisting only one state termed `exampleData` which is added in the `persist configuration`) if you need something more with `redux-persist` by following their docs. To wrap out our component in the `Persist Gate` which rehydrates the global state with combining the persisted values and global state values, we'll have to make some modifications in the implementation of Redux in `pages/_app.js`.

The example under `components/data-list.js`, shows a simple component that fetches data after being mounted and then dispatches an action to populate the redux state `exampleData` with the fetched data. And in `store.js`, since we have included the `exampleData` state to be persisted, So once the redux state receives the persisted data from browser's local storage, it will be then updated to the global redux state. So if you open the app next time and there is no Internet connection or whatsoever condition, the app will load the persisted data and will render it on the screen.

For simplicity and readability, Reducers, Actions, Redux Persist configuration, and Store creators are all in the same file: `store.js`