Zustand example

This example shows how to integrate Zustand in Next.js.

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 on how you can use Zustand that also works with Next.js's universal rendering approach.

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 (black) than the client one (grey).

To illustrate SSG and SSR, go to `/ssg` and `/ssr`, those pages are using Next.js data fetching methods to get the date in the server and return it as props to the page, and then the browser will hydrate the store and continue updating the date.

The trick here for supporting universal Zustand 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. That's what we accomplish on `store.js`.

All components have access to the Zustand store using `useStore()` returned from zustand's `createContext()` function.

On the server side every request initializes a new store, because otherwise different user data can be mixed up. On the client side the same store is used, even between page changes.

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-zustand with-zustand-app
# or
yarn create next-app --example with-zustand with-zustand-app
# or
pnpm create next-app -- --example with-zustand with-zustand-app
```

Deploy it to the cloud with Vercel (Documentation).