MobX State Tree with TypeScript 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 on how you can use mobx that also works with our universal rendering approach.

In this example we are going to display a digital clock that updates every second. The first render is happening in the server and the date will be `00:00:00`, then the browser will take over and it will start updating the date.

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 mobx 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`

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

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-mobx-state-tree-typescript with-mobx-state-tree-typescript-app
# or
yarn create next-app --example with-mobx-state-tree-typescript with-mobx-state-tree-typescript-app
# or
pnpm create next-app -- --example with-mobx-state-tree-typescript with-mobx-state-tree-typescript-app
```

Deploy it to the cloud with Vercel (Documentation).