Azi voi vorbi despre utilizarea react si redux la crearea partii front end a aplicatiei, care este si tema mea de teza.

In ultimii ani complexitatea aplicatiilor web a crescut considerabil, developerii fiind necesari sa aiba grija de tot mai multa si mai multa stare. In ajutorul lor au aparut asa frameworkuri si librarii ca Angular, Vue, React iar Librariile ca Redux faciliteaza interactiunea cu starea in continuare

Primul lucru: ce e Redux. Redux e o biblioteca care permite posibilitatea de a centraliza starea unei aplicatii. Aceasta librarie a aparut in 2015 si a fost create de Dan Abramov si Andrew Clark, in scurt timp dupa publicarea conceptului flux de facebook. Au fost destul de multe implementari a acestui concept in scurt timp dup ace a fost publicat, cele mai populare in 2015 fiind repatch si fluxxor. Dar redux-ul a primit popularitate imediat dup ace a fost prezentat la conferinta React din 2015 din cauza ideilor noi asupra fluxului.

Comparand reduxul cu flux, reduxul elimina complexitatea care nu e necesara si este prezenta in flux.

Putem observa ca in redux exista doar un singur store, fara dispatcher, in loc este functie reducer principal, si o stare imutabila, permitiand testarea mai usoara.

Redux te face sa te gandesti la applicate ca o stare initiala fiind modificate de actiuni.

Aceasta librarie e formata din 3 parti

Actiuni:

O actiune e doar un obiect Js ce are un tip care descrie actiunea, apoi toata informatia necesara.

A store holds the whole [state tree](https://redux.js.org/glossary#state) of your application.

The only way to change the state inside it is to dispatch an [action](https://redux.js.org/glossary#action) on it.

A store is not a class. It's just an object with a few methods on it. To create it, pass your root [reducing function](https://redux.js.org/glossary#reducer) to [createStore](https://redux.js.org/api-reference/createstore).

**Reducers** specify how the application's state changes in response to [actions](https://redux.js.org/basics/actions) sent to the store. Remember that actions only describe the fact that something happened, but don't describe how the application's state changes.

It's called a reducer because it's the type of function you would pass to [Array.prototype.reduce(reducer, ?initialValue)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/Reduce). It's very important that the reducer stays pure. Things you should **never** do inside a reducer:

* Mutate its arguments;
* Perform side effects like API calls and routing transitions;
* Call non-pure functions, e.g. Date.now() or Math.random().

We'll explore how to perform side effects in the [advanced walkthrough](https://redux.js.org/advanced). For now, just remember that the reducer must be pure. **Given the same arguments, it should calculate the next state and return it. No surprises. No side effects. No API calls. No mutations. Just a calculation.**

**<Provider store>**

Makes the Redux store available to the connect() calls in the component hierarchy below. Normally, you can’t use connect() without wrapping a parent or ancestor component in <Provider>.

If you *really* need to, you can manually pass store as a prop to every connect()ed component, but we only recommend to do this for stubbing store in unit tests, or in non-fully-React codebases. Normally, you should just use <Provider>