
REACT
CONTEXT

Head of Engineering @ eMoteev

former MTI && ACU @ EPITA

Intro

QUENTIN PRÉ

This is a just a text holder in which you assume that I wrote a lot of interesting things, which obviously I have not.
If you can’t read it easily, you are too far away, get closer to the scene.
If you still have difficulties reading this, raise your hand and ask your question.

RULES

You are encouraged to make mistakes,
You are *forbidden* to make faults.

EFFORT

Do not expect knowledge to fall into your hands.

FOCUS

If you do not intend to follow the lecture, assume consequences for your actions and stay home.

You don’t need your laptop outside of tutorials.

ASK

No question makes you stupid,
Asking no questions though…

note: your question might get answered by another question.

Context

Context

Provider

- A top component _providing_ data in the shape of the context to subcomponents
- All components underneath in the app tree can access the context’s current value

```
const MyContext =
React.createContext(‘blue’);

…

<MyContext.Provider>
  <AChildUsingMyContext />
<MyContext.Provider>
```

Context

Consumer

- **Context.Consumer**: Explicitly exposing the value from a context to children

- *<MyContext.Consumer>*
  { value => {value} }
  </MyContext.Consumer>

Context

Hooks

- A top-level context can be consumed through the *useContext* hook

- ```
  const MyConsumingComponent = () => {
    const value = useContext(MyContext);

    return ( {value}
  }
  ```

- *useContext(MyContext)* will return the value for the nearest *<MyContext.Provider>* in the app tree

- **Warning**: a component that uses context will **always** re-render when the context's value changes
  ```

Context

Consumer (legacy)

- **MyComponent.contextType** can be set on a React class Component to link it to a context

- **this.context** can be used to access the context for this component

- **Caveat:** you can only set one contextType, see MyContext.Consumer to use multiple contexts in one component

```
class MyComponent extends
React.Component {
  contextType = MyContext;

  render() {
    return (
      <div>{this.context.value}</div>
    );
  }
}
```

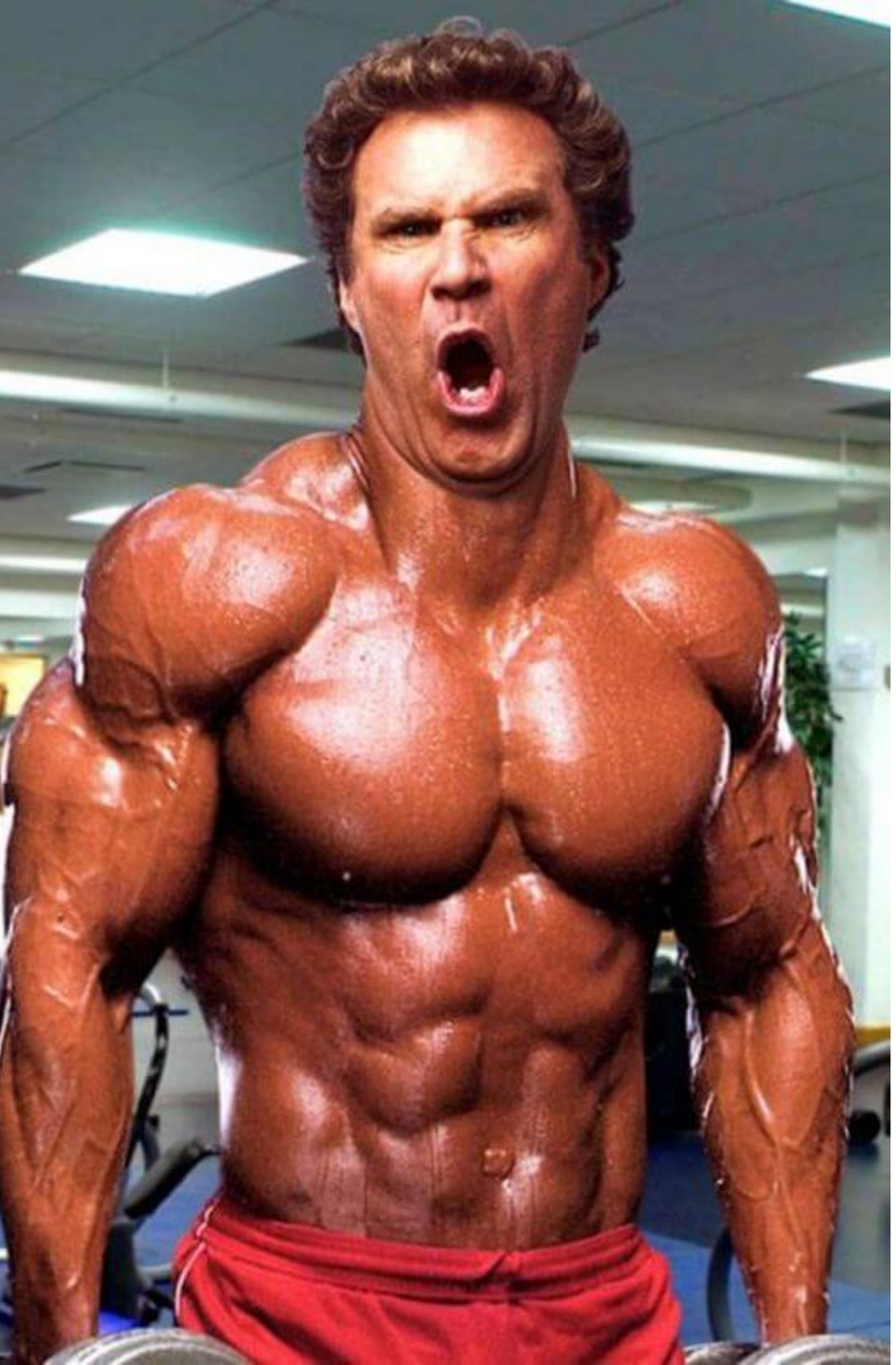
On Steroïds

- **Context.Provider** can be used along React Components to build powerful state handlers

```
const myContext = React.createContext({
  color: '',
  changeColor: () => {}
});

const MyCustomProvider = ({children}) => {
  const [color, setColor] = useState('blue');

  return (
    <myContext.Provider
      value={{ color, changeColor: setColor}}
    >{children}</myContext.Provider>
  );
}
```



On Steroïds

```
const CountContext = React.createContext({
  value: 0,
  dispatch: () => {}
});

const countReducer = (state, action) => {
  switch (action.type) {
    case ‘INCREMENT’:
      return state + 1;
    case ‘DECREMENT’:
      return state - 1;
    default:
      return state;
  }
}
```

```
const MyCustomProvider = ({children}) => {
  const [value, dispatch] = useReducer(countReducer);

  return (
    <myContext.Provider value={{ value, dispatch }>
      {children}
    </myContext.Provider>
  );
}
```

note: have we just achieved *Redux* ?

Reads

React: Rethinking best pratices	Pete Hunt
React Context Documentation	React Team

“Always bet on Javascript”

Brendan Eich