React.js

React 16.3

Leandro.frigerio@devugees.org

React 16.3

What?

The next version of React is alfa (v16.3.0-alpha.1) has been published and available on NPM.

How?

```
yarn add react@next  react-dom@next
```

What are the biggest, most interesting changes?

We'll check them right now!

New in 16.3

- New context API
- New life-cycle methods
- static getDerivedStateFromProps
- StrictMode
- AsyncMode
- New version of React Developer Tools
- Time Slice and Suspense (?)

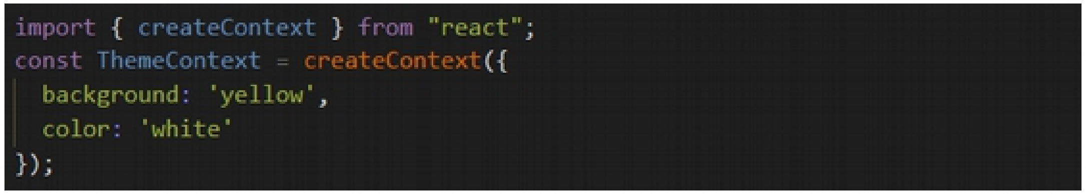
New context API

Context API was always a thing of mystery — it's an official, documented React API but at the same time developers warned us not to use it because the API might change with time, and it was missing some documentation on purpose. Well, that time is now — the RFC phase ("request for comments") has passed and the new API is merged. It is definitely more "user friendly" and might make your life a bit easier when all you want is simple state management without the "overhead" of Redux or MobX.

The new API is accessible as React.createContext() and creates two components for us:

- A **Provider**
- A **Consumer**

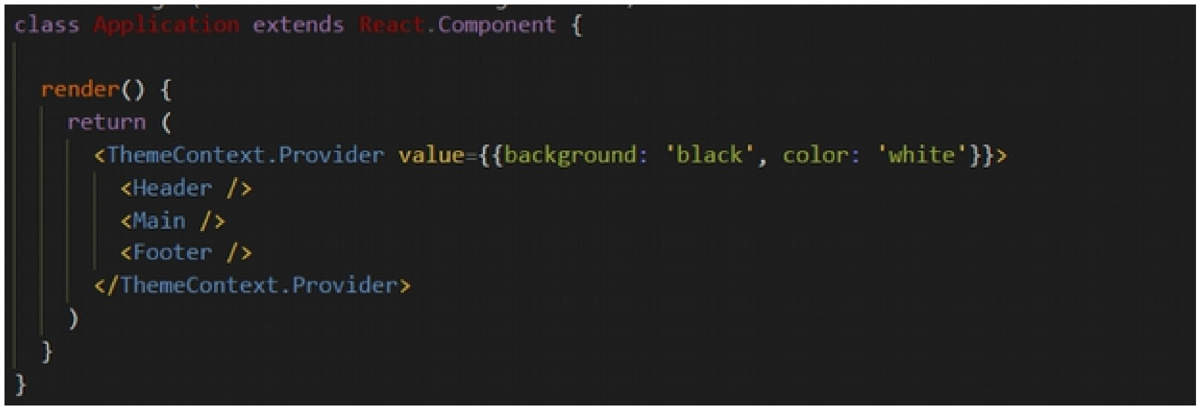
This is how we create a Context via React.createContext



```
import { createContext } from "react";
const ThemeContext = createContext({
  background: 'yellow',
  color: 'white'
});
```

New context API: Provider

The “Provider” is a special component which aims to provide data to all components in its sub-tree, so one example of usage would be:



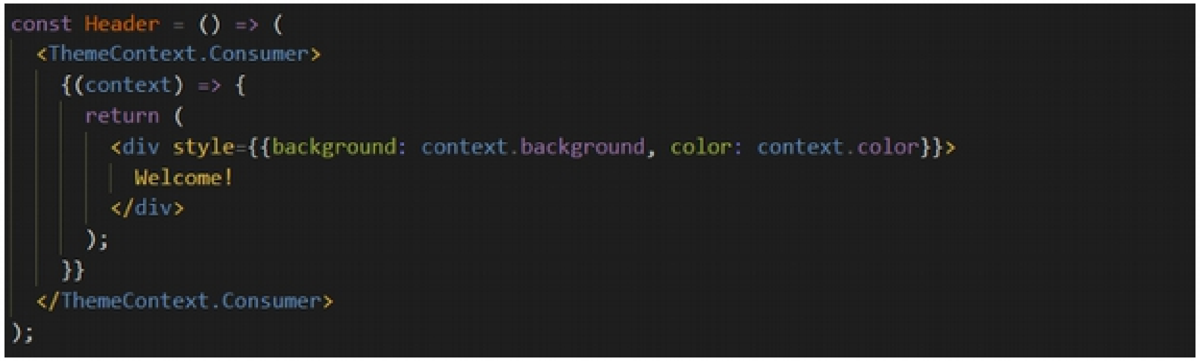
```
class Application extends React.Component {

  render() {
    return (
      <ThemeContext.Provider value={{background: 'black', color: 'white'}}>
        <Header />
        <Main />
        <Footer />
      </ThemeContext.Provider>
    )
  }
}
```

Here we select a sub-tree (in this case, the whole tree) to which we want to pass our “theme” context, and set the value we want to pass. The value can of course be dynamic (e.g. based on this.state).

New context API: Consumer

Next step is to use the Consumer:

Usage of the new context API — Context.Consumer



```
const Header = () => (
  <ThemeContext.Consumer>
    {(context) => {
      return (
        <div style={{background: context.background, color: context.color}}>
          Welcome!
        </div>
      );
    }}
  </ThemeContext.Consumer>
);
```

If you happen to render the “Consumer” without embeding it in a corresponding “Provider”, the default value declared at createContext call will be used.

New context API: Considerations

- the consumer must have access to the same Context component — if you were to create a new context, with the same parameter as input, a new Context would be created and the data would not be passed. For this reason please consider Context a component — it should be created once and then exported + imported whenever needed
- the new syntax uses the function as child (sometime called render prop) pattern — if you're not familiar with this pattern I recommend reading some articles on it
- it is no longer required to use prop-types to specify contextProps if you want to make use of the new Context API
- The data from the context passed to the function matches the value prop set in the providers Context.Provider component, and altering the data in the Provider will cause all consumers to re-render.

New life-cycle methods

Deprecation of some life-cycle methods and introduction of one (four) new. This change aims to enforce best practices which will be very crucial once the asynchronous rendering mode (React 16 “Fiber”).

The functions that will be in time considered deprecation are:

- **componentWillMount** —> use **componentDidMount** instead
- **componentWillUpdate** — > use **componentDidUpdate** instead
- **componentWillReceiveProps** —> a new function, static **getDerivedStateFromProps** is introduced

you are still able to use those functions — the deprecation notices are slotted for 16.4 and removal of them is planned in 17.0

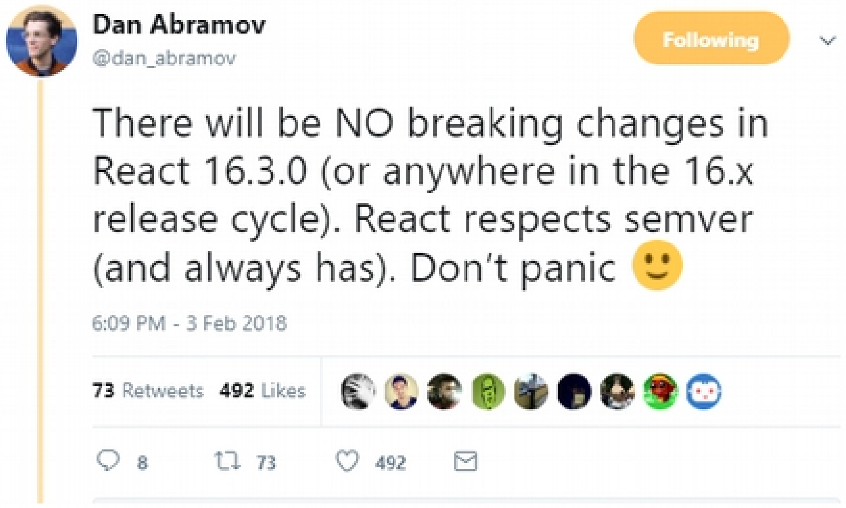
Does it still works?

Yep! You will only see the deprecation notices if you also opt in into the new StrictMode or AsyncMode in which case you can still suppress them by using:

UNSAFE_componentWillMount

UNSAFE_componentWillReceiveProps

UNSAFE_componentWillUpdate



Dan Abramov
@dan_abramov

There will be NO breaking changes in React 16.3.0 (or anywhere in the 16.x release cycle). React respects semver (and always has). Don't panic 🙂

6:09 PM - 3 Feb 2018

73 Retweets 492 Likes

static getDerivedStateFromProps

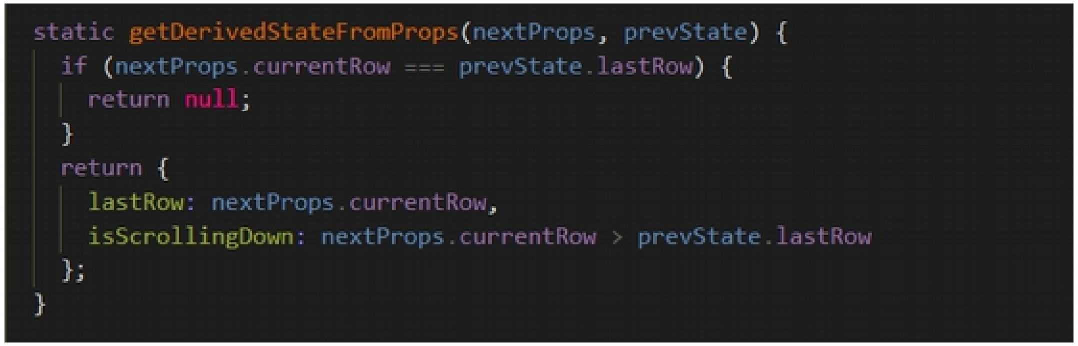
As **componentWillReceiveProps** gets removed, we need some means of updating the state based on props change — the community decided to introduce a new — static — method to handle this.

What’s a static method? A static method is a method / function that exists on the class not its instance. The easiest difference to think about is that static method does not have access to *this* and has the keyword **static** in front of it.

Ok, but if the function has no access to this how are we to call this.setState? The answer is — *we don’t*.
Instead the function should return the updated state data, or null if no update is needed:

Using getDerivedStateFromProps in order to update state

Look at the explaining example:

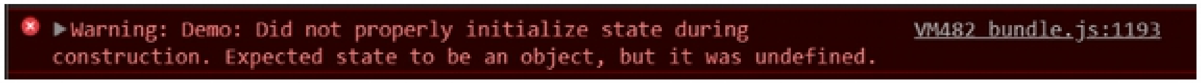


```
static getDerivedStateFromProps(nextProps, prevState) {
  if (nextProps.currentRow === prevState.lastRow) {
    return null;
  }
  return {
    lastRow: nextProps.currentRow,
    isScrollingDown: nextProps.currentRow > prevState.lastRow
  };
}
```

The returned value behaves similarly to current setState value — you only need to return the part of state that changes, all other values will be preserved.

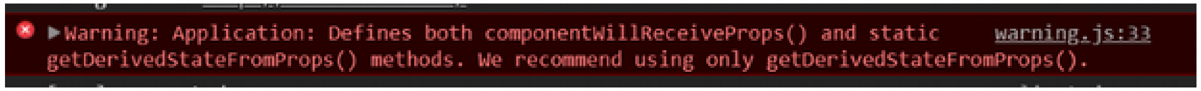
Things to keep in mind

- You still need to declare the initial state of the component (either in constructor or as a class field).



```
▶Warning: Demo: Did not properly initialize state during          VM482 bundle.js:1193
construction. Expected state to be an object, but it was undefined.
```

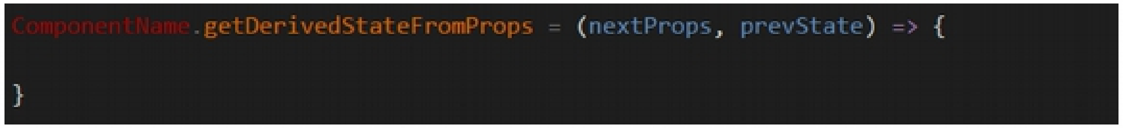
- getDerivedStateFromProps is called both on initial mounting and on re-rendering of the component, so you can use it instead of creating state based on props in constructor.
- If you declare both getDerivedStateFromProps and componentWillReceiveProps only getDerivedStateFromProps will be called, and you will see a warning in the console.



```
▶Warning: Application: Defines both componentWillReceiveProps() and static     warning.js:33
getDerivedStateFromProps() methods. We recommend using only getDerivedStateFromProps().
```

Things to keep in mind (more...)

- Usually, you would use a callback to make sure some code is called when the state was actually updated — in this case, please use componentDidUpdate instead.
- If you prefer not to use the static keyword, you can use the alternative syntax:

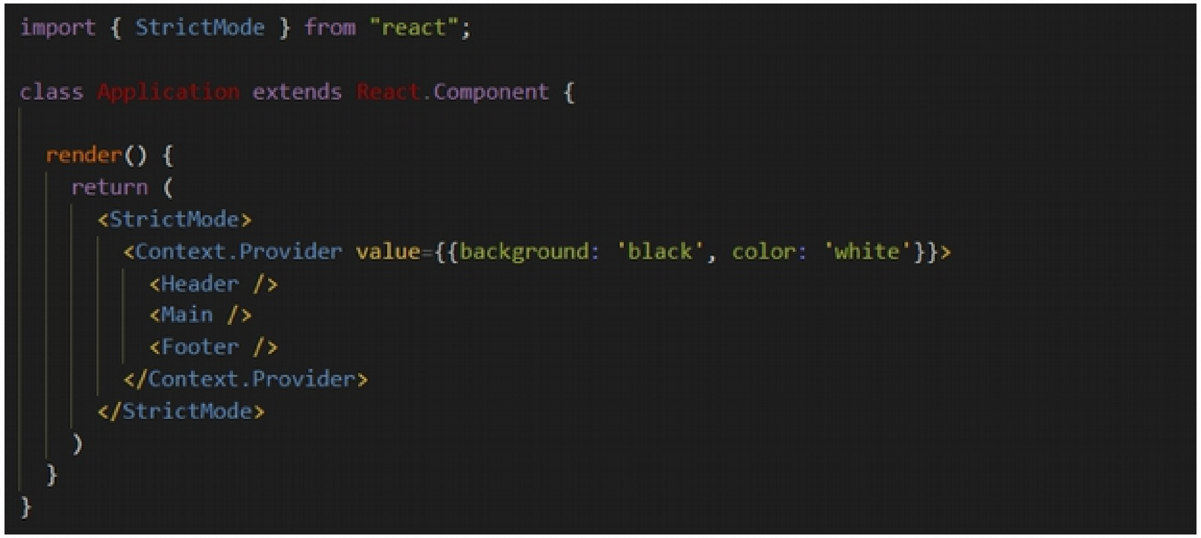


```
ComponentName.getDerivedStateFromProps = (nextProps, prevState) => {

}
```

This is a **declare** getDerivedStateFromProps without using static

StrictMode

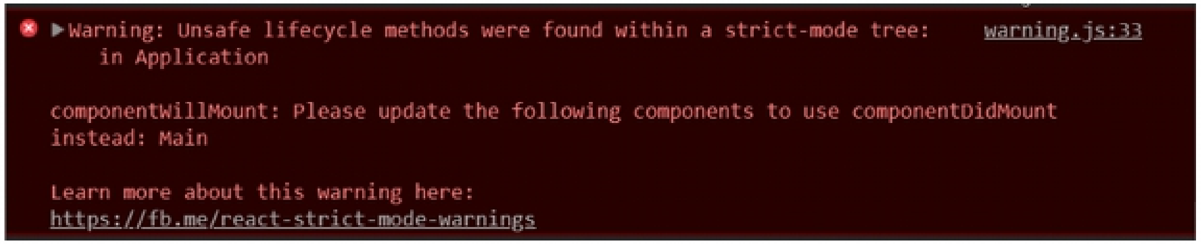
Strict mode is a new way to make sure your code is following the best practices. It’s a component available under **React.StrictMode** and can be added to your application tree or subtree:



```
import { StrictMode } from "react";

class Application extends React.Component {

  render() {
    return (
      <StrictMode>
        <Context.Provider value={{background: 'black', color: 'white'}}>
          <Header />
          <Main />
          <Footer />
        </Context.Provider>
      </StrictMode>
    )
  }
}
```

StrictMode being strict

If one of the children components, rendered in the **StrictMode** subtree uses some of the method mentioned in previous paragraphs (like **componentWillMount**, now deprecated) you will then see an error message in browser console when running an development build:



```
▶Warning: Unsafe lifecycle methods were found within a strict-mode tree:     warning.js:33
    in Application

componentWillMount: Please update the following components to use componentDidMount
instead: Main

Learn more about this warning here:
https://fb.me/react-strict-mode-warnings
```

Currently the error message points to the RFC for the life-cycle methods removal.

AsyncMode

The not-yet-active async Component support was renamed to be aligned with the StrictMode and is now available under React.unsafe_AsyncMode.

Using it will also activate StrictMode warnings.

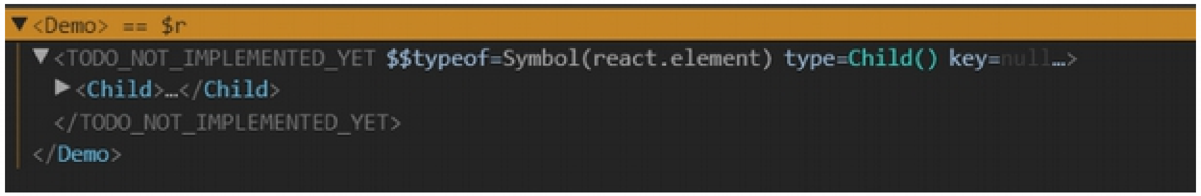
If you want to learn more about asynchronous components you might check some articles / examples at:

https://build-mbfootjxoo.now.sh

https://github.com/koba04/react-fiber-resources

New version of React Developer Tools

A new version of the Developer Tools was released to support debugging the new components.



```
▼<Demo> == $r
  ▼<TODO_NOT_IMPLEMENTED_YET $$typeof=Symbol(react.element) type=Child() key=null…>
    ▶<Child>…</Child>
    </TODO_NOT_IMPLEMENTED_YET>
  </Demo>
```

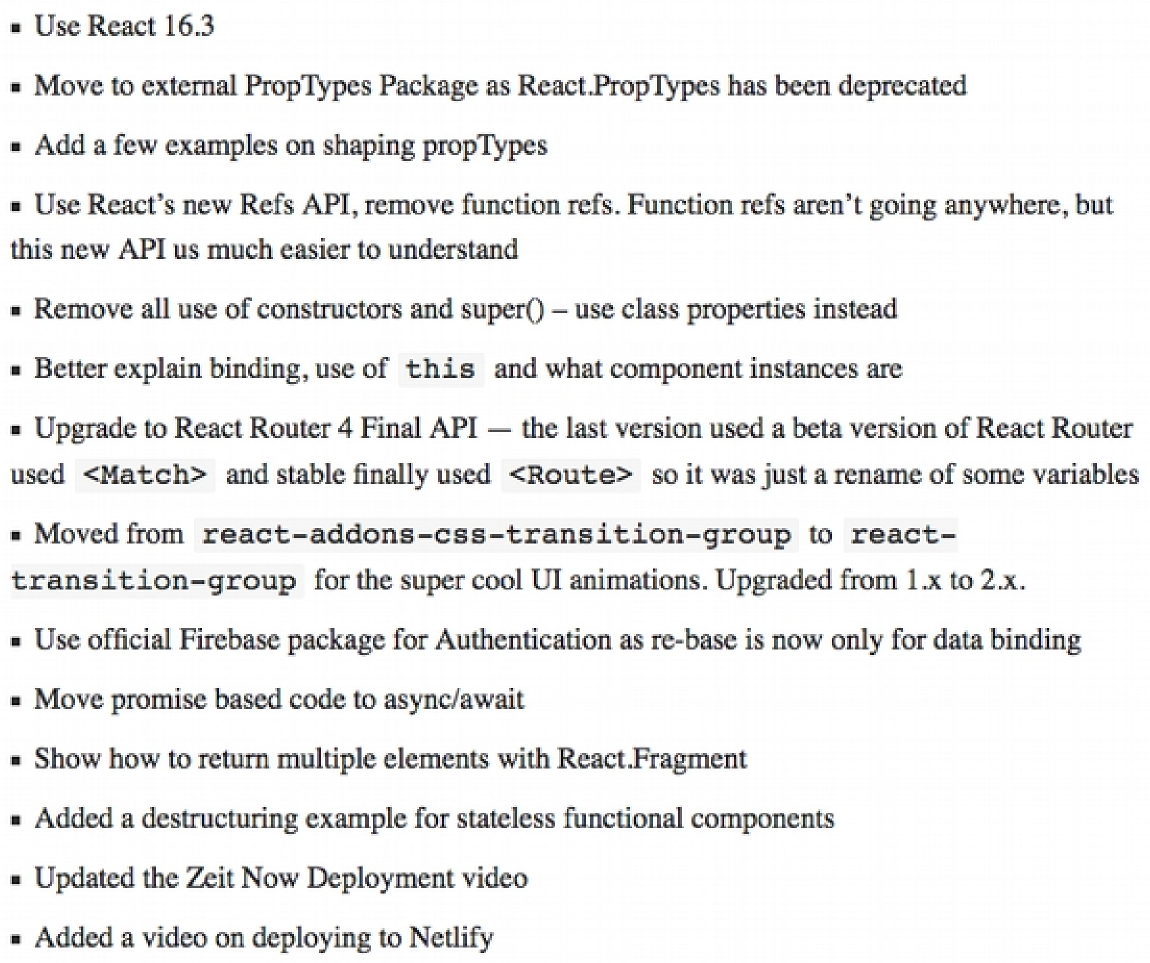
Time Slice and Suspense

At a JavaScript conference in Iceland, JSConf 2018, Dan Abramov from the Facebook team announced two improvements, Time Slice and Suspense coming to React is in the near future.

Originally slated in React 16.3 or the features might arrive later in 2018.

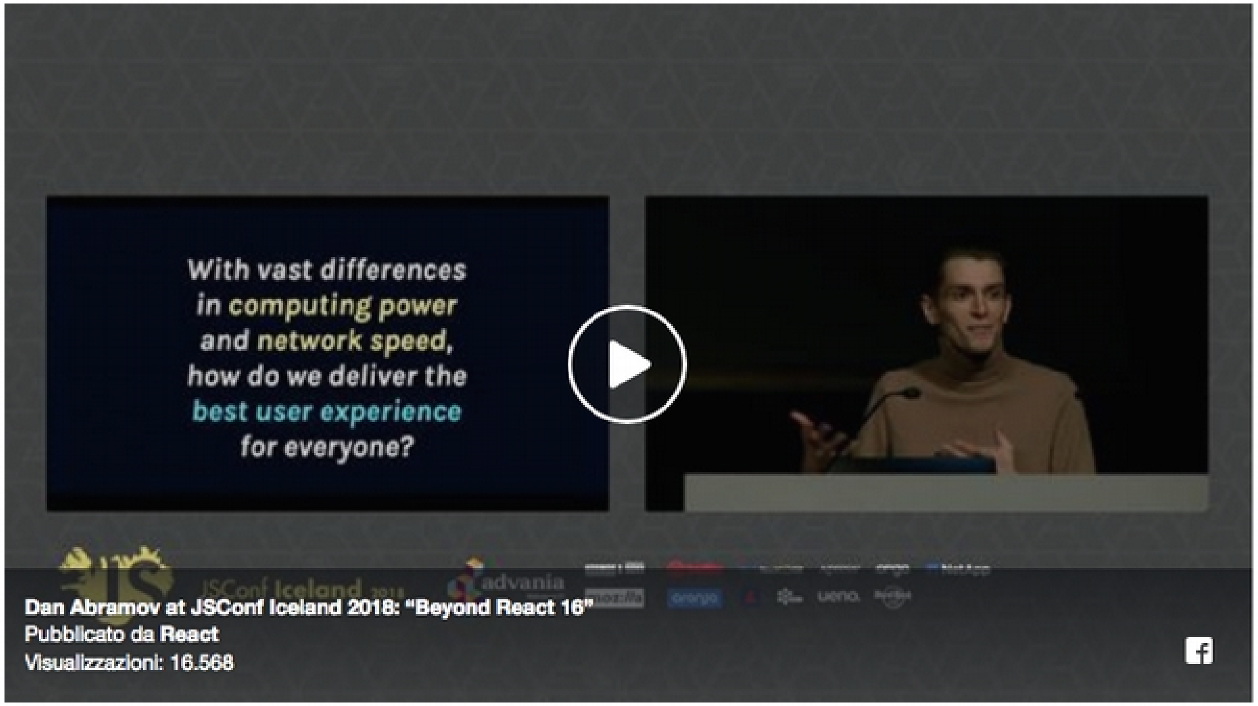
https://react-etc.net/entry/time-slice-and-suspense-features-coming-to-react-js-16-3-or-16-4

Wes Bos upgraded his React course



- Use React 16.3
- Move to external PropTypes Package as React.PropTypes has been deprecated
- Add a few examples on shaping propTypes
- Use React's new Refs API, remove function refs. Function refs aren't going anywhere, but this new API us much easier to understand
- Remove all use of constructors and super() – use class properties instead
- Better explain binding, use of `this` and what component instances are
- Upgrade to React Router 4 Final API — the last version used a beta version of React Router used `<Match>` and stable finally used `<Route>` so it was just a rename of some variables
- Moved from `react-addons-css-transition-group` to `react-transition-group` for the super cool UI animations. Upgraded from 1.x to 2.x.
- Use official Firebase package for Authentication as re-base is now only for data binding
- Move promise based code to async/await
- Show how to return multiple elements with React.Fragment
- Added a destructuring example for stateless functional components
- Updated the Zeit Now Deployment video
- Added a video on deploying to Netlify

Sneak Peek: Beyond React 16



Dan Abramov at JSConf Iceland 2018: "Beyond React 16"
Pubblicato da React
Visualizzazioni: 16.568

Blog Post
https://reactjs.org/blog/2018/03/01/sneak-peek-beyond-react-16.html

Video
https://www.youtube.com/watch?time_continue=206&v=v6iR3Zk4oDY

References

- What’s new in React 16.3(.0-alpha)
  https://medium.com/@baphemot/whats-new-in-react-16-3-d2c9b7b6193b
- Wes Bos: React for Beginners Re-Recorded (again!)
  http://wesbos.com/react-for-beginners-re-recorded-again/
-