Tuesday 05.12.2017 / **ISTANBUL**

React 101 &
An Introduction to Life Cycle Events in React

Vedat Mahir Yılmaz | **m:** mahir@vedatmahir.com

whoami?

V. Mahir Yılmaz

Solution Engineer @ Codespace / Akce Group

- React Components
- Lifecycle Methods


cube drone
GUYS! GUYS! STOP EVERYTHING!
24:18:37
JAVASCRIPT ORTHODOXY HAS CHANGED SLIGHTLY! WE HAVE TO MOVE TO A NEW FRAMEWORK.
AW NO.
I JUST LEARNED BACKBONE
AGAIN?
DAMN.
I KNOW, I KNOW, IT SUCKS TO START OVER.
00:00:00
BOOP

- A library for creating user interfaces
- View in MVC
- Created by Facebook
- Component Based UI Library **Not Framework**
- Virtual DOM

- Simple to learn
- Composable Components
- Easy to use with existing projects
- JSX

JSX is a preprocessor step that adds XML syntax to JavaScript.

```
import React, { Component } from 'react';

class CmpTest extends Component {
  render () {
    return (
      <div className="events">
        <h1>Latest Frontend Istanbul Events</h1>
        <ul>
          <li>Modern Web Uygulamalarında Redux</li>
          <li>Modern Frontend Mimarisi ve Progressive Web Uygulamaları</li>
          <li>Nuxt.js ile SSR & React ile Web Uygulamaları</li>
          <li>beer.js</li>
        </ul>
      </div>
    );
  }
}
```

Components are the heart and soul of React.

```
import React, { PropTypes } from 'react'

// Create a component named MessageComponent
const MessageComponent = React.createClass({
  render () {
    return (
      <div>{this.props.message}</div>
    )
  }
})

// Render an instance of MessageComponent into document.body
ReactDOM.render(
  <MessageComponent message="Hello!" />,
  document.body
);
```

A function is pure if...

- Given the same input, will always return the same output.
- Produces no side effects.

```
const HelloWorld = ({name}) => {
  return (
    <h1>{name}</h1>
  );
}

ReactDOM.render(<HelloWorld text="Mahir" />, document.body);
```

- If all you have is render, stay **functional**
- An abstract base class meant to be extended
- Don’t screw with props, they're Read Only

```
class HelloWorld extends Component {
  render () {
    const { name } = this.props;
    return (
      <h1>{name}</h1>
    );
  }
}
```

Re-render everything on every change

- Create a lightweight description of Component UI.
- Diff it with the old one.
- Compute minimal set of changes to apply the DOM
- Batch execute all updates

Props

- Data should always have a clear unidirectional flow. In react this is implemented with `**props**`.
- Props, or properties, are always passed from the outside into components.
- Props cannot be modified.
- If a parent re-renders and passes a different value for the prop, the component will be re-rendered as well.

State

- If you need to modify a value inside a component, you should first evaluate why.
- If that’s not possible, a React Component can hold it’s own state.
- If you mutate the state, the component will be re-rendered.

```
import React, { PropTypes } from 'react'

class CounterDemo extends React.Component {
  constructor(){
    this.state = {
      counter: 0
    }
  }
  incrementCounter(){
    this.setState({
      counter: this.state.counter + 1
    });
  }
  render () {
    <div>
      You hit the button {this.state.counter} times!
      <button onClick={this.incrementCounter}>Hit!</button>
    </div>
  }
}

export default CounterDemo;
```

```
class MessageComponent extends Component {
  render () {
    return (
      <div>
        {this.props.message}
        <span className="date">{this.props.time}</span>
      </div>
    );
  }
}

class ChatboxComponent extends Component {
  render () {
    return (
      <div>
        <MessageComponent message="React cool!" time="2017-12-04" />
        <MessageComponent message="Frontend Istanbul Rockz!" time="2017-12-05" />
      </div>
    );
  }
}
```

ReactJS Component’s Lifecycle

- **Mounting**
- **Updating**
- **Unmounting**

- constructor()
- componentWillMount()
- render()
- componentDidMount()

- componentWillReceiveProps()
- shouldComponentUpdate()
- componentWillUpdate()
- render()
- componentDidUpdate()

- componentWillUnmount()

- Perform any initial setup
- Called once per mounted component
- Initialize state
- Must call super(props)

```
constructor(props) {
  super(props);
  this.state = {

  };
}
```

- props and state both ready
- Can use for calling setState
- Most times just use constructor

- initialize jQuery plugins
- DOM is ready here
- Stand up plugins
- ref is now a function
- Dispatch actions

```
import Swiper from "swiper";

export default class Slider extends React.Component {
  constructor(props) {
    super(props);
    this.state = { /* init state */ };
  }
  componentDidMount() {
    this.slider = new Swiper(this.gallery, {
      speed: 400,
      spaceBetween: 100
    });
  }
  render() {
    return (
      <div
        ref={(gallery) => this.gallery = gallery}
      />
        {
          // slider-items
        }
      </div>
    )
  }
}
```

- Remove any event handlers or plugins
- No leaks

```
componentWillUnmount() {
  this.slider.destroy();
}
```

- **props** have changed
- NOT called on initial render
- Update state based on props
- Dispatch actions
- **Be careful, can cause loops**
- Update state if props don't match

```
class AvatarUploader extends Component {
  constructor(props) {
    super(props);
    this.state = { src: props.src };
  }
  componentWillReceiveProps(nextProps) {
    if (nextProps.src !== this.props.src) {
      this.setState({
        src: nextProps.src,
      });
    }
  }
  uploadFiles() {
    this.props.upload()
  }
  render() {
    /* ... */
  }
}
```

- Very useful in React Router SPA

```
componentWillReceiveProps(nextProps) {
  const { id: currentId } = this.props.params;
  const { id: nextId } = nextProps.params;
  if (currentId !== nextId) {
    this.props.fetchData(nextId);
  }
}
```

- Tell the component whether or not to render
- Can increase performance
- Used to call shallowCompare

```
shouldComponentUpdate(this, nextProps, nextState) {
  return shallowCompare(this, nextProps, nextState);
}
```

- Called BEFORE render like when setState called
- Do NOT call setState here

- Gives you previous props prevProps to compare before actions
- Called AFTER render
- Update the DOM

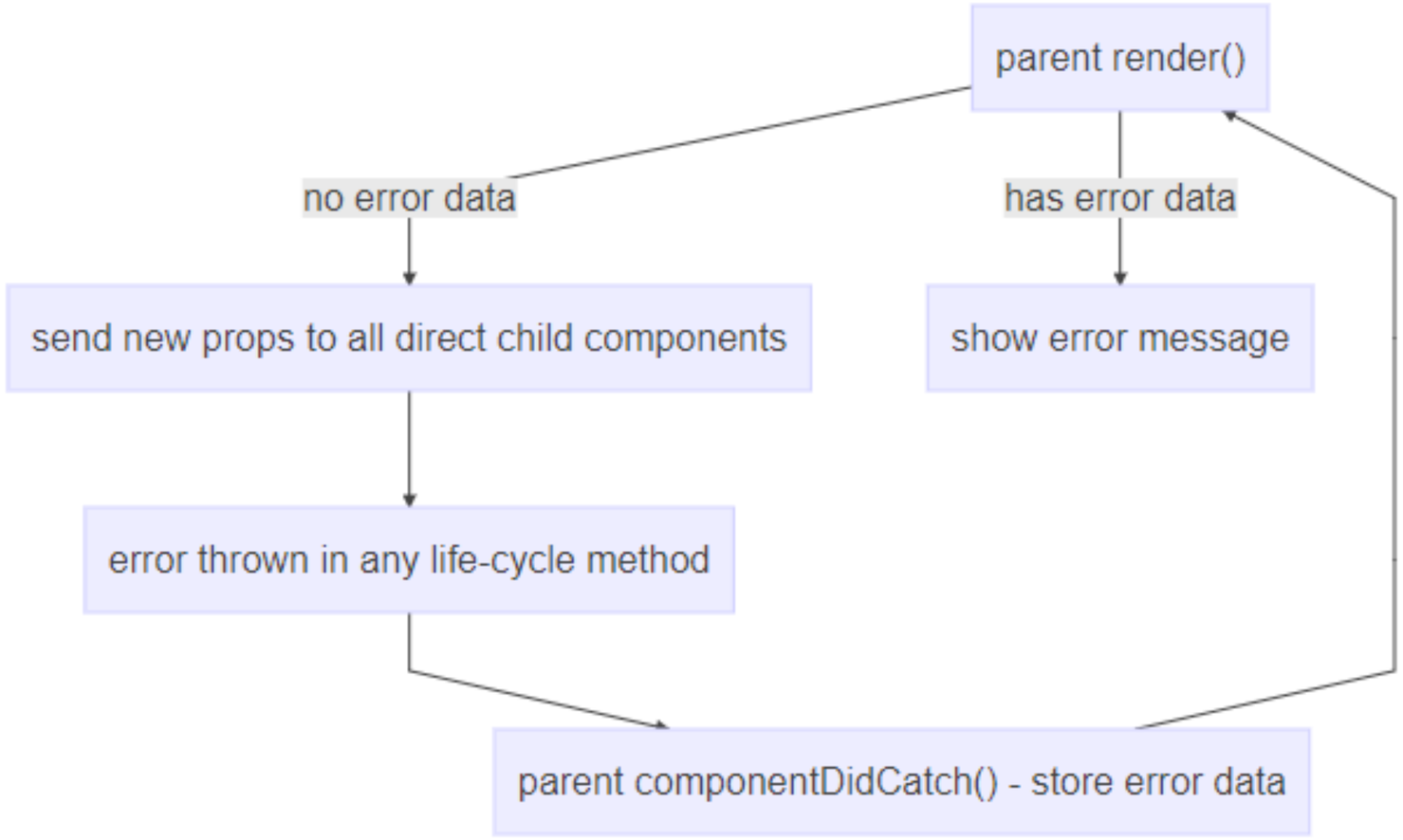
```
componentDidUpdate(prevProps) {
  if(prevProps.myProps !== this.props.myProp) {
    // this.props.myProp has a different value
    // we can perform any operations that would
    // need the new value and/or cause side-effects
    // like AJAX calls with the new value - this.props.myProp
  }
}
```

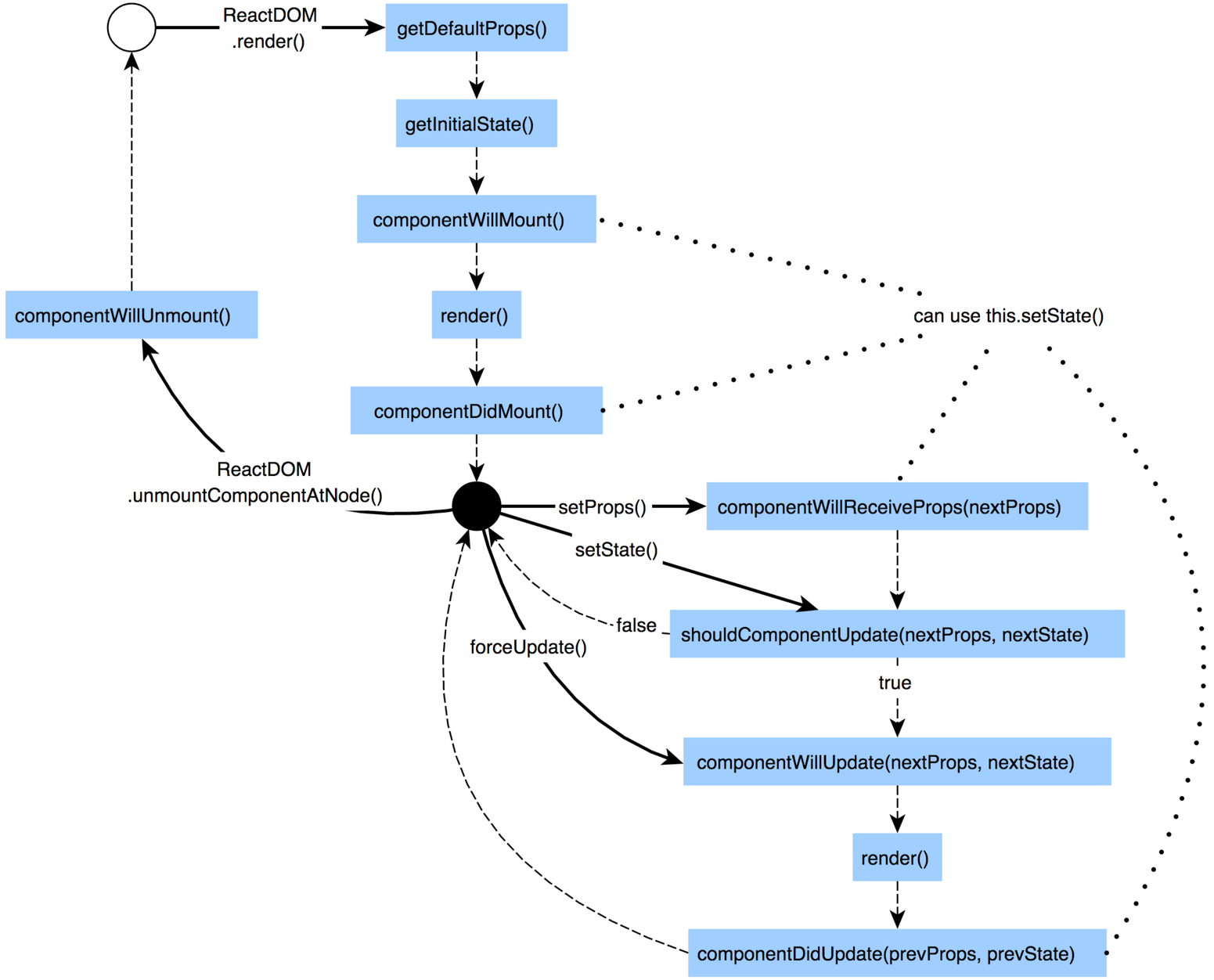
```
export default class PoiDetail extends React.PureComponent {
  constructor(props) {
    // ...
  }
  hasPoiUpdated() {}
  componentDidMount() {
    this.subscription = postal.subscribe({ /* ... */ })
  }
  componentWillReceiveProps(nextProps) {
    if (nextProps.params.id !== this.props.params.id) {
      this.props.fetchPoi(nextProps.params.id);
    }
  }
  shouldComponentUpdate(nextProps) {
    return nextProps.poi.id !== this.props.poi.id;
  }
  componentWillUpdate(nextProps, nextState) { /* ... */ }
  componentDidUpdate(prevProps) {
    const isNewPoi = prevProps.poi.id !== this.props.poi.id;
    if (isNewPoi) {
      window.scrollTop = 0;
    }
  }
  componentWillUnmount() {
    this.subscription.unsubscribe();
  }
  render() { /* ... */ }
}
```

ReactJS Component's Lifecycle

Recap

Name of Thing	Like ?	Mounted ?	Do Here?
constructor	initialize	no	init stuff NO side effects
componentWillMount	beforeDomReady	no	Only needed in createClass now use constructor for most things
render	render	no	render stuff and don't set any state please
componentDidMount	domReady	yes	DOM is a go init jQuery plugins dispatch stuf
componentWillReceiveProps	onChange	yes	Props changed feel free to update state if needed
componentWillUpdate	beforeRender	yes	The props or state changed need to do anything else before rendering?
shouldComponentUpdate	shouldRender	yes	So yeah something changed but do we REALLY need to update?
componentDidUpdate	afterRender	yes	Great success we've rendered a thing... anything else?
componentWillUnmount	destroy	:(	Only you can prevent memory leaks aka unbind things


parent render()
no error data
has error data
send new props to all direct child components
show error message
error thrown in any life-cycle method
parent componentDidCatch() - store error data


ReactDOM
.render()
getDefaultProps()
getInitialState()
componentWillMount()
render()
componentDidMount()
componentWillUnmount()
ReactDOM
.unmountComponentAtNode()
can use this.setState()
setProps()
componentWillReceiveProps(nextProps)
setState()
false
shouldComponentUpdate(nextProps, nextState)
forceUpdate()
true
componentWillUpdate(nextProps, nextState)
render()
componentDidUpdate(prevProps, prevState)

Thanks to

- Jonathan Creamer - Lonely Planet