Reactive
programming

in JavaScript with Reactjs

Forget about…

Established truths

Everything you thought you knew
about making web apps

Relax

It’s going to be okay

The Problem

How can we build large apps with data that **changes** over time?

But: local state that changes over time is the **root of all evil**

ModelViewController

The MVC pattern was developed in 1979

It was deviced as a **general solution** to the problem of users controlling a large and complex data set.

It's not 1979 anymore...

The MVC problem

Thin views / templates

Models and controllers that grows...

...and grows

until most of your time is spent
keeping them in sync

We need a better model

React

A JavaScript **library** for building composable user interfaces

React gives you

A lightweight **virtual DOM**

Powerful **views** without templates

Unidirectional **data flow**

Explicit **mutation**

A React app consists of

Reusable **components**

Components makes **code reuse, testing,**
and **separation of concerns** easy.

Not just the V

In the beginning, React was presented as the V in MVC.

This is at best a huge simplification.

React has state, it handles mapping from input to state changes, and it renders components. In this sense, it does everything that an MVC does.

games →

'A golden shining moment': the true story behind Atari's ET, the worst video game ever

16 comments

Angry Birds set sights on Candy Crush with new mobile puzzle games

1 comment

When will gamers understand that criticism isn't censorship?

Painting by numbers: getting creative with environmental data

0 comments

Review / Saints Row IV: Re-Elected And Gat Out Of Hell review

Xbox One, Xbox 360, PS3, PS4, PC; Deep Silver; £29.97-£43.99

3 comments

+ More games

<NewsFeed>

games →

'A golden shining moment': the true story behind Atari's ET, the worst video game ever

16 comments

Angry Birds set sights on Candy Crush with new mobile puzzle games

1 comment

When will gamers understand that criticism isn't censorship?

Painting by numbers: getting creative with environmental data

0 comments

+ More games

Review / Saints Row IV: Re-Elected And Gat Out Of Hell review

Xbox One, Xbox 360, PS3, PS4, PC; Deep Silver; £29.97-£43.99

3 comments

games →

'A golden shining moment': the true story behind Atari's ET, the worst video game ever

16 comments

Angry Birds set sights on Candy Crush with new mobile puzzle games

1 comment

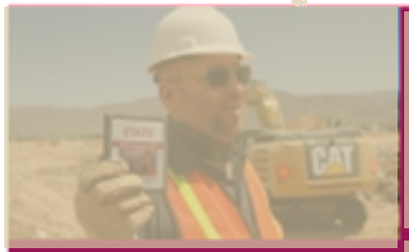
When will gamers understand that criticism isn't censorship?

Painting by numbers: getting creative with environmental data

0 comments

+ More games

<ItemCover>



'A golden shining moment': the true story behind Atari's ET, the worst video game ever

16 comments

<ItemTitle>

<ItemComments>

Angry Birds set sights on Candy Crush with new mobile puzzle games

1 comment

When will gamers understand that criticism isn't censorship?

Painting by numbers: getting creative with environmental data

0 comments

+ More games

NewsItem.jsx

```
var React = require("react");

var ItemCover = React.createClass({
    render:function(){
        return(
            <figure className="news-cover"> [...] </figure>
        )
    }
});
```

```
var NewsItem = React.createClass({
    render:function (){
        return(
            <article className="news-item">
                <ItemCover />
                <div className="news-title"> [...] </div>
                <div className="news-link"> [...] </div>
            </article>
        )
    }
});
```

NewsItem.jsx

```
var React = require("react");

var ItemCover = React.createClass({
    render:function(){
        return(
            <figure className="news-cover"> [...] </figure>
        )
    }
});

var NewsItem = React.createClass({
    render:function (){
        return(
            <article className="news-item">
                <ItemCover />
                <div className="news-title"> [...] </div>
                <div className="news-link"> [...] </div>
            </article>
        )
    }
});
```

NewsItem.jsx

```
var React = require("react");

var ItemCover = React.createClass({
    render:function(){
        return(
            <figure className="news-cover"> [...] </figure>
        )
    }
});
```

```
var NewsItem = React.createClass({
    render:function (){
        return(
            <article className="news-item">
                <ItemCover />
                <div className="news-title"> [...] </div>
                <div className="news-link"> [...] </div>
            </article>
        )
    }
});
```

NewsItem.jsx

```
var React = require("react");

var ItemCover = React.createClass({
    render:function(){
        return(
            <figure className="news-cover"> [...] </figure>
        )
    }
});

var NewsItem = React.createClass({
    render:function (){
        return(
            <article className="news-item">
                <ItemCover />
                <div className="news-title"> [...] </div>
                <div className="news-link"> [...] </div>
            </article>
        )
    }
});
```

JSX

A JavaScript XML based extension that makes it **easy** to mix HTML with JavaScript

“

We strongly believe that components are the right way to separate concerns rather than *"templates"* and *"display logic."*

We think that **markup** and the **code that generates it** are **intimately tied together.**

facebook

Component Life Cycle

Initial render

Get Initial State

Set initial value of this.state

Get Default Props

Set initial value of this.props

Component Will Mount

Calling setState here does not cause a re-render

Render

Return JSX for component
Never update state here

Component Did Mount

Called immediately after render

Component Life Cycle

PROPS Change

NOT called for initial render

Component will receive props

- Takes nextprops as input
- Previous props available as this.props
- Calling setState() here does not trigger re-render

Should component update

Can abort render if you return false here. If false, componentWillUpdate and componentDidUpdate will not be called.

Component will update

nextProps, nextState available here
Cannot use setState() here

Render

Component did update

Called immediately after render

Component Life Cycle

STATE Change

NOT called for initial render

Should component update

Can abort render if you return false here. If false, componentWillUpdate and componentDidUpdate will not be called.

Component will update

nextProps, nextState available here
Cannot use setState() here

Render

Component did update

Called immediately after render

Component Life Cycle

Statics

The statics object allows you to define static methods
that can be invoked on the component without creating instances

```
var Component = React.createClass({
    statics: {
        componentName: 'My Static Component'
    },
    render: function() {
        return <span>Hello World</span>
    }
});

console.log(Component.componentName);  // My Static Component
```

These methods **do not** have access to the component's props or state

Component Life Cycle

Unmount

Component will unmount

Invoked immediately before component is unmounted. For cleanup, invalidating timers etc.

Virtual DOM


Render
the DOM
Build a new
Virtual DOM
Diff with
old DOM
EACH UPDATE
Compute the minimal
sets of mutation and queue
Batch execute
all updates

State

For interactivity
in the component.
Mutable data

Props

For data passed
to the component
Should be treated as
immutable.

State

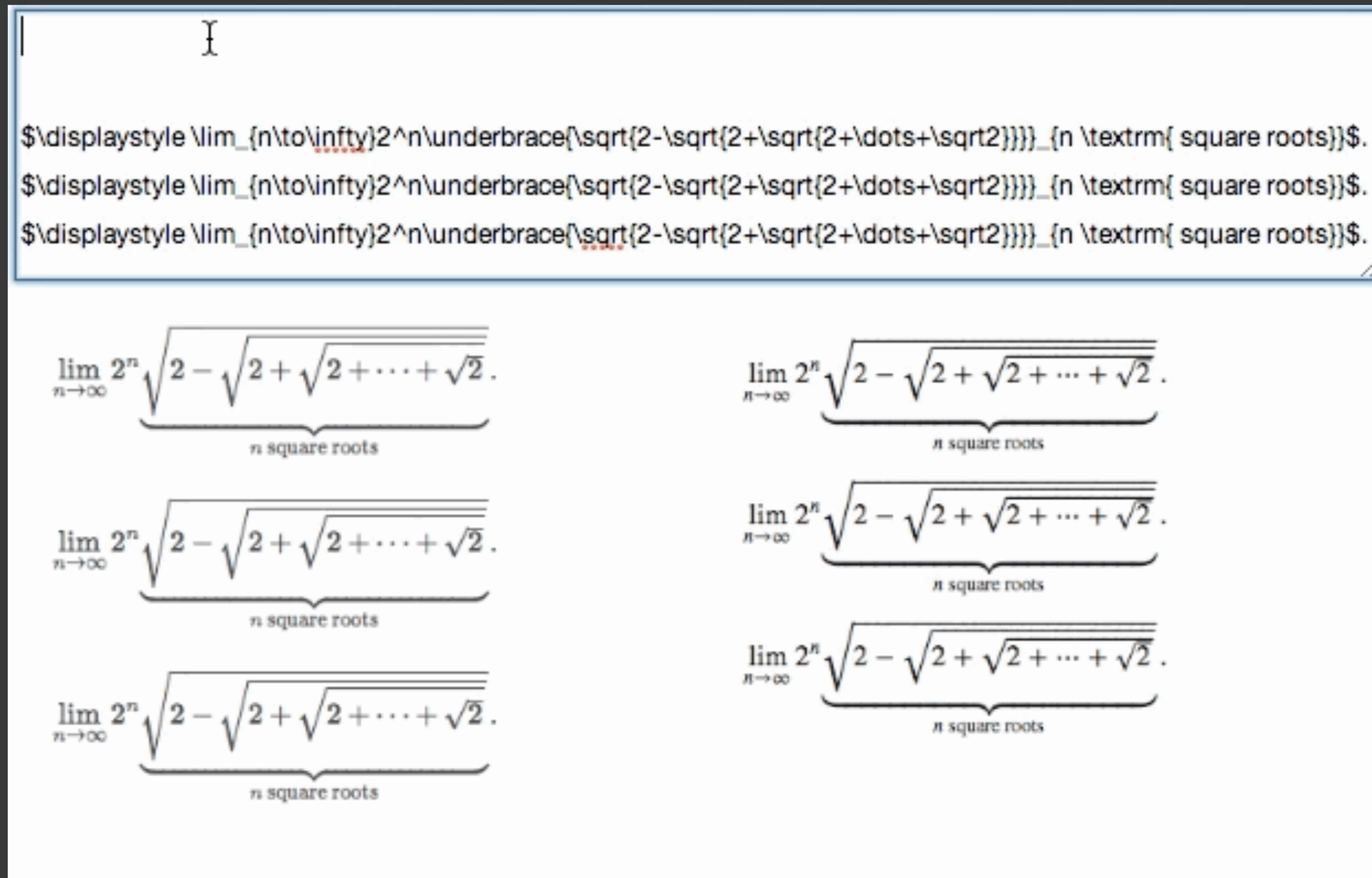
Is updated by calling setState()

Every call to setState() triggers a re-render

(except when called within componentDidMount)

React

jQuery



Only the changes are rendered

Everything is re-rendered

Server Rendering

Traditional JavaScript applications are hard to render on the server. This makes the app uncrawlable, and you miss out on SEO.

Server Rendering

Fortunately, React can handle this with ease.

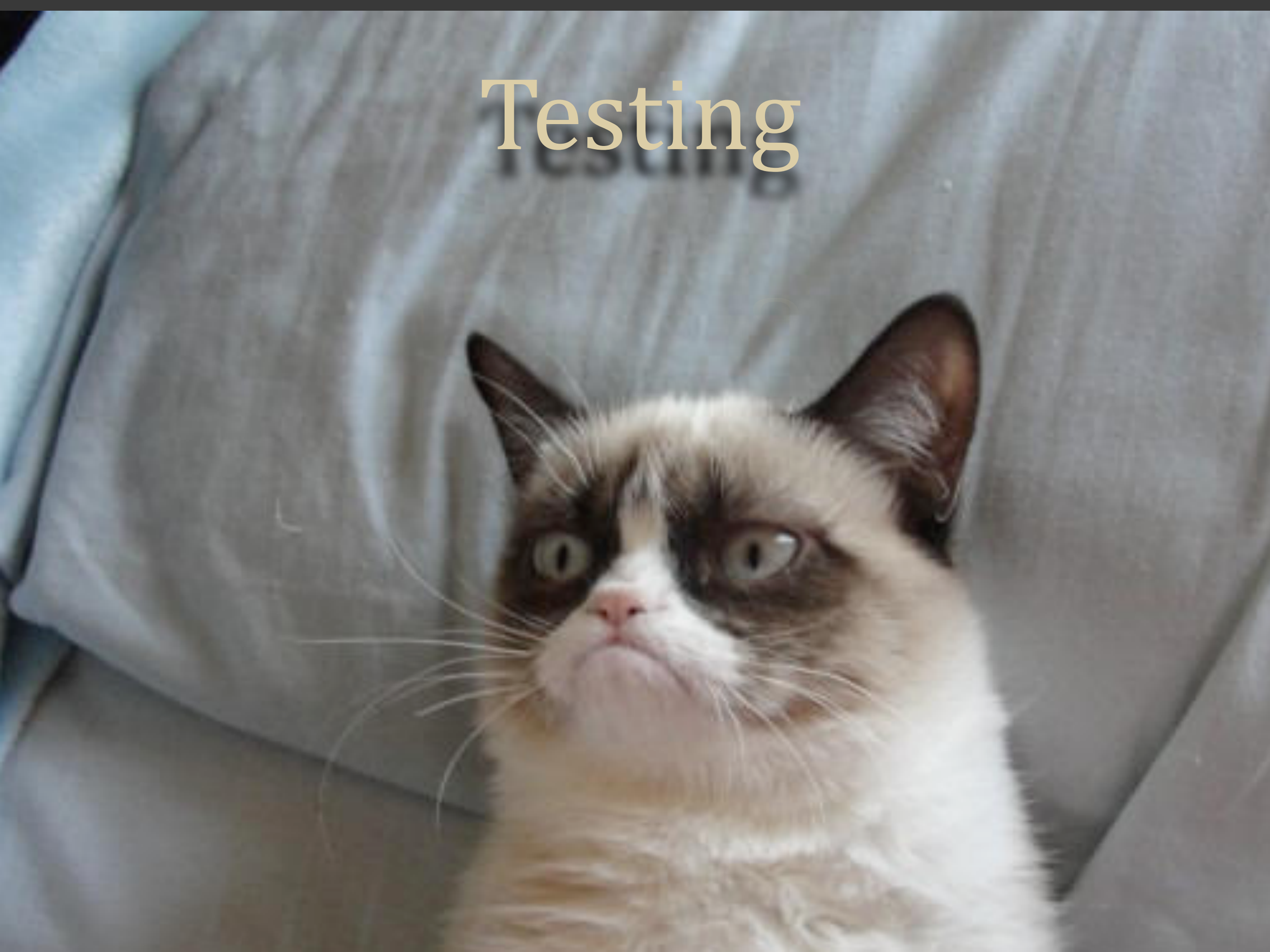
All you need to do is call **renderToString** instead of **render** and you've got a SEO ready component.

Server Rendering

Another option is to call
renderToStaticMarkup.

This is similar to renderToString,
except this doesn't create extra DOM
attributes such as data-react-id which
is useful if you want to use React as a
simple **static page generator**.

Testing



JEST

Built on top of the Jasmine test framework, using familiar expect(value).toBe(other) assertions

JEST

Automatically finds tests to execute in your repo

JEST

Automatically mocks dependencies
for you when running your tests

JEST

Allows you to test asynchronous code synchronously

JEST

Runs your tests with a fake DOM implementation (via jsdom) so that your tests can run on the command line

JEST

In short, if you want to test React code, use JEST.

Practical example

Unclicked State

Clicked State

```
jest.dontMock('../public/src/scripts/button/index.js');

describe('ClickButton', function() {
    it('changes state when user clicks a button', function() {
        var React = require('react/addons');

        var Component = React.createFactory(require('../public/src/scripts/button/index.js'));
        var TestUtils = React.addons.TestUtils;
        var instance = TestUtils.renderIntoDocument(Component);

        var button = TestUtils.findRenderedDOMComponentWithClass(instance, 'button');

        TestUtils.Simulate.click(button);
        buttonText = TestUtils.findRenderedDOMComponentWithClass(instance, 'buttonStatus');
        expect(buttonText.getDOMNode().textContent).toBe('Clicked me');
    });
});
```

```
jest.dontMock('../public/src/scripts/button/index.js');

describe('ClickButton', function() {
    it('changes state when user clicks a button', function() {
        var React = require('react/addons');

        var Component = React.createFactory(require('../public/src/scripts/button/index.js'));
        var TestUtils = React.addons.TestUtils;
        var instance = TestUtils.renderIntoDocument(Component);

        var button = TestUtils.findRenderedDOMComponentWithClass(instance, 'button');

        TestUtils.Simulate.click(button);
        buttonText = TestUtils.findRenderedDOMComponentWithClass(instance, 'buttonStatus');
        expect(buttonText.getDOMNode().textContent).toBe('Clicked me');
    });
});
```

```
jest.dontMock('../public/src/scripts/button/index.js');

describe('ClickButton', function() {
    it('changes state when user clicks a button', function() {
        var React = require('react/addons');

        var Component = React.createFactory(require('../public/src/scripts/button/index.js'));
        var TestUtils = React.addons.TestUtils;
        var instance = TestUtils.renderIntoDocument(Component);

        var button = TestUtils.findRenderedDOMComponentWithClass(instance, 'button');

        TestUtils.Simulate.click(button);
        buttonText = TestUtils.findRenderedDOMComponentWithClass(instance, 'buttonStatus');
        expect(buttonText.getDOMNode().textContent).toBe('Clicked me');
    });
});
```

```
jest.dontMock('../public/src/scripts/button/index.js');

describe('ClickButton', function() {
    it('changes state when user clicks a button', function() {
        var React = require('react/addons');

        var Component = React.createFactory(require('../public/src/scripts/button/index.js'));
        var TestUtils = React.addons.TestUtils;
        var instance = TestUtils.renderIntoDocument(Component);

        var button = TestUtils.findRenderedDOMComponentWithClass(instance, 'button');

        TestUtils.Simulate.click(button);
        buttonText = TestUtils.findRenderedDOMComponentWithClass(instance, 'buttonStatus');
        expect(buttonText.getDOMNode().textContent).toBe('Clicked me');
    });
});
```

```
jest.dontMock('../public/src/scripts/button/index.js');

describe('ClickButton', function() {
    it('changes state when user clicks a button', function() {
        var React = require('react/addons');

        var Component = React.createFactory(require('../public/src/scripts/button/index.js'));
        var TestUtils = React.addons.TestUtils;
        var instance = TestUtils.renderIntoDocument(Component);

        var button = TestUtils.findRenderedDOMComponentWithClass(instance, 'button');

        TestUtils.Simulate.click(button);
        buttonText = TestUtils.findRenderedDOMComponentWithClass(instance, 'buttonStatus');
        expect(buttonText.getDOMNode().textContent).toBe('Clicked me');
    });
});
```

```
jest.dontMock('../public/src/scripts/button/index.js');

describe('ClickButton', function() {
    it('changes state when user clicks a button', function() {
        var React = require('react/addons');

        var Component = React.createFactory(require('../public/src/scripts/button/index.js'));
        var TestUtils = React.addons.TestUtils;
        var instance = TestUtils.renderIntoDocument(Component);

        var button = TestUtils.findRenderedDOMComponentWithClass(instance, 'button');

        TestUtils.Simulate.click(button);
        buttonText = TestUtils.findRenderedDOMComponentWithClass(instance, 'buttonStatus');
        expect(buttonText.getDOMNode().textContent).toBe('Clicked me');
    });
});
```

```
jest.dontMock('../public/src/scripts/button/index.js');

describe('ClickButton', function() {
    it('changes state when user clicks a button', function() {
        var React = require('react/addons');

        var Component = React.createFactory(require('../public/src/scripts/button/index.js'));
        var TestUtils = React.addons.TestUtils;
        var instance = TestUtils.renderIntoDocument(Component);

        var button = TestUtils.findRenderedDOMComponentWithClass(instance, 'button');

        TestUtils.Simulate.click(button);
        buttonText = TestUtils.findRenderedDOMComponentWithClass(instance, 'buttonStatus');
        expect(buttonText.getDOMNode().textContent).toBe('Clicked me');
    });
});
```

```
jest.dontMock('../public/src/scripts/button/index.js');

describe('ClickButton', function() {
    it('changes state when user clicks a button', function() {
        var React = require('react/addons');

        var Component = React.createFactory(require('../public/src/scripts/button/index.js'));
        var TestUtils = React.addons.TestUtils;
        var instance = TestUtils.renderIntoDocument(Component);

        var button = TestUtils.findRenderedDOMComponentWithClass(instance, 'button');

        TestUtils.Simulate.click(button);
        buttonText = TestUtils.findRenderedDOMComponentWithClass(instance, 'buttonStatus');
        expect(buttonText.getDOMNode().textContent).toBe('Clicked me');
    });
});
```

```
jest.dontMock('../public/src/scripts/button/index.js');

describe('ClickButton', function() {
    it('changes state when user clicks a button', function() {
        var React = require('react/addons');

        var Component = React.createFactory(require('../public/src/scripts/button/index.js'));
        var TestUtils = React.addons.TestUtils;
        var instance = TestUtils.renderIntoDocument(Component);

        var button = TestUtils.findRenderedDOMComponentWithClass(instance, 'button');

        TestUtils.Simulate.click(button);
        buttonText = TestUtils.findRenderedDOMComponentWithClass(instance, 'buttonStatus');
        expect(buttonText.getDOMNode().textContent).toBe('Clicked me');
    });
});
```

```
jest.dontMock('../public/src/scripts/button/index.js');

describe('ClickButton', function() {
    it('changes state when user clicks a button', function() {
        var React = require('react/addons');

        var Component = React.createFactory(require('../public/src/scripts/button/index.js'));
        var TestUtils = React.addons.TestUtils;
        var instance = TestUtils.renderIntoDocument(Component);

        var button = TestUtils.findRenderedDOMComponentWithClass(instance, 'button');

        TestUtils.Simulate.click(button);
        buttonText = TestUtils.findRenderedDOMComponentWithClass(instance, 'buttonStatus');
        expect(buttonText.getDOMNode().textContent).toBe('Clicked me');
    });
});
```

```
jest.dontMock('../public/src/scripts/button/index.js');

describe('ClickButton', function() {
    it('changes state when user clicks a button', function() {
        var React = require('react/addons');

        var Component = React.createFactory(require('../public/src/scripts/button/index.js'));
        var TestUtils = React.addons.TestUtils;
        var instance = TestUtils.renderIntoDocument(Component);

        var button = TestUtils.findRenderedDOMComponentWithClass(instance, 'button');

        TestUtils.Simulate.click(button);
        buttonText = TestUtils.findRenderedDOMComponentWithClass(instance, 'buttonStatus');
        expect(buttonText.getDOMNode().textContent).toBe('Clicked me');
    });
});
```

```
jest.dontMock('../public/src/scripts/button/index.js');

describe('ClickButton', function() {
    it('changes state when user clicks a button', function() {
        var React = require('react/addons');

        var Component = React.createFactory(require('../public/src/scripts/button/index.js'));
        var TestUtils = React.addons.TestUtils;
        var instance = TestUtils.renderIntoDocument(Component);

        var button = TestUtils.findRenderedDOMComponentWithClass(instance, 'button');

        TestUtils.Simulate.click(button);
        buttonText = TestUtils.findRenderedDOMComponentWithClass(instance, 'buttonStatus');
        expect(buttonText.getDOMNode().textContent).toBe('Clicked me');
    });
});
```

```
⚡ ./node_modules/.bin/jest __tests__/clickButton.js
Using Jest CLI v0.2.1
 PASS  __tests__/clickButton.js (6.576s)
1 test passed (1 total)
Run time: 6.799s
```

WAIT TIL YOU SEE THE SIZE OF MY

ROUTING TABLE

Routing

React does not have a native router

There are however a few to choose between

React-router
React-router-component
Monorouter

React-router example

```
// Define react-router routes
var routes = (
    <Route name="/" handler={Layout}>
        <DefaultRoute handler={require('./home')} />
        <Route name="home" handler={require('./home')} />
        <Route name="contact" handler={require('./contact')} />
        <Route name="about" handler={require('./about')} />
        <Redirect from="/" to="home" />
    </Route>
);

// Run the router
Router.run(routes, function (Handler) {
    // Render the root app view-controller
    React.render(<Handler />, document.body);
});
```

Inline Styles

So inline styles, eh?

There's actually a good reason for doing this.

So inline styles, eh?

CSS pollutes the global namespace

At scale, this is bad because it leads to paralysis and confusion.

Can I add this element, or change this class? If you're not sure, you're in trouble.

So inline styles, eh?

Inline styles avoid this, because the CSS is scoped to the component you're working with.

How it looks

```
module.exports = React.createClass({
    displayName: "Home",

    render() {
        var inlineCss={
            padding:'10px',
            lineHeight:'16px',
            color:'red'
        };
        return <div >
            <div className="flyin-widget">
                <h1 style={inlineCss}>Home</h1>



            </div>
        </div>
    }
});
```

Not your 80s inline

```
<h1 style={inlineCss}>Home</h1>
```

It's not really "inline". We merely pass a reference to a rule that’s somewhere else in the file, just like CSS.

Style is actually a much better name than class. You want to “style” the element, not “class” it.

Finally, this is not applying the style directly, this is using React virtual DOM and is being diff-ed the same way elements are.

Still....

The goal is not to replace CSS as it's done today.
It's simply focusing on the fundamental problem with CSS and trying to solve it.

You do not have to use it. If you apply a **className** tag to your elements, you can use CSS as you've always done.

Mixins

Basically, pure React components that can be incorporated in your other components

Mixins

Components that use mixins inherits state and props from the mixin

Mixins

```
var SetIntervalMixin = {

    componentWillMount: functi
        this.intervals = [];
    },
    setInterval: function() {
        this.intervals.push(se
    },
    componentWillUnmount: func
        this.intervals.map(cle
    }
};
```

```
var Mixin = React.createClass({
    displayName: "Home",
    getInitialState(){ return{ seconds:0 } },
    mixins:[SetIntervalMixin],
    statics: { increment(n) { return n + 1; }},
    componentDidMount() { this.setInterval(this.tick, 1000); },
    tick() { this.setState(
        {seconds: Mixin.increment(this.state.seconds)})
    },
    render() {
        return <div >
            <div className="flyin-widget">
                <h1>Mixin</h1>
                {this.props.name} has been running
                for {this.state.seconds} {this.unit} seconds
            </div>
        </div>
    }
});
```

Last words

Virtual DOM, a native **event system**
and other technicalities are nice

But Reacts **true strength** are actually
none of these

Last words

Reacts true strengths are:

Unidirectional Data Flow

Freedom from **Domain Specific Language** (it's all JavaScript)

Explicit **Mutation**