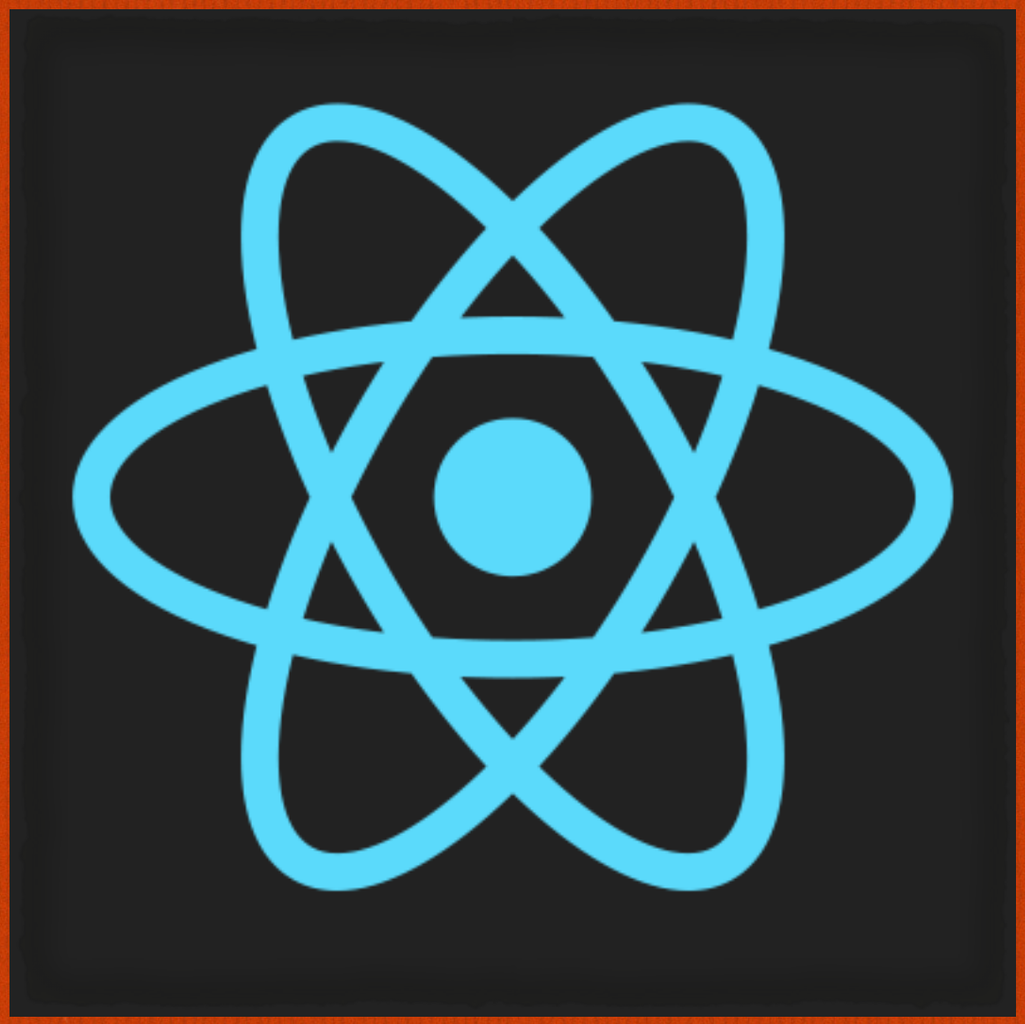
React.JS

Ynon Perek
ynon@ynonperek.com

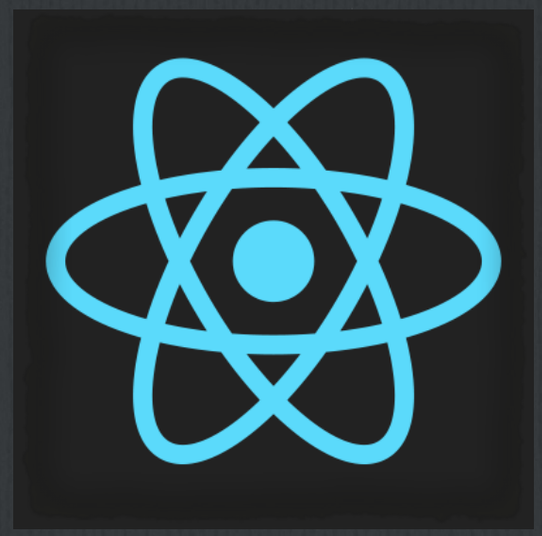


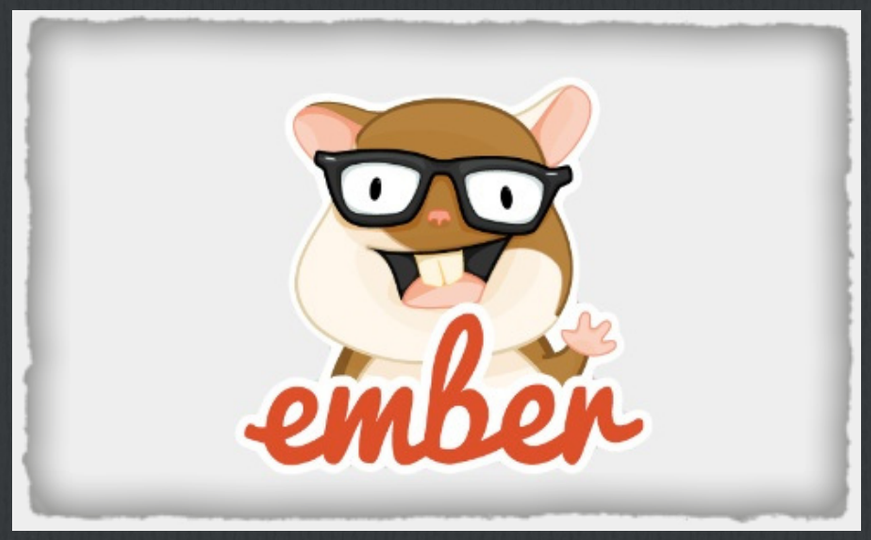
Offer a ***consistent*** and ***simple*** way to write web applications

Web Application Challenges

- [ ] Code architecture
- [ ] Split UI to components
- [ ] Navigation
- [ ] Application State

Client Side Frameworks





Why React

- [ ] Easy to learn
- [ ] Carries heavy weight

Who Uses React

React IS NOT

- [ ] An MVC framework
  (it won't tell you how to manage your data)

- [ ] A SPA framework
  (it won't handle client-side routing)

Let's See Some Code (our first React component)

```
class App extends React.Component {
  render() {
    return (
      <div>
        <p>Hello World</p>
      </div>
    );
  }
}

ReactDOM.render(<App />, document.querySelector('main'));
```

https://codepen.io/ynonp/pen/EwOEov

React How

- [ ] Use Babel to convert JSX code to JS
- [ ] Use React.render(…) to paint a component on screen

JSX vs. JS

```
App.prototype.render = function render() {
  return React.createElement(
    'div',
    null,
    React.createElement(
      'p',
      null,
      'Hello World'
    )
  );
};
```

Customising Components

- [ ] Components take properties when created
- [ ] That’s how outer components affect inner

Customising Components

```
class App extends React.Component {
  render() {
    return (
      <div>
        <p>{this.props.message}</p>
      </div>
    );
  }
}

ReactDOM.render(<App message="Hello World" />,
                document.querySelector('main'));
```

Customising Components

- [ ] Can pass any JS object as value for a property
- [ ] Remember JSX -> JS

Customising Components

```
class App extends React.Component {
  render() {
    return (
      <div>
        <p>I got {this.props.data.length} numbers</p>
      </div>
    );
  }
}

const numbers = [10, 20, 30, 40];
ReactDOM.render(<App data={numbers} />,
  document.querySelector('main'));
```

Using Properties

- [ ] Changing properties after rendering won’t repaint your components
- [ ] So best to keep your properties immutable

Dynamic Components

- [ ] A dynamic component has a “state”
- [ ] That state changes during the component’s life cycle
- [ ] Can have initial state

Demo: Let's Write a Click Counter

- [ ] Internal State: How many clicks performed so far
- [ ] External Input: None

Demo: Clicks Counter

```
class Counter extends React.Component {
  constructor(props) {
    super(props);
    this.state = { clicks: 0 };

    this.clicked = this.clicked.bind(this);
  }

  clicked() {
    this.setState((oldState) => ({ clicks: oldState.clicks + 1 }))
  }

  render() {
    return (
      <div>
        <button onClick={this.clicked}>Click Me</button>
        <p>You clicked {this.state.clicks} times</p>
      </div>
    );
  }
}
```

The Good

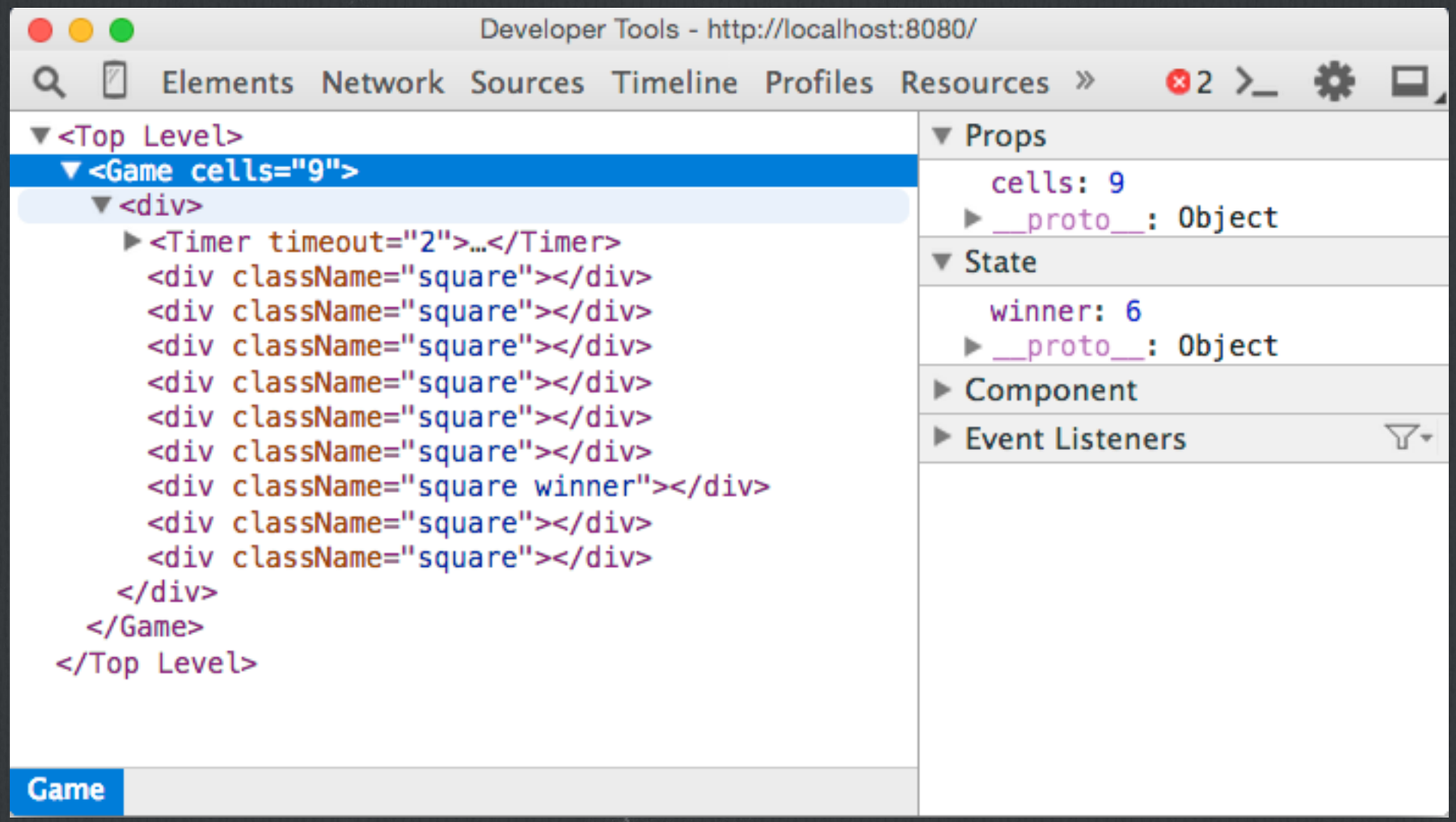
- [ ] No need to manipulate DOM nodes
- [ ] Combine components to build larger apps

React Debugging

- [ ] Webpack creates source maps:
  - [ ] Can debug inside chrome devtools
  - [ ] Debug with “original” source
  - [ ] Demo

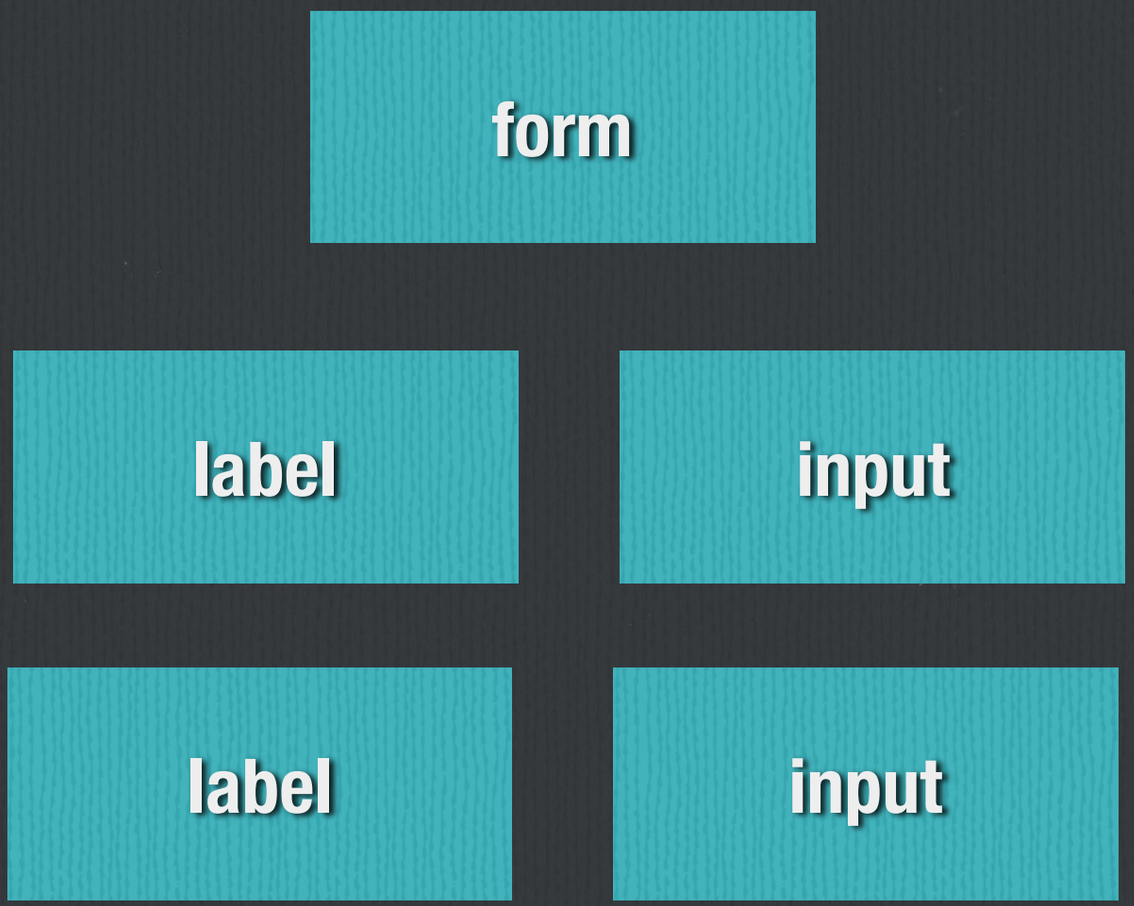
React Debugging

- [ ] React Chrome App



React: Behind The Scene

Virtual DOM



React: Behind The Scene

Virtual DOM

- form
  - label
  - input
  - label
  - input

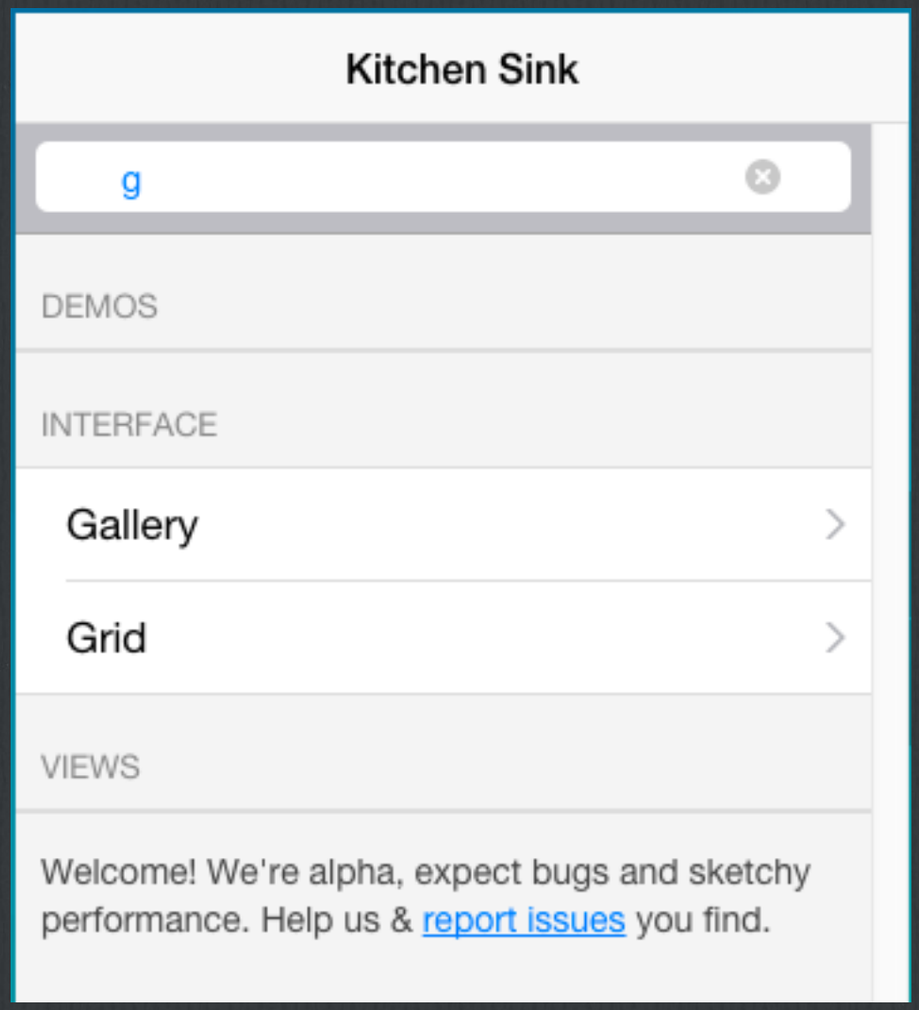
Virtual DOM

- form
  - label
  - input

React: Behind The Scene

- [ ] Calculate how to get from virtual DOM tree A to virtual DOM tree B
- [ ] In our example:
  - [ ] Delete last input and label

Component Based UI



Q & A

Handling External Inputs

- [ ] Define properties you component should use
- [ ] Example: Let's build a filterable list
  - [ ] State: current filter text
  - [ ] Properties: array of items to display and filter
- [ ] Solution: http://codepen.io/ynonp/pen/pJbRjV

React + TDD

Agenda

- [ ] TDD short intro
- [ ] The tools
- [ ] React It: translating old JS to React using TDD
- [ ] Conclusions & Takeaways

Hello TDD

- [ ] TDD helps you write easy to test code
- [ ] Write tests first

TDD + React

- [ ] React helps us avoid code pitfalls
- [ ] TDD makes sure we get it right

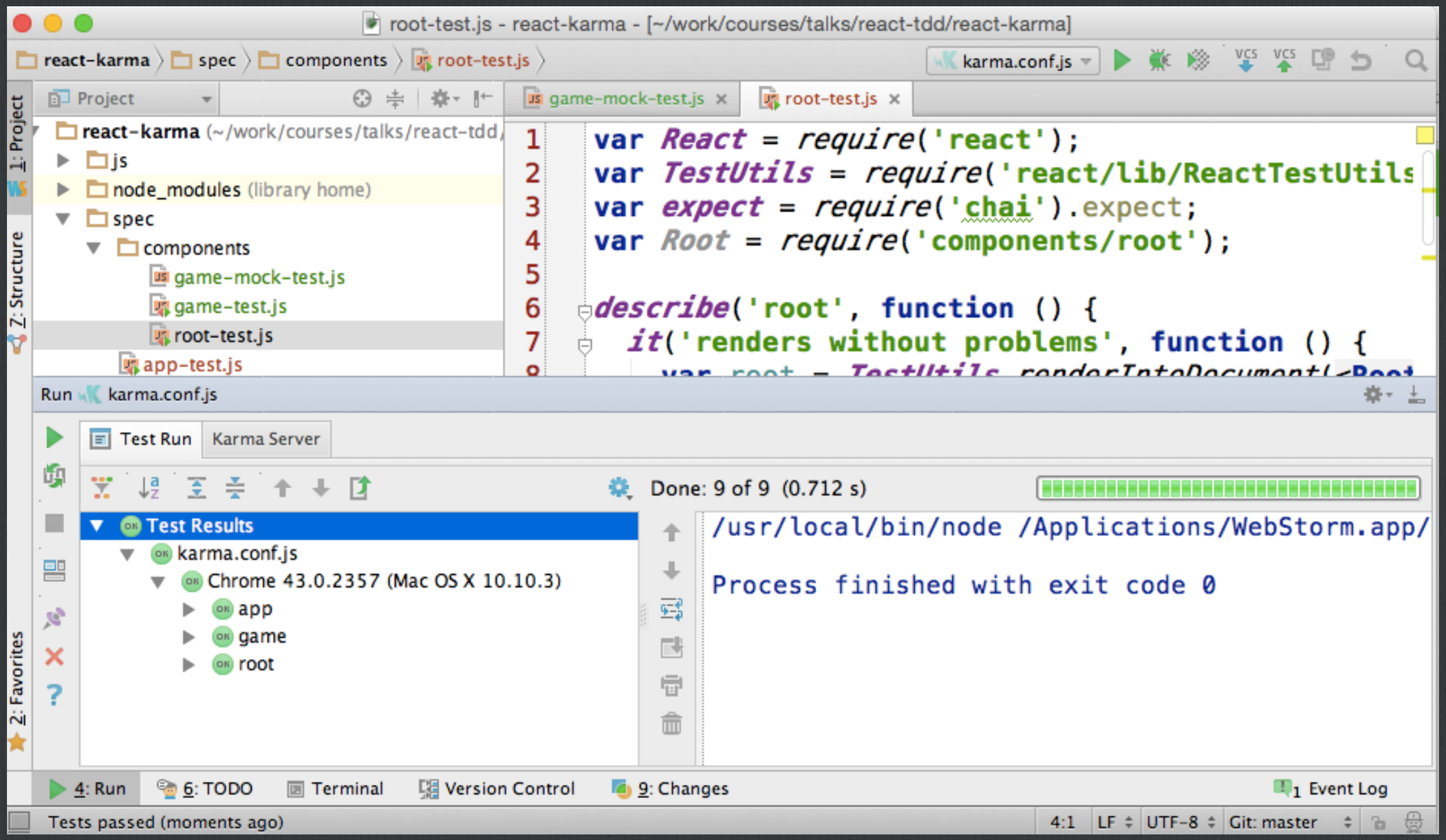
TDD + React

Markup

Code

Spec

TDD Workflow



TDD Workflow

- [ ] Run tests from within IDE, while coding
- [ ] Fix code and build more tests as you go along
- [ ] Eventually check look & feel in the browser

TDD Tools

- [ ] Webstorm
- [ ] Webpack
- [ ] Karma
- [ ] Mocha / Sinon / Chai
- [ ] React.addons.TestUtils

Demo: Vanilla -> TDD React

- [ ] Given the square game here: http://codepen.io/ynonp/pen/YXrPNj?editors=101
- [ ] Let's write a React version using TDD

Start with a test

```
describe('Game cells', function() {
  it('renders .square child nodes', function() {
    var game = TestUtils.renderIntoDocument(<Game cells={9} />);

    var squares = TestUtils.scryRenderedDOMComponentsWithClass(game, "square");
    expect(squares.length).to.eq(9);
  });
});
```

Write Some Code

```
render: function () {
  return (
    <div>
    { _.map(_.range(this.props.cells), this.renderItem)}
    </div>
  )
}
```

React.addons.TestUtils

- [ ] renderIntoDocument
- [ ] scry / find functions to find React components and DOM nodes
- [ ] Simulate functions to simulate DOM events

Simulate and stubs

- [ ] Let’s randomise the winner
- [ ] Let’s change the winning square after each click

Problem With Random

Stubbing with sinon

```
it('is selected randomly using _.sample', function() {
  sinon.stub(_, "sample").returns(42);

  var game = TestUtils.renderIntoDocument(<Game cells={3} />);
  expect(game.state.winner).to.eq(42);

  _.sample.restore();
});
```

Other useful stubs

- [ ] Stubbing clocks with sinon.fakeTimers
- [ ] Stubbing ajax with sinon.fakeServer

Other useful stubs

- [ ] Sub components (rewire)
- [ ] Demo: Game Timer

Q & A

Why TDD

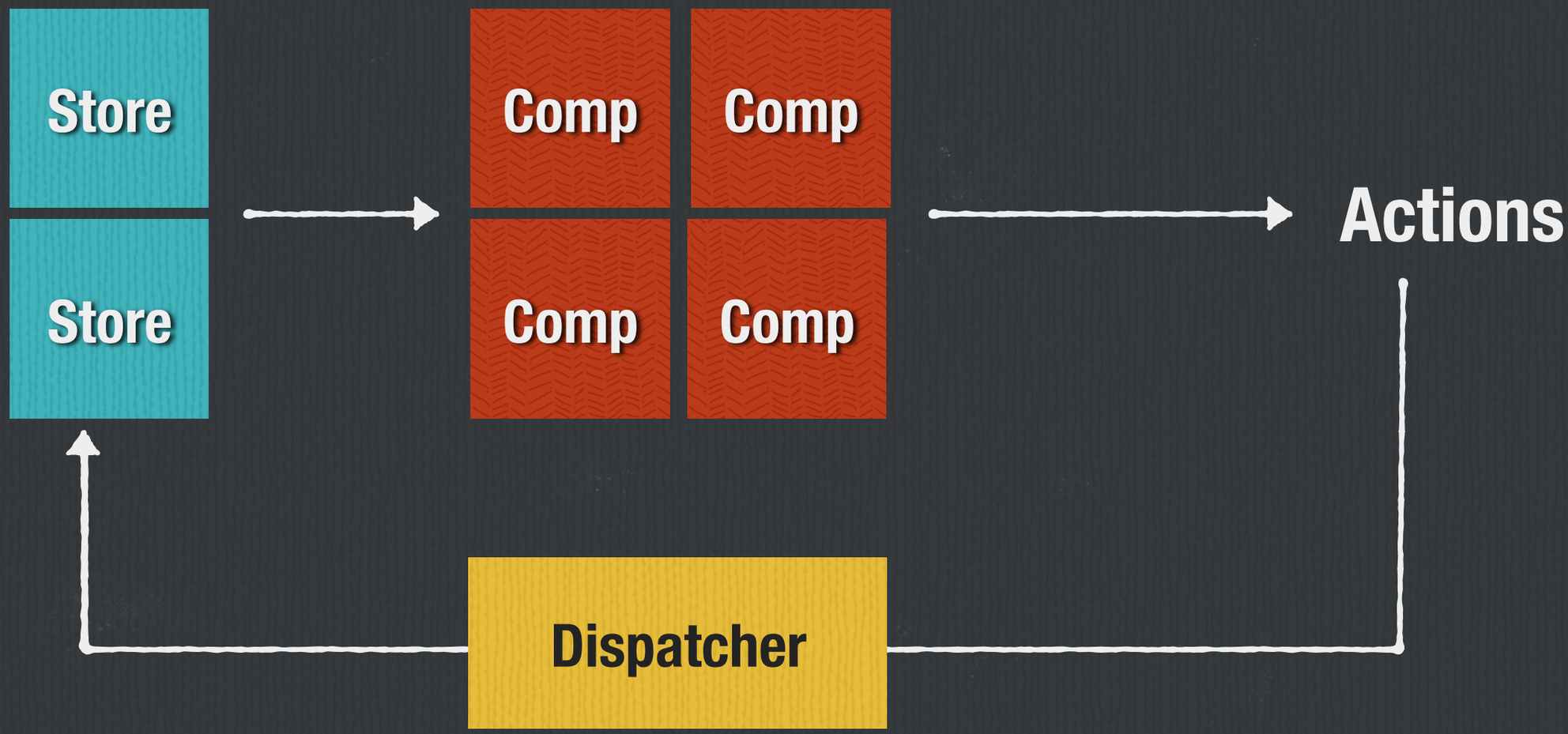
- [ ] Coding was fun
- [ ] The tests are useful going forward
- [ ] Better code

React + Flux Architecture

React Architecture

- [ ] No architectural constraints
  - [ ] Some write their own
  - [ ] Some use MVC (backbone, angular, ember)
  - [ ] Some use flux

Flux Architecture



Flux

- [ ] Unidirectional application architecture
- [ ] Has many implementations: Flux, Reflux, Lux, fluxthis and more
- [ ] https://github.com/voronianski/flux-comparison

Flux Components

- [ ] Stores: Save data
- [ ] Dispatcher: handles actions
- [ ] Actions: manage flow
- [ ] React Components

Flux Demo

- [ ] Random message generator:
  - [ ] Users store
  - [ ] Messages store
  - [ ] New messages => create new users
- [ ] Allow pause/resume using a ListeningStore

Q & A

Thanks For Listening

- Ynon Perek
- ynon@ynonperek.com
- www.tocode.co.il