

Intro To React - Day 1

Devpoint Labs - Jake Sorce / Dave Jungst

Rails is a full stack framework why use ReactJS?

- Speed!
 - ReactJS is fast and delivers a better user experience
 - No detectable client side latency
- Structured JS
 - No more spaghetti code in JS
- Modern web experience
- Happy developers
 - Writing JS is fun again

My friend said Ember / Angular / Handlebars / etc..

The subject of front end frameworks causes great debate in the industry.

Often ReactJS is compared to other frameworks that have been around for longer but this comparison is not fair.

Most front end frameworks such as angular implement a full MVC solution.

ReactJS is simply a view layer. This means we get to keep our logic in rails and let rails do what it does best while keeping our views in the front end framework.

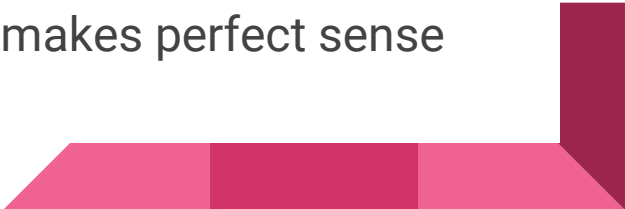


Breaking down some walls

The biggest struggle developers face with ReactJS is that we write our HTML directly in our Javascript. Developers and Bootcamps really try to emphasise the concept of separation of concerns. Controllers go here, models go here, views go here, javascript goes over there.

Separation of concerns is good practice and makes our apps more maintainable but just like everything else this concept is outdated and needs an update.

Don't think separation of concerns with ReactJS think separation of technologies. When you adopt the idea of separation of technologies it makes perfect sense that Javascript and HTML can exist together.



How ReactJS works

React is broken down into reusable individual components.

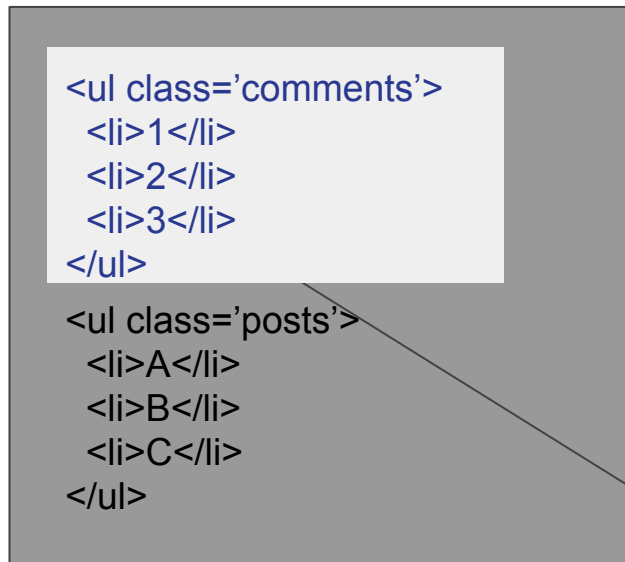
React elements have 4 components: props, key, ref, type. There are no methods on the prototype.

React uses the concept of state. Meaning the state of the app at any given point in time. It is bad practice to manually update the DOM in ReactJS, for example adding elements with JQuery. In React we preserve the state of the DOM at all times and allow React's virtual DOM to make updates for us.



Virtual DOM

DOM



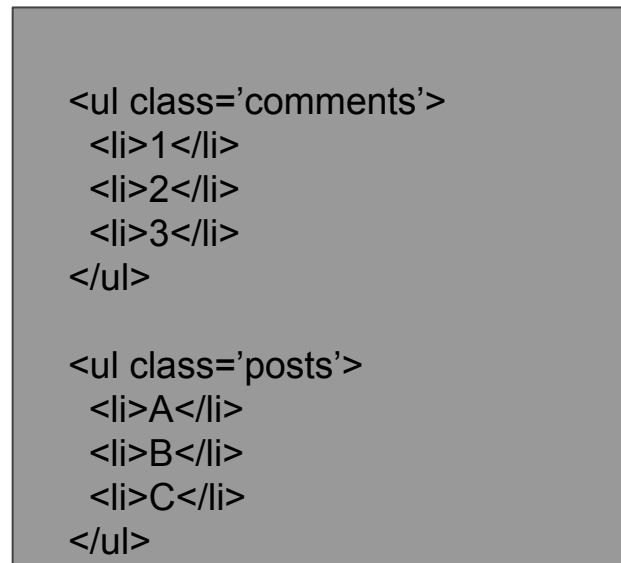
DIFF

3

Update

```
<ul class='comments'>
  <li>1</li>
  <li>2</li>
  <li>3</li>
</ul>
```

V-DOM



ReactJS on Rails

There are several different ways to incorporate ReactJS with Rails

The one that we will focus on in this lecture is through the `react_rails` gem

This is the most straightforward way and it will remove complexities so that the focus can remain on React VS implementations



Build a rails app

```
rails new react_todo_list -d postgresql -T --skip-turbolinks
```

(note if using turbolinks react must be required after turbolinks)

```
gem 'materialize-sass'  
gem 'react-rails'
```

*NOTE Put the react-rails gem at the bottom of the Gemfile bundle

```
rails g controller lists index
```

```
root 'lists#index'
```

```
rails g model item name complete:boolean
```

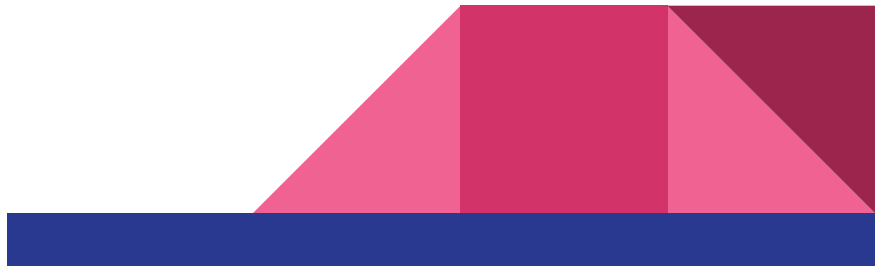
```
app/assets/stylesheets/application.css.scss
```

```
@import 'materialize';
```

```
app/assets/javascripts/application.js
```

```
//= require jquery
```

```
//= require materialize-sprockets
```



Installing react

rails g react:install

```
create app/assets/javascripts/components
create app/assets/javascripts/components/.gitkeep
insert app/assets/javascripts/application.js
insert app/assets/javascripts/application.js
insert app/assets/javascripts/application.js
create app/assets/javascripts/components.js
```

The react installer creates a folder called components and is also nice enough to inject all the necessary require fields into your application.js file

```
13 //= require jquery
14 //= require jquery_ujs
15 //= require materialize-sprockets
16 //= require react
17 //= require react_ujs
18 //= require components
19 //= require_tree .
```

Model, View, Controller

app/controllers/lists_controller.rb

```
def index
```

```
  @items = Item.all
```

```
end
```

app/views/lists/index.html.erb

```
<%= react_component 'List', { items: @items } %>
```



Components

The key to ReactJS is individual components.
In the view I specified a component called List.

This will be created in a file called List.js.jsx

List is the name of the component.

getInitialState sets lists to the lists property that was passed in.

getDefaultState takes care of cases where nothing is passed in.

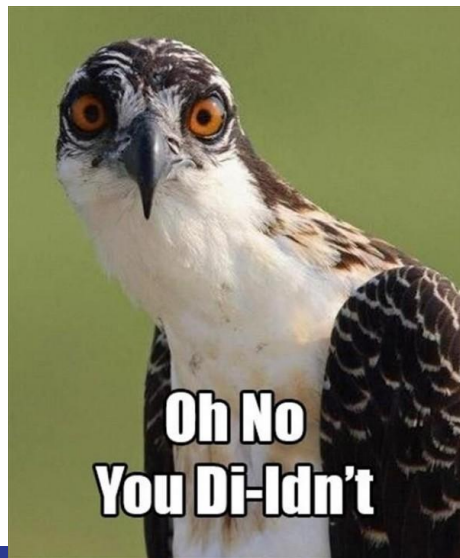
app/assets/javascripts/components/List.js.jsx

```
var List = React.createClass({  
  getInitialState: function() {  
    return { items: this.props.items }  
  },  
  
  getDefaultState: function() {  
    return { items: [] };  
  },  
});
```

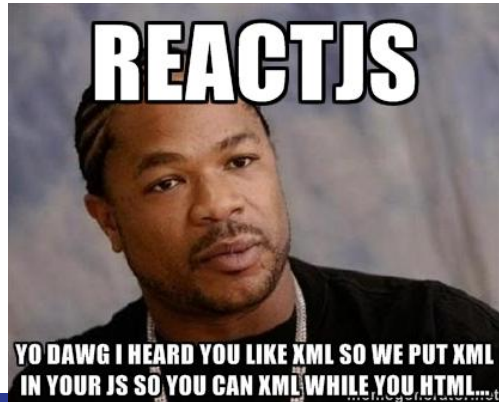
Render

The render function is what is actually shown on the page. JSX is react's way of parsing HTML written in our JS file to actually show up as an element. It is important that each React component is wrapped in a single div

```
render: function() {  
  return(<div>  
    <h1> My first component </h1>  
    </div>);  
}
```



Appropriate Memes



State / Props

Props is short for properties. Props are initially passed into components either from the view `<%= react_component 'MyComponent', { prop1: value, prop2: value } %>`

Or from another component.

State is the current state of the component at any given point. The value of a prop passed in can change with the state.

```
this.props.prop1 => 'A'
```

```
this.state.prop1 => 'B'
```



Click Events, className

Since we are using JavaScript we can't use keywords like class so in react an HTML class is defined with className.

```
14  render: function() {  
15    return (<div>  
16      <a className='waves-effect waves-light btn' onClick={this.showAddForm}>Add Item</a>  
17      <div className='card blue-grey darken-1'>  
18        <div className='card-content white-text'>  
19          <span className='card-title'>To Do</span>  
20        </div>  
21      </div>  
22    </div>);  
23  }  
24 }
```

On the <a> we have added a click event with onClick={this.showAddForm}

We use expressions { } to run a block of code in the JSX.

Changing State


```
10 showAddForm: function() {  
11   this.setState({ showAdd: !this.state.showAdd});  
12 },  
13
```

We are changing the state of our app with `setState` and setting a property of `showAdd` to the opposite of whatever it is at the current state. This creates a toggle effect.



Showing the form

```
26 render: function() {  
27   return (<div>  
28     <a className='waves-effect waves-light btn' onClick={this.showAddForm}>Add Item</a>  
29     {this.addItemForm()}  
30     <div className='card blue-grey darken-1'>  
31       <div className='card-content white-text'>  
32         <span className='card-title'>To Do</span>  
33       </div>  
34     </div>  
35   </div>);  
36 }  
37 })
```



We can call a method inside of our HTML by using expressions

Getting things ready for submit on the Rails side

rails g controller items

app/controllers/items_controller.rb

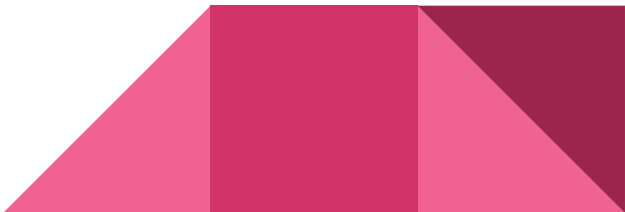
```
class ItemsController < ApplicationController
  def create
    item = Item.create(item_params)
    render json: item
  end

  private

  def item_params
    params.require(:item).permit(:name)
  end
end
```

config/routes.rb

```
1 Rails.application.routes.draw do
2   root 'lists#index'
3   resources 'items'
4 end
```



The reactJS form

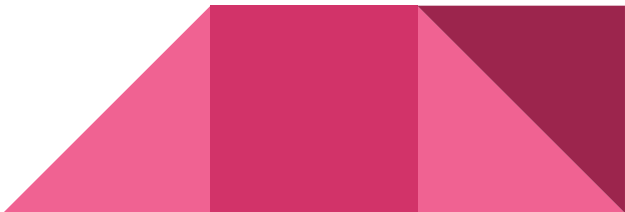
```
addItemForm: function() {  
  if(this.state.showAdd){  
    return(<div>  
      <form onSubmit={this.submitItem}>  
        <div className='input-field'>  
          <input autoFocus='true' placeholder='add item' type='text' onChange={this.addItemName} />  
          <button className='btn waves-effect' type='submit'>Save</button>  
        </div>  
      </form>  
    </div>);  
  }  
},
```

First we check the state of showAdd. If it is true then we return a form element.
In the form we call a method onSubmit
Notice autoFocus is camel cased.
onChange of the input box we are setting a state to use later.

itemName

Everytime the user types in the input box we want the state of itemName to change since we can't use serializeArray with ReactJS. We also need to pass in the event if we want to use it in the function.

```
addItemName: function(e) {  
  this.setState({ itemName: e.currentTarget.value });  
},
```



Submitting the form

```
submitItem: function(e) {  
  e.preventDefault();  
  var name = this.state.itemName;  
  var self = this;  
  $.ajax({  
    url: '/items',  
    type: 'POST',  
    data: { item: { name: name }},  
    success: function(data) {  
      var items = self.state.items;  
      items.push({ name: data.name, complete: data.complete });  
      self.setState({ items: items, showAdd: false, itemName: null });  
    }  
  });  
},
```

We pass in (e) so that we can prevent default.

We also need to set a variable to hold “this” so we can use it in the success function.

In the success function we get our current state of items and push the item returned from the ajax call on to the array.

Finally we set the state of items, set showAdd to false to hide the form and set itemName to null

Displaying Items

We want to call a method to display all of our items in the body of the materialize card.

```
render: function() {  
  return (<div>  
    <a className='waves-effect waves-light btn' onClick={this.showAddForm}>Add Item</a>  
    {this.addItemForm()}  
    <div className='card blue-grey darken-1'>  
      <div className='card-content white-text'>  
        <span className='card-title'>To Do</span>  
        {this.displayItems()}  
      </div>  
    </div>  
  </div>);  
}
```

Display items code

```
displayItems: function() {  
  var items = [];  
  for(var i = 0; i < this.state.items.length; i++){  
    var id = "checkbox-" + i;  
    items.push(<li>  
      <div className='row'>  
        <div className='col s10'>  
          {this.state.items[i].name}  
        </div>  
        <div className='col s2'>  
          <input type='checkbox' id={id} checked={this.state.items[i].complete} />  
          <label htmlFor={id}>Complete?</label>  
        </div>  
      </div>  
    </li>);  
  }  
  return items;  
},|
```