

React - Day 2

Devpoint Labs - Jake Sorce / Dave Jungst

componentDidMount

Often with ReactJS you want code to run after your component is rendered on the page. `componentDidMount` will do precisely this. Instead of passing properties in our views we can update them after the component is mounted.

```
1 var List = React.createClass({
2   getInitialState: function() {
3     return { items: [] };
4   },
5
6   componentDidMount: function() {
7     var self = this;
8     $.ajax({
9       url: '/items',
10      type: 'GET',
11      success: function(data) {
12        self.setState({ items: data });
13      }
14    });
15  },
```

`getInitialState` is required but we can just set the state to an empty array and get rid of `getDefaultState`.

Once the component is mounted we make an ajax call to the items index route and set the state of our items

On the server side

app/controllers/items_controller.rb

def index

render json: Item.all

end

app/views/list/index.html.erb

<%= react_component 'List' %>

We return all of the Items in the items#index route.

We no longer pass props in through the view.

We also need to update the lists_controller to not return items.

Benefit

It makes sense from an API standpoint that we shouldn't be returning items in our lists controller.

By moving this logic to the items_controller there is a more clear and reusable API.

Also by moving the logic to retrieve these items from the view to the React Component helps to further decouple the view from the component. Since we can have many react components on a single page they should be decoupled and made to work in a more reusable detached way.

Children

Components can have child components. In the case of this app we have Lists and items inside the same component and it would make more sense if an item was a stateless component of a List.

This would make calling methods on the child more self contained and keep our app even more separated. Let's refactor our List so that a Child is actually a separate component which receives it's own props.

Child components can contain their own props passed down from the parent component, their own state, and their own logic.

Items as a child component

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

```
1 var Item = React.createClass({
2   render: function() {
3     var id = "item-" + this.props.id;
4     return(<li>
5       <div className='row'>
6         <div className='col s10'>
7           {this.props.name}
8         </div>
9         <div className='col s2'>
10          <input type='checkbox' id={id} checked={this.props.complete} />
11          <label for={id}>Complete?</label>
12        </div>
13      </div>
14    </li>);
15  }
16 });
```

We are treating the item as a stateless component meaning we only care about the props that are passed down instead of the state. We will leave the state of the child to be managed by the parent. In this case the list.

app/assets/javascript/components/List.jsx

```
displayItems: function() {  
  var items = [];  
  for(var i = 0; i < this.state.items.length; i++){  
    var item = this.state.items[i];  
    items.push(<Item id={item.id} name={item.name} complete={item.complete} />);  
  }  
  return items;  
},
```

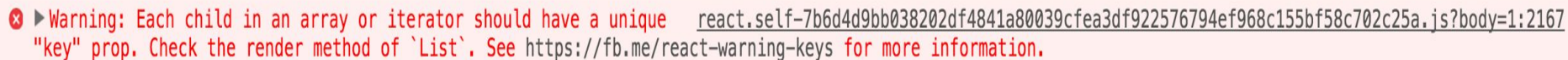
Update the displayItems method to return an Item component instead of the

Notice the props are passed in just as if they were HTML properties.

id, name, and complete can be accessed in the Item component as this.props.id, this.props.name, this.props.complete.

WARNING!!! keys

If you open up the console you will see a warning about child components not having a unique key. Anytime we are looping to create child components ReactJS needs a way to keep them separate. This is especially important if we want to call functions on these objects.

A screenshot of a browser's developer console showing a warning message. The message is in red text on a light pink background. It starts with a red 'x' icon and a right-pointing triangle. The text reads: 'Warning: Each child in an array or iterator should have a unique "key" prop. Check the render method of `List`. See https://fb.me/react-warning-keys for more information.' The URL is underlined.

Warning: Each child in an array or iterator should have a unique [react.self-7b6d4d9bb038202df4841a80039cfea3df922576794ef968c155bf58c702c25a.js?body=1:2167](https://fb.me/react-warning-keys) "key" prop. Check the render method of `List`. See <https://fb.me/react-warning-keys> for more information.

Keys should be passed as a prop and should be complete unique from any other keys on the page.

key

The easiest way to set a key is to call it whatever the object is concatenated with the object's ID ensuring that it will be unique.

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

```
displayItems: function() {  
  var items = [];  
  for(var i = 0; i < this.state.items.length; i++){  
    var item = this.state.items[i];  
    var key = "Item-" + item.id;  
    items.push(<Item key={key} id={item.id} name={item.name} complete={item.complete} />);  
  }  
  return items;  
},
```

Checking the item

Now that there is a key we can work on checking the item and updating the list to reflect checked items.

```
render: function() {  
  var id = "item-" + this.props.id;  
  return(<li>  
    <div className='row'>  
      <div className='col s10'>  
        {this.props.name}  
      </div>  
      <div onClick={this.checkItem} className='col s2'>  
        <input type='checkbox' id={id} checked={this.props.complete} />  
        <label for={id}>Complete?</label>  
      </div>  
    </div>  
  </li>);  
}
```

```
var Item = React.createClass({  
  checkItem: function() {  
    $.ajax({  
      url: '/check_item',  
      type: 'PUT',  
      data: { item: { complete: !this.props.complete }, id: this.props.id }  
    });  
  },  
});
```

On the Rails side

```
def check_item
  checked = params[:item][:complete] == 'true' ? true : false
  item = Item.find(params[:id])
  item.update(complete: checked)
  render json: item
end
```

```
Rails.application.routes.draw do
  root 'lists#index'
  resources 'items'

  put 'check_item', to: 'items#check_item'
end
```

Now if you check an item and refresh the page it will reflect it's checked status. This brings up 2 issues.

1. We don't want to have to refresh the page to reflect the change.
2. Items are coming back in an order that we don't really expect.

To fix the second issue let's just update `Item.all` to order by created at

```
def index
  render json: Item.all.order(:created_at)
end
```

Updating the list on check

We need the List to fetch all of the items again after an item is checked so that we can update the state and re-render to pass down appropriate props to the items. To do this we will use a callback. A callback allows the child to call a parent method when it's updated.

```
displayItems: function() {  
  var items = [];  
  for(var i = 0; i < this.state.items.length; i++){  
    var item = this.state.items[i];  
    var key = "Item-" + item.id;  
    items.push(<Item refreshList={this.refreshList} key={key} id={item.id} name={item.name} complete={item.complete} />);  
  }  
  return items;  
},
```

Callback methods are passed as props to the child element.

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

```
refreshList: function() {  
  var self = this;  
  $.ajax({  
    url: '/items',  
    type: 'GET',  
    success: function(data) {  
      self.setState({ items: data });  
    }  
  });  
},
```

This method just pulls in all of our items again and sets the state. Since we are doing this same logic twice let's update componentDidMount

```
componentDidMount: function() {  
  this.refreshList();  
},
```

Calling the callback

All that is left to do is call our callback inside of the child component, since it is passed as a prop it gets called with `this.props` as well....

```
checkItem: function() {  
  var self = this;  
  $.ajax({  
    url: '/check_item',  
    type: 'PUT',  
    data: { item: { complete: !this.props.complete }, id: this.props.id },  
    success: function() {  
      self.props.refreshList();  
    }  
  });  
},
```

A little more style

app/assets/stylesheets/custom.css

```
.checked {  
  text-decoration: line-through;  
  color: grey;  
}
```

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

```
render: function() {  
  var id = "item-" + this.props.id;  
  var checked = this.props.complete ? 'checked' : '';  
  var itemClass = 'col s10 ' + checked;  
  return(<li>  
    <div className='row'>  
      <div className={itemClass}>  
        {this.props.name}  
      </div>  
      <div onClick={this.checkItem} className='col s2'>  
        <input type='checkbox' id={id} checked={this.props.complete} />  
        <label for={id}>Complete?</label>  
      </div>  
    </div>  
  </li>);  
}
```

Update the add to take an ID

Our add doesn't work correctly anymore so we need to update the submitItem success to include the id.

```
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({ id: data.id, name: data.name, complete: data.complete });  
      self.setState({ items: items, showAdd: false, itemName: null });  
    }  
  })  
}
```


Edit an Item

First let's get our rails side ready

app/controllers/items_controller.rb

```
def update
  item = Item.find(params[:id])
  item.update(item_params)
  render json: item
end
```

rendering the item or a form

app/assets/javascript/components/Item.js.jsx

```
getInitialState: function() {  
  return { edit: false };  
},  
  
toggleEdit: function() {  
  this.setState({ edit: !this.state.edit });  
},
```

Since our Component will have a concept of editing or not editing as a state we need an initial state.

toggleEdit will just change the state of edit to whatever it currently is not.

updating the render function

app/assets/javascript/components/Item.js.jsx

```
render: function() {  
  if(this.state.edit) {  
    return this.edit();  
  } else {  
    return this.item();  
  }  
}
```

Now we can render based on the state of the component.

The edit function

```
edit: function() {  
  return(<li>  
    <div className='row'>  
      <div className='col s10'>  
        <form onSubmit={this.updateItem}>  
          <input autoFocus={true} type='text' defaultValue={this.props.name} ref='itemName' />  
        </form>  
      </div>  
      <div className='col s2'>  
        <a onClick={this.toggleEdit}>Cancel</a>  
      </div>  
    </div>  
  </li>);  
},
```

Notice the use of `defaultValue` instead of `value`. Setting `value` in a form will set the element to readonly.

Also we are using `ref` which will be explained in the next slide.

Update

```
updateItem: function() {  
  var name = ReactDOM.findDOMNode(this.refs.itemName).value;  
  var self = this;  
  $.ajax({  
    url: '/items/' + this.props.id,  
    type: 'PUT',  
    data: { item: { name: name }},  
    success: function() {  
      self.props.refreshList();  
    },  
  });  
},
```

The only thing new here is the `ReactDOM.findDOMNode` line.

This is a way to get the actual dom element in a safe way without using JQuery and without disturbing the state of the DOM.

Delete (rails side)

app/controllers/items_controller.rb

```
def destroy
  Item.find(params[:id]).destroy
  head :ok
end
```

Since we are not doing any validations we need to render HTML or JSON a shortcut in rails to send back a 200 JSON response is to just say head :ok

client side

```
deleteItem: function() {  
  var self = this;  
  $.ajax({  
    url: '/items/' + this.props.id,  
    type: 'DELETE',  
    success: function() {  
      self.props.refreshList();  
    }  
  });  
},
```

```
item: function() {  
  var id = "item-" + this.props.id;  
  var checked = this.props.complete ? 'checked' : '';  
  var itemClass = 'col s9' + checked;  
  return(<li>  
    <div className='row'>  
      <div onClick={this.toggleEdit} className={itemClass}>  
        {this.props.name}  
      </div>  
      <div onClick={this.checkItem} className='col s2'>  
        <input type='checkbox' id={id} checked={this.props.complete} />  
        <label for={id}>Complete?</label>  
      </div>  
      <div onClick={this.deleteItem} className='col s1'>  
        X  
      </div>  
    </div>  
  </li>);  
},
```