React - Day 3

Devpoint Labs - Jake Sorce / Dave Jungst

JBuilder

So far we have been sending back JSON objects that include data that we may not need or want.

Unused data is a waste of bandwidth and memory and can slow down the application with a large enough dataset.

Rails includes a gem by default called JBuilder to help build out complex JSON structures in a simplified way.

Formatting JSON

First remove the render line from the ItemsController index method and create an instance variable that can be passed to a jbuilder file.

Default

```
1 class ItemsController < ApplicationController
2   def index
3     @items = Item.all.order(:created_at)
4   end
```

Next create a file called app/views/items/index.json.jbuilder in this file we can format JSON object to only return what we need.

app/views/items/index.json.jbuilder

Since we took out the render line, when the index route is hit rails will look for a file in app/views/ called index.

```
1 json.items @items do |item|
2   json.id item.id
3   json.name item.name
4   json.complete item.complete
5 end
```

Here we create a json object called items and inside of items we loop over our @items adding only information that we need to the object.

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

Since the JSON object looks like { items: { id: .., name: .. }} we will need to fix the refreshList method to look for data.items instead of just data

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

Cleaning up the JSON

Since the JSON object is very simple we can still clean up the JBuilder file even more.

app/views/items/index.json.jbuilder

```
1 json.items @items do |item|
2   json.(item, :id, :name, :complete)
3 end
```

Getting URL’s with JBuilder

We can also return a full url with JBuilder very easily

app/views/items/index.json.jbuilder

```
1 json.items @items do |item|
2   json.(item, :id, :name, :complete)
3   json.url item_url(item)
4 end
```

```
> data.items
<- [▼Object i                                    , ▶Object, ▶Object, ▶Object, ▶Object]
      complete: true
      id: 7
      name: "Test23asdfasf"
      url: "http://localhost:3000/items/7"
    ▶ __proto__: Object
>
```

Updating AJAX with url

Now that we have an items URL we can make a few changes to our React components to make things cleaner. First pass the url to the Item component

app/assets/javascripts/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 refreshList={this.refreshList} key={key} url={item.url} id={item.id} name={item.name} complete={item.complete} />);
  }
  return items;
```

URL cont…

Next in app/assets/javascripts/components/Item.js.jsx notice that we are using '/items/' + this.props.id in updateItem and deleteItem. We can fix this by changing the AJAX url.

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

```
updateItem: function(e) {
  e.preventDefault();
  var self = this;

  $.ajax({
    url: this.props.url,
    type: 'PUT',
    data: {item: {name: this.refs.itemName.value}},
    success: function() {
      self.setState({edit: false});
      self.props.refreshList();
    }
  });
},
```

React Component Communication

Getting React components to communicate with each other is often one of the most confusing and misunderstood challenges we face in ReactJS.

When you start searching for solutions you will be led to a design pattern called Flux or some of it's common libraries like alt.js, redux, etc...

One thing to keep in mind is that Flux is not a library it is a pattern. In other words you can't really Gem install flux and be on your way. For this lecture we will discuss alternatives to using Flux when components need to communicate with each other.

Methods

The methods for a component to communicate with another component highly depends on how the component needs to communicate.

Options:

1. props
2. callbacks
3. parent container

Props

Props are a great way for a parent to communicate with a child component. We have already seen this approach using props to pass data from List to Item.

Props are by far the most important data feature of React so let’s get a little more practice passing props from parent to child.

Start by modifying the server side to have boards that contain lists that contain items.

rails g controller Boards index --skip-routes --skip-assets --skip-helper

rails g model Board name

rails g model List name percent_complete:float board:belongs_to

config/routes.rb
root ‘boards#index’
resources :boards
resources :lists
resources :items

```
1 class BoardsController < ApplicationController
2   def index
3     @boards = Board.all.order(:created_at)
4   end
5 end
```

Models

```ruby
class Board < ActiveRecord::Base
  has_many :lists, dependent: :destroy
end
```

```ruby
class List < ActiveRecord::Base
  belongs_to :board
  has_many :items, dependent: :destroy
end
```

```ruby
class Item < ActiveRecord::Base
  belongs_to :list
end
```

```ruby
class AddListReferenceToItems < ActiveRecord::Migration
  def change
    add_reference :items, :list, index: true, foreign_key: true
  end
end
```

Make sure to add belongs_to migrations as needed. See above.

bundle exec rake db:migrate

app/views/boards/index.html.erb

```
1 <%= react_component 'Boards', { boards: @boards } %>
```

app/controllers/boards_controller.rb

```
1  class BoardsController < ApplicationController
2    def index
3      @boards = Board.all.order(:created_at)
4    end
5
6    def create
7      board = Board.create(board_params)
8      render json: board
9    end
10
11   private
12     def board_params
13       params.require(:board).permit(:name)
14     end
15 end
```

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


```
var Boards = React.createClass({
  getInitialState: function() {
    return {boards: this.props.boards}
  },

  getDefaultState: function() {
    return {boards: []}
  },

  addBoard: function(e) {
    e.preventDefault();
    var self = this;
    $.ajax({
      url: '/boards',
      type: 'POST',
      data: {board: {name: this.refs.boardName.value}},
      success: function(data) {
        var boards = self.state.boards;
        boards.push(data);
        self.refs.boardName.value = '';
        self.setState({boards: boards});
      }
    });
  },

  removeBoard: function() {
    alert('this should remove the board');
  },

  displayBoards: function() {
    if(this.state.boards.length) {
      var boards = [];
      for(var i = 0; i < this.state.boards.length; i++){
        var board = this.state.boards[i];
        var key = 'board-' + board.id;
        boards.push(<Board key={key} id={board.id} name={board.name} removeBoard={this.removeBoard} />);
      }
      return boards;
    } else {
      return(<h3>No Boards Found Please Add One</h3>);
    }
  },

  render: function() {
    return(<div>
            <form onSubmit={this.addBoard}>
              <div className='input-field'>
                <input type='text' autoFocus='true' placeholder='Add Board' ref='boardName' />
                <button type='submit' className='btn waves-effect'>Add</button>
              </div>
            </form>
            <div className='row'>
              {this.displayBoards()}
            </div>
          </div>);
  }
});
```

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

```
var Board = React.createClass({
  loadBoard: function() {
    alert('this should load the list items');
  },

  render: function() {
    return(<div className='col s4 pointer' onClick={this.loadBoard}>
            <div className='row'>
              <div className='card blue-grey darken-1'>
                <div className='card-content white-text'>
                  <span className='card-title'>{this.props.name}</span>
                </div>
                <div className='card-action'>
                  <a onClick={() => this.props.removeBoard(this.props.id)}>Delete</a>
                </div>
              </div>
            </div>
          </div>);
  }
});
```

Callbacks

To communicate from a child to a parent we can use callback functions.

A callback function was stubbed out earlier when we passed removeBoard={this.removeBoard} to the Board.

In Board.js.jsx on click we call the callback and pass in the ID

Adding ( ) => prevents the callback from being fired when the component is loading.

Since we are passing the ID to the callback make sure and add id={board.id} when creating the board elements.

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

```
removeBoard: function(id) {
  var boards = this.state.boards;
  var self = this;
  $.ajax({
    url: '/boards/' + id,
    type: 'DELETE',
    success: function(data) {
      self.setState({boards: data});
    }
  })
},
```

```
displayBoards: function() {
  if(this.state.boards.length) {
    var boards = [];
    for(var i = 0; i < this.state.boards.length; i++){
      var board = this.state.boards[i];
      var key = 'board-' + board.id;
      boards.push(<Board key={key} toggleBoard={this.toggleBoard} id={board.id} nam
    }
    return boards;
  } else {
    return(<h3>No Boards Found Please Add One</h3>);
  }
},
```

```
class BoardsController < ApplicationController
  def index
    @boards = Board.all.order(:created_at)
  end

  def create
    board = Board.create(board_params)
    render json: board
  end

  def destroy
    Board.find(params[:id]).destroy
    render json: Board.all.order(:created_at)
  end

  private
    def board_params
      params.require(:board).permit(:name)
    end
end
```

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

```
loadBoard: function() {
  this.props.toggleBoard(this.props.id);
},
```

Parent Container

Another way to have items communicate is by keeping the state in the parent container. We will now refactor the app this way.

Rails Controllers

```ruby
class ListsController < ApplicationController
  before_action :board

  def index
    render json: @board.lists.order(:created_at)
  end

  def create
    board = @board.lists.create(list_params)
    render json: board
  end

  private
    def list_params
      params.require(:list).permit(:name)
    end

    def board
      @board = Board.find(params[:id])
    end
end
```

```ruby
class ItemsController < ApplicationController
  before_action :list

  def index
    @items = @list.items.order(:created_at)
  end

  def create
    item = @list.items.create(item_params)
    render json: item
  end

  def update
    item = @list.items.find(params[:id])
    item.update(item_params)
    render json: item
  end

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

  def destroy
    @list.items.find(params[:id]).destroy
    head :ok
  end

  private

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

  def list
    @list = List.find(params[:list_id])
  end
end
```

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

Pass list_id to all of the ajax calls and pass listId to the Item component

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

```
submitItem: function(e) {
  e.preventDefault();
  var self = this;
  $.ajax({
    url: '/items',
    type: 'POST',
    data: {list_id: this.props.id, item: {name: this.state.itemName}},
    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});
    }
  });
},
```

```
displayItems: function() {
  if(this.state.items.length) {
    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 listId={this.props.id} refreshList={this.refreshList} key={key}
    }
    return items;
  } else {
    return(<h5 className='center'>No Items To Display. Please Create One.</h5>);
  }
},
```

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

Rename Render to showBoards and create a new render function. Add the toggleBoard and showLists functions.

```
toggleBoard: function(id) {
  this.setState({ boardId: id, listView: !this.state.listView });
},

showLists: function(id) {
  return(<Lists showBoards={this.toggleBoard} boardId={this.state.boardId} />);
},

showBoards: function() {
  return(<div>
          <form onSubmit={this.addBoard}>
            <div className='input-field'>
              <input auotFocus='true' placeholder='add board' type='text' ref='boardName' />
              <button className='btn waves-effect' type='submit'>Add</button>
            </div>
          </form>
          <div onClick={this.showBoard} className='row'>
            {this.boards()}
          </div>
         </div>);
},

render: function() {
  if(this.state.listView)
    return this.showLists();
  else
    return this.showBoards();
}
```

../javascripts/components/Lists.js.jsx

```
var Lists = React.createClass({
  getInitialState: function() {
    return {lists: []}
  },

  componentDidMount: function() {
    this.fetchLists();
  },

  fetchLists: function() {
    var self = this;
    $.ajax({
      url: '/lists',
      data: {id: this.props.boardId},
      success: function(data) {
        self.setState({lists: data});
      }
    });
  },

  showAddForm: function() {
    this.setState({showAdd: !this.state.showAdd});
  },

  addListForm: function() {
    if(this.state.showAdd) {
      return(<div>
              <form onSubmit={this.submitList}>
                <div className='input-field'>
                  <input autoFocus='true' placeholder='List Name' type='text' ref="listName" />
                  <button type='submit' className='btn waves-effect'>Add</button>
                </div>
              </form>
            </div>);
    }
  },

  submitList: function(e) {
    e.preventDefault();
    var self = this;
    $.ajax({
      url: '/lists',
      type: 'POST',
      data: {id: this.props.boardId, list: {name: this.refs.listName.value}},
      success: function(data) {
        var lists = self.state.lists;
        lists.push(data);
        self.setState({lists: lists, showAdd: false});
      }
    });
  },
```

```
  displayLists: function() {
    var lists = [];
    for(var i = 0; i < this.state.lists.length; i++){
      var list = this.state.lists[i];
      var key = 'list-' + list.id;
      lists.push(<List key={key} id={list.id} name={list.name} />);
    }
    return lists;
  },

  render: function() {
    return(<div>
            <a className='waves-effect waves-light btn' onClick={this.props.showBoards}>Boards
            <a className='waves-effect waves-light btn' onClick={this.showAddForm}>Add List</a
            {this.addListForm()}
            <hr />
            <div className='row'>
            {this.displayLists()}
            </div>
          </div>);
  }
);
```

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

Finally add list_id to all of the ajax calls in Item

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

updateItem: function(e) {
  e.preventDefault();
  var self = this;
  $.ajax({
    url: this.props.url,
    type: 'PUT',
    data: {list_id: this.props.listId, item: {name: this.refs.itemName.value}},
    success: function() {
      self.setState({edit: false});
      self.props.refreshList();
    }
  });
},
```

```
deleteItem: function(e) {
  e.preventDefault();
  var self = this;
  $.ajax({
    url: this.props.url,
    type: 'DELETE',
    data: {list_id: this.props.listId},
    success: function(data) {
      self.props.refreshList();
    }
  });
},
```

app/controllers/items_controller.rb

remove the render json call. We will be using jbuilder to return the correct json with the item url in it

```ruby
class ItemsController < ApplicationController
  before_action :list

  def index
    @items = @list.items.order(:created_at)
  end

  def create
    @item = @list.items.create(item_params)
  end

  def update
    item = @list.items.find(params[:id])
    item.update(item_params)
    render json: @item
  end

  def destroy
    @list.items.find(params[:id]).destroy
    head :ok
  end

  private
    def list
      @list = List.find(params[:list_id])
    end

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

app/views/items/create.json.jbuilder

```
json.(@item, :id, :name, :complete)
json.url item_url(@item)
```

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

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

add url: data.url to the items.push call