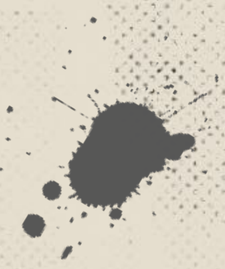
DAY 2: ADVANCED



React

SECTION No.

0

WHAT ARE WE COVERING TODAY

WHAT ARE WE COVERING TODAY

1. Redux
2. Redux Dev Tools
3. Routing
4. CSS & SASS
5. Bootstrap
6. Potpourri

LET'S GET BUILDING

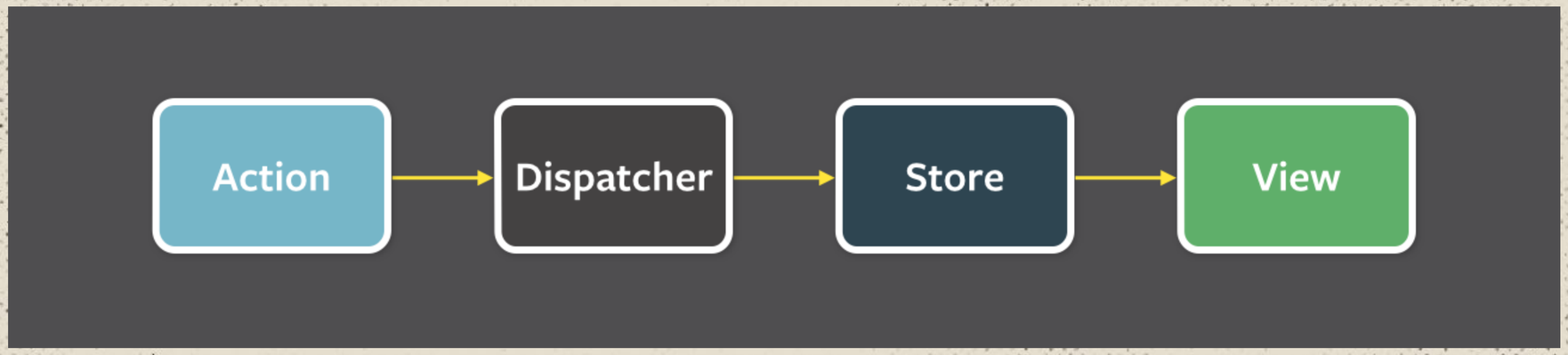
CHAPTER No.

REDUX

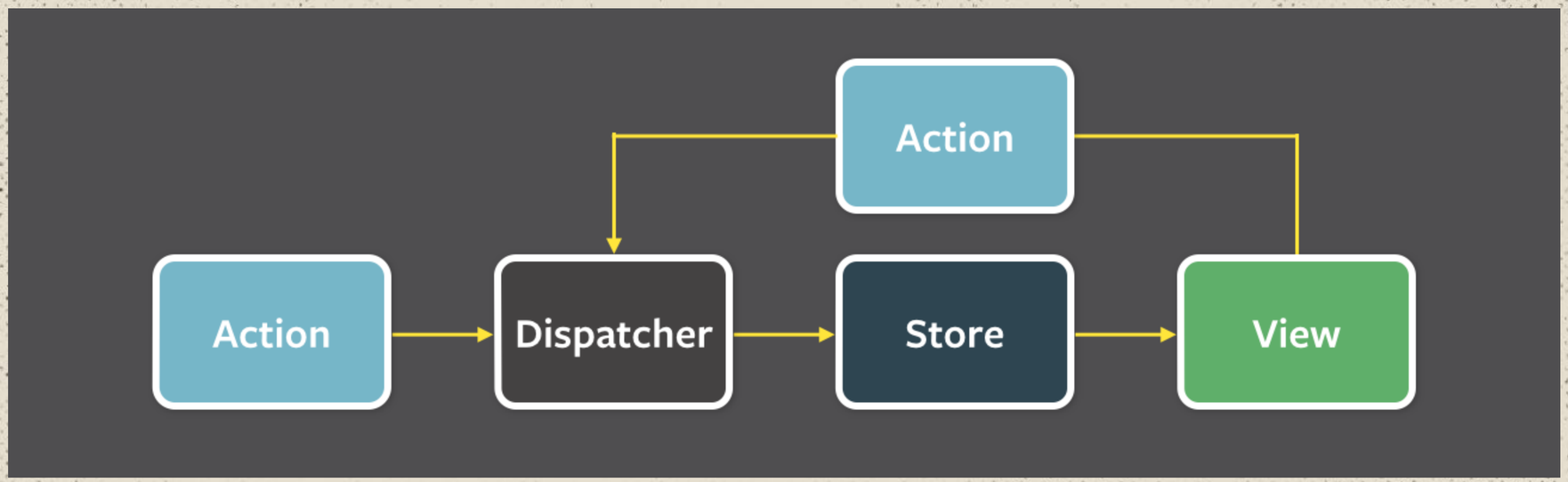
THIS TAKES SOME TIME ...

LET'S REVIEW

UNIDIRECTIONAL DATA FLOW



UNIDIRECTIONAL DATA FLOW



THIS PATTERN IS KNOWN AS FLUX

REDUX

- Redux is a twist on Flux
- Redux's aim is to create a "predictable state container"
- It does this by abstracting out the ideas of actions and stores even farther
- It removes the concept of multiple stores, reducing the application state to a single store
- Interesting note: it's not just for React, but can also be used with Angular

REDUX – ACTIONS

- Actions always return an object
- By our convention, that object will have two properties: type and payload
- Think of actions as the “data retrievers”, “calculators”, etc …
- Action files also by convention will export const strings for all possible types

REDUX – ACTIONS

```
export const LOAD_SUCCESS = 'LOAD_SUCCESS';
export const LOAD_FAILURE = 'LOAD_FAILURE';

export const loadData = () =>
{
  type: LOAD_SUCCESS,
  payload: [ { name: 'foo' }, { name: 'bar' } ],
};
```

REDUX – REDUCERS

- Each reducer is a function that returns a single variable (can be an array object, string, etc)
- Each reducer needs to have the signature (state, action) - state will be the current state
- Redux combines these results into a single application state
- Reducers respond to actions
- Every reducer is given a shot to react to every action
- If the reducer decides to change, it needs to emit a NEW value, NOT modify the existing value - this is what triggers the notice to views to change.
- If the reducer does not change, it needs to return the existing state

REDUX – REDUCERS

```
import { LOAD_SUCCESS } from 'actions';

export default (state = [], action) => {
  switch (action.type) {
    case LOAD_SUCCESS:
      return action.payload;

    default:
      return state;
  }
};
```

REDUX – REDUCERS

```
import { combineReducers } from 'redux';
import { routerReducer } from 'react-router-redux';

import data from './data_reducer';

const rootReducer = combineReducers({
  routing: routerReducer,
  data,
});

export default rootReducer;
```

REDUX – COMPONENTS

★ ★ ★ ★ ★ ★ ★ ★ ★

- React-Redux is the package that connects the reducers to our components
- import { connect } from ‘react-redux’
- The connect method using currying (returning a function from a function)
- The first function takes two functions, the first maps app state to props, the second maps actions to props
- The returned function (second in the curry chain) takes the component to be wrapped
- There are some syntactic sugar things we can do …

REDUX – COMPONENTS

```
const mapStateToProps = ({ data }) => ({ data });

export default connect(mapStateToProps, { loadData })(MyComponent);
```

BREATHE ... ALMOST THERE

REDUX – MIDDLEWARE

- By definition, Redux requires an action to return an object, so how do you handle async or complicated logic
- This is where middleware comes in, and why we have the “weird” passing of actions to the connect method
- When one of those actions is invoked by the component, you can attach middleware to intercept the result of the action

REDUX – THUNK

- Thunk allows us to return a function from an action
- It will pass us a dispatch function and a getState function
- When we are ready to actually return the action result, we pass it to the dispatch function, and it is processed like normal

REDUX – THUNK

```
export const loadData = () =>
  (dispatch) =>
    fetch('http://www.google.com')
      .then((response) => {
        dispatch({
        type: LOAD_SUCCESS,
        payload: response.data,
      });
    });
```

REDUX – PROMISE

- Redux Promise lets you return a promise from an action
- When the promise is resolved, the value passed to resolve is supplied as an action to the reducers

REDUX – THUNK

```
export const loadData = () =>
  fetch('http://www.google.com')
    .then((response) => {
      dispatch({
      type: LOAD_SUCCESS,
      payload: response.data,
    });
```

LAB / DEMO

CHAPTER No.

REDUX DEV TOOLS

REDUX – DEV TOOLS

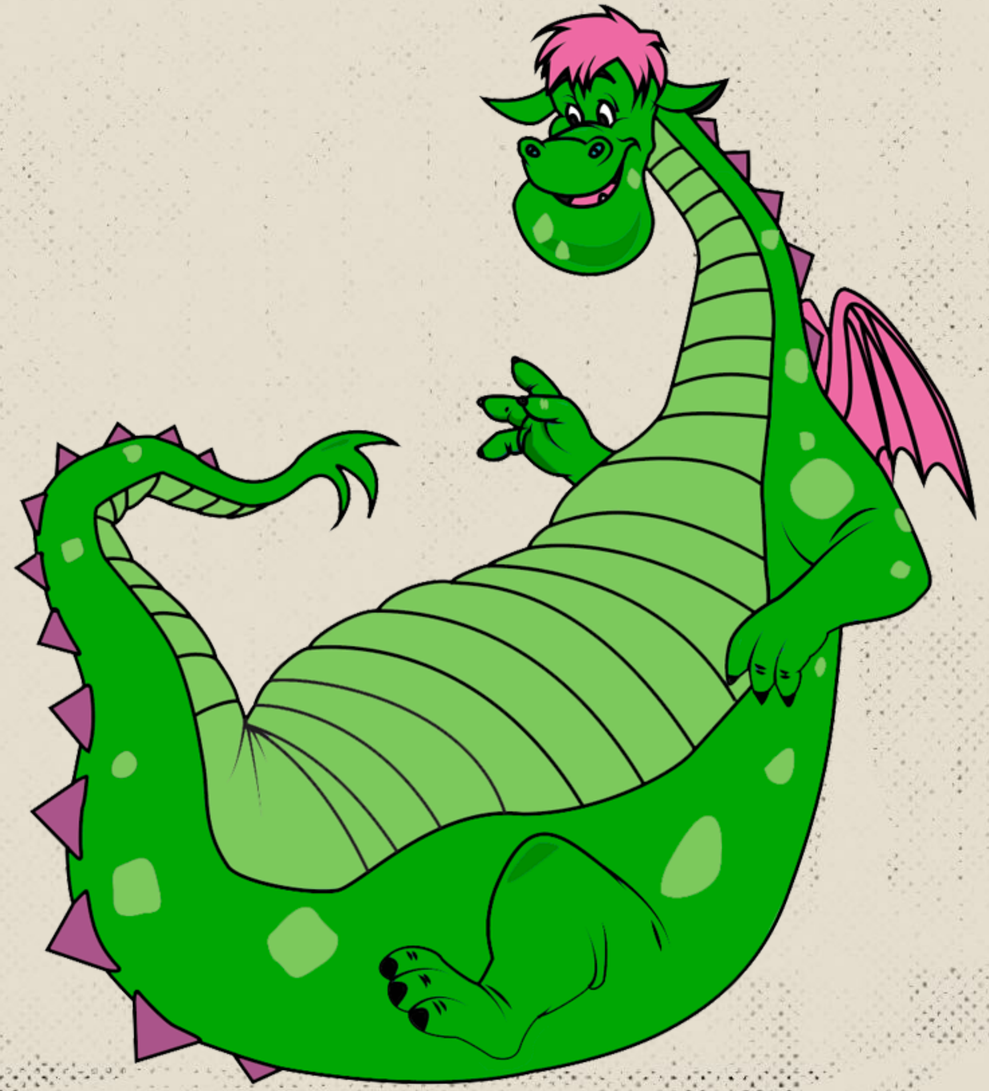
★ ★ ★ ★ ★ ★ ★ ★ ★

- Take a second to think about the pattern we have in redux …
- All state change is a series of actions type > resultant state pairs
- If we captured each of those somewhere, we would could replay the use of the app
- … thus redux dev tools

REDUX – DEV TOOLS

- At its simplest form, you can simply install a chrome extension (search the chrome store)
- We have support already defined in the template creation of the store (yay middleware)

LAB / DEMO



CHAPTER No. 3

ROUTING

ROUTING

- We use the React-Router plugin for routing
- It's typically the normal declarative path / component route
- Although you can specify a function that will evaluate state to return a component
- Allows nested routes, matched internal component will be passed to this.props.children
- IndexRoute will be used for the root route of a Route

ROUTING

```
<Route path="/" component={App}>
    <IndexRoute component={HomePage}/>
    <Route path="*" component={NotFoundPage}/>
 </Route>
```

ROUTING

- You can capture query string arguments by naming them with :<name> in the route.
- They will be found on this.props.params.<name>

DEMO / LAB

CHAPTER No.

CSS & SASS

CSS & SASS

★ ★ ★ ★ ★ ★ ★ ★ ★

- Although not required, SASS is probably the most common CSS “language” in the React community
- SASS is compiled down to CSS by Webpack via the node-sass plugin
- We bring in the root scss file in the index.js simply by importing it

CSS & SASS

★ ★ ★ ★ ★ ★ ★ ★ ★

- We won't go too deep in SASS today (that's another lecture all in itself), but we will touch on some highlights
- You can declare variables using $varname, these are often used for colors and standard sizes
- Just like with js, we can import files so we can logically separate our css.
- You typically prefix the name of your non-index files an _, but you do not need to include it in the imports statement

CSS & SASS

★ ★ ★ ★ ★ ★ ★ ★ ★

- SASS also supports mixins, inheritance, nesting, and operators
- You also can just default down to normal CSS, its mixes and matches just fine

DEMO / LAB

CHAPTER No. 5

BOOTSTRAP

BOOTSTRAP

★ ★ ★ ★ ★ ★ ★ ★ ★

- Bootstrap is a very popular CSS framework
- There are a ton of CSS frameworks out there (seemingly more than JS front end frameworks)
- Bootstrap is currently my default (although somewhat hesitantly at times)
- We use the React-Bootstrap package to map bootstrap css / js into React.
- Its important to start with the documentation for the react package rather than the root bootstrap docs

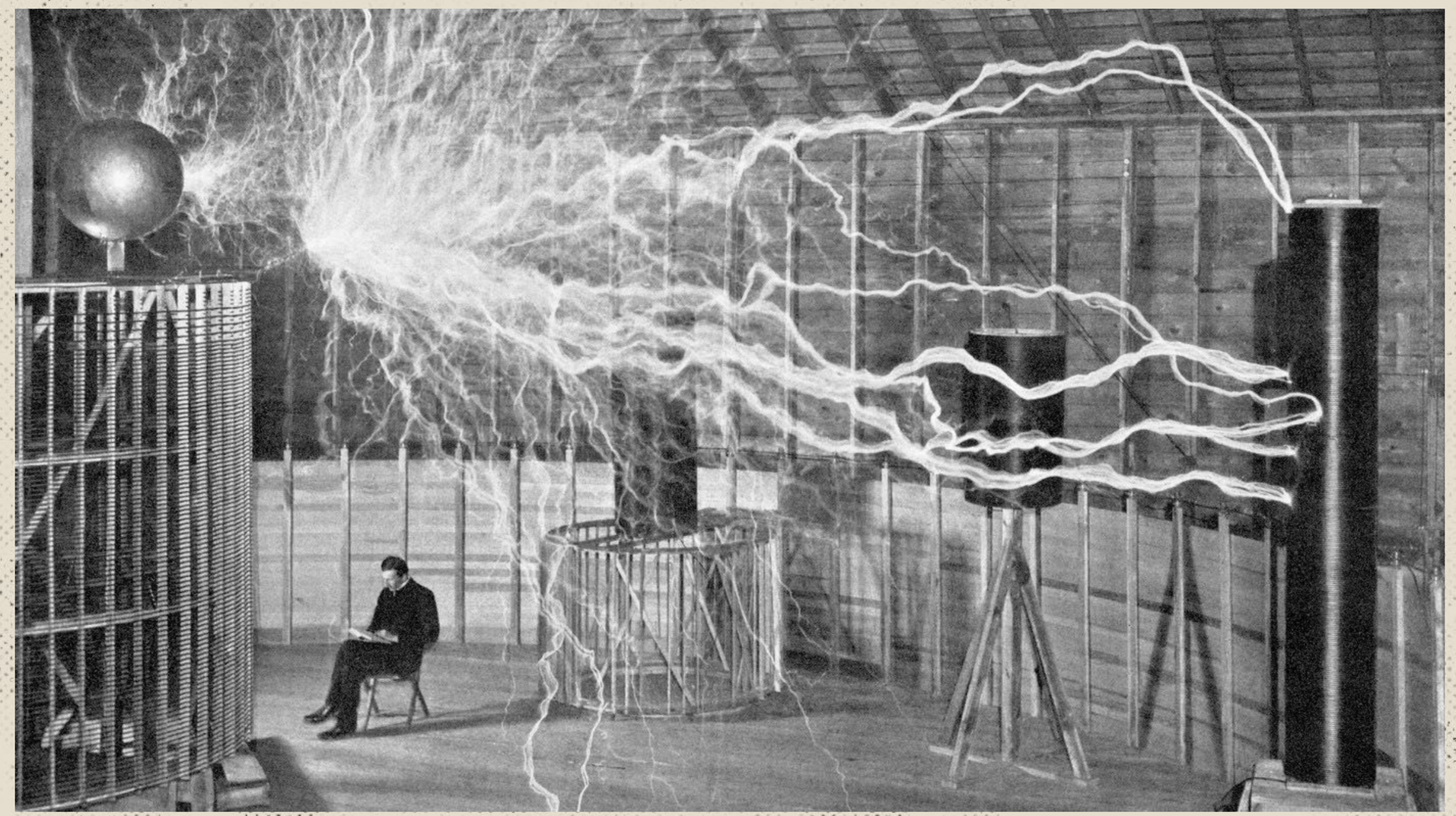
BOOTSTRAP – GRID

- Probably one of the most used features of bootstrap is the Grid.
- By default it is a 12 column grid (meaning 12 evenly spaced columns)
- These columns are contained with Rows, which is typically constrained within a root Grid element

BOOTSTRAP – COMPONENTS

- In additions there are ton of other components:
- Buttons, Overlays, Modals, Navigation, Tables, Forms, Media Content, and more ...

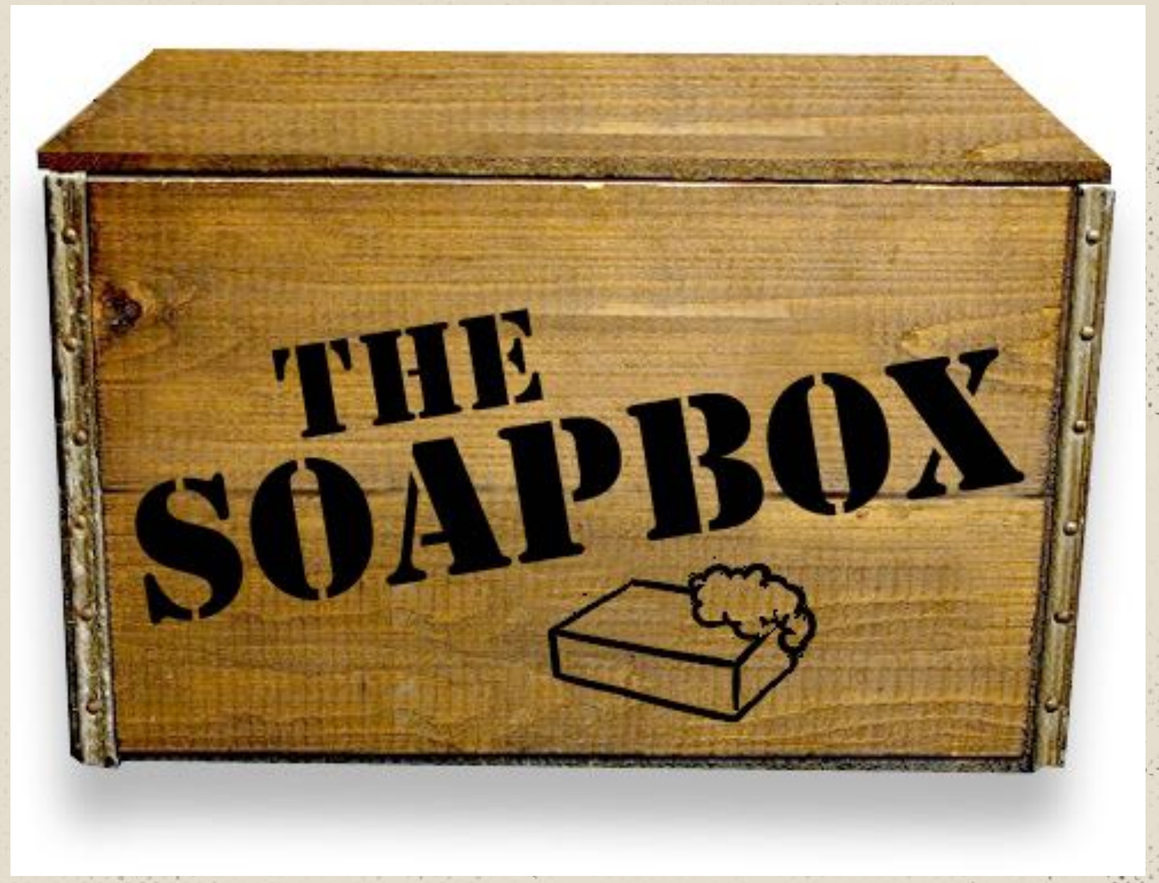
DEMO / LAB



CHAPTER No. 6

POTPOURRI

POTPOURRI



SOLID

SECTION No.

A

SOLID

★ ★ ★ ★ ★ ★ ★ ★ ★

- Single responsibility principle
- Open/closed principle
- Liskov substitution principle
- Interface segregation principle
- Dependency inversion principle

SECTION No.

B

COMPOSITION IS MORE THAN COMPONENTS

COMPOSITION

★ ★ ★ ★ ★ ★ ★ ★ ★

- You should strive to create clean and simple components
- Components though, should only contain the logic for displaying the information & accepting the user input
- Abstract out logic for business logic, validation, etc into their own files
- Strive for these to be simple, single use functions
- These are valid “bricks” the same as components

SECTION No.

C

DIRECTORY STRATEGIES

DIRECTORY STRATEGIES

- Like with every project in the world, there is not a hard solid “right” way.
- That said there are a couple of common patterns, and we will follow 1 of 2
- Probably the most popular in the community is: Type > Feature > File
- The underdog is: Feature > Type > File

SECTION No.

D

OTHER THINGS TO LEARN

OTHER THINGS TO LEARN

★ ★ ★ ★ ★ ★ ★ ★ ★

- Immutable
- Webpack 2.0
- Unit Testing
- Redux Form
- Redux Observable
- Higher Order Functions
- React Native


FINISH