REACT, UNIDIRECTIONAL DATA FLOW, AND YOU

DOUG NEINER @DOUGNEINER
ABOUT ME

- **Doug Neiner**  
  @dougneiner
- Married with 5 kids
- Work from home in rural Iowa, USA
- I enjoy making and building stuff (cooking, woodworking, coding)
- Front End Web Developer at LeanKit
WHAT IS REACT?

• React is a library for building user interfaces in JavaScript that are performant and predictable.


• Often described as the V in MVC…
WHAT IS FLUX?

• Flux is an architectural guideline for building web applications that pairs exceptionally well with React.js

• [https://facebook.github.io/flux/](https://facebook.github.io/flux/)

• Heavily focused on predicability

• Replaces MVC with an alternative architecture model

(The actual code part of Flux is simply the Dispatcher, which we'll discuss in a bit.)
MY JOURNEY TO FLUX

- jQuery + Widgets
- Backbone with Underscore Templates
- Backbone with Two-Way Binding
- Knockout
- React *(since May 2014)*
- React + Flux *(since September 2014)*
GOALS

• Understand the guiding principles behind Flux

• Understand what goes where in a React + Flux app

• See how to test React + Flux

• Gain tips and tricks for tackling React + Flux applications
LEARNING EXAMPLE

MAIL APPLICATION

Message One
Lorem ipsum dolor sit amet, consectetur adipiscing...

Message Two
Lorem ipsum dolor sit amet, consectetur adipiscing...

Message Three
Lorem ipsum dolor sit amet, consectetur adipiscing...

Message Four
Lorem ipsum dolor sit amet, consectetur adipiscing...

Message Five
Lorem ipsum dolor sit amet, consectetur adipiscing...

Message One

From: Doug Neiner

Sent: Feb 11, 2015 @ 10:26AM

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cumque, molestiae in inventore quae ratur explicabo a esse commodi quibusdam tempor a dolore. Deserunt vero debitis a, officia in itaque incident voluptatem fugiat.
THE HIGH POINTS

• Not a step-by-step tutorial

• Code is open source and available on Github: https://github.com/LeanKit-Labs/lux-mail-example
WHAT DO WE NEED?

FEATURES

- View list of messages
- View full text of selected message
- Compose new message
- Reply, Forward, Archive active message
- Check for new messages
- Distinguish between unread/read messages
BUILDING WITH REACT

- Static HTML + CSS
- Static React Components with props
- Dynamic React Components with state
REACT COMPONENTS
QUICK INTERFACES
WITH REACT

1. Copy all interface HTML into a single React component’s render call. Replace `class` with `className` and `for` with `htmlFor`

2. Break into smaller components, with each component still rendering static HTML.

3. Start to replace static HTML with prop values (Use `getDefaultProps` to help)

4. Keep as many components as you can to strictly using props
HOW DO WE BUILD IT?

COMPONENTS? MODELS? AJAX?
ANY CHANGE STARTS WITH AN ACTION
FLUX ARCHITECTURE

UNIDIRECTIONAL FLOW

From https://facebook.github.io/flux/docs/overview.html
ACTIONS

- Created by almost any part of your application
  - User Interaction
  - Results of AJAX requests
  - Timers
  - Web Socket events
- Globally unique actions names
- Once the cycle starts, a second action should not be created until the first action cycle has completed
WHAT DO WE NEED?

ACTIONS

- Load Messages
- Load User
- Select Message
- Start Reply, Start Forward
- Archive Message
- Start New Message
- Mark Message as Read
<table>
<thead>
<tr>
<th>ACTIONS</th>
<th>RESULTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Messages</td>
<td>Display all messages, first one selected</td>
</tr>
<tr>
<td>Load User</td>
<td>Populate user menu</td>
</tr>
<tr>
<td>Select Message</td>
<td>Highlight message, open message</td>
</tr>
<tr>
<td>Start Reply, Start Forward</td>
<td>Open compose window</td>
</tr>
<tr>
<td>Archive Message</td>
<td>Remove from list, close viewer</td>
</tr>
<tr>
<td>Start New Message</td>
<td>Open compose window</td>
</tr>
<tr>
<td>Mark Message as Read</td>
<td>Remove blue mark from list view</td>
</tr>
</tbody>
</table>

**Diagram: Action Flow**

1. **ACTION**
2. **DISPATCHER**
3. **STORE**
4. **VIEW**

18
**ACTIONS**

- Load Messages
- Messages Loaded
- Load User
- User Loaded
- Select Message
- Start Reply, Start Forward
- Archive Message
- Start New Message
- Mark Message as Read

**RESULTS**

- Data requested from server
- Display all messages, first one selected
- User data requested from server
- Populate user menu
- Highlight message, open message
- Open compose window
- Remove from list, close viewer
- Open compose window
- Remove blue mark from list view
**ACTIONS**

- Load Messages
- Messages Loaded
- Load User
- User Loaded
- Select Message
- Start Reply, Start Forward
- Archive Message
- Start New Message
- Mark Message as Read

**RESULTS**

- Show “Loading messages…”
- Display all messages, first one selected
- User data requested from server
- Populate user menu
- Highlight message, open message
- Open compose window
- Remove from list, close viewer
- Open compose window
- Remove blue mark from list view
Dispatcher

- Coordinates relaying the actions to the stores that are interested in the actions.
- Ensures the stores action handlers are executed in the correct order.
- Handled entirely by the library you are using (Flux, lux.js, etc)
- It is important, but you don’t have to think about it.
• **Application state lives in stores**, included content and state needed to render the views.

• Each store manages data and actions for a **single related domain of information**

• Stores can be dependent on other stores

• Stores should expose helper methods to return the data for the views
Stores

- The state of a store *does not change* outside of an action/dispatch cycle.
- Stores can choose to handle *any of the actions* being dispatched.
- Multiple stores can participate in the *same* action.
• React.js components
• Get their initial state from the stores
• Get notified when the stores change, and can optionally `setState` on themselves with new data.
• Can kick off the action/dispatch cycle. As a general rule, do not kick off the action dispatch cycle in response to a `setState` call or `render` call.
UNIDIRECTIONAL FLOW

ACTION → DISPATCHER → STORE → VIEW
BUILDING WITH REACT + FLUX

- Static HTML + CSS
- Static React Components with props
- Dynamic React Components with state
- State moved to Flux Stores
- User feedback added to React, triggering Flux actions
TIME FOR CODE

• Overview of the example mail app

• Breaking larger React components into smaller components

• Unit testing Stores and React Components

• Triggering actions from other pieces in your application

• Using non-React DOM widgets with React
LINKS


• lux-mail-example: http://bit.ly/luxmail

• LeanKit: http://leankit.com
THANK YOU

doug@dougneiner.com
@dougneiner on Twitter
dcneiner on Github
http://code.dougneiner.com