ACSU x AppDev React Workshop

Joe Antonakakis

Intros

Who am I?

- My name is Joe
- Senior in CS @ Cornell
- Learned what React was ~2 years ago
- Actually started writing a lot of it ~9 months ago
- Worked with it @ Facebook this summer
- Have never looked back since

Who are you?

- I’m assuming you have a basic grasp on HTML / CSS / JS
- Maybe you’ve struggled to learn frontend
- Maybe you’ve used things like jQuery, Ember.js, Backbone.js, Angular but are not completely satisfied
- Maybe you’re just curious
- Maybe you want to make something cool
- Maybe you’re just here for food

In any case, hopefully I address your interests

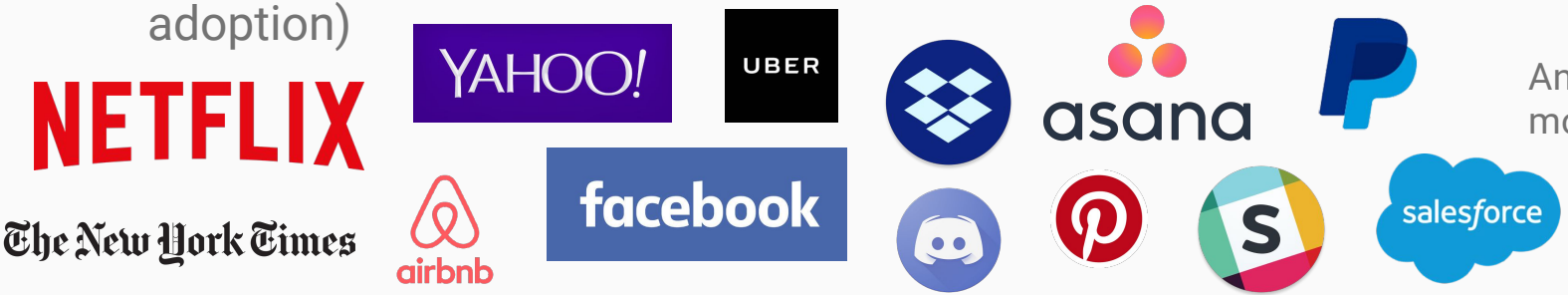
What is React? Why should I know it?

What is React?

- “A JavaScript library for building user interfaces” - React website
- Pitched features:
  1. Declarative: define simple views / features for each part of your app, with state-changes that make the app predictable and easy to debug
  2. Component-Based: encapsulate each part of your UI as a component, separation-of-concerns. No DOM state.
  3. Learn Once, Write Anywhere: can write React for UIs on multiple platforms (web, mobile, TV, etc.)
- Sick! Why is all this good, though?

Why should I know React?

- Scales to large products very quickly
- Ideologies behind it are powerful software engineering paradigms
- Makes working on an app with multiple people very easy
- Can easily be subbed in for your current UI solution gradually
- Actively changing frontend engineering as we speak (widespread adoption)



React Essentials

Physical DOM vs. Virtual DOM

- DOM - Document Object Model (entire idea of layout in HTML)
- Traditional JS edits physical DOM
  - Excessive writes → poor performance
  - Involves interacting directly with HTML, which couples this format to your JS functions (state is all over the place)
- Virtual DOM in React → JS has its own, in-memory idea of the DOM
- Allows shift in computation towards React, which optimizes how updates to DOM happen (no hand-updating individual DOM nodes, selecting DOM nodes, etc. like one would do in traditional JS, jQuery, etc.)

JSX - JavaScript XML (language of React)

Expression of React via XML means:

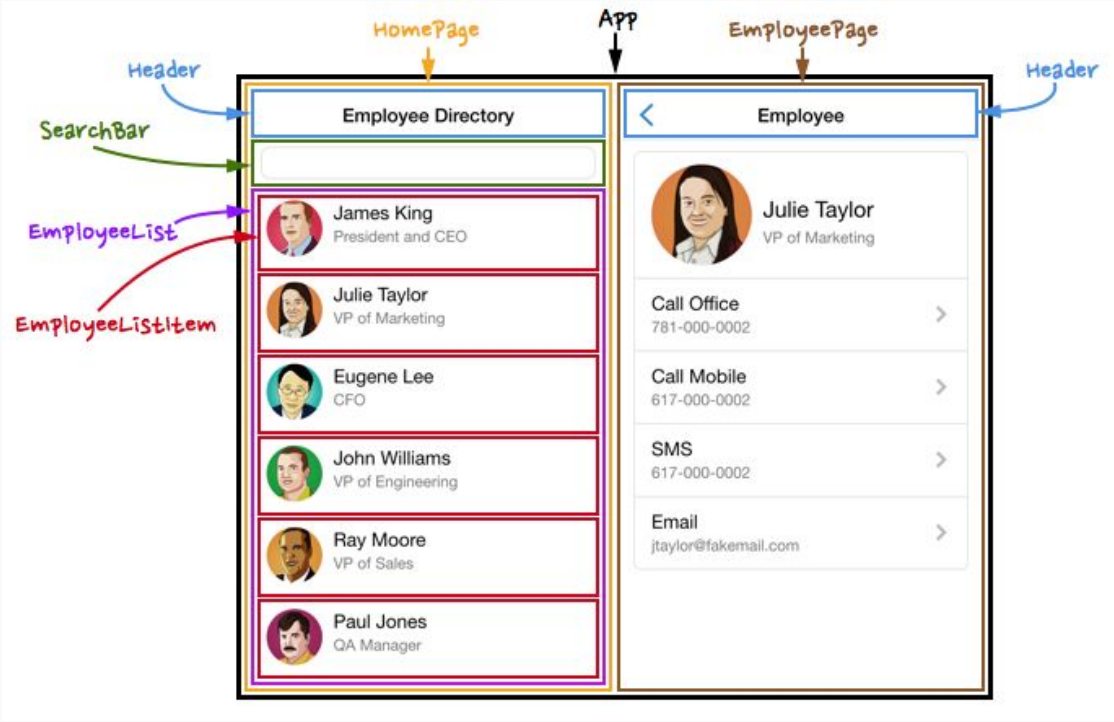
```
class MyList extends React.Component {
  render () {
    return (
      <div>
        <ListElement>My First Element</ListElement>
        <ListElement>My Second Element</ListElement>
      </div>
    );
  }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

Example of traditional XML, credit W3Schools

Components

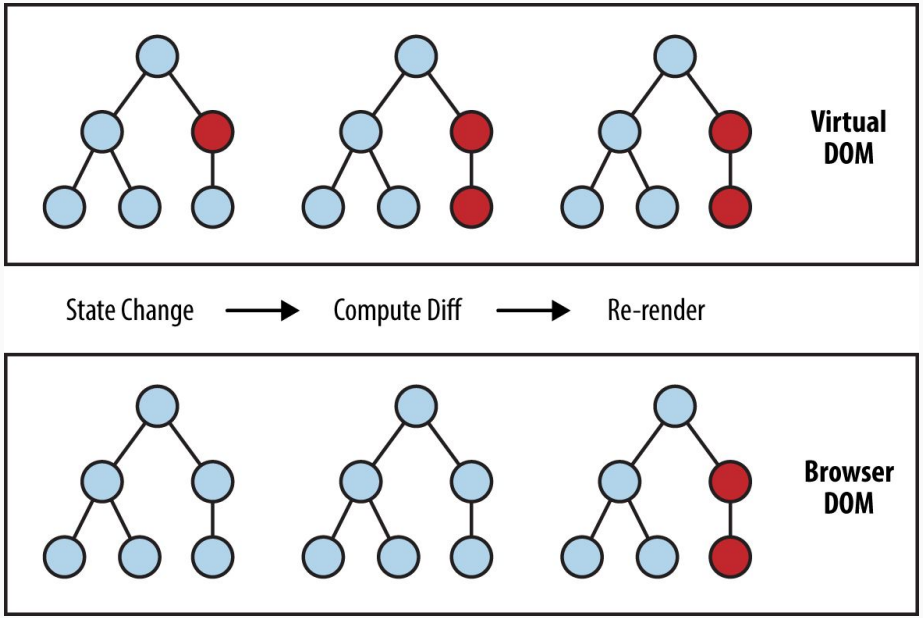
- Isolated, logical units
- Generally-accepted idea: 2 types, *dumb* and *smart*
- Dumb - just present, don't have any state-augmenting functionality, might attach functions to event handlers (e.g. *EmployeeListItem*)
- Smart - keep track of state, expose functions to dumb components for state-changing operations (e.g. *HomePage*)



Breaking up a mobile app into components, credit Christophe Coenraets

State & Props

- 2 types of data: *State* and *Props*
- *Props* are passed to the component to display or use for some logical operation (do not change)
- *State* is declared and maintained by the component
- Key function *setState* involves updating state → change in virtual DOM → update actual DOM (the UI itself) via diff-check algo



State change → change in UI via DOM diff-check, credit Learning React Native by Bonnie Eisenman

There’s more to the story, but those are the dead-simple basics

Coding Workshop

Starter Code

- Found here: https://github.com/Jma353/acsu-react
- README explains requirements (primarily Node + NPM)

- 2-3 minutes to let everyone get set up
- If you want to casually watch and absorb, equivalently good

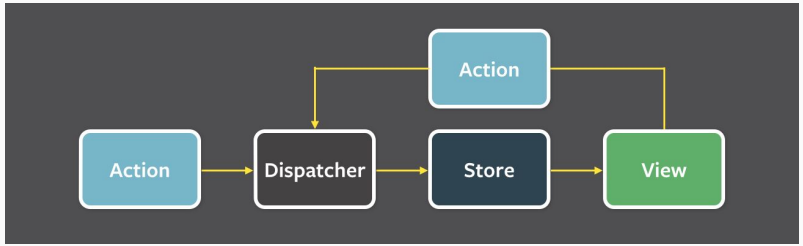
Comments

- Project uses the following JS technologies for building / dev-flow
  - Webpack - module bundler (for JS / CSS / assets / etc.)
  - Babel - transpiles “next-gen” JS (in this case, ES6) to JS that can run in most browsers
  - Flow - static type-checker for JS written by Facebook
  - ES6 syntax, modern JS with things like arrow-functions, classes, etc.
- Will explain as we code

Bonus - Flux Paradigm

Data in One Direction

- Saw that data moved in generally one direction, augmented by some sort of event
- Wouldn't it be great if we had an organized approach to state maintenance?
- Flux idea (credit Flux website):



Super High-Level Technical Details

- State is in the store, which is a centralized place for business-logic
  - As a result, *unit and integration testable* (wait, frontend engineers write tests???)
- Actions define state-changes of the UI (network requests for data, global error messages, etc.)
- Dispatch is centralized place where data flows through the application; can be used to manipulate the order in which updates are applied and to achieve advanced behavior that is inaccessible when state is not centralized

Libraries

https://github.com/facebook/flux - the original

https://redux.js.org/ - “state container for JavaScript” which implements the Flux paradigm and has additional benefits

Flux Paradigm + React = well-engineered UIs

Thanks!

ACSU, AppDev, and everyone who came today!

Feel free to contact me:
jma353@cornell.edu
jma353 (Github username)

What is a presentation without a stock photo of some metropolitan scenery? →