REACT 101

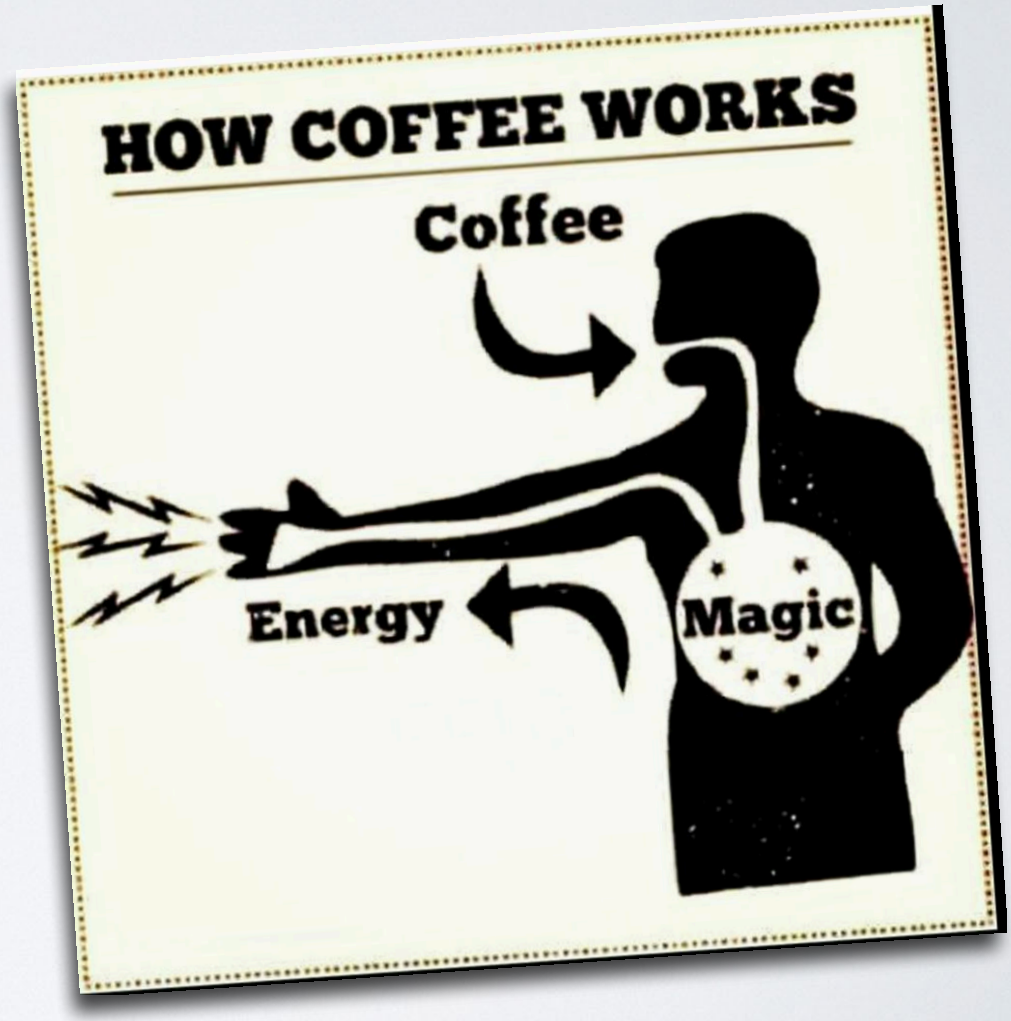
Let's build our first React.js app!

YOUR HOST MIKE

- Former band director
- Loves learning new stuff
- Writes React all day errday
- Four kids at home
- The Iron Yard grad and evangelist

SECRETS TO GOOD CODE

- Coffee
- Code Reviews
- Ask for help
- Even more coffee



SETTING UP A REACT APP

NEW, EASY WAY

- Create-React-App - from the team at FB
- "You'll need to have Node >= 4 on your machine"
- Go to the terminal/command line and do this:

```
➜ npm install -g create-react-app
➜ create-react-app dojo-app
➜ cd dojo-app
```

- We will come back to this in a few minutes

SOME BACKGROUND

WHAT IS REACTJS?

- React is a library used to craft modern UI
  - It creates views for the front-end in web, client and native applications
- Uses small compatible components that only focus on rendering a view
- Moves business logic out of the DOM and improves our app's performance, maintainability, modularity, and readability

SOME HISTORY

- First used by Facebook in 2011, then Instagram in 2012
- Went open source in May 2013
- Currently 64K ☆ on Github, 2nd highest
- Over 8K commits from almost 1000 contributors!

- React was born out of Facebook's frustration w/ traditional MVC model and other 2-way data flow frameworks
- Re-rendering something meant re-rendering a lot…or maybe everything
- Think about re-drawing a “Like” badge on Facebook…don’t need to re-draw every pixel on the page

REACT IN MVC

- React is only a library, not a full framework
- Could be considered the "Views" layer from the traditional MVC
- Use React agnostically throughout your stack. It's role is just to use data to render a UI.
- No real "data model" built in - can use other frameworks alongside react

TEXT EDITOR

- Open up the entire project in your text editor of choice
- Many use Sublime Text 3
- I use Atom (type `atom .`)
- Both are free, though you *should* buy a license for ST 😉

BACK TO THE APP

In the terminal type:

```
yarn start
```

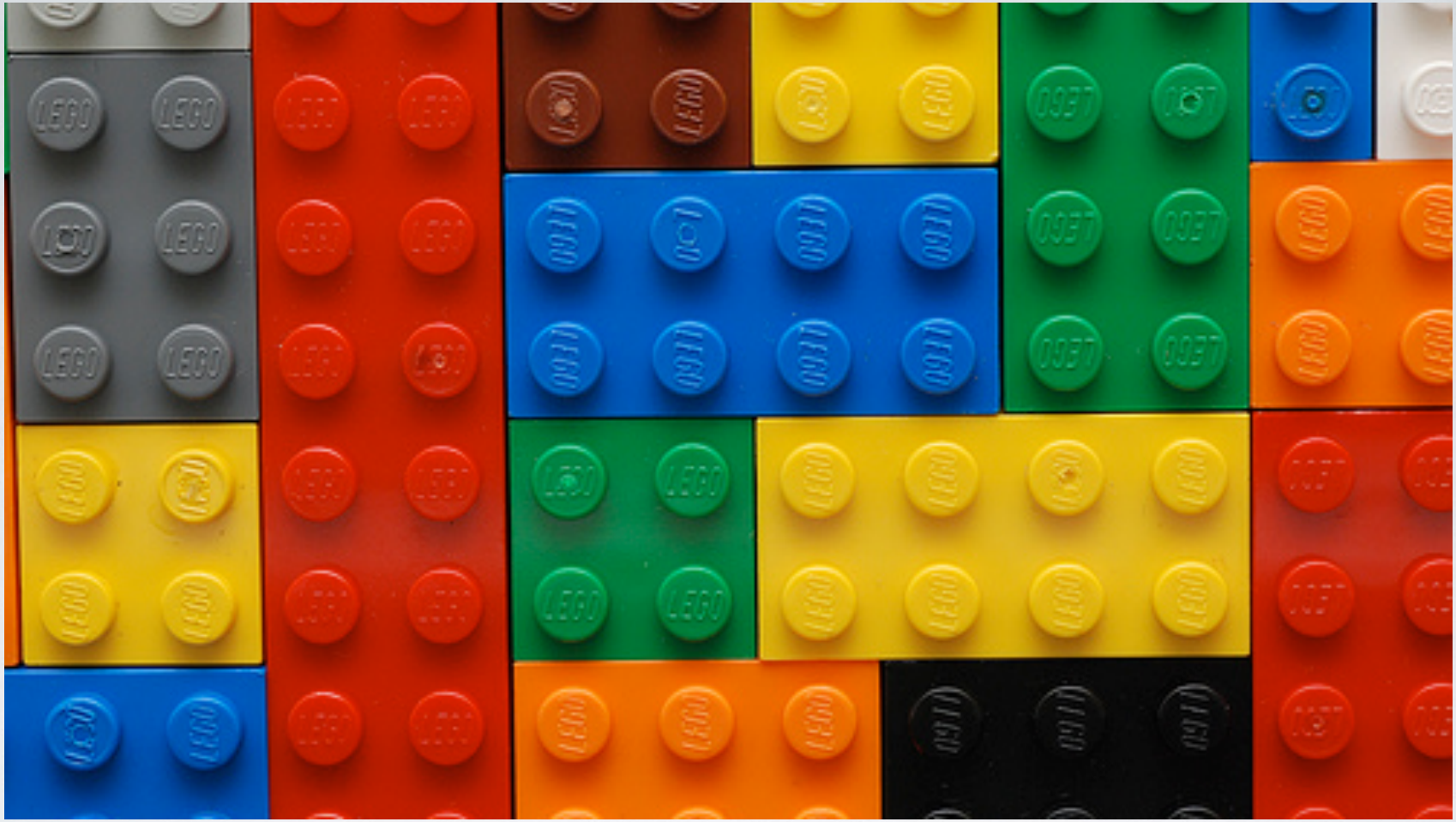
Structure

```
dojo-app/
  README.md
  yarn.lock
  node_modules/
  package.json
  .gitignore
  public/
    favicon.ico
    index.html
  src/
     App.css
     App.js
     App.test.js
     index.css
     index.js
     logo.svg
```

Structure

Look at both `App.js` and `index.js`

See if you can figure out what is happening in there!



COMPONENTS

CONSIDER THIS

- Let's look at Klyde Warren Park website
- Notice how some parts have a similar structure
- Take a second to consider the visual "components" that make up the page
- Do you see "nested" components? Are there some components that could have the same structure?

COMPONENTS

- Components are the basic unit in React
- They take in data and produce UI

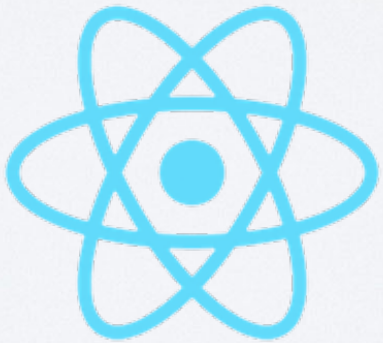
BLURRED LINES

- React blurs lines between our normal HTML/CSS/JS
- Now we organize an app around small, reusable components that define their own content, presentation, and behavior
- If a component is getting too big, separate the different concerns into a new component

LET'S CODE

MY FIRST COMPONENT

Let's make our own "Hello" component



BREAK IT DOWN

`class Hello`

- This is the component we are building out
- In this example, we are creating a "Hello" component

`extends Component`

- Our component definition is being built on top of this piece of the React library

`render()`

- Every component has "lifecycle methods," most of which are optional
- A component must always have a render method.
- This is what generates the **Virtual DOM** from earlier

`export default Hello`

- This exposes the `Hello` class to other files which may need to import it
- Only one default allowed per file
- This helps ensure that other files are bringing in the correctly named import, when needed

Update `index.js`

- Remember that we named a default export in App.js
- So we should keep that name consistent when it is used in other places

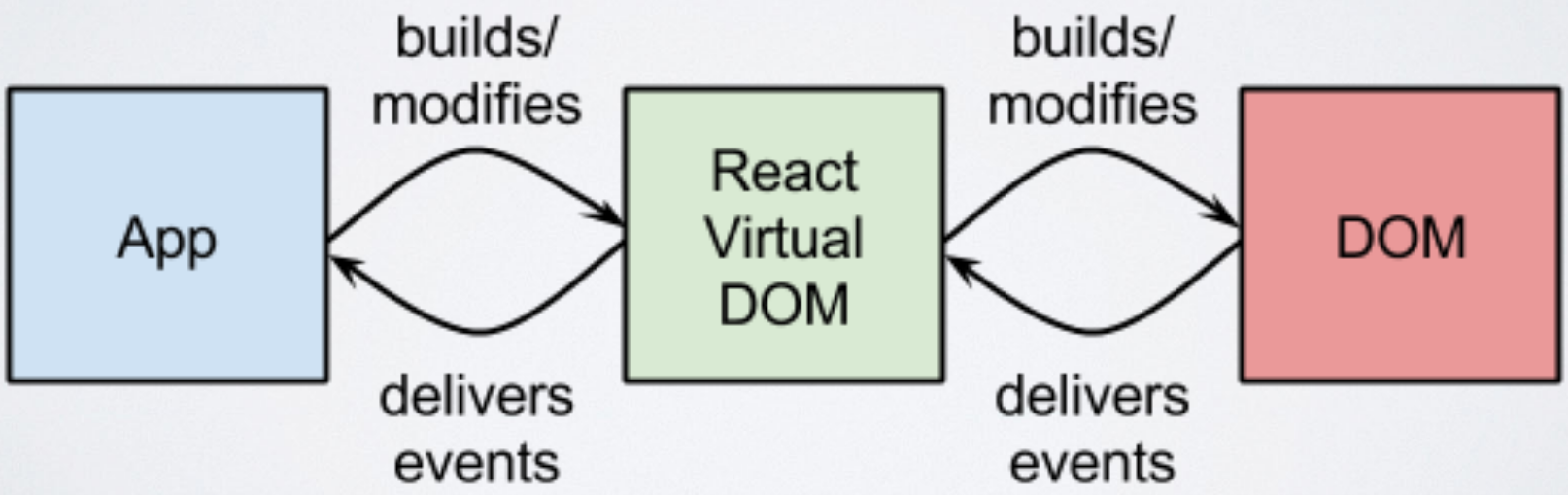
HTML IN JAVASCRIPT??

- **Welcome to JSX** - an alternative syntax for JavaScript that strongly resembles HTML
- It is eventually transpiled to JavaScript
- React then uses these objects to build out the "Virtual DOM" (there will be a quiz later for all these terms)
- JSX helps us see the custom "elements" being constructed by React

Virtual DOM v. Regular DOM

- **TL;DR** → Virtual DOM is a JS representation of the actual DOM
- This helps React keep track of changes in the actual DOM by comparing different instances of the Virtual DOM through a process they call "reconciliation"

- If you are familiar with the way **git** works, then the virtual DOM is similar to way git compares the difference - or "`diff`" - between commits



`ReactDOM.render`

- Takes the output from the `extends` `Component` in your app and adds it to the Virtual DOM

- It takes two arguments:
  1. The component
  2. The DOM location where we will append the component

THAT IS **JSX**

- Remember the `<Hello />` is JSX
- Looks like XML
- Note the self-closing tag

MAKE IT DYNAMIC

- Let's display a greeting to the user
- Not going to hard-code any info, just pass it down to the component
- ⚠ Pass in data where the component is rendered

WHY USE THE `props` THINGS?

`props`

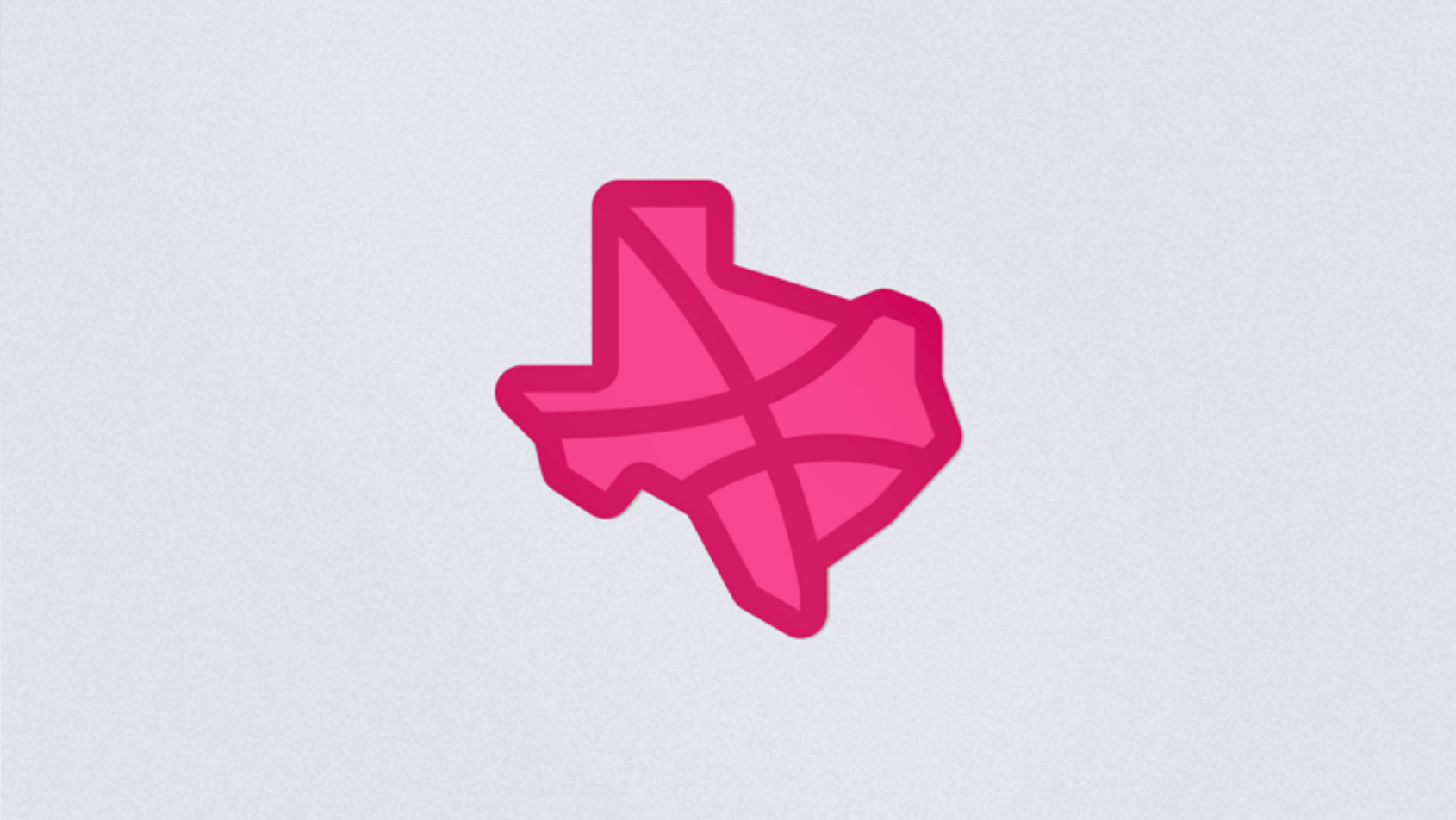
- Every component has `props` that passes "properties" through the component from a higher level
- Props are immutable i.e. they cannot be changed when the app is running
- The developer defines props and passes them into a JSX element as an attribute
- Let's add another prop!

Important!

- The return statement in the render method can only return one DOM element
- This will be a bit different in the next major version of React, which will reduce the number of `<div>`s in our code

This is

INTERNET



STATE

Illustration by Gustavo Zambelli
from dribbble.com

`props` v. `state`

- We have seen how props work
- `props` represent data that will be the same on every render
- `state` represents data that will change depending on the actions of the user

- Values stored in a component's state can be changed (they are mutable)
- We can access state values using `this.state.val`
- Setting up and modifying state is not as straightforward as properties
- Must explicitly declare the mutation, and then define methods to define how to update our state....

TRIGGER IT

- Initial state is set up, but we need to trigger it
- Let's make an event that will change the counter
- Back to the code…

MIKE, EXPLAIN THAT

- The `<button>` element takes an attribute called `onClick`. This lets us access the click event.
- The value of `onClick` is an anonymous function that invokes `handleClick`, defined earlier in the component

- Whenever we run `.setState`, our component looks at the "diff" between the current DOM and the Virtual DOM node
- It will only replace the current DOM with parts that have changed

THINGS WE DIDN'T COVER

- How do you even do styles like this??
- Can I use super-neato functional programming with this?
- When would we use lifecycle methods?
- How could we manage state for several things per page?
- Is there a way to unit test in React?

“Asking questions is awesome. Do that.”

–M. Mathew

STAY IN TOUCH!

@drumsensei

drumsensei.com