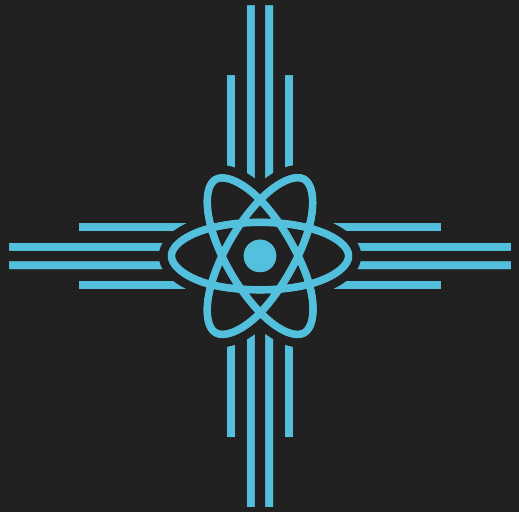


505 React

Meetup 1

Before we begin

Install Node.js for your operating system

https://nodejs.org/en/

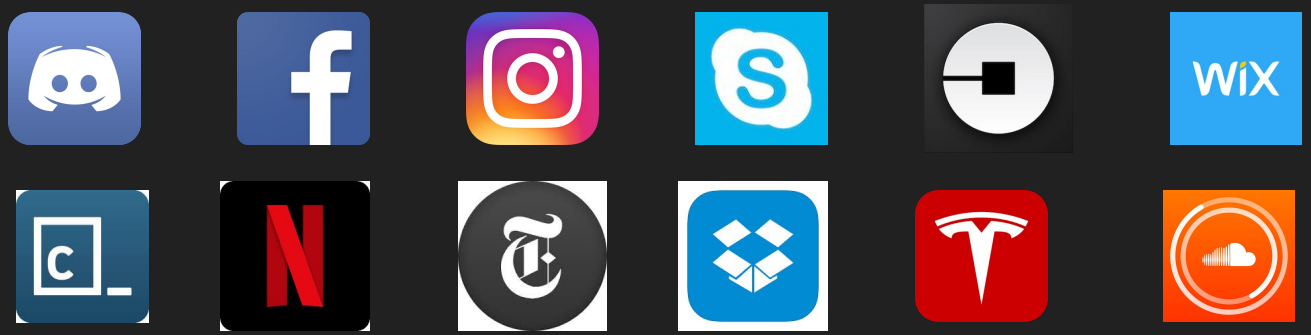
Welcome to 505 React

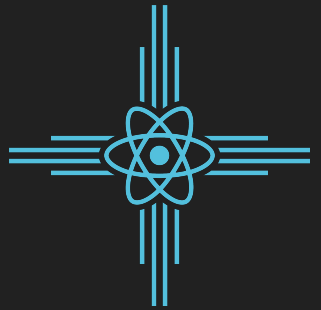
Tell us a little about yourself.

- Name
- Where you work
- What you know about React / React Native already
- What you want to know about React / React Native

What exactly is React

- It is a JavaScript LIBRARY not a framework
- Developed by Facebook and used internally until being released in 2013
- Learn once write ANYWHERE (React and React Native work similarly and work on both platforms)



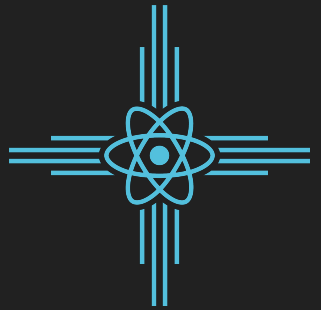


Why use React?

- It uses a **Virtual DOM** (React not React Native)
- Allows for rendering and management of data that changes **state** frequently
- **Component** based architecture makes applications highly scalable
- Utilizes **JSX** syntax (ES6, ES7, ES8)
- Allows for integrations with various other libraries and components through **npm** and **yarn**

React Pieces

- Components
- State/Props
- Redux (sort of)



React Components

- Lifecycle
  - render()
  - componentDidMount()
- render() method is where all your UI code goes
- Example Class

```
class 505ReactWidget extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

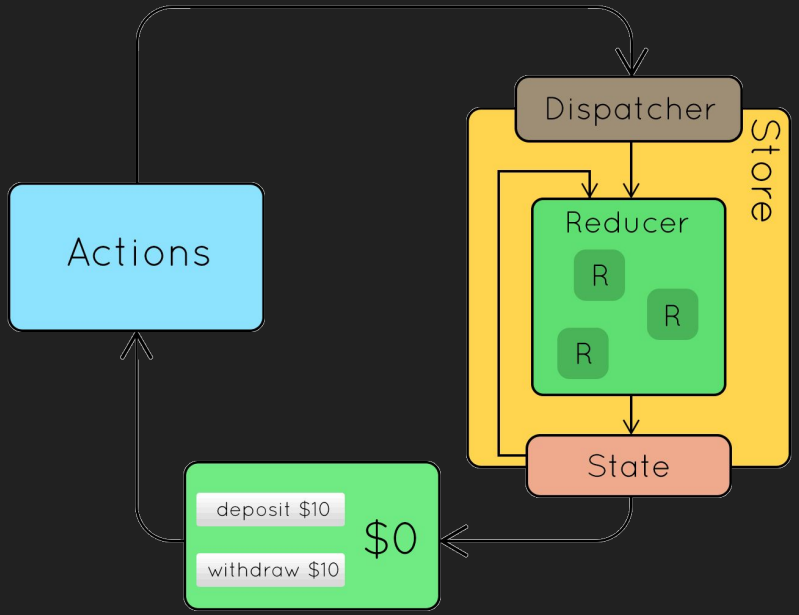
React Props and State

- Props/State both store data that you components need to render
- Props are passed in and State is local to your component
- Props can be used to pass both data and functions to child components

```
class Foo extends Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }

  handleClick = () => {
    this.setState({
      date: new Date()
    });
    console.log('Click happened');
  }

  render() {
    return <ChildComponent onClick={this.handleClick}>Click Me</ChildComponent>;
  }
}
```

React/Redux

- Redux is not technically part of React



Short intro to JSX

```
const element = <h1 className=”greeting”>Hello, world!</h1>;
```

This variable declaration is neither html nor a string -- it’s JSX!

- **JSX is a syntax extension to JavaScript (and comes with the full power of Javascript)**
- **We recommend using it with React to describe what UI should look like**
- **You can put *any* JavaScript expressions within braces inside JSX. Each React element is a JavaScript object that you can store in a variable or pass around in your program:**

```
const name = 'Josh Perez';
const element = <h1>Hello, {name}</h1>;

ReactDOM.render(
  element,
  document.getElementById('root')
);
```

Create React App

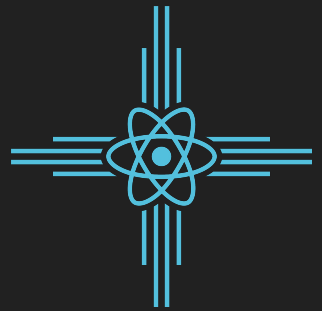
- Is a quick boilerplate for getting a React web application up and running
- Uses Webpack
- Installed once on your system and can be used over and over

In terminal run the following command

```
sudo npm i -g create-react-app@1.4.1
```

Enter your password

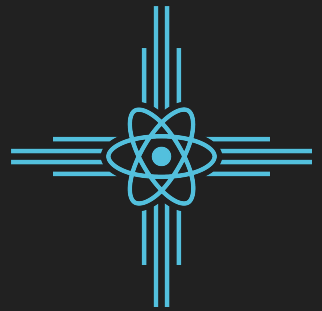
If you get an error, it may be because you installed Node via Homebrew. We recommend uninstalling node and reinstalling directly from https://nodejs.org/en/



In terminal run the following command

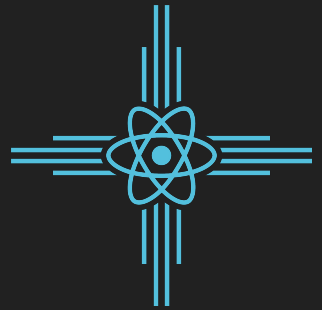
```
create-react-app
/Users/[username]/Desktop/chat-app
--scripts-version=1.0.14
```

This will create a new directory, named “chat-app”
on your Desktop that will hold our code



If you are using Chrome or find React Dev Tools in the Chrome plugin store, install and restart the browser.

Do the same if you are using FireFox



In terminal run the following commands

```
cd /Users/[username]/chat-app
```

This will get you into your new React app folder

```
npm start
```

This will launch your new app in the browser

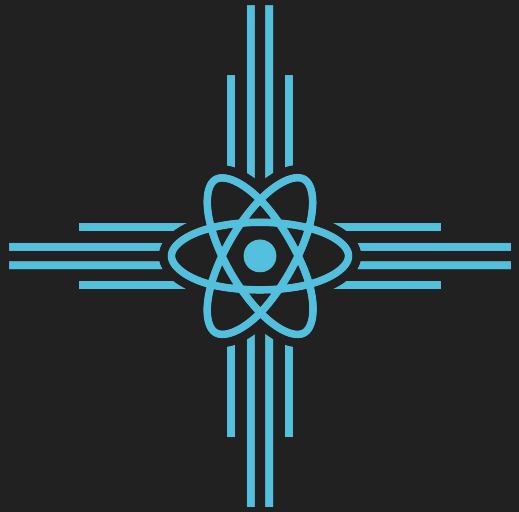
Your First React Component

```
import React, { Component } from 'react';

class App extends Component {
constructor(props) {
    super(props);
    this.state = {name: "Andy"};
  }
 render() {
   return (
     <div className="App">
       <header className="App-header">
         <img src={logo} className="App-logo" alt="logo" />
         <h1 className="App-title">Welcome to React {this.state.name}</h1>
       </header>
       <p className="App-intro">
         To get started, edit <code>src/App.js</code>
       </p>
     </div>
   );
 }
}

export default App;
```

You can use an ES6 Class to define a component

App is a React Component class. You have your imports at the top and export at the bottom.

A component takes in parameters, called props (short for “properties”), and returns a hierarchy of views to display via the render method.



505 React

Meetup 2

Before we begin

Make sure you have the code from our last meet up and have followed all the set up directions in the slides from the last meetup

https://github.com/samanthaandrews/505-React-Meetup

Welcome to 505 React

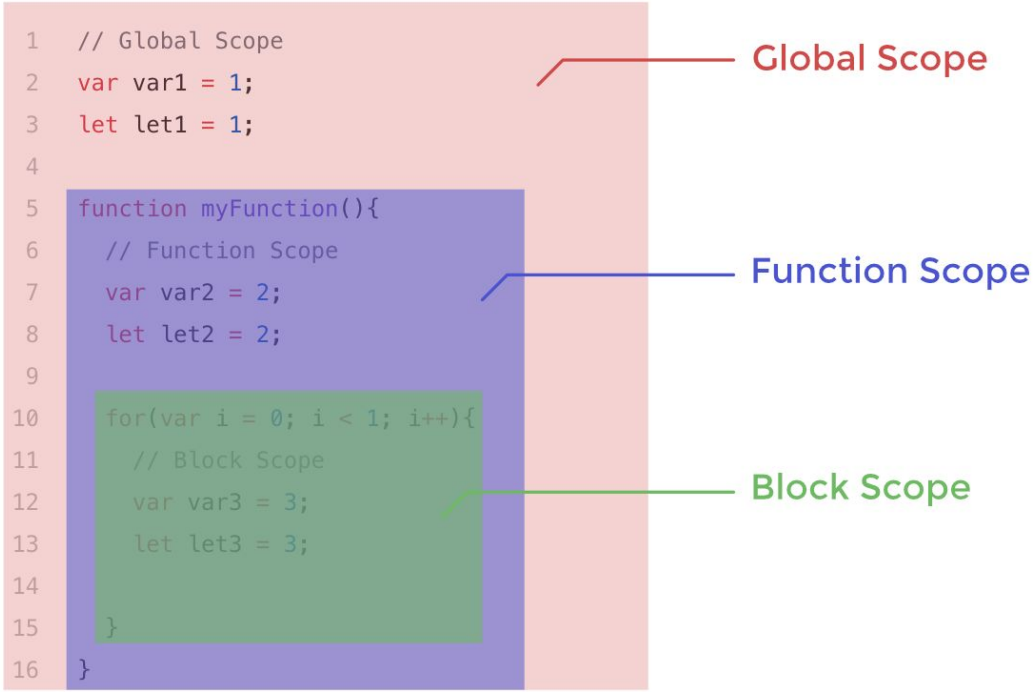
Tell us a little about yourself.

- Name
- Where you work
- What you know about React / React Native already
- What you want to know about React / React Native

ES6: `var`, `let`, and `const`

- ES6 came with the addition of `let` and `const`, which can be used for variable declaration.
- What makes them different from `var`? Scope, use, and hoisting

Scope

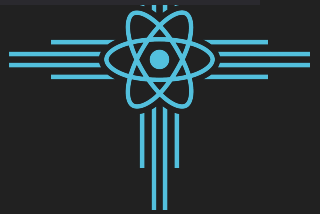

1 // Global Scope
2 var var1 = 1;
3 let let1 = 1;
4
5 function myFunction(){
6 // Function Scope
7 var var2 = 2;
8 let let2 = 2;
9
10 for(var i = 0; i < 1; i++){
11 // Block Scope
12 var var3 = 3;
13 let let3 = 3;
14
15 }
16 }
Global Scope
Function Scope
Block Scope

VAR

- Globally scoped OR function/locally scoped
- Can be redeclared and reassigned
- Hoisting of `var` - hoisting is a JS mechanism where variables and function declarations are moved to the top of their scope at code execution

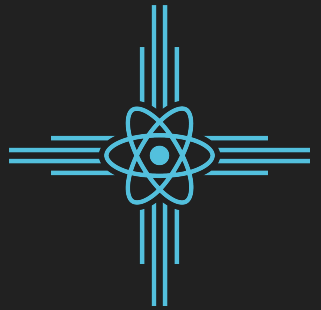
```
console.log (greeter);
var greeter = "say hello"
```

```
var greeter;
console.log(greeter); //greeter is undefined
greeter = "say hello"
```



The problem with VAR

```
var greeter = "hey hi";
var times = 4;

if (times > 3) {
    var greeter = "say Hello instead";
}

console.log(greeter) //"say Hello instead"
```



LET

- Block scoped - a block is a chunk of code bounded by `{}`
- Can be reassigned but not redeclared

```
let greeting = "say Hi";
let times = 4;

if (times > 3) {
     let hello = "say Hello instead";
     console.log(hello);//"say Hello instead"
 }
console.log(hello) // hello is not defined
```

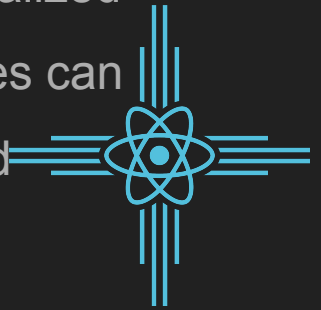
- Just like `var`, `let` declarations are hoisted to the top. But the let keyword is not initialized. So you will get a `Reference Error` instead of `undefined`.

CONST

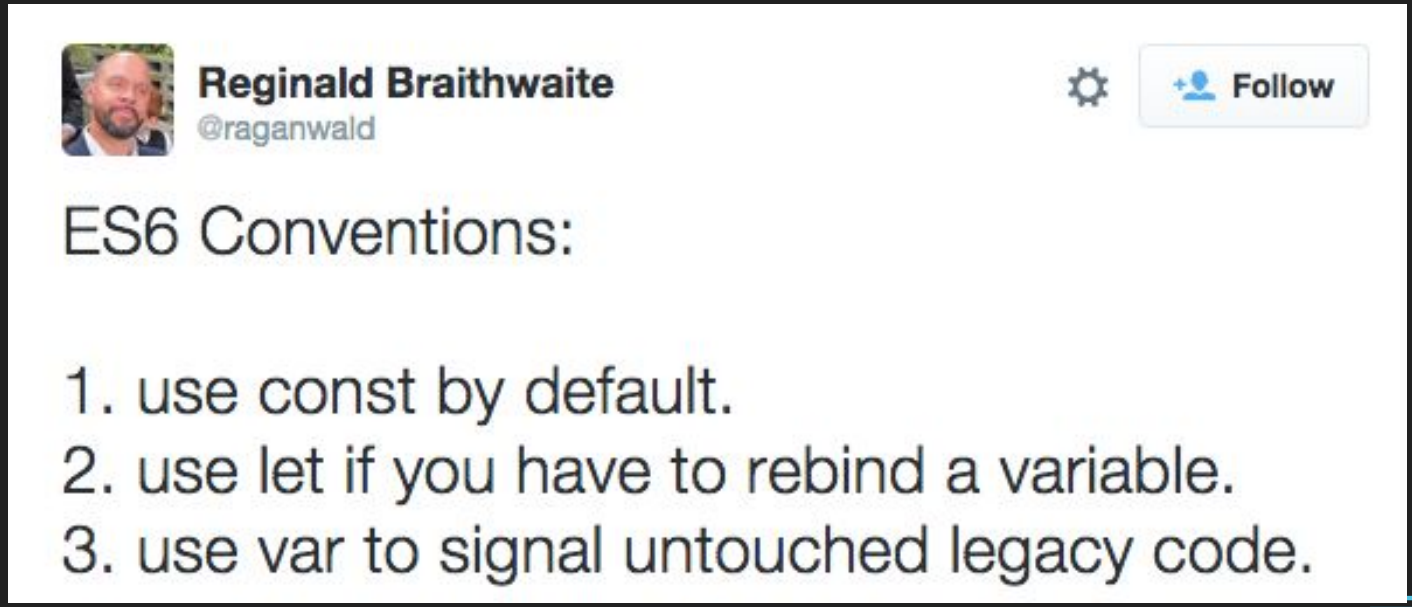
- Block scoped, just like `let`
- Cannot be redeclared or reassigned

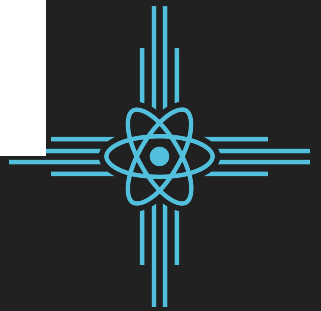
```
const greeting = "say Hi";
greeting = "say Hello instead";//error : Assignment to constant variable.
```

- Just like `let,` declarations are hoisted to the top but are not initialized
- Disclosure: When you declare an object using const, its properties can be modified or added, but the variable itself cannot be redeclared



ES6: var, let, and const

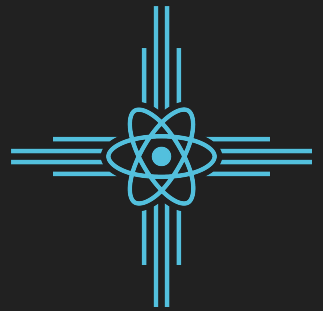




Ternary

- A ternary is a shorter way to do a conditional evaluation
- They implicitly return something regardless of the evaluation

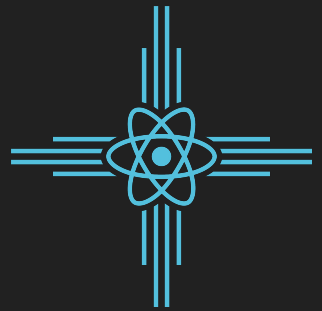
```
let bar;
if (foo === true) {
  bar = 'Hello World'
} else {
  bar = 'Goodbye'
}

const bar = foo ? 'Hello World' : Goodbye';
```



Ternary

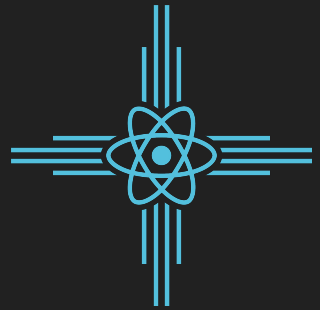
- 3 parts: condition ? true code to execute : false code to execute;

```
const bar = foo ? 'Hello World' : 'Goodbye';
```



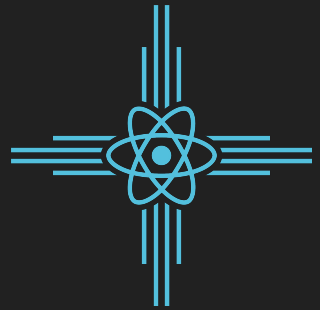
Firebase

- Firebase is a Backend-as-a-Service. It is your server, your API, and your datastore, all written so generically that you can modify it to suit most needs.
- We will be using it to store all of our chat messages and display them for everyone to see.



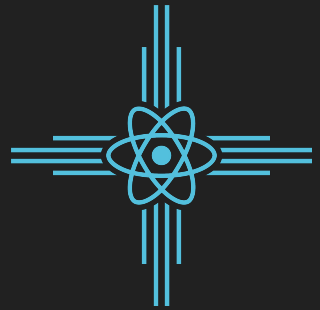
Setup

- yarn add firebase / npm i --save firebase



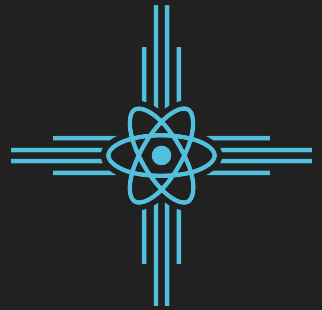
src/firebase.js

```
import firebase from 'firebase';

const config = {
   apiKey: "AIzaSyD9HEJmF66X6Q5NR70U2dt7pFOIoQyw9ys" ,
   authDomain: "reactchat-5e8e9.firebaseapp.com" ,
   databaseURL: "https://reactchat-5e8e9.firebaseio.com" ,
   projectId: "reactchat-5e8e9",
   storageBucket: "reactchat-5e8e9.appspot.com" ,
   messagingSenderId: "888828995621"
};

firebase.initializeApp(config);

export default firebase;
```



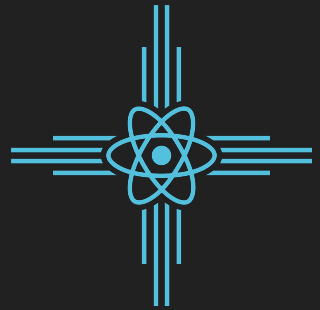
App.js changes (componentDidMount)

```
componentDidMount = () => {
  const messagesRef = firebase.database().ref('messages');

  messagesRef.on('value', (snapshot) => {
    let messages = [];
    snapshot.forEach(element => {
      messages.push(`${element.val().username}: ${element.val().message}`);
    })
    this.setState({
      messages,
    })
  });
}
```



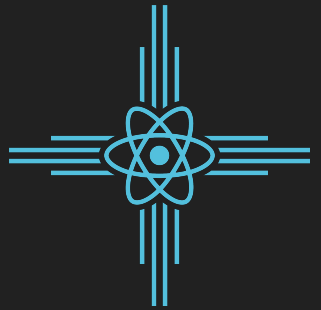
App.js changes (addNewMessage)

```
addNewMessage = message => {
   const messagesRef = firebase.database().ref('messages');
   messagesRef.push({
     username: 'jhonny#5',
     message,
   }, function(error) {
     if (error) {
       console.log(error)
     }else {
       console.log('success');
     }
   })
   //this.setState({ messages: [...this.state.messages , message] });
 }
```



Styled Components

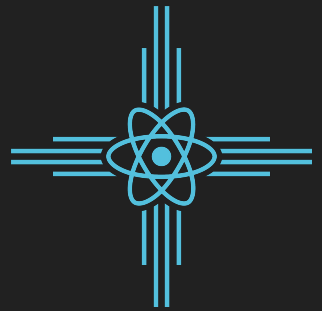
- Styled Components are a way of writing CSS in JavaScript
- The syntax is similar to regular CSS
- https://www.styled-components.com/



Styled Components

- Styled Components are less verbose than normal CSS in JS
- The syntax is like normal CSS
- The allow you to conditionally render styles in the CSS declaration
- They are easy to test
- They allow you set themes

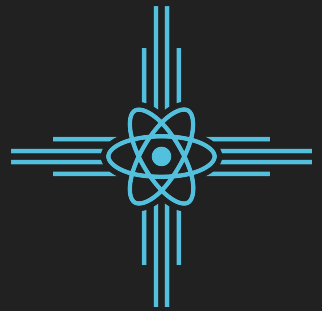
```
const NewStyledComponent = styled.div`
  color: #000000;
  margin: 25px;
`;
```



Styled Components

- They are created by declaring a new variable that extends an HTML element

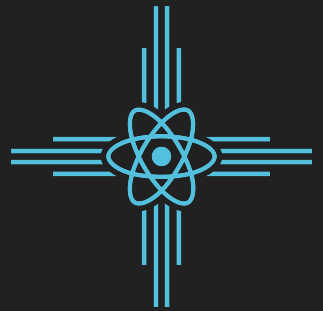
```
const StyledImage = styled.img`
  width: 100%;
  margin: 50px;
`;
class App extends Component {
  render() {
    return (
      <StyledImage />
    );
  }
}
```



Styled Components

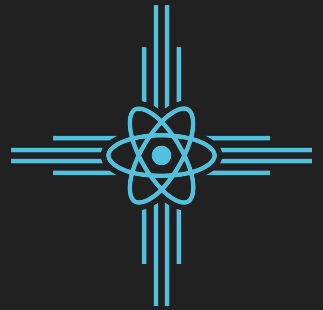
- They can also render the style conditionally

```
const mainColor = 'indianred'

const Title = styled.h1`
  color: ${props => props.color ||  'goldenrod'}
`;

class App extends Component {
  render() {
    return (
      <Title color={mainColor}>Mystagram</Title>
    );
  }
}
```



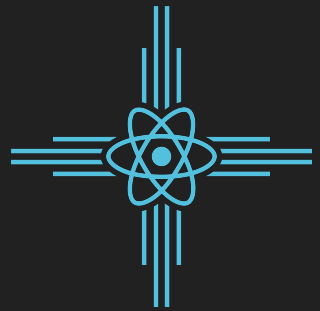
In terminal run the following command inside your app directory

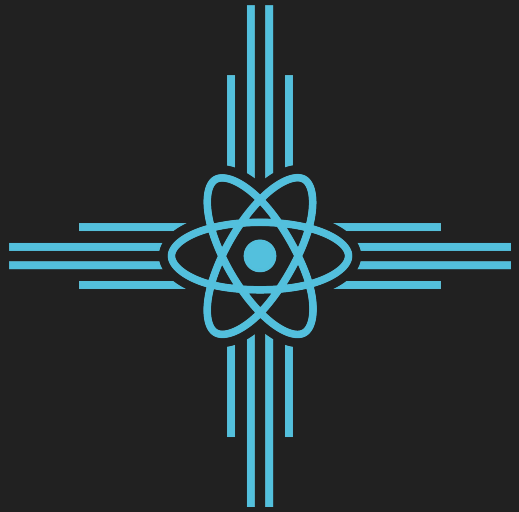
```
npm install --save styled-components
```



Next import them into App.js

```
import styled from `styled-components`;
```





505 React

Meetup 3

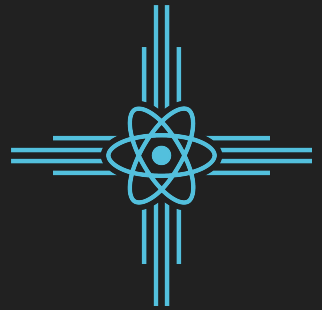
Before we begin

Make sure you have the code from our last meet up and have followed all the set up directions in the slides from the last meetup

https://github.com/samanthaandrews/505-React-Meetup

Arrow Functions

- More concise syntax
- Simplify function scoping and the `this` keyword
- Work much like lambdas in C# or Python
- We don’t have to use the `function` keyword, the `return` keyword, or curly brackets



```
// ES5

var multiplyES5 = function(x, y) {

  return x * y;

};


// ES6

const multiplyES6 = (x, y) => { return x * y };
```

Curly brackets aren’t required if only one expression is present; return is implicit

```
const multiplyES6 = (x, y) => x * y;
```

Parentheses are optional when only one parameter is passed
Use empty parentheses when passing 0 parameters

```
const phraseSplitterEs6 = phrase => phrase.split(" ");
```

Wrap object literals in parentheses

```
1 | var func = () => ({foo: 1});
```

Common Use Case for Arrow Functions: Array Manipulation

```
const array = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15];


// ES5
var divisibleByThrreeES5 = array.filter(function (v){
  return v % 3 === 0;
});


// ES6
const divisibleByThrreeES6 = array.filter(v => v % 3 === 0);


console.log(divisibleByThrreeES6); // [3, 6, 9, 12, 15]
```

Let’s talk about `this`

Until arrow functions, every new function defined its own `this` value (based on how function was called, a new object in the case of a constructor, undefined in strict mode function calls, etc.). This proved to be less than ideal with an object-oriented style of programming.

An arrow function does not have its own `this`, `this` comes from the surrounding lexical content.

i.e. arrow functions follow normal variable lookup rules

ES3/ES5

```
function Person() {
  var that = this;
  that.age = 0;

  setInterval(function growUp() {
    // The callback refers to the `that` variable of which
    // the value is the expected object.
    that.age++;
  }, 1000);
}
```

ES6

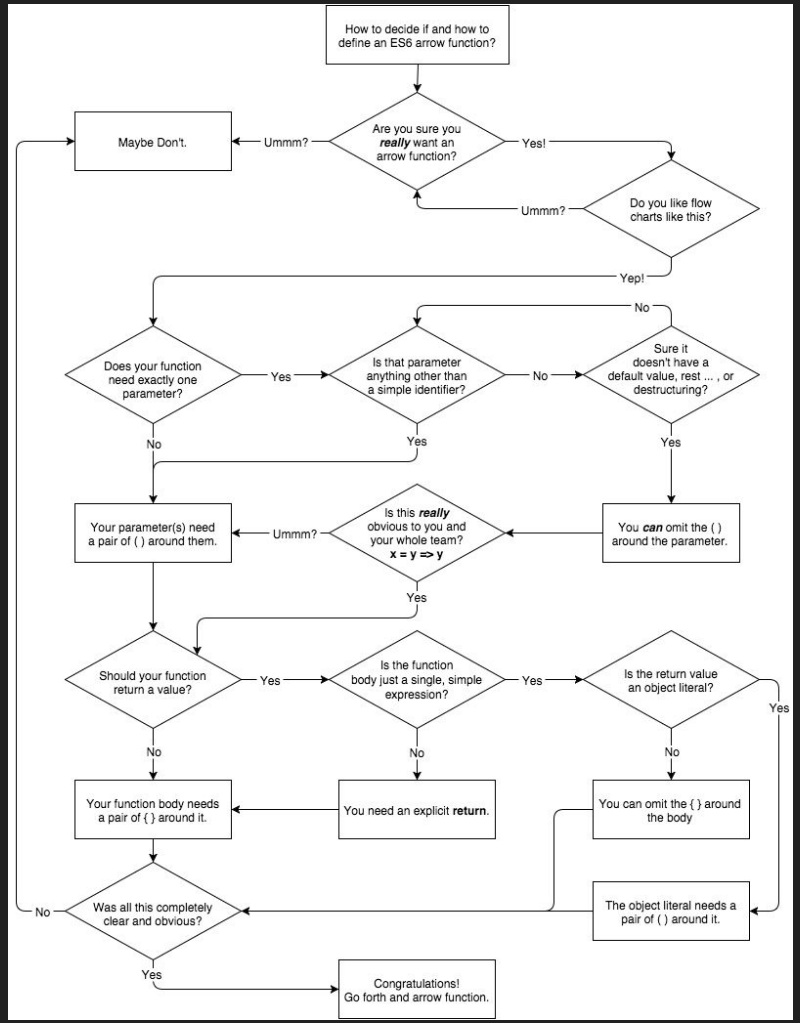
```
function Person(){
  this.age = 0;

  setInterval(() => {
    this.age++; // |this| properly refers to the Person object
  }, 1000);
}

var p = new Person();
```

Arrow Function Pitfalls

- Cannot be used as constructors.
- Do not have their own `arguments` object. Best to use rest parameters instead.
- Since arrow functions do not have their own `this`, the methods `call()` or `apply()` can only pass in parameters. `thisArg` is ignored.
- Best suited for non-method functions.
- Arrow functions do not have a `prototype` property.
- Using the `yield` keyword in ES6 arrow functions will throw an error. Use ES6 generators instead.

Arrow Function Pitfalls - Arrow Functions Used As Methods Example

```
1  'use strict';
2
3  var obj = {
4    i: 10,
5    b: () => console.log(this.i, this),
6    c: function() {
7      console.log(this.i, this);
8    }
9  }
10
11 obj.b(); // prints undefined, Window {...} (or the global object)
12 obj.c(); // prints 10, Object {...}
```



How to decide to use an ES6 arrow function

Flow chart by Kyle Simpson from the book *You Don't Know JS: ES6 & Beyond*

When should I use arrow functions?

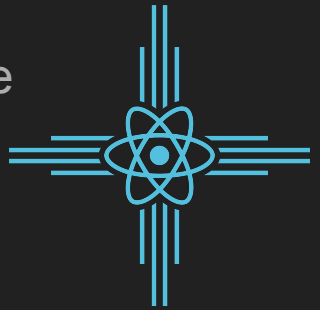
According to developer Lars Schoning

- Use `function` in the global scope and for `Object.prototype` properties
- Use `class` for object constructors
- Use `=>` everywhere else

Kevin Smith (a random guy) counted function expressions in various popular libraries/frameworks and found that roughly 55% of function expressions would be candidates for arrow functions.

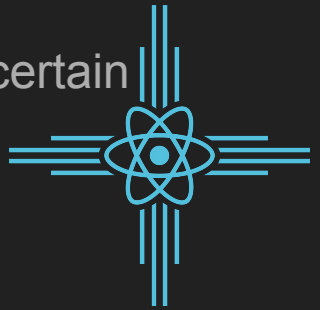
Redux

- Redux is a predictable state container for JavaScript apps. It is an open-source, JS library.
- No matter the size of your application, the whole state of your app is stored in an object tree inside a single *store*.
- The only way to change the state tree is to emit an *action,* an object describing what happened.
- To specify how the actions transform the state tree, you write pure *reducers*.



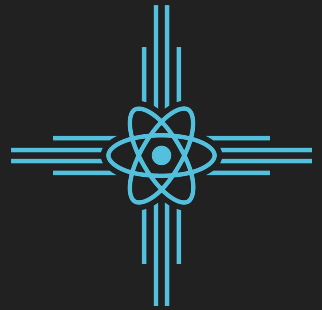
Why Use Redux?

- Our code must manage more state than ever before. This state can include server responses and cached data, as well as locally created data that has not yet been persisted to the server. Plus UI state!?
- Managing this ever-changing state is hard. At some point, you no longer understand what happens in your app as you have **lost control of the when, why, and how of its state.**
- Redux attempts to make state mutations predictable by imposing certain restrictions on how and when updates can happen.



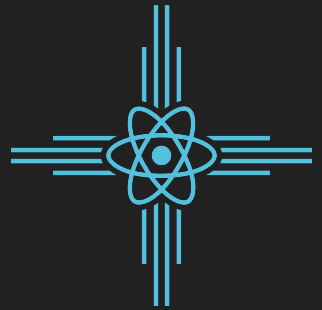
When it makes sense to use Redux

- You have reasonable amounts of data changing over time
- You need a single source of truth for your state
- You find that keeping all your state in a top-level component is no longer sufficient
- You don’t want to pass props down various generations



Three Principles of Redux

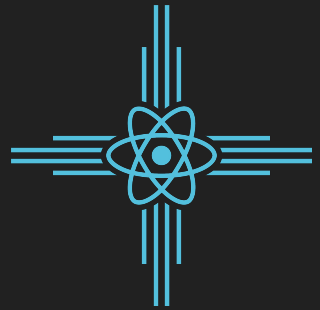
1. **Single source of truth** - the state of your whole application is stored in an object tree with a single store.
2. **State is read-only** - the only way to change the state is to emit an action, an object describing what happened.
3. **Changes are made with pure functions** - to specify how the state tree is transformed by actions, you write pure reducers.



Three Principles of Redux

1. **Single source of truth** - the state of your whole application is stored in an object tree with a single store.

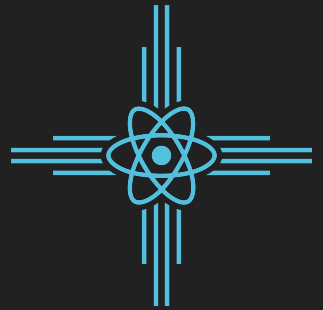
```
console.log(store.getState())

/* Prints
{
  visibilityFilter: 'SHOW_ALL',
  todos: [
    {
      text: 'Consider using Redux',
      completed: true,
    },
    {
      text: 'Keep all state in a single tree',
      completed: false
    }
  ]
}
*/
```



Three Principles of Redux

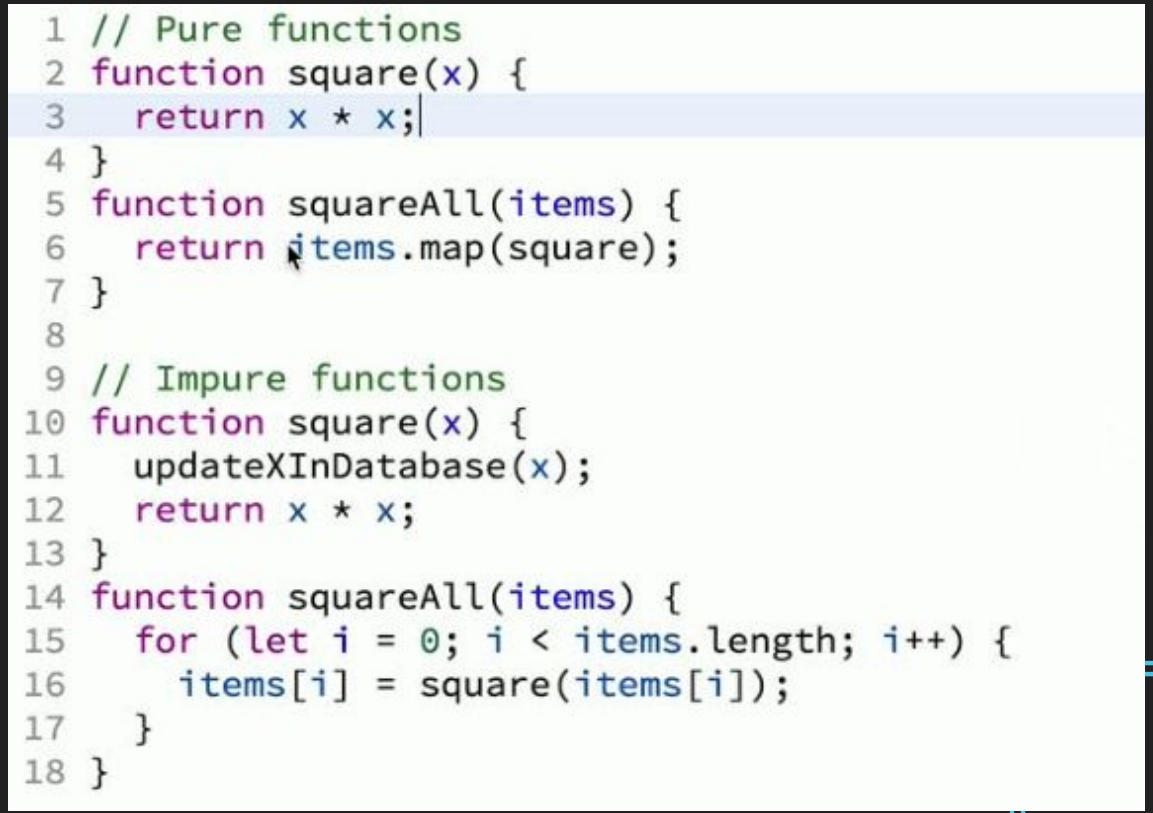
2. State is read-only - the only way to change the state is to emit an action, an object describing what happened.

```
store.dispatch({
  type: 'COMPLETE_TODO',
  index: 1
})

store.dispatch({
  type: 'SET_VISIBILITY_FILTER',
  filter: 'SHOW_COMPLETED'
})
```



Three Principles of Redux

The third principle is dependent upon an understanding of pure vs. impure functions



```
// Pure functions
function square(x) {
  return x * x;
}
function squareAll(items) {
  return items.map(square);
}

// Impure functions
function square(x) {
  updateXInDatabase(x);
  return x * x;
}
function squareAll(items) {
  for (let i = 0; i < items.length; i++) {
    items[i] = square(items[i]);
  }
}
```

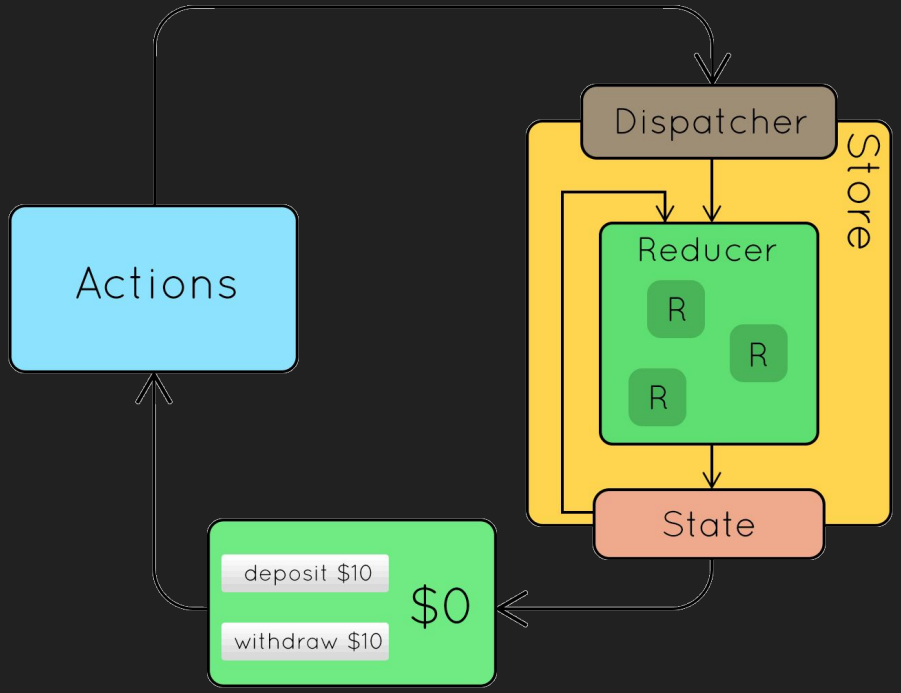
Three Principles of Redux

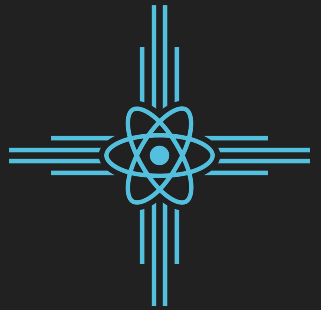
3. Changes are made with pure functions - to specify how the state tree is transformed by actions, you write pure reducers.

A reducer is just a function that takes the previous state & the action being dispatched, and returns the next state of your application.

```
function visibilityFilter(state = 'SHOW_ALL', action) {
  switch (action.type) {
    case 'SET_VISIBILITY_FILTER':
      return action.filter
    default:
      return state
  }
}

function todos(state = [], action) {
  switch (action.type) {
    case 'ADD_TODO':
      return [
        ...state,
        {
          text: action.text,
          completed: false
        }
      ]
    case 'COMPLETE_TODO':
      return state.map((todo, index) => {
        if (index === action.index) {
          return Object.assign({}, todo, {
            completed: true
          })
        }
        return todo
      })
    default:
      return state
  }
}

import { combineReducers, createStore } from 'redux'
const reducer = combineReducers({ visibilityFilter, todos })
const store = createStore(reducer)
```

Redux


Actions
Dispatcher
Store
Reducer
R
R
R
State
deposit $10
withdraw $10
$0



In case you don’t have it installed

Install Node.js for your operating system

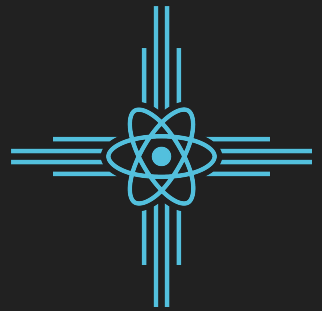
https://nodejs.org/en/

In terminal run the following command

```
sudo npm i -g create-react-app@1.4.1
```

Enter your password

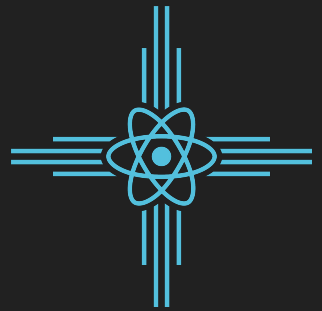
If you get an error, it may be because you installed Node via Homebrew. We recommend uninstalling node and reinstalling directly from https://nodejs.org/en/



In terminal run the following command

```
create-react-app
/Users/[username]/Desktop/trivia-app
--scripts-version=1.0.14
```

This will create a new directory, named “chat-app”
on your Desktop that will hold our code



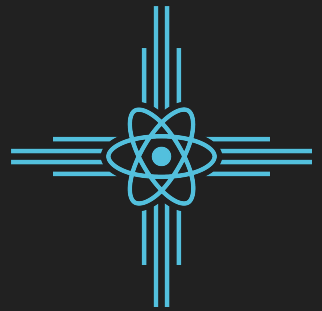
In terminal run the following commands

```
cd /Users/[username]/trivia-app
```

This will get you into your new React app folder

```
npm start
```

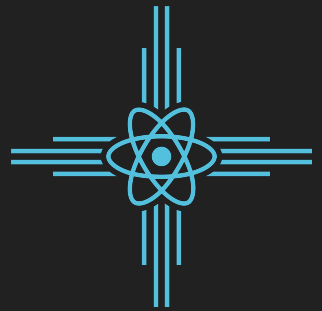
This will launch your new app in the browser



In terminal run the following commands

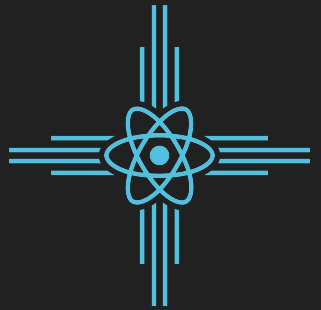
```
npm install --save react-redux
npm install --save redux
```

This will install react-redux and redux in your project



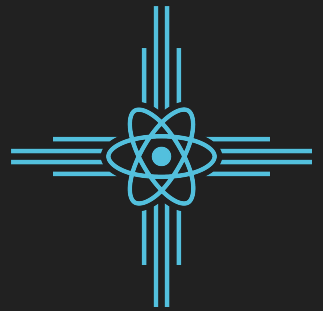
Add the Provider to your app

This will “provide” the store to the child components



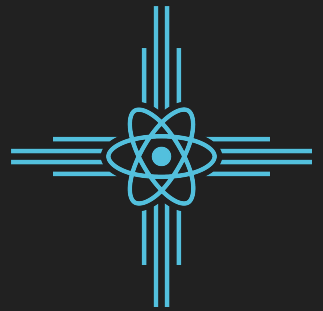
Create the store

This is what will “hold” your state



Create the store

This is what will “hold” your state



Create the actions

This is what defines the functions that modify the store.

You can more than one document outlining actions for cleanliness if you want.

Create the a reducer

This is what controls and changes your state in the store.

You can have more than one per project so we combine them just in case

Connect a component

This will make the store and/or the actions available for you to use

Resources we love:

- https://www.fullstackreact.com/
- https://github.com/getify/You-Dont-Know-JS