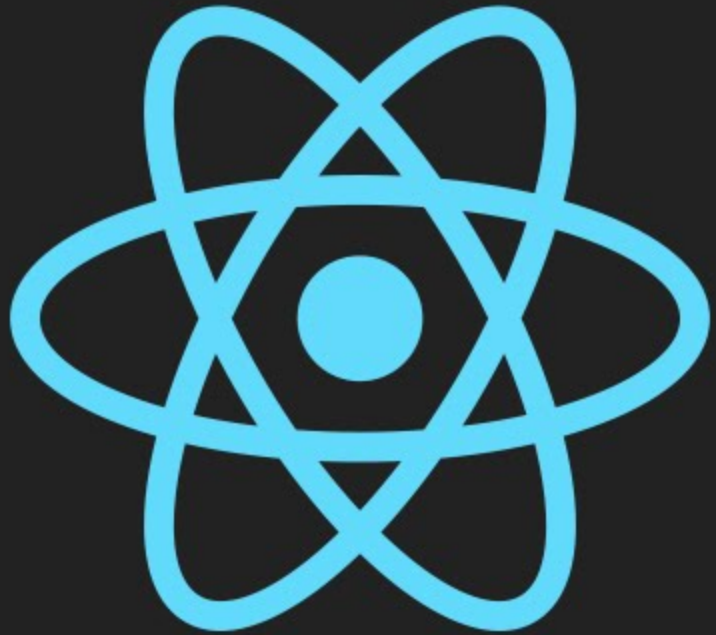


React

Introduction

@Chalach.mo @Vattanan.bu

BASICS



**What is
ReactJS?**

React

A JAVASCRIPT LIBRARY FOR BUILDING USER INTERFACES

Learn once , write anywhere



Mobile Web



Web



Front-End



Back-End

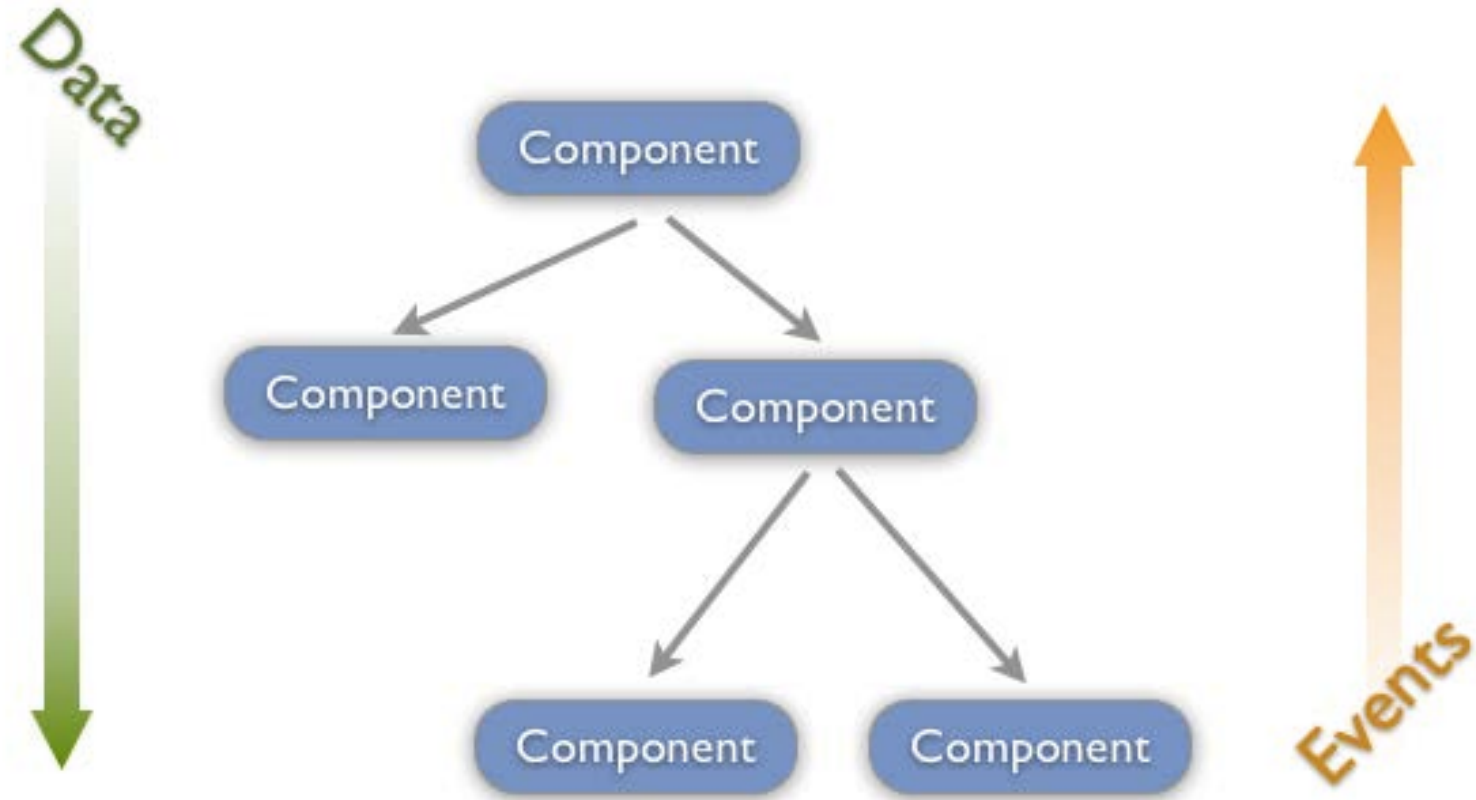


Database



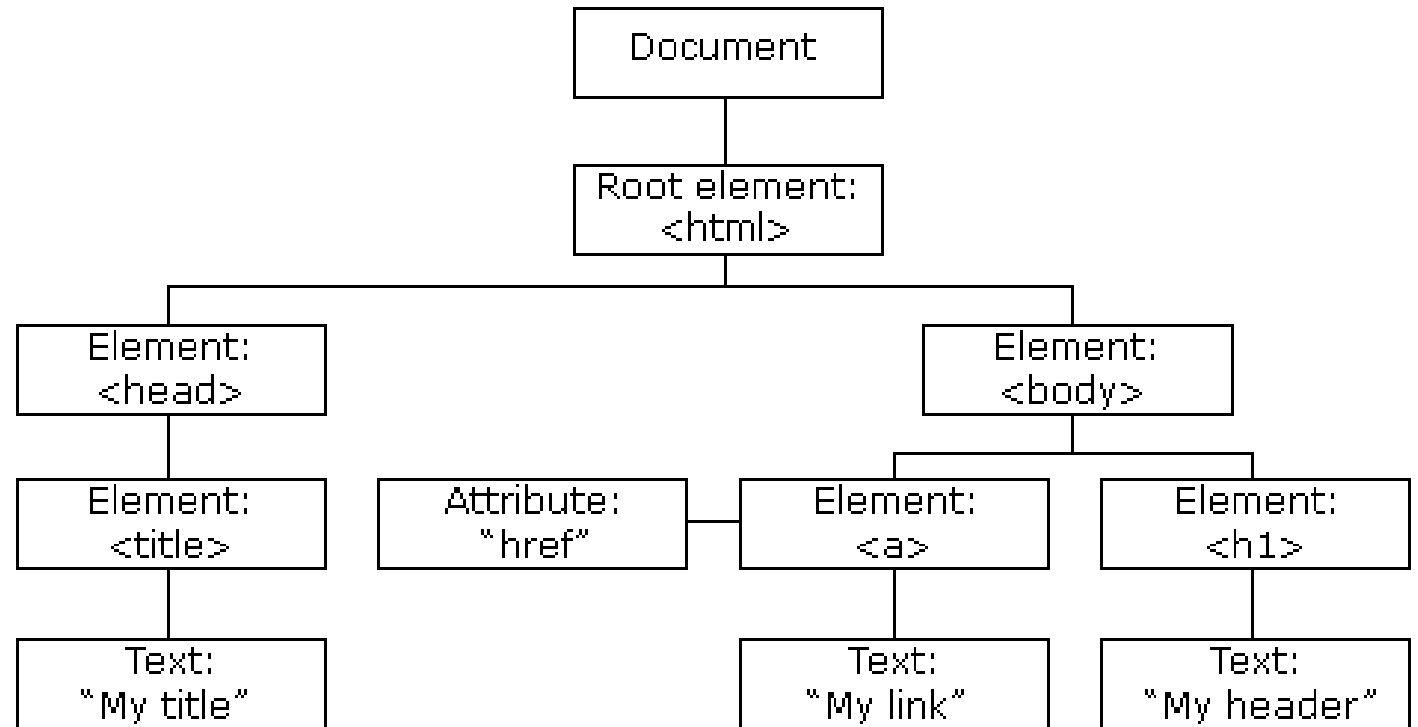
WHY DO WE USE REACT

Single-Way Data Flow

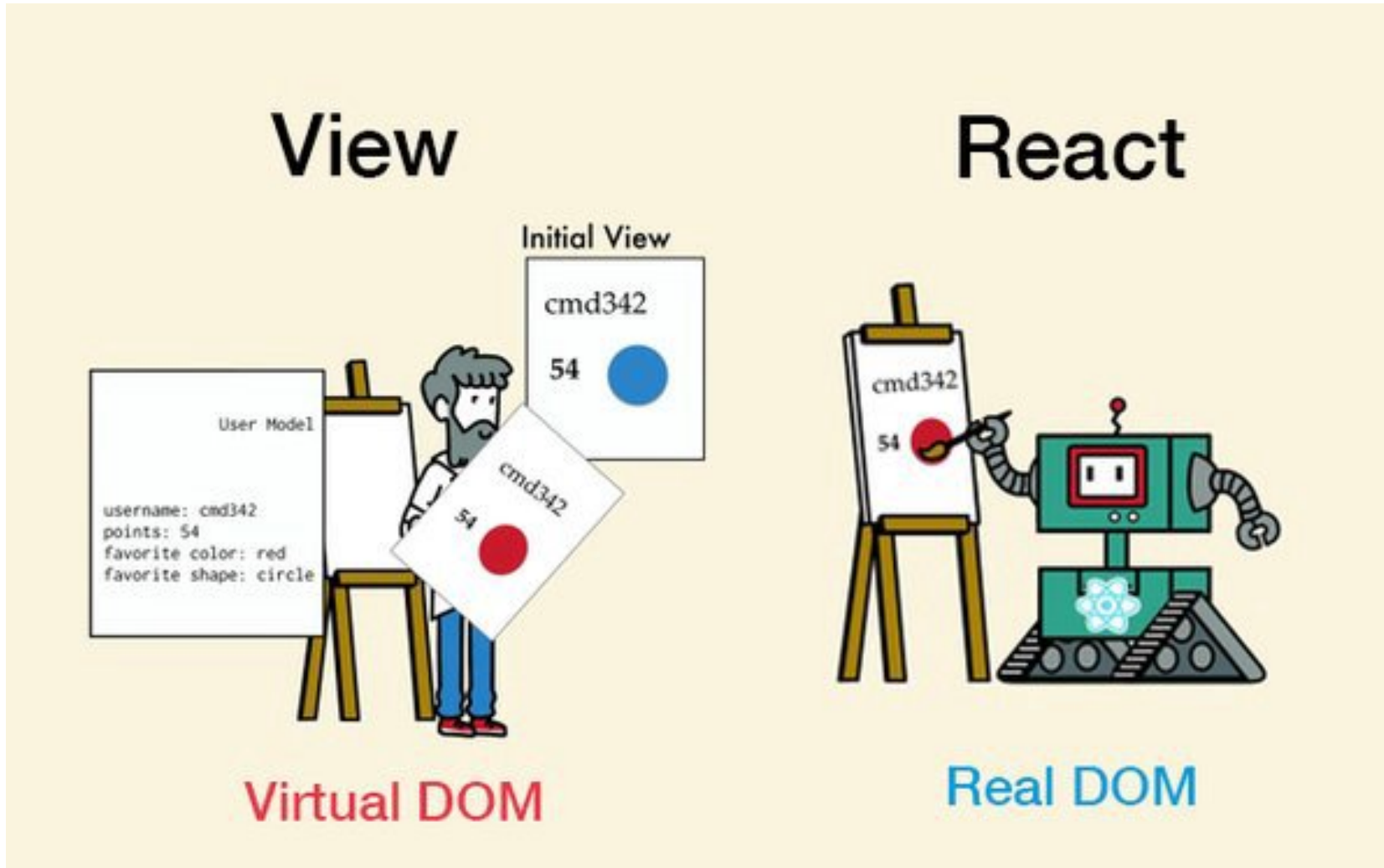


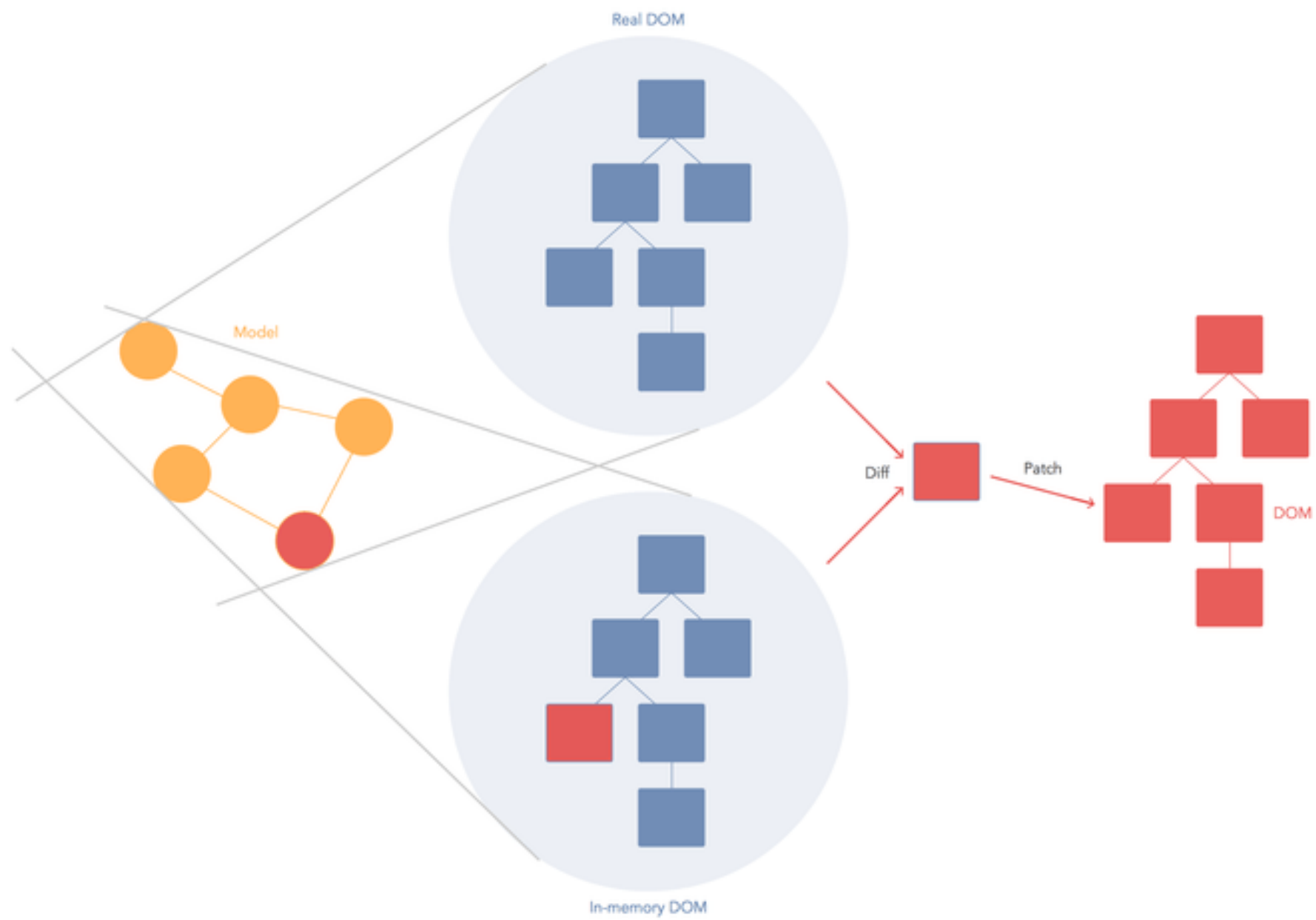
DOM(Document Object Model)

```
<html>
<head>
  <title>My title</title>
</head>
<body>
  <h1>My header</h1>
  <a href="test.html">My link</a>
</body>
</html>
```



Virtual Document Object Model









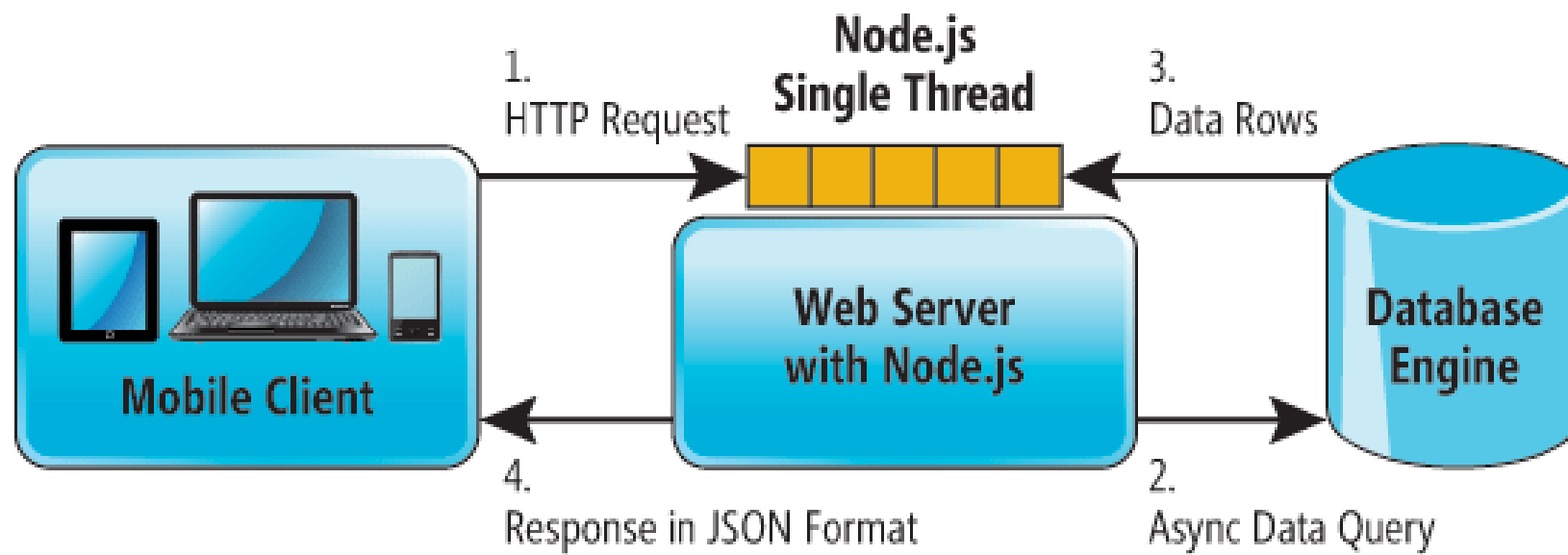
Instagram



What is Node.js?

- Node.js is an open source server framework
- Node.js is free
- Node.js runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- Node.js uses JavaScript on the server





What is NPM(Node Package Manager)?

- NPM is a package manager for Node.js packages, or modules if you like.
- www.npmjs.com hosts thousands of free packages to download and use.
- The NPM program is installed on your computer when you install Node.js



What is Webpack?

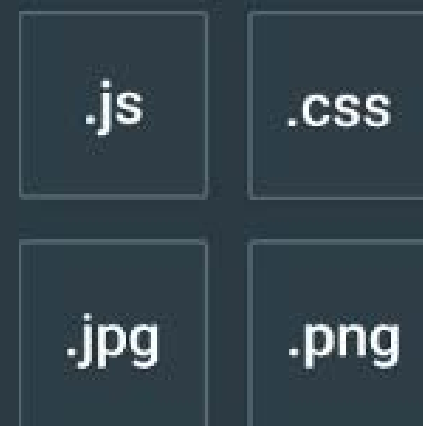
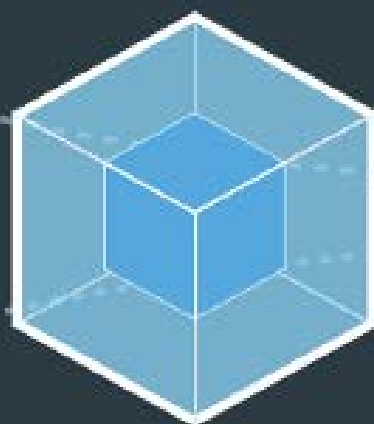
- Webpack is a module bundler for modern JavaScript applications.



webpack
MODULE BUNDLER



MODULES WITH DEPENDENCIES



STATIC ASSETS

Why Webpack?

1

```
// sum.js
var sum = function (a, b) {
  return a + b;
};
```

2

```
// multiply.js
var multiply = function (a, b) {
  var total = 0;
  for (var i = 0; i < b; i++) {
    total = sum(a, total);
  }
  return total;
};
```

3

```
// index.js - our application logic
var totalMultiply = multiply(5, 3);
var totalSum = sum(5, 3);
```

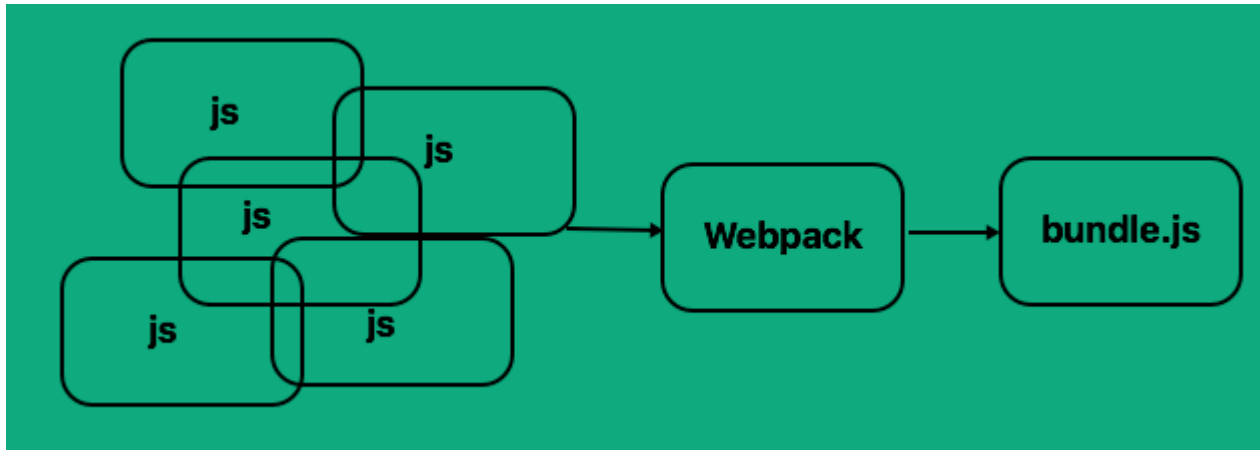
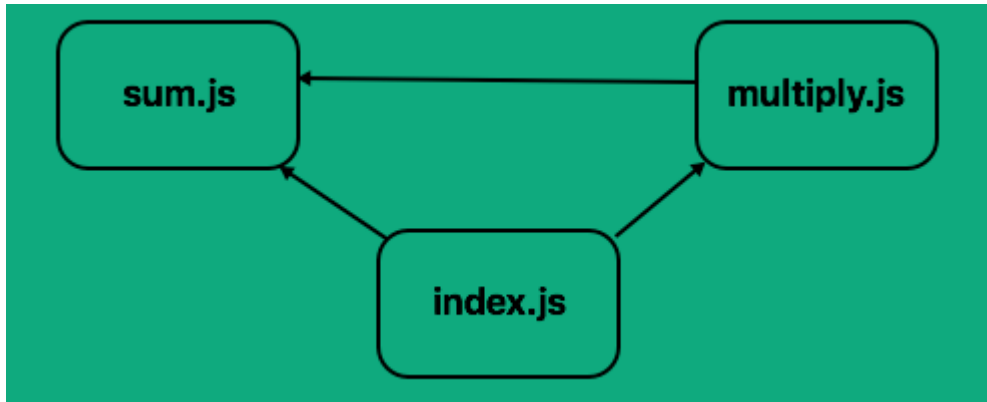
```
console.log('Product of 5 and 3 = ' + totalMultiply);
console.log('Sum of 5 and 3 = ' + totalSum);
```

4

```
// index.html - our entry point to our application
<html>
<head>
<script src="src/sum.js"></script>
<script src="src/multiply.js"></script>
<script src="src/index.js"></script>
</head>
</html>
```

5

```
Product of 5 and 3 = 15
index.js:17 Sum of 5 and 3 = 8
```



CommonJS From Week4(ES6 Modules)

1

```
// sum.js
var sum = function (a, b) {
  return a + b;
};
module.exports = sum;
```

2

```
// multiply.js
var sum = require('./sum');

var multiply = function (a, b) {
  var total = 0;
  for (var i = 0; i < b; i++) {
    total = sum(a, total);
  }
  return total;
};

module.exports = multiply;
```

3

```
// index.js - our application logic
var multiply = require('./multiply');
var sum = require('./sum');

var totalMultiply = multiply(5, 3);
var totalSum = sum(5, 3);

console.log('Product of 5 and 3 = ' + totalMultiply);
console.log('Sum of 5 and 3 = ' + totalSum);
```

4

```
// index.html - our entry point to our
application
<html>
<head>
<script src="./dist/bundle.js"></script>
</head>
</html>
```

How To Use Webpack?

Create File & Folder Based On This Structure

```
npm init
```

```
├── /app/  
│   ├── /index.html  
│   ├── /multiply.js  
│   ├── /sum.js  
│   └── /index.js  
├── /dist/  
├── /node_modules/  
├── /webpack.config.js  
└── /package.json
```

```
npm install webpack -g
```

```
npm install webpack --save-dev
```

Webpack.config.js

```
var path = require('path');
module.exports = {
  entry: './src/index.js',
  output: {
    path: path.resolve(__dirname, './dist/'),
    filename: 'bundle.js'
  }
}
```

Run this command

```
webpack
```

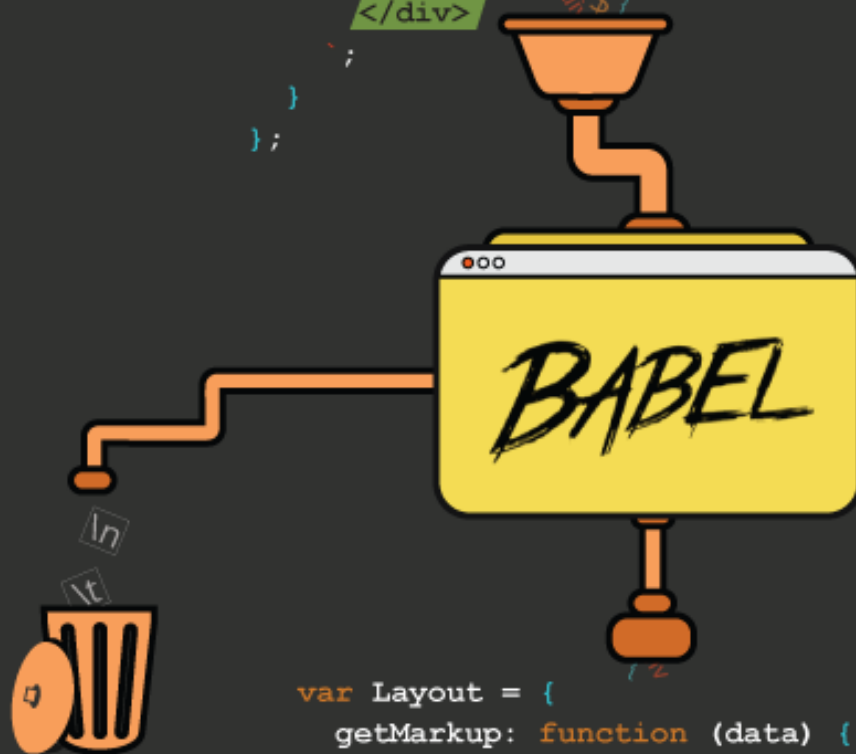
**How to use ES6
with webpack ?**

Ans : webpack + loader

Webpack + Babel



```
var Layout = {  
  getMarkup: function (data) {  
    return nowhitespace`  
      <div id="wrapper">  
        <div id="cta">  
          <a href="${data.href}"></a>  
        </div>  
      </div>  
    `;  
  }  
};
```



```
var Layout = {  
  getMarkup: function (data) {  
    return '<div id="wrapper"><div id="cta"><a  
href="' + data.href + '"></a></div></div>';  
  }  
};
```

ES6 for build webpack

1

```
// sum.js
const sum = (a, b) => a + b;

export default sum;
```

2

```
// multiply.js
import sum from './sum';

const multiply = (a, b) => {
  let total = 0;
  for(let i=0;i<b;i++) {
    total = sum(a, total);
  }
  return total;
};

export default multiply;
```

3

```
// index.js - our application logic
import multiply from './multiply';
import sum from './sum';
```

```
const totalMultiply = multiply(5, 3);
const totalSum = sum(5, 3);
```

```
console.log(`Product of 5 and 3 = ${totalMultiply}`);
console.log(`Sum of 5 and 3 = ${totalSum}`);
```

4

```
<!-- // index.html - our entry point to our
application -->
<html>
  <head>
    <script src="../dist/bundle.js"></script>
  </head>
</html>
```

Webpack.config.js

```
var path = require('path');
module.exports = {
  entry: './src/index.js',
  output: {
    path: path.resolve(__dirname, './dist/'),
    filename: 'bundle.js'
  },
  module: {
    loaders: [
      {
        test: /\.js$/,
        loader: 'babel-loader',
        exclude: /node_modules/,
        query: {
          presets: ['es2015']
        }
      }
    ]
  }
};
```

Run this command

```
npm install babel-core --save-dev
```

```
npm install babel-loader --save-dev
```

```
npm install babel-preset-es2015 babel-preset-react --save-dev
```

```
webpack
```

ES6 after past webpack

<http://babeljs.io/repl/>

```
1 // index.js - our application logic
2 import multiply from './multiply';
3 import sum from './sum';
4 •const totalMultiply = multiply(5, 3);
5 const totalSum = sum(5, 3);
6 •console.log(`Product of 5 and 3 = ${totalMultiply}`);
7 console.log(`Sum of 5 and 3 = ${totalSum}`);
8
```

```
1 'use strict';
2
3 var _multiply = require('./multiply');
4
5 var _multiply2 = _interopRequireDefault(_multiply);
6
7 var _sum = require('./sum');
8
9 var _sum2 = _interopRequireDefault(_sum);
10
11 function _interopRequireDefault(obj) { return obj && obj.__esModule ? obj : { default:
  obj }; }
12
13 // index.js - our application logic
14 var totalMultiply = (0, _multiply2.default)(5, 3);
15 var totalSum = (0, _sum2.default)(5, 3);
16 console.log('Product of 5 and 3 = ' + totalMultiply);
17 console.log('Sum of 5 and 3 = ' + totalSum);
```

```
1 // sum.js
2 const sum = (a, b) => a + b;
3 •export default sum;
4
5
```

```
1 "use strict";
2
3 Object.defineProperty(exports, "__esModule", {
4   value: true
5 });
6 // sum.js
7 var sum = function sum(a, b) {
8   return a + b;
9 };
10 exports.default = sum;
```

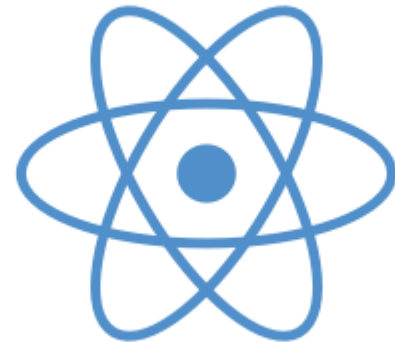
```
1 // multiply.js
2 import sum from './sum';
3 •const multiply = (a, b) => {
4   let total = 0;
5   for(let i=0;i<b;i++) {
6     total = sum(a, total);
7   }
8   return total;
9 };
10 •export default multiply;
11
```

```
1 'use strict';
2
3 Object.defineProperty(exports, "__esModule", {
4   value: true
5 });
6
7 var _sum = require('./sum');
8
9 var _sum2 = _interopRequireDefault(_sum);
10
11 function _interopRequireDefault(obj) { return obj && obj.__esModule ? obj : { default:
12   obj }; }
13
14 var multiply = function multiply(a, b) {
15   var total = 0;
16   for (var i = 0; i < b; i++) {
17     total = (0, _sum2.default)(a, total);
18   }
19   return total;
20 }; // multiply.js
21 exports.default = multiply;
```

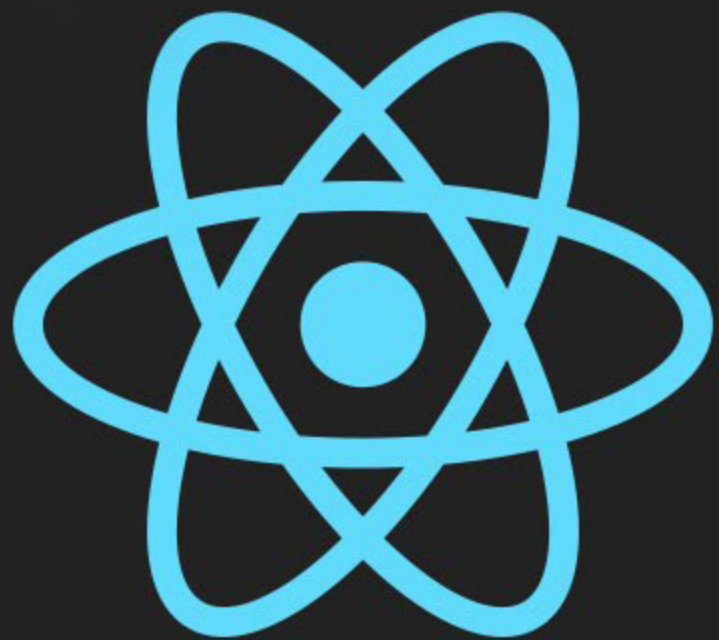
Configuration for React web development

- Setup React environment
- Polyfill JavaScript
- CSS Prefixer
- Dev Server
- build file for development and production
- Etc.

Create React App



Official. No Setup. Minimal.



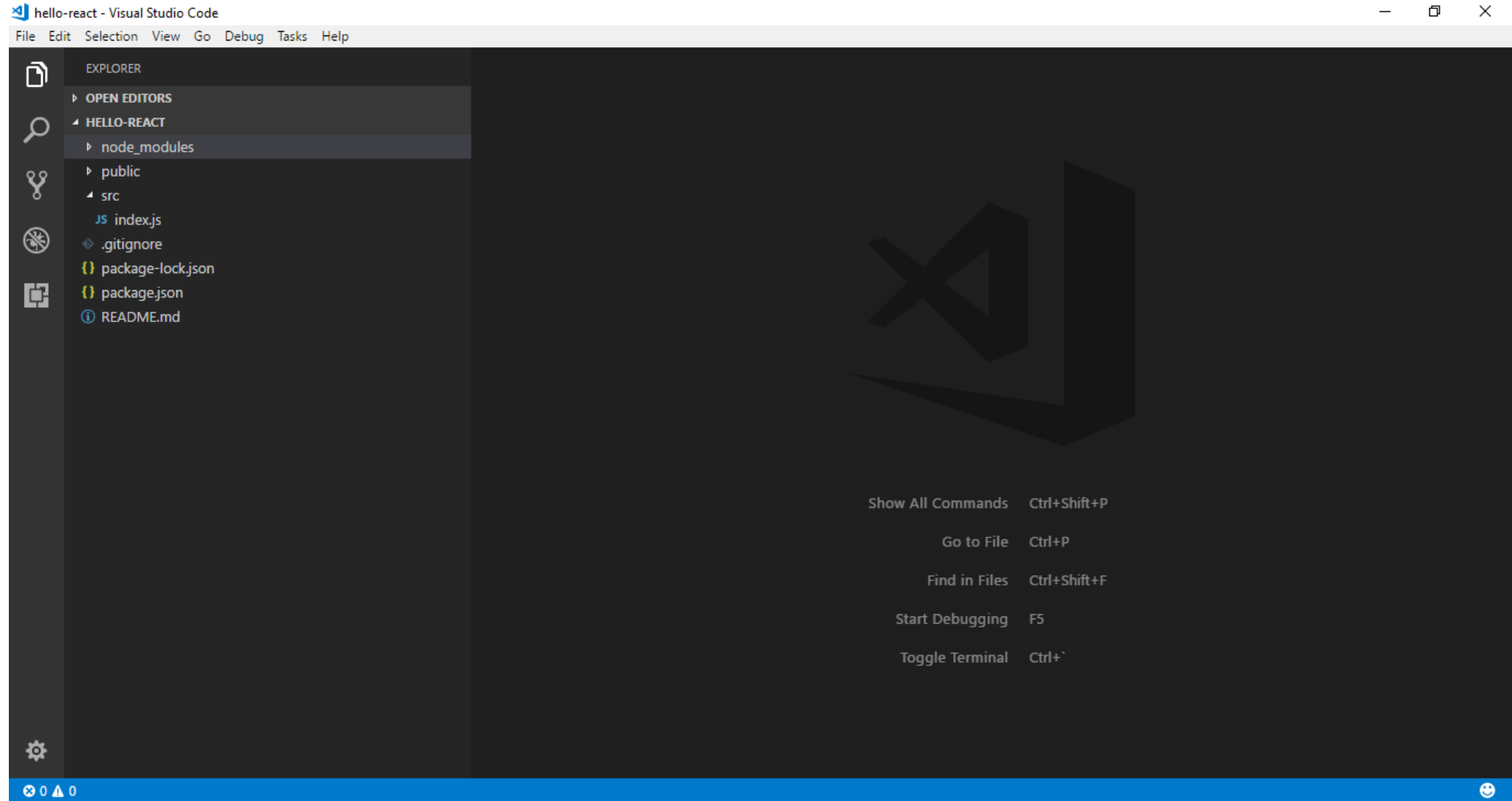
Hello World !
React

Hello World React

```
1 create-react-app hello-react  
2 cd hello-react  
3 npm start
```

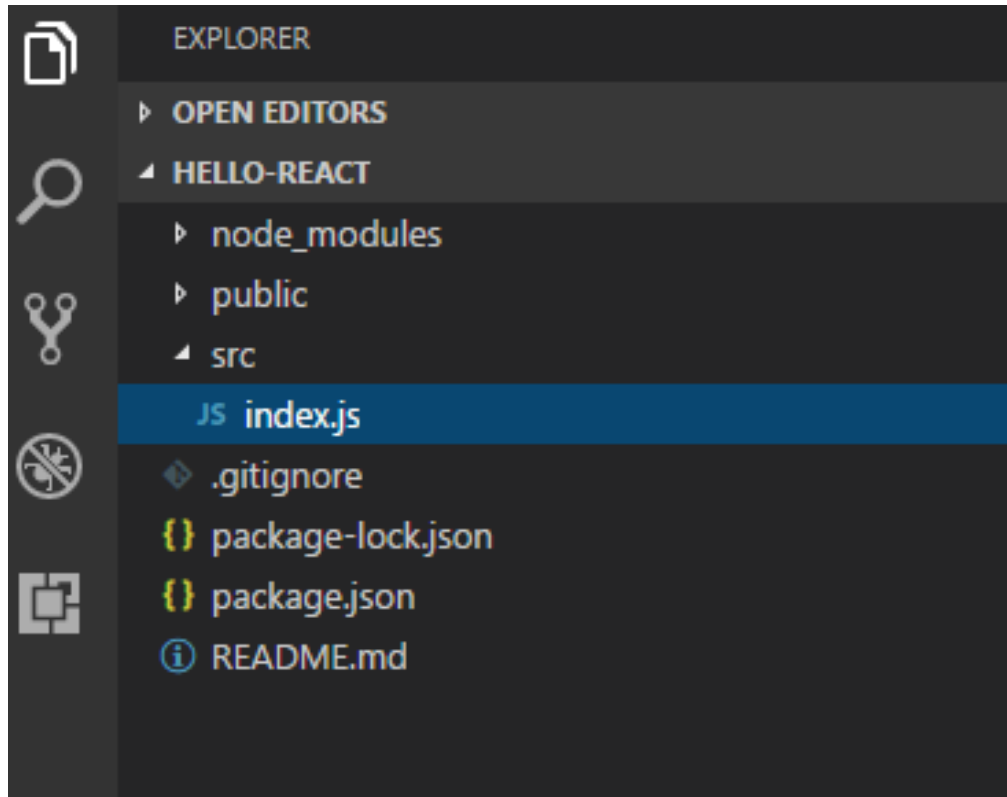
- Then open <http://localhost:3000/> to see your app.
- When you're ready to deploy to production, create a minified bundle with **npm run build**.

Open project in VS Code



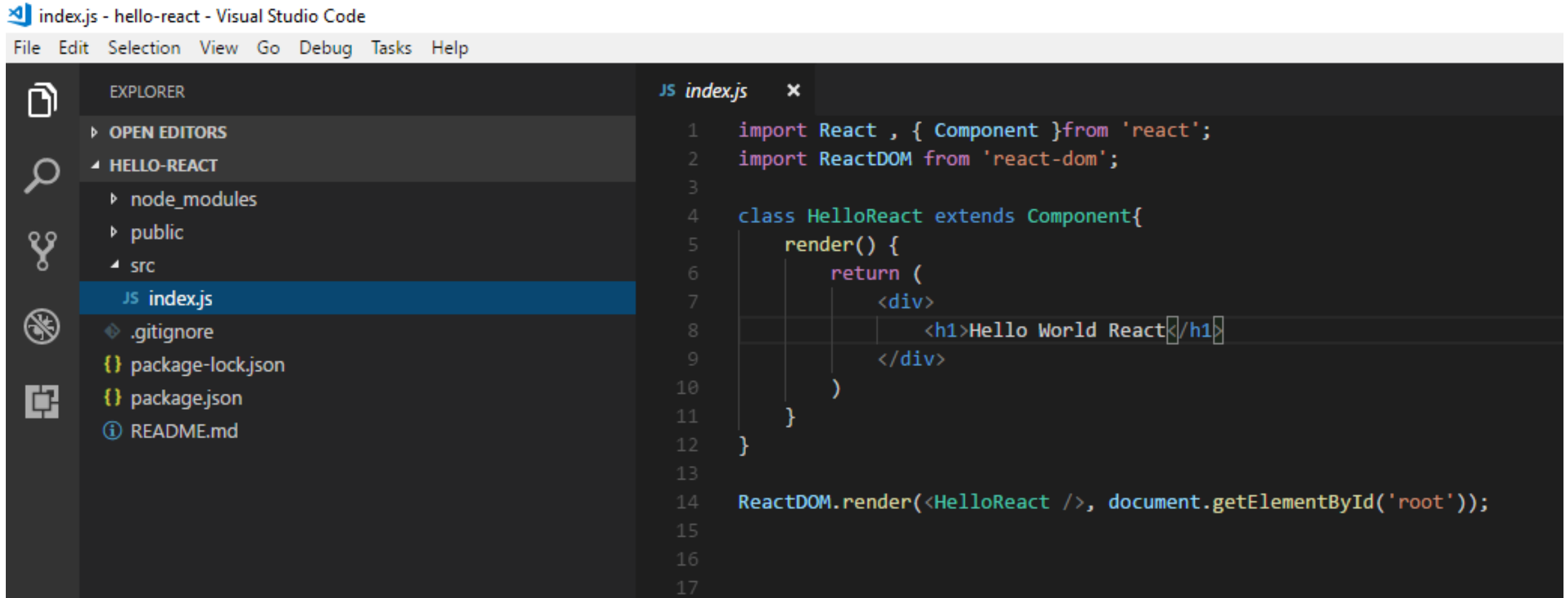
Step 1

- Delete all file in src folder after that it's like the picture below.



Step 2

- Edit code in file index.js as show below.

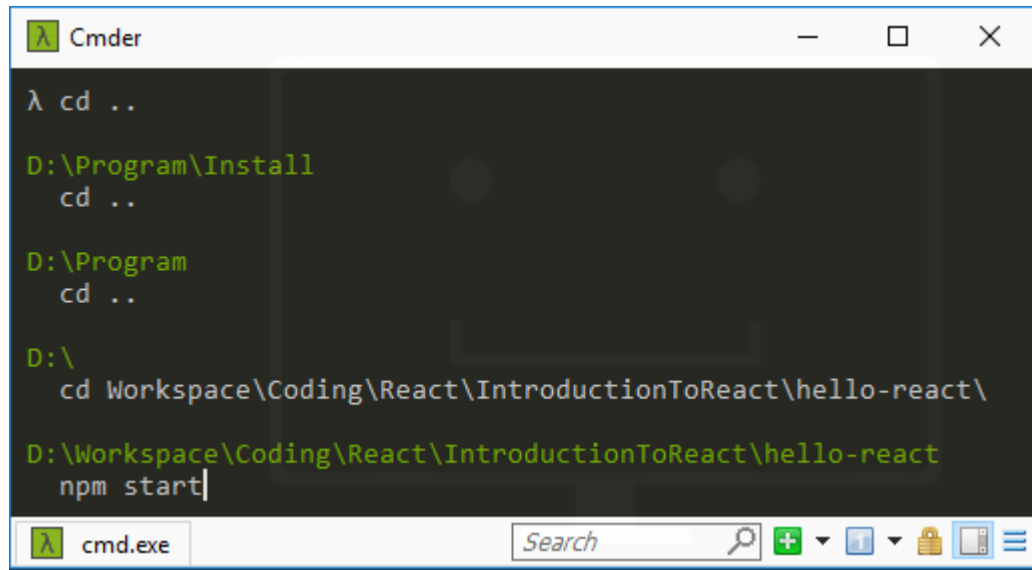


The screenshot shows the Visual Studio Code interface. The title bar reads 'index.js - hello-react - Visual Studio Code'. The menu bar includes 'File', 'Edit', 'Selection', 'View', 'Go', 'Debug', 'Tasks', and 'Help'. The Explorer sidebar on the left shows the project structure: 'HELLO-REACT' (expanded) contains 'node_modules', 'public', and 'src' (expanded). Inside 'src', the file 'index.js' is selected and highlighted in blue. Below 'src', other files are listed: '.gitignore', 'package-lock.json', 'package.json', and 'README.md'. The main editor area displays the code for 'index.js' with line numbers 1 through 17. The code is as follows:

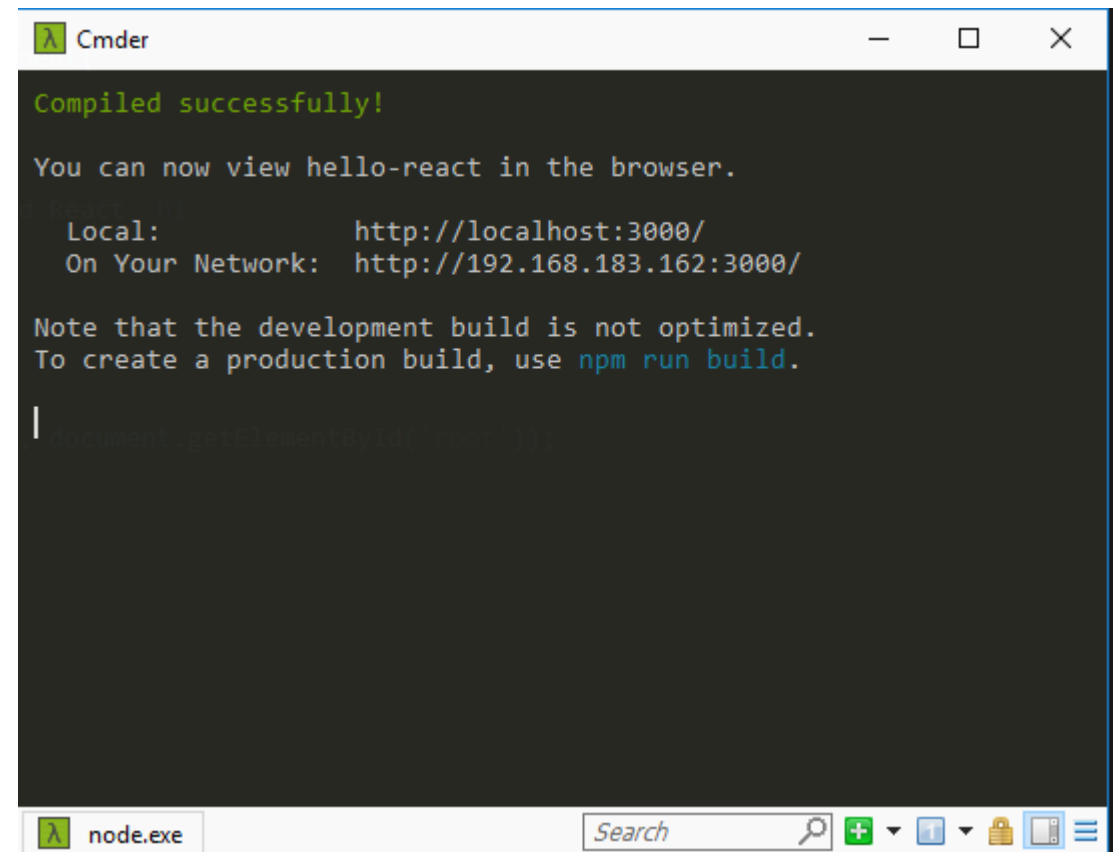
```
1  import React , { Component }from 'react';
2  import ReactDOM from 'react-dom';
3
4  class HelloReact extends Component{
5      render() {
6          return (
7              <div>
8                  <h1>Hello World React</h1>
9              </div>
10          )
11      }
12  }
13
14  ReactDOM.render(<HelloReact />, document.getElementById('root'));
```

Step 3

- Open Cmder and cd(Change Directory) to project path
- And use “ npm start ” for run project



```
λ cd ..  
D:\Program\Install  
cd ..  
D:\Program  
cd ..  
D:\  
cd Workspace\Coding\React\IntroductionToReact\hello-react\  
D:\Workspace\Coding\React\IntroductionToReact\hello-react  
npm start
```



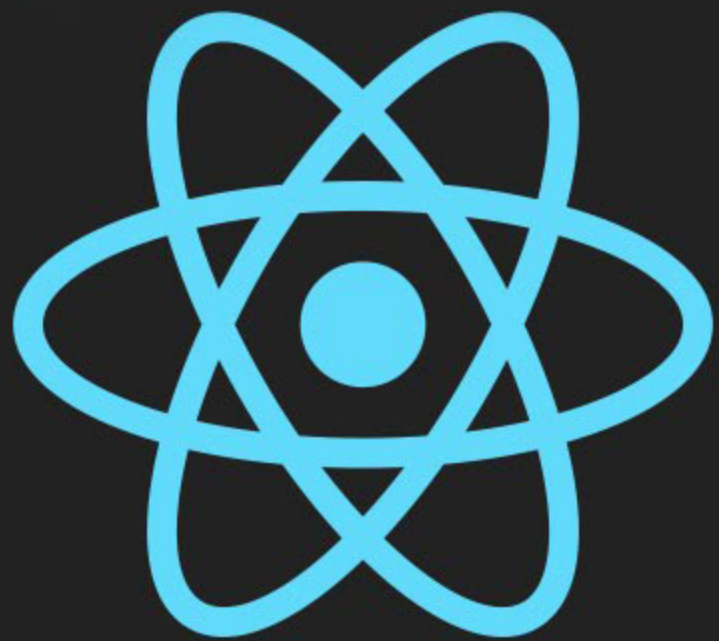
```
Compiled successfully!  
  
You can now view hello-react in the browser.  
  
Local: http://localhost:3000/  
On Your Network: http://192.168.183.162:3000/  
  
Note that the development build is not optimized.  
To create a production build, use npm run build.  
  
|
```

Step 4

- Open web browser at <http://localhost:3000/>



Hello React



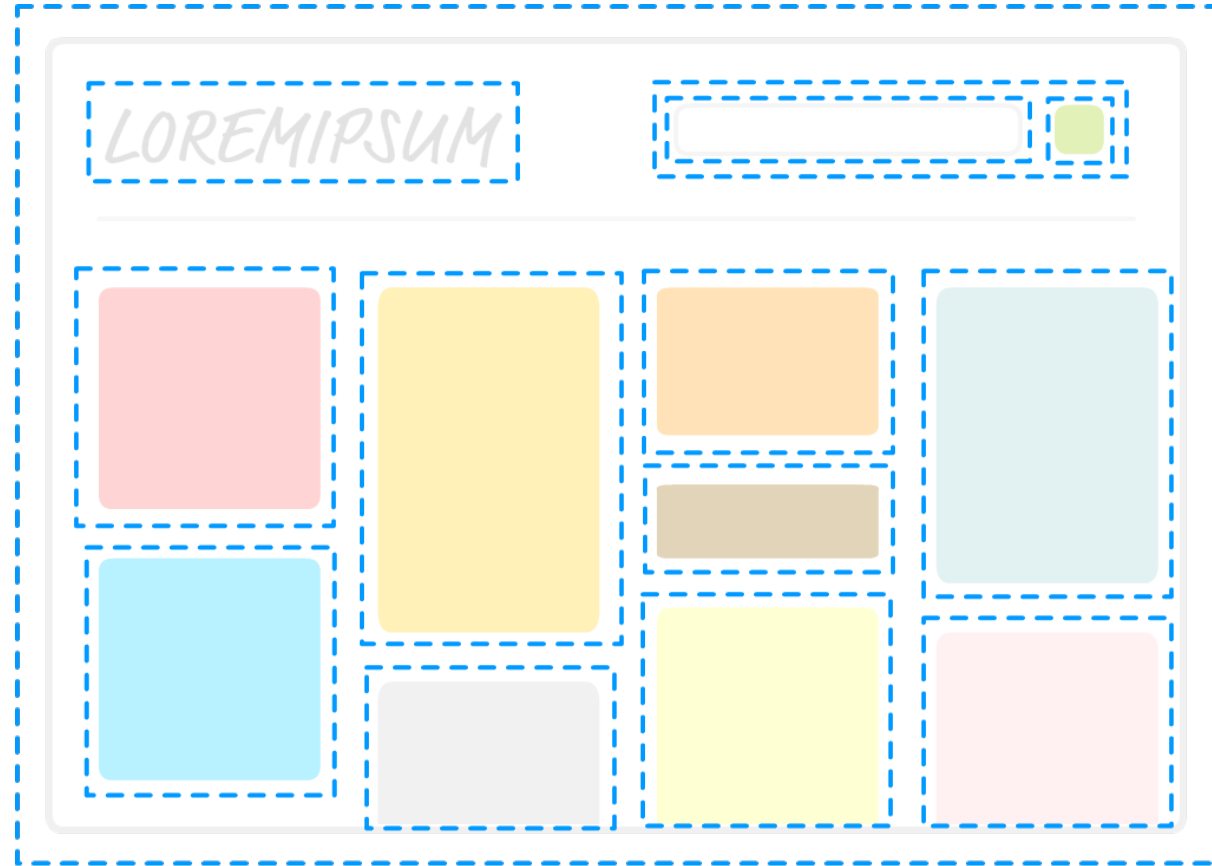
Component & JSX

Component

- **Component** Let you split the UI into independent , reusable pieces , and think about each piece in isolation.

React.Component is provide by React.

Component



That's a lot of COMPONENTS!

Create Component

```
class HelloReact extends Component {  
  render() {  
    return (  
      React.createElement(  
        'div',  
        null,  
        React.createElement('h1', null, 'Hello React')  
      )  
    )  
  }  
}
```

JSX : Javascript Syntax Expression

- **JSX** is javascript combined with XML for can write Tag in javascript.
- The file will have a .jsx extension.

```
class HelloReact extends Component {  
  render() {  
    return (  
      <div>  
        <h1>Hello React</h1>  
      </div>  
    )  
  }  
}
```

Create Component

Not used JSX

```
class HelloReact extends Component {  
  render() {  
    return (  
      React.createElement(  
        'div',  
        null,  
        React.createElement('h1', null, 'Hello React')  
      )  
    )  
  }  
}
```

Used JSX

```
class HelloReact extends Component {  
  render() {  
    return (  
      <div>  
        <h1>Hello React</h1>  
      </div>  
    )  
  }  
}
```

Reuse Component

- **App** , **SayReact** , **SayAngular** , **SayVue** is custom tag for reusable concepts.
- This tag for implementation in another page.

```
import React, { Component } from 'react';
import ReactDOM from 'react-dom';

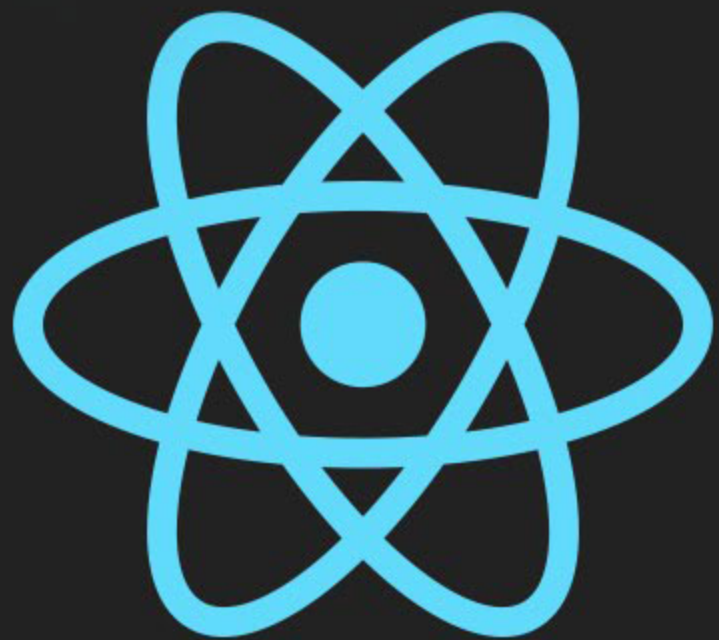
class SayReact extends Component {
  render() {
    return (
      <h1>Hello React</h1>
    )
  }
}

class SayAngular extends Component {
  render() {
    return (
      <h1>Goodbye Angular</h1>
    )
  }
}

class SayVue extends Component {
  render() {
    return (
      <h1>Miss U Vue</h1>
    )
  }
}

class App extends Component {
  render() {
    return (
      <div>
        <SayReact/>
        <SayAngular/>
        <SayVue/>
      </div>
    )
  }
}

ReactDOM.render(<App />, document.getElementById('root'));
```



State & Props

React Props/State Explained Through Darth Vader's Hunt for the Rebels

- If you've seen Star Wars, then you can understand props and state.

Here's a spoiler-free refresher of the basic premise of episodes 4–6:

1. Darth Vader hunts the rebels relentlessly, as they are the last resistance against the Galactic Empire.
2. The rebels, led by Princess Leia and Luke Skywalker, must fight back and exploit vulnerabilities within the Empire.
3. Darth Vader uses a variety of tactics to try and discover the movements of the rebels, including an army of Stormtroopers, a fleet of starships, and a variety of scouts.

The entire plan for the Empire's resources depends upon Vader's leadership.



- But in React, the idea is that when **state** is modified, the changes will **automatically trickle down** to all child components via **props**. So you only need to write the code to change one thing — the **state** — and watch as your UI updates.
- This is similar to the way that Darth Vader commands the three wings of his army. Once word gets back to him of the rebel location, his resources will automatically mobilize to launch an attack.

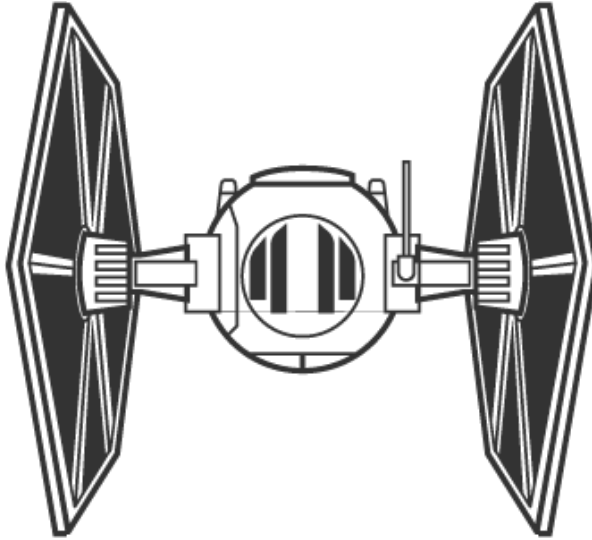


A Summary of The Galactic Empire

- Here are the three wings of the Galactic Empire.



The Imperial Army



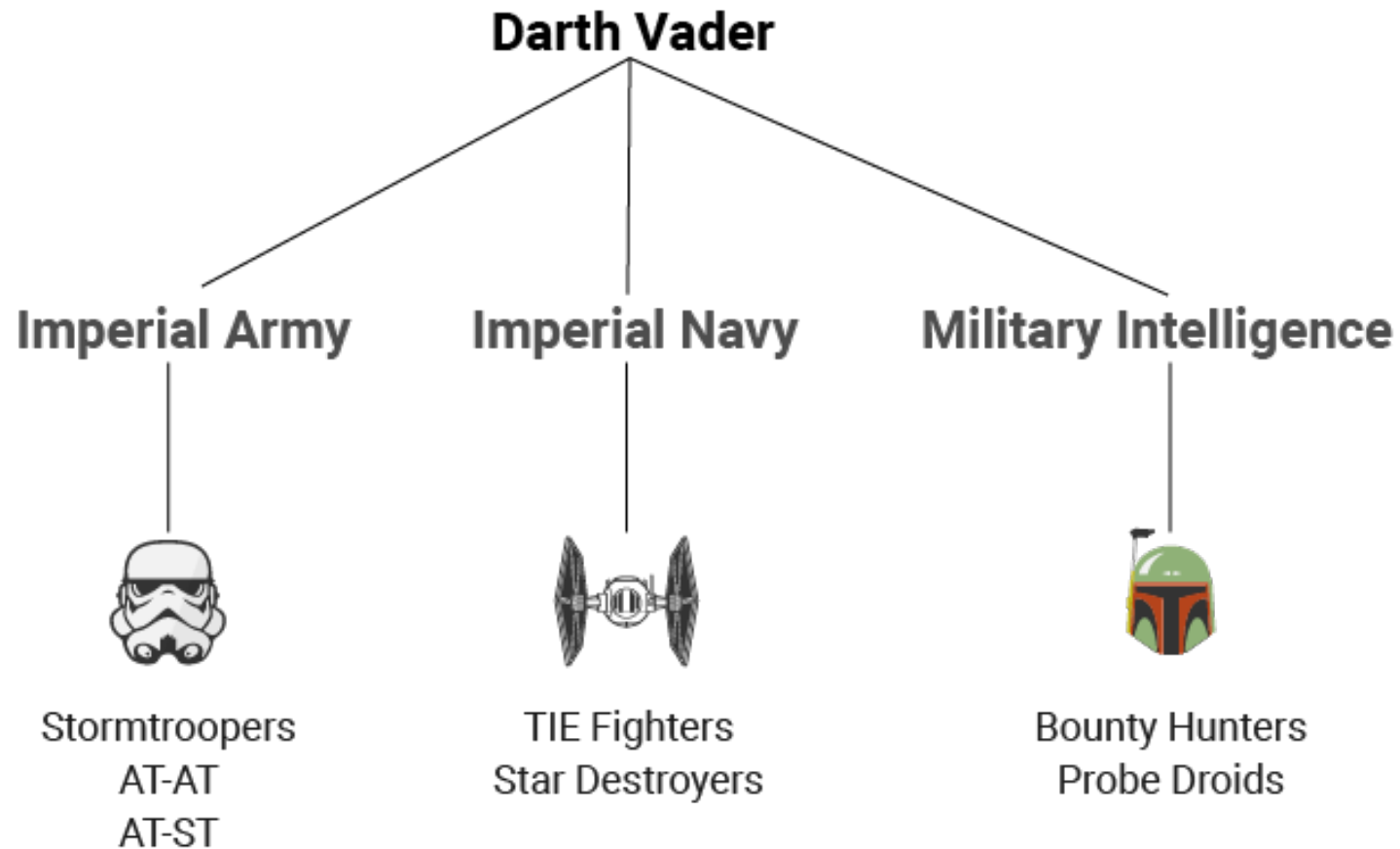
The Imperial Navy



Military Intelligence

- **The Imperial Army** is composed of Stormtroopers, AT-ATs, AT-STs and others.
- **The Imperial Navy** is composed of star destroyers, TIE fighters and others.
- **Military Intelligence** is composed of bounty hunters like Boba Fett, probe droids, and any other specialized scouts.

- Here is a quick org chart that will give some direction on how we will write our components.



- Stormtrooper encounters rebel base → Return to Darth Vader with location
- User clicks certain element → Update the state of some parent component

Here are the basics in code, which follows the org chart above: (Vaders Empire)

```
class vadersEmpire extends Component{
  state = {
    rebelLocation: ''
  };

  render() {
    return (
      <div>
        <ImperialArmy />
        <ImperialNavy />
        <MilitaryIntel />
      </div>
    );
  }
}
```



Here are the basics in code, which follows the org chart above: (Imperial Army)

```
class ImperialArmy extends Component{
  render() {
    return (
      <div>
        <StormTrooper />
        <ATAT />
        <ATST />
      </div>
    );
  }
}
```



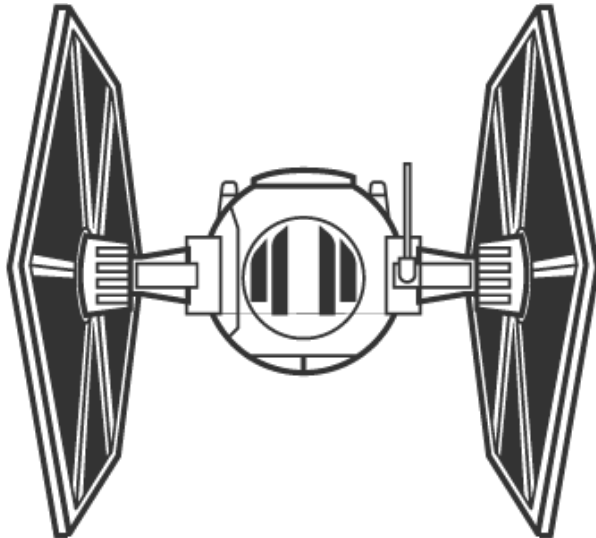
```
class StormTrooper extends Component{
  render() {
    return (
      <div></div>
    );
  }
}
```

```
class ATAT extends Component{
  render() {
    return (
      <div></div>
    );
  }
}
```

```
class ATST extends Component{
  render() {
    return (
      <div></div>
    );
  }
}
```

Here are the basics in code, which follows the org chart above: (Imperial Navy)

```
class ImperialNavy extends Component{  
  render() {  
    return (  
      <div>  
        <TIEFighter />  
        <StarDestroyer />  
      </div>  
    );  
  }  
}
```



```
class TIEFighter extends Component{  
  render() {  
    return (  
      <div></div>  
    );  
  }  
}
```

```
class StarDestroyer extends Component{  
  render() {  
    return (  
      <div></div>  
    );  
  }  
}
```

Here are the basics in code, which follows the org chart above: (Military Intelligence)

```
class MilitaryIntel extends Component{  
  render() {  
    return (  
      <div>  
        <BountyHunter />  
        <ProbeDroid />  
      </div>  
    );  
  }  
}
```



```
class BountyHunter extends Component{  
  render() {  
    return (  
      <div></div>  
    );  
  }  
}
```

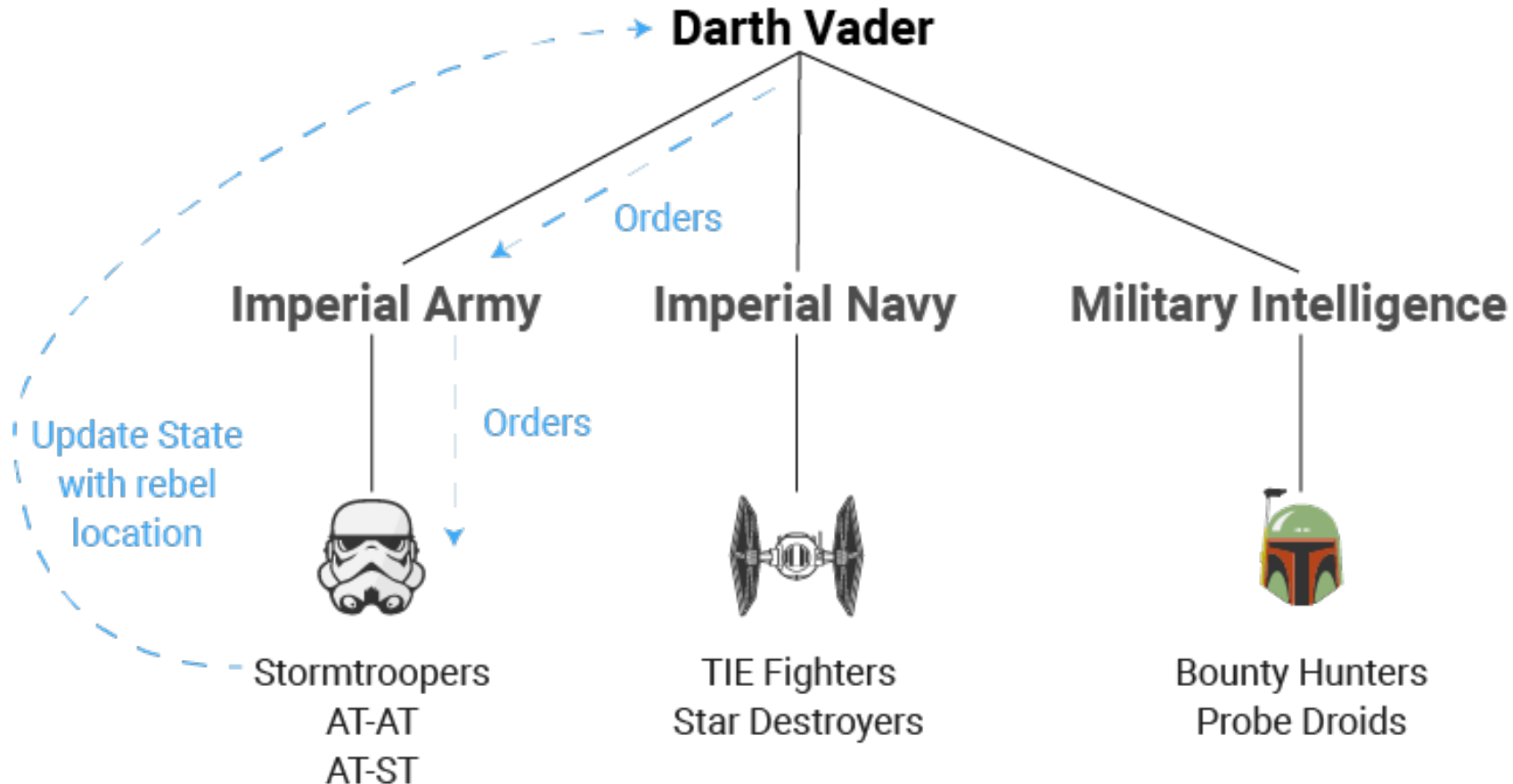
```
class ProbeDroid extends Component{  
  render() {  
    return (  
      <div></div>  
    );  
  }  
}
```

State

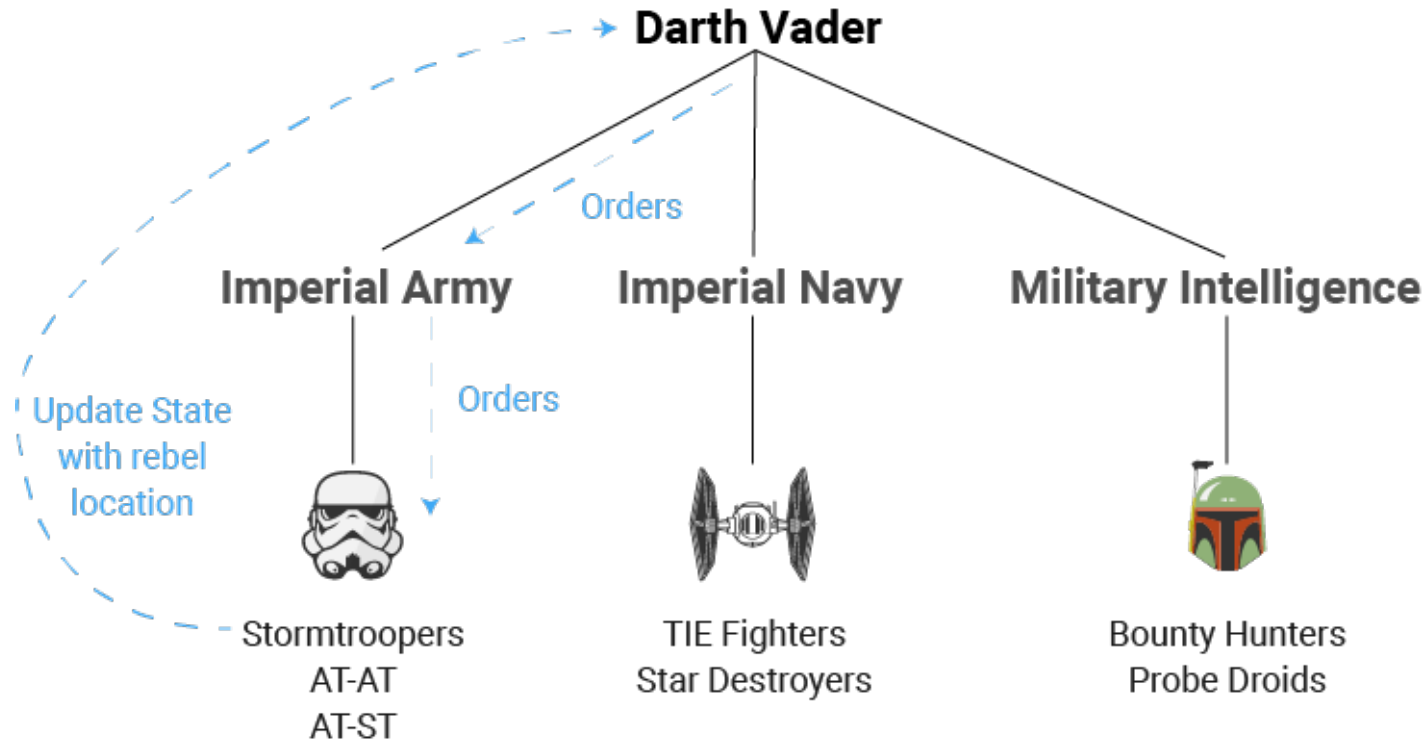
- **State** allows you to dynamically change many elements at once based on one variable. **State** encompasses the key parts of your UI that change basic on user input.
- With less things to keep track of in state, you will be able to write components with more clarity and fewer opportunities for bugs. When state changes, many components may change in accordance based on the one variable.

State in story

- Let's say Stormtroopers encounter the rebels. Vader has ordered them to report to him as soon as possible. Once they return with a rebel location, Vader can carry out the rest of his orders, which were contingent on the rebel location. Here is a modified diagram that charts the path through the components listed above.



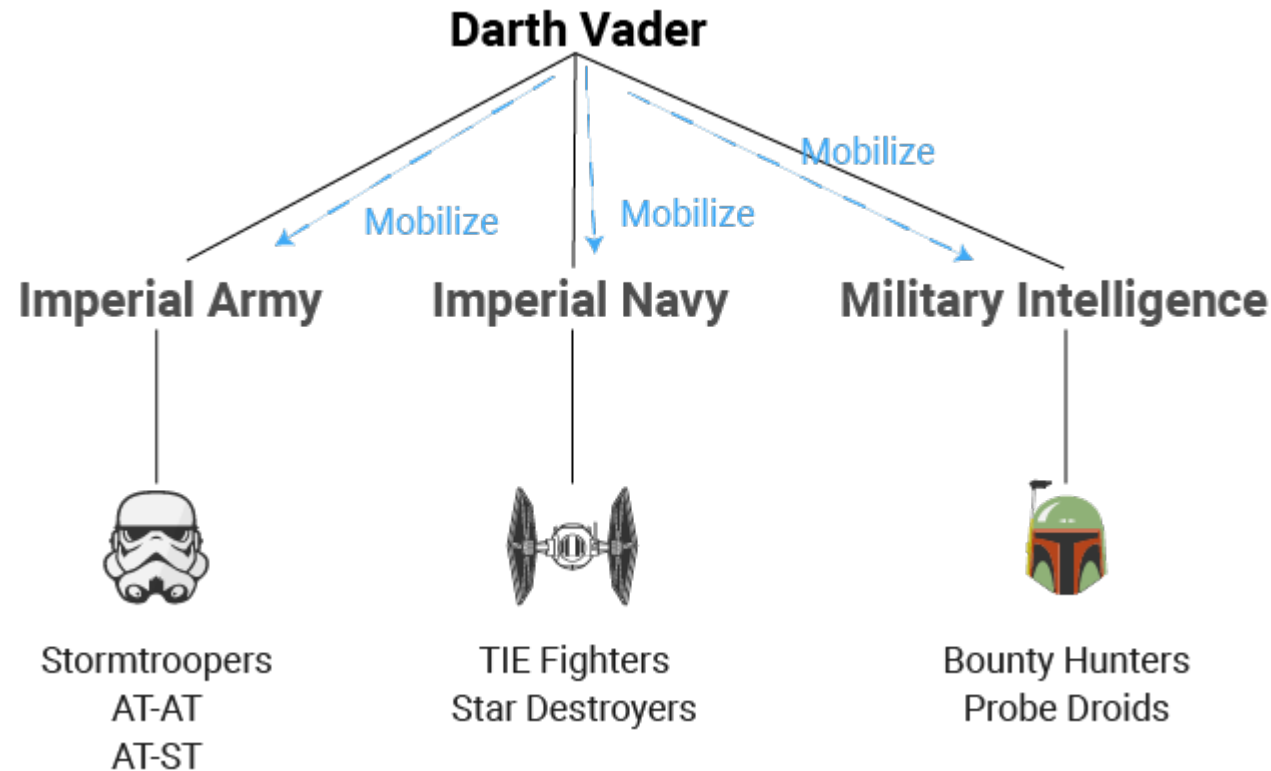
- Orders are already passed down to every member at the bottom of the chart. Once they run into rebels, they know to return to Lord Vader. The *rebelLocation* state will then be updated with the planet, be it “Endor”, “Hoth”, or somewhere else.



- **Above:** Stormtrooper nested within ImperialArmy nested within vadersEmpire
- **In a real app:** User input nested within parent div nested within parent div

What happens when state changes

- In this case, once the *rebelLocation* is discovered, the **state** would change to that planet. But that is only half the story. Darth Vader would have plans in mind to mobilize different assets based on this state change. He can prepare them in advance for this possibility. As in, “When we find their planet, travel there immediately and prepare for an assault!”
- Once state changes, the changes are automatically shared with all 3 wings of the Empire. Similarly, when the state of a parent component changes, the child components automatically inherit the new state.



- Every component can also have its own state. For example, the *ImperialArmy* component might have a *troopsCount* state which counts the members of the army. We will not modify that in this example, but you can imagine that a battle might affect *troopsCount*.
- Notice how this state does not depend on *rebelLocation*. If it did, we would not want to explicitly declare another state. We would want it to automatically update based on a change in *rebelLocation* state.
- Since it is independent, here is what the code looks like:

```
class ImperialArmy extends Component{  
  state = {  
    troopCount: 0  
  };  
  render() {  
    return (  
      <div>  
        <StormTrooper />  
        <ATAT />  
        <ATST />  
      </div>  
    );  
  }  
}
```

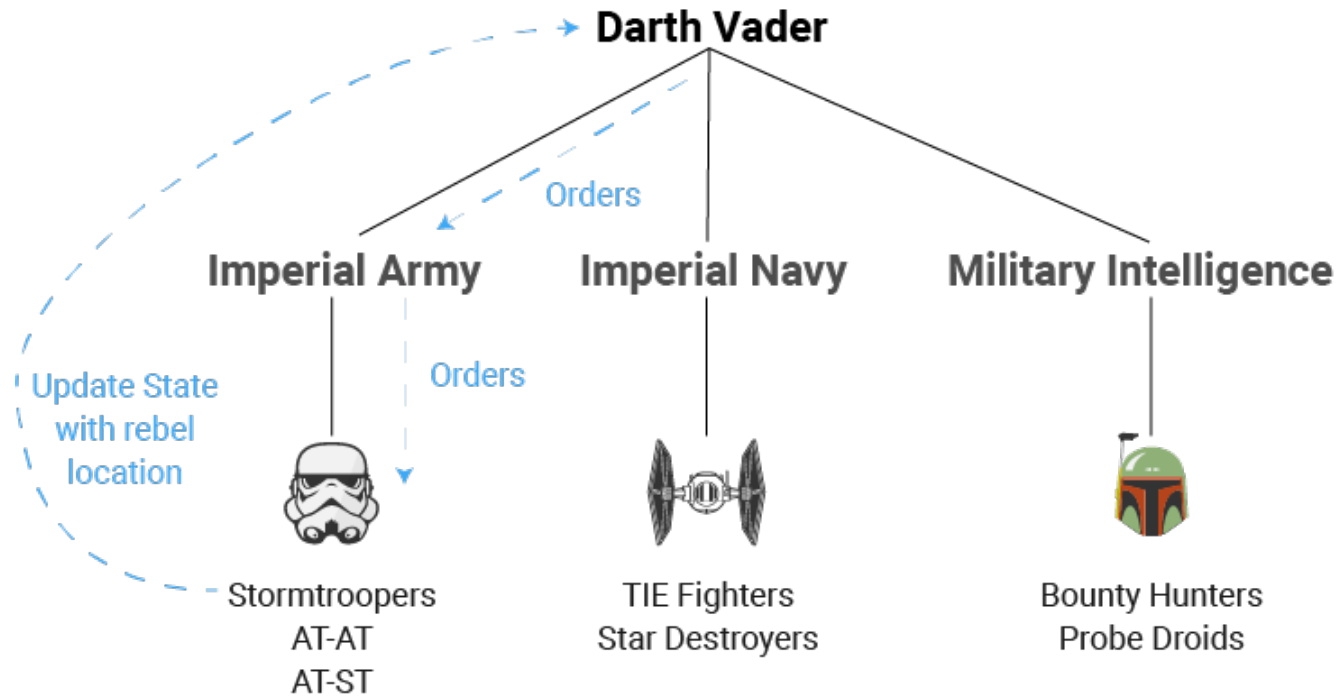


Wait, so how does this state get communicated between the different components? That brings us to... props!

Props

- With our Darth Vader case, we actually need two sets of instructions when it comes to commanding the Stormtroopers and other units on the bottom of our chart.
- **Question 1:** What should the Stormtroopers do if they encounter the rebels?
- **Answer:** Report back to Darth Vader.
- **Question 2:** Where should the Stormtroopers travel?
- **Answer:** *if* the rebels have not been found, search the galaxy at random. *Else*, go to the rebel location to attack them.
- **Props** allow us to continuously monitor the *rebelLocation* state, and order a troop movement if the state changes. *rebelLocation* is a string. But what about the orders that must happen when they initially find the rebels?

- We can actually pass a function as **props** as well! That means that we can pass a callback down to each Stormtrooper that will execute if that trooper discovers the target. In the following picture, you can follow the path outlined by “Orders” to trace the **props**.



- In a typical user interface, let's say that a user clicks a button, and you want to update the state of a parent component. You must also pass a callback from that parent component that will be triggered on the user's click. That callback can then update the state **because it originated with the same parent that set the state**.

- Here is what the code looks like:

```
class vadersEmpire extends Component {
  state = {
    rebelLocation: ''
  };

  reportRebelLocation(newLocation) {
    this.setState({
      rebelLocation: newLocation
    });
  };

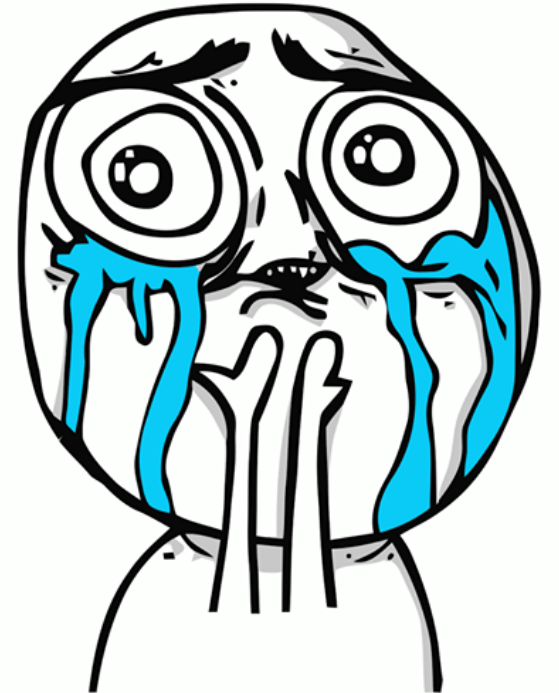
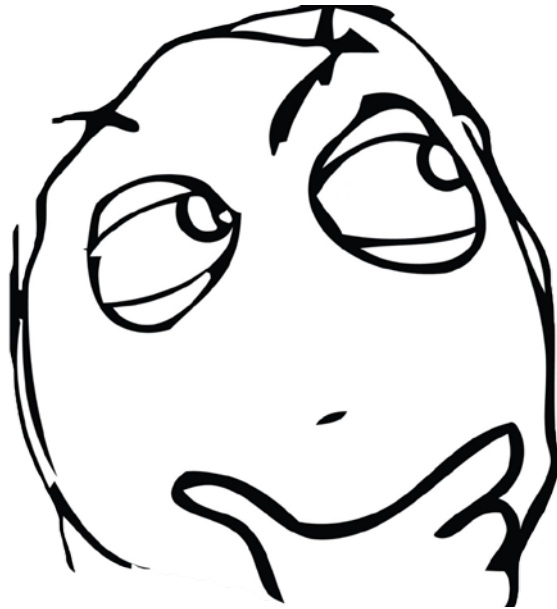
  render() {
    return (
      <div>
        <ImperialArmy
          rebelLocation={this.state.rebelLocation}
          updateLocation={this.reportRebelLocation}
        />
        <ImperialNavy />
        <MilitaryIntel />
      </div>
    );
  }
}
```

```
class ImperialArmy extends Component {
  render() {
    return (
      <div>
        <StormTrooper
          rebelLocation={this.props.rebelLocation}
          updateLocation={this.props.updateLocation}
        />
        <ATAT />
        <ATST />
      </div>
    )
  }
}

class StormTrooper extends Component {
  discoverLocation() {
    this.props.updateLocation(
      this.refs.secretBaseLocation.value
    );
  };
  render() {
    return (
      <input
        placeholder = {this.props.rebelLocation}
        value = ''
        onChange = {this.discoverLocation}
        ref="secretBaseLocation"
      />
    )
  }
}
```


Homework

1. Convert resume(html + css) from homework week2 to react.js
2. Push this homework to repository your self & repository subject



Reference

- <https://nodejs.org/en/>
- <https://www.npmjs.com/>
- <https://webpack.github.io/>
- <https://reactjs.org/>
- <https://devahoy.com/posts/getting-started-with-nodejs/>
- <https://stories.sellzuki.co.th/webpack-ทำงานยังไง-ไม่รู้ถือว่าบาป-7dd38131a78f>
- <https://medium.freecodecamp.org/react-props-state-explained-through-darth-vaders-hunt-for-the-rebels-8ee486576492>
- **React: Functional Web Development with React and Redux 1st Edition, Kindle Edition**