

# 3I017 - TECHNOLOGIES DU WEB

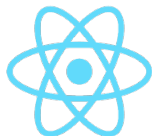
Les bases ReactJS

Thursday 14<sup>th</sup> March, 2019

Laure Soulier



# Contexte ReactJS



# React

- Librairie Javascript libre développée par Facebook en 2013
- Utilisé par : Airbnb, Bit.ly, Citymapper, Coursera, Dailymotion, Dropbox, Facebook, Housing.com, IMDb, Instagram, Netflix, Paypal, Tesla, Twitter, Walmart, Wordpress, Yahoo!
- et plein d'autres...  
<https://github.com/facebook/react/wiki/sites-using-react>

## Quelques stats...

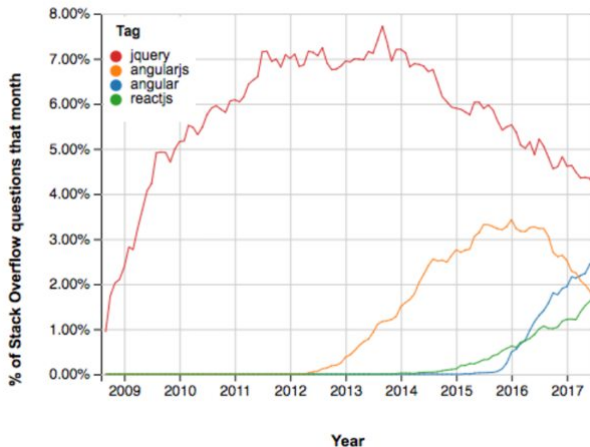


Figure 1: <https://stackoverflow.blog/2018/01/11/brutal-lifecycle-javascript-frameworks/>

# Quelques stats...

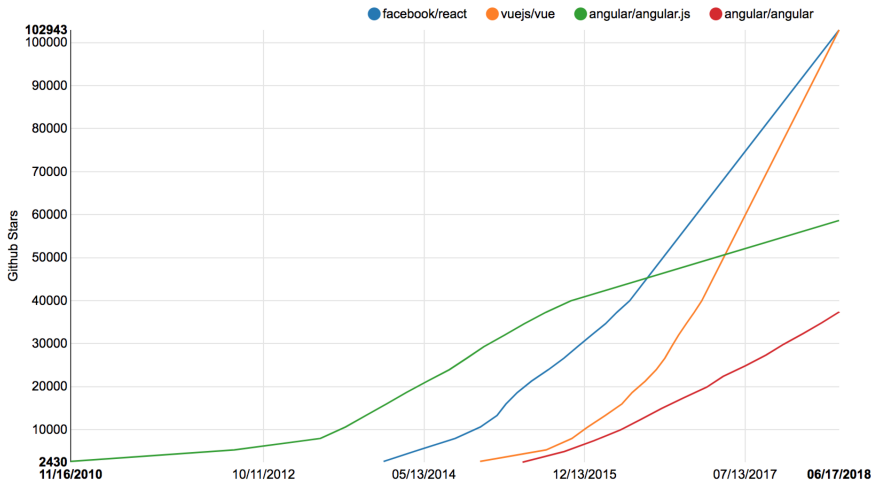


Figure 2: <https://zendev.com/2018/06/19/react-usage-beating-vue-angular.html>

## Quelques stats...

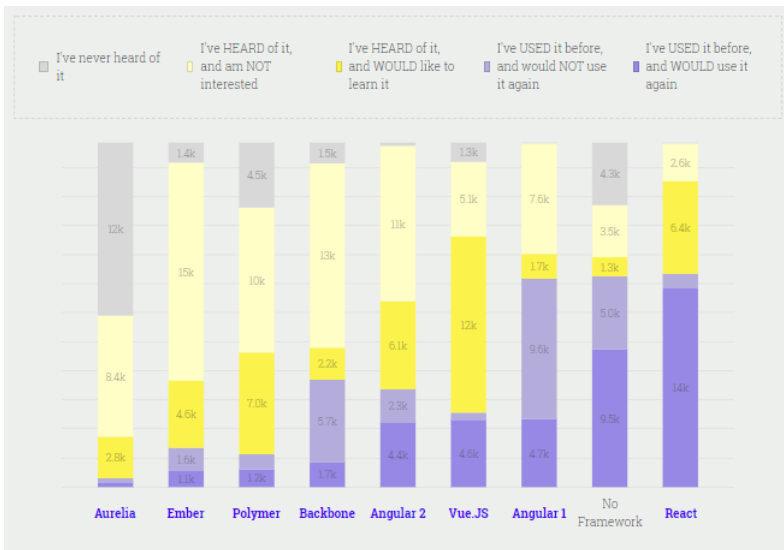


Figure 3: <https://hackernoon.com/the-status-of-javascript-libraries-frameworks-2018-beyond-3a5a7cae7513>

## Quelques stats...

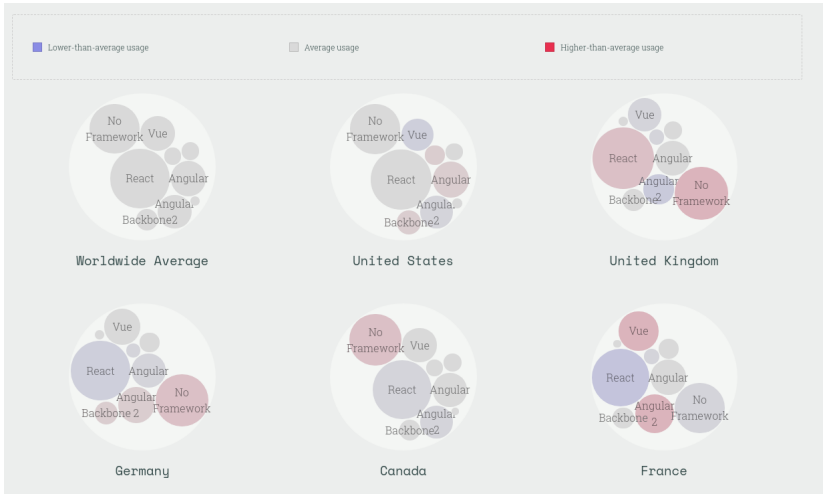


Figure 4: <https://2017.stateofjs.com/2017/front-end/worldwide/>

# ReactJS : une approche orientée composant

- Une page web est un arbre de composants dont les feuilles sont des balises html classiques.
- Un composant crée une représentation sous forme d'objets et de noeuds qui permet de générer du code HTML à chaque changement d'état
- Un composant ReactJS :
  - est défini par un "state", c'est à dire un état à l'instant t.
  - peut également posséder des propriétés en dehors de ses "states" qui peuvent être créées à la volée dans la classe ou bien passées en paramètres à l'instanciation du composant.
  - retourne un DOM virtuel en sortie

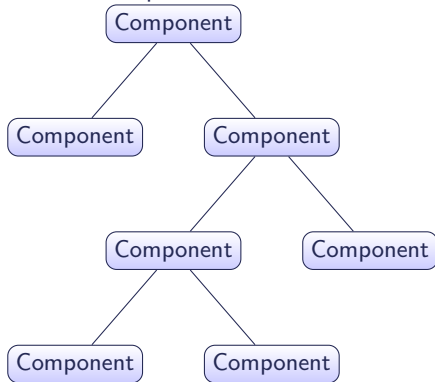
## Avantage

- Intégration facile dans une application existante (migration progressive d'une application)
- Réutilisabilité : "Learn Once, Write Anywhere"
- Si modification/erreur d'un composant, limité à ce composant, le reste de l'application fonctionne !



# ReachJS : Approche orientée composants

Arbre de composants ReactJS



Arbre DOM

```
<div>  
  <div>  
    toto  
  </div>  
</div>
```

# Une page web avec des composants

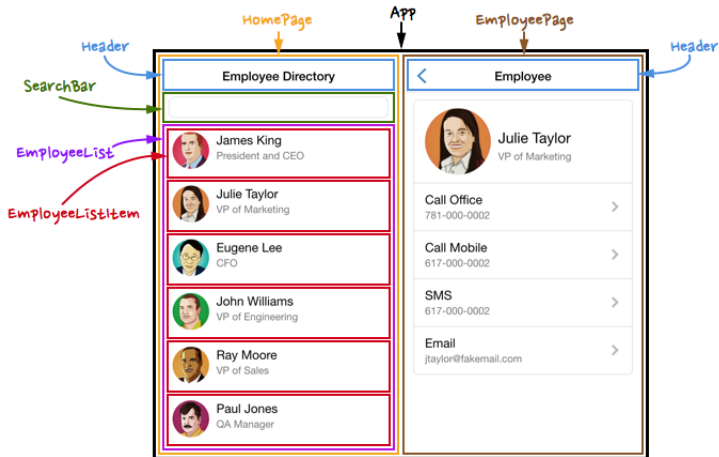


Figure 5: source :

<http://nitrajka.com/2016/08/15/setup-react-es6-redux-app-express/>

# ReactJS : Virtual DOM

- Modification d'un noeud du DOM demande de recharger l'ensemble de la page web (code HTML, CSS, javascript, ...) → cela prend du temps...
- Virtual DOM
  - Représentation mémoire de l'arbre DOM.
  - Il possède 2 représentations (état précédent et état courant)
  - Lors de la modification d'un composant, Virtual-DOM va prendre en compte sa représentation, la comparer avec le Virtual-DOM de l'état précédent et déduire les opérations minimales à exécuter pour que le DOM réel soit conforme au virtuel.
  - Les modifications du DOM sont faites par batch, une seule fois → on le "remplace".

*"rendering the Virtual DOM will always be faster than rendering a UI in the actual browser DOM"*

source [https:](https://www.accelebrate.com/blog/the-real-benefits-of-the-virtual-dom-in-react-js/)

[//www.accelebrate.com/blog/the-real-benefits-of-the-virtual-dom-in-react-js/](https://www.accelebrate.com/blog/the-real-benefits-of-the-virtual-dom-in-react-js/)

# ReactJS : Virtual DOM

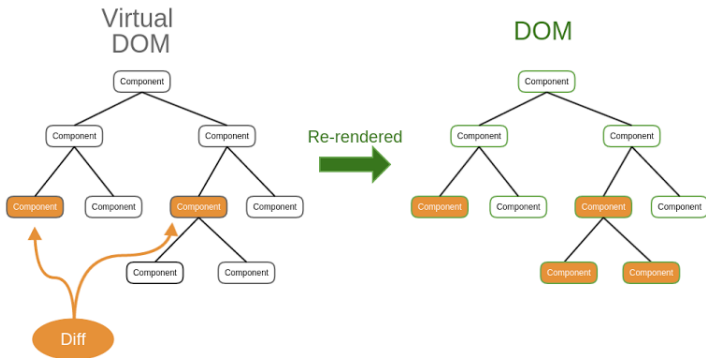


Figure 6: [source https://www.edureka.co/blog/what-is-react/](https://www.edureka.co/blog/what-is-react/)

# ReactJS : Single Page Application (SPA)

- Un seul point d'entrée pour le site web : index.html
- Ce sont les composants qui sont mis à jour au fur et à mesure des interactions



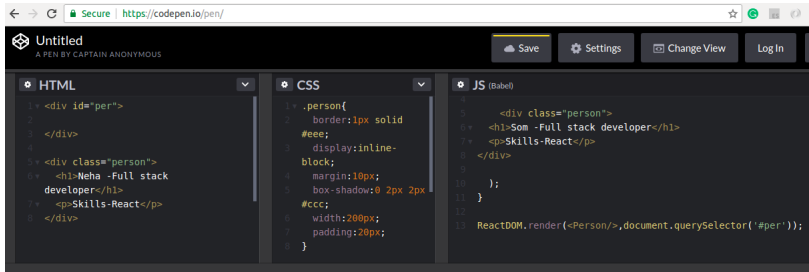
Figure 7: [https://www.kirupa.com/react/creating\\_single\\_page\\_app\\_react\\_using\\_react\\_router.htm](https://www.kirupa.com/react/creating_single_page_app_react_using_react_router.htm)

# Architecture de l'application ReactJS

```
.  
├─ README.md  
├─ package.json  
├─ public  
│   ├─ favicon.ico  
│   ├─ index.html  
│   └─ manifest.json  
├─ src  
│   ├─ App.css  
│   ├─ App.js  
│   ├─ App.test.js  
│   ├─ index.css  
│   ├─ index.js  
│   ├─ logo.svg  
│   └─ registerServiceWorker.js  
└─ yarn.lock
```

# Comment ça fonctionne en pratique

- Javascript "Next-Gen" : ES6
- Compilateur : Babel
- Outils de dépendance : npm ou yarn
- Utilise un serveur de développement



The screenshot shows a CodePen editor with three panels: HTML, CSS, and JS (Babel). The HTML panel contains a div with id="per" and a class="person" containing an h1 and a p. The CSS panel contains a .person class with various styles. The JS panel contains a Person component and a ReactDOM.render call.

```
HTML
1 <div id="per">
2
3 </div>
4
5 <div class="person">
6   <h1>Neha -Full stack
7     developer</h1>
8   <p>Skills-React</p>
9 </div>

CSS
1 .person{
2   border:1px solid
3   #eee;
4   display:inline-
5   block;
6   margin:10px;
7   box-shadow:0 2px 2px
8   #ccc;
9   width:200px;
10  padding:20px;
11 }

JS (Babel)
1
2 <div class="person">
3   <h1>Som -Full stack developer</h1>
4   <p>Skills-React</p>
5 </div>
6
7 );
8 }
9
10
11
12
13 ReactDOM.render(<Person/>,document.querySelector('#per'));
```

**Som -Full  
stack  
developer**

Skills-React

Ma première application



# Ma première application React

## index.html

```
<!doctype html>
<html>

<head>
  <meta charset="utf-8">
    <title>Hello React!</title>
    <script src="https://unpkg.com/react@16/umd/
      react.development.js"></script>
    <script src="https://unpkg.com/react-dom@16/umd/
      react-dom.development.js"></script>
    <script src="https://unpkg.com/
      babel-standalone@6.26.0/babel.js"></script>
</head>

<body>
  <div id="root"></div>
    // React code
</body>
</html>
```

# Ma première application React

index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import * as serviceWorker from './serviceWorker';

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

# Ma première application React

## App.js

```
class App extends Component {  
  render() {  
    return (  
      <div className="App">  
        <h1>Hello World!</h1>  
      </div>  
    );  
  }  
}
```

- JSX est un langage à balises, mais ni du HTML, ni du XML :
  - `<div />` à la place de `React.createElement('div')`
  - `<div className="shopping-list">` à la place de `React.createElement("div", {className: "shopping-list"})`

## Exemple

```
// Avec JSX
// Person.js
class Person extends Component {
  constructor(props) {
    super(props);
  }
  render() {
    return <div>
      Mon nom est {props.name} et j'ai {props.age}
    </div>
  }
}
// methode render() du App.js
<Person name="John" age="10 ans" />

// sans JSX
// Person.js : class, constructeur, render()
// methode render() du App.js
React.createElement(Person, { name: 'John', age: '10 ans' })
```

## JSX - quelques spécificités

- Formatage des valeurs :

### String ("blabla") vs. autres types/expressions JSX ({42})

```
<input
  type="nom"
  name="nom"
  maxLength={30}
  readOnly={false}
  onChange={this.handleFieldChange}
  value={this.state.value}
/>
```

- Attributs booléens

### pas besoin de spécifier true

```
<input type="nom" name="nom" autoFocus required=false />
```

- Attribut className : class est réservé pour la déclaration de la classe

### className vs. class

```
<div className="Person" />
```

## JSX - quelques spécificités

- Importance de la casse :
  - Sensible à la casse : `nomPersonne`  $\neq$  `nompersonne`
  - Pour les noms d'éléments :  
Si l'élément démarre par une majuscule, on estime qu'il s'agit d'un composant React ; sinon il s'agit d'un élément natif fourni par la plate-forme

pas besoin de spécifier `true`

```
[
  <CoolComponent/>,
  <coolComponent/>,
]
// donne :
[
  React.createElement(CoolComponent, null),
  React.createElement('coolComponent', null),
]
```

## Exercice

Ecrire le code JSX généré par le code React suivant :

```
React.createElement("div", {className: "shopping-list"},  
  React.createElement("h1", null, "Shopping List for ", props.name),  
  React.createElement("ul", null, React.createElement("li", null, "Instagram"),  
    React.createElement("li", null, "WhatsApp"),  
    React.createElement("li", null, "Oculus")));
```

## Exercice

Ecrire le code JSX généré par le code React suivant :

```
React.createElement("div", {className: "shopping-list"},
  React.createElement("h1", null, "Shopping List for ", props.name),
  React.createElement("ul", null, React.createElement("li", null, "Instagram"),
    React.createElement("li", null, "WhatsApp"),
    React.createElement("li", null, "Oculus")));
```

## Réponse

```
<div className="shopping-list">
  <h1>Shopping List for {props.name}</h1>
  <ul>
    <li>Instagram</li>
    <li>WhatsApp</li>
    <li>Oculus</li>
  </ul>
</div>;
```



# Les composants ReactJS

# Les composants

- Un composant peut posséder des propriétés qui peuvent être créées à la volée dans la classe ou bien passées en paramètres à l'instanciation du composant : **this.props**
- retourne un DOM virtuel en sortie : **render()**

## Exemple

```
// index.js
ReactDOM.render(<App name="toto"/>, document.getElementById('root'));

// App.js
class App extends Component {

  constructor(props) {
    super(props);
  }

  render() {
    return (
      <div className="App">
        <p>
          Hello {this.props.name}
        </p>
      </div>
    );
  }
}
```

# Les propriétés

- En lecture seule pour le composant courant
- Accéder au contenu (texte ou balises filles) d'une balise

## Exemple

```
{props.children} ou {this.props.children}
```

- Définition du type des propriétés

## Exemple

```
Person.propTypes = {  
  name : PropTypes.string ,  
  age : PropTypes.number.isRequired  
}
```

# L'état du composant

- rappel : `this.props` non mutable
- `this.state` : état d'un composant qui peut être modifié et permet de diffuser les attributs dans l'arbre de composants
- Etat accessible que depuis une classe qui étend un `Component`

## Exemple

```
class NewPost extends Component {  
  state = {  
    name: "toto"  
  };  
  
  render () {  
    return (  
      <Person name={this.state.name}></Person>  
    );  
  }  
}  
  
// {this.state.name} recupere par les props du composant
```

# Modification de l'état

- utiliser obligatoirement `this.setState()`

## Exemple

```
class Board extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      squares: Array(9).fill(null),
    };
  }

  handleClick(i) {
    const squares = this.state.squares.slice();
    squares[i] = 'X';
    this.setState({squares: squares});
  }

  renderSquare(i) {
    return (
      <Square
        value={this.state.squares[i]}
        onClick={() => this.handleClick(i)} />
    );
  }
}
```

## Deux façons d'écrire les composants

- Composants fonctionnels (sans état)

### Exemple

```
const cmp = () => { return <div>some JSX</div> }
    ou
const cmp=(props)=>{return <div> {props.name}</div>}
```

- Composant basé sur des classes (containers avec état)

### Exemple

```
class Cmp extends Component {
  render () {
    return <div>some JSX</div>
  }
}
!! quand props, on utilise this.props.names
```

→ La distinction permet de savoir les composants qui servent juste à faire de l'affichage (composants fonctionnels) et ceux qui vont être plus interactifs avec les données (avec état)

## Attributs des composants

- Attributs de type valeur : `<Person className="person" id=this.state.idP />`
- Attributs de type méthode : `<Person click=this.mamethode().bind(this, arg1, arg2, ... />`
- Si on souhaite récupérer les valeurs de formulaire et exécuter une méthode qui traite les données saisies dans le formulaire :
  - Balise de formulaire : `<input type="text" onChange=this.changed)/>`
  - Méthode de traitement : `changed = (event) => .... event.target.value ...`

## Gestion des listes

- Soit un état caractérisé par un tableau de personnes :

```
state = {personnes:[{nom : 'toto '};{nom : 'tata }]}
```

- Pour afficher une liste de personnes dans la méthode `render()` :

```
this.state.personnes.map(personne => {  
    return <Personne name={personne.nom} />;  
})
```

- Pour faire une copie d'un tableau (sinon modification dynamique du tableau) :

```
const personnes = [...this.state.personnes]
```



# React et style CSS

## Pour ajouter du style à vos composants

- Méthode "classique" : fichier CSS indépendant

### Architecture

```
|_ Component.css  
|_ Component.js  
  
// dans Component.js :  
import './Component.css'
```

## Pour ajouter du style à vos composants (1/3)

- Méthode d'intégration dans le code React
  - Définir une constante style avec les paires de clés-valeurs
  - Faire appel à ce style dans les balises React

### Exemple

```
render(){  
  const style = {color : 'red', margin : '1px'}  
  
  return <h1 style={style}> Titre </h1>;  
}
```

## Pour ajouter du style à vos composants (2/3)

- Dynamicité du code CSS

### Exemple

```
render(){
  const classes = [];
  if (this.state.nom="toto"){
    classes.push('person ');
  } else{
    classes.push('children ');
  }

  return <p className={classes}> this is the text</p>;
}
```

## Pour ajouter du style à vos composants (3/3)

- Gérer les évènements avec le code CSS
  - Librairie Radium

### Exemple avec l'évènement "hover"

```
import Radium from 'radium';

class Person extends Component{

  render(){
    const style = {color : 'red', margin : '1px',
      ":hover": {
        color: "green",
        cursor: "pointer"
      }
    }
    return <p style={style}> this is a text </p>;
  }
}

export default Radium(Person);
```

## Pour ajouter du style à vos composants (3/3)

- Pour aller plus loin...
  - Media Query :  
<https://gist.github.com/nickpiesco/9bef21b4f9e236b4430e>
  - CSS module in React :  
[https://medium.com/nulogy/  
how-to-use-css-modules-with-create-react-app-9e44bec2b5c2](https://medium.com/nulogy/how-to-use-css-modules-with-create-react-app-9e44bec2b5c2)  
<https://github.com/css-modules/css-modules>


React interactif

# Affichage conditionnel

- Dans la méthode render, il est possible d'afficher du contenu de façon conditionnelle. (pour rappel, on est en site mono-page) :  
`this.state.pageProfil ? <div> .... </div> : null`



# React interactif

- possibilité de rajouter des évènements dans les balises JSX : onClick, ...
- Notation : onClick={() => alert('click')}
-  onClick=alert('click') sans => va déclencher l'évènement à chaque exécution du "render".

## Exemple

```
// Avec JSX
// Person.js
class Person extends Component {
  constructor(props) {
    super(props);
  }
  render() {
    return <div onClick={() => alert('click')}>
      Mon nom est {props.name} et j'ai {props.age}
    </div>
  }
}
```

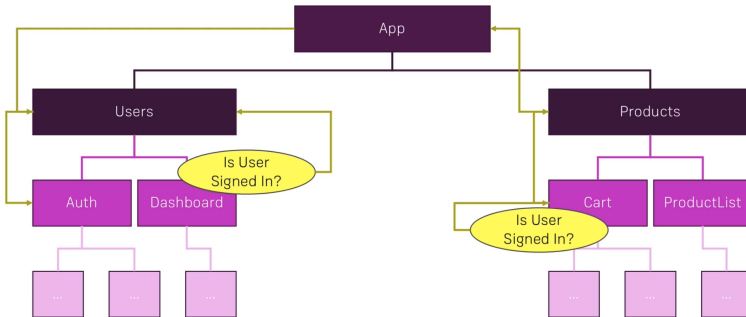
# React interactif

- Clipboard Events : onCopy onCut onPaste
- Composition Events : onCompositionEnd onCompositionStart onCompositionUpdate
- Keyboard Events : onKeyDown onKeyPress onKeyUp
- Focus Events : onFocus onBlur
- Form Events : onChange onInput onInvalid onSubmit
- Mouse Events : onClick onContextMenu onDoubleClick onDrag onDragEnd onDragEnter onDragExit onDragLeave onDragOver onDragStart onDrop onMouseDown onMouseEnter onMouseLeave onMouseMove onMouseOut onMouseOver onMouseUp
- Media/Image Events : onLoad onAbort onCanPlay onCanPlayThrough onDurationChange onEmptied onEncrypted onEnded onError onLoadedData onLoadedMetadata onLoadStart onPause onPlay onPlaying onProgress onRateChange onSeeked onSeeking onStalled onSuspend onTimeUpdate onVolumeChange onWaiting
- Mouse Events : onScroll
- Animation Events : onAnimationStart onAnimationEnd onAnimationIteration

Redux

# Redux

- Les états peuvent être compliqués à gérer entre les composants



## Références

- <https://blog.octo.com/pourquoi-sinteresser-a-react/>
- create-react-app:  
<https://github.com/facebookincubator/create-react-app>
- Introducing JSX: <https://reactjs.org/docs/introducing-jsx.html>
- Rendering Elements:  
<https://reactjs.org/docs/rendering-elements.html>
- Components Props:  
<https://reactjs.org/docs/components-and-props.html>
- Listenable Events: <https://reactjs.org/docs/events.html>