

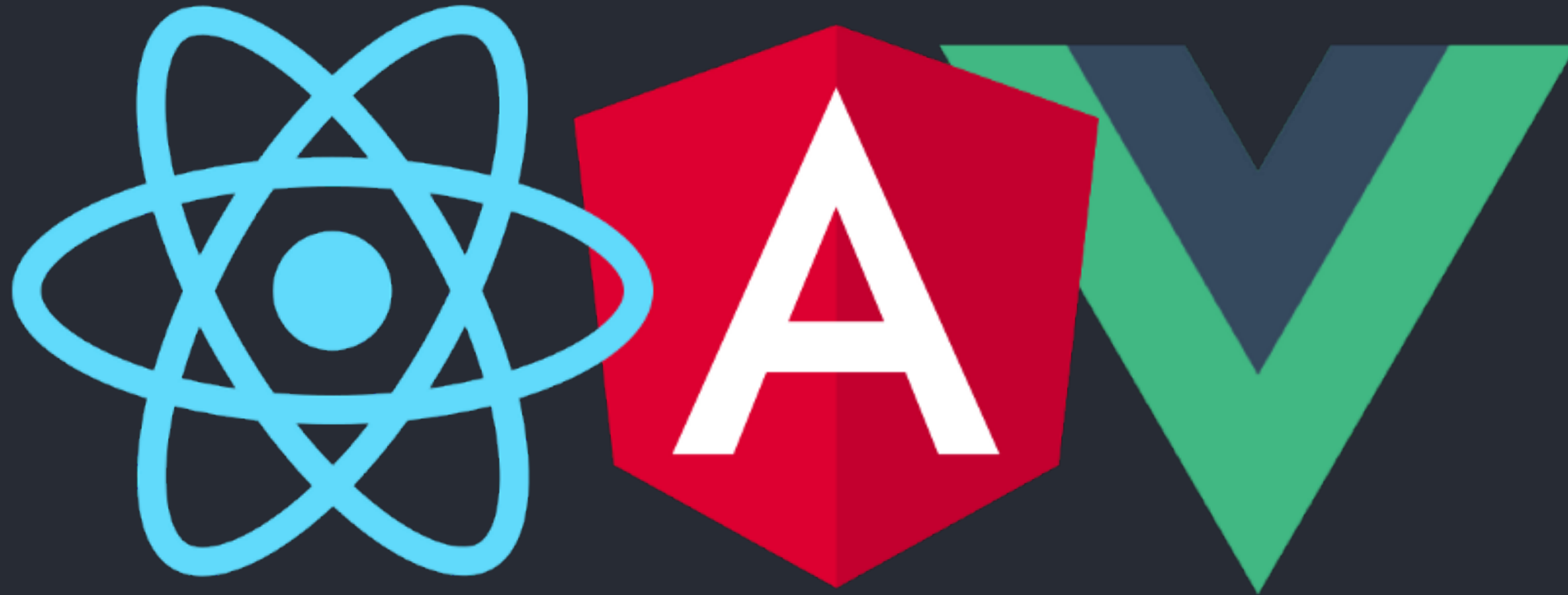
# React Introduction

송민석 @ 2019-08-09

# 자바스크립트



# 잘 나가는 프론트엔드(?) 프레임워크



페이스북

구글

Evan You

# 왜 React?

- 내가 관심이 많아서 \*^^\*
- 가장 큰 커뮤니티
- 페이스북
- 그러면 Vue는? Angular는? -> 그것도 다 좋아....

# 리액트가 뭔데?

- <https://reactjs.org> (<https://ko.reactjs.org>)
- 사용자 인터페이스를 만들기 위한 JavaScript 라이브러리
  - “난 View만 책임진다. 나머진 니가 알아서 해....!!! 😊”
- 때려줄까? 👊
- react-router, redux-react, ...
- 선언형
- 컴포넌트기반
- 한 번 배워서 어디서나 사용하기 (Electron, React Native, ...)



ELECTRON



React Native

# 선언형, 컴포넌트

```
class HelloMessage extends React.Component {  
  render() {  
    return (  
      <div>  
        Hello {this.props.name}  
      </div>  
    );  
  }  
}
```

```
ReactDOM.render(  
  <HelloMessage name="Taylor" />,  
  document.getElementById('hello-example')  
);
```

# 컴포넌트

```
class Timer extends React.Component {
  state = { seconds: 0 };

  tick() {
    this.setState(state => ({
      seconds: state.seconds + 1
    }));
  }

  componentDidMount() {
    this.interval = setInterval(() => this.tick(), 1000);
  }

  componentWillUnmount() {
    clearInterval(this.interval);
  }

  render() {
    return (
      <div>
        Seconds: {this.state.seconds}
      </div>
    );
  }
}

ReactDOM.render(
  <Timer />,
  document.getElementById('timer-example')
);
```



# 리액트 해 보자

- Node 설치 (<https://nodejs.org/ko/download/>)
  - tip: Mac을 쓴다면, nvm을 이용해서 설치

```
sudo curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.33.1/install.sh | bash
```

- React 프로젝트 생성하기

```
npx create-react-app my-app  
cd my-app  
npm start
```

- 프로젝트 개발 환경 (참고용 - 내 워크플로우)
  - Visual Studio Code
    - Prettier - Code formatter
    - Bracket Pair Colorizer 2
    - ES7 React/Redux/GraphQL/React-Native snippets
    - Material Icon Theme
  - Chrome


























# 리액트를 하려면 뭘 해야(알아야) 하지?

- 필수로 알아야 하는 거
  - NPM (or YARN)
  - ES6 (Javascript)
  - HTML
  - CSS
- Functional Programming
  - Immutable
- 알면 좋은 거
  - Node
  - ESLint
  - Prettier
  - Visual Code Editor (이건 내 취향)
  - 다양한 컴포넌트, 라이브러리, 프레임워크 ....

# 자 그럼 이제 뭘 할까?

- 기본 개념?
- 실습?
  
- 기본적인 개념이 없으면 어려워!!!!
- 좀 지루하겠지만 기본 개념들 중심으로 설명을 할게요.
  - (두서없이.... 생각나는데로...)

# Single Page App (CSR, SSR,...)

	Server				Browser
					
	Server Rendering	"Static SSR"	SSR with (Re)hydration	CSR with Prerendering	Full CSR
Overview:	An application where input is navigation requests and the output is HTML in response to them.	Built as a Single Page App, but all pages prerendered to static HTML as a build step, and the JS is <b>removed</b> .	Built as a Single Page App. The server prerenders pages, but the full app is also booted on the client.	A Single Page App, where the initial shell/skeleton is prerendered to static HTML at build time.	A Single Page App. All logic, rendering and booting is done on the client. HTML is essentially just script & style tags.
Authoring:	Entirely server-side <small>(request-response, HTML)</small>	Built as if client-side <small>(components, DOM*, fetch)</small>	Built as client-side	Client-side	Client-side
Rendering:	Dynamic HTML	Static HTML	Dynamic HTML <b>and</b> JS/DOM	Partial static HTML, then JS/DOM	Entirely JS/DOM
Server role:	Controls all aspects. <small>(thin client)</small>	Delivers static HTML	Renders pages <small>(navigation requests)</small>	Delivers static HTML	Delivers static HTML
Pros:	 TTI = FCP  Fully streaming	 Fast TTFB  TTI = FCP  Fully streaming	 Flexible	 Flexible  Fast TTFB	 Flexible  Fast TTFB
Cons:	 Slow TTFB  Inflexible	 Inflexible  Leads to hydration	 Slow TTFB  TTI >>> FCP  Usually buffered	 TTI > FCP  Limited streaming	 TTI >>> FCP  No streaming
Scales via:	Infra size / cost	build/deploy size	Infra size & JS size	JS size	JS size
Examples:	Gmail HTML, Hacker News	Docusaurus, Netflix*	<a href="#">Next.js</a> , <a href="#">Razzle</a> , etc	Gatsby, Vuepress, etc	Most apps

**CSR : Client Side Rendering**  
**SSR : Server Side Rendering**

# JSX

```
const element = (  
  <div>  
    <h1 className='header'>Hello, world!</h1>  
    <p className='subject'>What is React?</p>  
    <ul className='colorList' color='blue'>  
      <li>Orange</li>  
      <li>Melon</li>  
      <li>Banana</li>  
    </ul>  
    <HXButton onClick={handleClick}>Go to Helixtech Home</HXButton>  
  </div>  
const container = document.getElementById('root');  
ReactDOM.render(element, container);
```

\* 엘리먼트(element)  
React 앱의 가장 작은 단위  
참고. DOM 엘리먼트

- React를 이용해 View를 기술하는데 사용하는 XML 포맷
- Babel이 JS코드로 변환
- <http://bitly.kr/bClzv1>

# 컴포넌트

- 컴포넌트를 통해 UI를 재사용 가능한 개별적인 여러 조각으로 나누어 개발
- 리엑트는 컴포넌트를 개발하여 이를 조합하여 사용하는 것
  - 함수 컴포넌트 — 16.8 이후 함수 컴포넌트 추천

```
function Welcome(props) {  
  return <h1>Hello, {props.name}</h1>;  
}
```

- 클래스 컴포넌트

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

- html 컴포넌트는 소문자(h1, div,...)로, 리엑트 컴포넌트는 대문자(App..., User..., Layout...)로

# 컴포넌트 조합 (Composition)

```
function Comment(props) {  
  return (  
    <div className="Comment">  
      <UserInfo user={props.author} />  
      <div className="Comment-text">  
        {props.text}  
      </div>  
      <div className="Comment-date">  
        {formatDate(props.date)}  
      </div>  
    </div>  
  );  
}
```

```
function UserInfo(props) {  
  return (  
    <div className="UserInfo">  
      <Avatar user={props.user} />  
      <div className="UserInfo-name">  
        {props.user.name}  
      </div>  
    </div>  
  );  
}
```

```
function Avatar(props) {  
  return (  
    <img className="Avatar"  
      src={props.user.avatarUrl}  
      alt={props.user.name}  
    />  
  );  
}
```

# props and state

- props
  - 컴포넌트에 인자로 넘어가는 프로퍼티들(Properties)
- state
  - 컴포넌트 내부에서 관리하는 상태
- props나 state가 변경되면 컴포넌트가 다시 렌더링 됩니다.



# State 관리

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

```
  componentDidMount() {  
    this.timerID = setInterval(  
      () => this.tick(),  
      1000  
    );  
  }
```

```
  componentWillUnmount() {  
    clearInterval(this.timerID);  
  }
```

```
  tick() {  
    this.setState({  
      date: new Date()  
    });  
  }
```

```
  render() {  
    return (  
      <div>  
        <h1>Hello, world!</h1>  
        <h2>It is {this.state.date.toLocaleTimeString()}.</h2>  
      </div>  
    );  
  }
```

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

# Immutable 불변 객체

- props 는 내부에서 변경 불가
- state는 직접 변경하면 안 됨. 새로운 객체를 만들어서 저장
- Immutable.js, immer.js 등 불변 객체를 지원하는 라이브러리들이 있음
- \* 함수형 프로그래밍
  - 순수 함수
  - 사이드 이펙트 없어야 함
  - 함수를 변수처럼 다룰 수 있다. (First Object)
  - 함수를 파라미터, 리턴값으로 사용 가능.(High Order Function : 고차 함수)

# 렌더링 (Reconciliation), Virtual DOM

- 리액트 렌더링은 Virtual DOM을 생성
- Virtual DOM을 만들고 이를 이전의 Virtual DOM과 비교하여 변경된 것만 실제 DOM에 반영

# 이벤트 처리하기

```
class Toggle extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = {isToggleOn: true};
```

```
// 콜백에서 `this`가 작동하려면 아래와 같이 바인딩 해주어야 합니다.  
this.handleClick = this.handleClick.bind(this);
```

```
}
```

```
handleClick() {  
  this.setState(state => ({  
    isToggleOn: !state.isToggleOn  
  }));  
}
```

```
render() {  
  return (  
    <button onClick={this.handleClick}>  
      {this.state.isToggleOn ? 'ON' : 'OFF'}  
    </button>  
  );  
}
```

# this 바인딩

- JS는 기본적으로 함수 스코프가 기본
- Object Oriented 언어가 아닌 그럴 듯하게 흉내만 냄
- JS에서 Object를 쓰기가 너무 어려움. —> 함수형 프로그래밍 쪽으로 이동하는 이유

```
<button onClick={(e) => this.deleteRow(id, e)}>Delete Row</button>  
<button onClick={this.deleteRow.bind(this, id)}>Delete Row</button>
```

# ES6 - 1/1

- const, let
  - var: 기존 자바스크립트 변수 선언, 글로벌로만 사용, 함수 스코프
  - const: 상수 선언, 블록 스코프
  - let: 변수 선언, 블록 스코프
- Destructuring
  - object나 array의 일부를 쉽게 가져오는 방법
  - {name, age} = user // user 객체에서 name과 age 만 빼온다.
  - [first, second] = fruits // fruits 배열 중에 0,1번 것만 가져온다.
- Spread Operator
  - 기존 object나 object의 내용(properties)를 펼쳐서 가져오는 것
  - ...state, ...arr
  - const copyOfArray = [...arr] // 배열 복사
  - const concatArray = [1, 2, 3, ...arr] // 기존 배열에 1,2,3 추가

# ES6 - 2/2

- (Fat) Arrow functions
  - 화살표 함수
  - this가 정적으로 바인딩 됨
  - 기본적으로 이름이 없는 함수
  - `let myfunc = (a, b) => a + b` // 함수 본문이 한 줄이고 바로 리턴하는 경우 `{}` 생략 가능
- Promise, async, await
  - then 체이닝 가능
  - callback이 층층이 쌓이는 문제 해결. 직관적
  - 예외처리는 catch에서 수행

`axios.get(url)` // axios에서 get request를 하는 경우

`.then((data) => {...})`

`.then((data) => {...})`

`.catch((err) => {...})`



# 조건부 렌더링 if else

```
render() {  
  const isLoggedIn = this.state.isLoggedIn;  
  let button;  
  
  if (isLoggedIn) {  
    button = <LogoutButton onClick={this.handleLogoutClick} />;  
  } else {  
    button = <LoginButton onClick={this.handleLoginClick} />;  
  }  
  
  return (  
    <div>  
      <Greeting isLoggedIn={isLoggedIn} />  
      {button}  
    </div>  
  );  
}
```

# 조건부 렌더링 &&

```
function Mailbox(props) {  
  const unreadMessages = props.unreadMessages;  
  return (  
    <div>  
      <h1>Hello!</h1>  
      {unreadMessages.length > 0 &&  
        <h2>  
          You have {unreadMessages.length} unread messages.  
        </h2>  
      }  
    </div>  
  );  
}
```

```
const messages = ['React', 'Re: React', 'Re:Re: React'];  
ReactDOM.render(  
  <Mailbox unreadMessages={messages} />,  
  document.getElementById('root')  
);
```

# 조건부 렌더링 ? :

```
render() {  
  const isLoggedIn = this.state.isLoggedIn;  
  return (  
    <div>  
      The user is <b>{isLoggedIn ? 'currently' : 'not'}</b> logged in.  
    </div>  
  );  
}
```

# 리스트와 키

```
function NumberList(props) {  
  const numbers = props.numbers;  
  const listItems = numbers.map((number) =>  
    <li key={number.toString()}>  
      {number}  
    </li>  
  );  
  return (  
    <ul>{listItems}</ul>  
  );  
}
```

```
const numbers = [1, 2, 3, 4, 5];  
ReactDOM.render(  
  <NumberList numbers={numbers} />,  
  document.getElementById('root')  
);
```

# 리스트 좀 더 간결하게

```
function NumberList(props) {  
  const numbers = props.numbers;  
  return (  
    <ul>  
      {numbers.map((number) =>  
        <ListItem key={number.toString()}  
          value={number} />  
      )}  
    </ul>  
  );  
}
```

# 품

```
class NameForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = {value: ''};

    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

  handleChange(event) {
    this.setState({value: event.target.value});
  }

  handleSubmit(event) {
    alert('A name was submitted: ' + this.state.value);
    event.preventDefault();
  }

  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Name:
          <input type="text" value={this.state.value} onChange={this.handleChange} />
        </label>
        <input type="submit" value="Submit" />
      </form>
    );
  }
}
```

**Controlled Component (제어 컴포넌트)**  
**Uncontrolled Component (비제어 컴포넌트)**



# 휴~~

- 일단 주욱 훑어본듯
  - 근데 이거 다 react 홈페이지에 나오는 거 (<https://ko.reactjs.org/docs/hello-world.html> )
- 이제 뭐해야 하나요?
  - Router : react-router-dom
  - Redux : global state
  - React Hooks : Functional Component
- 차주부터 간단한 프로젝트 진행할까? 웹 하나 만들어볼까요?
- 혼자 해 보고 싶으면 -> 자습서: React 시작하기