

2강. 컴포넌트 스타일링, 프로젝트 만들기.

컴포넌트 스타일링

일반 CSS

```
.App {
  text-align: center;
}

.App-logo {
  animation: App-logo-spin infinite 20s linear;
  height: 80px;
}

.App-header {
  background-color: #222;
  height: 150px;
  padding: 20px;
  color: white;
}

.App-intro {
  font-size: large;
}

@keyframes App-logo-spin {
  from { transform: rotate(0deg); }
  to { transform: rotate(360deg); }
}
```

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';

class App extends Component {
  render() {
    return (
      <div className="App">
        <div className="App-header">
          <img src={logo} className="App-logo" alt="logo" />
          <h2>Welcome to React</h2>
        </div>
        <p className="App-intro">
          To get started, edit <code>src/App.js</code> and save to reload.
        </p>
      </div>
    );
  }
}

export default App;
```

단점: 클래스명이 중첩될 가능성이 있습니다

.App-header

.App-intro

.app .header

.app .intro .contents .title

다른 솔루션들..

CSS Module

SASS, LESS, Stylus ...

CSS in JS

CSS Module

CSS 파일을 불러온 컴포넌트 내부에서만 작동합니다


```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';

class App extends Component {
  render() {
    return (
      <div className="App">
        <div className="App-header">
          <img src={logo} className="App-logo" alt="logo" />
          <h2>Welcome to React</h2>
        </div>
        <p className="App-intro">
          To get started, edit <code>src/App.js</code> and save to reload.
        </p>
      </div>
    );
  }
}

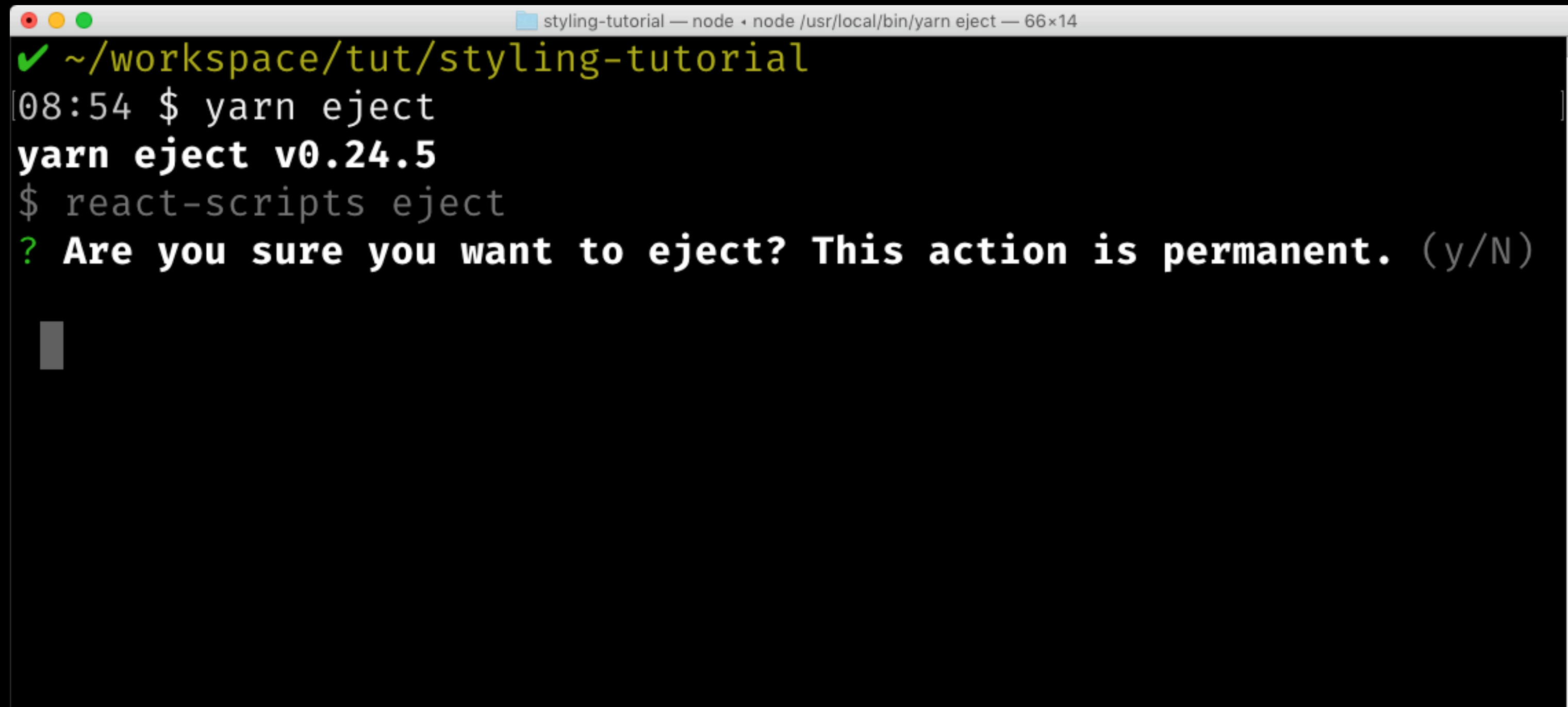
export default App;
```

```
import React, { Component } from 'react';
import logo from './logo.svg';
import styles from './App.css';

class App extends Component {
  render() {
    return (
      <div className={styles.App}>
        <div className={styles.header}>
          <img src={logo} className={styles.logo} alt="logo" />
          <h2>Welcome to React</h2>
        </div>
        <p className={styles.intro}>
          To get started, edit <code>src/App.js</code> and save to reload.
        </p>
      </div>
    );
  }
}
export default App;
```

프로젝트 생성

```
$ create-react-app css-module-tutorial
```

A terminal window with a title bar that reads "styling-tutorial — node • node /usr/local/bin/yarn eject — 66x14". The terminal content shows a successful checkmark and the path "~/workspace/tut/styling-tutorial". It then shows the command "yarn eject" being executed, followed by "yarn eject v0.24.5" and "\$ react-scripts eject". A confirmation prompt appears: "? Are you sure you want to eject? This action is permanent. (y/N)". A cursor is visible on the line below the prompt.

```
✓ ~/workspace/tut/styling-tutorial
[08:54 $ yarn eject
yarn eject v0.24.5
$ react-scripts eject
? Are you sure you want to eject? This action is permanent. (y/N)
█
```

프로젝트 기본설정을 할 땐 **eject**:
babel, webpack 설정파일들을 프로젝트 루트 디렉토리로 꺼내줍니다

Webpack 설정 변경

config/webpack.config.dev.js

before

```
{
  loader: require.resolve('css-loader'),
  options: {
    importLoaders: 1,
  },
},
```

after

```
{
  loader: require.resolve('css-loader'),
  options: {
    importLoaders: 1,
    modules: true,
    localIdentName: '[name]__[local]__[hash:base64:5]'
  },
}
```

Webpack 설정 변경

config/webpack.config.prod.js

before

```
{
  loader: require.resolve('css-loader'),
  options: {
    importLoaders: 1,
    minimize: true,
    sourceMap: true,
  },
}
```

after

```
{
  loader: require.resolve('css-loader'),
  options: {
    importLoaders: 1,
    modules: true,
    localIndentName: '[name]__[local]__[hash:base64:5]',
    minimize: true,
    sourceMap: true,
  },
},
```

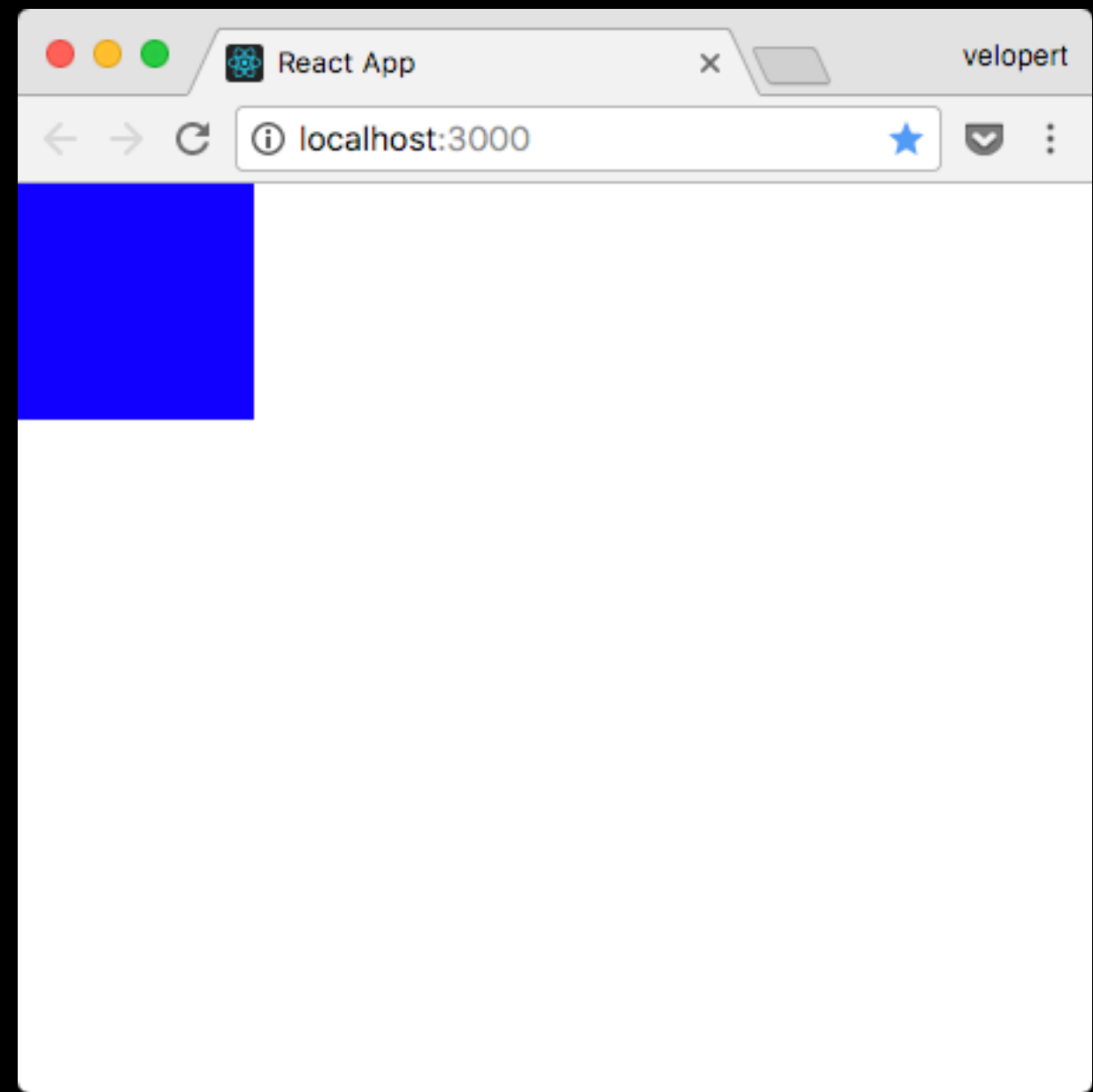
`$ yarn start`

src/App.css

```
.blueBox {  
  width: 100px;  
  height: 100px;  
  background: blue;  
}
```

src/App.js

```
import React, { Component } from 'react';  
import style from './App.css';  
  
class App extends Component {  
  render() {  
    return (  
      <div className={style.blueBox}>  
        </div>  
    );  
  }  
}  
  
export default App;
```

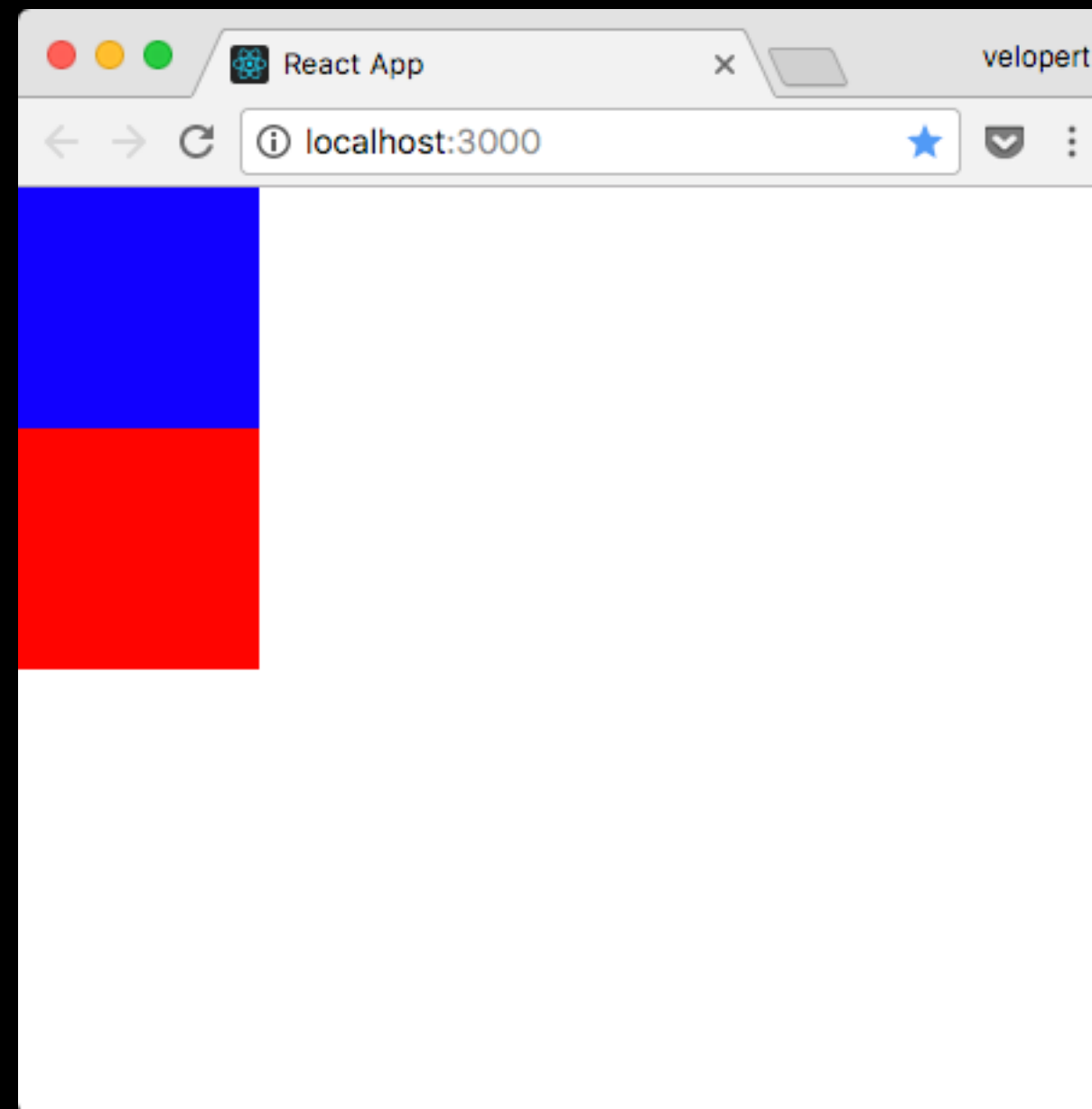
글로벌 스타일?

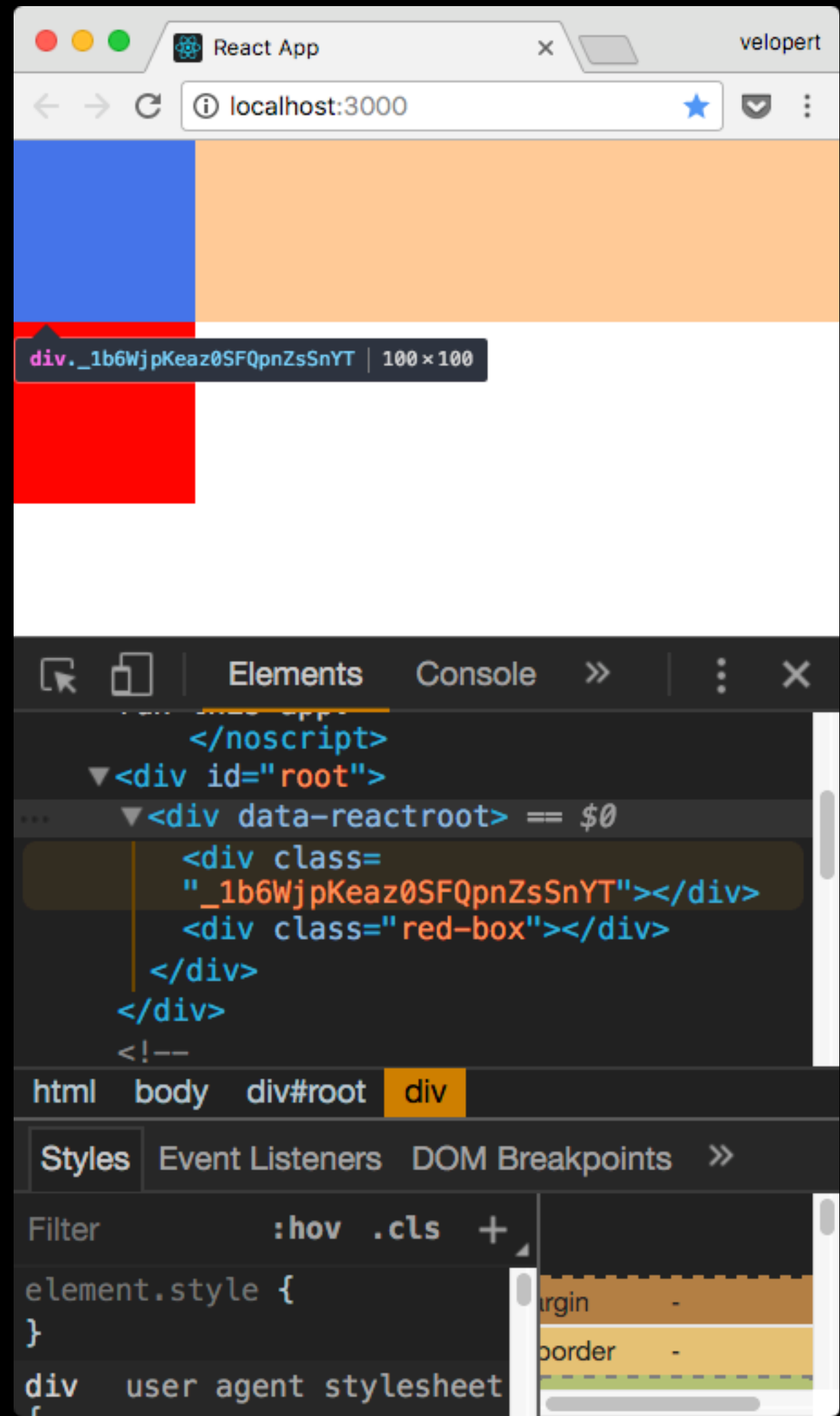
src/App.css

```
.blueBox {  
  width: 100px;  
  height: 100px;  
  background: blue;  
}  
  
:global .red-box {  
  width: 100px;  
  height: 100px;  
  background: red;  
}
```

src/App.js

```
import React, { Component } from 'react';  
import style from './App.css';  
  
class App extends Component {  
  render() {  
    return (  
      <div>  
        <div className={style.blueBox}>  
        </div>  
        <div className="red-box"></div>  
      </div>  
    );  
  }  
}  
  
export default App;
```





여러개의 클래스를 적용해야할때

src/App.css

```
.blueBox {  
  width: 100px;  
  height: 100px;  
  background: blue;  
}  
  
.whiteText {  
  color: white;  
}  
  
:global .red-box {  
  width: 100px;  
  height: 100px;  
  background: red;  
}
```

src/App.js

```
import React, { Component } from 'react';  
import style from './App.css';  
  
class App extends Component {  
  render() {  
    return (  
      <div>  
        <div className={` ${style.blueBox} ${style.whiteText}`}>  
          Hi  
        </div>  
        <div className="red-box"></div>  
      </div>  
    );  
  }  
}  
  
export default App;
```

```
$ yarn add classnames
```



```
classNames( 'foo', 'bar' ); // ⇒ 'foo bar'
```

```
classNames(style.blueBox, style.whiteText);
```

```
import React, { Component } from 'react';
import style from './App.css';
import classNames from 'classnames';

class App extends Component {
  render() {
    return (
      <div>
        <div className={classNames(style.blueBox, style.whiteText)}>
          Hi
        </div>
        <div className="red-box"></div>
      </div>
    );
  }
}

export default App;
```

styles. 생략하기

```
import React, { Component } from 'react';
import style from './App.css';
import classNames from 'classnames/bind';

const cx = classNames.bind(style);

class App extends Component {
  render() {
    return (
      <div>
        <div className={cx('blueBox', 'whiteText')}>
          Hi
        </div>
        <div className="red-box"></div>
      </div>
    );
  }
}

export default App;
```

조건부 스타일을 준다면..

```
<div className={cx({  
  blueBox: true,  
  whiteText: true  
})}>
```

```
<div className={cx('blueBox', true && 'whiteText')}>
```



```
<div className={cx(['blueBox', true && 'whiteText'])}>
```

특정 CSS 상속하기

```
.blueBoxChild {  
  composes: blueBox;  
  border: 3px solid black;  
}
```

```
.otherClassName {  
  composes: className from "./style.css";  
}
```

아쉬운 점:

SASS 나 LESS 처럼, function, mixin, variable 등의 기능이 없다.

SASS 를 사용해보자!

SASS 한눈에 보기 - <https://velopert.com/1712>

```
$ yarn add sass-loader node-sass
```

webpack.config.dev.js

```
{
  test: /\.css$/,
  use: [
    (...)
  ],
},
{
  test: /\.scss$/,
  use: [
    (...), /* CSS 로더 부분과 동일하게 복붙 */
    {
      loader: require.resolve('sass-loader'),
    },
  ],
},
},
```


webpack.config.prod.js

동일하게 작업해주세요.

src/styles/lib/_mixins.scss

```
@mixin box_shadow ($level) {  
  
  @if $level = 1 {  
    box-shadow: 0 1px 3px rgba(0,0,0,0.12), 0 1px 2px rgba(0,0,0,0.24);  
  } @else if $level = 2 {  
    box-shadow: 0 3px 6px rgba(0,0,0,0.16), 0 3px 6px rgba(0,0,0,0.23);  
  } @else if $level = 3 {  
    box-shadow: 0 10px 20px rgba(0,0,0,0.19), 0 6px 6px rgba(0,0,0,0.23);  
  } @else if $level = 4 {  
    box-shadow: 0 14px 28px rgba(0,0,0,0.25), 0 10px 10px rgba(0,0,0,0.22);  
  } @else if $level = 5 {  
    box-shadow: 0 19px 38px rgba(0,0,0,0.30), 0 15px 12px rgba(0,0,0,0.22);  
  }  
  
}
```

<https://gist.github.com/vlpt-playground/c4d3f4558ba21d6949905a035502dde7>

src/styles/lib/_variables.scss

```
$multiplier: 10px;
```

src/styles/lib/_all.scss

```
@import 'mixins';  
@import 'variables';
```

src/App.scss

```
@import './styles/lib/all';

.card {
  display: inline-block;
  width: $multiplier * 10;
  height: $multiplier * 10;
  margin: $multiplier;
}

.one {
  @include box_shadow(1);
}

.two {
  @include box_shadow(2);
}

.three {
  @include box_shadow(3);
}

.four {
  @include box_shadow(4);
}
```

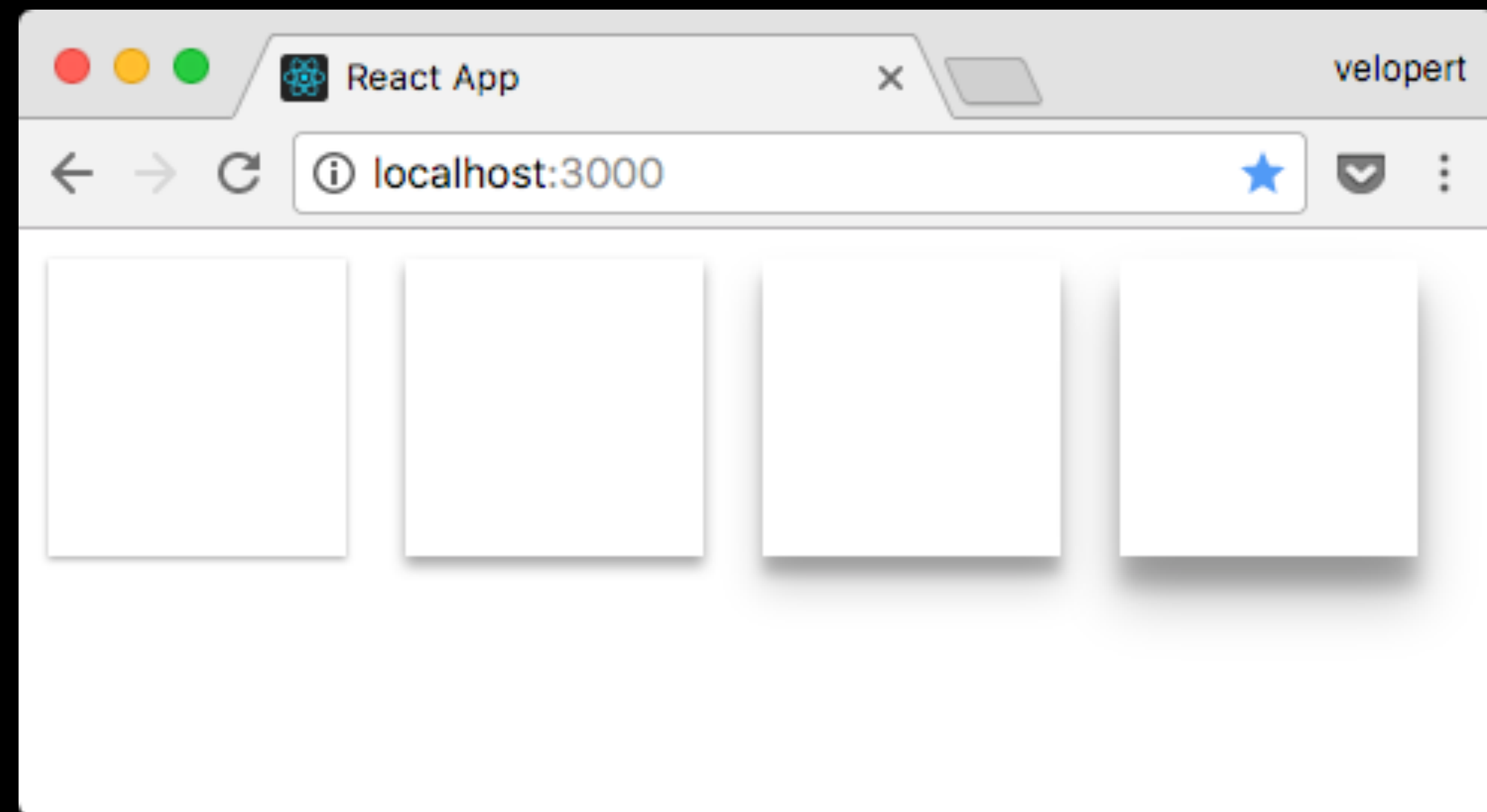
src/App.js

```
import React, { Component } from 'react';
import style from './App.scss';
import classNames from 'classnames/bind';

const cx = classNames.bind(style);

class App extends Component {
  render() {
    return (
      <div>
        <div className={cx('card', 'one')} />
        <div className={cx('card', 'two')} />
        <div className={cx('card', 'three')} />
        <div className={cx('card', 'four')} />
      </div>
    );
  }
}

export default App;
```



```
$ yarn add open-color
```

<https://yeun.github.io/open-color/>

src/styles/lib/_all.scss

```
@import '~open-color/open-color';  
@import 'mixins';  
@import 'variables';
```

~/ 는 node_modules/ 내부 디렉토리

src/App.scss

```
@import './styles/lib/all';

.card {
  display: inline-block;
  width: $multiplier * 10;
  height: $multiplier * 10;
  margin: $multiplier;
}

.one {
  @include box_shadow(1);
  background: $oc-red-6;
}

.two {
  @include box_shadow(2);
  background: $oc-orange-6;
}

.three {
  @include box_shadow(3);
  background: $oc-yellow-6;
}

.four {
  @include box_shadow(4);
  background: $oc-green-6;
}
```

태그에 해당되는 스타일은 글로벌로 설정 가능.

src/styles/base/_reset.scss

```
html, body {  
  background: $oc-gray-4;  
  font-family: 'Noto Sans KR', 'NanumGothic', sans-serif;  
  margin: 0;  
}
```

src/styles/base/_typography.scss

```
@import url(https://fonts.googleapis.com/earlyaccess/notosanskr.css);  
  
h1 {  
  font-size: 4rem;  
}  
  
h2 {  
  font-size: 3rem;  
}
```

src/styles/lib/_all.scss

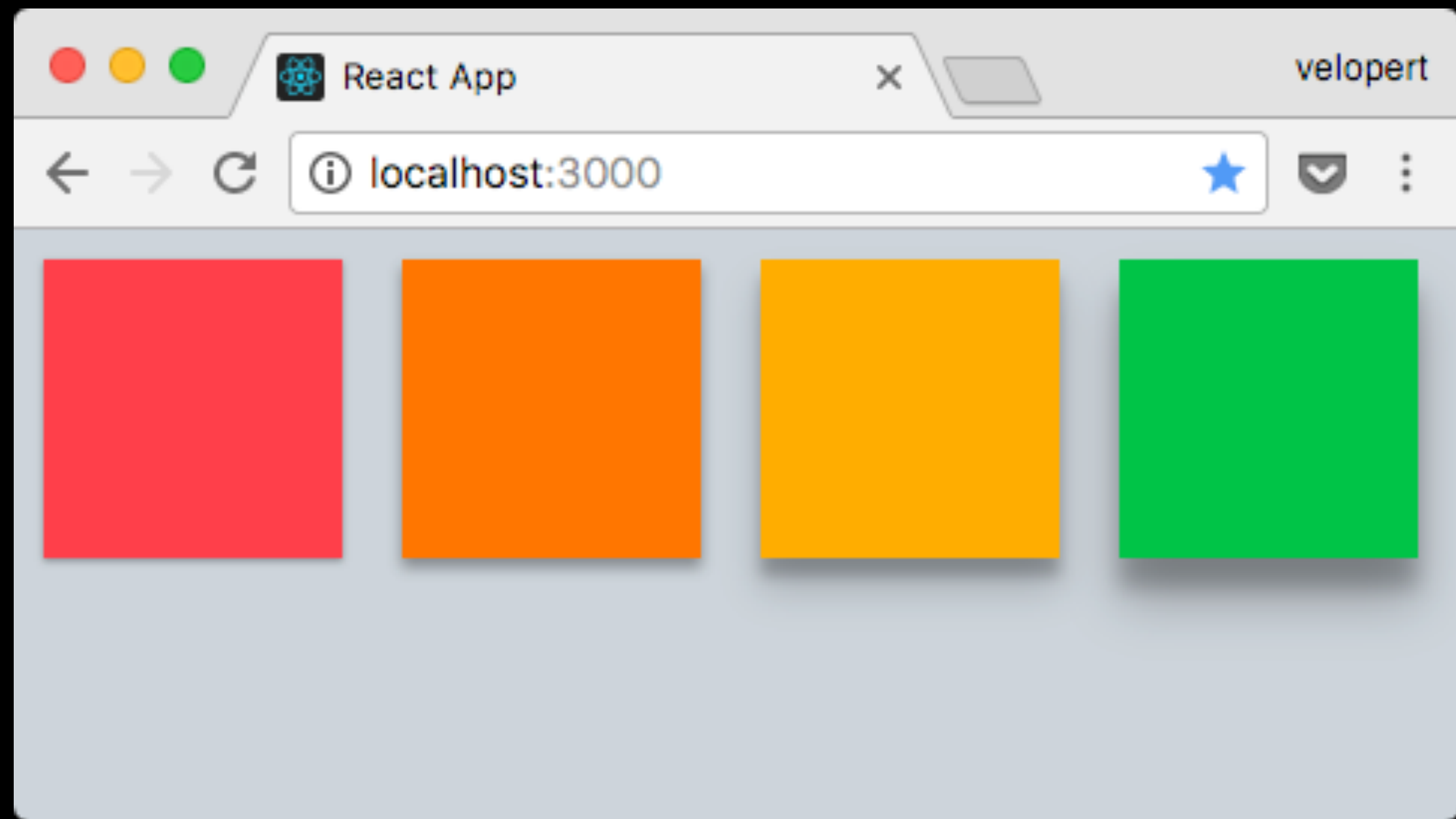
```
@import 'typography';  
@import 'reset';
```

src/styles/main.scss

```
@import '~open-color/open-color';  
@import './base/all';
```

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import registerServiceWorker from './registerServiceWorker';
import './styles/main.scss';

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



global 클래스

```
:global {  
    .something {  
        background: black;  
    }  
    .global {  
        background: blue;  
    }  
}
```


스타일 디렉토리 구조를 어떻게해야할까?

딱히 정해진 방법은 없습니다

1) CSS Module 을 사용하지 않고, 모든걸 일반 Sass 로만 관리:

<https://github.com/velopert/veloxy/tree/master/veloxy-frontend/src/styles>

2) CSS Module + Sass

<https://github.com/velopert/bitimulate/tree/master/bitimulate-frontend/src/components/atoms>

3) 자신이 편하다고 생각하는 구조..

읽어보면 좋은 글:

<http://www.webactually.co.kr/archives/13106>

SCSS 사용 할 때 강추 라이브러리

<http://include-media.com/>

반응형 디자인을 굉장히 간편하게 해준다.

CSS in JS

<https://github.com/MicheleBertoli/css-in-js>

styled-components

```
import React from 'react';

import styled from 'styled-components';

const Title = styled.h1`
  font-size: 1.5em;
  text-align: center;
  color: palevioletred;
`;

const Wrapper = styled.section`
  padding: 4em;
  background: papayawhip;
`;
```



```
<Wrapper>  
  <Title>안녕하세요</Title>  
</Wrapper>
```

foo`...`

https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Template_literals

```
function foo( ... args ) {  
    console.log(args);  
}
```

```
> foo`1+1=${1+1} and 2+2=${2+2}!`
```

```
▼ (3) [Array(3), 2, 4] ⓘ
```

```
▼ 0: Array(3)
```

```
0: "1+1="
```

```
1: " and 2+2="
```

```
2: "!"
```

```
length: 3
```

```
► raw: Array(3)
```

```
► __proto__: Array(0)
```

```
1: 2
```

```
2: 4
```

```
length: 3
```

```
► __proto__: Array(0)
```

왜?

```
styled.div`  
  color: tomato;  
  ${ props => props.background}  
`
```

```
> `1+1=${1+1} and 2+2=${(a,b) => a+b}!`  
< "1+1=2 and 2+2=(a,b) => a+b!"  
> foo`1+1=${1+1} and 2+2=${function(){} }!`  
▼ (3) [Array(3), 2, function] ⓘ  
  ► 0: Array(3)  
    1: 2  
  ► 2: function ()  
      length: 3  
  ► __proto__: Array(0)
```

CSS 가 JS 안에!

- 더욱 깔끔해지는 프로젝트 디렉토리 구조
파일을 왔다 갔다 할 필요가 없다. 👍
- CSS 에서 props 에 접근 할 수 있다!
- 자바스크립트와 CSS 의 경계가 허물어짐

스타일시트가 브라우저에서 만들어지기 때문에 **미세한 성능 저하**

디자이너가 HTML/CSS 작업을 따로 해줄때는,
프로세스가 조금 달라지기 때문에 복잡해질지도

자동완성이 안됨