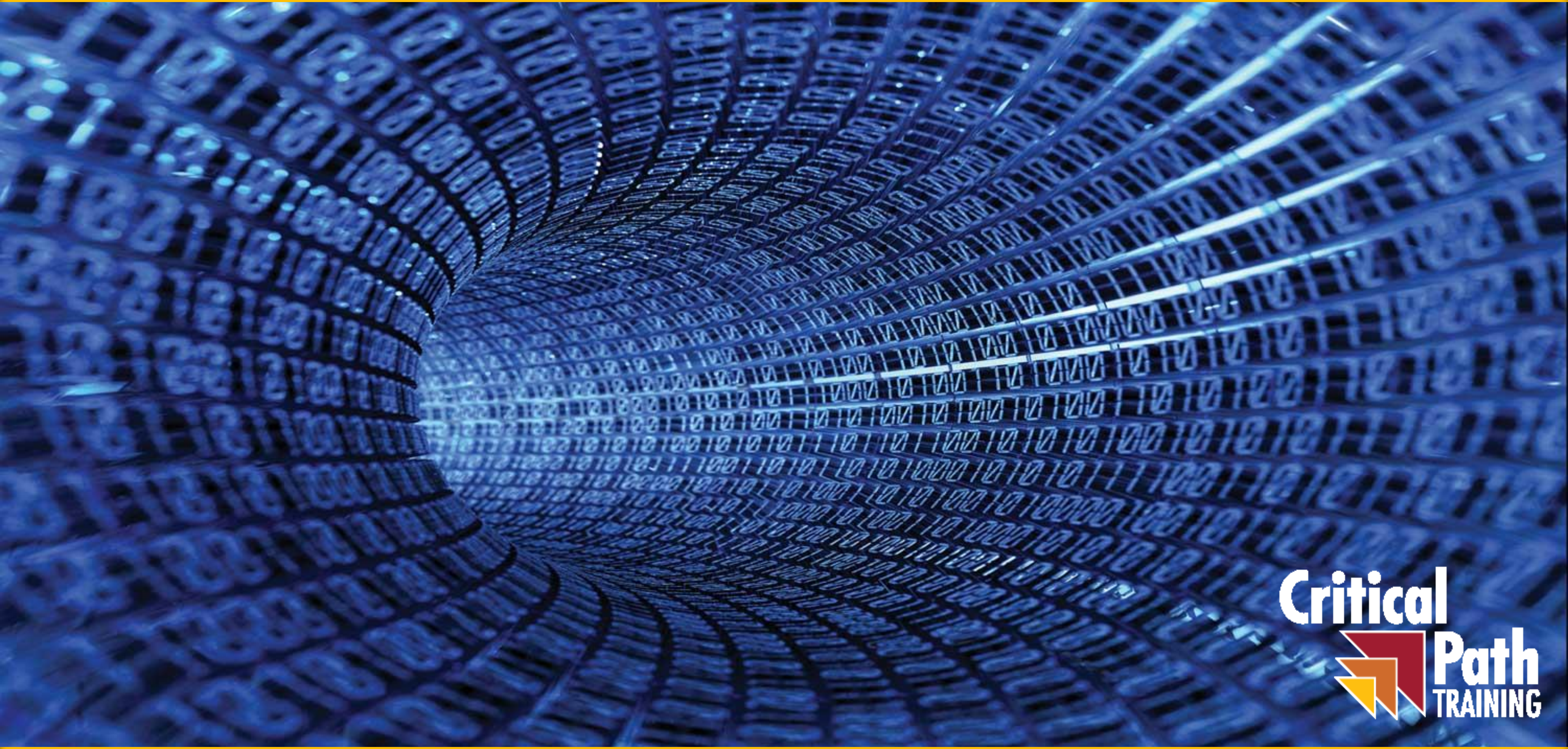
Developing SPFx Web Parts using React.js



Critical Path TRAINING

Agenda

- Getting Started with React.js
- Working with JSX and TSX files
- Understanding Component Properties vs. State
- Developing SPFx Web Parts using React.js
- Passing Web Part Properties to a Component

Introducing React

- React is a framework for building user interfaces
  - The framework *reacts* to state changes in the UI
  - Emphasizes component-based development
  - Lighter than other frameworks
  - Ideal for building web parts

React Fundamentals

- Obtain the framework withfrom a CDN or npm
  - npm install react --save
  - npm install react-dom --save
- React is the main entry point to APIs
- ReactDOM used to render elements
- React.DOM wraps standard HTML elements

Hello World, the React Edition

- Hello, World

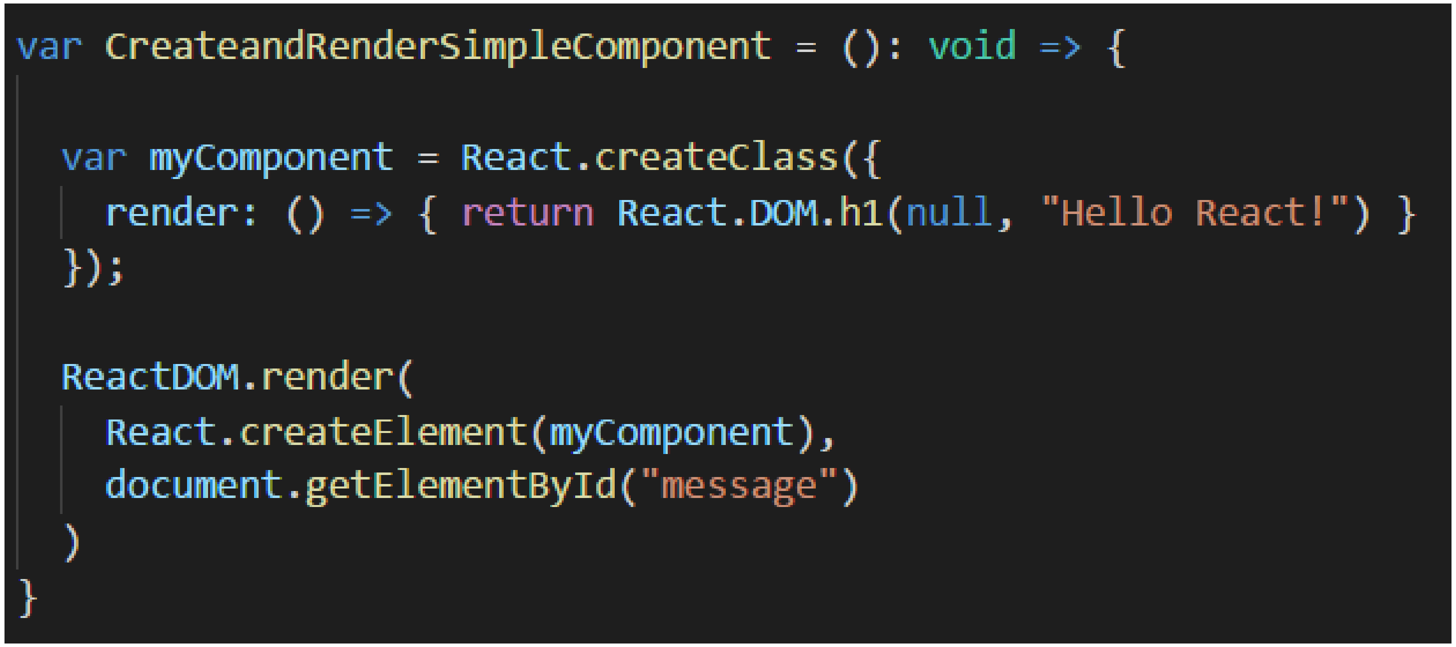
```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>React JavaScript Basics</title>
</head>
<body>

    <div id="app"></div>

    <!-- React Libraries -->
    <script src="https://cdnjs.cloudflare.com/ajax/libs/react/15.5.4/react.min.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/react/15.5.4/react-dom.min.js"></script>

    <script>
        ReactDOM.render(React.DOM.h1(null, "Hello, React!"), document.getElementById("app"));
    </script>

</body>
</html>
```

React Components

- A custom class extending `React.Component`
  - render method returns a React component
  - Immutable props for component configuration
  - Changeable state used to render component

ES5 Component

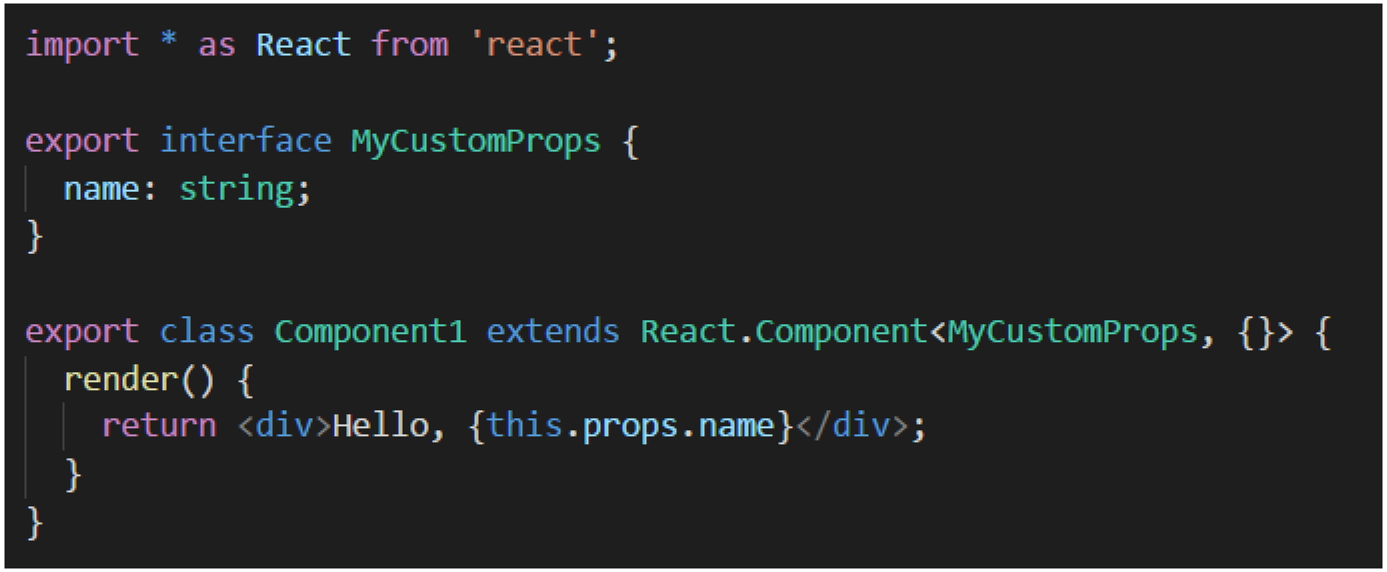
- Created using `React.createClass`



```
var CreateandRenderSimpleComponent = (): void => {

  var myComponent = React.createClass({
    render: () => { return React.DOM.h1(null, "Hello React!") }
  });

  ReactDOM.render(
    React.createElement(myComponent),
    document.getElementById("message")
  )
}
```

Component in TypeScript or ES6

- Component derives from `React.Component`
  - Base class can be parameterized with interfaces
  - First interface defines component properties
  - Second interface defines component state



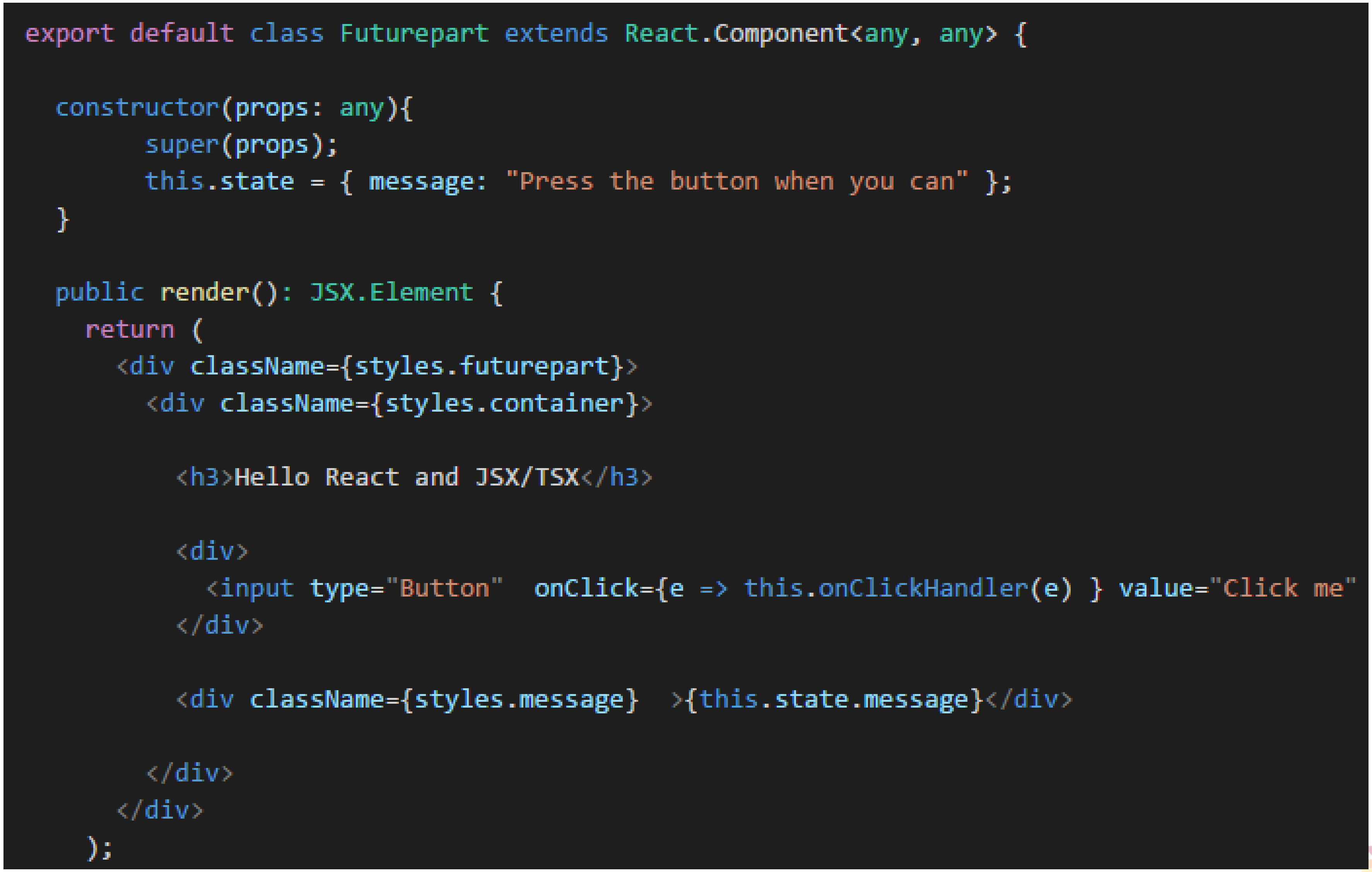
```
import * as React from 'react';

export interface MyCustomProps {
  name: string;
}

export class Component1 extends React.Component<MyCustomProps, {}> {
  render() {
    return <div>Hello, {this.props.name}</div>;
  }
}
```

Utilizing JSX (and TSX)

- JSX is a preprocessor step
  - It allows for XML syntax in to JavaScript code
  - It's optional, but very useful for organizing components
  - It requires a transpiler like TypeScript or Babel
- The following are equivalent:

```
ReactDOM.render(
        React.createElement(Component, { message: "My first component" }),
                document.getElementById("app"));


ReactDOM.render(
        <Component message="My first component" />,
                document.getElementById("app"));
```

React and JSX



```
export default class Futurepart extends React.Component<any, any> {

  constructor(props: any){
        super(props);
        this.state = { message: "Press the button when you can" };
  }

  public render(): JSX.Element {
    return (
      <div className={styles.futurepart}>
        <div className={styles.container}>

          <h3>Hello React and JSX/TSX</h3>

          <div>
            <input type="Button"  onClick={e => this.onClickHandler(e) } value="Click me"
          </div>

          <div className={styles.message}  >{this.state.message}</div>

        </div>
      </div>
    );
```

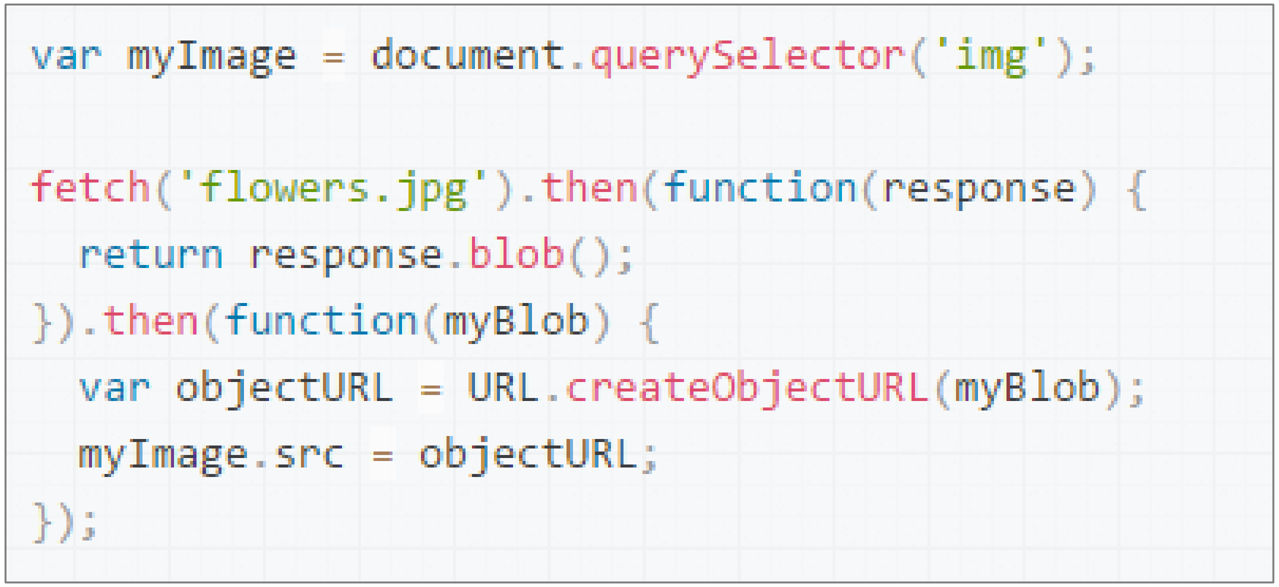
Component Lifecycle

- componentWillUpdate
  - executed before component is rendered
- componentDidUpdate
  - executed after component is rendered
- componentWillMount
  - executed before node is added to the DOM
- componentDidMount
  - executed after node is added to the DOM
- componentWillUnmount
  - executed before node is removed from the DOM
- shouldComponentUpdate(newProps, newState)
  - executed before component is updated

fetch()

Promise-based network requests

- Supported natively by Chrome 49 and above
- Supported by TypeScript for other browsers



```
var myImage = document.querySelector('img');

fetch('flowers.jpg').then(function(response) {
  return response.blob();
}).then(function(myBlob) {
  var objectURL = URL.createObjectURL(myBlob);
  myImage.src = objectURL;
});
```

Asynchronous Calls and State Update

```
public componentDidMount(): void {
          fetch(
              '../../_api/web/currentuser',
              {
                  method: 'GET',
                  credentials: 'same-origin',
                  headers: {
                      'accept': 'application/json'
                  }
              }
          ).then(response => {
              return response.json();
          }).then(json => {
              console.log(json);
              this.setState({ data: json.Title, isValid: true });
          }).catch(e => {
              console.log(e);
          });
  }
```

Critical for SharePoint

Event Handling

```
constructor(props: IMyProps){
        super(props);
        this.state.value = props.value;
        this.changed = this.changed.bind(this);
   }

   public render(): React.ReactElement<any> {
            return (<div className={ this.className }>
                      <input onChange={this.changed}  type="text“
                             value={this.state.value} />
                    </div>);
   }

   public changed(event): void {
     var newValue: string = event.target.value;
     event.stopPropagation();
     event.nativeEvent.stopImmediatePropagation();
   }
```

Be sure to bind ‘this’

Designate handler

Implement handler

Stop bubbling
Stop other handlers

Agenda

- Overview the SharePoint Framework (SPFx)
- Setting up an SPFx Development Environment
- Creating Projects using the SPFx Templates
- Deploying SPFx Projects using an Azure CDN

DEMO

Creating Web Parts with React.js