Building Applications for Office 365 and SharePoint with ReactJS

Scot Hillier

scot@scothillier.net

@ScotHillier

SharePoint *intersection*

Office 365 *intersection*

Agenda

- Setting up the environment
- React framework
  - Fundamentals
  - Lifecycle and async operations
- Developing with SPFX
  - Web parts and extensions
  - Deployment

Setting up the environment

Set up your SharePoint environment

- Cloud or on-premises
  - Office 365 tenant
  - SharePoint 2016, Feature Pack 2
- App catalog
- Developer site collection
- SharePoint workbench
  - Local
  - Tenant
  - On-premises

Set up your development environment

- NodeJS LTS v6.11.4 (https://nodejs.org/en)
  - Node package manager (npm) v3.10.10 automatically installed
  - NodeJS command prompt used to interact with npm
- Visual Studio Code
- Gulp
  - `npm install -g gulp`
- Yeoman
  - `npm install -g yo`
- SharePoint Generator
  - `npm install -g @Microsoft/generator-sharepoint`

React

Introducing React

- React is a framework for building user interfaces
- Emphasizes component-based development
- Lighter than other frameworks
- Ideal for SPFX

React Fundamentals

- Obtain the framework from a CDN or npm
  - https://cdnjs.cloudflare.com/ajax/libs/react/15.5.4/react.min.js
  - https://cdnjs.cloudflare.com/ajax/libs/react/15.5.4/react-dom.min.js
  - npm install react –save
  - npm install react-dom --save
- **`React`** object is the main entry point to APIs
- ReactDOM object is used to render visual elements
- React.DOM object wraps standard HTML elements

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.DOM contains HTML components

ReactDOM allows rendering of 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

ECMAScript component

```
class Component extends React.Component {

    constructor(props) {
        super(props);
        this.state = { text: props.message };
        this.updateTextState = this.updateTextState.bind(this);
    }

    render() { return React.DOM.h1(
                { onClick: this.updateTextState },
                this.state.text); }

    updateTextState(newText) { this.setState({ text: "Thank you!" }); }

}
```

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; }
```

Be sure to bind 'this'

Designate handler

Implement handler

Utilizing JSX

- JSX is a preprocessor step that adds XML syntax to JavaScript
  - It is optional, but very useful for organizing components
  - It requires a transpiler like Traceur, Babel, or TypeScript
  - 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"));
```

Demo

React fundamentals

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

- The Fetch standard defines how to fetch all resources
  - https://fetch.spec.whatwg.org/
- Also defines the fetch() JavaScript API

Fetching

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

Critical for SharePoint

Demo

React lifecycle

SPFX andReact

Project Types

- Web parts
  - Property pane
- Extensions
  - Application customizer
  - Field customizer
  - Command set

Demo

SPFX and React