Web Components with React

12/2016

Agenda

1. Introdução
2. Compound Components
3. Props, PropTypes, States
4. Dúvidas

Objetivo

Este material não tem por objetivo comparar os Frameworks Web existentes comparará-los listando pontos fortes e fracos. Nosso objeto é apresentar a facilidade que o React nos proporciona para componentização inteira da camada web. Alguns pontos precisam ser destacados:

- Cada funcionalidade do projeto será mapeada em um componente web dentro de sua namespace. Exemplo: Prodesp.Compras.Orcamento, Prodesp.Estoque.Fabricante, Prodesp.RBAC.Usuario;
- Passaremos a desenvolver componentes web e compartilhá-los entre projetos independente da linguagem: C# MVC, C# Web Forms, Java, PHP, etc.
- O cliente que consome nosso componente não precisa ter um Client React apenas as libs react necessárias assim como as versões do jQuery, Underscore, Bootstrap, etc.
- Os componentes serão versionados e terão suas APIs compatibilizadas,
  Exemplo: npm install Prodesp.Compras.Orcamento #1.0.3 que por sua vez é compatível com a versão 1.0.8 do serviço Prodesp.Compras.Orcamento.Svc
- Ao invés de um HTML extenso faremos um extenso uso de nossos componentes, Exemplo; Prodesp.Compras.Orcamento.Package.json com dependências para Prodesp.Core.Buttons, Prodesp.Core.Grid, Prodesp.Core.Navbar, Prodesp.Core.Menu

Sopa de letrinhas

Siglas:
Npm, Web Pack, JSX, Babel, Require, Uglyfy, Gulp, GruntJS, Minify, SPA, Routers, Redux, Flux

Links úteis:
https://github.com/gruntjs/grunt-contrib-uglify
https://github.com/gulpjs/gulp
http://webpack.github.io/docs/tutorials/getting-started/
https://github.com/babel/babel
https://facebook.github.io/react/

JSX

Criando o primeiro componente

Primeiro componente

```
<!DOCTYPE html>
<html>
<head>
<script src="https://fb.me/react-0.13.3.js"></script>
<meta charset="utf-8">
<title>Primeiro componente</title>
</head>
<body>
</body>
</html>
```

Sem JSX

```
var App = React.createClass({
  render: function(){
    return(React.createElement("div", null, "Ola Mundo!"));
  }
});

React.render(React.createElement(App), document.body);
```

Com JSX

```
var App = React.createClass({
  render() {
    return(
      <div>Ola mundo!</div>
    );
}});

React.render(<App/>, document.body);
```

Primeiro componente - babel

https://jsbin.com/

Dados Estáticos

```
var json = [{ "Codigo": "1", "Nome": "Item 1" }, { "Codigo": "2", "Nome": "Item 2" }];
var rows = json.map(function(row){
        return (<tr>
                <td>{row.Codigo}</td>
                <td>{row.Nome}</td>
              </tr>);
});
var App = React.createClass({
  render() {
    return(
        <table>
                <thead>
                        <th>Código</th><th>Nome</th>
                </thead>
                {rows}
        </table>
    );}
});
```

Output

Código	Nome
1	Item 1
2	Item 2

Propriedades

```
var json = { código da página anterior }

var App = React.createClass({
  render() {
    return(
        <table>
            <thead>
                <th>Código</th><th>Nome</th>
            </thead>
            {/* This is a comment */}
            {this.props.rows}
        </table>
    );}
});

React.render(<App rows={json}/>, document.body)
```

Output

Código	Nome
1	Item 1
2	Item 2

Compound Comp - TableData

Um componente que usa outros componentes

TableData - parte 1

```
var TableColumn = React.createClass({
  render() {
    return(<th>{this.props.column}</th>);
  }
});

var TableColumns = React.createClass({
  render() {
    var columns = this.props.columns.map(function(col) {
      return(<TableColumn column = {col}/>);
    });
    return (<thead><tr>{columns}</tr><thead>);
}});
```

```
var TaleRow = React.createClass({
  render() {
    return(<tr>
             <td>{this.props.row.codigo}</td>
             <td>{this.props.row.nome}</td>
           </tr>);
  }
});

var TableRows = React.createClass({
  render() {
    var rows = this.props.rows.map(function(row) {
      return(<TaleRow row={row}/>);
    });
  return (<tobdy>{rows}</tbody>);
}
});
```

TableData - parte 2

```
var TableData = React.createClass({
  render() {
    return (<table>
                <TableColumns columns={this.props.columns} />
                <TableRows rows={this.props.rows} />
              </table>);
  }
});
```

Usando o componente pelo ReactDOM

```
var headers = ['Código', 'Nome'];
var data = [{ "codigo": "1", "nome": "Item 1" }, { "codigo": "2", "nome": "Item 2" }];

ReactDOM.render(<TableData columns={headers} rows={data} />, document.body)
```

TableData - JS Expressions

```
var data = [{ "codigo": "1", "nome": "Item 1" }, { "codigo": "2", "nome": "Item 2" }];

ReactDOM.render(<TableData columns={['Código', 'Nome']} rows={data.length > 0 ? data : [ ]} />,
document.body)
```

Ou lendo propriedades para mudar o comportamento

```
var showResult = function() {
        if(this.props.success === true)
                return <SuccessComponent />
        Else
                return <ErrorComponent />
};
```

TableData - namespace

```
var TableData = React.createClass({
        render: function() {
                return(<table>
                                {this.props.children}
                        </table>);
        }});

TableData.Row = { código anterior }
TableData.Rows = { código anterior }
TableData.Column = { código anterior }
TableData.Columns = { código anterior }
```

Usando o componente

```
var headers = ['Código', 'Nome'];
var data = [
        { "codigo": "1", "nome": "Item 1" },
        { "codigo": "2", "nome": "Item 2" }
];

var App = React.createClass({
render: function(){
return(<TableData>
          <Columns columns={headers} />
          <Rows rows={data} />
        </TableData>);
}});
```

TableData - style / class

```
var TableColumn = React.createClass({
  render() {
        var headingStyle = {
              backgroundColor: 'FloralWhite',
              fontSize: '19px' };
        return(<th style={headingStyle}>{this.props.column}</th>);
  }
});

var TableColumn = React.createClass({
  render() {
        var headingStyle = {
              backgroundColor: 'FloralWhite',
              fontSize: '19px'
        };
        return(<th className={headingStyle}>{this.props.column}</th>);
  }
});
```

DataFlow

Props, PropTypes, State

DataFlow - Props

```
var TableData = React.createClass({
        propTypes: {
                columns: React.PropTypes.array,
                rows: React.PropTypes.array,
                title: React.PropTypes.string.isRequired
        },
        render: function(){
                return(<table className = 'table'>
                                <TableColumns columns = {this.props.columns} />
                                <TableRows rows = {this.props.rows} />
                        </table>);
        }
});
```

CONSOLE

Warning! Required prop **title** was not specified in TableData

DataFlow - Custom Props

```
var TableData = React.createClass({
        propTypes: {
                columns: function(props, propName, componentName) {
                        If ((propName === 'columns') && (props.length === 0))
                                return Error('É necessário que alguma coluna seja informada!');
                        }
                },
                rows: React.PropTypes.array,
                title: React.PropTypes.string.isRequired
        },
        render: function(){
                return(<table className = 'table'>
                                <TableColumns columns = {this.props.columns} />
                                <TableRows rows = {this.props.rows} />
                        </table>);
        }
});
```

DataFlow - Default Props

```
var TableData = React.createClass({
        getDefaultProps: function() {
                return {
                        title: “Titulo da Grade”
                };
        },
        propTypes: {
                columns: React.PropTypes.array,
                rows: React.PropTypes.array,
                title: React.PropTypes.string
        },
        render: function(){
                return(<table className = 'table'>
                                <TableColumns columns = {this.props.columns} />
                                <TableRows rows = {this.props.rows} />
                        </table>);
        }
});
```

DataFlow - State x Props

Usamos propriedades para inicializar ou configurar nossos componentes já os States estão associados a valores que podem ser alterados com o passar do tempo. Um exemplo básico seria a configuração de uma página de login.

Props:
WindowSize: Tamanho da janela
IsModal: Se a janela será modal
Border: Configurações da borda

State:
LoginName: o nome do usuário que está se logando
Password: a senha usada para se autenticar
AuthMode: o tipo de autenticação

DataFlow - State

Usamos propriedades para inicializar ou configurar nossos componentes já os States estão associados a valores que podem ser alterados com o passar do tempo. Um exemplo básico seria a configuração de uma página de login.

Props:
WindowSize: Tamanho da janela
IsModal: Se a janela será modal
Border: Configurações da borda

State:
LoginName: o nome do usuário que está se logando
Password: a senha usada para se autenticar
AuthMode: o tipo de autenticação

DataFlow - Contador usando State

```
var App = React.createClass({
  getInitialState: function () {
    return { count: 1 };
  },
  addOne: function(){
    this.setState({
      count: this.state.count + 1
    });
  },
  render() {
    return (
      <div>
        <button onClick={this.addOne}>+</button>Valor: {this.state.count}
      </div>
    );}
});

React.render(<App/>, document.getElementById("container"));
```

DataFlow - Life Cycle

Todos os eventos associados a renderização dos componentes na camada web:

- componentWillMount: antes de ser renderizado (antes de entrar no render)
- componentDidMount: após a renderização do componente (ao sair do render)
- componentWillReceiveProps(object nextProps)
- boolean shouldComponentUpdate(object nextProps, object nextState)
  Ao mudar o state determina se deverá haver renderização

- componentWillUpdate(object nextProps, object nextState)
  Após ter o state alterado é disparado antes de renderizar

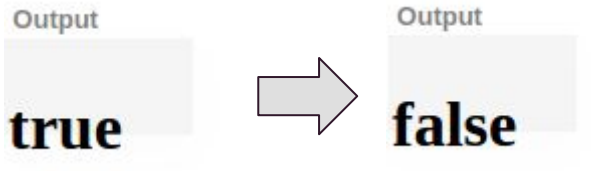
- componentDidUpdate(object prevProps, object prevState)
  Após a renderização quando disparada por um update State

- componentWillUnmount( )
- React.unmountComponentAtNode(document.body)

DataFlow - Life Cycle

```
console.log('Start')
var App = React.createClass({
        componentWillMount: function(){ console.log('componentWillMount'); },
        componentDidMount: function(){ console.log('componentDidMount'); },
        getInitialState: function(){ return { status: true} },
        componentWillReceiveProps: function(nextProps){ console.log('componentWillReceiveProps'); },
        shouldComponentUpdate: function(nextProps, nextState){ console.log('shouldComponentUpdate'); return true; },
        componentWillUpdate: function(){ console.log('componentWillUpdate'); },
        render: function() {
                console.log('render');
                return <h1 onClick={this.toggleState}>
                        {this.state.status.toString()}
                </h1>
        },

        componentWillUnmount: function(){ console.log('componentWillUnmount') },
        toggleState: function() {
                this.setState({status: !this.state.status})
        }
});
```



O que vem por aí

- Forms
- Routes
- Flux
- Redux
- NPM

Dúvidas

Fim