Client-Side Templating with ReactJS (JSX vs NoJSX)

See Scala Play React Seed for an example of how ReactJS could integrate with Clowder.

JSX (JavaScript XML) is a syntax for embedding XML directly into JavaScript code.

This allows for some neat new logical paths, such as returning a block of HTML code to the caller of a JavaScript function.

The Gist of JSX

Assuming you have imported a component called MetadataTable, you could call `React.createElement()` with classic JavaScript:

```
    <script>
        // Without JSX, we can only use plain old JavaScript
        var nojsxContainer = document.querySelector('#react_test_nojsx');
        ReactDOM.render(React.createElement(MetadataTable), nojsxContainer);
    </script>
```

Or you could choose to use JSX to directly embed HTML/XML into your JavaScript:

```
    <script>
        // With JSX, we can use a syntax that feels a bit more like Scala
        // Notice how we have HTML(XML) embedded in our JavaScript
        var jsxContainer = document.querySelector('#react_test_jsx');
        ReactDOM.render(
            <MetadataTable />,
            jsxContainer
        );
    </script>
```

Such a simple example doesn't really illustrate the power of JSX, so let's dive a bit deeper.

How JSX can Simplify Complex Components

Without JSX, the developer ends up calling `React.createElement()` a lot. Like, a LOT a lot.

When things get even the slightest bit complex, the `React.createElement()` syntax (and even the `e()` shorthand) become very tedious to continuously repeat.

This is where JSX can truly shine - rather than a mess of `e('div', {}, e('div', {}, e('div', {}, ... )))`, we can use the cleaner, more familiar syntax of `<div><div><div>...</div></div></div>`

Below, I have provided a few test files. To see these examples in action, simply copy them to your local machine.

You can then view them by navigating a browser to `file:///path/to/example.index.html`

Custom Components Written without JSX

Here is an example of a component written without JSX that can be run by your browser without any compilation necessary - notice that `e()` is a shorthand for `React.createElement()`:

react-nojsx.index.html

```
    <!-- Load Bootstrap CSS from CDN -->
    <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" integrity="
sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u" crossorigin="anonymous">
    <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap-theme.min.css"
integrity="sha384-rHyoN1iRsVXV4nD0JutlnGaslCJuC7uwjduW9SVrLvRYooPp2bWYgmgJQIXwl/Sp" crossorigin="anonymous">
```

```
    <!-- Load React from CDN -->
    <!-- Note: when deploying to production, replace "development.js" with "production.min.js". -->
    <script src="//unpkg.com/react@16/umd/react.development.js" crossorigin></script>
    <script src="//unpkg.com/react-dom@16/umd/react-dom.development.js" crossorigin></script>
    <div id="react_test">Component failed to load</div>

    <script>
        'use strict';

        var e = window.React.createElement;
        var ReactDOM = window.ReactDOM;

        class MetadataTable extends React.Component {
            constructor(props) {
                super(props);
                this.state = {
                    headers: [
                        { key: 'name', label: 'Name' },
                        { key: 'type', label: 'Type' },
                        { key: 'lastUpdated', label: 'Last Updated' },
                        { key: 'value', label: 'Value' },
                        { key: 'additionalActions', label: 'More...' }
                    ],
                    data: [
                        { 'name': 0, 'value': 'hello'},
                        { 'name': 1, 'value': 'react'},
                        { 'name': 2, 'value': 'without'},
                        { 'name': 3, 'value': 'jsx'},
                    ]
                };
            }

            getTableHeaders() {
                var tableHeaders = [];
                for(var i = 0; i < this.state.headers.length; i++) {
                    var header = this.state.headers[i];
                    var uniqueKey = 'table_header_' + header.key;

                    // Push table header cells into the header row
                    tableHeaders.push(e('th', { 'key': uniqueKey, 'width': `${100 / this.state.headers.length}%
` }, `${header.label}`));
                }

                // Accumulate and return table header elements
                return tableHeaders;
            }

            getTableRowData() {
                var tableRows = [];
                for(var i = 0; i < this.state.data.length; i++) {
                    var item = this.state.data[i];
                    var uniqueKey = 'table_row_data_' + i.toString();
                    var rowData = [];

                    // Loop over selected headers to print table row values
                    for(var j = 0; j < this.state.headers.length; j++) {
                        var header = this.state.headers[j];
                        var uniqueKey = 'table_' + header.key.toString() + '_' + i.toString();

                        // Push table cell values into a "row"
                        rowData.push(e('td', { 'key': uniqueKey, 'width': `${100 / this.state.headers.length}%
`}, `${item[header.key]}`));
                    }

                    // Push rows of cells into the table
                    tableRows.push(e('tr', { 'key': uniqueKey }, rowData));
                }

                // Accumulate and return table row elements
                return tableRows;
```

```
        }

        handleNewValueChange(event) {
            console.debug("OnChange Event:", event);
            this.setState({ 'newValue': event.target.value });
        }

        addNew(event) {
            console.debug("OnClick Event:", event);
            let data = this.state.data;
            data.push({ 'name': this.state.data.length, 'value': this.state.newValue });
            this.setState({ 'data': data });
            this.setState({ 'newValue': '' });
        }

        render() {
            // Re-fetch selected headers and row data
            var tableHeaders = this.getTableHeaders();
            var tableRows = this.getTableRowData();

            // Append an additional row with a submit button to the form
            tableRows.push(e('tr', { 'key': 'submitBtn' },
                e('td', {},
                    e('input', {
                        'placeholder': 'Dummy text input...',
                        'value': this.state.newValue,
                        onChange: (event) => this.handleNewValueChange(event)
                    })
                ),
                e('td', {},
                    e('button', {
                        'className': 'btn btn-xs btn-primary',
                        onClick: (event) => this.addNew(event)
                    }, 'Add new')
                )
            ));

            // Render the table data with React
            return e('div', { 'className': 'table-responsive' },
                       e('table', { 'className': 'table table-condensed table-hover'},
                           e('thead', {},
                               e('tr', { 'key': 'headers' }, tableHeaders)),
                           e('tbody', {}, tableRows))
                    );
        }
    }

    console.log("Rendering!");
    var domContainer = document.querySelector('#react_test');
    ReactDOM.render(e(MetadataTable), domContainer);
    console.log("Rendered!");
</script>
```

With JSX

Now, here is the same component converted to use JSX syntax - note that the browser can no longer directly interpret this code, which must be compiled or "transpiled" using Babel or something similar. Since we will need to compile the JSX down to CommonJS anyways, we can safely use ES6 syntax (such as the "let" keyword or arrow functions) without worrying about browser support, since the compiler should translate these shorthands to their equivalent CommonJS representation.

Notice that we are no longer passing around the values returned from `React.createElement` ... in fact, we don't even explicitly call `React.createElement` at all:

react-jsx.index.html

```
    <!-- Load Bootstrap CSS from CDN -->
    <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" integrity="
sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u" crossorigin="anonymous">
    <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap-theme.min.css"
```

```
integrity="sha384-rHyoN1iRsVXV4nD0JutlnGaslCJuC7uwjduW9SVrLvRYooPp2bWYgmgJQIXwl/Sp" crossorigin="anonymous">

    <!-- Load React from CDN -->
    <!-- Note: when deploying to production, replace "development.js" with "production.min.js". -->
    <script src="//unpkg.com/react@16/umd/react.development.js" crossorigin></script>
    <script src="//unpkg.com/react-dom@16/umd/react-dom.development.js" crossorigin></script>
    <div id="react_test">Component failed to load</div>

    <!-- STOP! The contents of the following script tag can be compiled with the debugger console, located
here: https://babeljs.io/en/repl -->
    <!-- See react-jsx-compiled.index.html for compiled browser-ready results -->
    <script>
        'use strict';

        let ReactDOM = window.ReactDOM;

        // TypeScript used here - feels like a more formal JavaScript
        class MetadataTable extends React.Component {
            constructor(props) {
                super(props);
                this.state = {
                    headers: [
                        { key: 'name', label: 'Name' },
                        { key: 'type', label: 'Type' },
                        { key: 'lastUpdated', label: 'Last Updated' },
                        { key: 'value', label: 'Value' },
                        { key: 'additionalActions', label: 'More...' }
                    ],
                    data: [
                        { 'name': 0, 'value': 'hello'},
                        { 'name': 1, 'value': 'react'},
                        { 'name': 2, 'value': 'with'},
                        { 'name': 3, 'value': 'jsx'},
                    ],
                    newValue: ''
                };
            }

            getTableHeaders() {
                let tableHeaders = [];
                for(let i = 0; i < this.state.headers.length; i++) {
                    let header = this.state.headers[i];
                    let uniqueKey = 'table_header_' + header.key;

                    // Push table header cells into the header row
                    // Notice that we no longer return an HTML element here
                    tableHeaders.push({ 'key': uniqueKey, 'width': (100 / this.state.headers.length).toString()
+ '%', 'label': header.label });
                }

                // Accumulate and return table header data
                return tableHeaders;
            }

            getTableRowData() {
                let tableRows = [];
                for(let i = 0; i < this.state.data.length; i++) {
                    let item = this.state.data[i];
                    let uniqueKey = 'table_row_data_' + i.toString();
                    let rowData = [];

                    // Loop over selected headers to print table row values
                    for(let j = 0; j < this.state.headers.length; j++) {
                        let header = this.state.headers[j];
                        let uniqueKey = 'table_' + header.key.toString() + '_' + i.toString();

                        // Push table cell values into a "row"
                        // Notice that we no longer return an HTML element here
                        rowData.push({ 'key': uniqueKey, 'width': (100 / this.state.headers.length).toString()
+ '%', 'label': item[header.key] });
                    }
```

```
                // Push the row of cell data into the table
                // Notice that we no longer return an HTML element here
                tableRows.push({ key: uniqueKey, 'data': rowData });
            }

            // Accumulate and return table row data
            return tableRows;
        }

        handleNewValueChange(event) {
            console.debug("OnChange Event:", event);
            this.setState({ 'newValue': event.target.value });
        }

        addNew(event) {
            console.debug("OnClick Event:", event);
            let data = this.state.data;
            data.push({ 'name': this.state.data.length, 'value': this.state.newValue });
            this.setState({ 'data': data });
            this.setState({ 'newValue': '' });
        }

        render() {
            // Re-fetch selected headers and row data
            let tableHeaders = this.getTableHeaders();
            let tableRows = this.getTableRowData();

            // Render the table data with React
            // NOTE: JSX syntax is returned - feels a bit like Scala
            return (
                <div className='table-responsive'>
                    <table className='table table-condensed table-hover'>
                        <thead>
                            <tr>
                                {tableHeaders.map((item, i) => {
                                    // For each column header in tableHeaders
                                    return (<th key={item.key} width={item.width}>{item.label}</th>);
                                })}
                            </tr>
                        </thead>
                        <tbody>
                            {tableRows.map((row, i) => {
                                // For each row in tableRows
                                return (
                                    <tr key={row.key}>
                                        {row.data.map((cell, j) => {
                                            // For each cell in the row
                                            return (<td key={cell.key} width={cell.width}>{cell.label}<
/td>);

                                        })}
                                    </tr>
                                );
                            })}
                            <tr>
                                <td>
                                    <input placeholder='Dummy text input...' value={this.state.newValue}
onChange={this.handleNewValueChange} />
                                </td>
                                <td>
                                    <button className='btn btn-xs btn-primary' onClick={(event) => { this.
addNew(event, this.state.newValue) }}>Add new</button>
                                </td>
                            </tr>
                        </tbody>
                    </table>
                </div>
            );
        }
    }
```

```
        console.log("Rendering!");
        let domContainer = document.querySelector('#react_test');
        ReactDOM.render(
            <MetadataTable />,
            domContainer
        );
        console.log("Rendered!");
    </script>
```

Compilation Step

JSX requires us to compile down to JavaScript - to do this, we simply pass our source code through a Babel compiler.

There are many build tools / plugins which can perform this step as well, but I chose to use the Babel web debugger for this simple example.

The result of the compilation might look as follows - Note that the compiler has replaced the JSX parts in the code below with nested calls to `React.createElement()`.

This compiled output can be then immediately consumed by your browser:

react-jsx-compiled.index.html

```
    <!-- Load Bootstrap CSS from CDN -->
    <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" integrity="
sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u" crossorigin="anonymous">
    <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap-theme.min.css"
integrity="sha384-rHyoN1iRsVXV4nD0JutlnGaslCJuC7uwjduW9SVrLvRYooPp2bWYgmgJQIXwl/Sp" crossorigin="anonymous">

    <!-- Load React from CDN -->
    <!-- Note: when deploying to production, replace "development.js" with "production.min.js". -->
    <script src="//unpkg.com/react@16/umd/react.development.js" crossorigin></script>
    <script src="//unpkg.com/react-dom@16/umd/react-dom.development.js" crossorigin></script>
    <div id="react_test">Component failed to load</div>

    <!-- STOP! The contents of the following script tag were compiled using the debugger console, located here:
https://babeljs.io/en/repl -->
    <!-- See react-jsx.index.html for original compilation source -->
    <script>
        'use strict';

        var _createClass = function () { function defineProperties(target, props) { for (var i = 0; i < props.
length; i++) { var descriptor = props[i]; descriptor.enumerable = descriptor.enumerable || false; descriptor.
configurable = true; if ("value" in descriptor) descriptor.writable = true; Object.defineProperty(target,
descriptor.key, descriptor); } } return function (Constructor, protoProps, staticProps) { if (protoProps)
defineProperties(Constructor.prototype, protoProps); if (staticProps) defineProperties(Constructor,
staticProps); return Constructor; }; }();

        function _classCallCheck(instance, Constructor) { if (!(instance instanceof Constructor)) { throw new
TypeError("Cannot call a class as a function"); } }

        function _possibleConstructorReturn(self, call) { if (!self) { throw new ReferenceError("this hasn't
been initialised - super() hasn't been called"); } return call && (typeof call === "object" || typeof call ===
"function") ? call : self; }

        function _inherits(subClass, superClass) { if (typeof superClass !== "function" && superClass !== null)
{ throw new TypeError("Super expression must either be null or a function, not " + typeof superClass); }
subClass.prototype = Object.create(superClass && superClass.prototype, { constructor: { value: subClass,
enumerable: false, writable: true, configurable: true } }); if (superClass) Object.setPrototypeOf ? Object.
setPrototypeOf(subClass, superClass) : subClass.__proto__ = superClass; }

        var ReactDOM = window.ReactDOM;

        // TypeScript used here - feels like a more formal JavaScript

        var MetadataTable = function (_React$Component) {
            _inherits(MetadataTable, _React$Component);

            function MetadataTable(props) {
                _classCallCheck(this, MetadataTable);
```

```
                var _this = _possibleConstructorReturn(this, (MetadataTable.__proto__ || Object.getPrototypeOf
(MetadataTable)).call(this, props));

                _this.state = {
                    headers: [{ key: 'name', label: 'Name' }, { key: 'type', label: 'Type' }, { key:
'lastUpdated', label: 'Last Updated' }, { key: 'value', label: 'Value' }, { key: 'additionalActions', label:
'More...' }],
                    data: [{ 'name': 0, 'value': 'hello' }, { 'name': 1, 'value': 'react' }, { 'name': 2,
'value': 'with' }, { 'name': 3, 'value': 'jsx' }],
                    newValue: ''
                };
                return _this;
            }

            _createClass(MetadataTable, [{
                key: 'getTableHeaders',
                value: function getTableHeaders() {
                    var tableHeaders = [];
                    for (var i = 0; i < this.state.headers.length; i++) {
                        var header = this.state.headers[i];
                        var uniqueKey = 'table_header_' + header.key;

                        // Push table header cells into the header row
                        tableHeaders.push({ 'key': uniqueKey, 'width': (100 / this.state.headers.length).
toString() + '%', 'label': header.label });
                    }

                    // Accumulate and return table header data
                    return tableHeaders;
                }
            }, {
                key: 'getTableRowData',
                value: function getTableRowData() {
                    var tableRows = [];
                    for (var i = 0; i < this.state.data.length; i++) {
                        var item = this.state.data[i];
                        var uniqueKey = 'table_row_data_' + i.toString();
                        var rowData = [];

                        // Loop over selected headers to print table row values
                        for (var j = 0; j < this.state.headers.length; j++) {
                            var header = this.state.headers[j];
                            var _uniqueKey = 'table_' + header.key.toString() + '_' + i.toString();

                            // Push table cell values into a "row"
                            rowData.push({ 'key': _uniqueKey, 'width': (100 / this.state.headers.length).
toString() + '%', 'label': item[header.key] });
                        }

                        // Push the row of cell data into the table
                        tableRows.push({ key: uniqueKey, 'data': rowData });
                    }

                    // Accumulate and return table row data
                    return tableRows;
                }
            }, {
                key: 'handleNewValueChange',
                value: function handleNewValueChange(event) {
                    console.debug("OnChange Event:", event);
                    this.setState({ 'newValue': event.target.value });
                }
            }, {
                key: 'addNew',
                value: function addNew(event) {
                    console.debug("OnClick Event:", event);
                    var data = this.state.data;
                    data.push({ 'name': this.state.data.length, 'value': this.state.newValue });
                    this.setState({ 'data': data });
                    this.setState({ 'newValue': '' });
```

```
                }
            }, {
                key: 'render',
                value: function render() {
                    var _this2 = this;

                    // Re-fetch selected headers and row data
                    var tableHeaders = this.getTableHeaders();
                    var tableRows = this.getTableRowData();

                    // Render the table data with React
                    // NOTE: JSX syntax is returned - feels a bit like Scala
                    return React.createElement(
                        'div',
                        { className: 'table-responsive' },
                        React.createElement(
                            'table',
                            { className: 'table table-condensed table-hover' },
                            React.createElement(
                                'thead',
                                null,
                                React.createElement(
                                    'tr',
                                    null,
                                    tableHeaders.map(function (item, i) {
                                        // For each column header in tableHeaders
                                        return React.createElement(
                                            'th',
                                            { key: item.key, width: item.width },
                                            item.label
                                        );
                                    })
                                )
                            ),
                            React.createElement(
                                'tbody',
                                null,
                                tableRows.map(function (row, i) {
                                    // For each row in tableRows
                                    return React.createElement(
                                        'tr',
                                        { key: row.key },
                                        row.data.map(function (cell, j) {
                                            // For each cell in the row
                                            return React.createElement(
                                                'td',
                                                { key: cell.key, width: cell.width },
                                                cell.label
                                            );
                                        })
                                    );
                                }),
                                React.createElement(
                                    'tr',
                                    null,
                                    React.createElement(
                                        'td',
                                        null,
                                        React.createElement('input', { placeholder: 'Dummy text input...',
value: this.state.newValue, onChange: this.handleNewValueChange })
                                    ),
                                    React.createElement(
                                        'td',
                                        null,
                                        React.createElement(
                                            'button',
                                            { className: 'btn btn-xs btn-primary', onClick: function onClick
(event) {
                                                    _this2.addNew(event, _this2.state.newValue);
                                                } },
                                            'Add new'
```

```
                                        )
                                    )
                                )
                            )
                        )
                    );
                }
            }]);

            return MetadataTable;
        }(React.Component);

        console.log("Rendering!");
        var domContainer = document.querySelector('#react_test');
        ReactDOM.render(React.createElement(MetadataTable, null), domContainer);
        console.log("Rendered!");
    </script>
```