Server-Side Rendering of Isomorphic Apps at SoundCloud

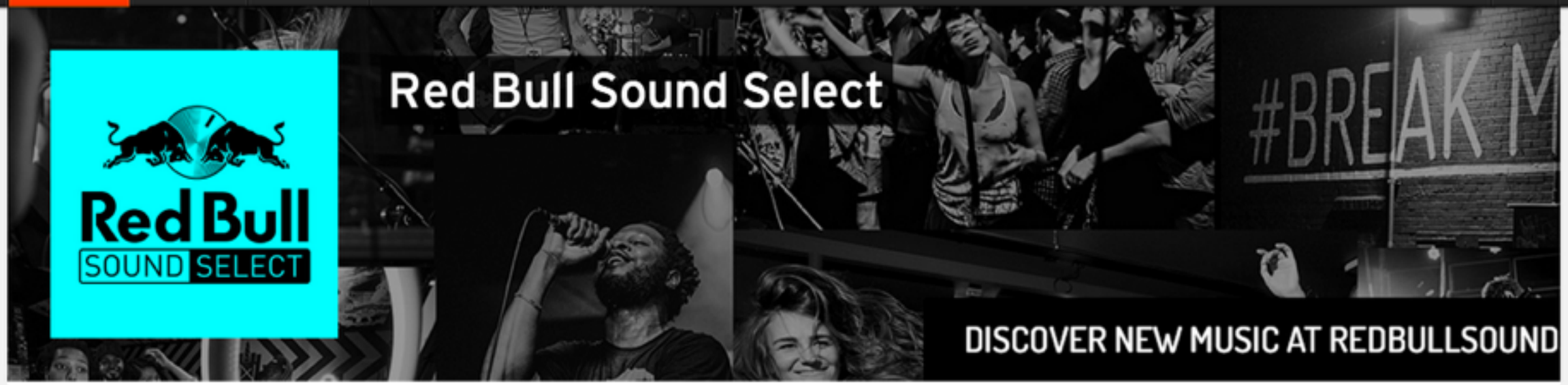
Andres Suarez
@zertosh

```
me = {
   name: "Andres Suarez",
   title: "Front-End Engineer",
   work: "SoundCloud",
   where: "New York, NY",
   team: "Revenue development"
}
```

"It’s a platform for people to discover new, original, music and audio, for creators to build audiences, and for everyone to share what they hear whether online or on mobile."

https://soundcloud.com/redbullsoundselect

Stream Explore Search Upload zertosh



Follow Share

45,668 followers

91 | 7

Discover new music at
www.redbullsoundselect.com

Instagram
Twitter
Facebook
RedBullSoundSelect.com
Block Red Bull Sound Select

Red Bull Sound Select
29 days
Thurz- 21 (produced by Marlon Travis Barrow)
#thurz

4:06

Like Repost Add to playlist Share

12,923 | 302 | 64 | 18

Red Bull Sound Select
7 months
Red Bull Sound Select Presents...
#rock, electronic, da...

4:32

Like Playlist Repost Playlist Share

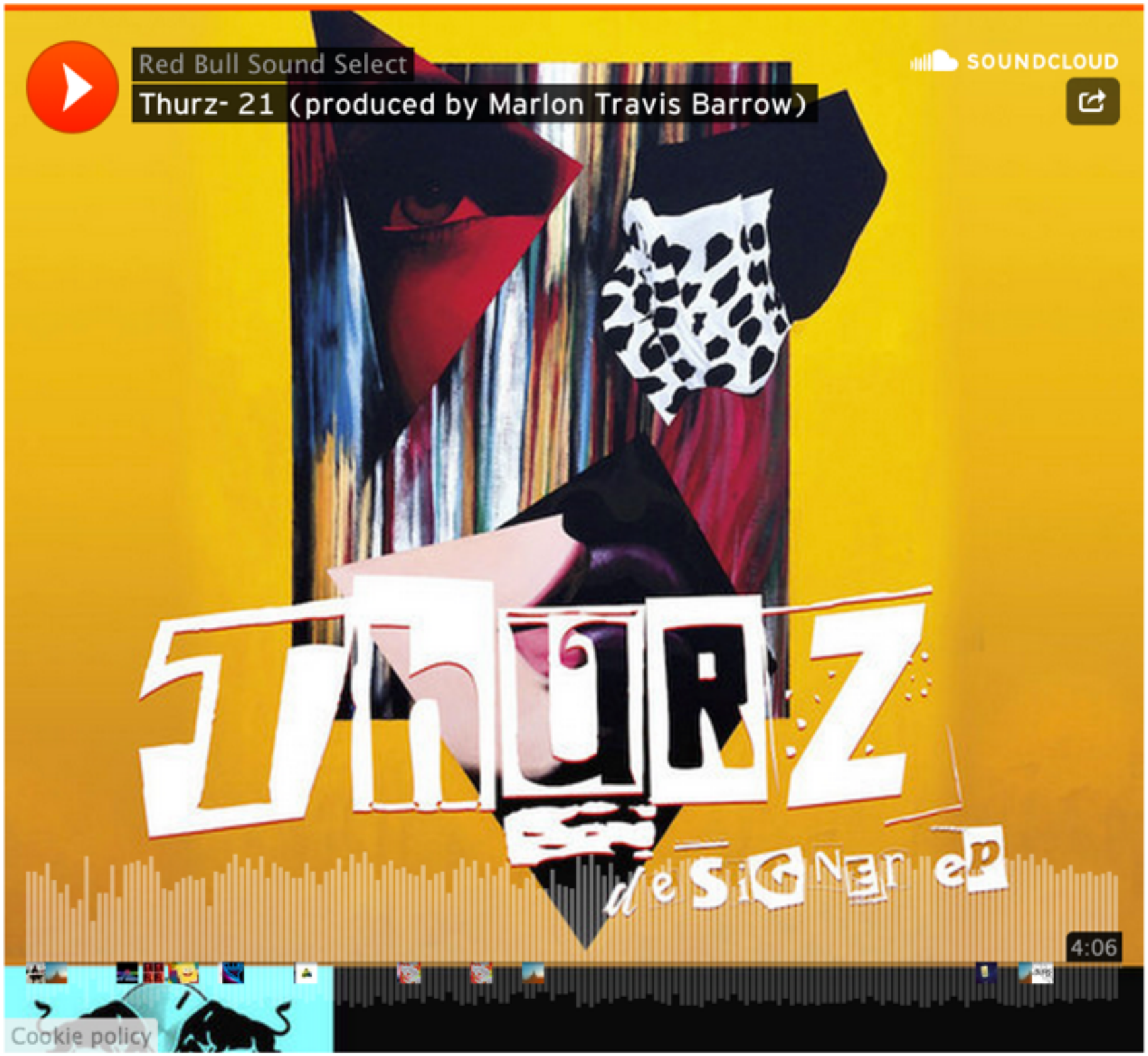
356 | 93

1 Wrestlers - Say Anything (Feat. Twin Shadow and D'Angelo Lacy) 474,921
2 Tapioca and the Flea- Take It Slow (Produced by Ethan Kaith of Crystal Castles) 401,973
3 Denitia and Sene- Divided (Produced by Christian Rich) 441,660
4 Thurz- High Castle (Produced by Christian Rich)

Red Bull SoundSelect @RBSoundSelect · Oct 2

Tag a #throwbackthursday of you at 21 for @Thurzday's new track. We want to see the most embarassing pics you got bit.ly/thurz21

SoundCloud



Server-Side Rendering (SSR)

Using code on the server-side to generate markup for the client.

We've been doing it forever: PHP, Rails, Node.js, etc...

Isomorphic Applications

Apps that share code between the client and the server.

Since JavaScript is the *only* choice on the client, that means:

A **browser** (Chrome, Firefox, IE, etc.) on the **"client-side"**,

And some runtime environment like **Node.js** on the **"server-side"**.

Server-Side Rendering !== Isomorphic Application

The goal is to leverage isomorphic code to render an application both server-side and client-side.

The Promise of Isomorphic Apps

All the goodness of SSR with the ease of client-side rendering:

Faster (perceived) performance,

Proper search engine indexing,

No code duplication,

Same language and the same tools,

Without javascript the user still sees something.

Isomorphic Spectrum

"In Pursuit of the Holy Grail: Building Isomorphic JavaScript Apps" by Spike Brehm (Airbnb)

None — Some sharing — Full isomorphic

Embedded player
Bare HTML with full client-side rendering, ("classical" SPA)

soundcloud.com (v2)
Shared helpers and templates, mostly the view layer

advertising.soundcloud.com, contests platform, internal tools & cool-new-upcoming-thing
Full application is shared

Few abstractions ← → More abstractions

Why Abstractions?

Client

Environment: The client has the DOM and **`window`**.

Culture: It's *okish* to leak globals (a.k.a trash your environment).

Server

Environment: The the server has **`module.exports`**, **`process.env`**, and **`global`**.

Culture: *Not ok* to leak globals.

Using Tooling to Abstract: Browserify

"Browserify is a tool for compiling node-flavored commonjs modules for the browser" - https://github.com/substack/browserify-handbook

Shims node builtins like **`Buffer`**, **`process`**, and **`global`**; even node core libraries like **`events`** and **`url`**.

Allows for using **`npm`** to manage dependencies.

Browserified modules also continue to work in Node*.

Using Libraries to Abstract: React

Abstracts the DOM by creating a *virtual DOM*.

Allows **rendering** as **DOM** or as string of **HTML**.

React can "adopt" existing client-side DOM from server-side generated HTML.

React Components have methods tailored for client-only and client/server environments.

Many more features...

Our Approach to Isomorphic Apps

Tools and Libraries

Node.js + Browserify + React

Architecture

Flux + Immutable data structures

How it works

The entire application is bundled up with Browserify; it exposes a `start` method capable of returning the app's HTML or mounting it to the DOM.

The server `require`'s the app's built module, calls the `start` method and serves the returned HTML.

The client loads the app's module via a `<script>` tag, calls the `start` method and it mounts itself to the DOM.

```
// main.js
var React = require('react');
var ExecutionEnvironment = require('react/lib/ExecutionEnvironment');
var DOM = React.DOM;
var App = React.createClass({
  render: function() {
    return DOM.html({lang: 'en'},
      DOM.head(null,
        DOM.meta({charSet: 'utf-8'}),
        DOM.title(null, 'minimal demo')
      ),
      DOM.body(null,
        DOM.div(null, 'minimal demo application'),
        DOM.script({src: '/main-built.js'}),
        DOM.script({dangerouslySetInnerHTML: {__html:
          'Application.start();'
        }})
      )
    );
  }
});
```

Go to GitHub

```
function start() {
  if (ExecutionEnvironment.canUseDOM) {
    React.renderComponent(App(), document);
  } else {
    return '<!DOCTYPE html>' + React.renderComponentToString(App());
  }
}

module.exports.start = start;


// server.js
var express = require('express');
var server = express();
server.use('/', express.static('./'));
server.get('*', function(req, res) {
  var Application = require('./main-built');
  var html = Application.start();
  res.send(html);
});
server.listen(8000);
```

Why Build a Single App Module?

Forces you to write better abstractions by getting you to think about your Application as single app that happens to run on Node and in a browser.

Get for free any build step transforms and optimizations (JSX, ES6-to-ES5, dead code elimination, variable substitutions, etc.) on the server.

No surprises, since you're literally running the same code on both sides.

The Truth

We don't actually use React to render the layout (`<html>`, `<head>` and `<script>`), we use React to render the Application.

We use a barebones underscore template for the layout, with a slot for the app's HTML.

Why? **Mostly for debuggability.** Makes it easy to "turn off" SSR and go full traditional client-side rendering (*node-inspector* is not as great as the real browser Chrome Dev Tools).

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>SoundCloud</title>
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
    <div id="ReactRootElement"><%= markup %></div>
    <script src="script.js"></script>
    <script>Application.start();</script>
  </body>
</html>
```

On the client, the Application's `start` method mounts the App into `#ReactRootElement`. On the server it returns the Application's HTML.

```
// server.js
server.get('*', function(req, res) {
  var Application = require('./main-built');
  var markup = Application.start();
  res.send( layout({markup: markup}) );
});

// main.js
function start() {
  if (ExecutionEnvironment.canUseDOM) {
    React.renderComponent(App(), document.getElementById('ReactRootElement'));
  } else {
    return React.renderComponentToString(App());
  }
}
```

hello-world demo

`hello-world` looks a lot like our Apps.

It follows Facebook's Flux architecture (see https://github.com/facebook/flux).

However, it does not use immutable data structures for application state.

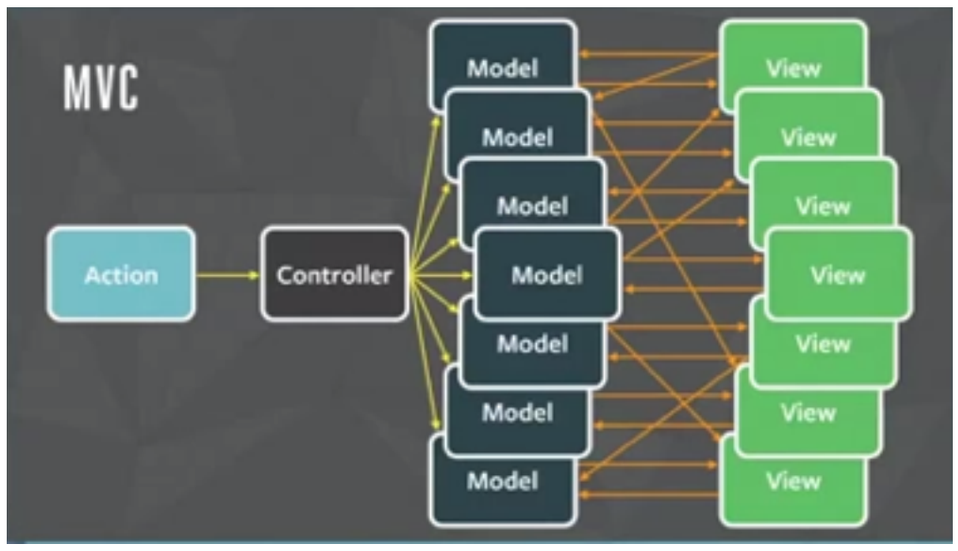
Some components use JSX and some don't- so those that don't care for JSX can still follow along.

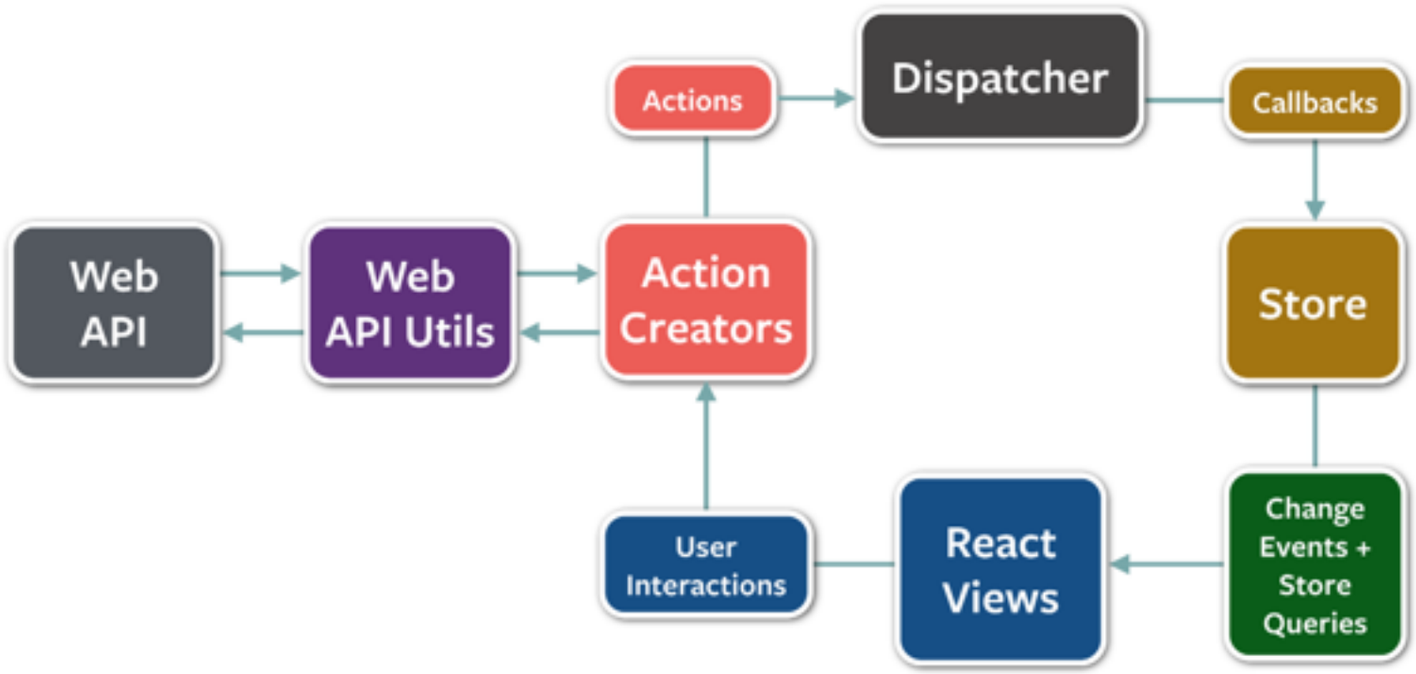
Find it at **https://github.com/zertosh/ssr-demo-kit**

hello-world demo: flux?

"Flux [is a] pattern [that] eschews MVC in favor of a unidirectional data flow." (https://github.com/facebook/flux)

Flux-related vocabulary: Action/ActionCreator, Dispatcher, Stores, Utils, and *APIUtils.





hello-world demo: Application.start()

```
// app/main.js

var Application = {
  start: function(bootstrap) {
    // Ready the stores
    AppActions.initialize(bootstrap);

    // Client-side: mount the app component
    if (ExecutionEnvironment.canUseDOM) {
      var rootElement = document.getElementById(LayoutConfig.ROOT_ELEMENT_ID);
      React.renderComponent(App(), rootElement);
    } else {
    // Server-side: return the app's html
      var rootComponentHTML = React.renderComponentToString(App());
      return rootComponentHTML;
    }
  }
};

// Modules needed server-side
if (!ExecutionEnvironment.canUseDOM) {
  Application.RouteUtils = require('./utils/RouteUtils');
}
```

hello-world demo: AppActions.initialize()

```
// app/actions/AppActions.js

var AppActions = {

  initialize: function(bootstrap) {
    if (!bootstrap) bootstrap = {};
    var path = RouteUtils.getBestAvailablePath(bootstrap);
    var page = RouteUtils.getPage(path);
    var action = {
      type: ActionTypes.APP_INITIALIZE,
      path: path,
      page: page
    };
    AppDispatcher.handleServerAction(action);
  }

};
```

hello-world demo: RouteUtils.getBestAvailablePath()

```
// app/utils/RouteUtils.js

var RouteUtils = {
  getBestAvailablePath: function(options) {
    if (options && options.path) {
      return options.path;
    } else if (ExecutionEnvironment.canUseDOM) {
      return window.location.pathname;
    } else {
      return RouteConfig.DEFAULT_PATH;
    }
  }
};
```

hello-world demo: AppActions.initialize()

```
// app/actions/AppActions.js

var AppActions = {

  initialize: function(bootstrap) {
    if (!bootstrap) bootstrap = {};
    var path = RouteUtils.getBestAvailablePath(bootstrap);
    var page = RouteUtils.getPage(path);
    var action = {
      type: ActionTypes.APP_INITIALIZE,
      path: path,
      page: page
    };
    AppDispatcher.handleServerAction(action);
  }

};
```

hello-world demo: AppStore

```
// app/stores/AppStore.js

AppStore.dispatchToken = AppDispatcher.register(function(payload) {
  var action = payload.action;

  switch (action.type) {
    case ActionTypes.APP_INITIALIZE:
      reset();
      /* falls through */
    case ActionTypes.SWITCH_PAGE:
      appState.page = action.page;
      appState.path = action.path;
      break;

    case ActionTypes.APP_RESET:
      reset();
      break;

    default:
      return;
  }

  AppStore.emitChange();
});
```

The Stores then hold the Application state

hello-world demo: Application.start()

```
// app/main.js

var Application = {
  start: function(bootstrap) {
    // Ready the stores
    AppActions.initialize(bootstrap);

    // Client-side: mount the app component
    if (ExecutionEnvironment.canUseDOM) {
      var rootElement = document.getElementById(LayoutConfig.ROOT_ELEMENT_ID);
      React.renderComponent(App(), rootElement);
    } else {
    // Server-side: return the app's html
      var rootComponentHTML = React.renderComponentToString(App());
      return rootComponentHTML;
    }
  }
};

// Modules needed server-side
if (!ExecutionEnvironment.canUseDOM) {
  Application.RouteUtils = require('./utils/RouteUtils');
}
```

hello-world demo: App Component

// app/components/App.js

```
function getPageComponent(page) {
  switch (page) {
    case Pages.HOME:        return require('./Home.jsx');
    case Pages.HELLO_WORLD: return require('./HelloWorld.jsx');
    case Pages.NOT_FOUND:   return require('./NotFound');
    default:
      throw new Error('Missing "Pages.' + page + '"');
  }
}

var App = React.createClass({
  getInitialState: function() {
    return {
      appState: AppStore.getState()
    };
  },
  render: function() {
    var appState = this.state.appState;
    var PageComponent = getPageComponent(appState.page);
    return PageComponent({appState: appState});
  }
});
```

hello-world demo: Application.start()

```
// app/main.js

var Application = {
  start: function(bootstrap) {
    // Ready the stores
    AppActions.initialize(bootstrap);

    // Client-side: mount the app component
    if (ExecutionEnvironment.canUseDOM) {
      var rootElement = document.getElementById(LayoutConfig.ROOT_ELEMENT_ID);
      React.renderComponent(App(), rootElement);
    } else {
    // Server-side: return the app's html
      var rootComponentHTML = React.renderComponentToString(App());
      return rootComponentHTML;
    }
  }
};

// Modules needed server-side
if (!ExecutionEnvironment.canUseDOM) {
  Application.RouteUtils = require('./utils/RouteUtils');
}
```

hello-world demo: Application.start()

```
// app/main.js

var Application = {
  start: function(bootstrap) {
    // Ready the stores
    AppActions.initialize(bootstrap);

    // Client-side: mount the app component
    if (ExecutionEnvironment.canUseDOM) {
      var rootElement = document.getElementById(LayoutConfig.ROOT_ELEMENT_ID);
      React.renderComponent(App(), rootElement);
    } else {
    // Server-side: return the app's html
      var rootComponentHTML = React.renderComponentToString(App());
      return rootComponentHTML;
    }
  }
};

// Modules needed server-side
if (!ExecutionEnvironment.canUseDOM) {
  Application.RouteUtils = require('./utils/RouteUtils');
}
```

hello-world demo: server

```
// server/index.js

server.get('*', function(req, res) {

  var bootstrap = {path: req.path};

  var layoutData = _.defaults({
    applicationStart: 'Application.start(' + htmlescape(bootstrap) + ');',
  }, LayoutConfig);

  var status;

  if (Config.SSR) {
    var Application = require(Config.APPLICATION_FILE);
    var rootComponentHTML = Application.start(bootstrap);
    layoutData.rootComponentHTML = rootComponentHTML;
    status = Application.RouteUtils.hasMatch(req.path) ? 200 : 404;
  } else {
    status = 200;
  }

  res.status(status).send(layout(layoutData));
});
```

hello-world demo: server

```
// server/index.js

server.get('*', function(req, res) {

  var bootstrap = {path: req.path};

  var layoutData = _.defaults({
    applicationStart: 'Application.start(' + htmlescape(bootstrap) + ');',
  }, LayoutConfig);

  var status;

  if (Config.SSR) {
    var Application = require(Config.APPLICATION_FILE);
    var rootComponentHTML = Application.start(bootstrap);
    layoutData.rootComponentHTML = rootComponentHTML;
    status = Application.RouteUtils.hasMatch(req.path) ? 200 : 404;
  } else {
    status = 200;
  }

  res.status(status).send(layout(layoutData));
});
```

Utils vs *APIUtils

```
// ContactFormUtils.js
var ContactFormUtils = {
  validate: function(obj) {
    if ( !(isValidEmail(obj) && isValidName(obj)) ) {
      return false;
    }
    return true;
  }
};

// MailAPIUtils.js
var MailAPIUtils = {
  send: function(data) {
    var xhr = new XMLHttpRequest();
    xhr.open(MailConfig.SEND_METHOD, MailConfig.SEND_URL, true);
    xhr.setRequestHeader('Content-Type', 'application/json');
    // ...
    xhr.send(JSON.stringify(data));
  }
};
```

Lessons: Restrictions Are Good

A single bundled module for the Application resulted in a better app.

Lessons: Remember the Obvious Truth

On the initial render, regardless of the environment, your application has to output HTML.

Lessons: Recycle Bundled Modules

Need React on the server? Don't **`require('react')`**. Expose it from your Application's built module and use that instead.

But don't stress re-requiring the little modules like underscore.

Lessons: Not Everything Is a Component

A lot of open-source router implementations are Components. For us, it made more sense for it to be Util. It can then be used by the server to determine the status code of the request.

Lessons: Flux Is Great

A big mindset shift. While you're getting the hang of it, don't cheat, trust the pattern.

Gotchas/Tips

Debugging is hard.

Use source maps for stack traces when your built Application module throws.

```
// Use source maps in stack traces
if (process.env.NODE_ENV !== 'production') {
  require('source-map-support').install();
}
```

Links

- https://github.com/evanw/**node-source-map-support**
- https://github.com/facebook/**flux**
- https://github.com/facebook/**immutable-js**
- https://github.com/facebook/**react**
- https://github.com/substack/**node-browserify**
- "In Pursuit of the Holy Grail: Building Isomorphic JavaScript Apps" by Spike Brehm (Airbnb)

soundcloud.com/jobs

Andres Suarez
@zertosh

https://github.com/zertosh