**Al Jazeera VOD Application**

Technical Guide

Ver 1, October 30, 2013 by Brian Franklin

# Overview

This document outlines the process involved with creating updated builds of the Al Jazeera VOD. Details are broken down into two sections: back-end build process and front-end build process. Back-end build process, or the process of creating an updated WAR file is necessary to create a new build when back-end files have changed, and the front-end build process is only necessary when active development occurs on the front-end source files. **Note that editorial updates to the VOD (as described in the VOD Management Guide) do not require a front-end build. Please skip to the Back-End Build Process section for creating updated WAR files.**

**Table of Contents**

[Front-End Build Process](#h.f3gz4235njr5)

[Back-End Build Process](#h.gk4pkmj2hbqy)

# **Front-End Build Process**

The Al Jazeera front-end code base exists separately from the back-end code base. The high-level integration workflow is that the front-end is created using stub or test data, a build tool produces production-ready static assets (HTML, CSS, JavaScript, and images) which are then integrated into the back-end JSP templates by hand. If no changes have been made to the HTML structure of the VOD application (and changes are isolated to stylesheets, images, and/or script files), then an updated build of these files can simply be copied and pasted into the VOD back-end source tree.

# Setup

To set up the front-end for development:

1. Ensure node.js is installed. If you're on OSX use Homebrew: brew install nodejs.
2. Clone this repository and cd into the Al-Jazeera-VOD-Front-End directory.
3. Install local dependencies: npm install.
4. Run grunt to kick off a build
5. Point your browser to a static HTML file in the dist/ directory.

#### Troubleshooting

* If you get an error that says grunt: command not found run the command which grunt. If it doesn't find anything try running npm install -g grunt-cli.
* If npm install throws an error, make sure you are cd'd into the Al-Jazeera-VOD-Front-End directory.
* The Bourbon CSS library is packaged in with this repo. You should get it simply by cloning the repo, but if you run into problems, try installing it manually: gem install bourbon

## Directory Structure

* src/ - Location of the front-end source code
* src/scripts/ - Location of all the JavaScript files
* src/scripts/config.js - Require.js configuration file
* src/scripts/app/app.js - The main application file where many configuration options are stored, and which facilitates functionality across the rest of the code base. This app.js file is a dependency for every other Require.js module.
* src/scripts/app/page-\*.js - Includes the dependencies for that page and boots it
* src/scripts/app/views/ - View code
* src/scripts/app/views/common/ - Common views that are not specific to a particular page, like a carousel or pagination.
* src/scripts/app/collections/ - Backbone collections
* src/scripts/app/models/ - Backbone models
* src/scripts/templates/ - Templates that the JavaScript is responsible for rendering
* src/scripts/libs/ - Third party JavaScript libraries
* dist/ - Statically-built version of the front-end
* data/ - Where stub .json files are stored for rendering the front-end Dust templates
* test/ - Unit tests
* node\_modules - Where npm installed all the dependencies
* ./Gruntfile.js - The build file
* ./package.json - The configuration file that defines all build/server dependencies
* ./jshintrc - Configuration options for JSHint

## Libraries and Frameworks Used

The front-end is build on top of the following libraries and frameworks:

**Dust Templates**

All HTML is written inside of Dust templates, which allows for functionality like partials, variables, dynamic rendering, and more.

**jQuery**

Used to facilitate cross-browser DOM and Ajax interactions.

**jQuery UI**

Provides the foundation for many widgets and interactions throughout the portal, like dialogs, drag and drop, etc.

**Underscore.js**

Underscore provides a number of utility JavaScript functions, and is a required dependency of Backbone.js.

**Backbone.js**

Backbone is used to provide an inheritance structure to the application. Most everything you see in the portal is a Backbone view that inherits functionality from other views. For example, within thesrc/scripts/views/common directory exists a single view named "page" which is the base view from which all other pages extend. The home page, detail page, etc. inherit functionality from the page page view. This architecture allows you to place functionality common to all pages in a single view, and override/customize the functionality later on a per-page basis.

**Require.js**

Require is the JavaScript dependency management system. It allows (for example) the detail page to define a list of dependency components: a carousel, navigation, etc.

**Grunt.js**

Grunt is responsible for building the application: compiling SASS files into stylesheets, resolving the Require.js dependencies, minifying the JavaScript, etc.

**SASS**

The CSS is written using the Scss variant of SASS, which allows the CSS to be architected in a reusable and maintainble way.

**Bourbon**

Bourbon is a library written in SASS that provides a number of utilties for making CSS easy to write and less verbose.

## Development Process

1. Add, edit, or delete CSS (Sass files), HTML (Dust files), etc. in the src/ files.
2. Re-build the page where you've made the change and want to re-test. For example, rungrunt page:detail to re-build the video detail page, or run grunt page to re-build all pages.
3. Point your browser to the dist/ directory and look at one of the HTML files.

If an image is changed or added, you must run the grunt copy to copy the image into the dist directory.

### Creating a release build

Once development of a bug/feature is done you need to sync the static files to GitHub so that the back-end engineers can pull in the changes. This is done by creating a release build and then checking in the dist/folder. See the section below for what exactly a release build does.

1. Checkout the release branch for the current epic. For example, git checkout release-epic-8
2. Run the command grunt release
3. Check in the files outputted into the dist/ folder. Since dist is in the .gitignore file, check in with the--force flag: git add --force dist/

The only time you should check in the dist/ directory is after running grunt release - do not check in at any other time unless you enjoy resolving merge conflicts.

### Building an individual page

If you only want to build one specific page, you can do so by running this command:

grunt page:detail

This example will only build the CSS, HTML, and JavaScript for the detail page. Available pages for building are:home, detail, channel, yourchannel, allshows, show, topics, topic, live, error, and search.

### What the build process is doing

When you run the command grunt the following happens:

1. Runs all JavaScript files in the src/ directory against JSHint to ensure code quality and consistency. If JSHint detects an error then the build will fail.
2. Runs the require.js build tool against all files in the src/scripts directory, and outputs the results into thedist/scripts directory. Each page in the portal will have a single .js file with all the relevant modules.
3. Processes all the Sass files in src/styles and outputs generated CSS files to dist/styles.
4. Renders all the Dust templates in src/templates against the data files in data/. The result is static HTML files which are placed in the dist/ directory.
5. Injects a build number into the JavaScript files based upon what is defined in package.json.
6. (RELEASE ONLY) runs any images through an image optimizer.
7. (RELEASE ONLY) minifies the Javascript and CSS.
8. (RELEASE ONLY) removes all comments from the Javascripts.
9. (RELEASE ONLY) runs the unit tests.

## Unit Tests

Tests are written using the [http://visionmedia.github.com/mocha/](https://github.com/BrightcovePS/Al-Jazeera-VOD-Front-End/blob/develop-epic-9/Mocha) framework with [http://chaijs.com/](https://github.com/BrightcovePS/Al-Jazeera-VOD-Front-End/blob/develop-epic-9/Chai) for assertions. Run the tests by opening up /tests/mocha/runner.html in your browser, or run them on the command line via the npm test command.

## Third-Party Service Interactions

### Taboola

Taboola powers the recommended videos hero unit on the Your Channel page. All interaction with the Taboola API exists inside the app/scripts/collections/common/taboola.js file. The API key is defined inside theapp/scripts/app.js file.

In order to support recommendations, a heartbeat call is sent to Taboola with current session data and the video ID each time a video begins to play on the detail page. The code that performs the API call is located in thesrc/scripts/app/views/detail/detail.js view.

### Gigya

###### Social login

Available on every page through the "Create an Account" and "Login" links in the header. Users can initiate social login at any time by using these links, but trying to perform various actions (like subscribe, playlist creation, etc.) will prompt the user to login. The view that powers this integration is logged at src/scripts/app/views/common/gigya.js.

###### Social login - linked networks

Authenticated users see a "social networks" link inside the Preferences pane.

###### Comments

The comments panel on the detail page is powered through Gigya.

###### Configuration

The screensets Gigya is capable of rendering are defined inside of the src/scripts/app/app.js file as constants. Each page contains the Gigya embed snippet before the closing </body> tag, which is where the API key, site name, enabled social login providers, and other configuration options are set.

## VOD API Interactions

The front-end makes a number of API calls to the VOD back-end to support various pieces of functionality. These API calls and their function are detailed in the Al Jazeera VOD API Guide

# Back-End Build Process

The Al Jazeera VOD is an API-driven video website created in Java using the Spring framework. Due to the nature of Java applications, changes to core files necessitate a recompilation of the web application. This document assumes the use of a Unix-like operating system during the build process (such as OSX, BSD, Linux, etc). The below functionality can be carried out in Windows, but the CLI commands may differ slightly. It is also possible to perform VOD builds using Eclipse or another Java IDE, but configuration of those tools is beyond the scope of this document. For simplicity, all actions will be performed over a simple command line on a system with Java JDK 1.6 using Maven 3.0.4.

**Requirements**

1. Java JDK 1.6+
2. Maven 3+

## **Preparatio**n

Prior to building the VOD, the source must be extracted into a working directory. Select a directory to keep the VOD source in. It is highly recommended that the VOD is kept under version control. Once extracted, you should have the following file structure:

* front-end
* pages-repository
* pom.xml
* src

The src directory contains the VOD application source as well as copies of the compiled front-end artifacts created through the front-end build process. The front-end directory contains front-end source files (also noted above in the front-end documentation). The pom.xml file is used by Maven in the build process, and the pages-repository directory contains modular codebases used by the VOD (such as Gigya and Brightcove API wrappers).

Before a first build, verify that your system knows where Java and Maven are, and that the versions are correct:

**$ java -version**

The result of this command should print the Java version and installed runtime information. Make sure that 1.6+ is properly being shown.

**$ mvn -v**

The result of this command should print Java and Maven information including the Maven version. Make sure that Apache Maven 3+ is properly being shown.

If both tests return the expected results, it’s time to perform a build. Before each build it’s a good idea to remove the previous binaries (this isn’t necessary on the first build because you don’t have a target directory yet):

**$ rm -rf target/**

To start the build process, perform the following Maven command:

**$ mvn package**

Maven will begin outputting information as it builds. You can generally ignore this process, as Maven is compiling and running unit tests on code. When the build has completed, you will be shown a BUILD SUCCESS message with compilation time information. You will now find binary artifacts in the “target” directory.

**$ ls target**

* **aljazeera-vod.war**
* aljazeera-vod
* classes
* generated-sources
* generated-test-sources
* maven-archiver
* surefire
* surefire-reports
* test-classes

You can ignore these generated directories. The aljazeera-vod.war file is your compiled copy of the VOD application. This WAR file can now be deployed to a Tomcat 7 server or to Amazon Elastic Beanstalk following the VOD Infrastructure & Deployment Guide. Remember to deploy newly built WAR files to test or staging environments to verify before deploying to a live environment.