generator-politico-interactives Documentation

Release 0.4.0

Jon McClure

Jan 09, 2018

Contents:


POLITICO

CHAPTER 1

Why this

At POLITICO we sometimes build custom webpages outside our CMS to showcase a special visual presentation or nonlinear story. We call these pages **interactives**, which is a metonym for pages that offer our readers a richer experience with our content.

This app helps us build those pages.

At its simplest, this app is a Yeoman generator, which means it simply sets up folders and files in a directory on a developer's computer. But its real value is those files, which include a complete build and publishing system for our interactives and enforce our house styles and conventions through templates.

The rest of these docs describe how to work with the build system contained within this generator.

1.1 Principles

The build system creates projects that are...

1.1.1 Decentralized

Every project is an independent application. This keeps our builds lean. It gives us the flexibility to use whatever front-end technology we need for the project at hand and to integrate easily with backends in a decoupled way.

Treating our projects as independent apps means we intentionally skip several conventions that are normally part of content management systems, like centralized revision control and release management.

1.1.2 Batteries included

An app's development environment is built from the command line with a complete set of templates and scripts to build and publish the project, all of which can be overwritten within an individual project.

1.2 What it does

- Scaffolds your project's development directory.
- Compiles SCSS and bundles JS written in either ES5 or ES2015 using your choice of browserify or webpack.
- Renders HTML templates with custom context.
- Creates responsive image sets optimized for mobile devices.
- Publishes your project to an Amazon S3 bucket.

1.3 What's in it

The build pipeline uses:

- Gulp to run tasks
- Babel to transpile ES6 Javascript
- Webpack to bundle scripts
- Express to serve your work in development
- node-sass to compile SCSS
- Sharp for image processing
- ESLint for syntax highlighting
- Nunjucks to compile HTML templates
- ArchieML & archieml-pipe to use Google docs to create template context
- secure-keys to encrypt and decrypt your access keys to AWS, GitHub and other services
- ngrok to share preview pages
- gulp-awspublish to publish to AWS

CHAPTER 2

Installing the generator

2.1 Dependencies

Make sure you have the latest version of node installed on your machine as well as the yarn package manager.

2.2 NPM

Install the package's dependencies globally.

```
$ npm install -g gulp-cli yo generator-politico-interactives
```

To use the Google Spreadsheet integration, you will need gdrive and its authentication setup.

```
$ brew install gdrive

$ gdrive list
```

2.3 Symlink

Alternatively, you can clone a copy of the generator's git repository and use a symlink to install the package. This is especially useful if you'll be developing templates within the generator.

```
$ git clone git@github.com:The-Politico/generator-politico-interactives.git

$ cd generator-politico-interactives

$ npm link
```

Note: To update a symlinked package, just `git pull` the latest changes in the symlinked directory.

CHAPTER 3

Starting an interactive

3.1 From scratch

1. Create a fresh directory for your project and move into it in your terminal.

```
$ mkdir my-project
$ cd my-project
```

2. Now run the generator and answer the questions it asks to build your development environment.

```
$ yo politico-interactives
```

3. Once the generator finishes, you can simply run gulp to start the development server.

```
$ gulp
```

3.2 From GitHub

1. Clone the project and `cd` into the project directory.
2. Run the passphrase subgenerator to create a new `.env` file:

```
$ yo politico-interactives:passphrase
```

3. Install dependencies.

```
$ yarn
```

4. If your project is using ArhcieML, run the archie subgenerator to reconfigure the integration.

```
$ yo politico-interactives:archie
```

Note: If you're cloning a project to use as a template for a new project, delete the `.git` folder in your project root and then initialize a new git repo for the new project.

```
$ rm -rf .git
$ git init
```

CHAPTER 4

Building an interactive

4.1 Working with templates

Templates are rendered using Nunjucks templating syntax. See Nunjucks' template inheritance, tags and builtin filters for details on using the syntax to its full effect.

4.1.1 Template context

Data to go into the template context can come from three places: an ArchieML doc, a spreadsheet, and the meta JSON file. Each of those data sources are prefixed in the template context. So, to use an Archie key, you would write:

```
{{ ARCHIE.key }}
```

Spreadsheet keys are prefixed with `DATA`, and meta keys are prefixed with `META`. For more, take a look at `server/context.js`.

4.1.2 Markdown

There is a custom filter included for rendering template context formatted in Markdown:

```
<!-- Render context data with markdown -->
{{someText|markdown}}

<!-- Remove the outer paragraph tags with the strip option -->
<h1>{{sectionTitle|markdown(strip=true)}}</h1>
```

4.1.3 Adding a new page

To add a new page to your interactive, use the `new-page` subgenerator:

```
$ yo politico-interactives:new-page
```

This will ask you to give a name for your page. You can see your new page by going to `localhost:3000/my-new-page/index.html`.

4.2 Responsive images

To make responsive images that load more quickly on smaller devices, drop a high-res jpg image into the `src/images` directory. If you have `gulp` running, then the image task should run automatically. If `gulp` isn't running. then you can run the image task manually:

```
$ gulp img
```

This is will create four optimized images from your source at 400, 800, 1200 and 1600 pixels width.

4.2.1 srcset macro

You can easily include these images in your template with our custom `jpg` nunjucks macro.

```
<figure>
    {{ jpg('cat', alt='A cat!') }}
    <figcaption>A pretty cat</figcaption>
</figure>
```

Renders as:

```
<figure>
    <img src="images/cat-1800.jpg" srcset="images/cat-400.jpg 400w, images/cat-800.
↪jpg 800w, images/cat-1200.jpg 1200w, images/cat-1800.jpg 1800w" alt="A cat!">
    <figcaption>A pretty cat</figcaption>
</figure>
```

4.2.2 Other image assets

For image assets that should not be converted using our responsive image task, such as svgs and gifs, you should save those directly in `dist/images`.

4.3 Data assets

For data that needs to be used on the front-end (i.e. data for a D3 chart), you should place those files in `dist/data` directly.

If you need to use files in `src/data`, such as data pulled in from `gulp spreadsheet`, those files will be automatically copied to dist when `gulp` is running. To copy files manually, run:

```
$ gulp data
```

4.4 ArchieML

Optionally, there is a gulp task available which allows you to use ArchieML and Google Docs to render content into your templates.

You will be asked if you want to use ArchieML when you start the generator. You can also add it to a project later by running:

```
$ yo politico-interactives:archie
```

You will need to provide the ID for the Google doc you wish to use, which you can get from the URL of your doc:

https://docs.google.com/document/d/**yourGoogleIDhere**/edit

Note: Your document must have access set at least to `Anyone with the link can view` to use this task.

The archie gulp task will access your Google doc and overwrite `src/data/archie.json` with ArchieML data. To run it:

```
$ gulp archie
```

Note: On first running the task, you will need to authorize access to the document through Google. The task will open the authorization dialogue in your browser. Follow the prompts and then copy and paste the code returned by Google.

This access token will be saved in `archie-token.json` so that you can run the task subsequently without needing to re-authorize.

Note: If you've added ArchieML after the project was already created, you'll also need to add the task to your gulpfile. Simply edit it into the array of other tasks in `gulpfile.js`:

```
const gulp = require('./gulp')([
  'aws',
  'archie', // Add this line
  // ...
]);
```

4.5 Spreadsheet

There is an optional gulp task for loading a Google Spreadsheet into JSON for use in your Nunjucks templates (or to load onto the page directly).

To set it up, run:

```
$ yo politico-interactives:spreadsheet
```

This will ask you for a spreadsheet ID. You can get that from the URL of your spreadsheet:

https://docs.google.com/spreadsheets/d/**yourGoogleIDhere**/edit

The spreadsheet gulp task will overwrite `src/data/data.json` with the data from the spreadsheet.

The conversion from spreadsheet to JSON takes each sheet and converts it to JSON using copytext's table converter. This makes each row an object, using the first row as a header row for keys inside the JSON object.

This is customizable at a sheet level in `gulp/tasks/spreadsheet.js`. See the copytext docs for more information on how to customize the parsing.

Note: If you've added the spreadsheet task after the project was already created, you'll also need to add the task to your gulpfile. Simply edit it into the array of other tasks in `gulpfile.js`:

```
const gulp = require('./gulp')([ 'aws', 'archie', 'build', 'dev', 'data', 'data-watch', 'dist', 'html', 'img', 'img-watch', 'spreadsheet', // add this line
]);
```

CHAPTER 5

Publishing

1. Complete the meta information in `meta.json`
2. Run the gulp's publish task.

```
$ gulp publish
```

Note: If you need to invalidate files you've previously published in CloudFront's cache, add the `--invalidate` flag:

```
$ gulp publish --invalidate
```

Your dist folder will be synced to the directory specified under `publishPath` in `meta.json`, which means files in AWS at that location that are *not* in your dist directory will be **deleted**.

The publish task will also version and gzip CSS and JS assets.

5.1 Previewing the rendered page

You can preview the rendered page by running:

```
$ gulp preview
```

This will run the same render process as `gulp publish`, but instead of pushing to AWS, it starts a local server inside the dist folder.

CHAPTER 6

Developing

6.1 Subgenerators

The generator is split into several subgenerators which are called in sequence (and may be called by other generators). Read up on generator composability. These can be called individually:

```
$ yo politico-interactives:<subgenerator>
```

6.1.1 app

The main subgenerator called by defauly when you run the generator.

6.1.2 archie

Sets up an ArchieML integration for the project, if requested.

6.1.3 bundler-webpack

Sets up a webpack-based bundler system.

6.1.4 github

Generates a GitHub repository for the project, if requested.

6.1.5 gulp-common

Common gulp tasks shared across our generators.

6.1.6 gulp-statics

Gulp tasks for building static assets beyond JS and CSS (i.e. Nunjucks templates, image processing).

6.1.7 keys

Will write a set of keys encrypted with a passphrase to a path in the user's directory, `~/.politico/interactives.json`. Writes the following keys:

- AWS access key
- AWS secret key
- Google client ID
- Google client secret key
- GitHub personal access token
- Slack token
- Ngrok token

6.1.8 linters

Sets up ESLint linter config.

6.1.9 meta

Writes meta file.

6.1.10 new-page

Generates a new page in your templates folder.

6.1.11 passphrase

Uses your passphrase to try to decrypt keys file.

6.1.12 router

Router file for Express server.

6.1.13 spreadsheet

Sets up gulp tasks and context loading for using a Google Spreadsheet

6.1.14 styles

Writes scss directory.

6.1.15 templates

Writes HTML directories.