BRIGHTSPOT®

Brightspot Front End Training

Agenda

1. FE stack and responsibilities overview
2. Root styleguide and Views
3. Styleguide application
4. Frontend bundle
5. Handlebars
6. Javascript
7. Styling
8. JSON for examples and configuration
9. Code Example

Overview

What is FE at Brightspot?

FE development on Brightspot encompasses everything you'd expect as a front end developer (HTML, Templating, CSS, JS). But in addition, it also includes controls to help create views and to incorporate publishing features into Brightspot.

Our FE stack

- JSON (for configuration & views)
- Handlebars (HTML templates)
- Less (CSS preprocessor)
- Javascript using ES6 syntax
- Webpack (module bundler)
- Yarn (automate workflow)
- Node.js (running local styleguide)

Root Styleguide

The glue between backend and frontend

Root Styleguide concepts

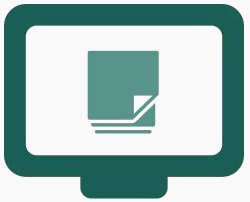
- Originally lived in the project root, hence the name
  - Now located in `frontend/styleguide/`, although older projects still have it in the root
- Specifies views, which are collections of fields
- Each view requires a `.json` file and a `.hbs` file
  - The `.hbs` file should always be empty, it's a renderer hook
- Each view results in a Java interface for the backend to implement

Root Styleguide concepts

- As FE developers, we create the JSON schema and fields, as that is the data we need to show on the site
- The BE devs do the data modeling, create modules inside the CMS, and create the View Models that place the data into the Views
- Our contract is the JSON file, as that creates the View. That’s the data that we need and because the View is auto generated off that data, the BE devs know what data they need to provide us. How they do that is up to the business. Could come from the CMS (most of the time it does), could come from a back end API, could be magic. You’ll learn about that tomorrow.

Styleguide application

Styleguide application



- Styleguide is our Node.js application that runs locally and processes example JSON view files via our Handlebar renderers to simulate what is happening with CMS data entry and rendering on the web
- Uses a dev environment of webpack to compile our JS and CSS
- Since it’s using the exact same code and build systems that the real site does, it allows us to work locally, see our work instantly, and have trust that this same code will work correctly once deployed

Frontend Bundle

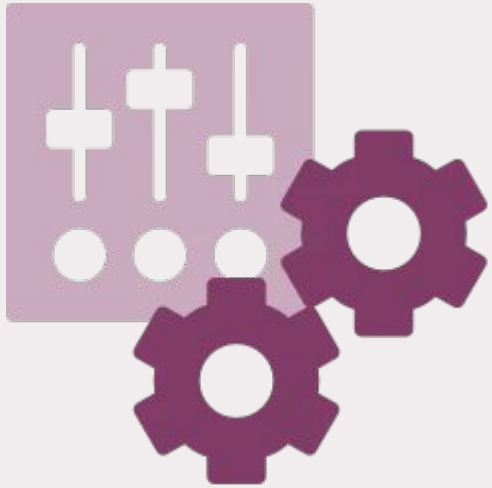
Frontend bundle overview

Originally called a theme, the bundle is the “front end” of the site. It is what we primarily work with as front end developers. It contains:

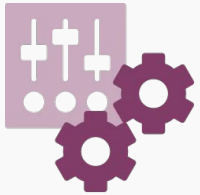
- **HBS** - HTML renderers
- **JS** - for site interactivity
- **Styling** - CSS built from LESS files for styling
- **Assets** - SVG or static image assets that we do not want in the CMS
- **JSON** - Configuration and view examples

Handlebars

Rendering HTML

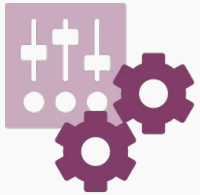


Handlebars overview



- Brightspot uses Handlebars as its templating language
- Handlebars provides strong separation between your templates, styles, and Javascript
- While it is defined as a logicless templating language, Brightspot provides a set of helpers that assist in the composition of templates
- Among those helpers include comparison helpers, logical helpers, math helpers, and more. You also have the option to even create your own helpers via Javascript (comprehensive collection here: http://docs.brightspot.com/themes/templating/helpers.html)

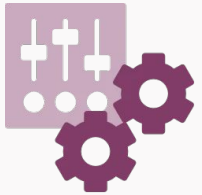
Handlebars helpers



- {{eq}} {{gt}} {{lt}} {{and}} {{or}}
  - Useful comparison helpers that are not in Handlebars by default
  - Great for moving through item arrays with some logic or conditional rendering

- {{set}} and {{get}}
  - Allow us to pass values from parent to children templates

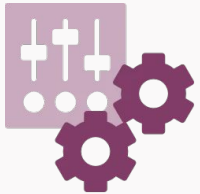
Handlebars helpers for template reuse



- {{include}}
  - Literally includes another template within a template and passes current context
  - Supports template reuse

- {{render}}
  - “Contextual rendering” concept, that allows us to render a specific style of a template

Handlebars helper for image sizing



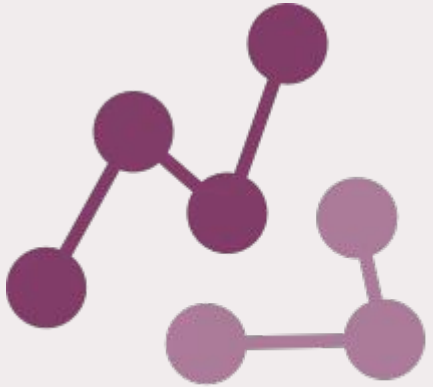
- {{resize}}
  - Directive to receive a specifically resized image from DIMS
  - Can be used along with {{set}} and {{get}} to allow for code reuse and contextual image sizing

https://github.com/perfectsense/training/blob/develop/frontend/bundles/brightspot-training-bundle-default/styleguide/image/ImageTag.hbs#L1

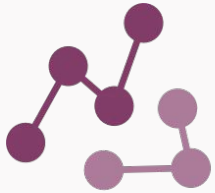
https://github.com/perfectsense/training/blob/develop/frontend/bundles/brightspot-training-bundle-default/styleguide/page/promo/PagePromoModuleB.hbs#L4

Javascript

Let’s make things interactive

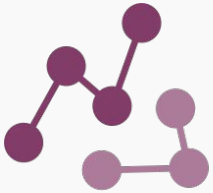


Javascript Overview



- ES6 written JS that is a mixture of classes and custom elements
- Use Babel to transpile JS into browsers we need to support
- Webpack packages our JS into a combined JS and chunks

Javascript Custom Elements



- We mostly use Custom Elements to bind JS to the DOM
- For most content driven “static” websites, there is no need for a full JS framework, as there is no state or views to handle
- Allows us to use JS to make the site more interactive and enhance the HTML vs use it for rendering
- Example:
  https://github.com/perfectsense/training/blob/develop/frontend/bundles/brightspot-training-bundle-default/styleguide/carousel/Carousel.js

Styling

Let’s make things look good

CSS overview

- Style packages allow us to have different stylesheets for the same set of renderers
  - CSS Zen Garden approach for micro sites for certain clients
  - Sites that do not need this can just use a single style package
- LESS as the CSS preprocessor
- BEM style syntax for class/element naming vs OOCSS
  - CSS changes and bug fixes are less likely to create regression issues
  - Allows for multiple developers to work together more easily

CSS continued

- Use descriptive names for modules/components and then use BEM to name their elements
- Use LESS to abstract certain styles
  - Allows us to use LESS extends and mixins, depending on the use case to abstract styles

https://github.com/perfectsense/training/blob/develop/frontend/bundles/brightspot-training-bundle-default/styleguide/styles/style-1/page/list/PageList.less

https://github.com/perfectsense/training/blob/develop/frontend/bundles/brightspot-training-bundle-default/styleguide/styles/style-1/page/list/PageListAbstracts.less

https://github.com/perfectsense/training/blob/develop/frontend/bundles/brightspot-training-bundle-default/styleguide/styles/style-1/page/list/PageListA.less

CSS vars and LESS vars

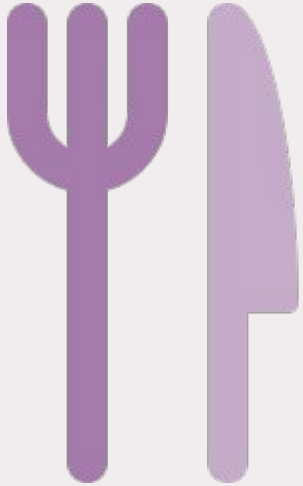
- CSS Vars allow us to set values, whether in CSS or inline HTML
- Allows us to create a pattern to set CSS vars through CMS editorial fields to allow more customization for editors
- Use LESS vars for places where variables are helpful, but overrides are not really necessary

https://github.com/perfectsense/training/blob/develop/frontend/bundles/brightspot-training-bundle-default/styleguide/global/Typography.hbs

https://github.com/perfectsense/training/blob/develop/frontend/bundles/brightspot-training-bundle-default/styleguide/styles/style-1/global/Typography.less

JSON

View examples and configuration



JSON for view examples

- JSON view examples are used in the frontend bundle to simulate how the CMS is going to fill out views
- This allows us to create examples of content types and entire pages locally

https://github.com/perfectsense/training/blob/develop/frontend/bundles/brightspot-training-bundle-default/styleguide/article/ArticlePage.json

https://github.com/perfectsense/training/blob/develop/frontend/bundles/brightspot-training-bundle-default/styleguide/page/HomePage.json

JSON config files - *.config.json

- Configuration files are a mechanism to define properties which impacts the way a bundle interacts with Brightspot and the Styleguide application
- Your configuration files contain:
  - Image size options
  - Additional fields that the FE can add into the CMS
  - Style Variations of content types
  - Configuration options for the Styleguide application
- Configuration files may be defined at the root of your styleguide directory or in a subdirectory. When placed in a subdirectory it is a best practice to keep your configuration local to the components in the same directory. If you accidentally define a configuration for a component outside of the same directory, Styleguide will try to warn you
- The global configuration file lives in the root of your Styleguide directory
- At runtime and build, all the *.config.json files are actually combined into one large configuration

JSON config for image sizing

- Image sizes are defined in a configuration file at the root of your theme. To define an image, you will at a minimum need to define the width (or maximumWidth) and height (or maximumHeight)
- Once defined, these image sizes can be used with the {{resize}} helper to request those particular images from DIMS

https://github.com/perfectsense/training/blob/develop/frontend/bundles/brightspot-training-bundle-default/styleguide/_imageSizes.config.json

JSON config for FE fields

- We can specify additional fields that are added to the styles tab in the CMS, which allow editors additional control over certain styles of content types
- These fields only persist within this particular front end bundle, and are not part of that content type's data
- Very useful for additional styling options for modules, such as background colors, image alignements, etc

https://github.com/perfectsense/training/blob/develop/frontend/bundles/brightspot-training-bundle-default/styleguide/_fields.config.json

https://github.com/perfectsense/training/blob/develop/frontend/bundles/brightspot-training-bundle-default/styleguide/page/promo/_PagePromoModule.config.json

JSON config for styles of content types

- Allows us to define style variations of content types
- Powerful concept that lets us create many different “looks” of a content type, and the same data schema
- For example, style variations of the PageList, PagePromo, and PagePromoModule content types are used to assemble lots of pages like the Homepage, Section Pages, etc

https://github.com/perfectsense/training/blob/develop/frontend/bundles/brightspot-training-bundle-default/styleguide/page/list/_PageList.config.json

https://github.com/perfectsense/training/blob/develop/frontend/bundles/brightspot-training-bundle-default/styleguide/page/promo/_PagePromoModule.config.json

Code Example

Making a new List style

- Step 1 - Add JSON for example and the HBS rendeder
  - https://github.com/perfectsense/training/pull/55/commits/927fe26da2e877bd8ea788008ace4d1ecff71dd2
- Step 2 - Update configuration file for Lists to have new style show up in the CMS
  - https://github.com/perfectsense/training/pull/55/commits/fd40764cb1627e97d3db0f3cf6e9abeb9a428b21
- Step 3 - Add styling for the new List style
  - https://github.com/perfectsense/training/pull/55/commits/067c7ba03d2a506fa49f885354d49b3b3fece6d9
- Step 4 - Update Styleguide configuration to show example of new list style in application
  - https://github.com/perfectsense/training/pull/55/commits/1d8fb2c7169137e7038452e57df6b57476b984d7