Using eleventy

We saw how to create a working templating system using one of many available templating engine (Nunjucks from Mozilla). But it takes a while to get that first step where the template building works as intended.

There are platforms that automate the process. The one that has been on my radar for a while, and was recommended to me, is eleventy

Unlike the previous two strategies, Eleventy requires a dedicated setup and is geared towards Node development rather than Gulp.

Rather than start from scratch I've taken the eleventy base blog template from Github as my starting point. I will also research other areas like creating sections for content other than a blog.

The first steps are to clone the base blog repository, change to the directory we just clones and installing the npm packages:

```
git clone https://github.com/11ty/eleventy-base-blog.git
cd eleventy-base-blog
npm install
```

I use Python's built-in server when testing static sites without Gulp or Node acccess:

```
python3 -m http.server 2500 --bind 127.0.0.1
```

When we talk about Gulp and whether integrating it with Eleventy makes sense or whether we should create new Evelenty plugins we'll revisit the question of what to use to serve the content while on development.

With the software installed we can do exploration and analysis of the tool, how it works and wheter it meets my needs given my workflows and needs.

What Eleventy gives out of the box

The following elements come with Eleventy out of the box:

- Markdown content authoring and conversion to HTML
    - Extensible via Markdown parser plugins
- Code syntax highlighting using Prism.js
- Templating with Nunjucks; other engines available
    - Different template layouts
    - Customizable using the template engine's infrastructure

In the next section I will take a look at what I need and want to add, if possible, and whether Eleventy itself or a third-party tool will best accomplish the tasks.

Tasks to add

There are several things I'd like to explore using in an environment provided by Eleventy.

Some of these are more conceptual and require thinking if I should use Eleventy alone or in conjunction with a Gulp-based build system.

Looking at sample sites like v8.dev, alexcarpenter.me, Zell Liew, duncan.dev, zachleat.com, and others in the list of sites using Eleventy (some of which also share their source code) makes it easier to figure out how to accomplish a task.

Adding additional types of content

Because it's Markdown it shouldn't be hard to add new directories with content and then just pass them to the processor.

We can also create collections of specific posts sorted however we need them to be to accomplish our goals. The code to create a collection of posts sorted by date looks like this:

```
eleventyConfig.addCollection('posts', collection => {
  return collection
    .getFilteredByGlob('src/blog/*.md')
    .sort((a, b) =>'src/blog/*.md'e);
});
```

We can create collections for each type of content we want to build and we can group them in different ways. Look at the documentation for collections, and Working with Collections for more information of what they can do and how you can organize them.

Definition Lists in Markdown

Markdown-It provides a plugin (markdown-it-deflist) to create definition lists (dl, dt, and dd) elements

The code to configure the Markdown plugins looks like this inside the Eleventy build file (`.eleventy.js`) looks like this.

The Markdown section of the Eleventy build file that deals with Markdown, Markdown plugins and configuration looks like the block below.

```
const markdownIt = require('markdown-it');
const markdownItAnchor = require('markdown-it-anchor');
const markdownItAttrs = require('markdown-it-attrs');
const markdownItDefList = require('markdown-it-deflist');

const markdownItConfig = {
  html: true,
  breaks: true,
  linkify: true,
};

const markdownItAnchorConfig = {
  permalink: true,
  permalinkClass: 'bookmark',
  permalinkSymbol: '#',
};

const md = markdownIt(markdownItConfig)
  .use(markdownItDefList)
  .use(markdownItAttrs)
  .use(markdownItAnchor, markdownItAnchorConfig);
```

See the Pandoc definition lists for the syntax.

Service Worker

There is an Eleventy plugin that provides a way to create a precaching service worker. In order to use a more sophisticated service worker with custom routes we would have to fork the plugin and make it do what we want or submit a PR for the code we want to use instead.

I'll go with the first route. In looking at the code I discovered that it uses a very generic `generateSW` worker builder script. See builder.js in the Eleventy plugin repo to see how it's built.

If I want to use a custom service worker I'd have to use `injectManifest` and have the customized service worker ready for the manifest to be injected. This allows me to use advanced features available to service workers later, if I choose to. A solution may be possible with Gulp as the driver, as discussed in the next section.

Integrating Gulp functionality: Gulp tasks or Eleventy plugins?

Eleventy comes with a small set of plugins for basic blog-like functionality but, beyond that, the functionality of Eleventy is sparse. Gulp, on the other hand, has a rich ecosystem of pluggable tasks, like image compression, SASS and Babel transpilation, service worker generation and precaching injection, and other tasks.

Rather than build Eleventy-specific plugins I've chosen to implement the functionality as Gulp tasks using its ecosystem.

The tasks are listed below and described after the list

```
Tasks for ~/code/eleventy-blog/gulpfile.js
├── generate-sw
├── sass
├── processCSS
├── babel
├── imagemin
├── createResponsive
├── clean
├── serve
└── default
```

The descriptions are as follows:

- `generate-sw` will inject the files specified in a configuration file into a `sw.js` template
- `sass` transpiles SASS into CSS
- `processCSS` uses PostCSS and Autoprefixer to add prefixes where needed
- `babel` transpiles ES2015+ into Javascript supported by older browsers based on the browsers we want to support
- `imagemin` compresses images using different libraries depending on the format. It supports PNG, JPG, WebP and SVG
- `createResponsive` creates a set of responsive images to be used later
- `clean` deletes the destination directory
- `serve` starts a local developer server with the content of the destination directory
- `default` runs the following tasks in the order specified: `processCSS`, `imagemin`, and `generate-sw`

Netlify CMS: Yay or nay?

There is an interesting option for Eleventy-based sites. Netlify provides a CMS package that creates a CMS Admin for your static site.

It also allows for integration with third party tools like Cloudinary, Netlify Large Media or Uploadcare for image management and editing.

What I like is that it provides a graphical way to interact with the static site and manipulate images without having to work with content directly and eliminating some of the Gulp workflow we discussed in the last section.

On the other hand this may be the best solution for client work where we want to hide the complexity of a static site when people are adding content.

Conclusion

Eleventy presents an interesting solution to the static site generation issue. It gives you enough flexibility to do things how you want them without getting on your way.

I’m pushing ahead with further analysis and explorations of what you can do with the platform and the technologies we discussed here. One thing in particular that I want to research further is how deeply tied are the Netlify CMS and the Netlify platform.