Accessible document workflow with Eleventy and Prince

This is a demonstration of how to use Eleventy and Prince to create a website and a PDF document from the same content.

This project was created by Larry Hudson. You can find the source code on GitHub.

Contents

Why use this approach?

A webpage is the best way to share information online

If you're sharing information online, the best format for the end user is a webpage:

- the content reflows to match the user's device
- the content will load faster, as the user can read the content while images and assets are downloaded (whereas a PDF needs to be fully downloaded before a user can read any of it)
- you can break the content up into separate pages and connect them through hyperlinks, so the user does not feel overwhelmed
- you can take advantage of the power of CSS to offer features like a dark mode

This website aims to demonstrate these benefits:

- the content is responsive, so the user does not need to scroll in two directions to read the content, regardless of their device
- on mobile devices, there is a 'sticky' navigation menu that stays at the top of the screen, so people don't need to look for the navigation if they are on a long page.
- the website uses a dark colour scheme for users who have set their system preference to prefer it.

For longform content, a print option is useful

If your content is long, or designed to be read in a specific order, then a 'print PDF' option can be useful for users.

Some users may prefer to print out the document and read it, especially if they are not comfortable reading on a screen.

If your content has many different sections, then a well-formatted PDF that has everything the user needs in one place, can be a great experience for the user.

A big part of accessibility is making sure people can access the information in their preferred way. So offering different formats is good for accessibility!

Maintaining multiple versions of a document is hard

If you are publishing a document in multiple formats (eg. a print document designed in InDesign, a Microsoft Word version and HTML content to be inserted into a CMS), it can be difficult to keep each version up to date. If a content change needs to be made, you might need to change it in three places.

By generating multiple formats from the same source of truth, we can be sure that any content changes automatically come through to the generated output.

How this works

This project uses two tools:

- Eleventy to turn your content into a website
- Prince to turn one of the webpages into an accessible PDF with special CSS rules that affect the PDF output

Here is a summary of the workflow:

1. You edit your content

Eleventy makes it possible to pull in content from many different sources. So you can write your content in the way you prefer.

You can use:

- Markdown (.md) files inside the Eleventy project - this is the most direct way to get content into your Eleventy site
- a content management system, which Eleventy pulls data from when the site is built - eg. WordPress, Drupal, KeystoneJS. The content management system needs to allow you to access the data through an API (eg. a REST API or GraphQL)
- Microsoft Word documents as input with the plugin eleventy-plugin-docx

2. You create the HTML template, and the PDF template

Inside your Eleventy project, you create the HTML template that the content will be inserted into.

You create two separate templates:

- a template for the website, including the navigation between pages
- a specific template for the PDF content, which Prince will convert to a PDF.

In this project, the /pdf-content/ page is the one that gets converted to a PDF.

You can browse the source code of this project to see an example of this.

3. Eleventy builds the website, and Prince converts the 'PDF content' webpage to a PDF

When you run the 'build' command, Eleventy will build the HTML pages ready for publishing.

With the eleventy-plugin-prince-pdf plugin installed, Prince will automatically convert the specified webpage to a PDF after Eleventy builds the website.

If you deploy to a platform like Netlify, it will automatically rebuild the website each time you make changes to the Git repository. If you're using a CMS, you can also set up a 'webhook' that tells Netlify to rebuild the site, when you make content changes.

Customising the PDF output

As you can see in the PDF, Prince does a lot more than your browser's 'print to PDF' function.

It makes it possible to do things like:

- add the current page number in the bottom corner
- add page numbers to hyperlinks, like in the table of contents
- add the document title in the header, and the current section title in the footer.

Prince does this with custom CSS rules.

This code, in `src/css/print.css`, controls the header and footer content:

```
/* Set variables to use in the header and footer */
#doctitle {
  string-set: doctitle content();
}

h2.page-break-before {
  string-set: sectionName content();
}
/* Add page number to bottom of pager */
@page {
  /* Add document title to top left */
  @top-left {
    content: string(doctitle);
  }

  /* Add section name to bottom left */
  @bottom-left {
    content: string(sectionName);
  }
```

```
  /* Add page number to bottom right */
  @bottom-right {
    content: "Page " counter(page) " of " counter(pages);
  }
}
```

You can find out more in the Paged Media section of Prince's User Guide.

Edit me!

This project is open source, available on GitHub.

Feel free to download the code, have a play and send some feedback. You can reach me on Twitter.