title: Fruits of the toolbox fallacy
plainTitle: Fruits of the toolbox fallacy
subtitle: How I moved this site to use Google Docs as a CMS
date: February 2023
[+content]

:newthought[I happened to see stuff around the web] about the so-called [“toolbox fallacy.”](https://henningjust.wordpress.com/2019/10/11/the-toolbox-fallacy/) The idea is that people avoid doing something they need to do by convincing themselves that they don’t have the right tools to do it. I don’t have the right equipment, so I can’t exercise. I don’t like the pen I have, so I can’t write. And so on. Even though I’m not sure whether real logicians or philosophers talk about this fallacy, it seems to be a semi-popular topic on self-help websites and YouTube channels.

But falling for the toolbox fallacy *can* yield interesting results. In particular, you might end up with a shiny set of new tools, as I found out for myself recently. I hadn’t added any writing to this site for a while, and clearly the problem was that the site wasn’t properly set up for me to write well and easily. So of course my next step was to rework the site.

### The Goal
The site was previously quite simple. Everything was pretty close to hand-written HTML, hosted on Github Pages.:sidenote[You can find the repo backing this site [here](https://github.com/gautamh/gautamh.github.io).]{#github-repo} I used a bit of [mustache](https://mustache.github.io/) templating to eliminate repetitiveness in a few areas and used SCSS for my styling, but other than that I was pretty directly writing the stuff that showed up on the page.

I was particularly annoyed by the fact that every time I wanted to write a longer piece to put in the “Thoughts” section:sidenote[Maybe once the number of posts here rises beyond single-digits I will actually call this a blog rather than just a random collection of thoughts. You shouldn’t hold your breath.]{#blog} of the site, I had to package whatever I wrote into HTML. Piecing together HTML in an IDE is not particularly great for fluid writing and editing. This led me to start off in Google Docs until I had what I thought was a final draft, convert the doc to HTML, and then tediously edit the HTML when I wanted to make a change after the conversion.

Every time I had to do this, I kept thinking it would be just great if I could somehow write in Google Docs and then magically have the contents of the doc show up, nicely formatted, on my

website. Essentially, I wanted to use Google docs as my CMS. This is of course not a new concept. There are plenty of examples out there of Google Docs being jerry-rigged into a website CMS.:sidenote[You can find
[example](https://css-tricks.com/using-google-drive-as-a-cms/) after
[example](https://twitter.com/davidwells/status/1356672763981778944) after
[example](https://github.com/sawyerclick/CMSvelte) of people doing this in various forms. I could probably have started from one of those examples, but I did have a few custom requirements, and I wanted to try making my own thing.]{#docs-example} So how hard could it be?

### Svelte
Step one to doing this was moving the site to some more standardized framework, rather than just having raw HTML assembled with a templating tool. I didn't really have a great idea or strong opinions on what framework to use, and there were a bunch of reasonable options out there. My one requirement was that I strongly preferred not to be in the business of running my own server, either locally or in the cloud,:sidenote[I have heard too many [horror stories about people waking up to massive surprise credit card bills](https://chrisshort.net/the-aws-bill-heard-around-the-world/) from their cloud provider.]{#cloud} so the framework had to be something that would compile my website into static content that could be deployed on Github Pages. I ended up choosing Svelte, not for any particular reason, other than that several of the other alternatives I looked at were React-based and I felt like not using React.:sidenote[Really it was just a gut feeling, I did not have clearly articulable reasons for making this decision (maybe I should have!)]{#reasons}

Setting up Svelte and moving my existing site to it was quite straightforward. Given that I was using a templating setup before, I already had some rough outlines of what things could be components and then how to use those components with the data for the site. Svelte allows projects to be compiled with a static compiler, and Github also allows Github Pages to be deployed based on Github Actions, so Github can automagically compile the site and deploy it to Github pages every time a commit is pushed to the backing repo:![Github Action to deploy Github pages run on repo commits](/img/github_pages_actions.png)

### Pulling from Google Docs
Then came the interesting part: using Google Docs as a CMS. Actually, wiring my Svelte project up to Google Docs so that it could pull content when the site was compiled worked pretty easily. There are various Node packages out there that nicely wrap the process of calling Google's Node packages to get the content for a given Google Doc. I just had to make the docs I wanted

to pull from for the site publicly viewable and provide the doc ID to the library I was using to pull the docs.:sidenote[The doc backing this page can be found [here](https://docs.google.com/document/d/1y9aQZRNvJVzKGLuCKFFb3c_HyL6Tmdj2pT38dPXiipA/edit?usp=sharing).]{#doc-link}

I did however pick a package that did a little more than just returning the contents of the doc. It occurred to me that I might want more than just plain text from my Google Docs content. It might be nice to have structured content for things like titles or metadata. Someday I might even want to add an interactive that consumes structured data. To allow all this, I decided to parse the contents of Google Docs I used as [ArchieML](http://archieml.org/), which is a structured text format that allows structure to be added simply while still being easy to write. I used the [doc-to-archieml Node package](https://github.com/rdmurphy/doc-to-archieml) to easily get the contents of docs and parse them into JSON from ArchieML. The JSON contents could then be processed for display on the page.

### Formatting

I also wanted to easily specify formatting in my Google Docs. The obvious solution here is of course to write Markdown, which is designed to allow text formatting from plain text, within the structured pieces of my ArchieML docs. I saw that the MDsveX package allowed Markdown to be mixed into Svelte components, which seemed great. However, I eventually realized that MDsveX Svelte components [don't seem to be able to load Markdown dynamically](https://github.com/sveltejs/kit/issues/2014), rather the Markdown has to be specified directly in the component definition. This obviously doesn't work if I want to pull my Markdown from a Google doc.

Eventually I decided to use the MDsveX's Markdown compiler as a library and then provide the result as raw HTML to my (standard) Svelte components. This was a little janky,:sidenote[...then again, so is this whole thing...]{#janky} but worked reasonably well to a first approximation. However there were issues with some specific elements that either didn't render as I wanted or which needed to be expressed in non-standard ways with the CSS framework I was using.:sidenote[which is my own flavor of [Tufte CSS](https://edwardtufte.github.io/tufte-css/)]{#tufte} Footnotes in particular required a more custom treatment than just the standard Markdown compiler.

To handle this issue, I added Remark and Rehype plugins to the compiler. These let me define my own custom markup handling for elements that required custom handling beyond the standard Markdown compiler. The

[remark-directive](https://github.com/remarkjs/remark-directive) plugin even allowed me to use my own custom Markdown syntax for elements that just aren't available in standard Markdown. For example, I was able to add syntax for the "newthought" element available in Tufte CSS:

```js
function tufteRemarkDirective() {
    return (tree, file) => {
      visit(tree, (node) => {
        if (
          node.type === 'textDirective' ||
          node.type === 'leafDirective' ||
          node.type === 'containerDirective'
        ) {
          const data = node.data || (node.data = {})
          const attributes = node.attributes || {}
          const id = attributes.id

          if (node.name === 'newthought') {
            data.hName = 'span'
            data.hProperties = {
              class: 'newthought',
            }
          }
          else return;
        }
      })
    }
  }
```

### Deployment

At this point, I pretty much had a working Google Docs-as-CMS setup running locally that did the things I wanted it to do. But I didn't want this to run locally, I wanted to push the Svelte code to Github and then have Github compile and deploy the site to Github Pages. Github makes this fairly easy in the basic case, but things get a little trickier when you want the site compilation process to fetch data from Google Docs. I had to include Google credentials in the repository without making them available to everyone. Luckily Github has a Secrets feature which allows

this, even if it is a little more complicated than it should be. I then added an [action](https://github.com/jsdaniell/create-json) to my deploy configuration to copy the secrets into an env file during deployment.

Of course, the deployment process didn't work the first time I tried it, and debugging from Github deployment logs can be cryptic and tedious. Again, however, the ecosystem of Github Actions comes to the rescue. As it turns out there is a [fantastic Github Action](https://github.com/marketplace/actions/debuggging-with-tmate) that will let you run a tmate session in the environment that is running your deployment and connect to the session remotely. This allows full inspection of the deployment outputs, which greatly simplifies the debugging process.:sidenote[The attached terminal also lets you do things like see secrets used in the deployment process, which are automatically scrubbed from the debug logs Github provides by default.]{#secrets}

I now finally had a fully deployed site, with a complete setup that looked something like this:![Site architecture](/img/site_architecture.png)

### Remaining issues

And so that was it, everything was working! Well, almost…there are still a few minor issues hanging around.:sidenote[If you have tips on how to fix any of these issues, please do drop me a line at gautamhathi[at]gmail[dot]com.]{#email} The first one came up when I did the initial move of the site to Svelte. I use [Fancybox](https://fancyapps.com/docs/ui/fancybox/) gallery to showcase photos on my site, but something seems to have broken Fancybox during the move to Svelte, and I can't quite figure out what. When opening images that were supposed to be handled by a Fancybox gallery I got errors saying "Attempted to preload a URL that does not belong to this app" and the browser just navigated to the image file rather than presenting it in a nice gallery. I saw some suggestions that I could add a `data-sveltekit-preload-data="off"` attribute to the image link elements, but that didn't seem to fix the issue.

The other still-lingering weird issue is with formatted code blocks. For some reason, spaces at the start of lines in code blocks don't seem to show up. The way I've gotten around this is by adding zero-width characters at the start of lines in code blocks, so they don't technically start with spaces. As a result the issue doesn't end up being something that is visible on pages, but it's still annoying to deal with.

There is, as always, more work to do. Of course, the code needs tests. Given that it's just me working on this and the codebase is pretty small (at least for now), I don't feel too bad about putting those off, but code without tests generally leaves me feeling nervous. Additionally, it would probably be good to move the js portions of the code to Typescript, which is something else I didn't feel like dealing with right now.

Farther in the future, I have thought about the fact that the current setup doesn't provide great content versioning. Changes to the docs show up on the site when the Github Action deploying to Github Pages is run and are not tied to repo commits. This is a rather tricky problem to solve, though, as while it might be possible to have the deploy Action commit doc changes to the repo, the repo commits with doc updates could easily get out of sync from the doc state. But I currently have no need for precise versioning of my content tied to versioning of the site, so this isn't something I'm too concerned about for now.

And of course, after doing all this, it would be good to write some more for the Thoughts section of this website. The toolbox is complete, right? At the moment however, I can't really think of anything great to write, so I wrote this instead.

:ignore
With the help of a generic templating tool. I wasn't really sure what ![Drag Racing](https://cdn.pixabay.com/photo/2016/11/19/14/00/code-1839406_1280.jpg)

Mdsvex templates don't seem to be able to take in markdown from parameters (so have to compile instead)
Need to use remark + rehype plugins to handle things that aren't standard markdown (specifically newthought elements and sidenotes). Rehype is needed for sidenotes because we need a single md element into multiple html elements
Compiler doesn't seem to work with Svelte components
Weird issue with code blocks when compiling: https://github.com/pngwn/MDsveX/issues/392
Spaces at the start of lines in fenced code blocks don't work (solution is use invisible chars at the start of the line, though can't just have a remark plugin add them at the start of all lines in the block)
Deployment issues: how to get secrets into github actions (repository secrets vs environment secrets, env files: https://stackoverflow.com/questions/60176044/how-do-i-use-an-env-file-with-github-actions, json creation tools: https://github.com/jsdaniell/create-json), how to debug (do you just add

logging with a bunch of commits? Actually you can add tmate sessions, including on failure: https://github.com/marketplace/actions/debugging-with-tmate)