radically simple documentation

Documentation is one of the biggest problems facing any software project. Ideally, documentation should be easy for developers to create and for users to read.

This post will explore the following topics:

- The concept of docs as code
- Possible tooling for docs as code
- Creating an experimental docs as code toolchain

Docs as code

The idea behind Docs as code is to treat your documentation the same way you treat your project's code; more specifically:

> Documentation as Code (Docs as Code) refers to a philosophy that you should be writing documentation with the same tools as code:
>
> - Issue Trackers
> - Version Control (Git)
> - Plain Text Markup (Markdown, reStructuredText, Asciidoc)
> - Code Reviews
> - Automated Tests
>
> This means following the same workflows as development teams, and being integrated in the product team. It enables a culture where writers and developers both feel ownership of documentation, and work together to make it as good as possible.

We will follow this methodology as we dive deepeer into g3docs and a similar tool for our projects.

Riona MacNamara is a technical writer at Google. In her presentation at SRECon 2016 has the following description she states that:

> Solving this problem [poor documentation] is tough. It's not enough to build tooling; the culture needs to change. Google internal engineering is attacking the challenge three ways: Building a documentation platform; integrating that platform into the engineering toolchain; and building a culture where documentation — like testing — is accepted as a natural, required part of the development process.

These are some of the goals to derive from docs as code:

- Docs are easy to write
- Docs are automatically built on every commit but can also be built on demand
- There is a UI built for compiled code

Now that we have a basic understanding of docs as code, we can move on to the next topic and see what other people have done to address the needs of docs as code.

Possible tooling for docs as code

Docusaurus

Docusaurus is a static site generator. According to its documentation:

> Docusaurus is a **static-site** generator. It builds a **single-page application** with fast client-side navigation, leveraging the full power of **React** to make your site interactive. It provides out-of-the-box **documentation features** but can be used to create **any kind** of site (personal website, product, blog, marketing landing pages, etc).

It is React-based, not surprising since it's also a Facebook product.

It takes Markdown files as input and provides the UI and the tooling to build a single page application from them

docToolchain

DocToolchain uses the same idea as Docusaurus, using the jBake for the site generation.

Some of the differences between the two products. With DocToolchain:

- You can create documentation from multiple Git repositories
- You can publish your documentation to Atlassian Confluence

VuePress

VuePress

> VuePress is composed of two parts: a minimalistic static site generator with a Vue-powered theming system and Plugin API, and a default theme optimized for writing technical documentation. It was created to support the documentation needs of Vue's own sub projects.

The theme is written with Vue.js so, if you're familiar with the platform, it should be easier to customize the layout of your documentation.

It uses Markdown-it for Markdown rendering and Markdown-it plugins for features, so we can customize both the render behavior and capabilities.

Repurposing a static site generator

as a g3doc-like tool

My static-gen-njk tool does the first part of any documentation system: It converts content into HTML.

We just need to augment it to make it into a full documentation system

The first thing to consider is how to build a site around our content, whether it's a single page app or a regular website.

An auxiliary requirement to that is the need to provide a list of the files to act as navigation

Adding tools

There are two additional Markdown-it plugins that I want to research for inclusion in the converter.

Mermaid

Mermaid allows you to create diagrams on Markdown that will be converted to HTML later in the process.

Particularly when writing documentation about code, diagrams like UML models should always be available for people writing documentation.

There are multiple Markdown-it plugins to handle Mermaid diagrams. I chode markdown-it-mermaid

This is an example of Mermaid included as Markdown-it plugin:

```
~~~mermaid <optional title>
  graph TD
    A[Christmas] -->|Get money| B(Go shopping)
    B --> C{Let me think}
    C -->|One| D[Laptop]
    C -->|Two| E[iPhone]
    C -->|Three| F[Car]
~~~
```

Markmap

Markmap gives us the option to use Mindmaps as part of our documentation projects.

I chose to use markdown-it-markmap for the implementation of mindmaps for this project.

```
\```markmap
# root
## child1
  - child3
## child2
  - child3
\```
```

Output Formats

The other side of the equation is to think about the formats that we want to generate from our Markdown content.

The default is HTML, the basic content unit that we need to build a website from.

Another format that would be useful is PDF. With it we can print our documentation in a more readable format.

How we generate the PDF is the main research question for this section of the project. We can choose to run puppeteer to generate the PDF, or we can use tools like Pandoc

Automating the documentation process

As configured the process runs manually. It would be nice if we could automate the process so that every time there is a pull request or a commit to the documentation repository, we rebuild the full documentation site.

If you host your documentation repository on GitHub, you can use the GitHub Actions to automate compiling and, if needed, publishing the documentation site.