What makes good documentation? - The importance of the README - What I look for in a library

Written by Seth Corker on Benevolent Bytes

As a web developer, picking packages is second nature. There's a library for everything and usually lot's of alternatives but, how do you pick which one is right for your project? There are lot's of factors that go into making a decision. You might look at how many people use it, how recent the last release was or perhaps you'll take a peek at the source code. The catalyst for choosing a library or framework for me is documentation. Specifically, the first impression, the readme.

The humble readme

A readme is the sales page for a developer, it sells the promise of what to expect. It's the first impression that should make an impact. A readme has background behind why the library exists, the prerequisites and how to perform common tasks. The humble readme is what separates a library that makes it into your project and the countless that don't.

Developers are busy. We don't have the time to work around a lack of documentation in many cases. If it's unclear how to install the library in question or there's no help on how to get started quickly, we'll likely abandon it in search of something else. Even an older, slower library that is more well documented will often win out over something faster, better and lacking documentation. Sites like NPM and GitHub prominently display a project's readme and a good one helps developers make informed first impressions. Links to relevant community resources, live demos, links to more in-depth documentation. This what e've come to expect. If developers have to look at the source code as the first impression, they might just choose to reimplement it themselves and forgo the exercise entirely. These aspects are all vital for figuring out if you should make the initial investment and, it *is* an investment. Once a library is decided upon, the work of integration with your project begins. This involves going beyond the basic quick start and installation. The likelihood of this happening with a poor documentation is unlikely. This endeavour kicks off the deep-dive into API reference, implementation guides and, tutorials so we can become proficient and help onboard the rest of our team. So what does good documentation look like?

Documentation

Beyond the readme, documentation is key for developers but it's difficult to get right. You can have the most amazing library that's amazing to use but render it invisible to the world without documentation to support it. There are a few key things I look for in documentation that make my life as a developer easier and the experience of using libraries smoother.

Cookbooks and Recipes

When you go in search of a library, you're often looking to solve some very specific problems. Cookbooks or recipes are guides on how to achieve very specific things with the target library. They are invaluable resources because they go further than just how individual functions work. They help you see the bigger picture on how each component part of the library fits together to solve a specific problem. I like cookbooks because they let me focus on just the information I need while giving me enough context to come up with more advanced solutions later on.

Cookbooks I think are great RedwoodJS

RedwoodJS is a full-stack framework for building web applications. The tutorial covers a happy path to build a specific project but there are other things you might need when building your own project. This is where cookbooks come in, they cover the other things like how to handle file uploads or setting up role-based access control. They're topics that won't interest everyone but when you need them, you'll be thankful there's a helpful cookbook available.

Examples

Examples go hand-in-hand with good context and explanations. You can't have one without the other. I find the best use of code snippets is littered throughout documentation. It gives developers the freedom to hack around, copy snippets and see how they work. You might choose to scroll down to the example, dig it apart then read the documentation around it. These two work in tandem and some of the foundational components of good developer documentation.

Examples that I think are great Chakra

Chakra is a component library for React that has great documentation because it provides context, code snippets, examples and demos all within the same page. You can pick any component from the library and see how to import it, where it can be used and then editable demos. This allows you to see if it will work for your use case without leaving the browser which means you tinker without too much upfront time investment.

Tutorials

Building something is often the best way to learn how a library is used and where it works best, tutorials are a great way to achieve that. They're not for everyone but they certainly lend a helping hand and show developers a happy path for using a library. They centre around a project which is where they differ from cookbooks and examples. A tutorial explains the steps to get started and build a specific project, touching on key points along the way. They're a slice of how the library could be used from start to finish and build a good foundation to allow developers to dig deeper when they need to.

Tutorials I think are great Svelte

Svelte is a tool for building user interfaces similar to React or Vue. It has one of the best tutorials I've ever seen. Why is it so good? It's tailored for developers. The authors of the tutorial know the audience well and it shows. The view is split into three sections, the context and tasks including examples are on the left and the right side contains a code editor and output panel. This interactive tutorial allows anyone with a web browser to just start learning without having to install anything. This is what makes it incredibly powerful, it's accessible and doesn't ask you to install anything.

RedwoodJS

RedwoodJS is a full-stack framework for building web applications. The tutorial takes the "build a blog" approach that made Ruby on Rails popular. It has video walkthroughs broken up into logical sections and is accompanied by textual documentation too. This flexibility gives developers the choice to learn how they learn best, some people learn better watching videos but the more traditional documentation is infinitely more searchable. They can be used in combination to get up to speed quickly.

API reference

Depending on the language, API reference as part of online documentation can be vital or supplementary. A language like JavaScript benefits from having a detailed API reference so developers can figure out how the library works, what functions expect and what output should look like because an IDE might not be able to figure this out. In the same ecosystem, a TypeScript library might not need the same attention because users will often receive that information in the editor which is arguably more useful. It makes the functions in your library findable, as developers use it they can explore themselves discover the library more naturally. Whatever the context is, API reference is a foundational component of good documentation but it shouldn't be the only form. It's easy to generate this type of documentation from source code but as a library author, make sure this is supplemented with some of the other aspects.

API reference I think are great Lodash

Lodash is a utility library that adds a lot of useful things you wish you had by default in JavaScript. The documentation leans more towards an API reference with extra context but it';s good nonetheless. The signature of the functions are shown alongside the arguments, their type and what the function returns. There's also an example afterward to show how it might be used. What makes it really great is the sidebar that contains all the possible functions broken down into sections, a search bar and links within explanations to similar functions and why you might use one over the other.

Playwright

Playwright allows you to control browsers for creating and orchestrating end-to-end tests. The reference documentation is good because it provides an explanation of what a function is used for and lists out the arguments, their types, defaults and some hints on how best to use them. There are also useful callouts for common gotchas to avoid you going down a rabbit hole wondering why something isn't working as you thought.

Tying it all together

There is a lot that goes into good documentation for libraries that can make or break a good library. It all starts with a good readme. To get developers in the door a readme should sell the library and make it as easy

as possible to evaluate at a glance. To get developers to stay, documentation is key. Great documentation consists of a few core pillars, cookbooks, examples, tutorials and, api reference. Cookbooks are micro-tutorials for very specific slices of functionality. Examples should give an idea of what's possible in easy to understand, digestible pieces. A tutorial centres around a project with the aim of familiarising developers with the library. Finally, an api reference is something for developers to fallback on and discover the finer details. Together, these components make documentation that tells developers a library is worth their time and investment.