DrDocx ships its first alpha release!

Faiz Surani

Jul. 05, 2020

Author's Note: This post was originally published on the DrDocx blog[1]*. It is reproduced here in its entirety.*

Yesterday, DrDocx officially shipped to outside users for the first time with our 0.1.3-alpha[2] release. We have a lot of work yet to do, but we're proud to have reached this milestone and put our work in the hands of our partners. It's been a long and winding road to this point, so I thought I'd write about how we got here, and the lessons we learned.

[1] *https://drdocx.com/blog/0.1.3-alpha-released*

[2] *https://github.com/DrDocx/DrDocx-Desktop/releases/tag/0.1.3-alpha*

1. **There's no such thing as cross-platform.** Okay, that might be a bit of an exaggeration, but not by much. From the very beginning, our goal was to build a fully cross-platform app, because we wanted to make DrDocx as widely available as possible but didn't have the resources or the motivation to maintain two separate apps. (We couldn't build a plain old web app because of medical privacy reasons.) That's alright though, we're living in the era of cross-platform! .NET Core, React Native, Xamarin, and so on are all here to help. Right? Right?!

   Not exactly. During SB Hacks, we decided on building an ASP.NET Core app, because everyone on the team had written either C# or Java (which is very syntactically similar), and, being built by Microsoft, it had APIs for interacting with Word documents. In short, it would allow us to build something quickly and figure out the exact details of desktop deployment later.

   The situation may well be better for mobile app development, but we ultimately discovered (over a not-negligible span of time) that there is really only two truly production-ready cross-platform desktop frameworks: Electron[3] and Qt[4].

   Electron, for the unfamiliar, is essentially a Chromium instance running a web app taped to some native desktop APIs. For some true believers in native code, the very concept inspires horror and symbolizes the decadence of a civilization in decline. While we weren't a fan of its bloat and resource consumption, it offered us inexperienced and heavily time-constrained developers something irresistible: access to a massive ecosystem of open-source UI libraries.

   [3] *https://www.electronjs.org/*

   [4] *https://www.qt.io/*

2. **OpenXML is an abomination.** OpenXML is the standard format of the flagship Microsoft Office products (Excel, PowerPoint, Word,

etc.). It is also the source of a great deal of misery and despair. One central feature of DrDocx is the ability to fill text into a predefined Word document to generate reports from templates; essentially, find and replace. One would think this would be as trivial a problem as one could conceive. We certainly did.

The reality is a little more complicated. Without delving too deep into the technical details of the OpenXML file format, there's a lot of quirks to its internal structure, resulting in a pretty high baseline of complexity of interaction. Our first instinct was to grab an off-the-shelf library for basic Word document operations. Vexingly, the cult of open source didn't seem to have reached the Word document-adjacent tech industry, and the only things we could find were closed source libraries with four-figure price tags. Being both morally offended and (perhaps more operatively) broke, we decided to stick with the low-level OpenXML SDK and implement all the functionality ourselves.

This, as alluded to, turned out to be pretty tricky. The most frustrating thing we found was the sheer amount of *wrong* solutions to our find-and-replace problem on everywhere from StackOverflow to, somewhat outrageously, Microsoft's official OpenXML documentation. Given this, we owe a huge debt of gratitude to Eric White, whose blog[5] was an invaluable source of information and implementation suggestions as we struggled through the trials and tribulations of OpenXML.

[5] *http://www.ericwhite.com/blog/*

In the coming months, we plan to refactor our various Word interaction APIs to be less tightly coupled to the rest of our codebase and release it as an open-source .NET Core SDK for all to use completely free of charge and contribute to. If even one other person saves all the time we spent, it'll have been a worthy contribution to the developer community.

3. **Working titles stick.** DrDocx is a name we've certainly come to love, but it didn't start that way. DrDocx was born of a 5-minute brainstorming session right before the hackathon started, because we needed *something* to name our source code repositories. We were building a tool for doctors that generated Word documents, so Doctor + .docx = DrDocx. As it turns out, though, once you have a name plastered over anything and everything associated with the project, you both lose both the ability and desire to easily change it.

4. **You probably won't make the right design decision the first time.** One of the reasons that agile development has caught on as much as it has is the increasing recognition that users use your product in ways that you can't really hope to foresee. It's all well and good to

hear someone say this or read about it, but we found it extraordinarily valuable to make the waterfall-type mistakes on our own.

We spent countless hours agonizing over different design decisions, whether they be over database structure or user experience. It is important, to be clear, to give these things forethought and get them reasonably right upfront, but we discovered you can only be so imaginative putting yourself in the shoes of a user. It's better to give something imperfect to your users and iterate from there than it is to give them a perfect product for a use case that doesn't actually exist.

We hope what we've learned and done can be of use to others. If you have any questions, comments, or concerns, we'd love to hear from you at dev@drdocx.com[6].

[6] *mailto:dev@drdocx.com*