Maintenance Plan: Spotify Recommendation Engine

Grant Jurgensen, Stephen Longofono, and Stephen Wiss

This document serves to identify the ongoing process of maintaining our song recommendation software, strategies for doing so, and the associated costs. The maintenance of our software will likely boil down to updating and adding functionality based on changes to spotify's API. This involves a contingency plan in the case that Spotify begins charging for access to it's developer accounts, and of course, adding refinement to our algorithm based off of functional differences in the API source code.

1. General Maintenance

Speaking in terms of General ongoing Maintenance, Our update schedule would be synchronous to the Spotify web-api team's. That is to say That there wouldn't be a schedule. Judging by the graphed history of commits to the spotify web-api's github, It appears that updating and changes to the source code really only occur when there is something wrong with it. This means that if our team is to be effective at maintaining quality of life and smooth operation for our userbase, then we will have to be on top of and ready to integrate the changes to spotify's web-api to our source code. One thing worth noting however, is that our application utilizes Spotipy - an open-source python module incorporating the spotify API. This could prove to be either beneficial or costly to us. On one hand, we have a buffer between our source code and incompatible changes implemented into Spotify's API. On the other, we are dependent on the maintenance and upkeep of an open-source library that appears (based once again on it's github commits) to be dead.

To combat these inevitable pitfalls, the most viable strategy would be to employ someone to act as a support representative/maintaining developer. The operative of this position would be two-fold:

- Address known issues. (changes to the API)
- Provide support and feedback to user submitted bugs

This is of course, a reactionary tactic. But the provided circumstances do not leave room for a lot of proactivity. However, if the project moves forward according to our Deployment Plan, more work will be required to maintain the additional features added to our application.

It should also be mentioned that our application will need to be updated frequently to remain compatible with current working versions of popular operating systems. Some current features will add further complication to this and as such, We will have to consider switching out some aspects of our application for ones that are more scalable.

2. Potential Maintenance

In this section we'll cover a few hypotheticals that we touched on briefly in our introduction and have been mentioned in our deployment plan.

In it's current state, our application is the opposite of user-friendly. This is fine at present, because our chief demographic are technically-minded users that wish to evolve and fit our source code and algorithm into more specific custom applications. In our deployment plan we mentioned that part of releasing our application will be to allow a user access to our algorithm in an effort to stream-line cross-implementation. This is great and fits perfectly with the open-source philosophy. However, in order to fund the upkeep and further development of our application it will be necessary to generate some revenue. This is discussed at length in our deployment plan. Maintenance for this portion of our release is relatively minor, We would need to have an active repository on github with perhaps a stickied thread on a technical/programming forum or board (ie. Reddit, stackexchange, etc...) with general tips and an FAQ.

The other phase of our release, which is the fully functioning song suggester application catering to less technically-conditioned users, will incur a significant amount of maintenance. First of all, The application in it's current state is a bit hard to use and involves running a shell-script to generate the users profile. As mentioned in our deployment plan we will have to port the application to IOS and Android. Doing so will alleviate the learning curve necessary to running our application. It will also incur more necessary maintenance. Which will be expanded upon in the next section.

Our maintenance Plan would also be incomplete without discussing in detail what would happen should Spotify decide to start charging for access to it's web API. As is the case with all hypothetical situations, we can't be certain exactly what this would entail, but we can piece together some sort of contingency plan in the event that it happens. According to some

research done, API pricing generally comes down to a tiered pricing list relative to depth of access or computational time. It's not for certain where our application would fall within this schema, but assuming the worst, this would add a large additional cost to maintaining our application. This would also incentivize us to revisit some portions of our application and make them more efficient in their use of API calls.

3. Maintenance Involving Services and Hosting

Adapting and integrating our application to a fully-scaled and marketable product will introduce a lot of work in the deployment stage as mentioned before, but will generate numerous areas for maintenance as well. Our application as is, is written in python using the Spotipy module. Porting our application over to android would require rewriting the bulk of our code to Java, This would be feasible, taking into consideration the object orientedness of our algorithm, as well as the extent of libraries available on the java platform. In the case of making it available to apple users, the process becomes more involved. The portability of python to IOS is less accessible than for Android, and Apple tends to be pretty stringent in their approval process for applications that require network access to function. Also, We would need to wrap our application in Csharp or Swift, and neither of these have included libraries for accessing the python web-API. The initial process of translating our application would require some work (a lot of work in the case of IOS), and maintaining it in it's translated state would require the same breed of maintenance already covered in the General Maintenance section.

In addition to ensuring compatibility with mobile operating systems, we would need to make sure that our application worked on different desktop operating systems. Fortunately python runs natively on windows, mac and linux. We would just need to employ someone to make sure that our application remains up to date with new releases and updates to these operating systems.

One way to potentially circumvent porting the code over to cross-platforms would be to host it remotely or just turn it into a web-based application. This would require switching over to using something besides sqLite for our Databases. (We could use AWS's remote file system). Doing this would introduce more fee's for hosting and another engineer to be paid potentially for supporting this side of our application. This would be a lot of work and would be rather expensive but it could potentially save us some

hassle down the road and therefore should be discussed in the maintenance plan.

4. Projected Costs

I mentioned in the previous sections about certain maintaining tactics that will be employed by our team based on a number of different hypothetical situations. In this section I'll list out and compute a rough estimate of the total monthly overhead for maintaining our application.

The backend costs will primarily pertain to salaries for our engineers and QA, as well as hosting fees on whichever platform we choose. To be exhaustive I'll include a few to choose from.

Assuming we hire someone new to manage the QA and the support side of our application (as opposed to doing it ourselves). We will need to pay that person.Because of the lack of technical acuity required, the job itself is relatively non-intensive and could be relegated to a new-grad or entry-level. According to some research, I determined that the average salary in Kansas City for a software engineer is around 59000. Given that the work would be part time and more akin to a consulting position. I've prorated the salary to align with a part-time position. I estimate that the position will require roughly 10 hours a month (conservatively) so therefore total cost of employment for our QA department will amount to 280 per month.

In addition to managing salaries of hired help, This section will also address running api and service costs. These were determined to be 1000 per month at their worst in our deployment plan.

5. Conclusion

In closing, The maintenance required for this application will extensive and unpredictable as it is contingent on a number of factors. This exercise has made it plain to see why a start-up always looks great in theory but tends to be a daunting task when it comes down to it. Between the exorbitant fees charged by App retailers in addition to how expensive it is for remote hosting and the fact that our application will barely draw enough revenue to fund our support staff, let alone turn a profit for the executives (Us). All of this in itself would be insurmountable for a small tech business, but added to the fact that we would need to spend time and resources (and money) to continue growing our application and take care of our userbase means that it in itself is not a feasible revenue generator and would last better as an open-source free-to-use resource for other developers.

References

1Start for Free. Upgrade When Ready." Pricing - People Data API. N.p., n.d. Web. 14 Dec. 2016.
2API Pricing Tiered Flexible." API Pricing Tiered Flexible — Mashery. N.p., n.d. Web. 14 Dec. 2016.
3Salary Data Career Research Center (United States)." Average Salaries - Job Descriptions - Annual Job Salaries — PayScale. N.p., n.d. Web. 14 Dec. 2016.