GitHub issue bot proposal - “Spark”

Electron currently has an issue tracker that helps to triage and solve issues that arise during development in the project. The issue tracker needs to be automated, to organize the issues pool, assist in issue resolution turnaround time, and streamline the issue to solution pipeline. This proposal details the development of such a bot, that would perform all these functions and serve as a standard for GitHub issue tracking automation. Spark would automatically tag issues, automate issue responses in certain conditions, and close specific issues that are no longer applicable.

Gameplan

I plan on using a workflow that involves consistent testing during the development process. Cation gives us a good starting point for identifying key data points related to issue handling. Much like cation, spark will monitor issues by categorizing them and labeling them accordingly. Issues will be categorized by issue type (documentation, enhancement, etc.) and also by the amount of time it’s been open. Spark bot will also give a certain response based on the issue reported, and category. Lastly, Spark will close issues that are already solved for by taking a page out of trop’s book. Backporting issue fixes to previous issue reports based on release version and related category.

Desired mentor

A good mentor for my project would be a great leader. I know that coding is a never-ending journey, so someone who is persistent and passionate about this work would be a great asset. I would also like someone who is open-minded. I believe that beginnings hide themselves in ends, so someone who is open-minded will be able to point out possibilities that I may look over. Above all, I want someone who is patient with me and the learning process of improving as a developer.

How this benefits The Electron project

As stated previously, spark would provide several benefits to the electron project. Spark would organize and sort issues by type, which greatly assists in the process of finding a solution. Accurate problem diagnosis makes up at least 30% of the software development process in most standard timelines, so this bot would decrease that time significantly.

More specifically, having myself as a contributor to electron would be beneficial as well. As a community college student, having me as a contributor would be a further testament to the diversity already welcomed in the Electron project. I also personally like to help the project advance so that more people can easily build cross-platform desktop apps, with only HTML, CSS, and JS.

Timeline

* Week 1-2: Research and brainstorming – The first week will be spent on exploring in depth the issue tracker desired responsibilities and examining the probot framework architecture. This will be done by reading through the probot documentation as well as the docs for Cation and trop. Other research including Automation in GitHub applications, error code types, etc.

Milestone – layout completed (including probot settings), clear documentation of layout on paper or word doc.

* Week 3-4: Organization – The first week gives us enough time to have a rough starting point to the bot, and here we want to plot out the key abilities that make up our bot. We have 3 main functionalities that we want to include, and I want to conceptualize each of them individually, then combine them all. This includes selecting appropriate environment variables, designing how the bot will organize issues (crash type, time open, and status). The bot will use node.js and will be created with the probot framework.

Milestone – Begin coding (create node application). Complete environment setup(including variables and npm script(s)).

Optional- read documentation on types of issue reporting systems.

* Week 5-7: Code block – This phase will focus on developing our Bot. Now we have a well-developed rough draft and enough information to create a unique project. Our goal is to focus on creating a working plugin during this phase.

Milestone – First working plugin by week 7, and clear documentation for how the other two functionalities will work (if not already added).

Optional – testing for backwards compatibility (in reference to electron versions)

* Week 8-9: Revision - After spending some time in development, we should be able to create a functioning bot that we can test specific cases for. This would also mark the beginning of testing for our more specific issues (i.e., automated issue reports, messages, and closures) In this phase I would also like to spend more time researching any other advantageous additions (new env variables, dependencies, further research on other GitHub automation bots).

Milestone- Bot should perform 3 main features. Bot documentation 60% complete (Referring to readme & code annotations). Begin working on making bot merge-ready with electron codebase by examining and fixing possible merge conflicts (can be done by comparing files or attempting to merge).

Optional- Back testing for older electron versions

* Week 10-12: Further/final testing, further development – This phase will focus on the bot’s functionality in the electron environment. Final testing/ compatibility for added functionalities, and bot integration. Testing the bot for ease of use. As with all things, there is room for improvement.

Milestone- Bot implementation. User testing, complete documentation on current Bot deployment (Readme completed, as well as code annotation).

Optional- further documentation on how to adapt bot for others to use in personal projects