GSoC 2022: Proposal to Electron

Project: Implementation of Github Issues Automation in Electron Core

Table of Contents

ABOUT ME

Basic Information:

- Name: Chukwuebuka Cyril Muofunanya
- Address: 8 Nnamdi Ibegbu Street Onitsha, Anambra State, Nigeria
- Email: muofunanyachukwuebuka@gmail.com
- Telephone: +234 813 857 9991
- Time Zone: Lagos, Nigeria GMT +1
- LinkedIn: chukwuebuka-cyril-muofunanya
- Github: cyrilchukwuebuka
- Discord: Hooolycode#0993
- Personal Blog: chuk-cy
- Resume: resume link

Education:

- University: Federal University of Technology Owerri, Nigeria
- Degree: Bachelor of Engineering
- Current Year: 4
- Graduation: 2023
- Undergraduate Course: Electrical & Electronics Engineering

Work Experience:

- MLH Fellowship: MLH Prep Fellow as an Open Source Software Engineer (Internship)
  Certifications/Achievements: i) certificate link ii) Graduation Letter iii) Mentor's Message
- Genesys Tech Hub: Backend Developer (Internship)
- HNGi8 (Zuri): Frontend Developer (Internship)
- Hacktoberfest: contributed to various projects during the 2021 Hacktoberfest program.

Volunteered:

- Code For Africa 2021 (August 2021): A social development project on a mission to bridge the digital divide in Africa. It targets kids in rural communities equipping them with the right tools to stay abreast of technological advancements.

Certifications/Achievement: Certificate link

- IEEE FUTO Student Body Secretary (Feb 2020 - June 2021) was in charge of all correspondences, Membership registration, and aided in facilitating events, while I worked in tandem with the FUTO SB President to build a very vibrant branch in FUTO

PROJECT INFORMATION

Proposal Title:

Implementation of Github Issues Automation in Electron Core

Proposal Abstract:

With recent advancements in technology, and in Github to be precise, most actions are better off carried out by bot(s) which helps to improve and make development workflow seamless. The development of a complete and working Github bot requires a good knowledge of the Github App as a first class actor in Github, comprehension of API consumption (REST or GraphQL) using Github Octokit library and its usage in software development. The starting up of a Github app is a long and expensive development process, which in most cases, it is often preferable to use a Github App Manifest such as what Probot offers which eliminates all the drudgery like creating, installing, extending a Github app, receiving and validating webhooks etc so one could focus on the features one intends to build. Electron is an open-source runtime framework for creating desktop-suite applications maintained by the OpenJS foundation and an active contributor community and is widely used by millions of users. The Electron Github repository already has a few bots (Github App) such as Trop(for Backporting), Cation(for Semantic Versioning), WIP (for setting PR status), Semantic Pull Request(which ensures pull requests have Semantic title) etc that automates flows around PRs.

My proposal to this GSoC is to create similar automation around Issue Tracker to help triage and group issues by automatically adding labels based on the issue identification such as Bug, Fix, Feature Request, Mac App Report, GPU etc, triaging existing active issues into either Crash, Regression, Reproducible or Critical, Medium, Low, adding automated responses under specified conditions which could involve missing Electron Version, failure to check the three checkboxes, operating system, operating system version, architecture etc. and automatically closing issues not meant for Github community such as discord related reports/issues etc.

With automation around issues that I intend to develop, contributors can directly interact with the bot in order to create well-structured and information-packed issues, meanwhile helping codebase maintainers easily access organized issues of the same labels and level of urgency while the bot closes old and deprecated issues and unrelated issues such as issue meant for the Discord community.

Benefits to Electron:

The benefits that can be actualized once the Github Issue Automation is implemented and released on the Electron core repository are:

1. Currently, Electron has approximately 1500 active issues, which is strenuous and exhausting to triage manually, therefore, issues would be triaged and assigned labels automatically with the help of the bot, thereby automating the process saving open source maintainers ample time, improving the software development process, and increase efficiency and productivity while encouraging open source contributions in the Electron community

2. It would also help create information-packed and prioritized issues which would be relevant while trying to resolve issues meanwhile filtering out irrelevant/empty issues.

Features and Deliverable Specifications:

- ➔ **TASK 1** (Create a Github App using Probot CLI command): Research deep into the requirements needed to create an Issue Automation Bot using Github REST API endpoints (Issues endpoints basically) alongside knowledge of Octokit (@octokit/rest), Probot and Probot CLI command (yarn create-probot-app <issue-bot-name>). The use of Probot is to abstract the Github App creation and setup process through Github App Manifest. Probot also offers a Webhook Proxy URL through smee.io that enables Github webhook to be received on a development local machine. Setup hosting environment on Heroku platform, and development environment locally with folder structure as shown in the Source Code Structure for Typescript usage including the constant.ts, interface.ts, typings and enums.ts in source code for the created Github App bot and use Sentry Node.js SDK for automatic reporting of errors, exceptions etc (REQUIRED).

- ➔ **TASK 2:** Create a ts module file containing exported functions that handle the label addition triage feature and an exported function that checks whether a label

should be added or not. The root function of the label-addition-triage file is exported and called inside the index.ts file where it receives a Probot app instance as a parameter. Also, in the Probot App instance, authentication occurs within the Probot application Class public method called auth (eg app.auth()) where the app is an instance of the Probot Application class and is stored in the private githubToken declared variable. The root function listens to issue events through Webhooks.WebhookPayloadIssues ("issues" | "issues.edited" | "issues.labeled" | "issues.opened" | "issues.reopened") got from @octokit/webhooks which helps to handle webhook events received from GitHub and returns a webhook payload. As it is a TypeScript project, the types for the webhook payloads are sourced from @octokit/webhooks-types. Meanwhile, labels added to an issue could be a bug, fix, feature_request, blocked/needs-repro, gpu, os/windows, os/mac, os/linux, regression, reproducible, crash, critical, medium, low, new_issue, ipc/handling, ipc/main etc (REQUIRED)

➔ **TASK 3:** Improve on the already created issues bot to include an automated response feature The automated response may come as a result of "a missing reproduction gist", "an issue being blocked and needs repro for above 60 days", "an issue is missing any important information such as Electron Version, Operating System, OS Architecture, OS Version etc". This feature contains separate exported module files with functions listening for events relating to issues, issues commenting and edited events through Webhooks.WebhookPayloadIssueComment ("issue_comment" | "issue_comment.created" | "issue_comment.deleted" | "issue_comment.edited") on Github and receives its unique webhook payload (REQUIRED).

➔ **TASK 4:** Create an additional issue bot action feature that automatically closes issues when specific conditions occur such as "Reporting an issue for deprecated versions of electron", "issues which have been marked blocked for a given number of days", "unrelated issues or issues meant for the Discord community" etc and as part of the auto-response feature, when a reporter or contributor submits an issue having discord related scenario, the issues bot sends a reply that the issue should be posted on the discord channel community attaching the discord community channel link to the response (REQUIRED).

➔ **TASK 5:** Create concise and lucid documentation on how to use the Issue Automation Bot project in any repository and a blog post of detailed exposition on how I went about the development processes (REQUIRED).

➔ **TASK 6:** Create a new set of test suites on the Issue Automation Bot to provide reliable testing on its features. In order to carry out the developmental process in a Test-Driven approach, test cases at various stages would be done using the Jest framework and Nock. Nock is a tool for mocking HTTP requests, which is often crucial to testing in Probot as it depends heavily on Github's API, meanwhile, Jest allows you to write tests with an approachable, familiar and feature-rich API that gives you results quickly. (REQUIRED).

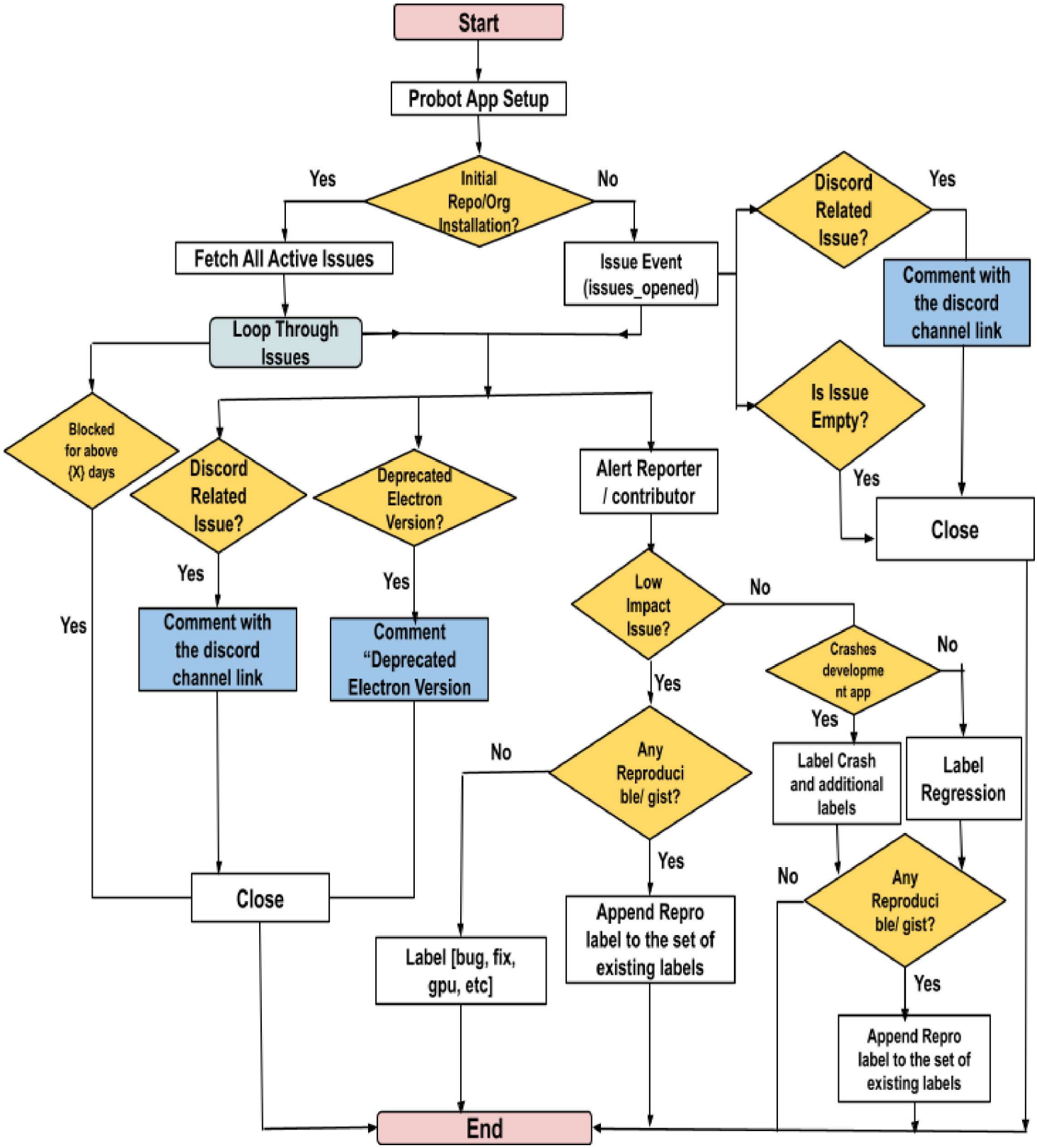
Program Flow Design:

On every Probot app initial installation in a repository, it fetches all active issues, triages them with the appropriate labels and closes issues having deprecated versions or blocked for above certain days. When a new issue is created, extra information is gotten from the reporter to help in triaging the new issue on the go. And as part of the auto-response feature, when a reporter or contributor submits an issue having a discord related scenario, the bot sends a reply that the issue should be posted on the discord channel community attaching the discord community channel link to the response.

In addition to the existing workflow, whenever a new issue is created, a **new_issue** label is added to the issue which lasts for a duration of 48hrs before the label is removed as a new issue.

A list of a few labels that could be added include**:**

i) bug, **ii)** fix, **iii)** feature_request, **iv)** gpu, **v)** platform/windows, **vi)** platform/mac, **vii)** platform/linux, **viii)** regression, **ix)** has-repro-gist, **x)** crash, **xi)** critical, **xii)** medium, **xiii)** low, **xiv)** new_issue, **xv)** hacktoberfest, **xvi)** has-repro, **xvii)** blocked/needs-repro **xviii)** z-y-x (Electron version labels) **xix)** unsupported version **xx)** app/slack **xxi)** app/discord **xxii)** chromium **xxiii)** component/<name> **xxiv)** memory-leak **xxv)** arch/<arch -type> etc



Flow Chart Schematic Diagram describing a pictorial overview of the automation process

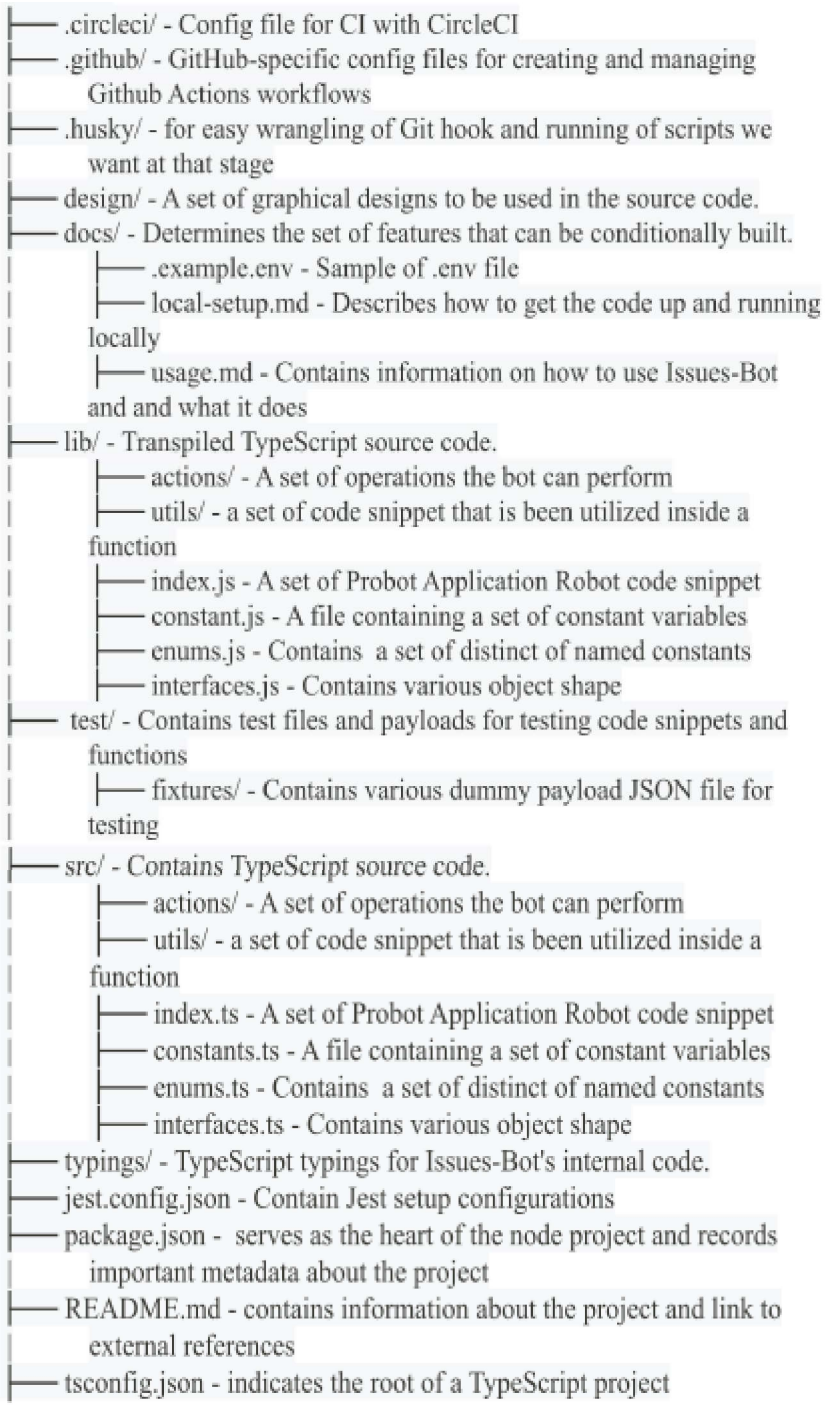
Technical Requirements (Libraries, Technologies and Languages):
According to research and information I could lay my hands on, alongside my experience of having worked on an automated feature using the Github OAuth App, the expected libraries, technologies and languages include:

- Probot
- Nodejs and Yarn
- TypeScript
- Jest and Nock
- @octokit/rest and @octokit/webhooks
- Sentry SDK
- Heroku
- Github and Github App

FUN FACT: Probot Context.octokit function was made from Octokit Core and currently has support for making calls to Github Rest API Endpoints.

Source Code Structure:

The source code structure uses several directory layers to arrange the codes. I propose to develop the Issues Automation basing the foundation on this structure and add relevant directories and files as the development progresses.



Source Code Structure format crafted from Electron official Documentation guide and standard

Project Timeline and List of Deliverables:

PHASE 0

3 Weeks (May 20 - June 12) (Community Bonding Period)

- Introduce myself and this project in the Electronjs.org discord channel.
- Remain in constant touch with my mentor through the discord channel or using Google Hangouts or any medium of my mentor's choice.
- Discuss with mentors about the implementation plan/workflow, specification writing and code style to be used.
- Set up a development environment for the project, and try to get myself familiarized and to get a further understanding of Trop, Cation and Probot for Github App creation and its architecture.
- Get a solid understanding of Github Actions, Events, Event Emitters, Webhooks, Octokit etc.
- Set up a blog post for the project and figure out a way to document everything during the development process of Github Issue Automation so that future development can refer to it.

PHASE 1

2 Weeks (June 13 - 24)

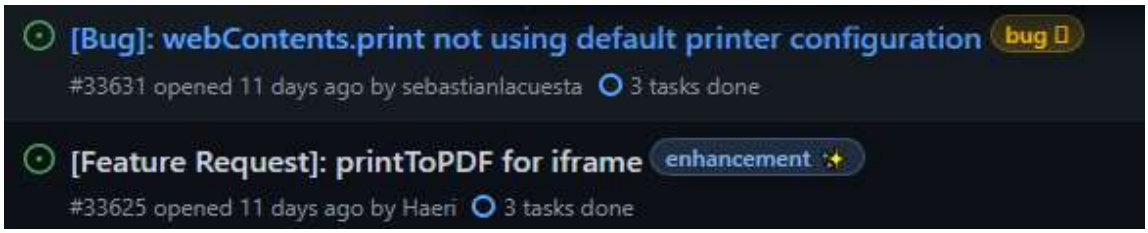
MILESTONE: Set up a Github App using Probot (Probot CLI Command), Connect to a Dummy Online Repository.

- Generate a Github App using the yarn create-probot-app <issue-bot-name> CLI command, organise the created folder structures with appropriate names and necessary utility code snippets and configs according to the final design with mentors and remove unwanted files. Efforts would be made to ensure the creation of **reusable functions** which will aid in unit testing.
- Test the generated Probot App (Github App) bootstrapped from the CLI command making sure the probot app starts properly locally and fetches the required .env variables such as WEBHOOK_PROXY_URL, APP_ID, PRIVATE_KEY, WEBHOOK_SECRET, GITHUB_CLIENT_ID and GITHUB_CLIENT_SECRET.
- Install the app on a test online repository (dummy repo).
- Debug the code of any error or bug, test the probot app manually with the dummy repository against an issue_opened event which creates a comment once an issue opens on the dummy repository.
- Submit an initial code commit to the main repository.
- Dedicate some time for community help.

2 Weeks (June 27 - July 8)

MILESTONE: Implement a Basic Issue Label Addition file

- Basic issues as the name imply involve **new_issue**🌱, **bug**🐞, **fix**😅, **feature_request** (**NB:** any feature request would be labelled as **enhancement**✨). Most information to help determine the basic label is obtained from the title inside the payload from the issue template using the probot octokit function octokit.issues.get() which fetches the issue.



- Create a label-handler-util file in the **util directory**, the file bears exported functions for triaging and labelling issues. Among the functions to be created includes an **addLabels** function in which **"octokit.issues.addLabels"** is utilized, **removeLabels** function utilizing **"octokit.issues.removeLabel"**, **getLabelsForIssue** function utilizing **"octokit.issues.listLabelsOnIssue", getIssueOpenedTime** for determining the time an issue was created against the current time and many more useful features that would come in handy. And also **token-util.ts** (contains a function for generating Github access token to be used while accessing protected Github REST API endpoints which call the **auth()** found in the Probot Application class).
  NB: the octokit is obtained from Probot Context.
- Separate constant code variables into the **constant.ts** file (NEW_ISSUES_LABEL, CRASH_LABEL, REGRESSION_LABEL etc), enums into **enum.ts** file (LogLevel, BasicIssueLabel etc) and few files as **.json files** such as Electron current versions.
- The Basic Issue Label Addition file receives a Probot app as its parameter and listens to Github events such as Webhooks.WebhookPayloadIssues ("issues" | "issues.edited" | "issues.labeled" | "issues.opened" | "issues.reopened"). Inside the file, there's a 48hr Rule function that adds and removes the **new_issue** 🌱 label from the set of labels in the issue and updates the issue label with other necessary basic labels. In the 48hr Rule, it also listens specifically to "issue.close" event so as to remove the **new_issue** 🌱 label if an issue was closed on or before the 48hr duration.
- Set up an automated testing environment in the **test** directory using **Jest** and **Nock**. As already described earlier, the nock package would help to mock Github API

events, and HTTP requests while making use of a dummy webhook event payload data gotten from the Github App settings page (Advance setting page) which would be stored in the **test/fixtures** directory. **NB:** my previous testing process has been manually carried out on the little feat made.

- Write all test cases for each function written inside of the Basic Rule file inclusive of the 48hr Rule function and at the same time, learn best practices for writing test cases in the process.
- Write documentation based on the feat I've attained, and write a blog about the development progress.
- Submit code commits to the main repository.
- Dedicate some time for community help.

2 Weeks (July 11 - 22)
MILESTONE: Implement a Label Triage Rule file

- In this file, functions that facilitate the addition of other secondary yet crucial labels are found here. These functions peruse the entire body of the issue when fetched from the payload and check for the **Electron Version**, **Operating System**, **Operating System Architecture** and whether the operating system has **Reproducible Gist**. It also checked for an issue **crashing the app** or being a **regression** or an issue being of **Critical, Medium or Low** effect.
- This file function would be utilizing most **label utility functions** available in the utils directory while listening to similar events and webhook of the **Basic Issue Label Addition file.**
- Write detailed documentation of the new features, what each function does and test out the functions using **Jest** and **Nock** alongside **dummy payloads.**
- Submit the code to the main repository
- Dedicate some time for community help

*** First Evaluation Period**
1 Week (July 25 - 29)

- Prepare for evaluation.
- Deliver working automation around label addition and issue triage for evaluation from my mentor.
- The bot can be installed into the Electron core repository and triage all active issues and future issues, adding labels according to the severity and classification of the issue.
- Fix minor bugs that may arise during the Electron installation process.

- Planning and specification of the second and third features.

PHASE 2

2 Weeks (July 11 - 22)

MILESTONE: Implement an Automated Response Feature

- Automated responses could include **“a missing reproduction gist”**, **“an issue being blocked and needs repro for above 60 days”**, **“an issue is missing any important information such as Electron Version, Operating System, OS Architecture, OS Version etc”**, **"a response asking if the issue is a Crash or Regression"**, or **"a response asking for the severity of the issue such as being Critical, Medium or Low"**. All these responses could be stored in a **.json** file or as an **Object** constant file.
- Create a comment-handler-util file in the **util directory**, the file bears exported functions for creating automated response issues. Among the functions to be created includes a **createComment** function in which **“octokit.issues.createComment”** is utilized, **deleteComment** function utilizing **“octokit.issues.deleteComment”**, **getCommentsForIssue** function utilizing **“octokit.issues.getComment”.** There also would be a function that checks if the comment on an issue was created by a bot or not and whether the commenter is an owner of the issue created so as to know how best to respond to the comment event trigger.
- In this auto comment scenario, Github event triggers listen to events from Webhooks.WebhookPayloadIssueComment ("issue_comment" | "issue_comment.created" | "issue_comment.deleted" | "issue_comment.edited")
- With the test environment having been set up using **Jest**, **Nock** and **dummy payloads**, verify the functionality of various utils functions created for the automatic commenting feature to be sure each of them works as it should.
- Some time would be utilised to debug and take feedback from the community.
- Write documentation on the automatic commenting features created, a blog about my progress and submit the code to the main repository
- Dedicate some time for community help

2 Weeks (August 15 - 27)

MILESTONE: Implement an Automatic Issue Closure Feature

- Conditions in which an issue could be automatically closed could be as a result of **“the version of an electron being older than the supported Electron versions”**,

“when an issue has been marked blocked for more than a specified period of time” or **“when the issue opened is specifically meant for the Electron discord channel”** and many more likely conditions that could warrant issue closure. Meanwhile, when an issue is discovered to be a Discord issue, the issue is commented with the **Electron Discord Channel Link** using the **createComment** utility function created earlier before the issue is closed so as to make it easier for the reporter/contributor to get into the discord community and raise the issue in the community.

- To attain this automated issue closure feature, an **autoCloseIssue** function which takes a probot app as a parameter is created and set to listen to events as listed in the braces ("issues" | "issues.edited" | "issues.labeled" | "issues.opened" | "issues.reopened"). The function utilizes the **getIssueOpenedTime** utility function to get the open time of an issue that has been marked as blocked and compare it with the current time. If the time frame has attained or exceeded the set limit of a blocked issue, the issue is automatically closed.
- For the Electron version, once an issue is opened, reopened or edited, it checks for the version from which the issue is being reported, once noticed to be a deprecated/unsupported version, the issue is being automatically closed.
- With **Jest**, **Nock** and **dummy payloads**, verify the functionality of various functions created for the automatic issue closing feature to be sure each works as it should.
- Some time would be utilised to debug and take feedback from the community.
- Write documentation on the automatic issue closing features created, a blog about my progress and submit the code to the main repository
- Dedicate some time for community help

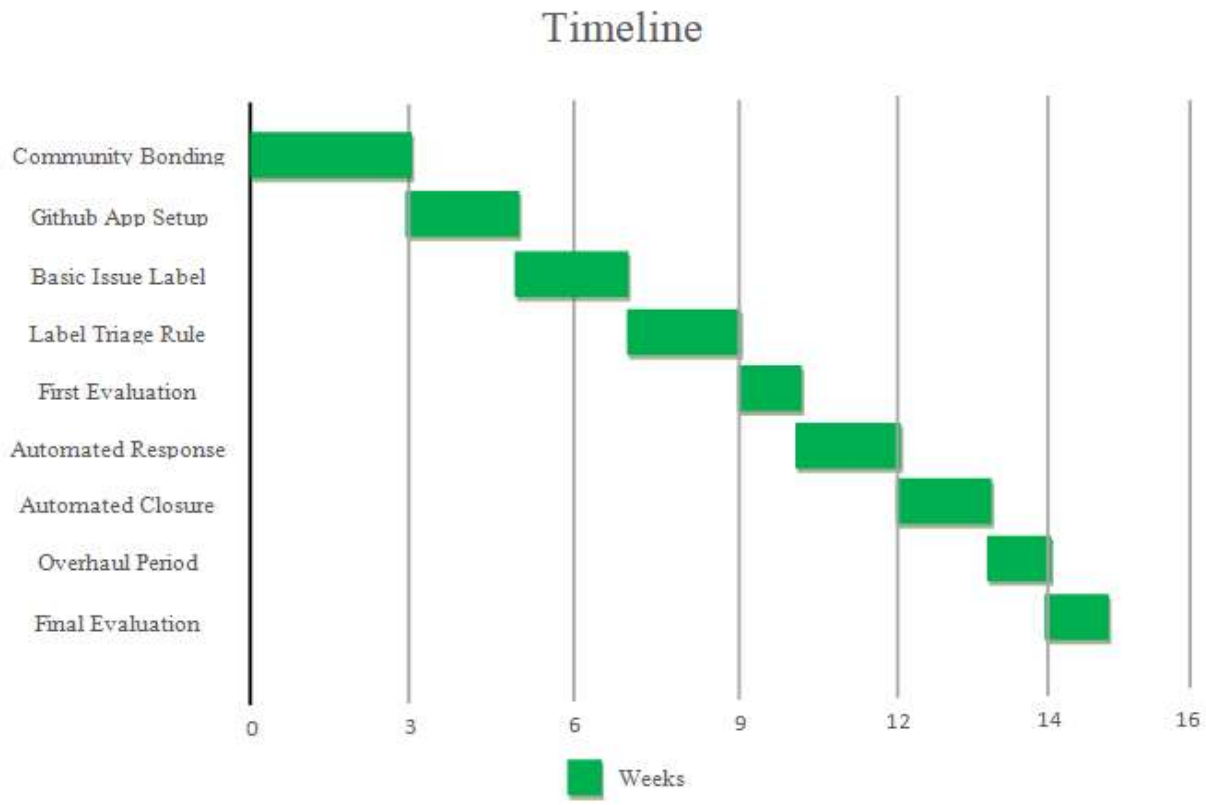
1 Week (August 30 - September 3)
MILESTONE: Period to overhaul the entire project and check out items from the List

- Go through the entire project and fix issues that were skipped in the past.
- Add any minor feature found relevant during the project development and idea/knowledge exchange with mentors and the community.
- The added feature could be assigning an issue to a specific maintainer based on the issue triaging suggestions.
- Write more detailed documentation about the Issue Automation bot to help users and the community.

- Create a complete test suite that puts into account all the Issue Automation capabilities/features and sees that everything is working as it should on the Electron core repository.
- Test the Issues Automation Bot with other Electron GSoC projects.

Final Evaluation Week (September 5 - 12)

- Finish everything, especially documentation!.
- All the deliverables promised should be accomplished by this stage.
- Wrap up the project and submit it for final evaluation from my mentor
- Start working on a long term goal with Electron, continue my contribution/maintenance on this project idea and support the Electron Discord community and Github.
- Plans to become an Electron mentor for future batches of GSoC or other Open Source programs.



Gantt Chart

EXTRA INFORMATION

Studies:

I am an undergraduate student at the Federal University of Technology Owerri and would love the GSoC program to serve as a place for my forthcoming summer internship placement. I major in Electronics and Computer Engineering and use Github a lot for source code storage and for project development. I have developed both Frontend projects and Backend API projects and also used Github OAuth App to automate a feature in Github. I'm also working on a book reading platform project called NeverLand, with a team of five folks(pods) where I happen to be amongst the organization owners, working on both the frontend and backend features (cross-functional role) at the same time, handling the various actions and merging of a PR that occur in the repository.

This proposed project will definitely contribute to my study because it will not only make it more convenient to understand the field of automation but also help me improve my skills in Github App's vast application areas. I have strong proficiency in Javascript and Typescript usage alongside Node.js, React library and Express framework and uses Windows OS on a daily basis.

Lastly, I'm also in for an Industrial Training(Summer Internship) official granted to me from my University as it's a very important component of academic training from a University.

Below is my Industrial Training Attachment Form Image Link:
i) Request for Industrial Attachment
ii) Placement Form

Related Work:

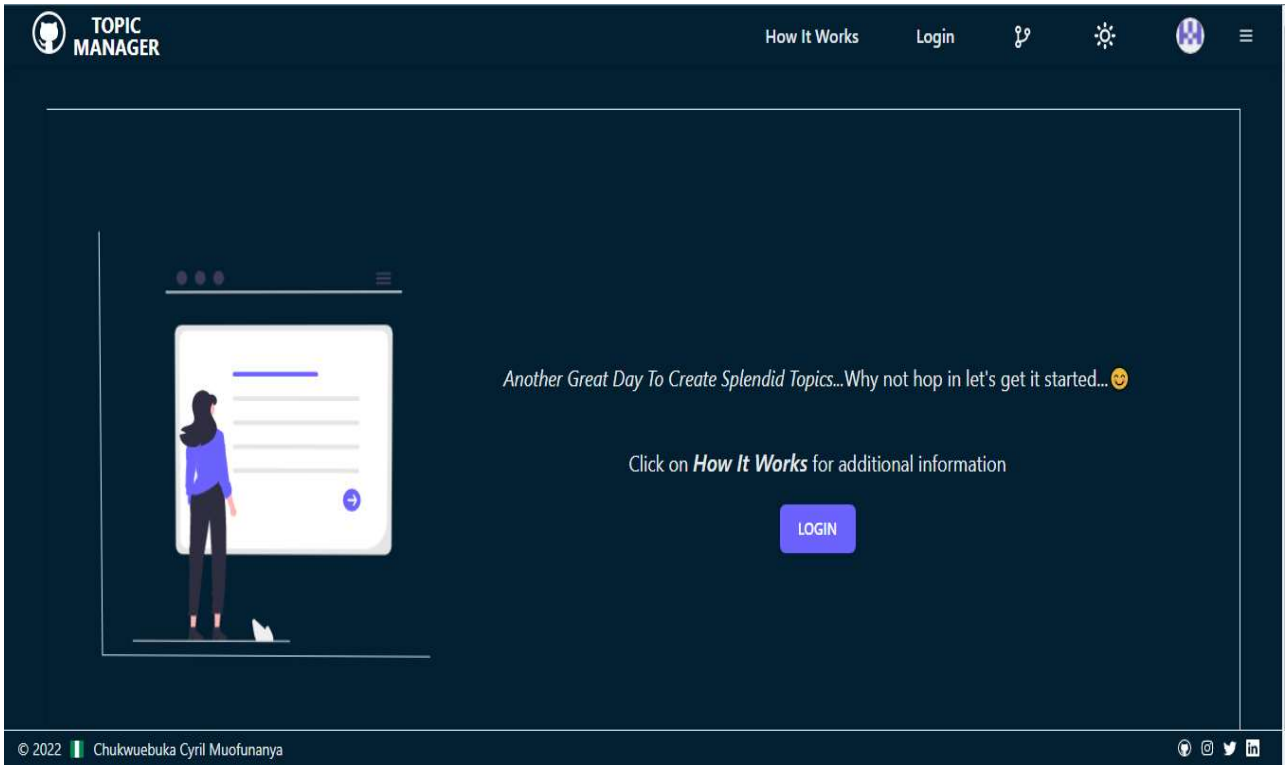
Topic Manager Project is a Github Oauth web app that streamlines the process of adding or removal of Topics by developers and Github users to one or more Github repositories. With topics, exploring repositories in a particular subject area, finding projects to contribute to, and discovering new solutions to a specific problem becomes easy.
Topics appear on the main page of a repository. Clicking on a topic name refers you to see related topics and a list of other repositories classified with that topic. And as a Developer, I find it strenuous whenever I wish to add Topic(s) to a repository or multiple repositories of mine and also as a lover of automation and one who loves to proffer

solutions to challenging problems, I came up with this project which passed through the thorough thought process and software development cycles and solely built from scratch without any external input or clone of an existing project of such functionality to help solve the obvious problem.

Below is the Github link: https://github.com/cyrilchukwuebuka/github-topic-manager
Live link: https://github-topic-manager.netlify.app
Interesting facts about the project which are useful as regards the Github Issue Automation Project are:

- It is a Github Oauth App
- It uses Github Octokit to make requests to Github API endpoints
- It performs user authentication and uses the access token to make requests to secure API Endpoints
- The API endpoints used are REST APIs

I would appreciate it if you also give it a Star.
NB: In terms of the web app usage, it's only in the User scope of access as most individuals don't permit apps with access to the Organisation.



Snapshot of the Topic Manager Project

SUMMER AVAILABILITY / ADDITIONAL NOTES

I would be officially free from school activities, the MLH Fellowship program and Genesys Tech Hub Program in April 2022 for professional and ethical reasons thereby having no commitments in the summer as I would be staying back home for the most part of it so as to enable me to place a focused mindset and be available full time on GSoC at around 41hrs per week (8hrs per day and few hours spanning across weekends) to finish the program in good faith. Also, since I would be less occupied on weekends, I would be using it to make a personal plan with respect to the proposed timeframe to make the coming week more fruitful. But I would also use most time at the weekend for resting as it is important both for health and better commitment to the planned time frame. Meanwhile, I will also engage in programs and activities taking place in the discord community channels while learning new things, unlearning wrong ones and relearning at the same time.

GSoC PARTICIPATION and WHY ELECTRON?

This 2022 GSoC program is my first time applying for Google Summer of Code so naturally, I was a bit timid to interact with the organizations. I felt most comfortable interacting with the Electron community. The mentors are really nice, enthusiastic, active and cordial.

I like this Electron project idea as a whole because of its idea of taking repeated tasks away from human beings into robots' care so humans can focus on advancing research and finding new ways to make life easier and fun.

I want to be a long term contributor to Electron, help in future mentorship programmes such as GSoC, GSoD, Outreachy, Hacktoberfest etc. and implement new automation and plugin features. I also did not submit a proposal to any other organization as this project idea will have a positive impact on my academic life.

SUM UP

I believe that my proposal project idea is an opportunity to bring more developers to Electron projects and even better open source contributors and happier codebase maintainers. The Electron organizational repositories are lacking essential automation features around Issues.

With my undergraduate experience on a swarm, I have the knowledge both in Engineering needs and in my programming experience that makes me a good candidate for this GSoC.