The Tor Project: Tor Weather

About me

Name	Sarthik Gupta
Email	sarthikg@gmail.com
Mobile Number	+91-8872425152
Location	Chandigarh, India - IST (UTC +5:30)
Work	Software Engineer at Soroco
Education	B.Tech (2021) in Civil Engineering from Punjab Engineering College, India
Website	https://sarthikg.com
Resume	https://sarthikg.com/resume.pdf
Github	https://github.com/sarthikg
Matrix	@sarthikg:matrix.org

Introduction

Hi, I am Sarthik Gupta, a recent graduate from Punjab Engineering College, Chandigarh who just completed his B.Tech - Civil Engineering in 2021. For a long-time, I have been on a look-out for tools to express my ideas, and my first exposure to programming drew all my interest into it.

So, the journey started in 2018. As a self-taught developer, I explored languages, fields, possibly everything that I could. During this time, I worked as a full-time Business Analyst Intern for 6 months, and finally picked up a full-time position as a Software Engineer where I am working for the past 8 months.

After these 14 months of corporate experience, I still feel there is plethora to learn out there. In the past, I have made some attempts to participate in the open-source community, but for some reason, I couldn’t find it easy to enter. There were issues I

raised, and asked if I could raise a PR for the fix, but before I received a reply, one of the maintainers fixed the issue.

Recently, I came to know that GSoC expanded its horizons beyond college students, and I was super excited when I heard about. I could finally see a way to step in the world of open-source & contribute to such amazing tools that we use almost daily in our lives without even realising.

Project Information

Project Abstract

The Tor-Weather project aims at creating a notification service upon which the relay operators can rely upon for incidents & updates of their relay nodes. The relay operators will finally feel that someone actually cares about their effort of running the nodes.

As a regular user of tor, and an advocate of privacy, the tor project was always a project I would have loved to contribute to. While glancing through the GSoC projects, Tor-Weather caught my eye as it was very similar to something I had worked on before.

The idea of Tor-Weather isn’t new, and has been attempted at-least 3 times before this GSoC, and unfortunately didn’t work out in either of them. This time around, I aim at joining it as a long-term maintainer even after GSoC, but also making it rather easy to maintain and easy for developers to contribute to.

A service is as good as its promotion among the target user-group. With this in mind, I believe that there should be as little friction from using the service as possible. This includes, advertising the service among relay operators, but also preventing the targeted spam the service could introduce. For this reason, its best to send a Thank You email to the old relay operators, and a Welcome email to all the new one’s which advertises tor-weather, while also providing an email-verification system to prevent spam.

In-terms of the tech-stack, I think for an open-source project its best to go with something that is popular among developers & easy to contribute to. For this reason, I chose a micro-service architecture with Flask as the backend.

By the end of the GSoC, I aim at creating a tor-weather service that is able to:

1. Send notifications to the subscribed relay operators upon incidents.

2. Provide a web-frontend for users to signup and tinker with the notification preferences.
3. Send welcome emails to the newly onboarded relay operators.
4. An admin-flow that allows the admins to push high-priority emails to the relay operators.
5. Perform all the above operations in future even without a developer actively maintaining it.

Project Description

Tor-Weather is a notification service which helps relay operators get notified when some incident happens with their relays. The service has 2 major functions:

1. To provide notifications on possible incidents with the relays
2. To incentivise relay operators for their work

It is an essential project for the tor network as it aims at keeping the network updated & running at all times by making sure the operators are well aware of the status of their relays. The project has been undertaken multiple times in the past, which includes:

1. Tor-Weather (2010 Version)
2. Tor-Weather (2014 GSoC Version)
3. Tor-Weather (Unofficial Version)

Unfortunately, none of the implementations were continued for a long period due to a lack of maintenance. This year, the project has again been chosen to be a part of GSoC, and I feel confident that this would be the final implementation.

Fundamental Architecture

For the new implementation of Tor-Weather, there were a few available options for the user-flow :

1. **Subscribe users from the the ContactInfo field from torrc**

   Pros:

   1. ContactInfo field from the torrc will work exactly as intended.

2. Greatly reduces the friction from using Tor-Weather

Cons:

1. The email provided in the ContactInfo is obfuscated in most of the cases as they should be according to the torrc documentation. Hence this information cannot be relied upon.
2. Spam can be introduced if someone creates multiple relays with the victim's email-id. We cannot verify this information, hence the spam is unavoidable.
3. The service subscribes relay-operators for the updates without their consent. This will cause a lot of them to flag the emails as spam, which will cause email providers like Gmail, Outlook, etc to mark our email server as spam.

2. **Create an entirely separate signup-flow**

   Pros:

   1. Reduces the chances of spamming when integrated with an email-verification user-flow.
   2. Users can subscribe to updates of relays which don't necessarily have their ContactInfo in their torrc's.

   Cons:

   1. High friction from using Tor-Weather.
   2. No way for the users to know that such a service exists which they can subscribe to.
   3. The ContactInfo field is rendered useless

3. **Hybrid approach using ContactInfo to advertise Tor-Weather & a separate user-flow**

   Pros:

   1. Advertisement of Tor-Weather happens among existing relay operators (at initiation) & among newly joined relay operators.
   2. Reduced chances of spamming when integrated with an email-verification user-flow.
   3. Less friction from using tor-weather

4. Less probability of emails getting marked as spam.
5. The ContactInfo field is partially used for the purpose it was intended for.

Cons:

1. ContactInfo field cannot be completely used, as the text is still obfuscated in some cases.

Post this preliminary research, the approach-3 seemed like the best approach to go with. Hence I did some research to calculate how much of an impact the only con could have:

- **Total number of relays** = 7576 (as of 15th April 2022)
- **Percentage of relays with no ContactInfo** = 20-21%
- **Percentage of relays without obfuscated email** = 34% of relays with ContactInfo
- **Percentage of relays with parsable obfuscation** = 84%-85% of relays with ContactInfo

The last figure of relays that had a parsable email obfuscation was done with a small program which was scribbled in a few minutes that tried to replace the common obfuscations like “[at]” with “@” upon small observation of data. This program can be accounted for about 10% of error cases, which still holds up for 74-75% of emails that were readable. Upon further improvement, this can be improved to about 90% readability rate.

Taking into account the current percentage of emails that could be read from the ContactInfo field, the launch of Tor-Weather will be broadcasted to about 4500-5000 relay operators out of the total of about 7500, which seems like a good surface area to begin with. Hence, the approach-3 should be the best suited approach.

Technical Design Considerations

There were a few options available for the technical design of the Tor-Weather. Some of them were:

1. **Should data be fetched from Onionoo or Stem?**

   The previous implementation of Tor-Weather (2014 version) used Stem for the purpose of knowing the status of relays on the tor network.

What is Stem?

Stem is a python controller library for tor. It was launched in 2013, and has been last updated in 2019. Stem could be used by projects which planned on making multiple requests querying the status of the tor-network, as it promoted to run a relay and query status from the local relay. This prevented projects from overburdening the network.

What is Onionoo?

Onionoo is a web-based protocol for applications to query the status of the tor-network. It was a service primarily designed for applications that presented data in a human-readable format. Onionoo was launched in 2014, and was last updated in 2020.

Conclusion

It seems like Stem was used in the 2014 implementation of Tor-Weather since Onionoo was a very recent service back then, while Stem was still there for about a year, and hence was more stable. In-terms of development, using Stem doesn't seem to have any advantages over Onionoo as Stem is less-maintained, and will be relatively hard to setup and contribute to by a developer. Currently, the unofficial tor-weather implementation also relies on Onionoo, and it seems like Onionoo should be the way to go since the number of queries executed by our application should be relatively small, and Onionoo should be able to easily handle them.

2. **What should be the best tech-stack for the project?**

   Personally, I believe there is never any "best tech-stack" available. With the ever-evolving tools & technologies, we can choose the one which is common and, popular enough that it doesn't become a roadblock for any developer to contribute to. Keeping that in mind, the two possible programming languages available at our disposal should be:

   1. **Python**

      Python has been very famous among developers, be it for the purpose of backend, or data analysis. For the backend, there are 2 major options available:

      a. **Django (Framework)**

         - Django was used previously in the Tor-Weather (2014 Version) and is tried & tested before.

- Django comes with many services inbuilt like an ORM, Email Service, etc which can be extremely useful in our case.
- A framework makes it really hard for contributors unaware of the framework to contribute.

b. **Flask (Library)**

- Flask gives the freedom of being a library and users knowing python can easily contribute to it.
- Some functionality will have to be imported from other libraries which could have been available out-of-the-box incase of Django.

2. **JavaScript**

   JavaScript is also very popular among developers as it can be used for both frontend & the backend. Though, during the initial conversation with my mentor, it seemed like the organisation didn't prefer using javascript for various reasons. So dropping the option of using it.

Taking into account the available options and their pros & cons, it seems that **Flask** will be a better fit. Django even though used previously and providing extensive out-of-the-box features, limits the subset of developers who can contribute to it, which is a major factor when opting for an open-source project.

3. **What is the code reusability from the previous implementations?**

   Code-reusability from the previous implementations shouldn't be a direct copy/paste effort as the data source is changing. Though, the previous codebases can be used/referred for:

   a. Structuring the application
   b. Email Templates used in the previous versions
   c. Database structure & authentication management
   d. Default conditions under which various subscriptions were triggered

4. **What would be the maintenance effort?**

   The planned structure of the application is to move from a monolithic architecture to a micro-service architecture. This should have the following benefits:

a. Maintenance effort will greatly reduce as small services will handle separate responsibilities.
b. Contributing to the project will be easier for developers as complexity is divided among the application.
c. Security will be greatly improved & will demand less effort as there will be few public facing systems.
d. Adding features in the application will be easier as new services can be introduced. This also opens door for using different technologies for different services in the future.
e. Few services can be deployed as FAAS (Function as a Service) which will also reduce server costs.

Email Notifications

1. **Email me when my node is down - Here we should decide how long before we send a notification?**

   A notification is triggered when the relay is down for X time. The time before a notification is triggered will be a configurable value that the relay operator can set in the web-frontend. Though, for getting started, there will be a default value of say 4hrs.

   Implementation

   The proposed database design currently supports this feature. The option needs to be added in the web-frontend, and will work with the python scripts accordingly.

2. **Email me when my relay is affected by a security vulnerability**

   A notification is triggered when the relay is affected by a security vulnerability. Currently, this feature will rely on the the version information. Incase of specific outdated versions, a notification will be sent to all the relay operators with relays on that specific outdated version.

   Implementation

   The proposed admin screen will have an option for the admins to release a notification for the relays on a specific version. This will execute a one-time script

notifying all the relay operators having a relay operating on that specific version. In the current implementation, this will be a manual effort for a more granular control, but the idea of automating this will be discussed with the team.

3. **Email me when my relay runs an end-of-life version of tor**

   A notification is triggered when the relay is running an “end-of-life” version of tor. The end-of-life version will correspond to the “obsolete” version from the Onionoo data.

   Implementation

   The proposed database design currently supports this feature. Every hour, upon the execution of python scripts, there will be a notification rolled out for the relays running on end-of-life version of tor. A mechanism for making sure each relay operator is notified only once for a relay is also taken care of.

4. **Email me when my relay runs an outdated tor version**

   This notification is similar to the one for the “end-of-life” version, but will be fired when a new version of the tor is available.

   Implementation

   Implementation of this notification will be based out on the data from the Onionoo. The `version_status` field contains the relevant data for fetching the latest version of the tor. The proposed database design is capable of handling notifications for this use-case.

5. **Email me when my relay loses the stable/guard/exit flag**

   This notification is triggered whenever the relay loses any of the stable/guard/exit flag.

   Implementation

   The proposed database design supports this functionality. The Onionoo data contains data for the flags, and comparing with the earlier state of flags in the database, a notification will be fired whenever a flag is missing.

6. **Email me when my MyFamily configuration is broken**

   This notification is triggered whenever a relay exits a MyFamily configuration.

   Implementation

   The proposed database design supports this functionality. Upon execution of the python script (happens every hour), the existing relays in the MyFamily will be compared with the new set of relays in the MyFamily. If any relay is lost, a notification is fired.

7. **Email me with suggestions for configuration improvements for my relay**

   This notification is triggered when the configuration of a relay can be improved in some possible way.

   Implementation

   The application can have a set of configurations (best practices). The configuration of relays as available from Onionoo can be compared with these best practices, and a notification can be triggered with the possible steps to mitigate the issue. The proposed database design does not support this feature, but with further communication, an ideal way of handling such a use-case can be proposed and implemented.

8. **Email me when my relay is on the top 20/50/100 relays list**

   Whenever a relay is a part of the top 20/50/100 relays, a notification is triggered for the subscribers of that relay.

   Implementation

   Relays can be ranked on different metrics. The best metrics for the use-case are `advertised_bandwidth` & `consensus_weight`. A discussion with the team is necessary for deciding among these & an approach for handling this will hence be implemented.

9. **Email me with monthly/quarterly status information, e.g what is my position in the overall relay list, how much traffic did my relay do during the last month,**

etc...

This notification is triggered every month notifying the relay operators of metrics like the position of their relays, the overall bandwidth handled by their relays.

Implementation

A python script for this functionality can be scheduled to run every month by the crontab. The corresponding data for this notification is available from Onionoo directly, hence notifications can be sent to the relay operators. A discussion with the team is required for an option of changing the preferences for this type of notification before implementing it.

10. **Email me about new relay requirements**

    This notification is triggered for the new relays recently joining the tor-network for the relevant requirements of a relay.

    Implementation

    This notification can be triggered once a new relay is registered on the tor-weather. A static email for basic requirements for new relays could be mailed to the operators. The complete specifications outlining the use-case will be discussed with the team before implementing it.

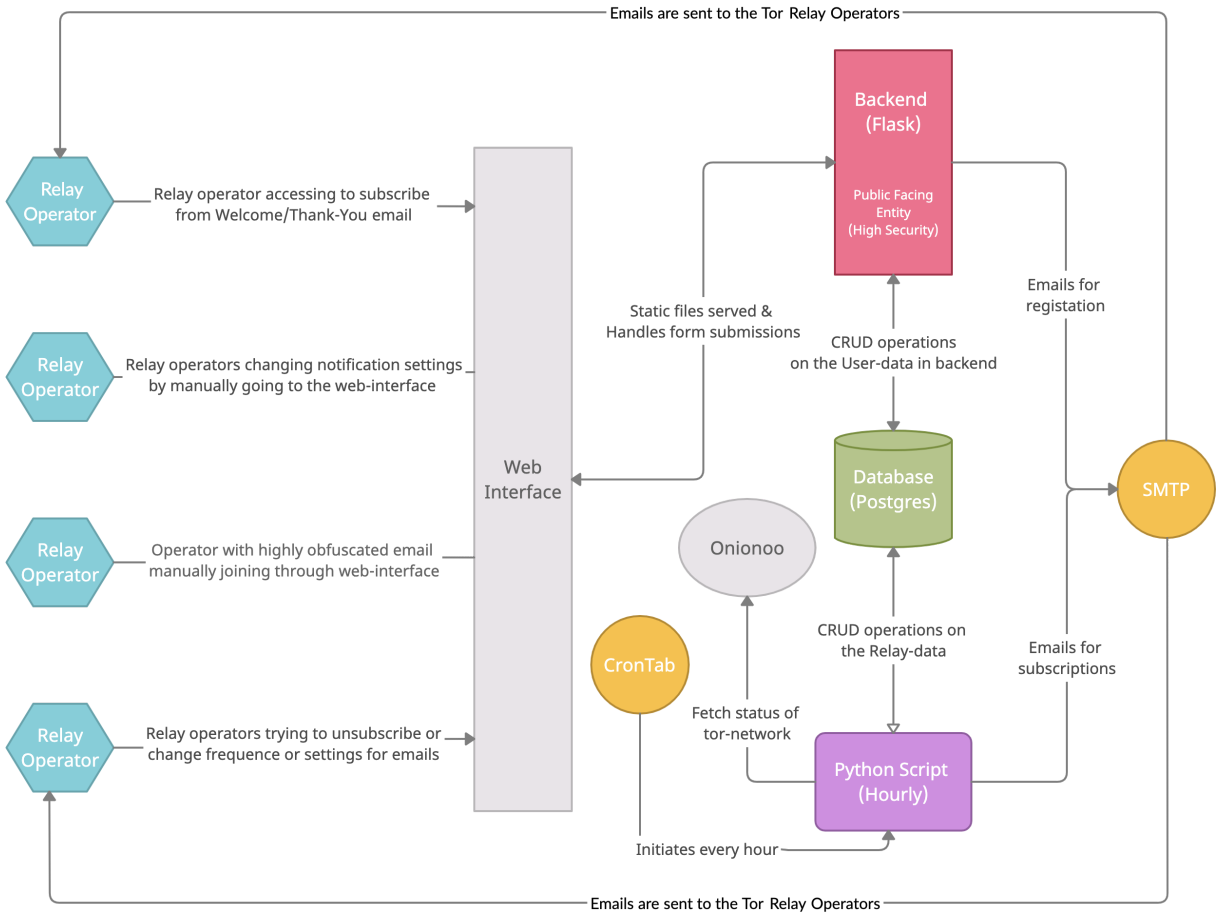
11. **Email me about tor relay operator events**

    This notification will be triggered to the tor-operators when an event for tor operators is upcoming.

    Implementation

    In the first phase, this notification can be manually triggered by the admins from the admin dashboard, but a discussion for automating the process will be done while exploring the possible ways for handing event invites.

System Design



The design for the tor-weather app will follow a micro-service architecture where the backend will be a completely separate entity from the scripts. Let's dive deep in the architecture:

1. **Python Script**
    a. Python scripts will be handling the main logic for the status of the tor-network.
    b. Crontab will be invoking the Python scripts every fixed interval (1hr in this case).
    c. Once triggered, the script will fetch the status of the network from the Onionoo API.
    d. With the previous state of the network already available in the database, the script will send relevant email notifications to the relay operators while updating the relay data in the database.

2. **Backend (Flask)**
    a. The backend primarily focuses on user-centric tasks like sign-in, sign-up, changing frequency/parameters for notifications, unsubscribing, etc.
    b. Backend will fetch this data from the database & will serve server-rendered templates to the client.
    c. It will also handle the form-submissions from the user, and will modify the user preferences accordingly.
3. **Database (Postgres)**
    a. PostgreSQL is being used for the database since its a stable & highly popular SQL database that's also open-source.
    b. The database will be preserved between the application states, and shouldn't be a part of the deployment script.
4. **SMTP Server**
    a. The SMTP Server will be used to send emails to the relay operators.
    b. Both the backend & the script can trigger emails. The credentials will be a part of the shared environment file, and hence can be used by both of them.
    c. The mail-server used for sending these emails could be built from scratch or a 3rd party mail-server could be utilised. The pros & cons for each of them are:
        i. <u>Self made mail-server:</u>
            1. It will provide us the functionality to send emails which are potentially free (only the server charges will be applicable).
            2. A mail-server requires port-25 to be unlocked, which is extremely difficult as most ISP's have blocked it to prevent spam.
            3. There will be an extensive effort in building out the mail-server, as improvements will be required for raising the mail score (preventing our emails from landing into the spam folder.)
        ii. <u>Third-party mail-server:</u>
            1. Third party mail servers will provide the stability & will abstract away the hardships of managing a mail-server.

2. A minor cost will be incurred upon sending emails as most of the companies charge per mail.
3. The prior implementations also relied upon a 3rd party mail-server whose credentials were used in the application for sending mails.

d. A third-party mail-server seems better in the short-term, and will be preferring that to start with. Though, post the GSoC period, an effort (major) can be spent on hosting a self-made mail-server as our project heavily uses email.

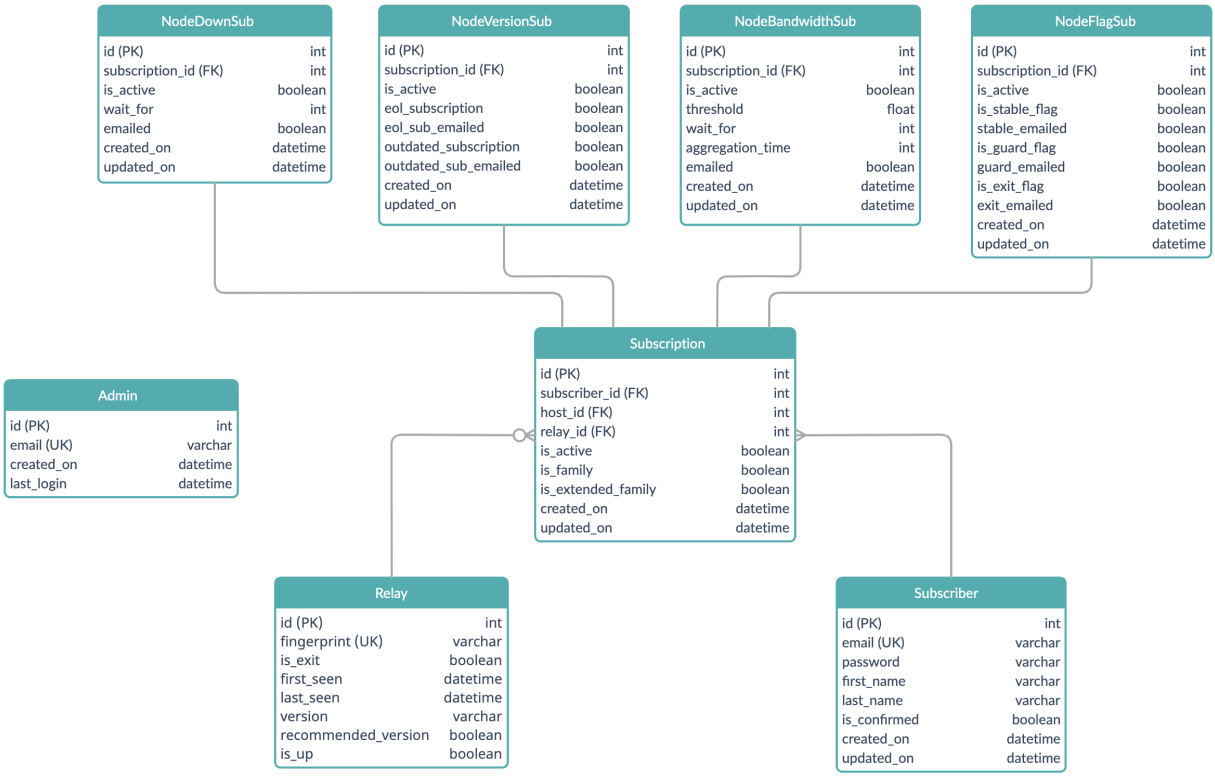
5. **Web-Interface**

a. The web-interface for the tor-weather can be created on a subdomain of tor if permitted, or on an entirely separate domain.

b. Relay operators can directly land here by accessing the link & logging in.

c. Relay operators can also land on the web-interface from the link in the emails (includes welcome, thank-you, or notification emails).

Database Design



A preliminary design for database was created for the application. This is subjective to changes upon further communication. A document explaining each column & its usage is available here.

Bi-Weekly Timeline

Community Bonding (May 20 - June 12)

1. Connect with the Tor Team and get onboarded on the relevant platforms.
2. Discuss the proposal with the team & look for possible improvements or changes.
3. Take relevant courses / read documentation of the new technologies being used.
4. Initiate the repository, database & setup the environments for crontab, smtp, etc.
5. Get better understanding of the working of the tor network.

Week - 1 & Week - 2 (June 13 - June 26)

1. Write relevant models for the API, database responses. Also define input & output parameters for the functions.
2. Develop user-flow including sign-in, sign-up, email-verification, authorisation, unsubscribe including basic frontend templates, backend routes & email-templates.

Week - 3 & Week - 4 (June 27 - July 10)

1. Initiate development of the python script to fetch data from onionoo & modify the data in the database.
2. Write email templates for the subscription notifications & get them integrated with the python script.

Week - 5 & Week - 6 (July 11 - July 24)

1. Complete the development of the python script & test its working with some mock-data on the mock-database.
2. Initiate the development for changing the preferences of the subscription services. This includes basic frontend templates & backend routes for the same.

Week - 7 & Week - 8 (July 25 - Aug 07)

1. Complete the development for changing the preferences of the subscription services & test its working with some mock-data on the mock-database.
2. Initiate the process of styling the frontend with the relevant design specifics as provided in the Tor Styleguide.

Week - 9 & Week - 10 (Aug 08 - Aug 21)

1. Complete the styling of the frontend for all the pages across the application. This includes designing & implementing the designs.
2. Initiate the process of writing basic unit tests for the functionality across the application. This includes the backend & the python scripts.

Week - 11 & Week- 12 (Aug 22 - Sept 04)

1. Complete the process of writing tests. Will discuss the consideration of writing end-to-end tests for the application with the time available.
2. Fix any bugs, edge-cases left across the application during the development process.
3. If time permits, develop the admin workflow for some high-level functionality like releasing a notification for all the relay operators incase a security vulnerability is found.

Final Week (Sept 05 - Sept 12)

1. Write documentation for all the pieces of the project including the backend, scripts & the database.
2. If time permits, automate the process of deployment using tools like Docker or Vagrant.

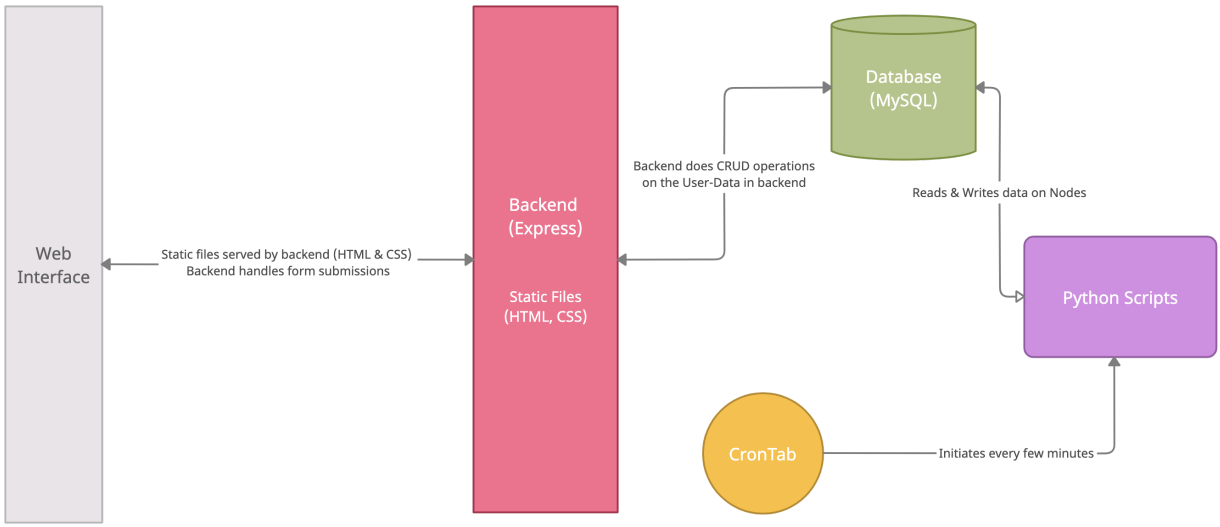
Post GSoC Period (After Sept 12)

1. Continue fixing bugs & adding features to the application.
2. Become a maintainer for the tor-weather project while also contributing to other projects at the organisation.

How can I be a good fit?

During my internship, I got a chance to develop an application from the ground-up that had a very similar structure to the proposed tor-weather app. I developed a News-Dashboard application which also followed a micro-service architecture approach.

There were multiple python scripts that fetched data from multiple news sources at regular intervals as managed by the crontab. These news-articles were compared with the one's already present in the database, and were ranked based on recency. Next, there was a backend written in Express that served static files (EJS & CSS) to the frontend. The app did support user-authentication, but at that time, it was mostly done with an external library.



For a personal project, I developed a module to handle the authentication, authorisation of users using JWT's which was mostly written from scratch. For the same project, I developed another module to handle the email flow from a personal email server.

In my current job, I am developing a design-system with multiple components ready for out-of-the-box consumption. A preview of which is available here. This helped me get a really good hand at styling the websites.

I feel confident that my past endeavours have helped me get a hands-on experience in almost every part of the proposed application, and I should be able to complete this project successfully within the GSoC timeline.

Commitments & Availability

This is a medium-sized project, and I aim to provide at-least 15hrs a week to this project. During the project period, my commitments includes a full-time job which takes around 40hrs per week. During the program, my availability for this project will be:

- Weekdays - 08:00PM - 12:00AM (GMT +5:30)
- Weekends - 11:00AM - 6:00PM (GMT +5:30)