Implement SPDX License Matching in Python

Basic Information

Name: Anshul Dutt Sharma

Github Id: anshuldutt21

Email Id: anshuldutt21@gmail.com

Skype Name: live:anshuldutt21

Phone Number: +91-9810928467

Location: Roorkee, Uttarakhand, India 🇮🇳

Timezone: IST(UTC + 0530)

Education: B.Tech 2nd year Electronics and Communication Engineering, Indian Institute of Technology Roorkee

Communication with Mentors

- Reachable anytime through **email, gitter, slack, contact number**, or a planned video-session.

Typical work hours

- UTC 0430 - 0730 hrs (IST 1000 - 1300 hrs)
- UTC 0930 - 1230 hrs (IST 1500 - 1800 hrs)
- UTC 1530 - 2030 hrs (IST 2100 - 0200 hrs)

Personal Background

I am Anshul Dutt Sharma, an undergraduate in Indian Institute Of Technology Roorkee pursuing Electronics and Communication Engineering in Uttarakhand, Roorkee. My love for computers and various computer languages began in the 10th standard when I learned the basics of C++ and was introduced to the world of web development when HTML was included in my curriculum. In my freshman year of college, I explored web development and worked on some projects which were enlightening learning experiences for me.

In my second year, I explored various Web Stacks like Python, ReactJs, Django, DjangoRestFramework, Docker, shell-scripting, etc which increased my interest in Development.

Also, I am an active member of the Information Management Group (IMG), IIT Roorkee which is a group of passionate developers and designers who constantly strive to lead in the innovation, development, and maintenance of advanced information technologies including computer systems, software and database systems in the institute. At IMG, I learned to work along with a product cycle with a team of approximately 50 members.

In my pursuit of acquiring development skills, I started contributing to the open-source community so that I can give back to the organizations whose products have been valuable assets to me.

Project Abstract

The Software Package Data Exchange (SPDX) specification is a standard format for communicating the components, licenses, and copyrights associated with a software package. SPDX Java Tools provide the Command Line interface for

performing a variety of functionalities using many tools like SPDX format converters, compare utilities, SPDX Viewer, Identifier and Generator. Apart from these, the SPDX tools are also helpful for comparing 2 license texts or matching texts with the SPDX license lists.

The SPDX tools implement all the above features in the java programming language which can be found in the tools. Java, though is very straightforward to use, write, compile, debug, and learn than alternative programming languages, different platforms, Users and application environments prefer a native python implementation primarily because Python is a high-level, interpreted dynamic programming language that focuses on code readability. The syntax in Python helps the programmers to do coding in fewer steps as compared to Java. The biggest advantage over Java is that it provides large standard libraries that include areas like string operations, File Parsing, Internet, web service tools, etc which are already scripted in python reducing the number of lines of code that need to be written. Also, requiring Java in a Python environment adds a lot of complexity and negatively impacts performance.

This project aims to implement as much of the SPDX License Matching Guidelines as practical in Python which would replace the current Java implementation of tools. A python implementation for SPDX tag/value and RDF parser, validator and handler is already done which can be found in tools-python. This project will mainly provide an interface to match a license text against a license template, any other license text against license texts present in the SPDX listed licenses which will return all the matching license IDs. Also, when there is not a match, it will provide a return value making it possible to describe where and why the license does not match.

There are certain inbuilt python libraries that I will be using in this project to implement these features and to make the code more efficient performance-wise and more readable. The project would be coded independently and would render a programmatic as well as a command-line interface. It will be able to be used by the Check License online tools as well.

Project Milestones

- Normalize the License Texts according to the License Matching guidelines.

- Provide an interface (programmatic) that takes 2 license texts as input and returns a boolean indicating if the 2 licenses match as per the license matching guidelines.
- Provide an interface (programmatic) that will check text against a license template using the license matching guidelines.
- Provide an interface (programmatic) which will check text and return all matching SPDX listed license ID's.
- Provide the command-line interface using shell scripts for the above features.
- Write tests to check the continuous functioning of the project and the performance of the project.
- Provide a return value to describe where and why the license text does not match.
- Document all the code written using the help of the sphinx tool.

BucketList

I'd like to improve this project in the following way if the above-mentioned tasks get completed before the end of the GSoC timeline.

- Add some examples of matching license texts in the project and a **'How To Use'** heading in the Readme file so that the user learns how to use the command-line interface and thoroughly follows the code.

Project Implementation

The SPDX License List Matching Guidelines provide guidelines to be used for the purposes of matching licenses and license exceptions against those found on the SPDX License List. Examples of how to apply some of the matching guidelines to a license or exception are provided via templates. Templates provide an easy format that is helpful in comparing and parsing license texts.
A template is composed of text with zero or more rules embedded in it. A rule is a variable section of a license wrapped between double angle brackets

"<< >>" and is composed of 4 fields - type, name, original and match. Each field is separated with a semicolon ";".

For example:

```
<<var;name=organizationClause3;original=the copyright
holder;match=.+>>
```

License Text needs to be first normalized according to the SPDX Matching guidelines before being parsed or compared to the other License Texts. The key function for working with files in Python which I will be using in the code is the open() function. The open() function returns a file object, which has a read() method for reading the contents of the file. By looping through the lines of the file, the whole file can be read, line by line.

```
f = open("demofile.txt", "r")
for x in f:
  print(x)
```

Normalize the license text

Below is the summary of the guidelines I will implement to normalize the license text.

- **Convert all the text to Lower Case Letters.**
  - This can be done using the .lower() inbuilt method in python.

- **Normalize the equivalent words**.
  - The words in each line of the text file available at https://github.com/spdx/license-list-XML/blob/master/equivalentwords.txt are considered equivalent and interchangeable.
  - I plan on replacing all of them using the replace inbuilt method in python.

- **Normalize the Whitespace.**
  - All whitespace should be treated as a single blank space including different spacing of words, line breaks or paragraphs.

- The regex for the white space is “\t” and “\r” and for the line breaks is “\n”. All of these can be replaced with a single space “ “ using the replace method in python.

❖ Normalize the Copyright Symbol.

- "©", "(c)", or "Copyright" need to be considered equivalent and interchangeable.
- I plan on replacing all of these with a single equivalent character.

❖ Normalize the Punctuations.

- Any hyphen, dash, en dash, em dash, or other variation should be considered equivalent and any variation of quotations (single, double, curly, etc.) should be considered equivalent.
- I plan on replacing the punctuations (‘,” ”,{},) with a single one using the replace method.

❖ Normalize the Code Comment Indicators.

- If the license text starts with comments indicated by indicators like /, /*, *, etc, the comment line is useless and should not have an impact in matching or comparing.
- Python has a built-in package called re, which can be used to work with Regular Expressions. Using the search function to match the regex with the first line, we can determine whether there is a comment and then normalize it. The matching regex would be somewhat like “^(\/)|(\/*)$” which would include more comment indicators.

❖ Normalize bullets and Numbering.

- If a line in the text starts with a bullet, number, letter, or some form of a list item we need to ignore the list item for matching purposes.
- This can be done again using the search function of the re package of python by matching the line with the regex, then removing the bullet or number using the sub-method of re.

❖ Normalize the Copyright Notice.

- ➢ If a line consists of copyright notices, it needs to be ignored and not included in the matching or comparing methods. Copyright notice consists of the following elements, for example: "2012 Copyright, John Doe. All rights reserved." or "(c) 2012 John Doe."
- ➢ This could be done in a similar way to the above case, only the regex would be a little complicated.

❖ **Normalize the License Title and the Text at the end of the License.**

- ➢ If a license Text comprises of some extraneous text at the end of the file such as, "END OF TERMS AND CONDITIONS," or an exhibit or appendix that includes an example or instructions on to how to apply the license to the code, it needs to be ignored and not included in the license matching or comparing.
- ➢ I plan on solving this issue in a way similar to the above two cases, matching the line with a regex and later using the sub-method to delete the line if it matches the regex.

The above normalizing techniques are used before comparing 2 license texts. There are some more techniques but those are applied for normalizing and parsing the license templates which I will implement later.

License Matching between 2 license Texts

After implementing the above normalization techniques, I plan on providing an interface that will take 2 license texts as inputs and return a boolean value indicating if the 2 licenses match as per the license matching guidelines.

For this, I intend to use the filecmp module which has inbuilt functions to compare files and directories, with various optional time/correctness trade-offs.

The cmp function will compare the 2 normalized input text files named *f1* and *f2*, and return True if they seem equal, False otherwise. No external programs are called from this function, giving it portability and efficiency. This function also uses a cache for past comparisons and the results, with cache entries invalidated if the os.stat() information for the file changes making it efficient and faster.

```
>>> import filecmp
>>> filecmp.cmp('normalized_file1.txt', 'normalized_file1.txt')
True
>>> filecmp.cmp('normlized_file1.txt', 'normalized_file2.txt')
False
```

Implementation of this module will return a boolean value True if the License Text files match and False otherwise.

License Matching between A License Text And a License Template

We basically need to convert the License Template to the License Text format and implement the above matching between 2 License texts to get the answer, except for some changes that need to be kept in mind while comparing.

In a license template, there are 2 important things that need to be changed in order to convert to the License text format.

- **Omitable Text**

  Some licenses have text that can simply be ignored. The intent here is to avoid the inclusion of certain text that is superfluous or irrelevant in regards to the substantive license text resulting in a non-match where the license is otherwise an exact match. In these cases, there should be a positive license match. The omitable text is indicated in the template with markup.

  The substrings in the template between **<beginOptional>** and **<endOptional>** are considered to be omitable and must be either a match if the substring indicated is present and matches OR the substring is missing altogether.

  In order to solve this issue, I plan to use the python package re, which can be used to work with Regular Expressions. The Re.findall() module is used when we want to iterate over the lines of the file, it will return a list of all the matches in a single step.

  The regex will somewhat be **\<beginOptional\>.*\<endOptional\>.**

The findall module will iterate over the template and find all the text that matches the regex provided and will replace the omitable text (between the tags) in the template with the regex - **{the text portion between the optional tags}?**.

The **?** in the regex language means that the text before it can be repeated either once or none at all adhering to the meaning of the omitable text. In this way, the position of the tags won't be lost as well.

This normalized template can now be separately matched with the license text provided using the method of matching 2 license texts as described earlier. The pseudo-code for the implementation of the omitable text is provided below.

```
import re;
omitable_text =
re.findall("\<beginOptional\>.*\<endOptional\>",
template_file);
#remove "<beginOptional>" and "<endOptional>" from the
substrings of omitable_text.

for x in omitable_text:
   x.replace({text_between_tags}),{text_between_tags}?)
      #omitable text matches.
      #now compare the 2 license texts.

Compare_2_license_text(original_text_file,license_template_file
)
```

❖ Replaceable Text

Some licenses include text that refers to the specific copyright holder or author, yet the rest of the license is exactly the same as a generic version. The intent here is to avoid the inclusion of a specific name in one part of the license resulting in a non-match where the license is otherwise an exact match.

If the substring lies between <<var;name="**copyright**"..>>, it is considered to be replaceable and won't be used in matching licenses.

A somewhat similar approach can be used for converting this format to license text. The findall module in the re package can be used to identify strings that are replaceable. The regex to identify the replaceable text would be **\<\<var;name=\'copyright'\>\>.** All the matched texts will be stored in a list and whatever the text is present after **{original='....'}** will be replaced with **.*** which is the regex for any alphanumeric character any number of times. Basically, it means that the text present between the replaceable tags can be replaced with any character or string. The pseudocode for the implementation of replaceable text is provided below.

```
import re;
replaceable_text = re.findall("<<var;name=\'copyright;"),
template_file);
# remove the markup rule from the substrings of
  replaceable_text.

for x in replaceable_text:
  x.replace({ReplaceableTextAfteroriginal},(.*));
      # text after original='..' in the markup rule of template
      # is replaced with the regex .* implying it can be
      # substituted with any string.

compare_2_license_text(original_license_file,License_template)
# Pass the template for matching and comparing with the
# original license text file.
```

❖ Embedded Tags

Sometimes, there are embedded tags also present in the template. For example Optional text with var text embedded within it (e.g. <beginOptional>Copyright<var...owner><endOptional>) or optional tags embedded within optional tags.

For checking those what we can do instead of using the normal findall function as described above in the pseudo-code, is start at the deepest tag level (the tag which doesn't have any embedded tags) and replace it with the regex, then move on to the higher levels and go on replacing them with the regex. The regex for the deepest tag will somewhat be **<beginOptional>(?:.(?!<Optional>)|(?!<var>)*?<endOptional>.** This will only match the deepest tags in the license template and replace them. We can iterate the findall function to match the deepest tags in the template against the license text until there are none left.

- **Bulleted Text**

  Some license templates have rules written along with the text which starts with numbers or bullets creating a lot of confusion by unmatching when it is actually a match. If a substring lies between <<var;name="**bullet**"...>>, the markup will be removed and only the rule will remain which can be matched against the license text.

After following the above methods, the license template converts to a license text format. We can now pass the license template (already converted to license text format) and the license text file to the implementation of comparing 2 license fields which will involve normalization of both the texts and then using the filecmp module (The pseudo-code has this step written at the end as shown above).

Match the License Text with the SPDX listed License IDs.

After implementing the interface for matching 2 license texts and for checking a license text with a license template as mentioned above, in this step, we need to load all the SPDX listed License information into a list and check each of the listed license templates individually with the license text provided as the input. The list of licenses is loaded from http://spdx.org/licenses/licenses.json and each individual license template is downloaded from http://spdx.org/licenses/[licenseid].json where the [licenseid] is the ID of the license which will be used for license matching. Python has a **json** module which makes it easy to parse JSON strings and files

containing JSON objects. The pseudocode for implementing this feature is presented below.

```
import json;
# license_list_json is the file downloaded from license_list
containing the list of all Licenses.

License_list = json.loads(license_list_json);
Matched_licenses = []; # stores the matched licenses ids.

for i in range(len(License_list)):
   License_id = License_list["Licenses"][i]["LicenseId"];
   # License_id contains a list of all license ids.

   If license_matches(input_file, {License_id}.template.txt):
      # All the License template files will be stored as {Name
      # of License id}.template.txt.

      Matched_licenses += {License_id};
```

So, all the Matched licenses ids will be stored in the list Matched_licenses and will be rendered in the output.

Provide a Return value explaining where the license doesn't match.

For implementing this feature, I intend to use the python module difflib. This module provides classes and functions for comparing sequences. It can be used for comparing files and can produce different information in various formats, including HTML and context and unified diffs.

Difflib has many subclasses, one being the Differ class for comparing sequences of lines of text, and producing human-readable differences or deltas. Differ uses the SequenceMatcher to compare sequences of lines, and to compare sequences of characters within similar lines.
The **unified_diff** method compares *a* and *b* (lists of strings) and returns the difference in unified diff format. The changes are shown in an inline style

(instead of separate before/after blocks). The number of context lines is set by *n* which defaults to three. Below is the syntax for the output file format.

Code	Meaning
`'- '`	line unique to sequence 1
`'+ '`	line unique to sequence 2
`'  '`	line common to both sequences
`'? '`	line not present in either input sequence

The method will take in input the 2 normalized license text files and output a new file which will show the word differences between the 2 input files in each line. These differences can be rendered after further parsing the output file. (removing unwanted texts, alphanumeric characters,etc.)

```
import difflib

with open('normalized_file1') as f1:
    f1_text = f1.read()
with open('normalized_file2') as f2:
    f2_text = f2.read()

# Find and print the diff:
for line in difflib.unified_diff(f1_text, f2_text,
fromfile='normalized_file1', tofile='normalized_file2',
lineterm=''):

    print line
    line.sub('^\@.*$');
# Further parse the output file as it consists of other details
such as the line numbers,etc.
```

Provide a Command-Line interface (CLI) to the above features

I also plan to provide a command-line interface apart from the programmatic interface for the above features by writing **shell scripts**. Shell scripts are an efficient and simple way to provide the CLI. They can take inputs as files from the user and execute them in the shell and write the output in a different file. There will be 3 scripts, one for the matching between 2 license texts, one for the matching of a license text against a template and the other for returning the spdx listed license IDs. A format for a shell script taking 2 files as input and writing the output to an output file is provided below.

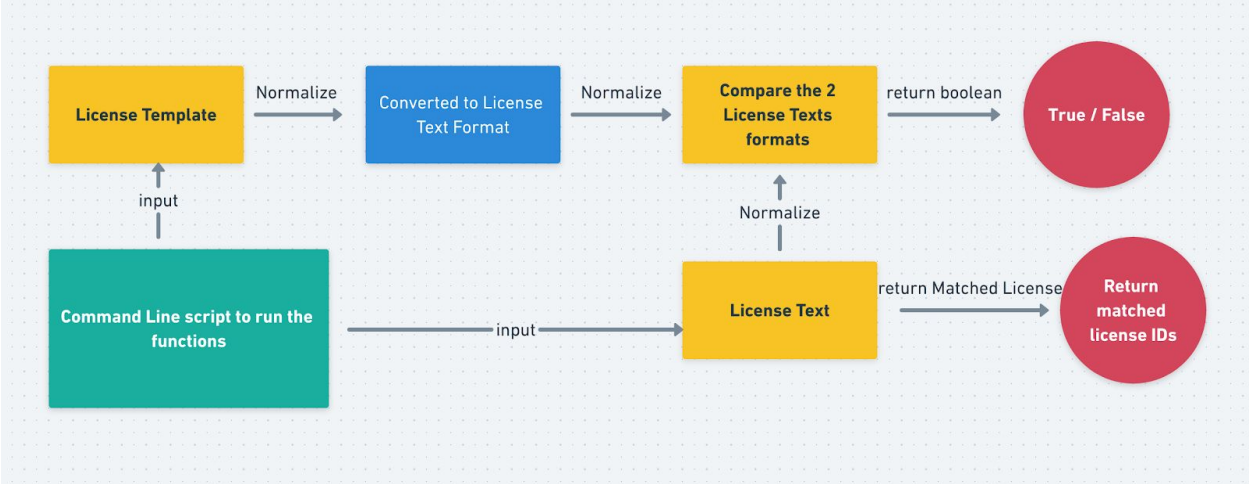
```
#!/bin/bash
cat input_file1.txt input_file2.txt | python
Match2LicenseTexts.py > output_file.txt.
```

I will be writing the scripts in a similar fashion.

There are certain things that have been kept in mind while drafting the implementation of the proposal.

- ❖ I will be writing tests side by side with the code for each of the implemented functionalities to ensure the coverage of all cases for which I will be using unittest which is a unit testing python framework.
- ❖ I will also be documenting my work side by side with the code for which I plan to take the help of the sphinx python framework.

Below is the basic proposed outline of the project.



Project Timeline

I. 1st July- 5th July

Read more from the SPDX documentation, appendix and revisit the tutorial by david-a-wheeler. Also, Go through the xmi format of the SPDX spec model to understand it better.

Go through the concepts of Regular Expressions and learn how to apply them in python using the re module. Study the different methods of the module.

Relevant Reading

- https://www.w3schools.com/python/python_regex.asp

II. 6th July - 10th July

Normalize the license text following the steps mentioned in the implementation.

Relevant Reading

- https://www.afterhoursprogramming.com/tutorial/python/reading-files/.

III. 11th July - 20th July

Implement License Matching between 2 license texts. Provide a proper command-line interface for executing the feature using the filecmp module.

IV. 21st July - 25th July (Milestone 1 evaluation)

- Document all the code written for the feature.
- Check for performance and any bugs.
- Get it ready for the Phase 1 evaluations.

V. 26th July - 28th July

Study the re module and its methods findall and search which need to be used for the implementation.

Relevant Reading

- https://docs.python.org/3/library/re.html.

VI. 29th July - 12th August

Implement the normalization of the Omitable text. Also, check some other factors which I might have missed and might need to be normalized later.

VII. 13th August - 15th August(Milestone 2 evaluation)

- Document all the code written for the feature.
- Check for performance and any bugs.
- Get it ready for the milestone 2 evaluations.

VIII. 16th August- 18th August

Study about the JSON package, it's implementation and its applications.

Relevant Reading

- https://www.w3schools.com/python/python_json.asp

IX. 19th August - 31st August

- Implement the normalization of the bulleted text, embedded text and the replaceable text.
- Implement the License matching between License Text and all the listed License IDs.
- Check for performance tuning and try to enhance it as much as possible.

X. 1st September - 5th September

- Study the difflib python library, it's functions and methods as well as its applications.

XI. 6th September - 10th September (Milestone 3)

- Write unit tests and format the code to make it more readable.
- Document all the code written for the feature.
- Check for performance and any bugs.
- Get it ready for the milestone 3 evaluations.

XII. 11th September - 16th September

Provide a return value explaining where the license doesn't match and return all the possible values where the license could have matched.

XIII. 17th September - 24th September

Implement the command-line interface for all the above features.

XIV. 25th September - 30th September (Milestone 4)

- Write unit tests and format the code to make it more readable.
- Document all the code written for the feature.
- Check for performance and any bugs.
- Get it ready for the final evaluations.

Application Questions

About SPDX

When did you first hear about SPDX?

I wanted to contribute to an organization suiting what I wanted to try my hands on. I was scrolling through the previous year's organization list where I stumbled across SPDX. I read the documentation and I really liked the work and want to learn from it as much as possible.

Describe your participation in our community (bug fixes, communication via mailing lists, IRC and Gitter)?

I have had a keen interest in software development since my freshman year. In my pursuit of exploring this field and enhancing my knowledge, I became a contributor to the open-source community. My open-source contributions in different repositories are listed below.

Spdx-online-tools

Pull Requests

#150 : Add new file version information to keep record of the versions. Render the version information in the about page of spdx.

#152 : Make a migrations file in the app folder to avoid using the command makemigrations in the READme file.

#149 : Solve an error in the app.models file of License Request.

#146 : add links to OSI approved list. Also, fix a typo.

#143 : Change the file extension to .xls in convert view for spreadsheets.

#154 : Solve Archive Requests issue.

Issues

#148 : Submit a new license request doesn't work since the License Request is not imported.

#147 : Add the command python src/manage.py makemigrations in the docs since the migrations for the app doesn't work unless we do makemigrations for it.

#153 : Improve testing to check for edge cases.

#144 : Running the server shows an error no module named secret in settings.py file.

Tools-Python

Pull Requests

#129 : Fix write_tv file error in examples directory.

#127 : Correct typo in a document file.

Issues

#128 : example file write_tv showing invalid document #128

Apart from SPDX, I have also contributed to other organizations in the past. Below are the links to the contributions.

Sugarlabs

Pull Requests

#577 : Add the window resize function in the sugarizer activity.

#569 : Use media queries to remove the overriding of text.

Issues

#580 : alert "user already exists" should come earlier.

#579 : Improve Sugarizer spiral in the home view.

My other contributions to open source can be found in my GitHub profile anshuldutt21. Apart from GSoC, I also participated and successfully completed the **HacktoberFest 2019**. I have communicated with SPDX through mailing lists as well as the Gitter channel. I also attended a web conference call hosted by the mentor and would try to involve as much as possible with the community in the future, and even after the program ends.

In less than 3 sentences, why should we pick YOU?

- From what I know, I am hardworking and have the patience to commit to my work. Even if I get stuck somewhere, I try my best to solve the problem on my own before asking out for help.

- At the Information Management Group (abbreviated as IMG), I learned to work along with a product cycle with a team of 50 members and so I consider myself experienced with the ability to work with the community in integrating results with other projects.

About Me

Describe any plans you have for the summer in addition to GSoC (classes, thesis, job, vacation, etc.)

I can easily devote 50-55 hours per week until my college reopens and 45-50 hours per week after that. I am also free on weekends and I intend to complete most of my work before my college reopens.
Other than this project, I have no other commitments during the summer vacations. I shall keep my status posted to all the community members on a weekly basis and maintain transparency in this project.

Describe any work on other open-source projects.

Omniport is an open-source portal for educational institutes, designed from the ground up to be extensible and customizable. Here are some of the projects I worked on which are a part of omniport.

Electorate

- Electorate is a platform to interact with candidates contesting for institute candidature and Bhawan candidature. We can view the manifestos and ask queries pertaining to the same. We can also start a comment's thread at a question asked by a person.
- My work was mainly at the backend of the platform which is written with the Django Rest Framework.
- Tech Stack - ReactJs, Django Rest Framework.

I also participated and successfully completed the **HacktoberFest 2019**.

What programming projects have you completed?

Info-Portal

- It is a web-application designed where the user can login and make his profile page by entering his information and his profile picture. He can also chat with people online.
- The login page is designed to check information on the frontend as well as on the backend.
- Tech Stack - Regex, Ajax, PHP, websockets, etc.

IMG Sched

- It is a real-time web application designed to schedule the meetings and lectures of IMG (Information Management Group) within IIT Roorkee.
- Tech Stack - Django, Django Rest Framework, Django Channels for web sockets, React.js, Oauth for authentication.
- The link for the github repo is IMGSched.

Private Key Encryption

- It is a project designed for transmitting secured information with the private key generated using random mouse movements and communicated using the Diffie-Hellman Algorithm.
- I also wrote a blog where I explained thoroughly about this project, the link for which can be found out here.

What are your favorite programming tools (editor, etc.)?

Computer Languages	JavaScript, Java, Android Studio, Regex, Python, PHP, CSS, SQL, HTML, Bash, C/C++
Web Frameworks	React, Redux, Django, Django REST, Django Channels, Web sockets
Utilities	Git, Vim, VS Code, Linux shell utilities

Apart from the technologies listed above, I have some knowledge of object-oriented programming.