

---

# Google Summer of Code 2019

## Project Proposal

### Clio -Software Components and IP Management System

---

#### Basic Details:

#### Personal Information

- Name: Shivanshu Raj Shrivastava
- Primary Email: [shivanshu1333@gmail.com](mailto:shivanshu1333@gmail.com)
- Alternative Email: [sshrivastava@ec.iitr.ac.in](mailto:sshrivastava@ec.iitr.ac.in)
- GitHub: [shivanshu1333](https://github.com/shivanshu1333)
- Zulip/Slack/Gitter nickname: shivanshu1333
- First language: Hindi, proficient in English
- Time zone: Indian Standard Time (GMT +5:30)
- Contact: +91 9425646127
- GPG ID: 8BAC8D70
- GPG Fingerprints: 4C9C 2794 58F8 5AE2 8AC8 999D F8D7 0AFF 8BAC 8D70
- Blog: [shivanshu1333.github.io](https://shivanshu1333.github.io)

#### University and Current Enrollment

- University: Indian Institute of Technology, Roorkee
- Major: Electronics and Communication Engineering (Batch of 2020)

#### Meeting with mentors

- Accessible anytime through email, Zulip chat or a well-planned video session (if required).
- UTC 5:30 AM to UTC 7:30 PM [IST 11:00 AM to IST 1:00 AM].
- Typical working hours (UTC):
  - 3:30AM to 7:30AM, 9:30AM to 3:30PM, 4:30PM to 7:30PM

## Share links, if any, of your previous work on open source projects?

In my freshman year, I was primarily interested in electronics and worked on various hands-on projects of my own (mentioned later on). It was where I got my start and since I have always been curious to explore new fields in tech, I got introduced to Github and Web development later on. Since then, I kept myself involved in developing and learning about software, most of my time goes in reading and writing software.

Trying out new technologies by taking up various projects and participating in **contests/hackathons** became my hobby in my first year itself. I have been contributing to open source since about two months now - looking over to repositories of the products I use/come across, trying to contribute back to the organization whose product has been an asset to me.

I have been able to put up a few projects on my Github [profile](#) ever since I came to know about it.

- **[LED-GRID:](#)**
  - Developed a Led Grid Display 6\*24 which can display characters, symbols etc just like a normal display but with low resolution (144 leds acting as pixel).
  - Worked with **Arduino** to control Led Grid panel made by me.
- **[Interactive-Led-Display:](#)**
  - Developed a Trex game to play over a Led Grid using Arduino as microcontroller.
- **[Humanoid Bot:](#)**
  - Worked with my college robotics team ([marsitr](#)) to make lower body of a Humanoid.
- **[Their real time, GIS based tracking for water Transportation in India:](#)**
  - This project was done under a Hackathon organised by Indian government.
  - An android app was developed to track level of water in water tankers using various sensors and Arduino as microcontroller (see link for more details).
- **[Smart-Spell:](#)**
  - OCR to read text from books from an **Android** App.
- **[ToDo App:](#)**
  - A basic ToDo application built as a beginner level project to learn the basics of web development.
  - Used **Flask**, **Vertabelo**, **SQLAlchemy** and **Bootstrap** for development.
  - Used **Heroku** for deployment of the completed project.
- **[C.R.U.D.M.E](#)** (Create.Read.Update.Delete for Managing Employees):
  - A simple CRUD employee management web application.
  - Flask was used as the micro web framework.
  - **MySQL** was used as the relational database to make tables.
  - Learned to make complex models and one-to-many/many-to-many database structures.
  - Learnt about ORM and SQL

## Your Motivation:

### What is your motivation to take part in Google Summer of Code?

Being a software enthusiast I always loved to think and develop ideas of my own. My interest in web development introduced me to git and Github which, for me, ultimately paved a path towards open-source.

Making web pages and databases, developing games and then pushing those to Github to increase the count of my repositories and commits always fascinated me.

When I was in my second year, I heard about Google Summer of Code from my seniors as many of them were contributing to different organizations. They shared their experience that how good it feels when you get your first **pull request** merged in a big organization and that your work directly affects the open source community and a large number of people using that software.

On a personal note, I think that GSoC is the best way to learn to work collaboratively in large teams. I feel that the three-month duration of GSoC will also allow me to apply my current skills as well as acquire a new set of them under the guidance of some really experienced people in the world of open source. The remote working policy of GSoC makes it the best program a college student can get involved in during their summer holidays for a great learning experience.

## Why did you choose GFOSS?

### Codebase:

Among all the programming languages and frameworks, I am quite comfortable with **HTML/CSS/javascript** and **python**. So I decided to search for organizations having their code bases in these languages from GSoC '18 organization list and luckily found GFOSS as one of those.

So, I started exploring projects of GFOSS from December itself. I found Clio which aligned to my skill set and seemed to be a pretty decent idea as well.

The code base of GFOSS seems to be well organized but at the same time, I also feel that a lot needs to be done to make this project a great success. I surely feel that I can leverage my technical skills to make this happen.

### Great support:

Making my first contribution to Clio couldn't have been easier, thanks to the awesome people maintaining it, particularly **@zvr** and **@gopuvenkat**. It happened quite often that I got stuck in some problem and put a lot of time/effort solving it, and if it hadn't been for the amazing support and quick responses, it would have been a difficult journey for me till now.

## Why do you want to work on this particular project?

### Language and Framework:

This project entirely involves working with **HTML/CSS, python, MySQL** and **Flask** and a majority of my past contributions have been in same tech stack, which made me quite comfortable with the source code and working with the project.

### Learning opportunities:

This project has lots of learning opportunities for me, it will help me to enhance my knowledge of python language, allow me to work with and learn more about the Flask framework and also provide me a platform to showcase my designing skills.

Moreover, the best part will be getting to collaborate with large teams which will teach me skills like learning to use Git like a pro, just to name a few.

## What are your expectations from us during and after successful completion of the program?

During the program, I expect GFOSS to continue providing me guidance and support similar to what they have been doing till now. Being my first big open source project, the stakes are high that I do things the wrong way or make mistakes, so I expect my mentors to correct me.

After the program, I am willing to keep contributing to GFOSS. Thus, I expect support and collaboration from the community.

Also, I think that it would be a great idea to hold a meetup of all GSoC students and mentors upon successful completion. I personally feel that such activity would make the bond between the students and the organization even stronger.

## Project Details:

### What are you making?

In short, Clio is a typical software project often reuses hundreds of third-party packages. License and origin information is not always easy to find and not normalized. It is a web-based system to manage data on software components and their relations.

Nowadays every piece of software is including and using many other software components with each one coming with their own license. The goal of this project is to build a web system to be able to fetch/take input from the user and maintain this information.

Thus, we are planning to improve the basic implementation of Clio.

### Following is the project task checklist in chronological order:

#### Milestones:

##### A. UI improvements:

- a. Improve landing page of Clio
  - i. Design logo and add FavIcon
- b. Improve license page
  - i. Better tables, efficiently handling lots of data
  - ii. Better search and result presentation
- c. Improve components page
  - i. Better tables, efficiently handling lots of data
  - ii. Better search and result presentation
- d. Improve product page
  - i. Better tables, efficiently handling lots of data
  - ii. Better search and result presentation
- e. Improve Sign in page
- f. Add "Upload SPDX document" field
- g. Add "Show report" field in components page to know about Complex components

##### B. Backend and few Frontend changes in create/update license, components and product page

### C. Content Management

- a. SPDX files for components
- b. Better initial data seeding, incorporating data for well-known components
- c. Add functionality to warn users while adding duplicate entries

### D. Report Generation

### E. External Service Integration

### F. Improve data handling

### G. Fix Existing Issues

\* Major UI improvements will be done explicitly, Minor UI improvements will be done as per requirement.

## Implementation Details:

### A. UI improvements:

**AIM:** The basic aim of UI/UX improvement involves faster and efficient fetching of data along with minimal use of frontend tools for a faster website. The aim is to optimize and modify the current code base of Clio to handle huge data efficiently.

**Note:** It also involves improving the app's general UI/UX by making it optimized for tablets and redesign parts of Clio by following material guidelines. User Experience can be improved by making some important changes in the design flow.

#### a). Improve landing page of Clio

##### Current Problems:

- The main page of Clio takes up a lot of space without delivering exactly what is required.
- Some basic functionality (like **header, footer, and favicon**, etc) are also absent which in general increase the user experience.
- The current setup is not responsive.


##### Solution:

- Redesign the landing page from scratch following **Material Guidelines**.
- Design logo for Clio and also use it as a Favicon.
- Make the landing page responsive to various devices.
- Add header and footer to every page having essential quick links.






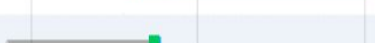

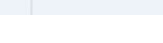
#### b,c,d). Improve License, Components, Product page

##### Problem:

- Basic functionality (displaying the desired information in the table) has been implemented but the User experience is not as desired.
- Some static files (Eg. semantic.min.css size ~500kB) usually takes a lot of time to load which adds a lot of latency (See below image for insights).

|   |                                 |                      |                  |   |
|---|---------------------------------|----------------------|------------------|---|
|   | semantic.min.css<br>/static/css | 2 ( 537 KB<br>537 KB | 9.53 s<br>1.44 s |  |
|  | jquery-1.12.4.js<br>/static/js  | 2 ( 287 KB<br>287 KB | 9.22 s<br>2.36 s |  |
|  | semantic.min.js<br>/static/js   | 2 ( 276 KB<br>275 KB | 4.09 s<br>1.95 s |  |
|  | base.js<br>/static/js           | 2 ( 483 B<br>233 B   | 5.44 s<br>5.43 s |  |



- Another problem is the lack of proper functioning of **pagination**. Here's how it works:
  - All the entries available in the database corresponding to a component/license/product query are fetched and sent to the client.
  - Next, the entries are paginated in the frontend data table which again takes a lot of time to execute owing to a large number of entries.
- Static files associated with **Export-Pdf option** also takes a lot of time and it loads on opening the webpage where it might not be required (See below image for insights).













|  |                                    |                        |                    |  |
|--|------------------------------------|------------------------|--------------------|--|
|   | jszip.min.js<br>/static/js         | 2 ( 99.8 KB<br>99.6 KB | 8.57 s<br>6.83 s   |   |
|   | pdfmake.min.js<br>/static/js       | 2 ( 948 KB<br>948 KB   | 27.41 s<br>10.52 s |   |
|   | vfs_fonts.js<br>/static/js         | 2 ( 850 KB<br>850 KB   | 22.29 s<br>9.90 s  |   |
|  | buttons.html5.min.js<br>/static/js | 2 ( 23.7 KB<br>23.4 KB | 9.19 s<br>9.19 s   |  |

- Show better search results.

#### Solution:

- Current setup is using **Bootstrap** to render data. UX can be improved by displaying more information on same pages by designing compact tables. Modify the code base for better Search and Result Presentation.
- Improve the pagination algorithm by using **flask's in-built pagination functions** to render server-side data. Also, functionality to set preferences for the number of entries per page can be added .
- On the client side, there is a need to more efficiently use the jquery data-tables to render entity tables to improve the user interaction part.

| Name  |            | Size                 | Time             | Waterfall   |
|---|------------|----------------------|------------------|---|
|  | component/ | 2 ( 286 KB<br>4.1 MB | 9.78 s<br>8.04 s |  |

|   |  |                    |                  |   |
|---|--|--------------------|------------------|---|
|   | dataTables.semanticui.min.css<br>/static/css | 2.8 KB<br>2.6 KB   | 2.37 s<br>2.37 s |   |
|  | component.css<br>/static/css                 | 447 B<br>211 B     | 1.61 s<br>1.61 s |  |
|  | buttons.semanticui.min.css<br>/static/css    | 3.2 KB<br>2.9 KB   | 2.37 s<br>2.37 s |  |
|  | iquery.dataTables.min.js<br>/static/js       | 80.2 KB<br>80.0 KB | 7.96 s<br>5.40 s |  |
|  | dataTables.semanticui.min.js<br>/static/js   | 2.5 KB<br>2.2 KB   | 4.27 s<br>4.27 s |  |
|  | dataTables.buttons.min.js<br>/static/js      | 18.1 KB<br>17.8 KB | 5.42 s<br>5.41 s |  |

Implementation details of pagination are as follows:

1. We can replace the `all()` call that terminates the query for an "entity" with: **`entity.objects.paginate(1, 20, False).items`**
2. The `paginate` method can be called on any query object from **Flask-SQLAlchemy**. It takes three arguments:
  - a. the page number, starting from 1
  - b. the number of items per page
  - c. an error flag. If **`True`**, when an out of range page is requested, a **404** error will be automatically returned to the client. If **`False`**, an empty list will be returned for out of range pages.
3. Next, I need to decide how the page number is going to be incorporated into application URLs. A fairly common way is to use a ***query string*** argument to specify an optional page number, defaulting to page 1 if it is not given. Here are some example URLs that show how I'm going to implement this:
  - a. Page 1, implicit: **`http://localhost:8000/components`**
  - b. Page 1, explicit: **`http://localhost:8000/components?page=1`**
  - c. Page 3: **`http://localhost:8000/components?page=3`**

```
def explore():
    page = request.args.get('page', 1, type=int)
    entities = Entity.query.paginate(page, app.config['ENTITY_PER_PAGE'], False)
    next_url = url_for('explore', page=posts.next_num) if posts.has_next else None
    prev_url = url_for('explore', page=posts.prev_num) if posts.has_prev else None
    return render_template(
        "index.html",
        title='Explore',
        entities=entities.items,
        next_url=next_url,
        prev_url=prev_url
    )
```

- Currently search algorithm of Component, License or Product page works very well but **does not highlight the search results**, it just updates the data table according to search items. It is desired to highlight the search results.

#### e). Improve Sign In page

##### Problem:

- The current sign-in page does not follow registration form usability guidelines and opens in full page.
- It does not provide **“Show Password”** field leading to very bad user experience.
- There are no alert messages except those of wrong credentials (like caps lock is on, etc).
- Does not provide any field to remember user preferences (save my credentials).

##### Solution:

- An optimized user-friendly Sign-In will be designed following registration form usability guidelines and essential fields like **“Show Password”** will be added.
- Integration of mandatory alert messages.
- Also in the demo mockup, I added **“Show Password”** field and **“Remember Me”** option. **The source code for this demo can be found in my repo [here](#).** (See the mockup below for insights)

- Now, to remember user preferences we have to add **Server-side Session** to Clio. I have to load the configuration of choice and then create the Session object by passing it the application. ***flask.ext.session.Session*** provides different classes to efficiently perform this task.

#### f). Add Upload SPDX document field in Add Component page.

**Problem:** In add component page, it is desirable to add a field through which the user can upload corresponding SPDX document for that particular Component.

##### Solution:

- We will begin with **adding a new database migration** for a new file field to hold the SPDX doc file. The field will be optional to avoid conflict.



- Finally, the field will be added to Add-Component form to incorporate this functionality (later on it may be extended to accept any format by choosing from the drop-down menu, but the initial setup will accept only one format).

#### g). Add “Show report” field in components page to know about Complex components

**Problem:** Currently clio stores data about all type of components in its database but does not provide a feature to produce a report about Components.

**Solution:** A new field is to be designed to display information about Complex components. The backend of this is weakly implemented and needs to be improved for better data management which will be done in **Milestone D**.

### B.) Backend and few Frontend changes in create/update license, components and product page

#### **Problem:**

- Major Problem is the large loading time of Create Component page which is due to the following reasons:
  - All the component and licenses to be displayed in the dropdown menu are fetched at the time of loading the form template.
  - The component and license data turn out to be massive which adds to latency.
- Also, since there are a lot of components and licenses present in dB to choose, a very long dropdown will be created which makes it hard for the user to select one entry, in other words, constitutes to a bad UI.

#### **Solution:**

- Changing the way the component and license form field appears:
  - We wish to which greatly improves the UI/UX.

- User will start typing first letters of the Component/License after which a list of Component/License containing those words will be listed in a dropdown.
- This way we can get rid of long drop-down and narrow down the choices the user wants to select.

A screenshot of a web form. At the top is a dropdown menu with a dashed line and a downward arrow. Below it is a text input field. The input field has a blue border and a blue background. Below the input field, a dropdown list is visible with two items: "Mine !" and "No owner !".

A screenshot of a web form, similar to the one on the left. The text input field contains the text "New item". Below the input field, a dropdown list is visible with one item: "Create 'New item'".

**Implementation details for the autocomplete feature are as follows:**

- We will begin by replacing the form field with an autocomplete field which will be a combination of a text field and a dropdown field.
- Next, an **onKeyDown()** event will be attached to the text area which will fire an **ajax request** to fetch the list of components according to the search term.
- The corresponding function in search will do a **full-text-search** to find an exhaustive list of components and license and sends it as the response
- Finally, the success part of the ajax request populates the form upon receiving the response and makes it available for the user to select.

## **C. Content Management**

### **a). SPDX files for components**

**Problem:** In the current setup, the user can provide various information about a component upon creating it but there is no functionality to incorporate importing of SPDX document.

**Solution:**

- Currently, different fields like **Component Name, Version, License Expression, Created By, Origin Website, Source URL, External Link, Publication Date and Add existing Component** (for complex components) are present.
- Apart from these a very important and usable content is SPDX document of that particular component. A field is required in the UI to incorporate this functionality. This will be done by completing Milestone **(f)** of **A**.
- Now, this data needs to be saved in the database and the corresponding tag for the component needs to be attached with it, so that in future the complete Component info. along with the SPDX data can be retrieved.

## b). Better initial data seeding, incorporating data for well-known components.

**Problem:** Currently there is no method for initially seeding of the database, Components data is arbitrarily populated by Debian packages with no to zero information about **Publication Date**, **Origin**, **Source URL**, **External Link**, **Associated Components** and **License Expression**.

**Note:** It is very important and essential to initially import information about some well-known components and Display it in components page.

### Solution:

- Import initial data from <https://clearlydefined.io/workspace> (open source) which contains information about approximately 1331 Components and populate it to MySQL database.
- Display all the necessary fields in the components page.
- Efficiently handle MySQL database to effectively display the contents of Components.
- Do data pagination to reduce the response time of the website.

**Example:** Approach described below was used to populate the database directly from the latest release of SPDX license list, a similar approach can be used to populate the database with components list.

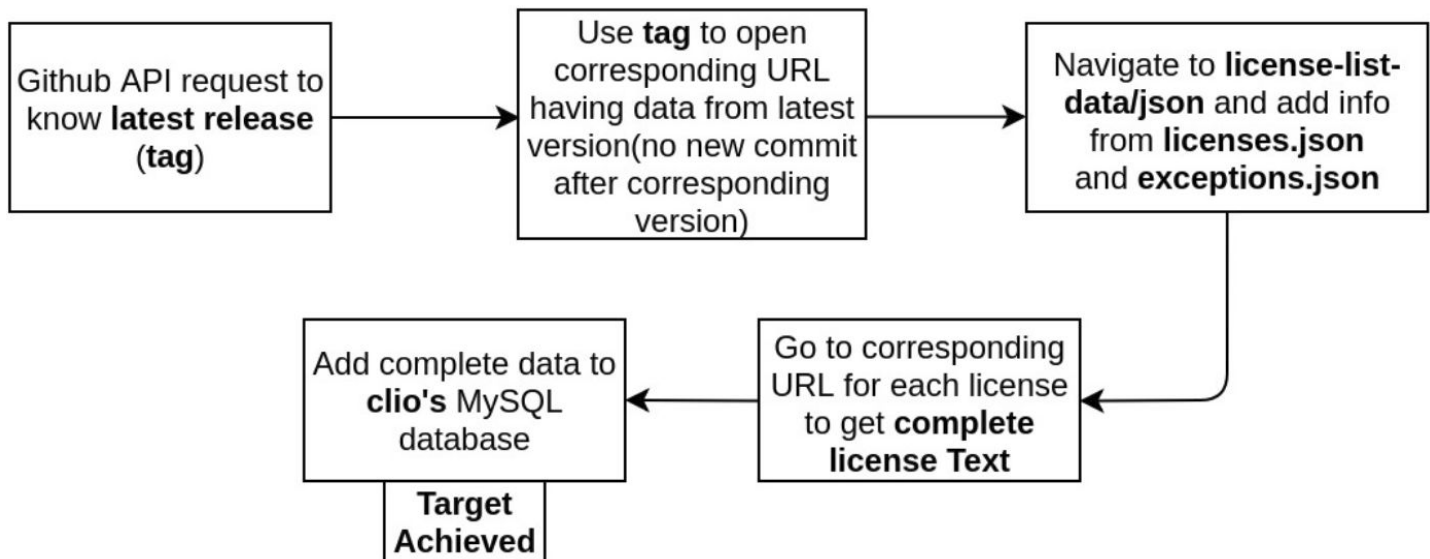
### Previous implementation:

```
def populate_license(directory):
    path = os.path.join(os.getcwd(), os.path.join(
        directory, 'license-info.csv'))
    with open(path, 'r', encoding='utf-8') as input_file:
        read_csv = csv.reader(input_file, delimiter=',')
        for row in read_csv:
            full_name = row[0]
            identifier = row[1]
            if(row[2] == 'FSF Libre'):
                fsf_free_libre = True
            else:
                fsf_free_libre = False
            if(row[3] == 'OSI Approved'):
                osi_approved = True
            else:
                osi_approved = False
            license_category = row[4]
            license_text = row[5]
            l = License(full_name, identifier, fsf_free_libre,
                        osi_approved, license_category, license_text)
            db.session.add(l)
```

The **disadvantages** of the above existing implementation in clio is that the user has to manually generate a .csv file containing license information and then using that .csv file to populate the database.

For every new release user/developer has to manually modify data from the database.

### My Improvement:



```
def populate_license(directory):
    url = 'https://api.github.com/repos/spdx/license-list-data/releases/latest'
    tag=requests.get(url).json()['tag_name'] #get latest version release info in form
of tag
    fetch_url='https://raw.githubusercontent.com/spdx/license-list-data/'+tag+'/json/licens
es.json' #generate destination URL
    jsonRes=requests.get(fetch_url).json()["licenses"]
    for i in range(len(jsonRes)):
        full_name = jsonRes[i]['name']
        identifier = jsonRes[i]['licenseId']
        fsf_free_libre=False
        if 'isFsfLibre' in jsonRes[i]:
            fsf_free_libre=True
        osi_approved=False
        if(jsonRes[i]['isOsiApproved']):
            osi_approved=True
        license_category = ''
        text_url=jsonRes[i]['detailsUrl']
        license_text = requests.get(text_url).json()['licenseText']
        l = License(full_name, identifier,
fsf_free_libre,osi_approved,license_category, license_text)
        db.session.add(l)
```

### c). Add functionality to warn users while adding duplicate entries.

**Problem:** There is no method to check Duplicate Components entry when a user creates it.

**Solution:**

**Task 1:** Implement a method which can warn users for duplicate Components entry.

**Task 2:** Suggest matched/similar components.

**Note:** Primary aim would be to complete Task 1.

## D. Report Generation

**Problem:** Currently Clio is unable to produce reports about complex components that contain other components. So, a functionality is required to produce reports about complex components.

**Solution:**

**1) Identifying the problem:** In the following section, I intend to seek a solution to the problem with the help of an example.

**Sample Components:** Consider the following components-

- **name:** zlib, license: Zlib, version: 1.2.11 (Simple Comp.);
- **name:** xyHash, license: BSD-2-Clause, version: 0.6.2 (Complex Comp.);
- **name:** my\_software, license: BSD-3-Clause, version: 0.1 (Complex Comp.)

Their relationships with **my\_software**:

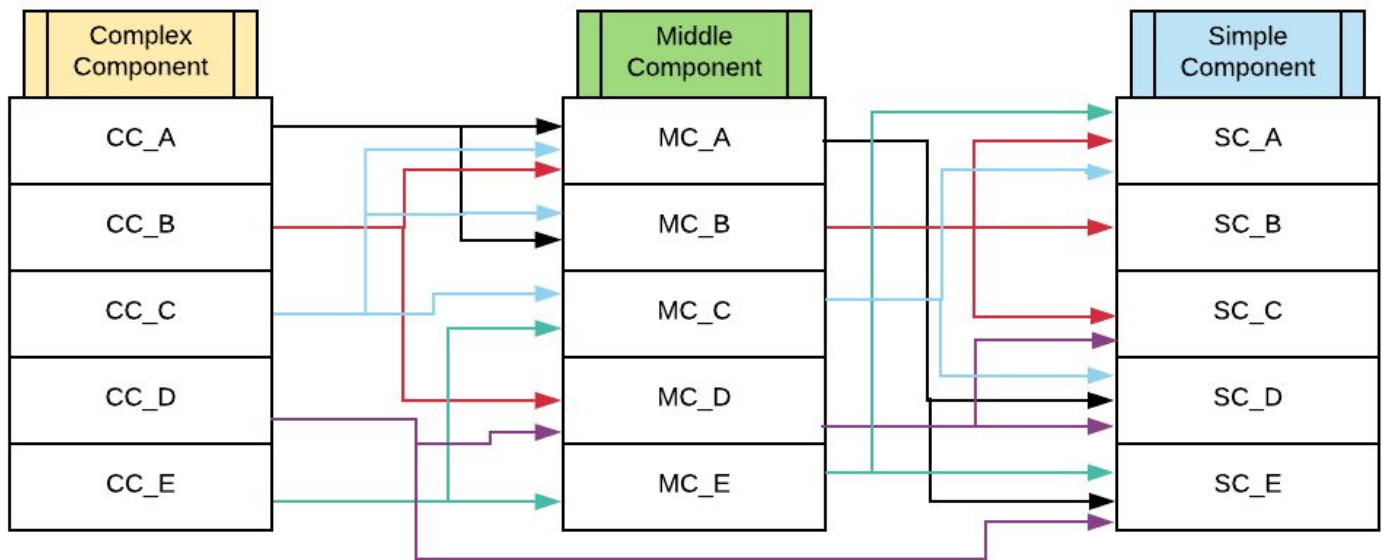
- **my\_software** INCLUDES **zlib**
- **my\_software** STATICALLY\_LINKS **xyHash**
- **my\_software** DYNAMICALLY\_LINKS **libc**

**Understanding the Relationships in Components:**

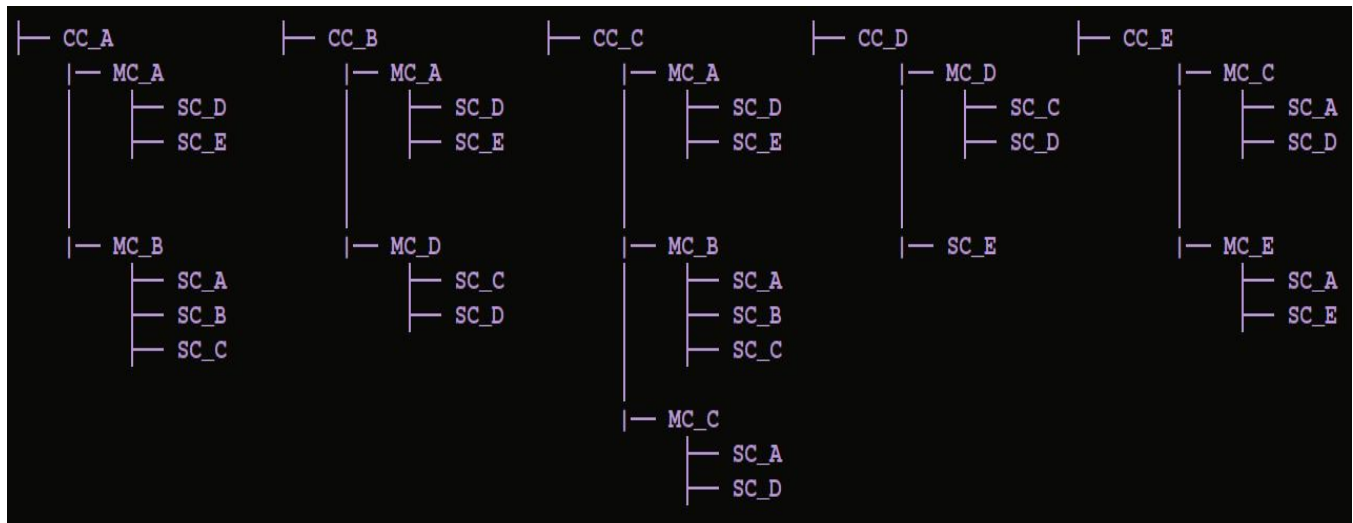
- A simple component can be *included* in multiple complex components
- A complex component can *include* multiple other complex components, the relationship turns out to be *n to n* on itself. Also, a complex component can *include* multiple simple components, thus making the relationship *n to n*

Hence, a middle Component Component relation is used to resolve the **n to n relation** into two **1 to n relations**. Defining **Components** relation as the middle table that is used to resolve the **n to n relation** between components into two **1 to n relations**, thus the relation established is **many-to-many non-identifying relationship** which boils down to **one-to-many non-identifying relationship**.

While it is required to have a unique identifier for each row in a table (called the primary key), this comes in handy as we start defining relationships. In order to link tables to each other, a **foreign key** is needed to say that a specific row in a table is linked to a specific row(s) in another table. (See the mockup for insights)



One-to-many non-identifying relationship database model for Components for CLIO



**Tree structure that would be generated following the above scheme.**

Current implementation of this looks like this in Clío:

```
components = db.relationship("Component", secondary="component_conn",
                             primaryjoin=id == component_conn.c.parent_id,
                             secondaryjoin=id == component_conn.c.child_id,
                             backref=db.backref("component",
                                                  lazy="dynamic"),
                             lazy="dynamic"
                             )

product_conn = db.relationship(
    "Product_Component_conn", back_populates="component")
```

In order to make a **relationship** between a **Middle/complex component and simple component**, we need to add a **foreign key** to the model to indicate the relationship to a Component. The best method for making this relationship is to specify the **primary\_key** value of the **User model**, which is the **'id'** field

```
component_conn = db.Table("component_conn",
                           db.Column("parent_id", db.Integer,
                                       db.ForeignKey("component.id")),
                           db.Column("child_id", db.Integer,
                                       db.ForeignKey("component.id"))
                           )
```

The most complicated relationship is many-to-many, as it requires the use of an intermediary table (ie. a linker table) to create the many-to-many relationship.

Now after analyzing and populating database in a proper manner, the only thing remaining would be gathering information from Complex component table and then checking the corresponding tag to get data from Middle component and then further check tags to get info about Simple components.

In this way, a complete report about Complex components (CC and MC) can be generated.

## E. External Service Integration

### Problem:

We need to be able to connect other services to clio to retrieve/validate data such as **SPDX** for licenses, **ClearlyDefined** for component license info, package managers. For eg., if someone wants to add a component X under license Y, a check could be made and a warning issued "hey, you want to add X under Y, but Maven (for example, or Pypi, or npm, or ClearlyDefined) says it's under Z"

### Solution:

- **ClearlyDefined**, **Pypi** and **npm** are some of the sources available for integration. The selection of service to be integrated mostly depends on the use-case.
- For checking if the license associated by the User for a particular component matches the original one, we need to initiate a GET/POST request with the component name as a request parameter.
- Check the license information in the response received and create warnings if something seems to be wrong.
- If the component is unavailable on ClearlyDefined, we might also go for some other sources like pypi and npm based on kind of component defined (python package or a node module).

## F. Improve data handling

**Problem:** Currently on license, component, product page and on Create component, Create product page, all the existing data loads at one shot making the website slow.

### Solution:

- Most of the data handling optimization will be completed after completing milestone upto E.

- After that, I'll try to find any vulnerability and other existing methods to improve the handling of the huge data.
- In this, I will try to explore more methods after discussing with my mentor and try to come up with a robust method which can handle the huge data in a few seconds.

## G. Fix Existing Issues

**Problem:** There are a plenty of existing issues in Clio which are required to be fixed.

**Solution:** After achieving all the milestones till F, most of the issues would be fixed, still if some major issues are left, I will try to solve them before GSoC 2019 ends. I have already created a list of existing issues in clio here <https://github.com/eellak/clio/issues>. I understand and know how to fix each of these issues, I also target to fix these issues after a few discussion with my project mentor.

The explicit list of issues created by me are as follows;

1) [Issue#36\(\)](#), 2) [Issue#37](#), 3) [Issue#39](#) (partially fixed), 4) [Issue#41](#), 5) [Issue#44](#), 5) [Issue#49](#), 6) [Issue#52](#), 7) [Issue#54](#), 8) [Issue#55](#), 9) [Issue#56](#), 10) [Issue#57](#), 11) [Issue#58](#), 12) [Issue#59](#)

## What technologies (programming languages, etc.) will you be using?

- **Backend framework / Tools :** Python3, Flask, Flask-SQLAlchemy (ORM), LDAP.
- **Database :** MySQL
- **Front end tools :** HTML, CSS, JavaScript

## How many hours will you spend each week on your project?

My summer vacations are starting from 1 May to 15 July, in that period I can give about 45-50 hours per week and after college starts, I will be able to manage 35-40 hours a week. I have no other commitments for the summer vacations, so I can devote most of my time to GSoC.

## Timeline:

| S.N. | Date              | Work  |
|------|-------------------|---|
| 1)   | 15 April - 12 May | <ul style="list-style-type: none"> <li>• Hiatus due to Institute End-Term examinations(24 April-10 May) and moving back home from college for Summer vacations.</li> <li>• Will be available on Zulip chat for further discussions</li> </ul> |



|    |                 |   |
|----|-----------------|---|
| 2) | 12 May - 26 May | <ul style="list-style-type: none"> <li>Community Bonding period</li> </ul> <p>Work on designing the landing page of clio, create mockups and get it reviewed from community and background study of all the frontend that will be used for optimization, propose new methods and finalize the techniques to readily start working on UI</p> |
|----|-----------------|---|

### Coding officially begins!

|    |                 |   |
|----|-----------------|---|
| 3) | 27 May - 2 Jun  | Start and complete major part of landing page   |
| 4) | 3 Jun - 9 Jun   | <p>Work on license, Components and Product redesign</p> <p>And better search representations</p> <p>And adding pagination</p> |
| 5) | 10 Jun - 16 Jun | Work on building new Sign in page   |
| 6) | 17 Jun - 23 Jun | Replace the redundant dropdown field with an autocomplete dropdown field in component form                                    |
| 7) | 24 Jun - 26 Jun | Add and integrate upload SPDX document field in component form  |
| 8) | 27 Jun          | Update/Modify weekly blog at <a href="https://shivanshu1333.github.io">shivanshu1333.github.io</a>                            |

### Phase 1 Evaluation (Deadline June 28 18:00 UTC)

|     |                   |   |
|-----|-------------------|---|
| 9)  | 28 Jun - 29 Jun   | Modify Code base after Phase 1 evaluation and background study for further work   |
| 10) | 30 Jun - 10 July  | Add "Show report" field in components page to know about Complex components   |
| 11) | 11 July - 14 July | <ul style="list-style-type: none"> <li>Hiatus due migration from home to college (29 hrs journey via train).</li> <li>College reopens.</li> <li>Will be available on Zulip chat for further discussions.</li> </ul> |
| 12) | 15 July - 20 July | <p>Better initial data seeding, incorporating data for well-known components</p> <p>Add functionality to warn users while adding duplicate entries</p>  |
| 13) | 21 July - 24 July | Change models and add migrations for maintaining components relation for report generation  |
| 14) | 25 July           | Update/Modify weekly blog at <a href="https://shivanshu1333.github.io">shivanshu1333.github.io</a>  |

### Phase 2 Evaluation (Deadline July 26 18:00 UTC)

|  |                 |  |
|--|-----------------|--|
| 15)  | 27 July - 5 Aug | Integrate external services for validating the component and license information entered by the user |
| 16)  | 5 Aug - 13 Aug  | Improve data handling  |
| 17)  | 13 Aug - 18 Aug | Start working on issues and complete any leftover work   |
| Final week starts (August 19 - 26 18:00 UTC)                   |                 |  |
| 19)  | 19 Aug - 25 Aug | Tidy up the code and add documentation for final submission  |
| Final Evaluation (August 26 - Sept 2 18:00 UTC)                |                 |  |
| Final results of Google Summer of Code 2019 announced (Sept 3) |                 |  |

## Contributions to GFOSS:

My contributions to open source have helped me gain experience in understanding the flow of any pre-written code at a rapid pace and enabled me to add/update features. My previous involvements in Clio includes various patches which are mentioned below:

- [#45](#)(merged): Update: Updated license data to SPDX license list v3.4
- [#48](#)(open): Bug Fix: Added complete license text
- [#35](#)(open): Bug Fix: fixes #34
- [#34](#)(closed): Bug: No progress bar in populate\_dp.py
- [#36](#)(open): Improvement: Needs a logo
- [#37](#)(open): Improvement: Improvement required for Clio's landing page
- [#41](#)(open): Bug: Website not Responsive
- [#44](#)(open): Bug: Add update Button on License Page
- [#49](#)(open): Improvement: Add a Linter to ensure consistent code style
- [#52](#)(open): Improvement: Populate database from upstream data
- [#53](#)(open): Update: Modified populate database method to directly fetch upstream data.
- [#54](#)(open): Bug: No facility to display exceptions and deprecated license. [UI improvement]
- [#55](#)(open): Improvement: Download License information
- [#56](#)(open): Bug: Update license category field
- [#57](#)(open): Improvement: Add a dropdown to add license category
- [#58](#)(open): Bug: Remember user credentials on login page
- [#59](#)(open): Bug: Export failure in components