

Sample Platform

CCEXtractor's Sample & Continuous Integration Platform

Table of Contents

1. **Basic Information**
2. **Sample Platform**
 - 2.1. Abstract
 - 2.2. Background
 - 2.3. Motivation
 - 2.4. Finalizing Sample Platform
 - 2.5. Why Do I Want To Improve Sample-platform?
 - 2.5.1. Why Sample-platform?
 - 2.5.2. Why CCEXtractor?
 - 2.6. Proposed Deliverables (during GSoC)
 - 2.7. Brief Working And Implementation
 - 2.7.1. Improve Testing, Results And Design
 - 2.7.2. Unit Tests Coverage
 - 2.7.3. Custom Error Handlers
 - 2.7.4. Database Migration
 - 2.7.5. Python 3.7 Compatibility
 - 2.7.6. Notifications
 - 2.7.7. Improved Installer
 - 2.8. Brief Timeline
 - 2.9. Detailed Project Timeline
 - 2.9.1. Phase 0 [Pre-GSoC Period]
 - 2.9.2. Phase 1 [Community Bonding Period]
 - 2.9.3. Phase 2 [Coding Period 1]
 - 2.9.4. Phase 3 [GSoC Phase 1 Evaluations]
 - 2.9.5. Phase 4 [Coding Period 2]
 - 2.9.6. Phase 5 [Phase 2 Evaluations]
 - 2.9.7. Phase 6 [Coding Period 3]
 - 2.9.8. Phase 7 [Final Evaluation]
 - 2.10. Additional Information Regarding Timeline
 - 2.11. Passive Tasks
 - 2.11.1. Overhaul Sample-platform's Documentation
 - 2.11.2. Resolve CCEXtractor Bugs
 - 2.12. Requirements
3. **Personal Information**

- 3.1. [Personal Details](#)
- 3.2. [Working Environment And Schedule](#)
- 3.3. [Communication](#)
- 3.4. [Contributions to CCEXtractor And Sample Platform](#)
- 3.5. [Post-GSoC Plans](#)

4. [References](#)

Basic Information

Name: Shivam Kumar Jha

Major: Computer Science And Engineering

University: [Indian Institute of Technology Kharagpur, India](#)

Expected Graduation Year: 2022

Github: [thealphadollar](#)

Slack: thealphadollar

Email: shivam.cs.iit.kgp@gmail.com

Phone: (+91)7830380698

Postal Address: C113, B. R. Ambedkar Hall of Residence
IIT Kharagpur, 721302
West Bengal, India

Timezone: Indian Standard Time (UTC +5:30)

Sample Platform

Abstract

Sample platform is CCEXtractor's platform that manages a test suite bot, sample uploads, running regression tests and much more. The purpose of the platform is to provide all the functionalities related to managing the CCEXtractor's repository; including but not limited to email notifications for new issues and pull requests, taking in samples from users (over FTP), testing each pull request and much more.

Background

CCEXtractor has become an integral part of many systems and the Github repository should always contain an error-free and ready to use version of the same. This involves checking every patch and update that comes and making sure nothing gets broken with a new feature or fix.

To tackle this issue, sample-platform comes into play. Sample-platform manages the task of running predefined tests after building CCEXtractor with the new patch and send a detailed report for all the tests (especially new ones) which are failing with the patch. Sample platform also has other functionalities such as providing notifications about new issues and pull requests on the repository.

Motivation

Sample-platform is fully functional and provides the basic facilities well but it requires a lot of work to finalize it. Firstly, since CCEXtractor is highly dependent on the correct functioning of sample-platform, we have to make sure it keeps running all the time and no patch breaks it.

It should be easy to install, setup and modify in case we ever need to. It is crucial to have version control systems for the database in place and hence we need to finalize the database migration manager for being future ready for schema changes.

Another (and the most) important factor are the tests, and at the moment there is very little customisation that can be done. The tests need to have improved comparison, better methods to provide the output, improved concurrent and parallel testing and more. A lot of other changes need to be implemented which are mentioned in details later on.

Finalizing Sample Platform

During GSoC 2019, I'll be working on sample platform to improve its existing functionalities along with adding multiple features.

Below is a tentative (after discussion with both the mentors) list of tasks we will be aiming to achieve:

- Improve Testing, Results And Design

The main purpose of sample-platform is to provide rigorous testing for CCEXtractor, and although it is being done nicely there are scopes for improvement in the algorithm as well as ways the results are conveyed to the user. We will also be implementing asynchronous testing which was erroneous previously; overall design improvements will also be implemented.

- Writing More Unit-tests

Currently, the test coverage for sample-platform is 63% only (as reported by [codecov](#)). I'll increase it to at least 90+% by the end of GSoC. Many of the poorly written tests will also be refactored.

- Improved Error Handling

Sample-platform will have its own separate class of errors to provide easy handling and better information for debugging purposes. This will also help in removing redundant code. An example is served below where try-except block can be modified into a single Exception Class.

```
50     try:
51         with open(session_file_path, 'rb') as session_file:
52             application.config['SECRET_KEY'] = session_file.read()
53     except IOError:
54         traceback.print_exc()
55         print('Error: No secret key. Create it with:')
56         if not os.path.isdir(os.path.dirname(session_file_path)):
57             print('mkdir -p', os.path.dirname(session_file_path))
58         print('head -c 24 /dev/urandom >', session_file_path)
59         do_exit = True
60
61     try:
62         with open(csrf_file_path, 'rb') as csrf_file:
63             application.config['CSRF_SESSION_KEY'] = csrf_file.read()
64     except IOError:
65         print('Error: No secret CSRF key. Create it with:')
66         if not os.path.isdir(os.path.dirname(csrf_file_path)):
67             print('mkdir -p', os.path.dirname(csrf_file_path))
68         print('head -c 24 /dev/urandom >', csrf_file_path)
69         do_exit = True
70
```

- Database Migration

Database management is an important part of sample-platform and it needs easier methods for migration in case we ever need to upgrade systems or need to update the schema. A git-like version control for database management upgrades is what we need.

- Migration To Python 3.7

Python 3.7 has a plethora of new features; the most exciting of which is [typing](#). One step towards static-typed variables, typing allows [soft] limiting type of arguments.

- [Notifications](#)
Sample-platform will provide notifications to the user (role-specific) regarding test completion, failure and give him the link where he can see the results in detail.
- [Improvement in Installation Method](#)
The current installer has issues such as [silently] not installing MySQL client. These issues will be noted during the community bonding period, discussed and then taken care of.

There will be discussion with users, contributors, and maintainers of sample-platform, and accordingly, more features will be added. Post GSoC sample-platform will be ready with all the listed changes. An aim of the project will also be to create smaller tasks which can be taken up by students during GCI 2019.

Why Do I Want To Improve Sample-platform?

Why Sample-platform?

I've worked last year on Nephos and since then I've gained experience in frontend and backend operations. The tasks in sample-platform require the skills I've gained and there are components which I can't wait to learn such as VM handling, Comparison testing and much more.

I also have familiarity with the code of the sample-platform as I was a mentor for many GCI tasks and helped students' with their tasks. I've also contributed to it and have reviewed several patches to the platform.

Above all, I'm highly motivated to contribute to CCEXtractor with the knowledge I've gained, and want to be part of the team to finalize an important part of the organisation.

Why CCEXtractor?

Choosing CCEXtractor as the organisation to work with [again] during summers is a choice motivated by multiple but two major factors. First, I have done GSoC 2018 with CCEXtractor and I found my experience to be very riching. I've got to learn a lot from my mentors, who were very friendly and supportive.

Secondly, the fact that the CCEXtractor community is very responsive, helpful and guiding to new contributors. I wish to be a permanent part of the community and do my part in spreading open source and helping others get on board.

Proposed Deliverables (during GSoC)

1. **Improved Testing, Results, and Display:**
 - 1.1. Improved Results Page
 - 1.2. Improved Expected Time Prediction
 - 1.3. Improved Tolerance For Minor Changes
 - 1.4. Make Output Files Available
 - 1.5. Concurrent Platform Running
 - 1.6. Revamp Sample-Platform Design
 - 1.7. Bug Fixing for Current Testing Issues
2. **>90% Line Coverage on Tests**
3. **Custom Error Handlers**
4. **Detailed Method for Database Migration**
5. **Full Compatibility with Python 3.7**
6. **Notifications for Test Events**
7. **Improved Installer**
8. **Tasks For GCI**
9. **Monthly blogs on developmental advances and milestones.**

Brief Working And Implementation

Improve Testing, Results, And Design

The following changes and features are proposed in addition to solving existing bugs in the platform and improving the basic design.

Before adding new features, I would focus on rectifying the bugs present in the current module. A few of the known bugs are listed below. During the Pre-GSoC and Community Bonding period, I'll be discussing in more detail the bugs we might face (which are currently not present).

- **Improve Results Page**

The result page currently is based on an old style which leads to multiple slower operations. One such example is the display of the below failed test result from [test#2087](#). Once clicked, the diff (a new model) takes around 10 seconds to show up due it's large size (roughly three hundred lines).

82	3b276ad8bf...	-autoprogram - out=srt -latin1 - teletext -tpage 398	7012	0	Fail
83	b236a0590b...	-autoprogram - out=txt -latin1	7609	0	Pass

To tackle this error, from the API to the model we will return only the first 50 headlines of the diff and provide the user with an option to download the full diff in a text format. The solution addresses two major concerns:

- It is more convenient to view smaller diffs in a browser and get a rough idea about the cause of failure.
- A technical user (or developer), who wants to analyze the diff and rectify the bug, would be more interested to get data in a format which is easier to operate with using scripts and programs (format such as text).

NOTE: Pagination was discussed as an option with mentors and it was found that since if anyone wishes to know in detail, they'd have a technical purpose for which a txt file is better. Also, most of the failures have a pattern which can be easily noticed from the first few diffs and it is not efficient to implement a pagination system to the API and increase the backend complexity.

- **Better Estimated Time Prediction**

The current method (added via [#a8d713](#)) of test estimation uses the below formula to estimate the remaining time for the test.

$$estimatedTime = [(num_{tests\ in\ queue} + 1) * avgTime_{on\ platform}] - TimeSpent_{queued\ tests}$$

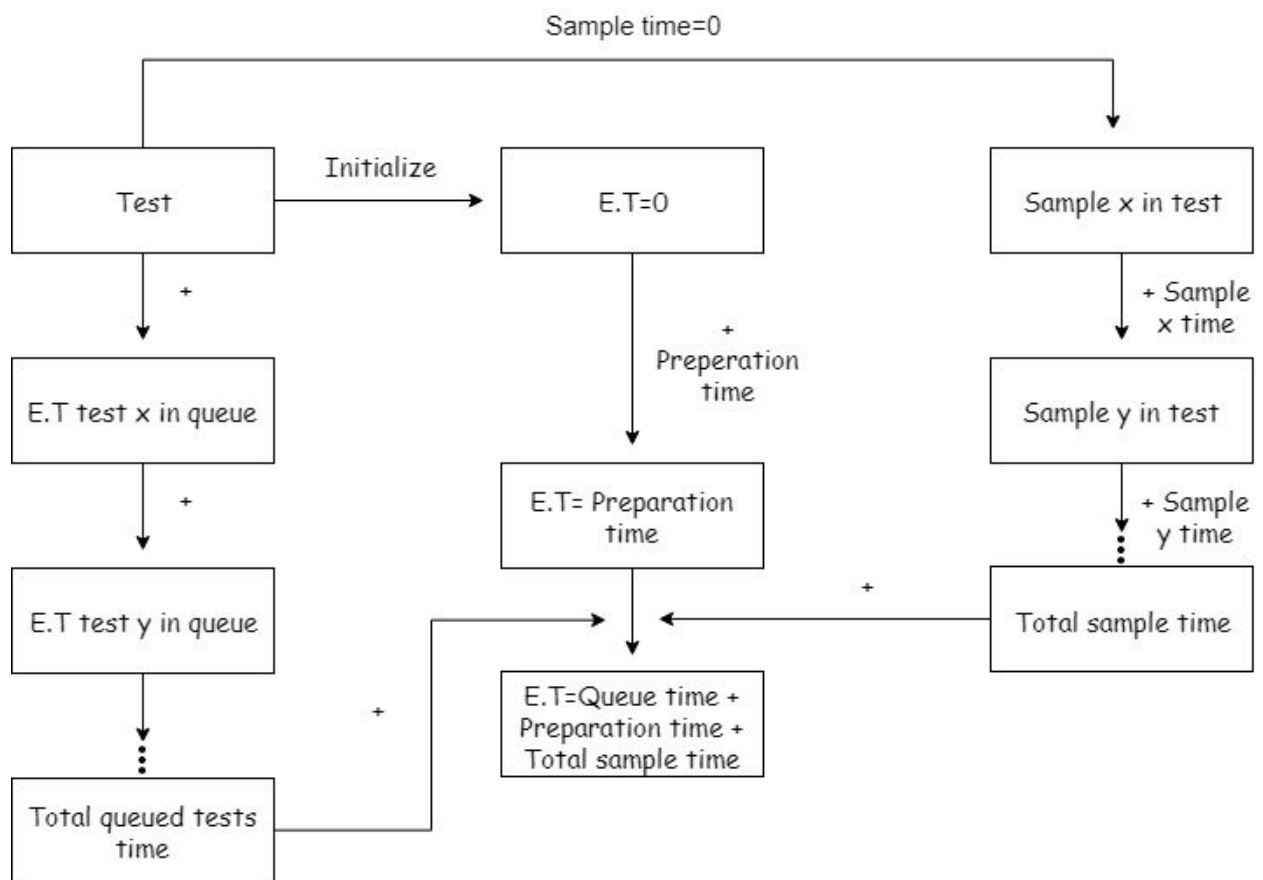
This method of estimation has the following issues:

- It doesn't take into account the number of tests user has selected. So, even for one test queued the sample-platform shows the estimated time as if all tests are queued (the case for an automated test on PR).
- It doesn't take into account that every platform has a specific preparation time (involving copying files, compiling CCEXtractor, etc). So, directly dividing the time by the number of total tests and then taking a product with the number of tests selected is also not correct.

A better approach would be the following algorithm which has been explained well with the help of a flow diagram.

$$queueTime = [(estimatedTime_{test} - (currentTime - beginTime))]_{for\ test\ in\ queue}$$

$$estimatedTime = queueTime + prepTime_{platform} + avgTime_{for\ sample\ in\ test}$$



This method estimates the time required to complete a test more accurately though it is much more expensive than the previous method. The method requires the following

changes in the way we currently do tests.

- Store average preparation time for each platform alongside average time.
- Add new fields in the sample table and store average time for each sample on each platform.
- While the test is running and reaches a particular sample, add a beginning timestamp and similarly add an ending timestamp which will be used to calculate it's running time.
- The above newly added values are updated at the end of each test.

Since this change will require updating the schema of the table(s), I've placed the implementation of database migration method in the first coding period and improvement related to tests in the second coding period.

The following new features (as of now) have been discussed and will be implemented. I'll be in constant conversation about different aspects of the sample platform and algorithms used by it, and we'll be identifying areas of improvement.

- **Improve Comparison Algorithm**

The comparison of expected output and the output got from the test is tightly hard-coded as it is done by simply matching the file hashes. This causes a lot of issues, a few of which are mentioned below.

- Intolerance to a slight change in the timestamp.
- Intolerance to detection of “,” as “.” or vice versa due to update in OCR.
- Intolerance to a slight error in whitespace matching.

A more flexible approach can be used; we will not mark a test as a [failure if the results value returned is not None](#). Instead, if the results are not None, a separate function will take in the result lines of the files and do a diff again once the below operations are performed. If the diff is not empty yet, the test will be marked fail. A small code snippet is attached for an example processing of the string.

```
import difflib

string1 = "I've something to prove here, if it is not obvious!"
string2 = "I've something to prove here. if it is not obvious!"

change_tolerate = [{"`", "'"}, [".", ","]]
tolerate_whitespace = True

# preprocessing
if tolerate_whitespace:
    string1 = string1.replace(' ', '')
    string2 = string2.replace(' ', '')
for toleration in change_tolerate:
    string1 = string1.replace(toleration[1], toleration[0])

for line in difflib.unified_diff(string1, string2):
    print(line)
```

I went through a couple of failed tests and I could gather that the following toleration options could be provided while setting the tests (and a global config for CI). All these operations can be done using the replace method on strings (PoC at the end).

- Whitespace: Whitespace differences should be ignored.
- List of similar characters: Characters which are similar and can be ignored such as “.” and “,” or “!” and “1” can be made.

- Slight delay in time: If the time variance is varying by only a slight, the file can be marked as safe. For this, we will parse all the integers from the different lines and compare them and see if the difference is less than specified by the user.

Above configuration options are tentative and more will be added (or removed) as we see fit during the actual implementation of the task and after more discussion with mentors.

- **Make Output Files Available**

For each test, we have a page which lists the sample and other related information.

Regression test 1

Sample: [Sample #1](#)

Command: -autoprogram -out=srt -latin1

Input type: File

Output type: File

Output files: Coming soon

The expected output for the sample is currently stored on the server and can be easily exposed at an endpoint as we already serve [samples](#), [sample info](#), and [additional information](#) under mod_sample/controllers.py

The file path for the expected test results can be fetched using the following function present in the [model for regression test](#). This fetched filename needs to be combined with `os.path.join(config.get('SAMPLE_REPOSITORY', ''), 'TestResults')` to get full file path for providing download.

```
def filename_expected(self, sample_hash):
    """
    Return expected filename

    :param sample_hash: sample_hash of RegressionTestOutput
    :type name: str
    :return: String containing name, expected filename, particular extension
    :rtype: str
    """
    return "{sha}{extra}{ext}".format(sha=sample_hash, extra=self.expected_filename, ext=self.correct_extension)
```

NOTE: Under this task, I'll also be changing the outdated result files because there are multiple cases as below ones which fail due to changes such as better detection of font color in newer versions.

n°	Result	Expected
4	<pre>in this sub in jeOPardv — in this sub in jeopardy</pre>	
13	<pre>What did we find down there? What did we find down there?</pre>	

- **Rectifying Parallel Testing**

Running multiple platforms in parallel was added in sample-platform long back but got reverted in [#0b7511](#) as it was causing many errors (maybe a race issue between the processes).

As currently, the sample platform doesn't implement parallel processing, I've not been able to test for the bugs. Since a lot has changed since the revert, I'll implement concurrency and see for the errors. I've experience with implementing parallel processing (implemented in Nephos for pre-processing but later removed since it was overkill) and multi-threading (implemented in multiple network I/O heavy projects).

As it is not possible to pinpoint improvement without running the sample-platform in concurrent mode, I'll be listing a few general methods I'll use:

- Use multiprocessing and not threading since virtualisation is an intensive task bottlenecked only by processing power and not I/O from disk or network.
- Limit the maximum number of connections to the number of CPU cores since we are using Virtualisation on our platforms and it benefits from number of cores.
- In my experience with python concurrent programming, I've observed that scaling to multiple processes leads to inconsistency in logging and output. I'll take special care of this.

NOTE: I've had a conversation with Satyam and it is not feasible to go into details into this part without actually having run sample-platform with code for parallel processing.

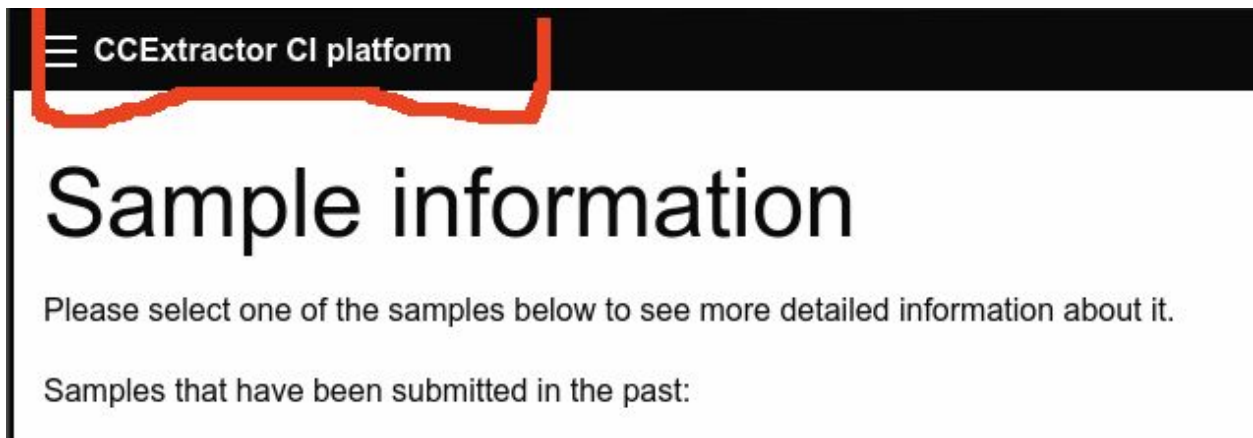
- **Revamp Sample-Platform Design**

We use Foundation 6.4 and the latest version, as of 5 April, is 6.5. There are no major changes in the framework as such but porting to the new version is always better. So, at first, I'll update our framework to version 6.5 and if any [bugs get introduced after the upgrade](#), I'll duly solve them.

Moving on, I'd be implementing design changes; these changes would solely be there for improving the visuals and will not affect functionality in any way. I've been using sample-platform for a while now and have a few improvements listed below. I would be, throughout the period, making note of other changes and implementing them in parallel with other tasks.

- Make the title bar consistent over all screen sizes: currently, it's just shown on small screen sizes and not large ones.

Title On Smaller Screens



Nothing On Large Screens



- Improve menu style: a vertical style menu is good for smaller screens but doesn't look good on large screens and hence I'll be implementing a horizontal style.
- Improve [typography](#) on each page: Currently, there is no appropriate emphasis on the title and all the text tends to blend with each other. There is no proper alignment of the elements as well. I'll address these issues for all the pages of the sample platform.

Since this doesn't bring any functional improvements (but definitely makes the website more professional), I'll implement them as and whenever I'm ahead of schedule for a phase.

Unit Tests Coverage

Currently, the test coverage of sample-platform is 63.95% and is unevenly distributed among different aspects of the platform as shown below.

For testing purposes and paradigms, I'll be using the methods and principles mentioned over [pythontesting.net](#).

Files	≡	●	●	●	Coverage
install	46	0	0	46	0.00%
mod_auth	349	212	8	129	60.74%
mod_ci	741	334	34	373	45.07%
mod_customized	118	116	2	0	98.31%
mod_home	45	43	0	2	95.56%
mod_regression	226	220	0	6	97.35%
mod_sample	378	171	6	201	45.24%
mod_test	466	344	21	101	73.82%
mod_upload	388	297	22	69	76.55%
config_parser.py	9	2	0	7	22.22%
database.py	69	61	2	6	88.41%
decorators.py	53	44	4	5	83.02%
log_configuration.py	25	23	1	1	92.00%
mailer.py	12	12	0	0	100.00%
mod_deploy/controllers.py	94	39	5	50	41.49%
run.py	116	86	1	29	74.14%
utility.py	2	2	0	0	100.00%
Project Totals (36 files)	3,...	2,...	106	1,...	63.95%

Firstly, project files which should not be included in the test coverage results will be added to the existing .coveragerc in order to get more accurate information about the coverage.

Since the application is mostly based on Flask, we will be using standard [flask testing methods using Py.test](#) to create tests. There are many tests (an example is below) which need to be refactored, and single responsibility principle should be implemented in order to avoid redundant testing.

```

@mock.patch('mailer.Mailer')
def test_inform_mailing_list(self, mock_email):
    """
    Test the inform_mailing_list function
    """
    from mod_ci.controllers import inform_mailing_list
    from mailer import Mailer

    email = inform_mailing_list(mock_email, "matejmecka", "2430", "Some random string",
                                "Lorem Ipsum sit dolor amet...")

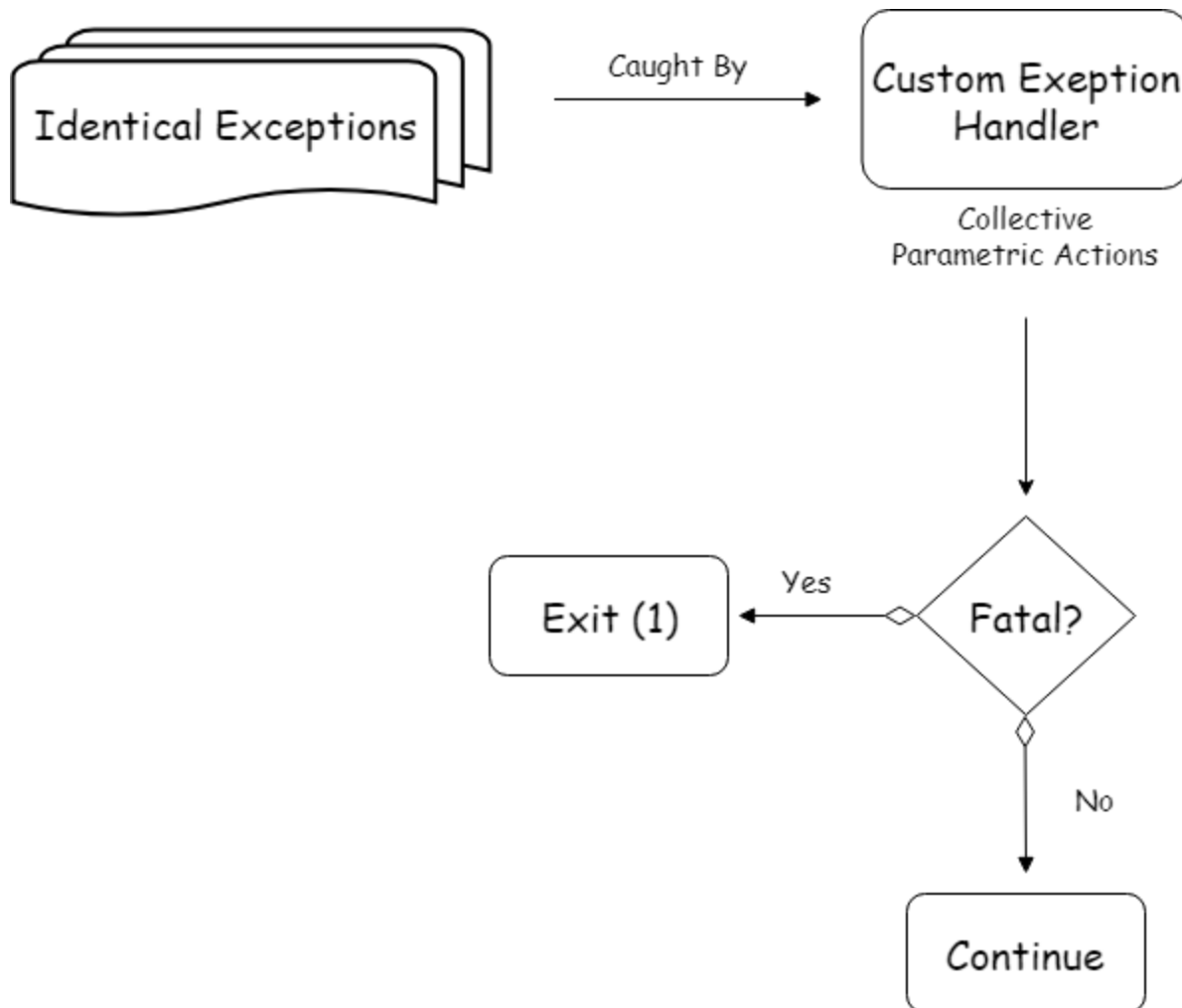
    mock_email.send_simple_message.assert_called_once_with(
        {
            'text': '2430 - Some random string\n\n'
                    '    Link to Issue: https://www.github.com/test_owner/test_repo/issues/matejmecka\n\n'
                    '    Some random string(https://github.com/Some random string)\n\n\n'
                    '    Lorem Ipsum sit dolor amet...\n            ',
            'subject': 'GitHub Issue #matejmecka', 'to': 'ccextractor-dev@googlegroups.com'
        }
    )

```

For instance, the [above test](#) should be re-written as follows to avoid checking the side-effect [get_github_issue_link](#) in the function [inform_mailing_list](#). It is a bad practice which should be avoided in order to not have redundant testing as well as specific points of fault; a bug in `run.py` should **not** be caught in `mod_ci.controllers`.

The aim will be to achieve re-factorisation and around 90% test coverage before first evaluations.

Custom Error Handlers



There are two main reasons why custom error handlers are helpful for a program:

- They provide with the ability to pinpoint the nature of the error. An error due to invalid database settings can be better notified by an exception subclass, name "DatabaseConfigException", of SQLAlchemyError.
- Helps reduce code redundancy by collecting a set of actions for a particular error under a single class.
- Reveals the location of the code where the error was generated (although this is possible via Logging as well).

Below is [an instance](#) where this will be useful in removing code redundancy and improving overall readability.

```
try:
    with open(session_file_path, 'rb') as session_file:
        application.config['SECRET_KEY'] = session_file.read()
except IOError:
    traceback.print_exc()
    print('Error: No secret key. Create it with:')
    if not os.path.isdir(os.path.dirname(session_file_path)):
        print('mkdir -p', os.path.dirname(session_file_path))
    print('head -c 24 /dev/urandom >', session_file_path)
    do_exit = True

try:
    with open(csrf_file_path, 'rb') as csrf_file:
        application.config['CSRF_SESSION_KEY'] = csrf_file.read()
except IOError:
    print('Error: No secret CSRF key. Create it with:')
    if not os.path.isdir(os.path.dirname(csrf_file_path)):
        print('mkdir -p', os.path.dirname(csrf_file_path))
    print('head -c 24 /dev/urandom >', csrf_file_path)
    do_exit = True

if do_exit:
    sys.exit(1)
```

In place of identical code (bare minimum, no docstrings present) inside multiple try-catch blocks above, we can have a single Exception handler as below.

```
1 class KeyFileNotFound(IOError):
2
3     def __init__(self, key_file_path):
4         traceback.print_exc()
5         LOG.error("Error: secret key not found. Create it with:")
6         if not os.path.isdir(os.path.dirname(key_file_path)):
7             print('mkdir -p', os.path.dirname(key_file_path))
8         LOG.error('head -c 24 /dev/urandom >', key_file_path)
9         sys.exit(1)
```

I'll also be implementing better error handling (more try-catch blocks) to the code to make it more robust and error-resistant. Currently, we do not have try-catch blocks at necessary segments of code such as [databases.py](#); we can have try-catch block on creating a session, committing, etc. which may lead to exceptions.

Database Migration

Sample-platform uses SQLAlchemy to manage the database and crucial information such as users, test progress, samples' information, CCExtractor's version, and last commit is stored in the database only.

Not very often, but sometime in future, we might need to update the existing SQLAlchemy models and hence it is [very important to have a database migration](#) method in place to avoid data loss and system unavailability. Just like we can roll back to a previous version in case of bugs (or unwanted feature) using version control systems for our code, similar functionality can be implemented to ORM managed databases using database migration tools.

Currently, we have [two PRs](#) for sample-platform which implement different methods of database migration but both use Alembic underneath. After doing some solid research, I've come to the conclusion that it is best to not mix up Alembic with Flask (by using PR with flask-migrate) since database migration is a rare but crucial event and, in my opinion, it's best to let it remain separate. Adding to that, standalone Alembic provides the following features (brief tutorial [here](#) for all the points discussed below) which increase the security and transparency of the process.

- **Easy Rollbacks**

As expected from a migration tool, easy reversibility is a feature of Alembic; it allows upgrade and downgrade of database system similar to checking out commits in version control systems (except in some cases).

- **SQL Commands Generation**

One of the two crucial features of Alembic is that it allows the developer to see the SQL operations it is going to carry out (if asked). This feature can be used to ensure that nothing unwanted happens in the migration. An example is that the auto-generated script for renaming a table actually drops the table completely. This can be avoided if SQL commands are generated and verified before migration.

- **Auto-generate Migration Scripts**

This is the other crucial feature of use to us; in case of changes in models, Alembic automatically takes in the new schema and generates the migration script (different from SQL commands). In this way, we only need to define new models and the rest is taken care of by Alembic.

I would be doing a little more research before concluding on Alembic since database migration is very crucial and the selection is everlasting since the generated scripts are dependent on the tool used (as well as the DMBS) and so is the method of migration (the process). It is also

important to have a very transparent method of migration since (in most cases) it is not easy (but possible) to get back lost data (and hence drop/rename tables are very skeptical operations).

Python 3.7 Compatibility

One of the drawbacks of Python is that it is not statically typed; variables are checked for data-type consistency at runtime (and only if they are executed). This is a problem in large projects as it makes it hard for finding errors and debugging code.

This can affect the sample-platform adversely since not all the code is run most of the time and specific parts of the code are run at a time. So, if a wrong data-type is passed and that branch of the code is run, it'll lead to a fatal error.

We'll be using a two-step process to tackle the above process which has been explained below.

- **Set Typing For Existing Functions And Variables**

Python *typing* library was introduced in version 3.5 and allows for the setting of type hints for variables and function parameters. Quoting [official python docs](#), "As a result of PEP 560 work, the import time of typing has been reduced by a factor of 7, and many typing operations are now faster". This makes it the best time to add support for typings in sample-platform's code.

Below is a time comparison([by RealPython](#)) of importing *typing* module between Python 3.6 and Python 3.7.

Shell

```
$ python3.6 -m timeit -n 1 -r 1 "import typing"
1 loops, best of 1: 9.77 msec per loop
$ python3.7 -m timeit -n 1 -r 1 "import typing"
1 loop, best of 1: 1.97 msec per loop
```

Implementation of typing and annotations will not be an "all-or-nothing" feature and it'll only be applied to all functions and crucial variables in order to maintain Python's ease of writing code (i.e. using dynamic typing).

- **MyPy With Unit Test**

Despite adding typings, the data types are checked only at runtime; adding typings does not make any change to how the code is interpreted and errors will be caught only at

runtime. To avoid this, we will be using [pytest-mypy](#) to integrate testing for type violations in our unit-tests.

[Mypy](#) is a static type checker for Python and uses the written typings to “enforce” static types and checks for consistency (which by default is not done).

NOTE: For enforcing (in “hard” manner) we can use third-party [enforce module](#) but it is not advisable since it’ll lead to a loss in code readability due to redundant [use of decorators](#) and also it’ll lead to the loss of flexibility.

Notifications

Sample-platform implements a class *Mailer* which is the single point of execution in case of communicating with users (currently we will targeting users with the ability to run custom tests, i.e. users under the role of “testers”, “contributors” and “admins”) over email. The function requires a dictionary of the following format, using which any number of emails (personalized using Jinja Templating System which we currently use for the same) can be sent.

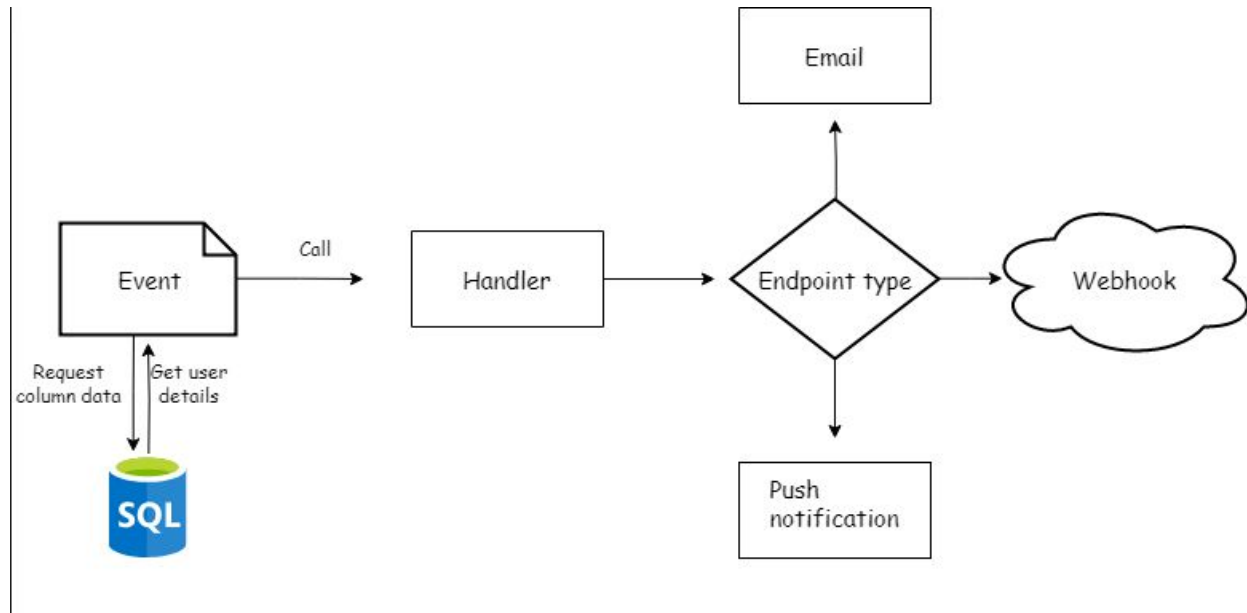
```
{  
    "to": $EMAIL_ID,  
    "subject": $SUBJECT,  
    "text": $CONTENT  
}
```

The following steps need to be done for creating test notification:

- **Fetch User Email Address:** Currently, in the payload, we receive for an event, the user id and login name are present. For e.g. in the Pull Request payload, the username can be accessed easily via `payload[pull_request][user][login]`. Once we have the user login name, we can query (GET request) the [Github API v3](#) at `/user/emails` to get the user’s public email addresses.
NOTE: The next two steps are followed if the user has at least one public email address.
- **Create A Personalized Email Body:** We will create personalized email addresses using the already existing templates. The templates will have a minimal body and the link to the test results.
- **Call To send_simple_message:** We will call the method from Mailer class with an appropriate subject.

After a thorough discussion with mentors, the categories for notifications, the frequency, and the exact template will be decided at the time of implementation. Currently, there are two categories for which the notifications can be sent: failures and success after an initial failure(s) for a single PR.

I'll be calling a handler function once all the information regarding the notification to be sent is generated. The handler function will then disseminate the information according to various mechanisms; this will enable plug and play model for different notification methods such as push, email, webhook, etc.



After discussion with mentors for the project, we have decided that I will try to implement a browser-based push notification system in order to notify users of the test completion, sample upload complete, etc. This deliverable was provided recently and more work needs to be done. One of the mentors (Satyam Mittal) have implemented such a system and I'll be using his experience and guidance in the same. A detailed groundwork on the same will be done during the PreGSoC and community bonding period. This system will be implemented alongside email notifications.

NOTE: The idea is tentative at the moment as discussed with a mentor (Willem) since we do not want to spam the user with notifications. Once we have implemented other features and improvements during GSoC, a thorough analysis of deliverable information (such as output from failures, etc) will be done and the notification system will be implemented (hence placed at the end in the timeline).

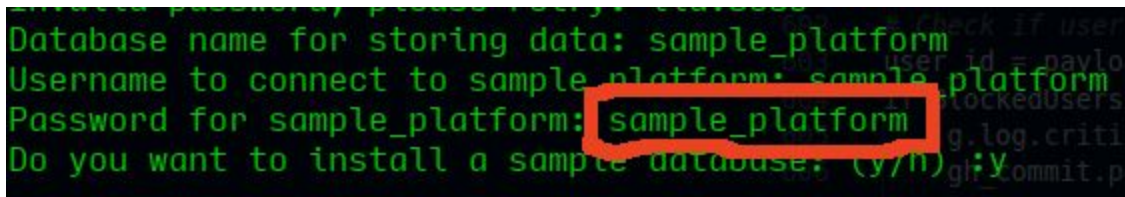
Improved Installer

Currently, the sample-platform has a shell script to help in installation on Linux platforms and the same script is used for windows installation after setting up native Linux app runner Cygwin on Windows.

The current installer serves the purpose well but there are improvements that can be made. One of the areas that can be improved is implementing post-installation checks. For e.g. if MySQL client is not found, installation is run for the same but irrespective of the outcome of the installation process, the script moves forward.

A check should be put in place (for e.g. running “mysql --version”) and if the check fails, the installation can do either of the two actions depending on the severity of the [installation failed] package; abort installation (with a warning message) or ask user if he wishes to continue mentioning the affected functionality.

Another small improvement would be not displaying the passwords that the user enters or sets for the first time. This can be easily done (and will be done before GSoC proposal submission) in [multiple ways](#); I would be using the method of turning off ‘echo’. A caveat here is that the user might set a wrong password than what he intends to. To tackle the same, I will be implementing a confirm password method.



```
Database name for storing data: sample_platform
Username to connect to sample_platform: sample_platform
Password for sample_platform: sample_platform
Do you want to install a sample database: (y/n):y
```

Since installer is something which might get updated as we incorporate features and changes into the existing code, I'll be improving the installer at the end of the GSoC period, at the time of wrapping up my work.

Brief Timeline

- (Phase 0) Till 6 May: Pre-GSoC Period
- (Phase 1) 6 May - 27 May: Community Bonding Period
- (Phase 2) 28 May - 24 June: Coding Period 1
- (Phase 3) 24 June - 28 June: Phase 1 Evaluations
- (Phase 4) 29 June - 22 July: Coding Period 2
- (Phase 5) 23 July - 26 July: Phase 2 Evaluations
- (Phase 6) 27 July - 26 August: Coding Period 3 and Mentor Evaluation Submission
- (Phase 7) 27 August - 2 September: Final Evaluation

Detailed Project Timeline

Phase 0 [Pre-GSoC Period]

- **3 Weeks (9 April - 21 April)**

I've been getting familiar with the existing sample-platform code. I've worked on a few issues and will be working on more issues during this time. I aim to have gained a better understanding of the sample-platform from my mentors.

I'll also gain more insights about the technologies and dependencies to be used, and also make note of different error cases so that I can prepare the error handlers easily.

- **1 Week (22 April - 30 April)**

End semester exams begin at my institute, and I will not be contributing actively during this period. Nonetheless, I'll be actively following the progress and participate in conversations over slack.

Reference: [Institute's annual calendar](#)

- **1 Week (1 May - 6 May)**

I'll be going on vacation during this period. I'll be readily available on communication but not actively involved in the communication.

Phase 1 [Community Bonding Period]

- **2 Weeks (7 May - 27 May)**

During the community bonding, the main focus will be to frame a roadmap for the project with the guidance of the mentor (along with improving bonding, which is what the period is for). This period will also involve continuing the groundwork for sample-platform. I'll begin creating the base for writing tests and learning methods and paradigms involved in the same.

A dropbox paper with task listing and distribution will be created which will be jointly managed by mentors and me to keep track of progress.

I'll also work on creating docstrings for all functions during this period and finish it before the end of the period.

Phase 2 [Coding Period 1]

- **3 Weeks (28 May - 18 June)**

This period will be given to the first set of microtasks; python 3.7 compatibilities, writing unit tests, custom error handlers, and database migration. First and foremost, the existing code will be made compatible with Python 3.7 before writing any more code. This will also include implementing new features (such as typings which goes hand-in-hand with docstrings).

From there on, I'll start writing custom error handlers (using the different error classes collected in earlier periods). Once error handlers are done, I'll start writing unit tests and try to get coverage of more than 80% for the existing code.

- **1 Week (19 June - 24 June)**

More tests will be written making the coverage reach about 90%. During this period, I'll also do the research for database migration methods ([using the two existing PRs](#)) and document and implement the same.

Phase 3 [GSoC Phase 1 Evaluations]

This period will be used to write a detailed report on the work done in Coding Period 1. All the work done will be sent as separate PRs and all concerning documentation will be done.

Deliverables

- Python 3.7 Compatibility
- Custom Error Handlers
- Approx. 90% Unit Test Coverage
- Database Migration Method

Phase 4 [Coding Period 2]

- **2 Weeks (29 June - 14 July)**

This period will be utilized solely for improving regression tests' algorithms and results.

Most of the backend work will be done during this period such as implementing tolerance, making the flask route for output files, implementing and testing concurrency, and improving the method of predicting estimated time. This will most probably be carried forward into the next section as well.

- **1 Week (15 July - 22 July)**

The work from the previous two weeks will be finished this week. Thorough testing of the implementation will be done before working on frontend changes.

Phase 5 [Phase 2 Evaluations]

A detailed report on the working of the second coding period will be written during this time.

Deliverables

- Improved Testing And Results Backend

Phase 6 [Coding Period 3]

- **2 Weeks (27 July - 12 August)**

Frontend work for the improved tests' results will be done including adding configuration options to customized test panel and revamping results display page. Appropriate documentation patches will also be sent during this period. I'll also be carrying out the design upgradation.

Afterward, notification implementation and improvement of the installer (at last after all changes have been finalized) will be done.

- **2 Weeks (13 August - 26 August)**

Finishing up, any leftover unit tests (so that code coverage is more than 90%), documentation updates, bug fixes, and other fixations will be completed.

This will also be used as a buffer period in case of any lag in the schedule.

Phase 7 [Final Evaluation]

All documentation, improvements, UI updates and unit tests will be provided in PRs.

All the deliverables promised for GSoC will be provided by this stage.

Additional Information Regarding Timeline

- The above timeline is tentative and gives a rough idea of my planned project work. I'll try to keep progress at, at the very least, the proposed schedule. I'll share a detailed plan of work with my mentor at the beginning of each week.

- I've no other commitments during summer and hence, will be able to dedicate 30 hours to 36 hours a week. During the last month of the project, my college will begin and I'll be able to denote a max of 20 hours a week. Due to the same, a major portion of the work will be done during the first and middle phase of the timeline.
- Each week, time will be divided (according to workload) amongst planning, learning, coding, documenting and testing features. All documentation will go hand in hand with the development (in the form of docstrings and typings). This will help to keep a profound grasp over the code implementation and working, minimizing bugs in the later stages.
- Weekends will be mostly non-working unless there has been slack in the schedule. Monthly (per phase) blogs will be maintained at <https://thealphadollar.github.io> and will include highlights of the development process and also methods used to overcome hurdles.
- One of the reasons I've been active in CCEXtractor is due to the immensely helpful community and I'll try to keep myself continuously updated with other developments going on. I'll be available for communication at all times decided by my mentor and discuss new ideas and methods throughout the project. I'll try to guide the newcomers with my one year of experience with the community.

Passive Tasks

The above timeline has been made with keeping in mind the maximum time any task could take and the worst case scenario of having a lot of bugs. Most probably, the work will be running ahead of schedule and hence I'll be passively working on below targets as well.

• Overhaul Sample-platform's Documentation

With the help of my mentor, I would like to revamp sample-platform's existing documentation. With Season of Docs at the doorsteps, I'd be readily available to make requested amendments to the documents to facilitate the creation of technical documentation through an improved developer and user documentation. I'll also be available to help the Season of Docs' student in creating required technical documentation.

• Resolve CCEXtractor Bugs

I've been involved in the CCEXtractor community for a year and have been passively involved in solving issues by either taking them up by myself or helping others. I'll keep this attitude in

continuity and will be an active contributor to the CCExtractor and related organisations' coding issues.

Requirements

- **Remote High-Speed Server**

A server is required for the purpose of testing the introduced commits as well as experimenting with different options. It'll also be used to keep test files and the samples to host an independent sample-platform apart from the production server.

- **Updated Results**

I'll improve the current set of test results we have on sample platform as several are outdated and hence lead to true negatives.

- **Wiki Edit Access [Already Have]**

This will be required to create detailed documentation of the changes introduced during GSoC period along with the creation of a personal GSoC page at [CCExtractor's homepage](#).

Personal Information

Personal Details

I'm Shivam Kumar Jha, an undergraduate student at Indian Institute of Technology, Kharagpur (India). I had been fascinated by programming since I was 14; aligned my work to always involve fun and programming at the same time.

My experience with open source has been very elaborate. It began with Kharagpur Winter of Code 2017 and quickly moved onto GSoC 2018 with CCExtractor. I have had the privilege to mentor in multiple open source programs such as GirlScript Summer of Code 2018 and Google CodeIn 2018. I've also been on the organising panel for GirlScript Summer of Code 2019 and Kharagpur Winter of Code 2018. Recently, I was invited to speak about open source at Pragma '19, organised by Indian Institute of Information Technology Allahabad,

I have apt knowledge of Algorithms and Data Structures and have been involved in competitive coding for a year now with codechef handle is [alphadollar](#).

I have a firm knowledge of C, C++, Python, Go, Java and experience with web development technologies such as HTML, CSS, and JavaScript. I'm highly influenced by Python's philosophy of "It's easier to ask forgiveness than permission" and have been using Python as my main language. I occasionally switch to other languages for specific features they provide, such as goroutines in go.

Working Environment And Schedule

I'll be mostly working full-time on the code on weekdays. On weekends, I'll be focusing on clearing any delay in the schedule, otherwise utilising it to communicate progress with my mentor. My awake hours would usually be in between 10 AM IST (4:30 AM UTC) to 2 AM IST the next day (8:30 PM UTC) and I'm comfortable working anytime during this period.

Except for a week or so of excursion (which I'll be informing in advance to my mentor), I'll be having no other absences. Anyhow, in cases of emergency, I'll responsibly notify my mentor of the same with enough detailing.

I'll be initially working from home where I've a constant internet connection. In the latter parts of the project, I'll be working from my college campus which provides unlimited high-speed internet.

Communication

I'm very flexible with my schedule and already have the habit of working at night and hence timezone variation (with my mentor) won't be an issue. I'm comfortable with any form of communication that suits my mentor. Below are the various options available:

- **Email:** shivam.cs.iit.kgp@gmail.com
- **Phone (Call and WhatsApp):** (+91) 7830380698
- **Hangouts:** shivam.cs.iit.kgp@gmail.com
- **Slack:** [thealphadollar](#)
- **GSoC blogs:** thealphadollar.github.io

Contributions to CCEXtractor And Sample Platform

I've been part of CCEXtractor for Google Summer of Code 2018 during which I worked on [Nephos](#). I also had the opportunity to be a mentor for Google CodeIn 2018 for the organisation. I've also been very active throughout my journey after joining the organisation, and have helped a lot of newcomers get acquainted with the community.

Below is the index of Github code contributions I've made to CCEXtractor and sample-platform.

● Issues Opened

Arrangement: Top is the oldest

- [\[PROPOSAL\] Compilation guide should have instructions to compile with OCR](#)
- [\[BUG\] Error while using "-hardsubx"](#)
- [\[GSoC 2018\]\[PROPOSAL\] Project Nephos Discussion](#)
- [\[PROPOSAL\] Show quantisation mode used](#)
- [\[BUG\] No option to compile with HARDSUBX using cmake](#)
- [\[BUG\] Hardsubx extraction from the sample stops at 11%](#)
- [\[PROPOSAL\] Send Issue Link And User In Issue Mail](#)

● Pull Requests Merged

Arrangement: Top is the oldest

- [\[IMPROVEMENT\] Update .gitignore](#)
- [\[IMPROVEMENT\] Code made compatible with Python 3](#)
- [\[IMPROVEMENT\] Modify -quant 0 option](#)
- [\[FEATURE\] Display quantisation mode](#)
- [\[FIX\]-nocf not working with OCR'ing](#)
- [\[IMPROVEMENT\] Add LICENSE File](#)
- [\[IMPROVEMENT\] Update COMPILATION.md](#)
- [\[FEATURE\] Allow build with hardsubx using cmake](#)
- [\[FIX\] Correct -HARDSUBX Bug In CMake](#)
- [\[IMPROVEMENT\] Update libGPAC](#)
- [\[IMPROVEMENTS\] Update CLI help and HARDBUX Installation Instructions](#)
- [\[IMPROVEMENT\] Refactor new issue mail code](#)
- [Update README to include GSoC '18](#)
- [\[FEATURE\] Shift to pipenv](#)
- [\[IMPROVEMENT\] Sanitize Install.sh Using Shellcheck](#)
- [\[FEATURE\] Add pep8](#)
- [\[FIX\] remove install dir from test coverage](#)

Post-GSoC Plans

I'm **not** applying for GSoC under any other organisation this year since I am motivated by the journey I've had till now in CCExtractor and I wish to contribute further being part of it.

I'll be looking forward to helping the student in the project for the web interface of rClone if the project is taken up in GSoC 2019. It was a project I had collected information about and had

done my research but I wanted to contribute to something integral to CCExtractor and that's why chose sample-platform.

Along with that, I'll always be a part of CCExtractor and will be following (as well as contributing towards) its development. I will always be available to make changes to Project Nephos (GSoC 2018) and sample-platform (GSoC 2019) and expand their feature set as and when need be.

References

In the research and making of this proposal, multiple sources have been used and they have been hyperlinked at the place of their usage for ease of reading and verification.

All the diagrams were made with the help of draw.io's interactive editor.

The timeline, methods, and technologies mentioned in the proposal are tentative, and a more robust discussion will be done when beginning each part of the task (deliverable).