GANGA

GRAPHICAL USER INTERFACE

PROPOSAL BY

VARUN BANKAR

Abstract

Ganga is a computational task-management tool that allows for the specification, submission, bookkeeping, and post-processing of computational tasks on a wide set of distributed resources.[1] It is used to process data effectively on inhomogeneous resources. Its functionality in most cases is accessed by the user with the help of a text-based command line in Python or file-based scripting. Having these two interfaces to interact with the tool might be sufficient to access all of its functionality but the lack of a proper Graphical User Interface (GUI) built into the tool costs the user time, decreases productivity and may degrade the user experience. Furthermore, a GUI will benefit the new users in the computing community in adopting the tool with relative ease.

Introduction

This project aims to change the user experience of the tool by implementing a Graphical User Interface (GUI) directly into Ganga, which could run alongside the existing text-based command line in Python. Implementation of the GUI will add an additional interface for the user to submit, monitor and manage the job. Moreover, this project is more than just an implementation of a GUI, the creation of Web APIs will add modularity to the existing software and give users an ability to integrate the tool in other applications, to pull jobs data and perform actions on it, locally as well as remotely. This project has 7 major components along with 1 experimental component, all of which will be implemented over the period of 3 months. The components are:

- Integration of a web server into Ganga.
- Creation of Web APIs for every possible interaction with the frontend.
- Creation of a clean, minimal looking, mobile responsive frontend UI.
- Connection of the frontend to the backend using AJAX requests.
- Improving the server security and implementation of authentication methods.
- Enhancing the reliability and responsiveness of the GUI and the Web APIs.
- Creation of unit tests and documentation of the features implemented.
- (Experimental) Integration of a CLI directly into the frontend UI.

Implementation of the above major components will be complimented with development of several minor features such as: monitoring sessions, browser notifications, and browser file navigation for navigating job folders.

Detailed information on the implementation process of the above features is discussed further in the proposal.

[1] Citation - https://twiki.cern.ch/twiki/pub/ArdaGrid/GangaGeneralJournal/ganga.pdf

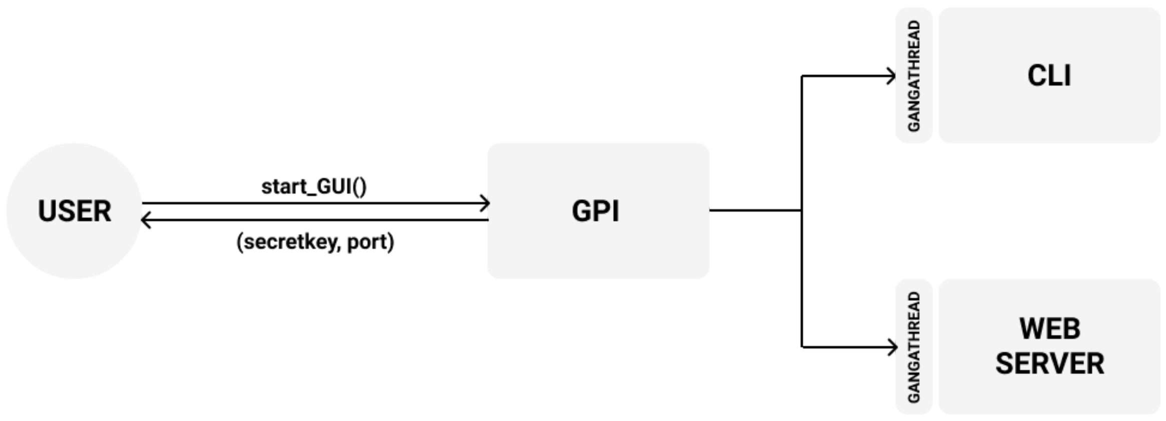
Project Components

Integration of Web Server into Ganga:

Most users of Ganga heavily rely on the text-based command line interface in order to create, customise, submit, monitor and post-process their jobs. It is therefore essential for the continuity of their workflow that after implementation/integration of the Graphical User Interface (GUI) they still have access to the text-based command line interface.

This will be achieved by running the web server on a separate 'GangaThread' in the background, which would enable the user to still have full access to the command line interface and as well run the web server in the background listening to all the web requests and running the GUI on a specific port.

For this functionality, the web server needs to be single threaded which means it can listen to only one request at a time, making it a practical choice for serving only a few clients at once. This is not a problem in context to Ganga as not much traffic is expected per Ganga session aside from users managing the jobs. But in case, if the GUI needs to be accessed by a large group of people, a Gunicorn server could be started which can handle multiple requests concurrently in order to serve the GUI. The user will not be able to access the command line interface with Gunicorn running as it uses 'gThread' for running the server.



After the integration, the user will be able to start the web server whenever needed using a simple command "start_GUI()" from the command line interface and as aforementioned, the web server will start in the background, listening to requests on the default port. The user can use another simple command "stop_GUI()" to stop the web server.

The intuitive commands to start and stop the web server will contribute to a better all round experience for existing Ganga users while making the software more accessible to a wider computing community.

Creation of Web APIs:

Web APIs are the most important component of the project. They will make the web server more than a server for Graphical User Interface (GUI) by adding modularity to the software, allowing it to be integrated into different applications and also be controlled by them.

Various endpoints will be created in order to share information with the client in JSON format. The APIs will have an endpoint for each of the following and more:

- Job
    - GET: Get crucial job data like attributes, parameters, status etc.
    - POST: Create a job using a pre-existing template.
    - PUT: Perform action on the job.
    - DELETE: Remove the job (if allowed).
- Jobs
    - GET: Get a list of jobs along with their data (can be filtered & sorted).
- Config
    - GET: Get configuration data (can be filtered).
    - PUT: Modify the configuration value (if allowed).
- Templates
    - GET: Get a list of all stored templates along with their data (similar to Job endpoint data).
    - DELETE: Remove the template.
- Tasks:
    - GET: Get a list of tasks along with their attributes and parameters.
    - PUT: Perform actions on the tasks.
- Credential Store:
    - GET: Get a list of CredentialStore data.
    - PUT: Renew a credential.
- Queues
    - GET: Get data related to user threads, monitoring threads, user queue and monitoring queue.
- Job Tree:
    - GET: Get information on job folders and their jobs.
- Box:
    - GET: Get data from Box Registry.
    - DELETE: Remove item from Box Registry (if allowed).

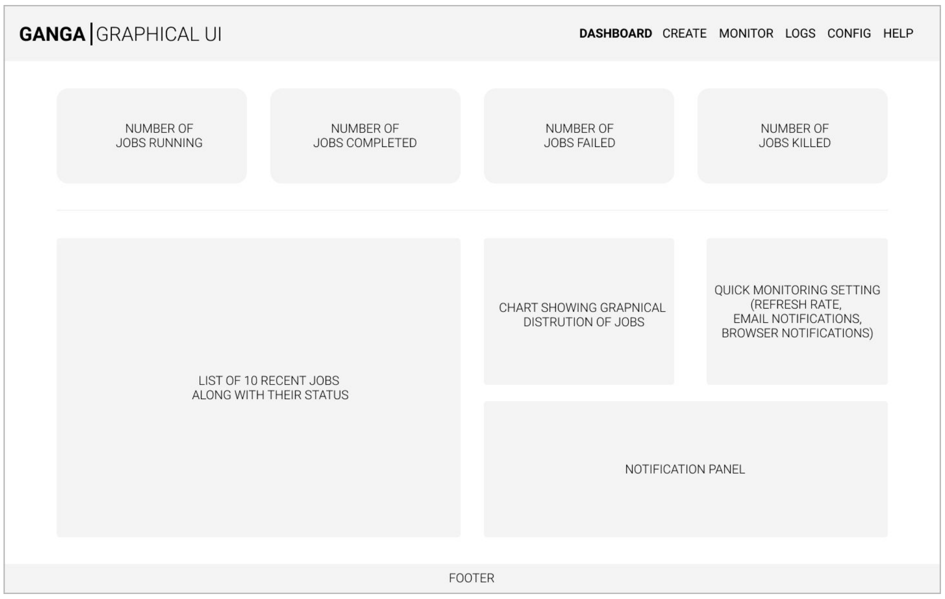
The above list of API endpoints can be further expanded, and any additional requirements will be discussed with the mentors during the Community Bonding Period.

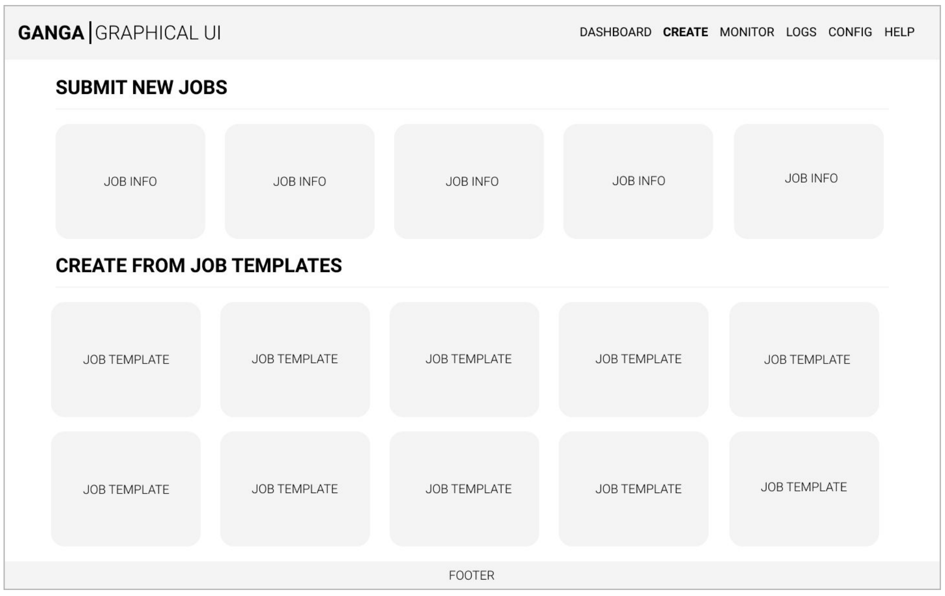
Creation of Frontend User Interface:

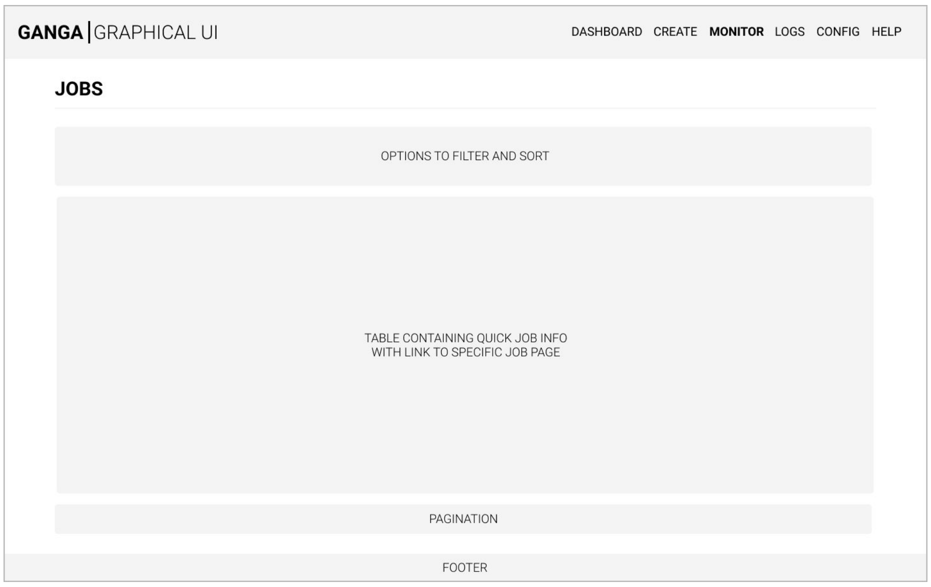
A clean, minimal looking, light-weight User Interface (UI) is very important for managing and monitoring thousands of jobs. The UI should not feel bloated and the user should be able to navigate through it with ease. The UI should feel intutivate so that users do not have a difficult time adopting/adapting to it.

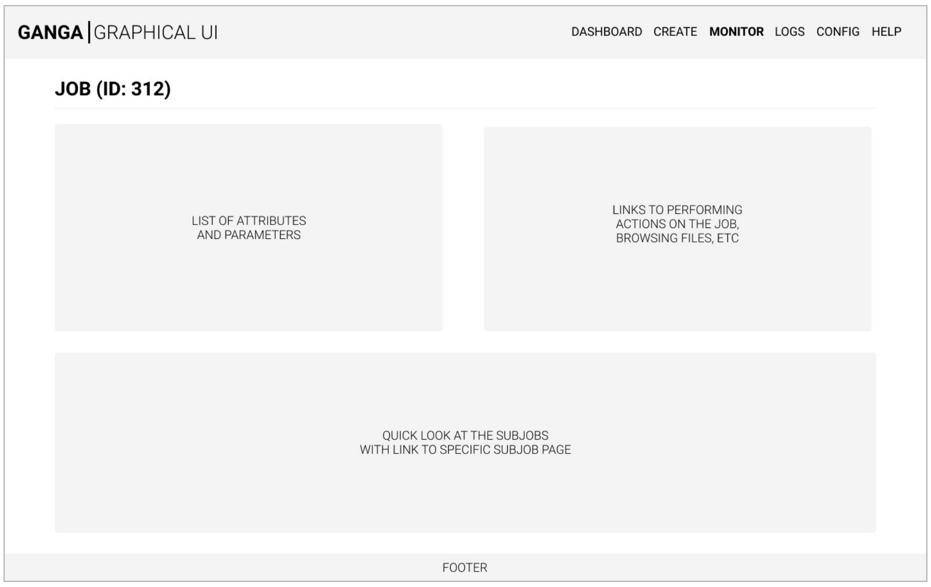
With the above specifications in mind, this project will use Bootstrap 4 for styling all of the frontend UI components so that they look clean, modern and are mobile responsive. Usage of Bootstrap 4 will ensure that the front end UI components work on most of the modern browsers. If needed, the UI components will be further customized and animated to improve the user experience.

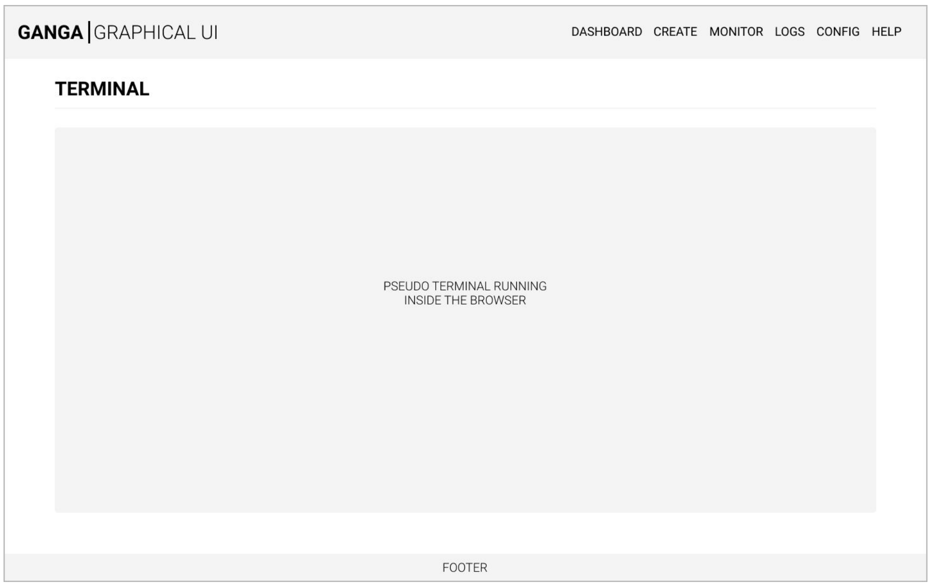
Following are few wireframe models of how the UI might look and any necessary changes will be done according to the mentor's suggestions.


GANGA | GRAPHICAL UI
DASHBOARD
CREATE
MONITOR
LOGS
CONFIG
HELP
NUMBER OF JOBS RUNNING
NUMBER OF JOBS COMPLETED
NUMBER OF JOBS FAILED
NUMBER OF JOBS KILLED
LIST OF 10 RECENT JOBS ALONG WITH THEIR STATUS
CHART SHOWING GRAPNICAL DISTRUTION OF JOBS
QUICK MONITORING SETTING (REFRESH RATE, EMAIL NOTIFICATIONS, BROWSER NOTIFICATIONS)
NOTIFICATION PANEL
FOOTER



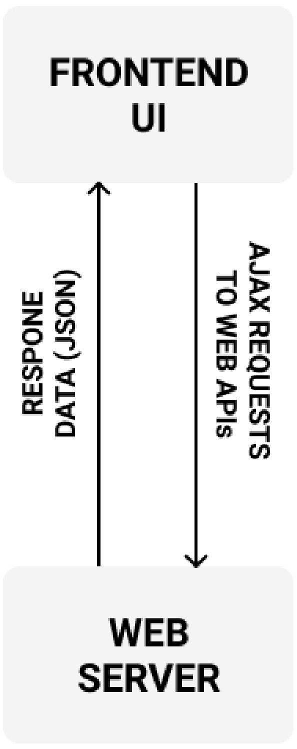

GANGA | GRAPHICAL UI
DASHBOARD CREATE MONITOR LOGS CONFIG HELP
JOBS
OPTIONS TO FILTER AND SORT
TABLE CONTAINING QUICK JOB INFO WITH LINK TO SPECIFIC JOB PAGE
PAGINATION
FOOTER


GANGA | GRAPHICAL UI
DASHBOARD CREATE MONITOR LOGS CONFIG HELP
JOB (ID: 312)
LIST OF ATTRIBUTES AND PARAMETERS
LINKS TO PERFORMING ACTIONS ON THE JOB, BROWSING FILES, ETC
QUICK LOOK AT THE SUBJOBS WITH LINK TO SPECIFIC SUBJOB PAGE
FOOTER


GANGA | GRAPHICAL UI
DASHBOARD CREATE MONITOR LOGS CONFIG HELP
TERMINAL
PSEUDO TERMINAL RUNNING INSIDE THE BROWSER
FOOTER

Connection of Frontend to the Backend:

Ganga users may need to submit and monitor thousands of jobs, and the Graphical User Interface (GUI) should not lag or become unresponsive while performing these tasks. To achieve this, the project takes a design decision to separate the frontend client completely from the backend server. Any interaction of the client with the server will be done through Asynchronous JavaScript requests to the server, making the GUI extremely smooth and responsive. The backend server will serve lightweight static files and once these files are loaded in the browser, AJAX requests will be made to the server through the web APIs and a promise will be returned containing the requested information in JSON format. This data would be parsed and displayed to the user. With this approach, the user does not need to wait for the server to process the information and then server the page, increasing the page loading time drastically.



Everytime the user performs an action through the GUI, a request will be sent in the background to the server, the server processes the request and returns the status back to the client which is then displayed to the user.

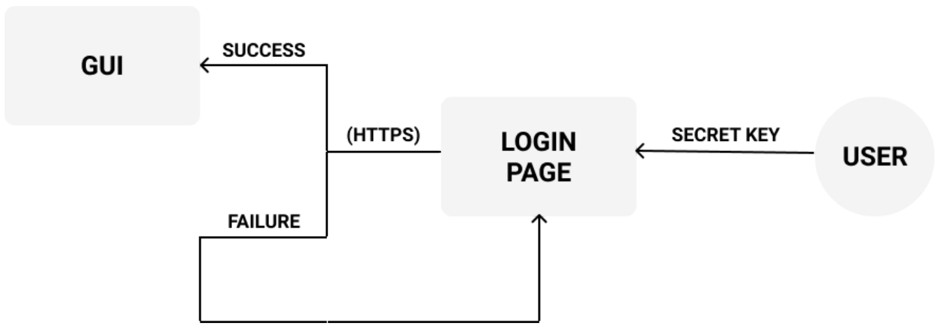
Security & Authentication:

Integration of the web server and web APIs in Ganga may expose important and sensitive data outside of the localhost machine for the user to access the Graphical User Interface (GUI) remotely. To ensure that this data will only be accessed by the user with proper credentials, the following steps will be taken to secure the web server and the web APIs.

Securing the Graphical User Interface (GUI):

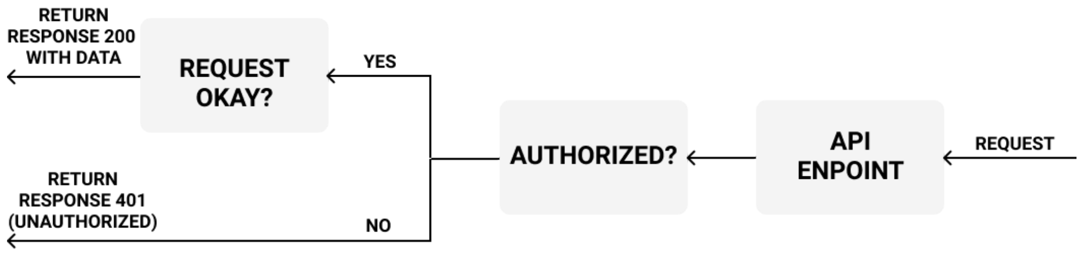
A login mechanism will be implemented in order to access the GUI. The access will only be given if the user enters a valid secret key at the login page. The secret key can be set by the user while starting the server. If not then, a secure secret key will be auto generated and displayed to the user. 'Flask Login' plugin will be used in order to implement this security feature. Once the user logs into the GUI with the valid secret key, a temporary cookie is stored in the browser of the user. Hence, the user will not need to login repetitively in order to access the GUI.

If the user provides a CA (e.g. letsencrypt) signed certificate or a self-signed certificate while starting the server, the connection between the client and server will be encrypted over HTTPS. Generation of a self-signed certificate is a manual process and can be done easily with the command line using openssl. With a self-signed certificate, the user will also need to trust it manually in the browser. All the instructions for the above will be provided in the documentation of this project.



Securing the Web APIs:

Similar implementation as for the GUI in order to keep the implementation semantic in nature for the user. But in case of web APIs, no cookie will be stored in the user's browser and the user will always need to provide the secret key to request the data from the server, failing of which will lead to return of status code 401 (Unauthorized).


RETURN RESPONSE 200 WITH DATA
REQUEST OKAY?
YES
AUTHORIZED?
API ENPOINT
REQUEST
RETURN RESPONSE 401 (UNAUTHORIZED)
NO

Testing Reliability & Responsiveness:

Ganga is used to manage a large number of jobs to process huge amounts of data. Henceforth, the GUI and web APIs need to be able to handle such a big load and work reliably. In order to ensure that, web APIs will be tested to submit, monitor and manage a large number of trivial jobs, emulating to some extent the load it might experience during practical usage. Furthermore, the mentor may help test the GUI and web APIs on an actual infrastructure, opening the door for optimisation of the two for practical usage.

Apart from testing the backend, the frontend will also be tested on various different browsers to check reliability and browser compatibility of UI components, AJAX requests and responsiveness on different browser types.

A few design decisions have already been taken in order to improve the overall responsiveness of the GUI, such as:

- Pagination wherever large numbers of rows are to be displayed. For example: A table displaying a full list of jobs (2000 jobs) will display only a few jobs (50 jobs) per page and the rest on the different pages (the user can decide how many jobs to display per page).
- The user can decide the time between the AJAX requests to the server for monitoring purposes.
- The user can pin the jobs, and monitor them separately on a different page, decreasing the number of AJAX requests and data transfer to and from the server.

As the project proceeds, there will be many opportunities for optimization which would then be discussed with the mentors.

Unit Tests & Documentation:

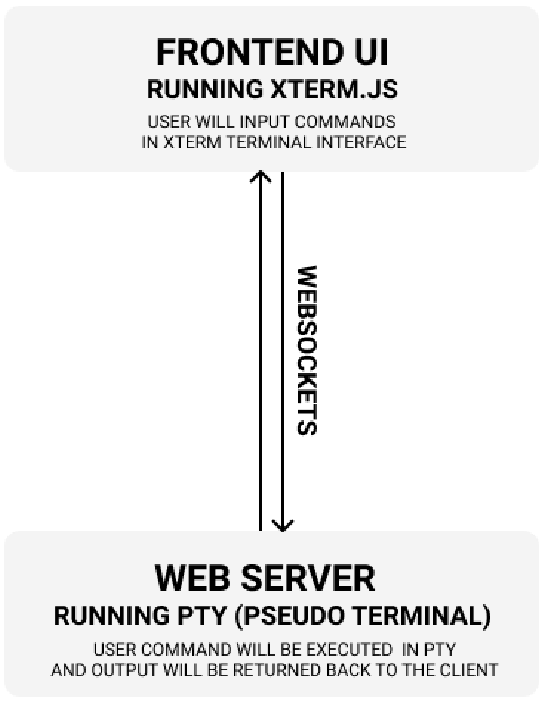
The whole project is structured with the goal for improving the unit testing workflow. Necessary unit tests will be created in order to test the functionality of the features implemented throughout the project.

All though this project aims to be intuitive to use for Ganga users. All implemented features (especially web APIs and working of the frontend client) will be completely documented along with necessary screenshots and video tutorials.

(Experimental) Integration of CLI in GUI:

After this project, Ganga users will have the ability to run the text-based command line interface and Graphical User Interface (GUI) at the same time. But in some cases, the user might need to access the command line from the GUI. One situation would be when the user is monitoring the jobs remotely, and needs to access the command line of the server running Ganga. One option would be to SSH into the server but this feature attempts to add an additional option by letting the user use the terminal directly from the browser without the need to SSH into the server. The user will need to log into the GUI before accessing this terminal, adding a similar level of security.

This feature will be implemented with the use of WebSockets, Xterm.js and pty to emulate a pseudo terminal inside the browser. Xterm.js will run on the client side giving the look and feel of running a terminal, the user commands will be communicated to the server in real time using WebSockets, which would then be processed by pty module by spawning a pseudo terminal and it's output would be relayed back to the client.


FRONTEND UI
RUNNING XTERM.JS
USER WILL INPUT COMMANDS IN XTERM TERMINAL INTERFACE
WEBSOCKETS
WEB SERVER
RUNNING PTY (PSEUDO TERMINAL)
USER COMMAND WILL BE EXECUTED IN PTY AND OUTPUT WILL BE RETURNED BACK TO THE CLIENT

Due to the uncertain nature of the deliverable, this feature has been marked experimental. But it does not mean any less efforts will be put on implementation of this feature. Sufficient time of the project will be dedicated in implementing this functionality and all the necessary details will be communicated to the mentors during its implementation.

Other Minor Components:

Monitoring Sessions

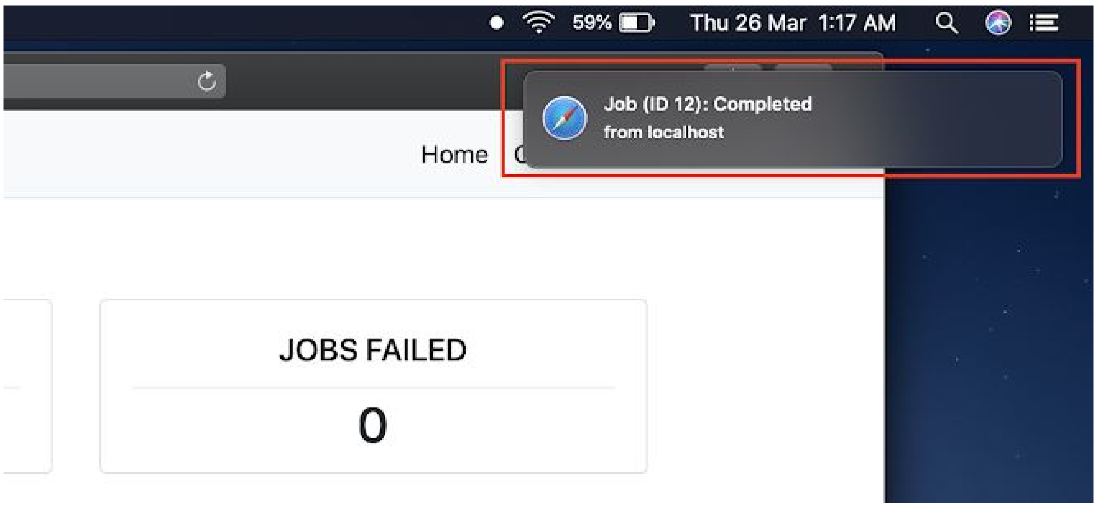
A database based feature which will let the user monitor only a few selected jobs (along with their subjobs) instead of the full jobs repository. Therefore, reducing the clutter, increasing the server response time and enabling the user to focus on monitoring of few selected jobs effectively. The user can have multiple such sessions with the selected jobs id’s stored in a file based Flask session. The GUI will only request information related to these job ids and display them to the user. The user can then choose to save this session to the database.

Job Input Directory & Output Directory Navigation (with Download)

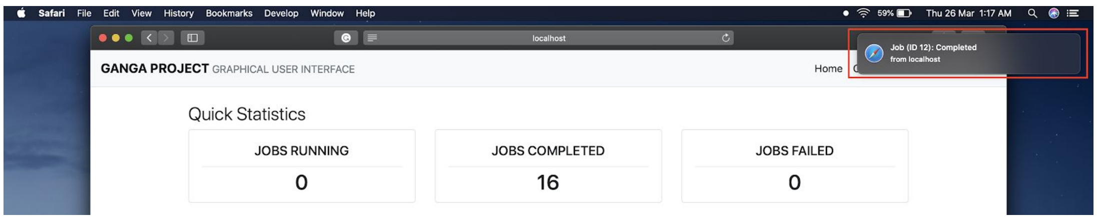
Every job will have a job page and from that the user can navigate through the Input and Output directory of the job in a graphical manner. The user can see all of the files of both the directories and may choose to download them if needed. This feature will be helpful in peeking into the job’s input and output directory remotely without the use of the command line interface. This feature will be implemented with the use of the ‘os’ library of Python and ‘send_file’ function from the Flask module.

Browser Notifications

The user may choose to receive a browser notification whenever the status of specific jobs changes or if a credential is about to expire. This feature will use JavaScript to access the Notification API of the browser and will only be available if the connection is over HTTPS. The user will need to keep the GUI running in the background. Following is an example of how it might look:



Job (ID 12): Completed
from localhost



Job (ID 12): Completed
from localhost

GANGA PROJECT GRAPHICAL USER INTERFACE

Quick Statistics

JOBS RUNNING	JOBS COMPLETED	JOBS FAILED
0	16	0

Key Deliverables

1. A secure Graphical User Interface (GUI) which could run alongside the text-based Command Line Interface (CLI) at the same time.
2. Ability to access the GUI remotely over HTTPS.
3. Secure and powerful Web APIs for creating, monitoring and managing jobs and other components, and integrating the software with other applications.
4. A clean, minimal looking, mobile responsive, lag-free frontend User Interface.
5. Ability to perform monitoring and managing tasks in real-time locally as well as remotely over a secure connection.
6. Complete documentation of all of the features implemented along with proper unit tests.
7. (Experimental) A working CLI accessible directly from the GUI.
8. Ability to create monitoring sessions, browser notifications, and to access job input and output directory.

Key Technology Used

- **Flask -** will be used as the main web framework (Python) for the project, and the Flask application will be directly integrated into Ganga.
    - Jinja2 - Templating language used by Flask.
    - Flask Login - For managing login sessions.
    - Flask Sessions - For managing client browser sessions.
    - Flask SQLAlchemy - For SQLite database to store monitoring sessions.
    - Gunicorn - WSGI server for serving Flask app for heavy web traffic.
    - Flask SocketIO - Socket.IO integration for Flask.
    - Flask RESTful - For building REST APIs.
- **Front End**
    - Bootstrap - For styling UI components.
    - Javascript - For making AJAX requests, WebSockets & Xterm.js.
- **Others**
    - OpenSSL - for creating self signed certificates.
    - Pty - for spawning pseudo terminal.
    - Pytest - for testing purposes.

Timeline

#	DATES	TASKS
1	04 May - 31 May Community Bonding Period	Connecting with the mentors, discussing the API endpoints requirement, setting up the development environment.
2	01 June - 14 June Week 1 & Week 2	Finish integrating the basic Flask app into Ganga, make necessary changes to the GPI for the web server, start implementation of the web APIs.
3	15 June - 28 June Week 3 & Week 4	Finish creation of all the endpoints of the APIs, test their functionality and document them with examples.
4	29 June - 03 July Phase 1 Submission	Submission of implementation of web server and web APIs.
5	29 June - 12 July Week 5 & Week 6	Designing and structuring a clean, minimal UI, optimising it for mobile, taking mentor's input regarding the design.
6	13 July - 26 July Week 7 & Week 8	Connecting every element of the frontend UI to the backend server using JavaScript, optimising for improved responsiveness.
7	27 July - 31 July Phase 2 Submission	Submission of fully working GUI, along with unit tests and documentation.
8	27 July - 09 August Week 9 & Week 10	Implementing security features, testing the backend and frontend for different workload and updating the documentation.
9	10 August - 23 August Week 11 & Week 12	Integrating CLI into GUI, adding unit tests for complete coverage, fixing unexpected bugs, finishing full documentation.
10	24 August - 31 August Phase 3 Submission	Final submission, with all of the key deliverables and their documentation.

Personal Information

General Information	
Full Name	Varun Balasaheb Bankar
Institute	Sophomore at Birla Institute of Technology and Science, Pilani - K.K. Birla Goa Campus
Course	Dual Degree in Computer Science & Chemistry
Contact Information	
Email Address	f20180295@goa.bits-pilani.ac.in / mail.varunbankar@gmail.com
Phone Number	+91 9503669327
Gitter Username	varunbankar
Address	AH3-335, BITS Pilani - K. K. Birla Goa Campus, Zuarinagar, Sancoala, Vasco Da Gama, Goa, India - 403726
Timezone	UTC+5:30
Github Link	https://github.com/varunbankar/

Why Me?

I am a self learning individual who loves to solve problems using technology. I have good experience in programming with Python, JavaScript, and using frameworks like Flask and Bootstrap which are needed very much for this project. I started my open source journey very recently with Ganga, and have been an active contributor to it. Moreover, the past few weeks have been absolute fun as I was identifying issues, diving into the code base, tinkering with it, solving a problem and submitting a Pull Request with documentation of changes made. It was a great overall learning experience. This process has made me very familiar with the code base and inner working of Ganga. I have also demonstrated provisional working of many of the above components to the mentors which ensures that I will be able to finish this project properly and in a timely manner.

Benefits to Community

This project aims to improve the user experience of Ganga users (mostly community of scientists) and increase their productivity by providing an additional interface to interact with Ganga. This project will provide scientists and researchers a Graphical User Interface (GUI) for creating, monitoring and managing jobs as well as performing other important tasks while adding little to zero learning curve in adapting to this new functionality. This project will enable the users to monitor and manage jobs remotely further enhancing their workflow. It will also benefit new and inexperienced users in the computing community in adopting/adapting to the tools with relative ease.

Why CERN-HSF?

Science has always fascinated me, to such an extent that one of my majors is in Pure Sciences. The contributions of CERN to science and the world are tremendous. To even get an opportunity to actually work on something that could be used by the scientists of such a prestigious organisation really excites me and is my main motivation for applying for this project. I am very eager to work on this project knowing that after it's implementation somewhere at CERN, someone would be interacting with the code that I wrote, and this really inspires me to deliver my best with this project.

Contributions to Ganga

Since February of this year, I have been an active contributor to Ganga and it has been a great learning experience for me, exploring and solving issues. Following is the list of my contributions to Ganga. In past few weeks, I have:

- Closed 3 existing Issues.
- Reported an issue in the functionality of a decorator.
- Solved the reported issue by extending the functionality of the decorator - **Link**
- Revived Ganga Tutorial module - **Link**
- Submitted over 10 Pull Requests (of which 10 have been merged into the main repository) - **Link**

I further plan to keep contributing to Ganga, solving issues and learning new things during the process. This is the beginning of my beautiful open source journey, and I wish to keep on contributing to the community.

Availability

I have my end semester examinations from May 01, 2020 to May 15, 2020 during which I will not be able to dedicate much time towards this project. Apart from that there are no commitments during the official coding period and will be able to give about 7-8 hours per day towards the completion of the project.

Post GSoC

This project has limitless opportunities for expansion and I would like to take advantage of these opportunities to learn and improve my skills, while also contributing to Ganga. I will keep implementing additional functionalities and optimising the existing ones in order to improve and enhance the GUI even further. It will benefit the community as well as allow me to get a hand on experience in contributing and maintaining a project which is used by a large number of people in the scientific community.

Having gotten familiar with the code base of Ganga, post Google Summer of Code, I would like to develop mentorship skills by helping new contributors navigate around the code base and by reviewing their contributions, so that I can give back to the community and help others learn.