Felucca

MITS CMU

Requirement Document

Project Felucca

Team BugHunter

Authors

Di Mu (dimu@andrew.cmu.edu)
Sudi Lyu (sudil@andrew.cmu.edu)
Zihao Zhou (zihaozho@andrew.cmu.edu)
Guancheng Li (guanchel@andrew.cmu.edu)

Revision History

Version	Date	Change	Updated by	Reviewed by
0.1	Mar 2nd, 2020	Add user behaviors and template of use case	Di Mu, Sudi Lyu	
0.2	Mar 9th, 2020	Add the initial version of use cases	Di Mu, Sudi Lyu, Guancheng Li, Zihao Zhou	Hasan Yasar
0.3	Mar 13th, 2020	Add use case diagram	Di Mu, Sudi Lyu	Jeffrey Gennari
0.4	Mar 27th, 2020	Add glossary	Sudi Lyu	
0.5	Apr 2nd, 2020	Update use cases Fix some contents	Di Mu, Sudi Lyu, Guancheng Li, Zihao Zhou	Hasan Yasar
0.6	Apr 6th, 2020	Add deployment mode	Sudi Lyu, Di Mu	Jeffrey Gennari
1.0	May 1st, 2020	Re-organize contents	Guancheng Li, Di Mu	

Contents

1 Project Context

Pharos is a set of tools, supporting **Program Analysis** and **Reverse Engineering**. It is open-source, developed, and maintained by the CERT Executable Code Analysis team. But the tools are not well-organized and do not integrate with each other, as Figure 1.1 shows below.

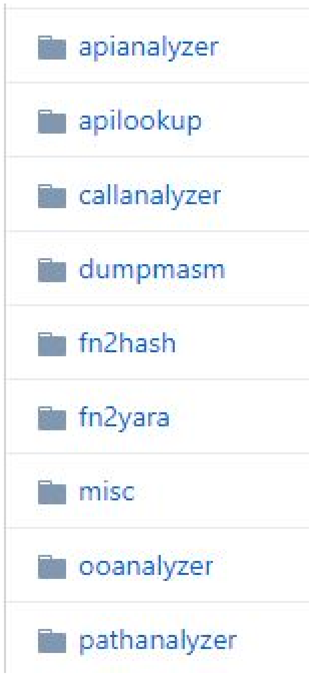

Figure 1.1 Pharos Toolset

The goal of our project is to develop a **Control Center** of Pharos toolset and to improve both its **integrity** and **usability**. Using our control center, users can submit jobs and retrieve the outputs after execution. Complex jobs other than single and simple execution are also supported. For example, the output of execution can be the input of the next execution inside a job, whose pattern is defined by the user. In other words, complex jobs with multi-execution and pipelined-execution are supported.

2 Users Behavior

We discussed the behaviors of three kinds of users and summarized them below.

2.1 Analysts (A)

They get a deliverable that contains the Felucca control center and Pharos Toolset by an offline medium (U disk). Then, they use the felucca to execute Pharos tools analyzing binary executable on their machine in a completely offline environment because they concern a lot about security. All the execution would be handled locally and the output is stored in a local database. Generally, users can update Pharos tools by downloading the latest version and put it on the right path. Since there may be no internet connection on their machine, the toolset update also needs to be done through an offline medium. If they have a local network that could access the latest Pharos tools, they could just update it in an online manner.

2.2 Researchers (R)

They have less concern about security, so they could use the Felucca control server using a web browser. They do not need any pre-install components on their computer. Their executable will be uploaded to the server and the output would be stored in a database on the server. So, the updated toolset would be automatically applied to the server, so they are always using the latest version of it.

2.3 Developers (D)

The developer will use the Felucca control center to develop and test the Pharos toolset. The developer would execute the binary executable locally and store the output in a local database. Moreover, the developer has full control access to the Pharos toolset that allows him to dynamically update it and run tests on it, which means that the developer's update on the toolset should be immediately applied to Felucca.

3 Glossary

- **Binary:** a binary is an executable file needed to be analyzed using tools, it is the input of tools
- **Tool:** a tool is a program which inputs a binary with arguments and then generates several outputs, it could be tools in Pharos or customized tools from users
- **Pharos tools:** the original pharos toolset provided by the developer, it contains multiple tools
- **Customized tools:** A customized tool provided by users, users must provide program and arguments information and output information
- **Task:** a task is an execution of a single tool with a single binary executable
- **Job:** a job is a collection of tasks and the relationship of tasks, the tasks in jobs need to be executed by some pattern determined by their relationships, the job is finished only if all the tasks are finished in a good manner
- **Status:** if a task is finished or not
- **Outputs of tools**:
  - **Log:** the log file generated by tools, the inner information of the execution
  - **Output(file):** the output file generated by tools, the outcome of the execution
  - **Command-line output:** the output of standard output of tools, explicit information of the execution

4 Three Deployment Mode:

- **Single-user offline mode:** Each user installs the Felucca in their machine and only themselves could use this installed version of Felucca, then the account is not needed and the execution and storage are local. The use of Felucca doesn’t demand network connection and updates of pharos could be done by Internet or physical medium.
- **Multi-user offline mode:** Felucca is installed in a machine and many users could use this installed version of Felucca, then the account is needed and the execution and storage are local. The use of Felucca doesn’t demand network connection and updates of pharos could be done by Internet or physical medium.
- **Online mode:** Users use Felucca by visiting the server, so an account is needed and the execution and storage area on the server. The use of Felucca requires network connection and the update of pharos is not needed since it is managed by the server.

5 Non-functional Requirements

- **Isolation**: As the binary files could be malicious, it is important to protect the host OS by isolating the analysis process. We decide to use Docker to achieve this. The reason is that Pharos tools are released in Docker-style.
- **Security**: Analysts may use Felucca on their devices without Internet access. To meet their requirements, Felucca must support the offline mode and offline updates of the Pharos Toolset.
- **Access Control**: Developers can update the Pharos toolset while researchers could only modify their own tools. Analysts won’t touch the source code of the tools.

6 Use Case

After knowing the requirements, we summarized them and created 12 use cases. The relationship between them is shown in Figure 6.1. And the details are listed in the tables below. For each use case, we list its flow, alternative flow, precondition, postcondition, exceptions, and requirements.

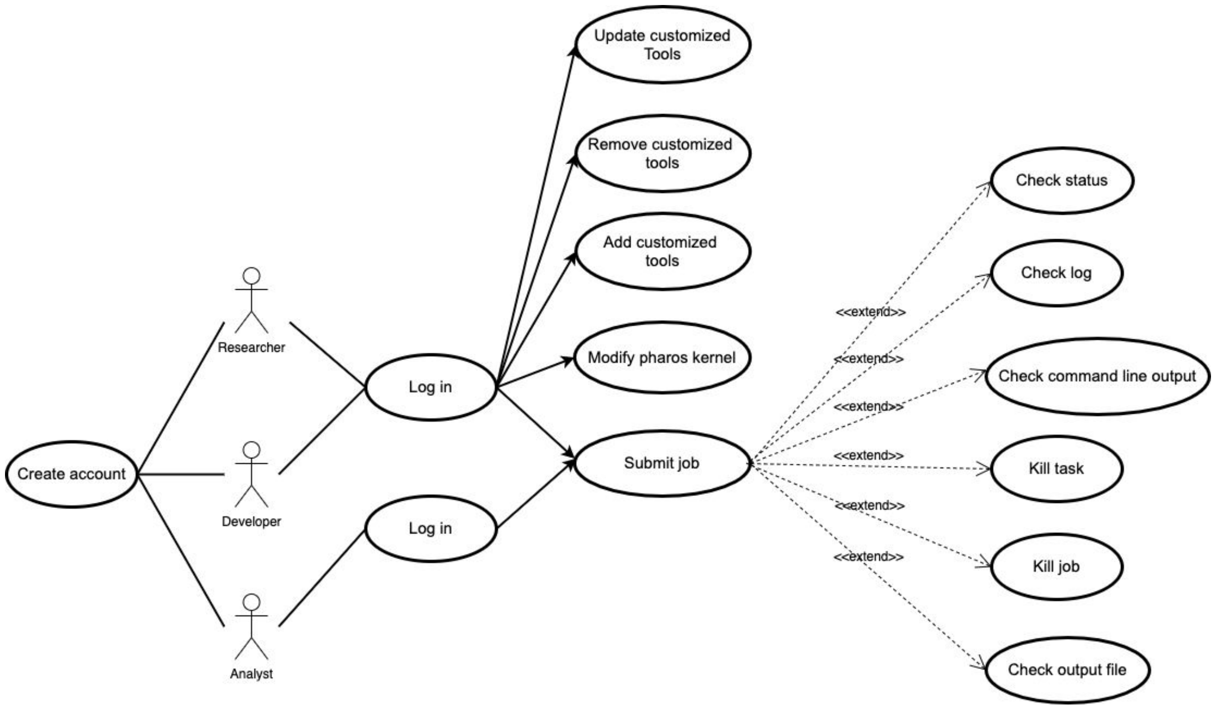

Figure 6.1 Use Case Diagram

Name of Use Case:	Create Account and Login		
Created By:	Sudi Lyu	Last Updated By:	Sudi Lyu
Date Created:	03/09/2020	Last Revision Date:	03/27/2020
Description:	Sign up a new account or Sign in an existing account		
Actors:	Research, Developer, Analyst (Users)		
Precondition:	1.If sign-in, users need to have an account 2.Database available for verification 3.Secured Environment		
Postcondition:	1. If sign up, the account information update into database 2. If sign in, the account information is fetched and showed to users 3. If the user forgets the password, new password update into database		
Flow:	1. Users go to the login page 2. Users click “sign up” 3. Users provide username, password and security questions		

	4.Felucca update information into the storage 5.Felucca redirect to login page 6. Users type in username and password than "sign-in" 7. Felucca verify the authentication information 8. Felucca fetch account information back to users
Alternative Flow:	2.If users have account, he could go to step 6 2.If users forget his password 1. users click "forget password" and provide username 2. Felucca fetch security question from database 3. Users provide answers for questions 4. Felucca verify answers, if wrong, decline 5. If right, users provide new password 6. Felucca update new password, then back to step 5 in main flow
Exceptions:	7.If username and password are wrong, Felucca shows error information and back to step 5
Requirements:	Username and password must be secured in database and in transmission
Prioritize	Nice to have

Table 6.1 Use Case for Create Account and Login

Name of Use Case:	Modify Pharos toolset		
Created By:	Sudi Lyu	Last Updated By:	Sudi Lyu
Date Created:	03/09/2020	Last Revision Date:	03/27/2020
Description:	The developers update the pharos tool in felucca		
Actors:	Developer		
Precondiction:	1.Developer provide new pharos toolset and new arguments 2.Developer are logged in		
Postcondition:	1. After update, the Felucca should execute the updated pharos 2. The updated pharos should be in database and could be used for next time 3. Support the new arguments in new pharos toolset		
Flow:	1.Developer entered update pharos toolset page 2.Developer provide new pharos toolset 3.Developer specify arguments 4.Developer check the sample command generated by Felucca 5. Developer submit the new pharos toolset		
Alternative Flow:	4.If developer think the command is wrong		

	1.Felucca back to step 3 to re-specify the arguments
Exceptions:	If submitted pharos tools is not a valid tools, Felucca report the error when job execution
Requirements:	1.Pharos tools need to be managed properly, and support for new arguments or new tools
Prioritize:	Must have

Table 6.2 Use Case for Modify Pharos Toolset

Name of Use Case:	Submit Job		
Created By:	Guancheng Li	Last Updated By:	Guancheng Li
Date Created:	03/13/2020	Last Revision Date:	04/21/2020
Description:	User submits a job to Felucca.		
Actors:	Researchers, Developer, Analyst (Users)		
Precondiction:	The user has logged in.		
Postcondition:	Felucca will run all the tasks of the job in order. The new job will appear in the record list and users can manage it. After the new job finishes, users can download its output and		
Flow:	1. User click the “Create New Job” button 2. User create a series of tasks, and for each task: 1. User upload a binary file 2. User choose a tool from the tool list 3. User type in/choose parameters (maybe checkbox) 3. User specify the execution order of these tasks 4. User click the “Submit” button 5. Felucca will launch a execution environment for the new job, or add it to the job queue		
Alternative Flow:	None		
Exceptions:	Creation of the new job fails because of some reasons and the user gets a notification like “Failed to create the new job because of …”. If a job fails in the middle of processing, users can be aware of it through “Check status”. Felucca will notify users if there are any invalid parameters.		
Requirements:	None		
Prioritize:	Must have		

Table 6.3 Use Case for Submit Job

Name of Use Case:	Add Customized Tools		
Created By:	Guancheng Li	Last Updated By:	Guancheng Li
Date Created:	03/13/2020	Last Revision Date:	03/27/2020
Description:	Developers or researchers can add their own tools.		
Actors:	Research Developer		
Precondiction:	User has logged in		
Postcondition:	The new tool will appear in the tools list		
Flow:	1. User click the “Add New Tool” button 2. User upload an executable file, which is the new tool 3. User specify the name 4. User click the “Confirm” button 5. Felucca analyzes the path and adds the new tool to database		
Alternative Flow:	None		
Exceptions:	The path can’t be recognized as a tool. Felucca will report an error		
Requirements:	Database should record the name, the parameters and the path of all tools.		
Prioritize:	Optional		

Table 6.4 Use Case for Add Customized Tools

Name of Use Case:	Update Customized Tools		
Created By:	Guancheng Li	Last Updated By:	Guancheng Li
Date Created:	03/27/2020	Last Revision Date:	03/27/2020
Description:	Developers or researchers can update tools that are added by themselves.		
Actors:	Research Developer		
Precondiction:	User has logged in. The tool is added by the current user.		
Postcondition:	The tool will be updated.		
Flow:	1. User click the “Update Tool” button 2. User choose a tool from the list of current tools, which must be added by the current user		

	3. User can upload an executable file, which is the new version of tool 4. User can change the name of the new tool 5. User click the "Confirm" button 6. Felucca analyzes the path and update the file of the specified tool to database
Alternative Flow:	None
Exceptions:	None
Requirements:	Database should record the name and the path of all tools.
Prioritize:	Optional

Table 6.5 Use Case for Update Customized Tools

Name of Use Case:	Remove Customized Tools		
Created By:	Guancheng Li	Last Updated By:	Guancheng Li
Date Created:	03/13/2020	Last Revision Date:	03/27/2020
Description:	Researchers and developers can remove tools that are added by themselves.		
Actors:	Researcher & Developer		
Precondiction:	User has logged in. The tool can be seen in the tool list. The tool is added by the current user.		
Postcondition:	The tool is no longer visible from the tool list. The metadata related to this tool is removed from Felucca's database.		
Flow:	1. User click "Tools" button and see the list of current tools 2. User click "Remove a Tool" button and checkboxes appear for tools added by the current user 3. User check all tools that he/she wants to remove 4. User click "Confirm" button and checked tools will disappear from the list 5. Felucca will remove the tools and clean the files		
Alternative Flow:	None		
Exceptions:	Tools cannot be removed because of some reasons. User would get a notification like "Removing xxx tool failed because of …"		
Requirements:	Tools from Pharos can't be removed.		
Prioritize:	Optional		

Table 6.6 Use Case for Remove Customized Tools

Name of Use Case:	Check Status		
Created By:	Zihao Zhou	Last Updated By:	Zihao Zhou
Date Created:	03/13/2020	Last Revision Date:	03/27/2020
Description:	User can check the current status when job is running or done		
Actors:	Researchers, Developers, Analysts		
Precondition:	The user has logged in and submitted a job.		
Postcondition:	1. The current status of tasks in a job will show up on the user's interface. 2. If a task in job is finished, output the finished sign of that task 3. If a task is still running, output the progress status of this task. 4.If the job is done, output the finished sign of the job		
Flow:	1. User click "Check Status" button 2. Felucca will output status of all the tasks in the job		
Alternative Flow:	None		
Exceptions:	1. If some tasks are done but the its log file is too large, it may take times to load the log file		
Requirements:	None		
Prioritize:	Nice to have		

Table 6.7 Use Case for Check Status

Name of Use Case:	Check Command Line Output		
Created By:	Zihao Zhou	Last Updated By:	Zihao Zhou
Date Created:	03/13/2020	Last Revision Date:	03/27/2020
Description:	User can check the command line output during the job is running or after the jobs are finished		
Actors:	Researchers, Developers		
Precondition:	The user has logged in and submitted a job.		
Postcondition:	The current existed stdout of tasks in the job will show up		
Flow:	1. User clicks "Check Command Line Output" button 2. Felucca will output the current stdout of tasks in the job 3. The user scans the stdout in browser		

Alternative Flow:	1. If the Researchers and Developers want to download the command line output file, click ‘Download CMD Result’ button
Exceptions:	1. If there is no task done, the list after user clicking “Check Command Line Output” will be empty
Requirements:	None
Prioritize:	Nice to have

Table 6.8 Use Case for Check Command Line Output

Name of Use Case:	Check Output File		
Created By:	Zihao Zhou	Last Updated By:	Zihao Zhou
Date Created:	03/13/2020	Last Revision Date:	03/27/2020
Description:	User can check the final output after the jobs are finished		
Actors:	Researcher, Developer, Analyst		
Precondition:	The user has logged in and submitted a job.		
Postcondition:	The output of finished tasks in a job will show up		
Flow:	1. User clicks “Check Output” button 2. Felucca will list all the output files of finished tasks in this job. 3. The user clicks one of them. 4. The user scans the log file in browser		
Alternative Flow:	1. If a user want to download the final output file, click Download Result button		
Exceptions:	1. If there is no task in this job done, the list after user clicking “Check Output” will be empty 2. If the task’s output is too large, it may take times to load the log file		
Requirements:	None		
Prioritize:	Must have		

Table 6.9 Use Case for Check Output File

Name of Use Case:	Check Log		
Created By:	Di Mu	Last Updated By:	Di Mu
Date Created:	03/12/2020	Last Revision Date:	03/27/2020

Description:	Check the log content for a task
Actors:	Researcher Developer Analyst (User)
Precondition:	1. The user has logged in 2. There should be a task in a job either finish running or is running in pharos 3. The user should provide parameters which indicate the log level
Postcondition:	1. The log context can be viewed in browser 2. The user gets the log file after download
Flow:	1. User selects a running or finished task. 2. User clicks the check log button 3. User scans the log file in browser
Alternative Flow:	2a. If the user wants to download the log file, click download log file button 2b. If the user wants to delete the log file, click delete log file button
Exceptions:	1. If the task does not start, the log file will be empty, Felucca will show a hint window saying “the task is not running yet” 2. If the log file is too large, it may take times to load the log file
Requirements:	None
Prioritize:	Must have

Table 6.10 Use Case for Check Log

Name of Use Case:	Kill Job		
Created By:	Di Mu	Last Updated By:	Di Mu
Date Created:	03/12/2020	Last Revision Date:	03/27/2020
Description:	Kill a running job user specified		
Actors:	Researcher Developer Analyst (User)		
Precondition:	1. the user should log in 2. there should be a running job		
Postcondition:	1. the job should be terminated		
Flow:	1. User opens the job status page 2. The user selects a running job 3. User clicks kill job button		
Alternative Flow:	3a. If the user wants to restart the killed job, after the job status is shown as terminated, they can click the restart button		

Exceptions:	1. if the job takes a long time to be killed, the status page should show “terminating”
Requirements:	3a. All tasks under this job should be terminated 3b. All the resources related to this job should be cleaned 3c. All records related to this job in the database should be removed.
Prioritize:	Must have

Table 6.11 Use Case for Kill Job

Name of Use Case:	Kill Task		
Created By:	Di Mu	Last Updated By:	Di Mu
Date Created:	03/27/2020	Last Revision Date:	03/27/2020
Description:	Kill a running task user specified		
Actors:	Researcher Developer Analyst (User)		
Precondition:	1. the user should log in 2. there should be a running task in a job		
Postcondition:	1. the task should be terminated		
Flow:	1. The user opens the task status page from a job status page 2. The user selects a running task 3. The user click kill task button		
Alternative Flow:	None		
Exceptions:	3a. if the task takes a longer time to be killed, the status page should show “terminating”		
Requirements:	3a. All the resources related to this task should be cleaned 3b. All records related to this task in the database should be removed.		
Prioritize:	Nice to have		

Table 6.12 Use Case for Kill Task

7 Appendix

7.1 Functional Requirements

Based on the use cases listed above, we break them into functional requirements of different modules. When we start to implement the modules, these can be great references and we can make the development plan according to them.

Create Account

[CRAC-001]Create account page, check username password format, call resource manager
[CRAC-002]Check validation, store into the database, return status to front-end

Log in

[LOIN-001]Create Login page, call resource manager
[LOIN-002]Check validation, return status to front end

Update Pharos Toolset

[UPPH-001]Add an “Update pharos tools” button and upload file page
[UPPH-002]Submit updated Pharos
[UPPH-003]Store the updated Pharos

Submit job

[SUJO-001]Add a “Create New Job” button and the according functions
[SUJO-002]Upload all information about the new job after user click "Submit"
[SUJO-003]Sequentialize the order of the tasks of a job
[SUJO-004]Store the metadata of the submitted job through Job Metadata Manager
[SUJO-005]Call Execution Manager to start a task
[SUJO-006]Update the metadata through Job Metadata Manager after a task finishes
[SUJO-007]Check if the specified tool is from Pharos toolset or not, and do the according operations
[SUJO-008]Run the task using the specified tool and parameter
[SUJO-009]Notify the Job Manager and store the output through Resource Manager after a task finishes
[SUJO-010]Store the metadata of a new job
[SUJO-011]Provide the latest version number of the Pharos toolset (and the image if necessary)
[SUJO-012]Provide the path of customized tools
[SUJO-013]Store the output of the finished task

Add Customized Tools

[ADCU-001]Upload the executable file of the tool with its name
[ADCU-002]Display the list of tools after successful addition

[ADCU-003]Store the executable file and insert its name with the path into the database
[ADCU-004]Query and return the list of available tool

Update Customized Tools

[UPCU-001]Request and display the list of available tools
[UPCU-002]Display the template of the customized tools with a “Submit” button
[UPCU-003]Upload the information of the new tool
[UPCU-004]Display the list of all tools
[UPCU-005]Store the executable file and update its name with the new path in the database
[UPCU-006]Return the list of all tools after updating a customized tool

Remove Customized Tools

[RECU-001]Request and display the list of available tools, with a “Delete” button next to each customized tool
[RECU-002]Display a dialog with two buttons, “Confirm” and “Cancel”
[RECU-003]Send the remove request of a specific tool
[RECU-004]Return the list of all tools
[RECU-005]Remove the specified customized tool and return the list of available tools after deletion

Check status

[CHST-001]Add a function that users can request and display the list of jobs. After the user selects and clicks one specific job, display the tasks in that job tools
[CHST-002]Add a function that users can request and display the list of tasks in a specific job after the user selects and clicks one specific task, display a function “Check status output”.
[CHST-003]Add “Check status output” button for each task. After the user clicks this button, Front-end will request Resource Manager for the corresponding status output file. After that, the browser will display the template of status output.
[CHST-004]Receive “Check status output” request from Front-end, get status file from database, return to the Front-end.
[CHST-005]Add “Download status output” button for each task to download the command line store file to the user.

Check command line output

[CHCO-002]Add a function that users can request and display the list of tasks in a specific job, after the user selects and clicks one specific task, display a function “Check command line output”.
[CHCO-003]Add “Check command line output” button for each task. After the user clicks this button, Front-end will request the Resource Manager for the corresponding command line store file. After that, the browser will display the command line output.
[CHCO-004]Receive“Check command line output” request from Front-end, get command line store file from database, return to the Front-end.
[CHCO-005]Add “Download command line output” button for each task to download the command line store file to the user.

Check output file

[CHOU-002]Add a function that users can request and display the list of tasks in a specific job, after the user selects and clicks one specific task, display a function “Check output”.
[CHOU-003]Add a “Check output” button for each task. After the user clicks this button, Front-end will request the Resource Manager for the corresponding output file. After that, the browser will display output.
[CHOU-004]Receive “Check output” request from Front-end, get output file from database, return to the Front-end.
[CHOU-005]Add the “Download output” button for each task to download the output file to the user.

Check Log

[CHLO-001]Request and display the job list, Add “Check log” button for each task
[CHLO-002]Add a url and page template to show the log file, After click the button, request Resource Manager for the log data
[CHLO-003]Get log file from database, return to the front end
[CHLO-004]Add “Download” button, download the log file to user

Kill Job

[KIJO-001]Request and display the job list, Add “Check log” button for each task
[KIJO-002]Request the Job Manager to kill the job, show corresponding status once killed.
[KIJO-003]Get the task list to be killed based on metadata, call execution manager to kill the task, send status back to Front-end
[KIJO-004]Kill the corresponding task, Send status to Job Manager, call Resource Manager to update status.
[KIJO-005]Update job metadata for the killed job

Kill Task

[KITA-001]Request and display the job list, add a “Kill”/”Cancel” button for each task
[KITA-002]After clicking the button, request the Job Manager to kill/cancel the corresponding task, show corresponding status once deleted
[KITA-003]Handle exception when kill the task is a dependency for other task
[KITA-004]Kill/cancel the corresponding task, send status to Job Manager, call Resource Manager to update status
[KITA-005]Update job metaData for the job