Mission Support System: Collaborative editing of flight path in real-time

About Me

Name: Shivashis Padhi, (Github/Bitbucket/IRC username) - plant99, mss-devel.slack.com - shiv
University info:

> University Name: National Institute of Technology, Tiruchirappalli
> Major: Computer Science and Engineering
> Current Year: III year
> Expected Graduation date: May 2020
> Degree: Bachelor of Technology

Contact info:

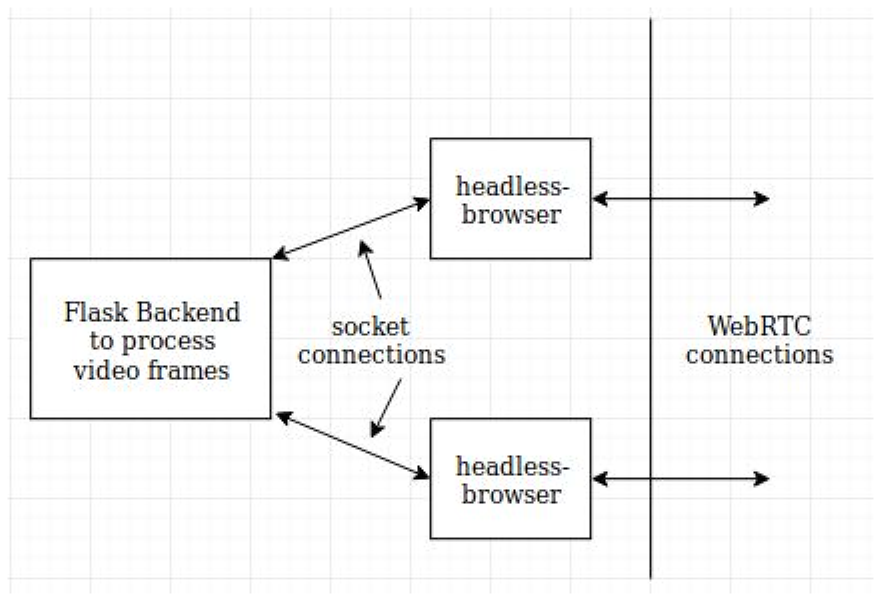
> Homepage: https://plant99.github.io
> Email: shivashispadhi@gmail.com
> Alternate Email: 106116085@nitt.edu

Time Zone: Indian Standard Time (GMT + 5:30 hours)
Link to resume: https://plant99.github.io/files/resume_shivashis_padhi.pdf

Why I chose this idea to submit a proposal on?

Having written ~7000 lines of code in Python, I can safely say it's my favorite programming language. So I had no doubt that contributing to a Python based project would be the easiest and most rewarding for me. PSF was my first priority when I planned to apply for GSoC. Besides computer science, I spend a good amount of my spare time browsing through research updates in the area of Geo-sciences. Thus, when I stumbled upon mss, I decided to submit a proposal for one of its ideas.

I have around 2 years of experience with development of real-time applications as a student developer. In the past, I have worked with chat and notification modules, some of which were deployed to production logged ~680 users and >25000 games requests during Code-Character 2019, which was online for 21 days. This proposal has similar algorithms incorporated to handle connections and network I/O.
During my internship at Flytbase, I designed and developed a video-chat platform for human and drone clients, with different permission levels to publish/view the streams using core concepts of WebRTC. A big issue was to build an interface for an onboard device like Raspberry PI to publish stream over WebRTC, because there were no reliable python clients for WebRTC. I wrote a python client with websockets, asyncio, and Flask, to receive a stream from web-browser, process the frames, and send it back to web-browser through websockets and eventually WebRTC clients. The architecture is represented in the following illustration:



Certainly, the best of my work till date as a student developer involved concepts related to this project. I look forward to making the best use of my experience, interests, and abilities to work with MSS on this idea and contribute a lot to open source over the summer, if given a chance.

Code Sample

From the time PSF's sub-org list was released, to this date, I have worked with mentors from MSS to discuss issues/features and successfully got my work accepted. Following is a list of links to PRs I have worked on, sorted by date.

Note: PRs marked with a * are major ones.
https://bitbucket.org/wxmetvis/mss/pull-requests/597/fix-pep8-indentation-exception/diff
https://bitbucket.org/wxmetvis/mss/pull-requests/599/renamed-_tests-utilspy-to-_tests/diff
https://bitbucket.org/wxmetvis/mss/pull-requests/601/update-gitignore-for-project-structure/diff
https://bitbucket.org/wxmetvis/mss/pull-requests/603/add-delete-and-insert-functionalities-in/diff *
https://bitbucket.org/wxmetvis/mss/pull-requests/604/fix-pep8-trailing-space-exception/diff
https://bitbucket.org/wxmetvis/mss/pull-requests/607/show-approximate-coordinates-of-point-on/diff *
https://bitbucket.org/wxmetvis/mss/pull-requests/609/enhancement-of-insert-waypoint-function-to/diff *
https://bitbucket.org/wxmetvis/mss/pull-requests/617/stable/diff
https://bitbucket.org/wxmetvis/mss/pull-requests/616/i324/diff *

Project Information

Sub-org name:

Mission Support System - MSS

Project Abstract:

Mission Support System is a flight planning software which a researcher can use to analyze predicted atmospheric data, and plan a flight-path with 3D waypoints. The software in the present state allows editing by a single user per flight-path. To share this work, one has to export the work as a `$name.ftml` file and send it to other researchers for further planning. This back and forth communication not only consumes a lot of human efforts and time, but also can be frustrating when the number of researchers involved in a project is bigger, say >=3.

I propose a solution to this problem, the development of `Mscollab` which stands for 'Mission Support Collaboration'. `Mscollab` would facilitate *real-time*, *collaborated* editing of flight-paths by *authorized* users. By design, it will also provide a chat facility for the users who collaborate on the project. Its UI would be a part of *msui,* the core User Interface module of mss. It will additionally provide insights about changes and the users who created the changes, for analytics purpose. `Mscollab-server` will be a standalone server built with Python, Flask, and python-socketio.

Project Description:

A brief introduction to MSS software:

Mission Support System provides a suite which can be used by atmospheric research scientists to plan a research flight. It has two essential components

- msui
  It's the core UI module of mss software, this shows atmospheric data on a 2D map. A researcher has to analyze the data and mark the waypoints (a set of points which the research aircraft has to pass through to collect experimental data).
- wms/mswms
  WMS stands for Web Map Server, which essentially serves maps given origin coordinates and other geographic parameters. A sample wms server can be setup with 'mswms', and some demo data which can be setup easily as per instructions here. In a single sentence, mswms serves the data to be displayed by msui.

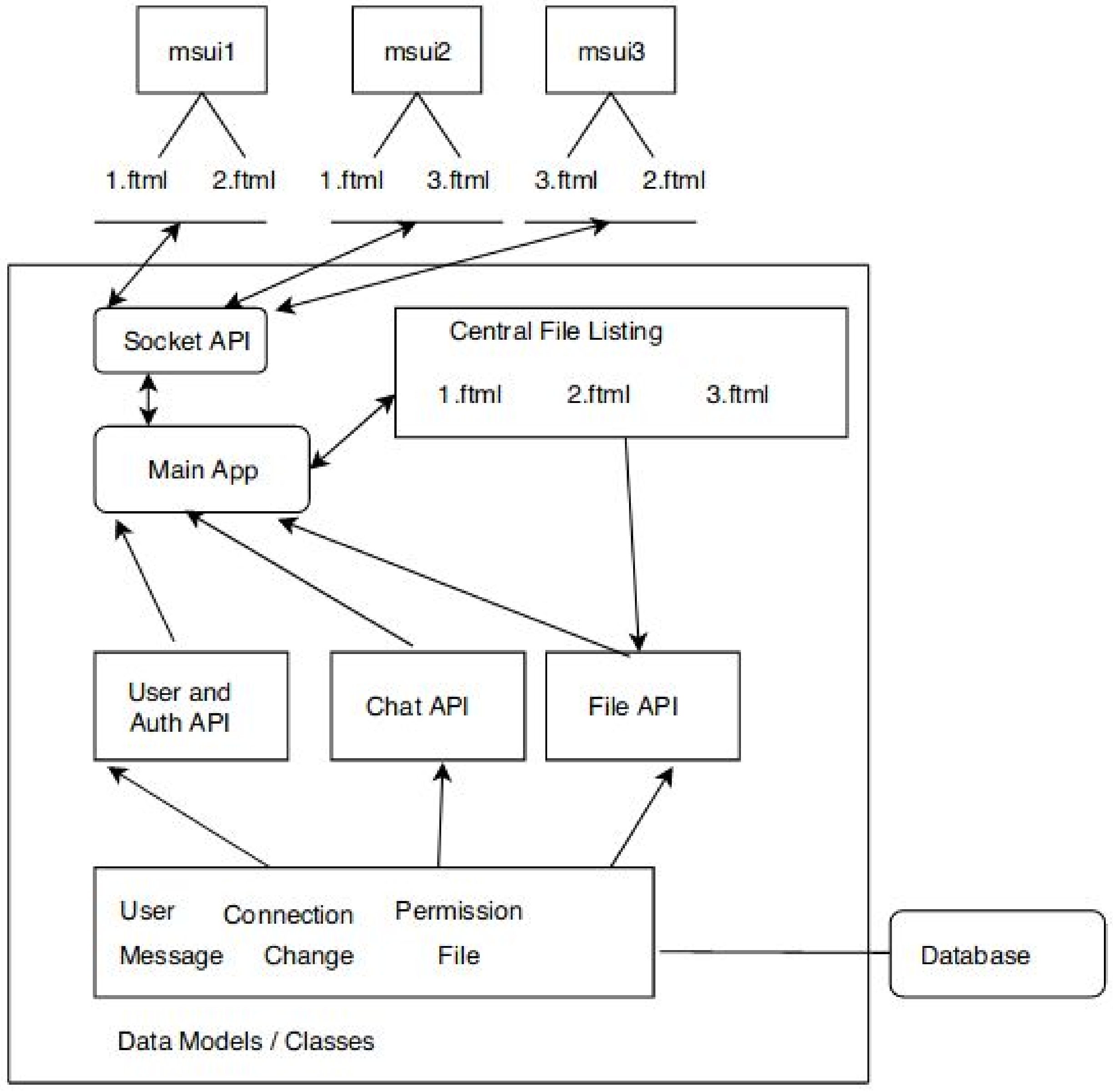
In the current state of mss, the waypoints created by a researcher is volatile in nature. To save it, one has to export the waypoints to a '.ftml' file (by default) which is saved and accessed by user through fs API.

- So, to share the flightpath with other researchers, one has to use an external channel. This can cause considerable waste of time and efforts when working in a big team.
- Plus, the changes made by members get mixed up which creates a lot of confusion down the line.

Thus, a collaboration tool is highly necessary to synchronize individual work of group members and keep track of changes made in the process.

Mscollab:

`Mscollab's` design solves the aforementioned problems, in a simple and structured fashion. Following is a helicopter view of the same.

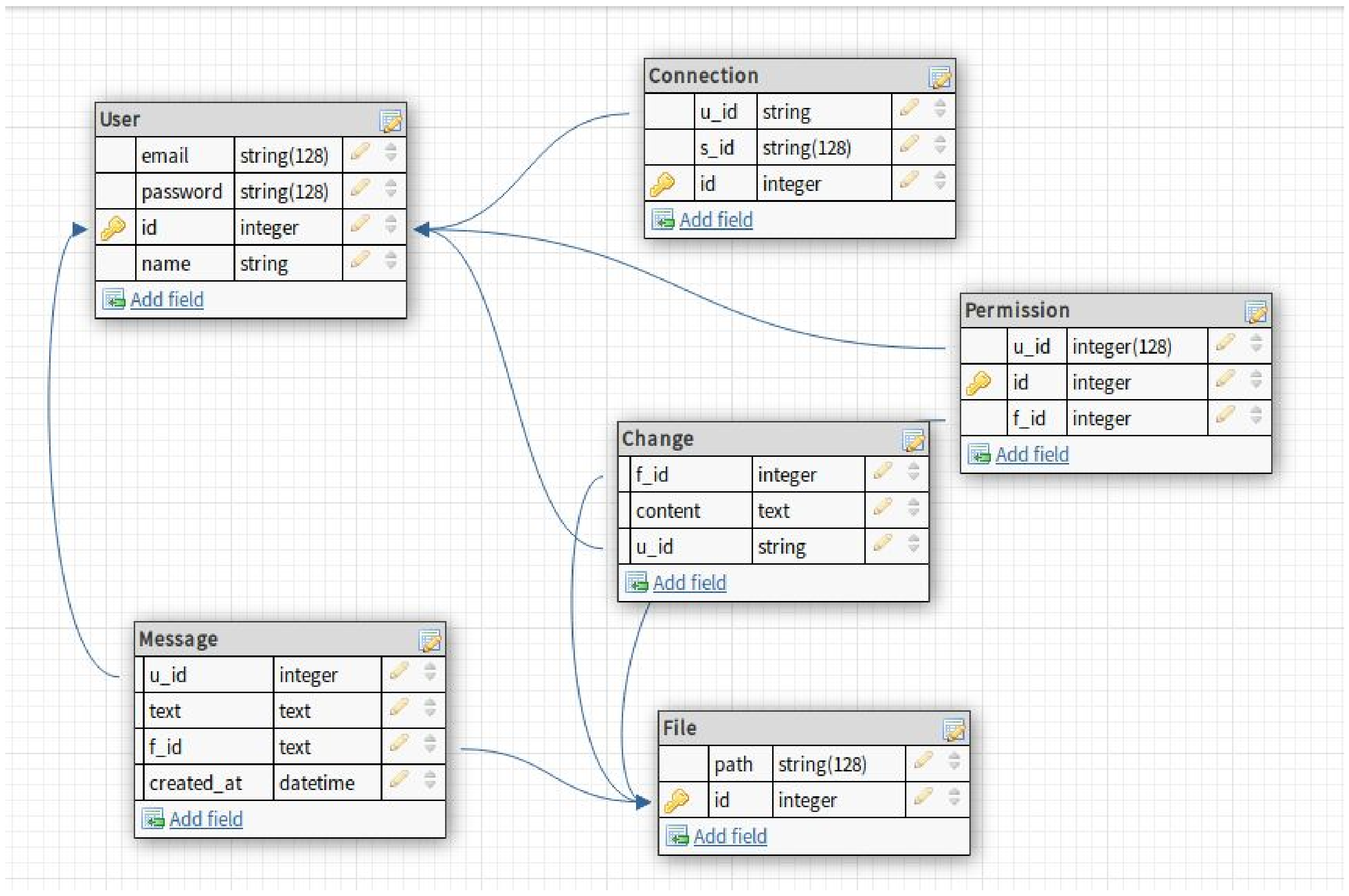


`Mscollab` has four major components:

- Data Models / Classes
- Main App
- APIs
- UI

Among these, the first three modules would be developed within `Mscollab` server. UI of this software would be integrated with msui module of mss software.

Data Models



Connection	
u_id	string
s_id	string(128)
id	integer

User	
email	string(128)
password	string(128)
id	integer
name	string

Permission	
u_id	integer(128)
id	integer
f_id	integer

Change	
f_id	integer
content	text
u_id	string

Message	
u_id	integer
text	text
f_id	text
created_at	datetime

File	
path	string(128)
id	integer

The project needs six basic models named:

- <u>Connection</u>
  This class is defined to keep track of the socket connections maintained by users.
  u_id: user-id
  s_id: socket-id

- <u>User</u>
  This class represents an user of `mscollab`.
  id: user-id
  email: email-id of the user
  name: Screen name of the user
  password: bcrypt hashed password

- <u>Permission</u>
  This data represents the authorization of an user to collaborate on a file.
  u_id: user-id
  f_id: file-id
  access_level: enum[“admin”, “collaborator”, “viewer”]

- Message
  Message represents a single message unit, linked with a chat(further linked to a file).
  u_id: user-id
  f_id: file-id
  text: message content
  created_at: used to order the messages in a conversation

- Change
  Change represents a change in .ftml file submitted by an authorized user.
  u_id: user-id
  f_id: file-id
  content: ‘diff’ of file made by the user
  created_at: used to sort the changes w.r.t time

- File
  File represents an .ftml file.
  id: to be used as fileId
  path: filepath - unique
  description: description of the file, stating its purpose of creation.

APIs

Some of the major APIs and their functionalities are listed as follows:

- Socket API
  - connect_user
    Checks email-id and password, if authorized saves user-id and socket-id to Connection table.
  - disconnect_user
    Removes connection associated to the socket
  - message
    Notifies main app about incoming message from one of connected socket clients.
  - emit
    Emits ‘file change’ or ‘message’ events to other users to change their local data in real-time, sent by main-app.
  - is_online
    Used to check if an user is online by looking for an entry in connection
- User and Authentication API
  - add_user
    Add email-id, name, and password to ‘User’
  - remove_user
    Remove user from ‘User’ table.
  - change_password
    Change password of user

    - authenticate
      Checks if an email-id and password passed as parameters are valid, and matching with one in the database.
    - user_exists
      Checks if an email-id exists in 'User'

- File API
  File API will be based on fs library, and the storage options can be modified as per need. A configuration file would be used to control the same.
    - file_save
      Adds an entry in the 'Change' table.
      '$name.ftml' is tweaked to append 'Change-Id' to change-log attribute inside each <Waypoint/> tag. This would help to display information specific to this waypoint in UI.
      Saves the new file atomically, (this process will be made efficient by saving only the 'diff' at the right file cursor)
    - get_file
      Returns file as ASCII string or buffer, as instructed in the parameter.
    - get_authorized_users
      Returns a list of users who are authorized to collaborate on file identified by f_id/f_name.
    - get_change_log
      Returns a list of changes by collaborating users sorted by timestamps from Change table with f_id as key.
    - exists
      Returns boolean value if the file with 'file-name' exists or not.
    - list
      Returns an array of 'File' data with permission level for each file. This data can be used to view projects dashboard in client's side, as a list of projects the user is admin of and another list of projects the user is collaborating on.
    - delete_file (access_level = admin)
      Deletes file from file-path and 'File' table, preferentially delete entries related to this file from 'Message', 'Permission', 'Change'.
    - add_permission (access_level = admin)
      Checks if user with user-id exists. If yes, add user-id and file-id to 'Permission' table.
    - remove_permission (access_level = admin)
      Checks if user with user-id exists. If yes, remove entries of 'user-id and file-id together' in 'Permission' table.
    - rename_file (access_level = admin)
      Rename a file, would basically change 'path' in 'File' table corresponding to file-id.
- Chat API
    - message_save
      Adds an entry in the 'Message' table.

- get_messages
  Return an array of messages corresponding to a file_id (chat_id as each file can have one 'Chat' entry), sorted by 'created_at' values.

Main App

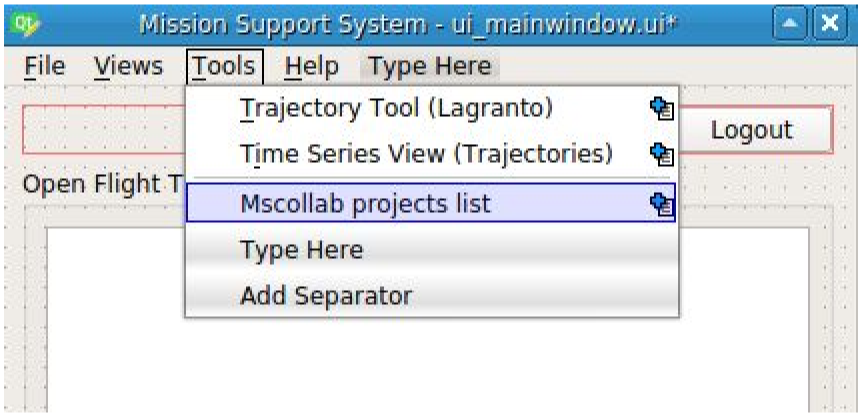
This module orchestrates all other services and APIs and regulates the data-flow.
A pseudo-code of main app is as follows:

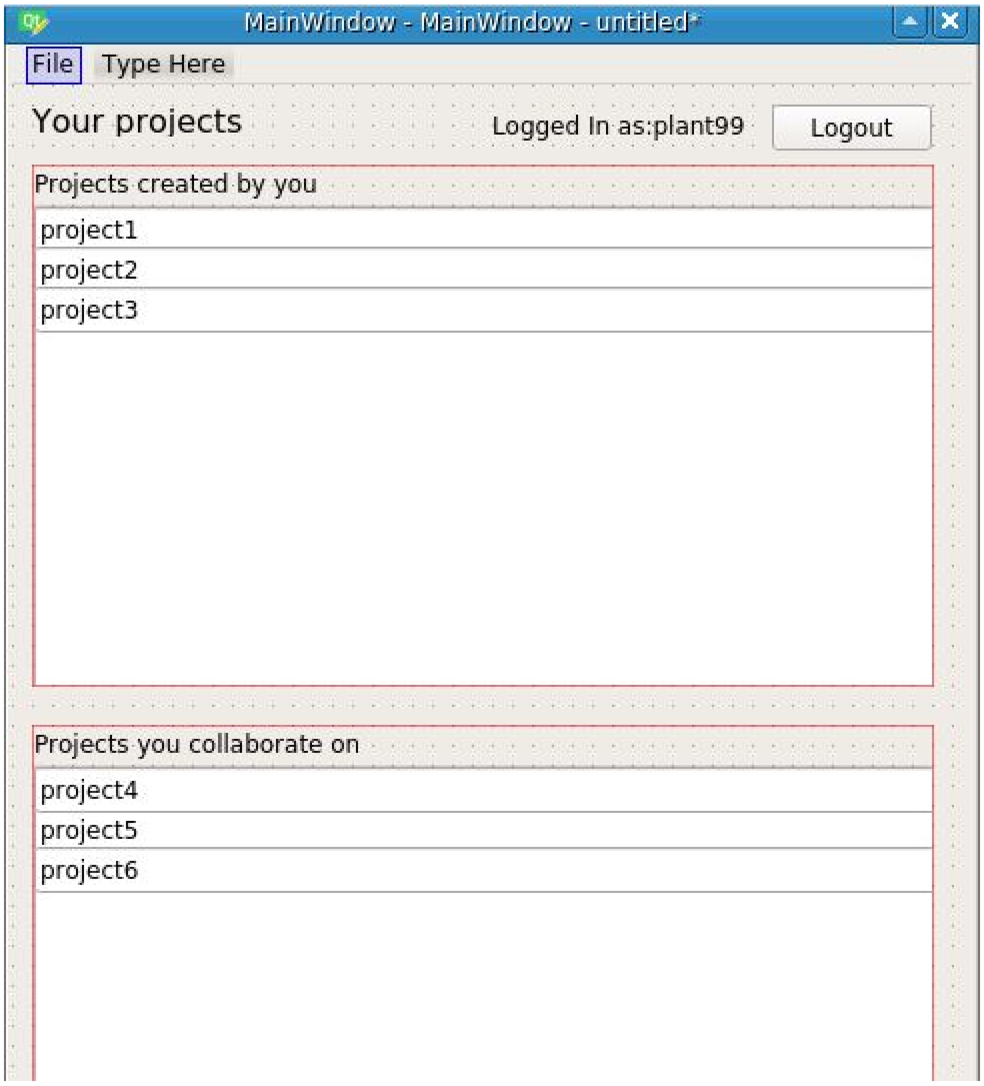
```
import socketManager
import fileManager
import authManager
import chatManager
import Flask
app = Flask(__name__)

@socketManager.sio.on('connect')
def connect(sid, env):
       # check auth here by authmanager
      socketManager.connect_user()


@socketManager.sio.on('disconnect')
def connect(sid, env):
      socketManager.disconnect_user()

#  use decorator to check auth
@sio.on('message')
def message(sid, data):
      if data.type == "file":
            # fileManager handles file here
             pass
      else:
             # data.type = message
            # chatManager handles messages here
            pass

app.route('/user')
def user_handler():
      args = request.args
      # authManager handles the rest
      # would be used to add remove user, etc
```

UI

UI module of `mscollab` will be integrated with msui, the core UI module of mss software. The temporary files would be stored in '~/mssdata' directory or as configured in 'mss_wms_settings.py'.

A list of projects which the user is working on can be displayed as illustrated below, which can be activate by clicking on Tools->Mscollab projects list, on MSS' main window.


Mission Support System - ui_mainwindow.ui*
File
Views
Tools
Help
Type Here
Trajectory Tool (Lagranto)
Time Series View (Trajectories)
Mscollab projects list
Type Here
Add Separator
Logout
Open Flight T

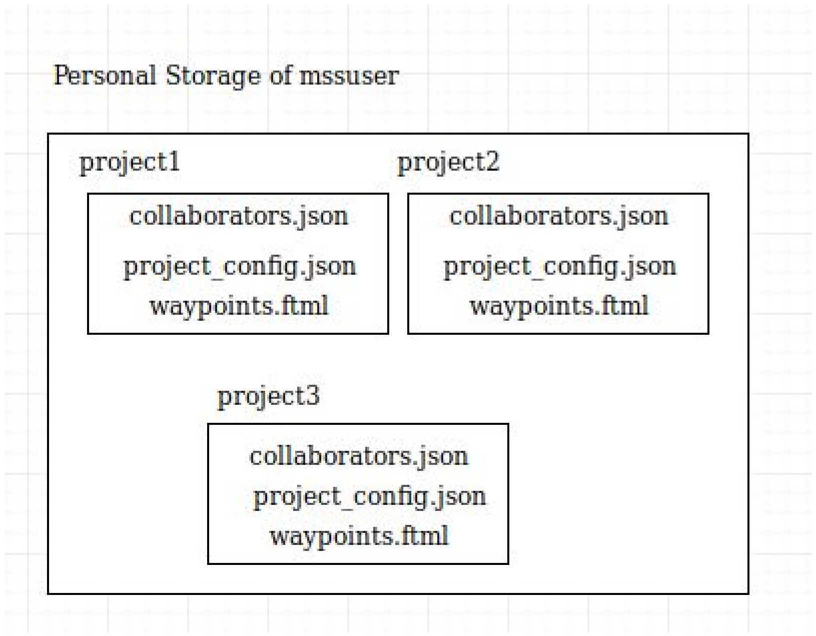


Double clicking on a project opens the mscollab-ui window as shown on page 11.

Project

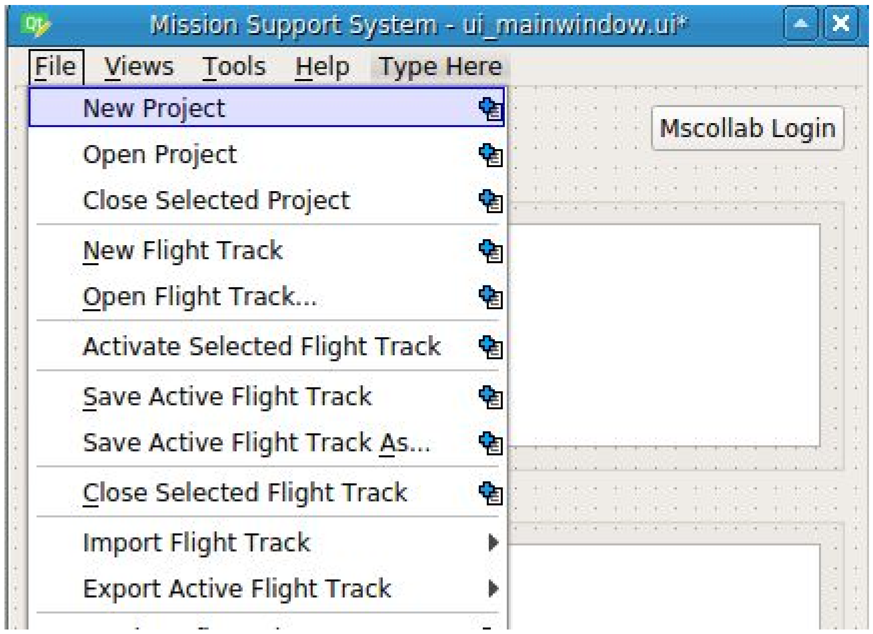
- A 'project' is a data storage model implemented by popular code IDEs, like vscode,pycharm etc.

- In this case, instead of treating a '$filename.ftml' as a project, since it won't be aesthetic to store configuration data in ftml file, it would be better if we introduce a project as a collection of some files.
- To start with, it will have a flightpath related file, and a configuration file, and a contributors file showing waypoint details and collaborators who contributed to change of the waypoint.
- An illustration is shown below.

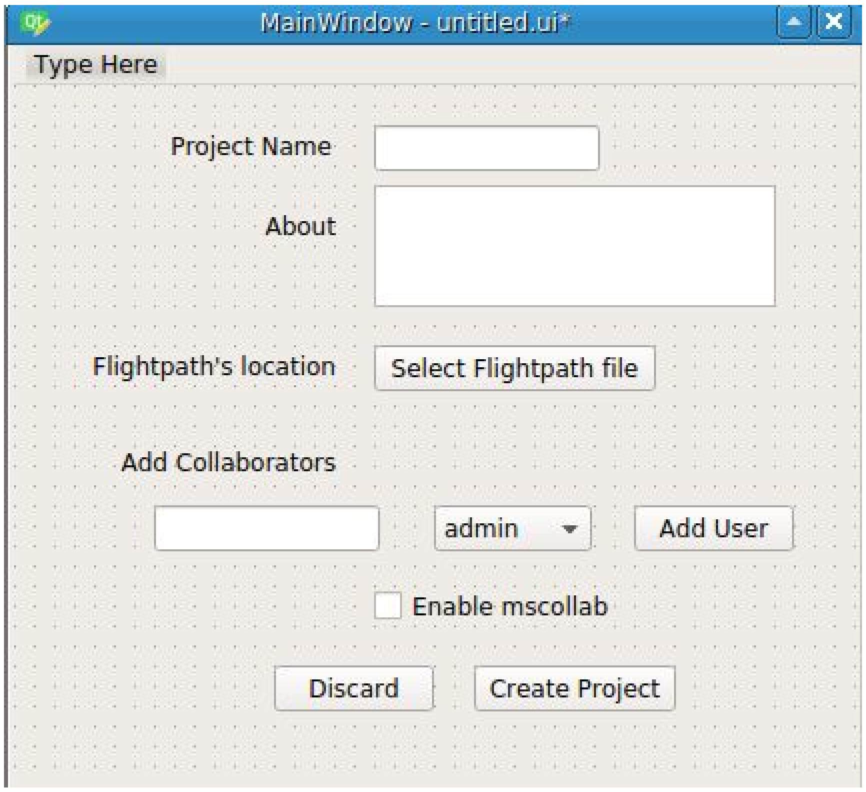


- To integrate this with mss, create_new_view function can be modified to open a flightpath in a particular view mode. The attribute `self.active_flight_track` can be changed to `self.active_project` . And each window opened would have a flightpath with some configuration obtained from project_config.py, and contribution details of each waypoint from project_config.py.

Projects can also be opened/created directly from main-window.

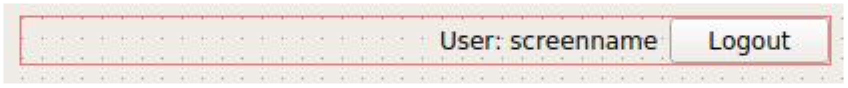


The configuration for a new project can be input by the user in a graphical manner.

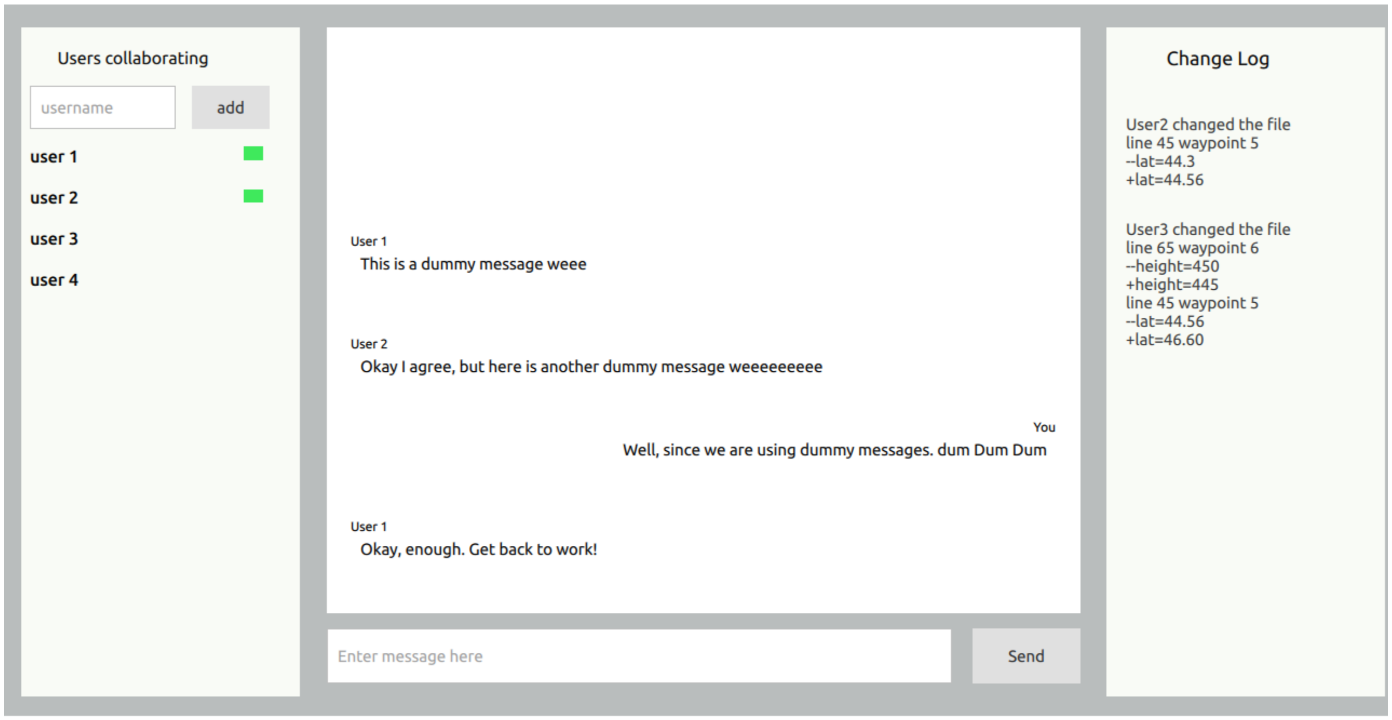


This creates a project on mscollab server with a single administrator.

Clicking on ‘Mscollab Login’ would show a dialogue-box with email-id and password, for login. If the User with email-id doesn’t exist in the mscollab database, user registration dialogue-box is opened. Once login is completed, login button gets replaced by the following display.



Opening a project which the user is administrator of would open the following window. Clicking on a project which the user is the collaborator of, opens a similar window, without options to add/remove collaborators.



User 1
This is a dummy message weee

User 2
Okay I agree, but here is another dummy message weeeeeeeee

You
Well, since we are using dummy messages. dum Dum Dum

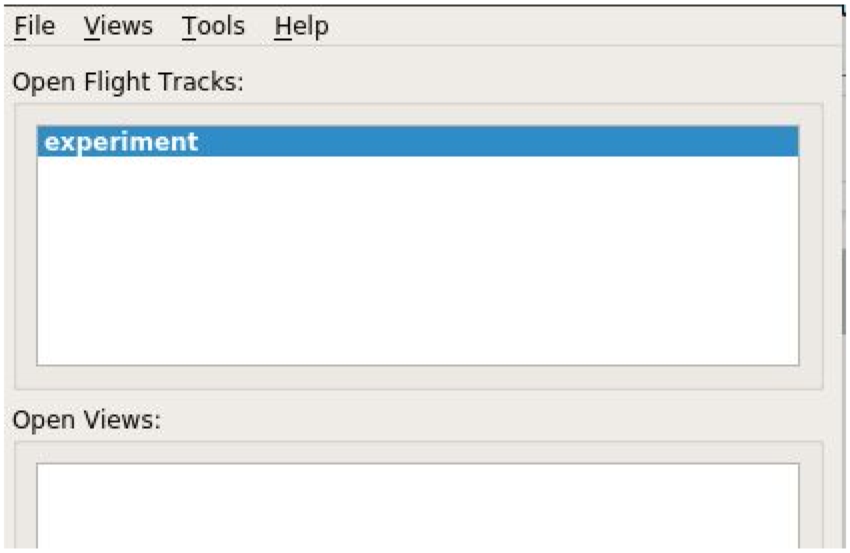
User 1
Okay, enough. Get back to work!

User2 changed the file
line 45 waypoint 5
--lat=44.3
+lat=44.56

User3 changed the file
line 65 waypoint 6
--height=450
+height=445
line 45 waypoint 5
--lat=44.56
+lat=46.60

- Left side of the window as seen by the user, has a list of users collaborating on the experiment.
- Right side of the window as seen by the user, has a log of recent history of changes.
- Central space lists the group chat messages which will support important markdown syntax (e.g bold, italics) during editing.

Once a new file is created or a file is opened with `mscollab`, say 'experiment.ftml', msui window gets updated with new file in the listing which can be edited in an usual manner and each change gets saved in `mscollab` server after a certain duration. The continuous backup can be disabled/enabled by the user by an UI element.

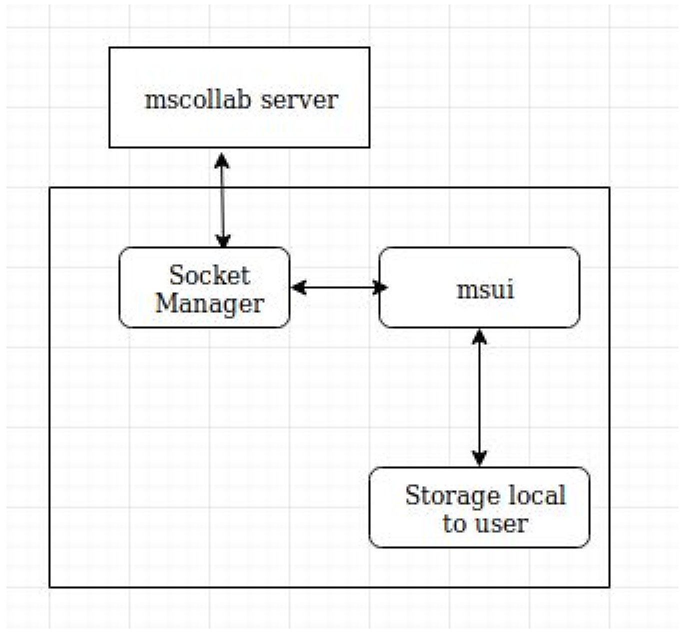


If one opens an old stored file, say 'old-experiment.ftml', following API calls are made with filename/file-id as parameter.

- /get_file handled by FileManager
  If file is not found in $data_dir, it's created in $data_dir as an intermediate save point.
- /get_authorized_users handled by FileManager

- /get_log handled by FileManager
- /get_messages handled by ChatManager

Once this data is received, it is suitably rendered to a new `mscollab-ui` window as shown in the mock-up. The overall data-flow diagram in front-end would resemble the following.



SocketManager class would be a simple class, with two major functions:

- connect()
  Used to start connection after authentication is complete and the client receives a token
- on_message()
  Used to handle messages incoming from `mscollab` server (when SocketManager.emit() is called). Event handlers would be written in '`msui/mscollab_ui.py`'.

Timeline

I have tried to schedule the project work as per GSoC's timeline.
Note:

- Every time span starting from 27th May till 12th August, if it doesn't involve 'buffer for improvement' or 'bug-fixes', would include unit tests.
- Holidays like weekends are included in the time blocks. (I plan to take 1 day off per week, if work is up-to date as per schedule)

Time Span	Work
6th May	Accepted student proposals announced
7th May - 26th May	Community Bonding Period
7th May - 8th May (2 days)	Setup logistics - Decide on schedule and mode of

	communication for weekly and emergency meetings. - Setup other logistics i.e GSoC blog
9th May - 21st May (~2 weeks)	Finalize proposed architecture with mentors - Discuss with mentors about any changes or improvement to be made in proposed architecture and draft a final design document. - Discuss and confirm the selection of software libraries and tools, replace them with better ones if needed. - Help solve some issues related to 1.9.0 release of mss. - Dive a little deeper into mss' way of handling File IO in client's side. - Start working on 'Project' for core msui
21st May - 26th May (~1 week)	Setup project environment - Finish up on Project integration for msui. - Make a list of software and tools dependencies of the project. - Install and verify their installation.
27th May	Official coding starts
27th May - 5th June (~1.5 weeks)	Starter template setup and development of User related route. - Setup `mscollab` server with Flask, and python-socketio. - Migrate database models and write classes for the same as Schemas. - Write API endpoints of /user route i.e login, signup, authentication.
6th June - 12th June (~1 week)	Development of Socket API - SocketManager class is developed along with all the member functions. - Integrated with Main App. - Test connect/disconnect/message functionality with a dummy client setup externally.
13th June - 16th June (~0.5 weeks)	Development of Chat API - ChatManager class is developed along with

	event handler functions. - Integrated with ‘message’ event in SocketManager.message() function. - Test functionality with dummy python client.
17th June - 24th June (~1 week)	Development of File API - FileManager class is developed. - Separate handler functions for different storage options (e.g local and WebDAV) are considered and implemented. - Development of Permission class and associated handlers. - Test functionality with python client
24th June - 28th June	First round of evaluations
24th June - 28th June (~0.5 weeks)	Buffer time to cover back-logs, solve newly discovered bugs, refactoring needs, and improve tests.
29th June - 6th July (~1 week)	Development of ‘Change’ related handlers. - Development of FileManager class is continued and the ‘diff’s are stored in ‘Change’ table. - Design algorithm to handle a change, link it to waypoint(s) and save this change-id to XML tag <Waypoint/> for further insights. - Test this with a dummy python client.
7th July - 14th July (~1 week)	Development of auth/connection for front-end and Projects’ Dashboard - Setting up a login modal for users when they open `mscollab’s` dashboard. Development of socket utilities for front-end - Development of dashboard which shows users’ projects which they collaborate on or are admin of. - Setting up click handlers which should open a dummy window as of this duration.
15th July - 22nd July (~1 week)	Development of Project Window upto some features i.e users-list and messages - Development of Project window where users would discuss on a project.

	- Identify admin/non-admin and setup corresponding UI i.e (Add/delete collaborators, etc.)
22nd July - 26th July	Phase 2 evaluation
23rd July - 28th July (5 days)	Continue development of Project Window - Development of change-log column - Thorough testing of the required functionalities implemented. - Listing down bugs
29th July - 3th August (~0.5 weeks)	Buffer time to cover back-logs, solve newly discovered bugs, refactoring needs, and improve tests.
5th August - 12th August (~1 week)	Load ‘change’ attributes from <Waypoint/> XML and use these changes to show insights like - Last edited time - Last 5 contributors in topview/sideview window.
13th August - 18th August(~1 week)	- Implement integration testing, find and solve more bugs if found. - Improve on general algorithm implementations to achieve a more efficient solution, if needed. - Document code and integrate documentation to that of mss’.
19th August - 26th August	Work Submission
19th August - 25th August(~1 week)	- Ask mentors for a more detailed review, and work on fixes covering code/documentation/deployment options etc. - Project and documentation submission.
26th August - 2nd September	Mentors submit final evaluation
3rd September	Final results announced.

Future Work

`Mscollab’s` development doesn’t stop with GSoC’19. Once the initial base is established, I want to keep working along further development and maintenance at my spare time. Some features which I hope to work on after GSoC are as follows.

- Multiple workspaces in single window
  The above layout shows a single collaboration task in one window. This can be extended to multiple

workspaces in a single window as ‘tabs’. This work would mostly involve working with the UI of `mscollab` software.

- Waypoint as a model
  If two other models are introduced as ‘Waypoint’ and ‘Path’, this design can be largely exploited to save better insights about the changes made to Waypoints.
- Make `mscollab` generic
  The server can be used for any other XML or structured data with some modification. This can help open doors to make `mscollab` available for broader use-cases.
- Version Control
  Version control of files can be introduced in the central storage, using GitPython or a similar library. Some concepts like branches, cherry-pick/revert can be exploited to implement functionalities like undo, or redo operations made by commit(s).

Other Commitments

- If I am selected as a GSoC student, it will be my full-time commitment. I don’t have any other jobs or internships during GSoC.
  I have some important classes and assessments to attend on the third week of August, so for this duration, I’d be able to work a few hours less.
  May 6 - August 12: 45-48 hours/week
  August 13 - August 19: 36-38 hours/week
  August 20 - August 26: 45-48 hours/week
  Though I’d join my school after summer break on 29th June, it won’t affect the numbers of hours I work on the project. I will inform my mentors about any changes in schedule in a timely manner.

- Other than PSF, I am not applying to any other organisation to participate in GSoC. I am submitting only one proposal, for Mission Support System under PSF.