Mission Support System: Mission Support Collaboration Improvements

About Me

Name: Tanish Grover (Bitbucket: Tanish Grover, Github: Tanish0019, mss-devel.slack.com: Tanish Grover)
University info:
University Name: Delhi Technological University
Major: Computer Engineering
Current Year: 3rd Year
Expected Graduation date: May 2021
Degree: Bachelor of Technology

Time Zone: Indian Standard Time (GMT + 5:30 hours)

Code Contribution:

I have constantly been working with MSS mentors, discussing issues, bugs and features to improve MSS and have made substantial contributions to MSS, gaining more understanding of the code base with each contribution. The following is the list of PRs I have worked on:

- https://bitbucket.org/wxmetvis/mss/pull-requests/814/fixes-issue-553-add-user-account-delete/diff
- https://bitbucket.org/wxmetvis/mss/pull-requests/803/fixes-error-in-opening-topview-wms-control/diff
- https://bitbucket.org/wxmetvis/mss/pull-requests/800/fixes-issue-549-tests-failing-on-osx/diff
- https://bitbucket.org/wxmetvis/mss/pull-requests/799/fixes-issue-517-add-json-support-for-bulk/diff
- https://bitbucket.org/wxmetvis/mss/pull-requests/798/fix-test_registration-failing/diff
- https://bitbucket.org/wxmetvis/mss/pull-requests/785/fix-for-issue-537-and-484/diff
- https://bitbucket.org/wxmetvis/mss/pull-requests/797/fixes-issue-538-revoke-permission-not/diff
- https://bitbucket.org/wxmetvis/mss/pull-requests/787/fixes-issue-539-handling-project/diff
- https://bitbucket.org/wxmetvis/mss/pull-requests/784/fix-mscolab-test_save_fetch-in-develop/diff

- https://bitbucket.org/wxmetvis/mss/pull-requests/781/fix-mscolab-osx-test-fails-fixes-issue-531/diff
- https://bitbucket.org/wxmetvis/mss/pull-requests/777/mscolab-save-fix-fixes-534/diff
- https://bitbucket.org/wxmetvis/mss/pull-requests/776/fixes-issue-533-project-being-unselected/diff

Project Information:

Sub-org name: Mission Support System (MSS)

Project Abstract:

The Mission Support system is an application for scientists in the field of atmospheric science to help them simplify the process of planning a scientific flight in which parameters of the atmosphere are measured. It allows users to map out the possible flight paths while taking into consideration different meteorological data and forecasted parameters and viewing them along possible regions of the flight path. In Google Summer of Code 2019, the Mission Support Collaboration(mscolab) module was developed which allowed real-time collaboration and editing of flight paths of a project. However, there are some key features which are currently missing from the mscolab module. I propose to work on the following features and improvements to Mission Support Collaboration

1. **Local/Offline Flight Path Editing:** There are many instances where a user might want to test out his/her own changes to the flight path locally without changing the path on the server. Another scenario is that the user has a slow network connection at the flight campaign location which is a common problem. Both these situations require a method for the users to be able to make changes to the flight path locally and then be able to compare and merge it with the flight path on the server later.
2. **Chat Service Improvements:** Currently, a chat service exists to help users communicate while working on a project in Mscolab. However, sending plain text messages is the only option available to the users at present. I would like to add some important features to the chat service to improve communication between collaborators. These features are - markdown support, ability to send images, searching through messages, ability to delete messages and replying to a specific message.
3. **Admin Dashboard:** There can be hundreds of users who need to be added to a project. Currently, the only way to add a user to a project is through a simple text box which takes one username/email at a time. To solve this, a new admin window needs to be developed. This admin window would allow the project creator and admins to easily and quickly add/remove users from the project and modify their access levels.

Project Description:

Mission Support System Introduction:

Mission Support System is made up of the following major components:

1. **MSUI:** Built using the PyQt5 framework, MSUI is the client-side or GUI of the application. The GUI includes different views so that researchers can analyse the flight path from different perspectives and work on the route accordingly. It displays maps served using the MSWMS server on which researchers can mark their waypoints.
2. **MSWMS:** WMS(Web Map Server) is the server built using Flask that produces on-demand generated visualisations of meteorological predictions. This is particularly useful when the location at which the flight campaign is being held does not have access to high bandwidth internet. MSUI uses this server to fetch the predicted visualizations and then display them on top of maps for researchers to analyse and plan their flights better.
3. **MSCOLAB:** MSCOLAB(Mission Support Collaboration) is the component which allows real-time collaboration among researchers. Allowing researchers to create “projects”, edit flight paths in real-time and communicate with each other using a chat service, mscolab greatly improves the time taken by researchers to come up with a flight path. The mscolab server is built using Flask.

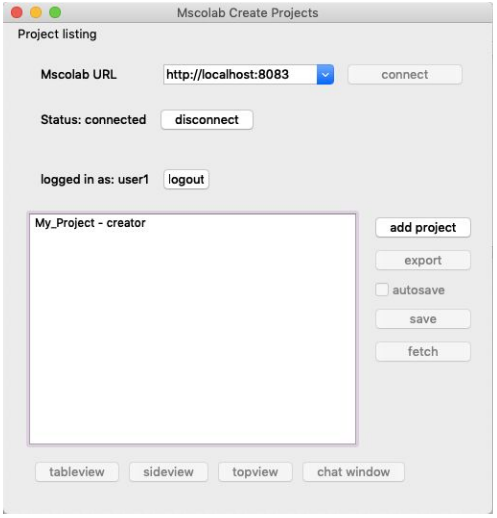
A Brief Description of Mscolab’s Working:

Mscolab currently has 2 windows:

1. The Project Listing Window:

A user has to connect to the mscolab server then log in. The user has the following options:

- **Add Project:** Allows the user to create a new project
- **Export:** Save the project as a .ftml file.
- **Autosave:** Only the creator and admins have this option. If checked, when any user makes a change to the project it is automatically saved to the server. If unchecked all changes made by a user are saved in a temporary file in *~/mss/tempfile_mscolab.ftml*. In this case the user has 2 options. Either **“save”** the changes to the server. This completely overwrites the server with the user’s changes and creates an entry in the database so if the changes are not desirable we can checkout to a previous commit. The other option is **“fetch”**. This fetches the



Mscolab Create Projects

Project listing

Mscolab URL http://localhost:8083

Status: connected

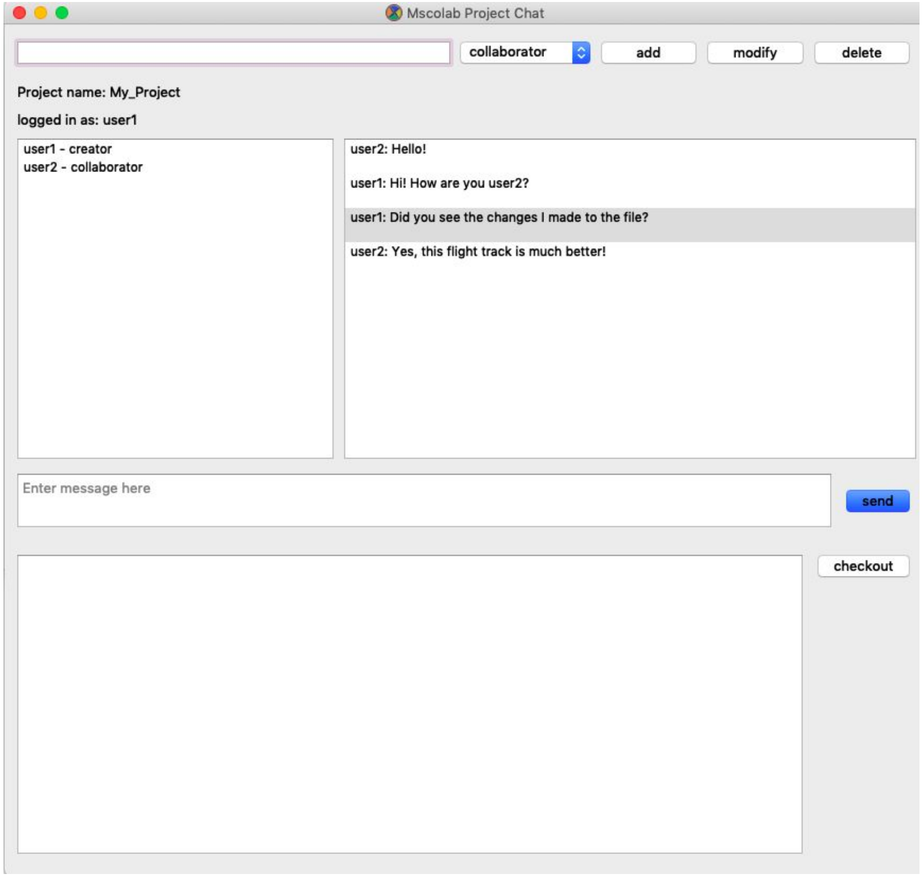
logged in as: user1

My_Project - creator

changes from the server and completely overwrites the user's local changes. If autosave is turned on at any moment all users will lose their local work which will be overwritten by the server .ftml file.

- **Table/Side/Top View:** This allows users to visualise the flight path in different views and make changes to the flight track.

2. The Chat and User Management Window:



Project name: My_Project

logged in as: user1

user1 - creator
user2 - collaborator

user2: Hello!

user1: Hi! How are you user2?

user1: Did you see the changes I made to the file?

user2: Yes, this flight track is much better!

This window has the following functionality:

- **Sending Messages:** Users can chat with each other using plain text messages.
- **Project Users Management:** The mscolab project admins can add/modify/delete users from the project using the textbox provided.
- **Commit History:** Each time a user presses "save" in the main mscolab window when autosave is off, a commit is recorded in the *Change* table in the database. This commit shows up in the commit history box at the bottom of the window and a *diff* between the new and old *.ftml* file is shown in the message box to all users. If the pushed changes are not good the admins can checkout to an older commit.

Current Issues in Mscolab:

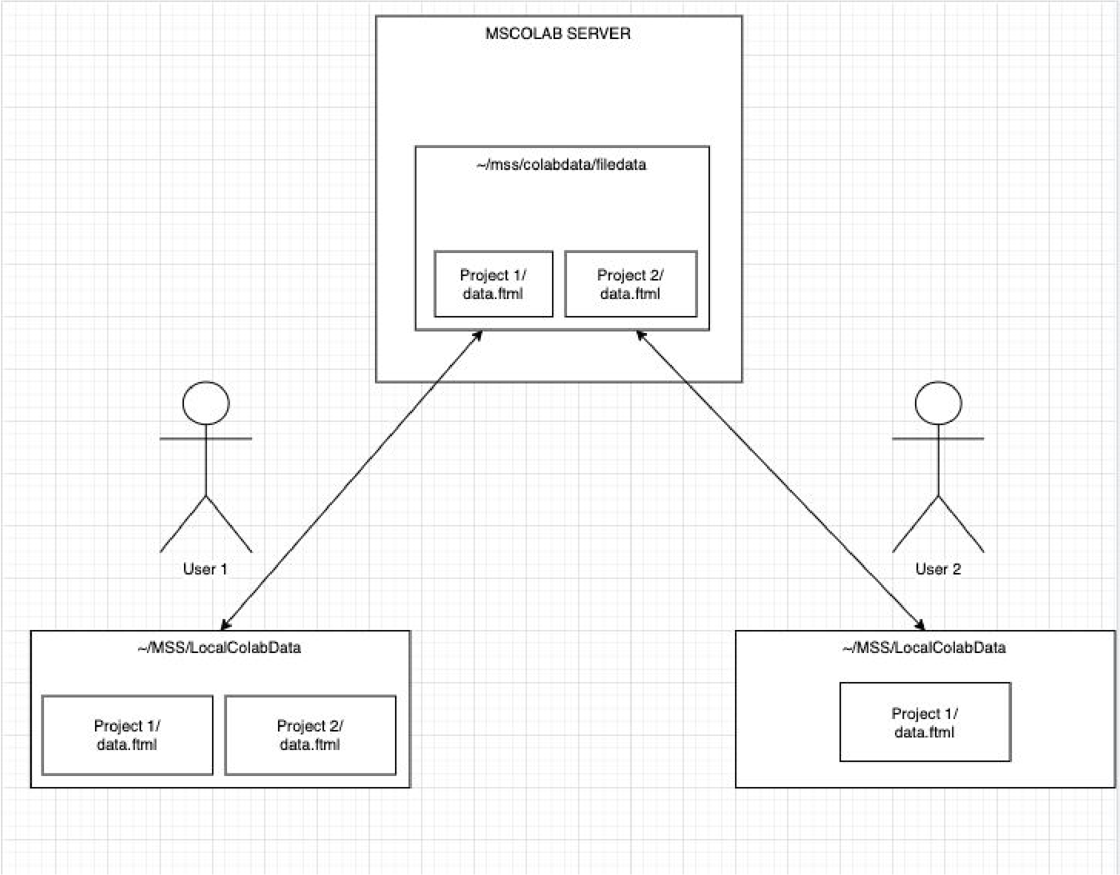
There are a few issues with the current capabilities of mscolab. The following are the areas of improvement:

1. **Flight Path Editing:** There is very little flexibility for users when it comes to editing flight paths. These constraints include:
   - If any user wants to try out some changes locally or doesn't have access to good internet, they don't have an option to work on a local file. They are dependent on whether the admins have turned autosave on or off.
   - No way exists to compare your changes with the server file when pushing to the server. The only option is to overwrite the server file completely with your changes when clicking save.
   - All the changes made while autosave is off are stored in a single temporary file(*~/mss/tempfile_mscolab.ftml*). This data is completely volatile and starting to work on a different project would overwrite it.
   - No control on when a commit can be made. The only time a commit is made when a user "saves" his changes when the autosave is off. Admins might want to manually add a commit as a checkpoint while developing a flight path.
2. **Chat Limitations:**
   - The current chat system only has a feature to send simple plain text messages. This might make users shift to an external chat application.
   - The chat can get cluttered with *.ftml* change alerts every time a user "saves" his/her changes.
3. **User Management:** The only way for admins to add users to a project is through a text box by entering the usernames one by one. This is okay for smaller projects but when the number of users goes above say 10 this process can be very time-consuming.

My Proposal:

If selected for GSOC, I would be working on fixing the issues mentioned above. The following is a detailed explanation of what I would be working on to tackle these issues during my GSoC period.

Flight Path Editing: To handle the issues related to flight path editing, the "autosave" feature needs to be removed and instead of just using a temporary file to store local changes each project would require its own file. The new directory structure is shown below. Each project would have a .ftml file inside *~/mss/localColabData* which would reflect the user's personal work. The "autosave" functionality would be removed and instead 2 new toggles will be added. These would be:



1. **Work Offline Toggle**: Available to every user. Turning it on would start a ***work offline mode*** in which only the local .ftml file in *~/mss/localColabData* would be changed. As each project has its own .ftml file, the user can easily switch between projects without necessarily saving his/her changes. Each time a user decides to push changes to the server, a commit is recorded.
2. **Auto Sync Toggle:** Only available to mscolab project admins. If turned on, every change any user makes would be directly made to the .ftml file on the server in real-time. If turned off, each user would be sent into the ***work offline mode*** and would have to use the “push to server” button if they want to write their changes to the server.

To help users compare changes and merge local and server files according to their needs whenever they push changes to the server or fetch from the server, a **new merge window** would be required. This window would allow the user to compare the data side by side and let the user keep whatever changes he/she prefers.

A commit button would be provided to the mscolab project admins to make a commit at any point as a checkpoint to which they can later revert back to.

Chat Service Additions: For improving the communication between the users, I would be working on adding the following features to the chat service provided in mscolab:

1. **Markdown Support**: Users would be able to send messages which can be formatted using markdown. A preview tab would be available for them to first preview the message.
2. **Image Upload:** Users would be able to send images in the chat.
3. **Message Delete:** Users would be able to delete their messages to prevent confusion from accidentally sending a wrong message.
4. **Message Search:** Users would be able to search for a message in the chat to help them find the information they are looking for in a long conversation.
5. **Message Replying:** Users would be able to reply to specific messages. This feature is particularly helpful in group chats like in mscolab to remove ambiguity.

Apart from these features, a new UI for the chat window would be developed. The commits history section would be moved to a separate window of its own where the commit alerts and commit checkout facility would be available making the chat less cluttered.

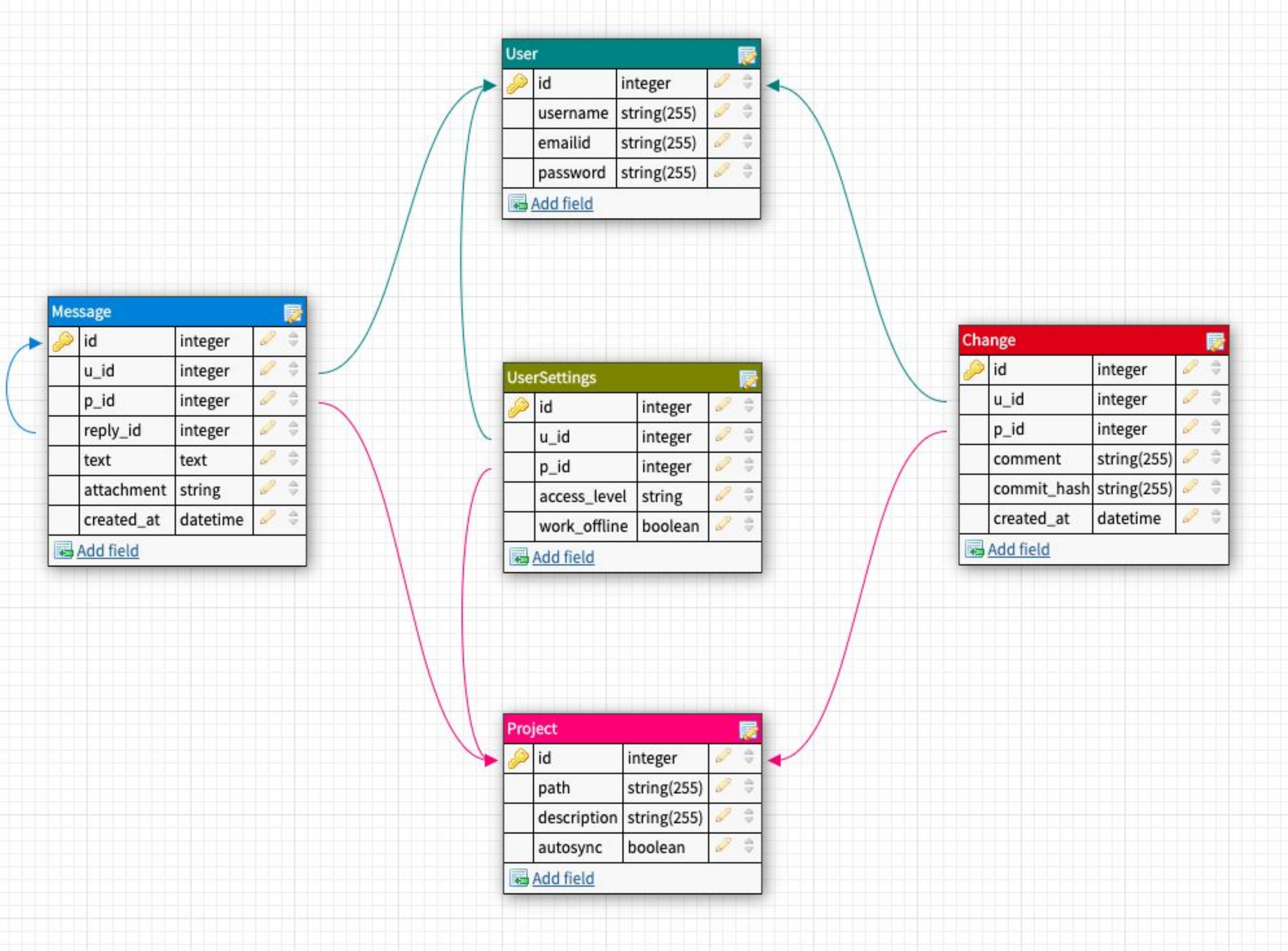
User Management: To make managing users of a mscolab project easier for the admins, a new ***Admin Dashboard*** would be developed. This window would let admins view all the users in a table and add them quickly by selecting them from the table and adding/modifying/deleting their permission in bulk.

From here on I have described what my work for GSoC will be in terms of database schema designing, the API's that would need to be developed or modified and the user interface for the proposed ideas.

Database Schema:

For managing the auto-sync setting, a new attribute - **"autosync"**, would be added in the *Project table*. The *Permissions table* would be renamed to UserSettings table and a boolean attribute - "**work_offline"** would be added to it to hold the value of the user's offline mode setting.

For incorporating the chat service features new attributes in the *Message* table would be added. These would be **"attachment"** which would store the path of the image saved on the server or would be null and the other would be **"reply_id"** which would have the id of the message to which this message replies to. After making the changes the final database schema for mscolab would be:



User

id	integer
username	string(255)
emailid	string(255)
password	string(255)

Message

id	integer
u_id	integer
p_id	integer
reply_id	integer
text	text
attachment	string
created_at	datetime

UserSettings

id	integer
u_id	integer
p_id	integer
access_level	string
work_offline	boolean

Change

id	integer
u_id	integer
p_id	integer
comment	string(255)
commit_hash	string(255)
created_at	datetime

Project

id	integer
path	string(255)
description	string(255)
autosync	boolean

- **User**
  - Id: Id of user. Primary Key
  - username: Unique username of each user.
  - emailid: Unique email address of each user.
  - Password: Hashed password of each user.
- **Project**
  - Id: id of project. Primary Key
  - path: name of the project.
  - description: Short description of the project.
  - autosync: automatic sync is on by admin or not.
- **UserSetting**
  - Id: id of setting. Primary Key.
  - u_id: id of a user. Foreign Key.
  - p_id: id of a project. Foreign Key.
  - access_level: access level of the user. (Creator, Admin, Collaborator, Viewer)
  - work_offline: If the user has work offline enabled or not.
- **Message**
  - id: id of the message. Primary Key.

  - u_id: id of the user who sent the message. Foreign Key.
  - p_id: id of the project to which this message belongs. Foreign Key.
  - reply_id: id of the message this message is replying to. Null if not replying to any message.
  - text: The text content of the message.
  - attachment: The path to the image file on the server. Null otherwise.
  - created_at: timestamp on when the message was made.
- **Change**
  - id: The id of the change. Primary Key.
  - u_id: The id of the user who made the change. Foreign Key.
  - p_id: The id of the project of this change. Foreign Key.
  - comment: The commit comment message.
  - commit_hash: The git commit hash for the change.
  - created_at: Timestamp of when the change was made.

API:

The following are the *new APIs* or the existing ones which would need to be modified on which I would be working on for my ideas proposed for GSoC.

Flight Path Editing:

- **Fetch User Settings:** Need to update existing endpoint */project_details* to include the *work_offline* value and the *autosync* value (enabled or disabled)
- **Toggling Work Offline on/off:** New endpoints */work_offline_enabled* and */work_offline_disabled* would be developed to set the status of the *work_offline* in the *UserSetting* table.
- **Toggling Auto Sync:** New endpoints */enable_autosync* and */disable_autosync* need to be developed. Both endpoints update the autosync value for the project in the *Project Table*. The */disable_autosync* endpoint emits a socket event to all the users and enables work offline for every user. Whereas the */enable_autosync* emits a socket event to let users know that auto-sync is on and if they want they can turn off the work offline mode.
- **Commit Changes:** Requires new endpoint - */commit_changes* which takes a commit message and commit hash to insert an entry in the *Change Table*. A python wrapper for git like GitPython can be used to make the commit and get the commit hash.
- **Fetch Server Waypoints:** Requires new endpoint - */fetch_waypoints* data in the server .ftml file. This endpoint is used in the *Merge Window* to compare the local and server file changes.
- **Pushing To Server:** Each time a user pushes to the server, he/she first needs to make a choice of which changes to keep and how the local and server file should be merged. This is handled by the new *Merge Window.* After selecting the changes the data is

pushed to the server and both server .ftml file and local .ftml are synced. A commit is made and saved as an entry in *Change Table.*

- **Fetching From Server:** When a user fetches changes from the server, he/she needs to resolve the merge conflicts with his/her local file. This is done in the new *Merge Window.* New functions would need to be added to update the local .ftml file after the user has selected what changes he/she wants.

Chat Service Additions:

The following are the APIs and changes I'll be working on for improving the chat service in mscolab.

- **Markdown support**: It is handled in MSUI while rendering the messages using the python markdown package to get the equivalent HTML.
- **Saving Images:** To handle attachments, when the user uploads the image, the file is read as binary from the UI and the binary image data is emitted to the server along with the other message details through a socket event. The *add_message* method in ChatManager class would need to be modified to save the image on the server in *mss/mslib/mscolab/images* using a combination of image name and timestamp as the image name. The path to this image on the server is then saved in the Message Table in the *attachment* column.
- **Retrieving Images:** The *get_message* function inside the ChatManager class needs to be modified to check if the *"attachment"* attribute is null or not. If it is not null then the image is read as binary and sent back to the UI where it is displayed using *QPixMap*.
- **Deleting Messages:** New endpoint */delete_message* in mscolab server is needed. It will receive the message id and delete the message from the database also emitting a socket event to remove the message from every user's chat window. It is usually preferable to only be able to delete recent messages. Therefore, the message's *created_at* attribute can be used to check if the message is deletable or not.
- **Message Replying:** The UI will pass the *reply_id* to the ChatManager class. The *reply_id* points to the message to which this message is replying to. It is saved in the database along with all other message details.

Admin Dashboard:

The following APIs need to be added to the mscolab server for the admin dashboard:

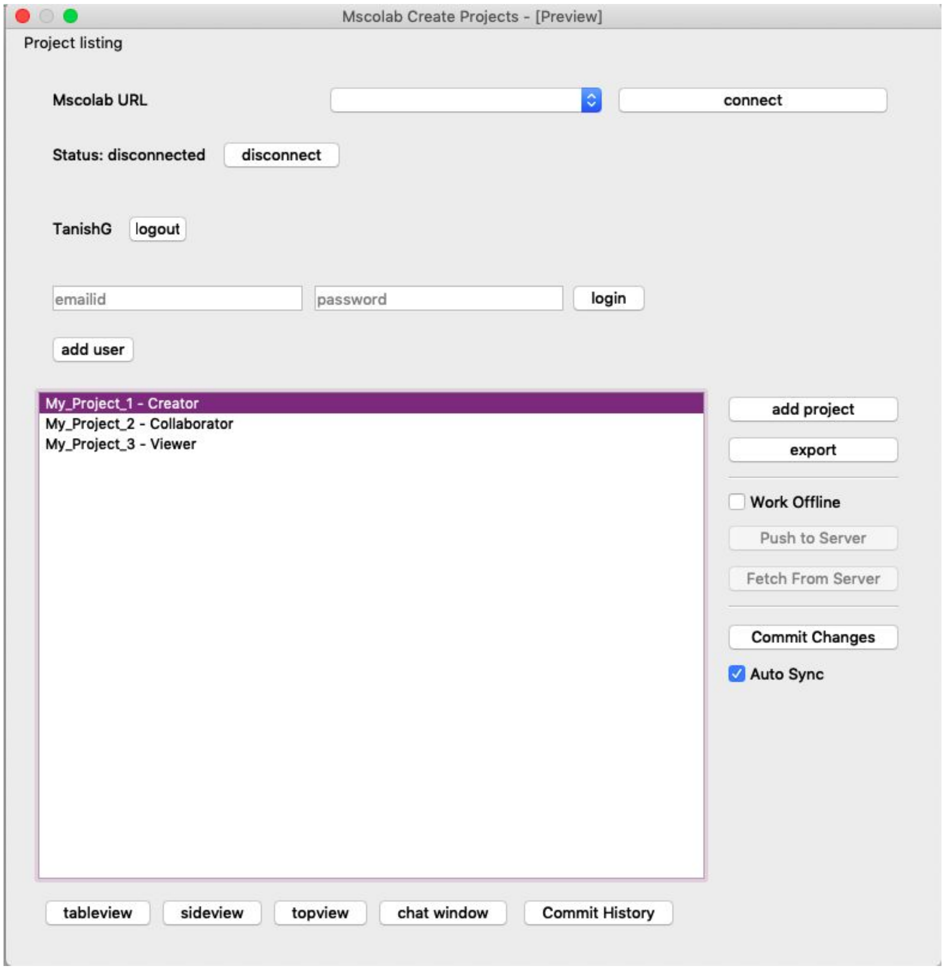
- **Fetch All Users Without Permission:** Requires new endpoint - */get_users_without_permission* to get all the users on the server who do not have permission to the project. Each flight campaign is running their own server so the number of users won't be very high.
- **Fetch All Users With Permission:** Requires new endpoint- */get_users_with_permission* to get all users who have permission to the project along with their access level.

- **Add Permission for Multiple Users:** Requires new endpoint - */add_permissions.* Takes an array of user ids along with an access level. Adds the corresponding permission in the *UserSetting* Table.
- **Modify Permission of Multiple Users:** Requires new endpoint - */modify_permissions.* Takes an array of user ids along with the new access level. Modifies the values in the *UserSetting* Table. Emits a socket event so the changes show up on every user's mscolab project window.
- **Revoke Permission of Multiple Users:** Requires new endpoint - */revoke_permissions.* Takes an array of user ids and deletes their entry from the *UserSetting* Table. Emits a socket event to remove the users from chat rooms.

User Interface:

Here I have shown the user interface changes and the new windows that I would be working on for the proposed ideas.

Flight Path Editing:



Status: disconnected

TanishG

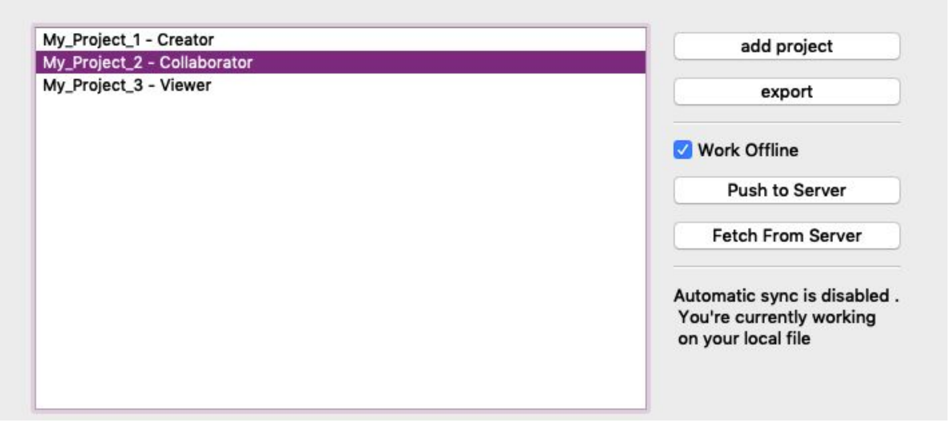
My_Project_1 - Creator
My_Project_2 - Collaborator
My_Project_3 - Viewer

☐ Work Offline

☑ Auto Sync

The creator and admins have access to making commits and toggling auto-sync. The function of auto-sync is if the user does not have “working offline” turned on, all the changes will be synced with the server in real-time.

- **Toggling Auto Sync off:** If auto-sync is off, work offline is toggled for everyone. No changes will be saved to the server without the users pushing to the server manually.

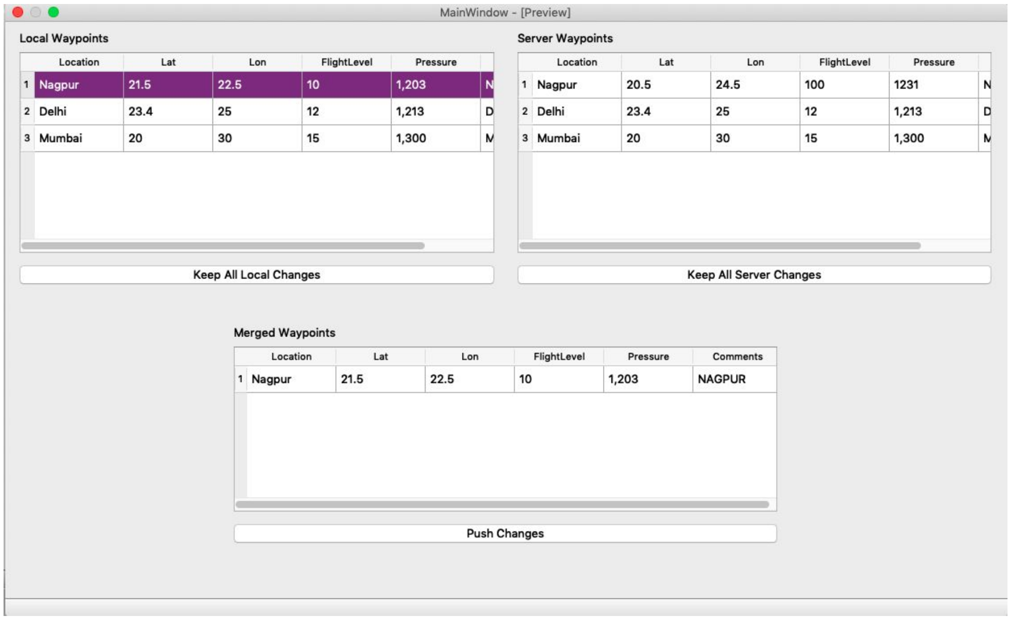


My_Project_1 - Creator
My_Project_2 - Collaborator
My_Project_3 - Viewer

☑ Work Offline

Automatic sync is disabled .
You're currently working
on your local file

- **Pushing and Fetching from Server:** Every time a push or fetch from the server is made, the user can choose to either keep the local data, keep the server data or merge them. This is done in the new *merge_window.* The user can view the two waypoint data side by side and make his/her choices on how to merge by selecting the rows to keep in the final flight track.



MainWindow - [Preview]

Local Waypoints

	Location	Lat	Lon	FlightLevel	Pressure	
1	Nagpur	21.5	22.5	10	1,203	N
2	Delhi	23.4	25	12	1,213	D
3	Mumbai	20	30	15	1,300	M

Server Waypoints

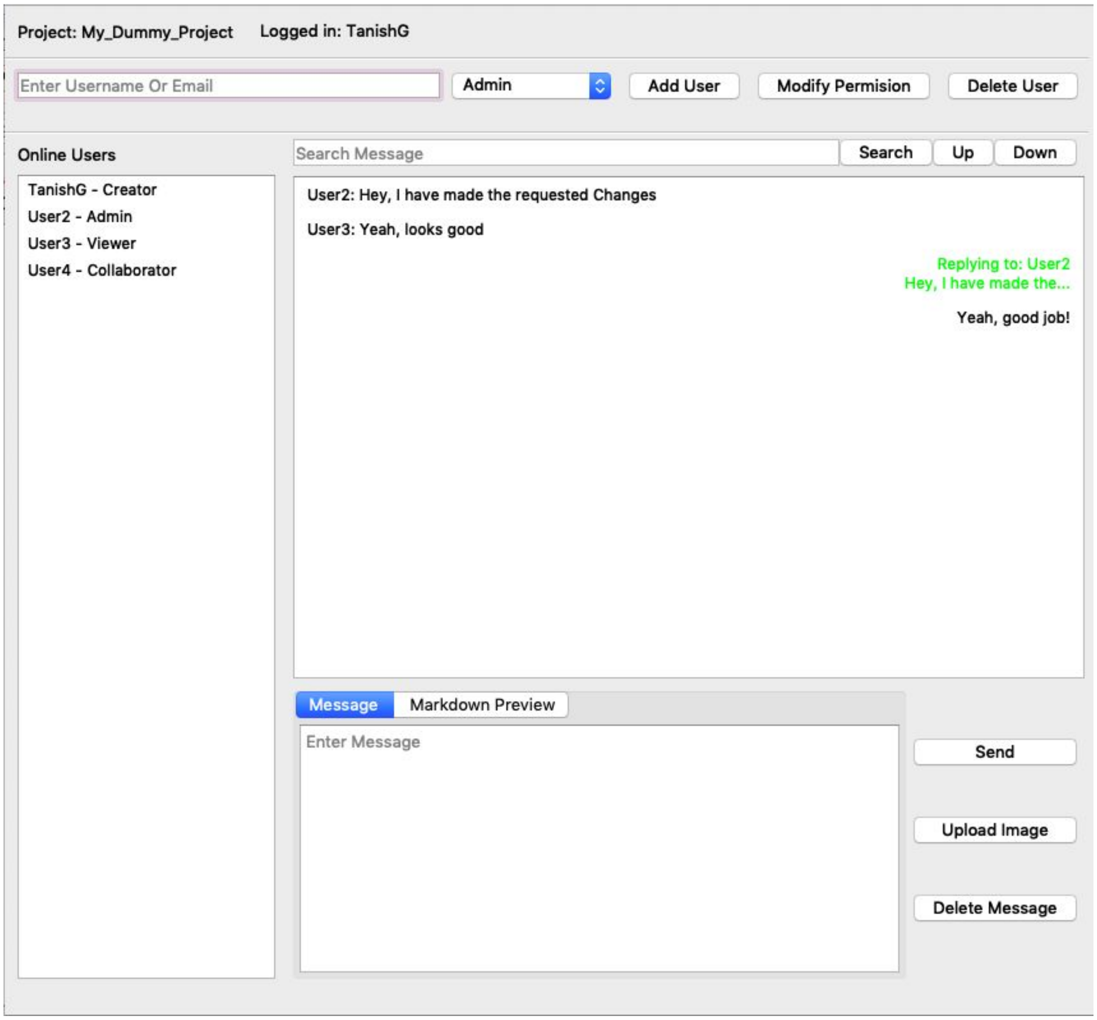
	Location	Lat	Lon	FlightLevel	Pressure	
1	Nagpur	20.5	24.5	100	1231	N
2	Delhi	23.4	25	12	1,213	D
3	Mumbai	20	30	15	1,300	M

Merged Waypoints

	Location	Lat	Lon	FlightLevel	Pressure	Comments
1	Nagpur	21.5	22.5	10	1,203	NAGPUR

Chat Window Changes:

The new UI for the chat window on which I would be working on would be



Project: My_Dummy_Project   Logged in: TanishG

Online Users

TanishG - Creator
User2 - Admin
User3 - Viewer
User4 - Collaborator

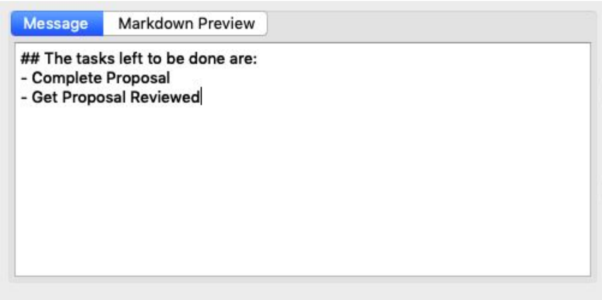
User2: Hey, I have made the requested Changes

User3: Yeah, looks good

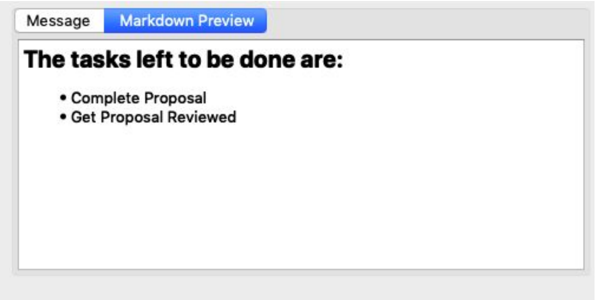
Replying to: User2
Hey, I have made the...

Yeah, good job!

- **Markdown Support:** The message box has two tabs - Message and Markdown Preview. In the Markdown Preview tab, the user can preview how the markdown looks before sending the message. The markdown python package is used to get HTML from text content and display in the preview tab.



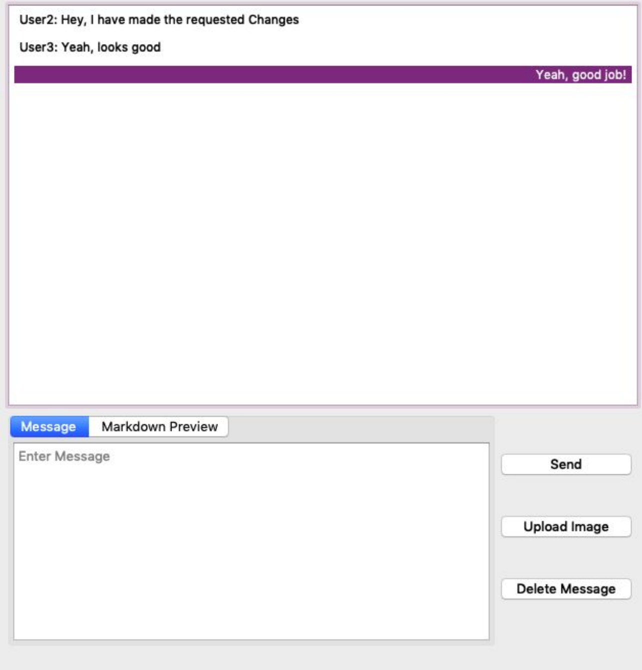
```
## The tasks left to be done are:
- Complete Proposal
- Get Proposal Reviewed
```



The tasks left to be done are:

- Complete Proposal
- Get Proposal Reviewed

- **Message Delete:** The delete button is disabled by default. When a user selects his/her own message, the delete button gets enabled and the user can click it to delete the message. Invokes the */delete_message* endpoint to delete the message from the DB and emit a socket event to remove the message from every user’s chat window.

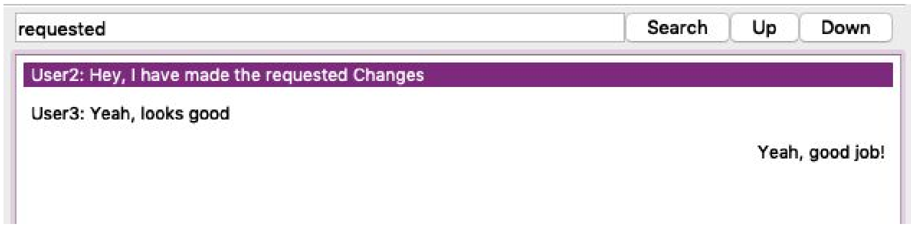


User2: Hey, I have made the requested Changes

User3: Yeah, looks good

Yeah, good job!

- **Message Search:** To Search a message the user needs to type in the text and click on the search button. This would take the user to the most recent message matching the search string. The user can use the up and down buttons to go to a previous or more recent message which matches the search string.



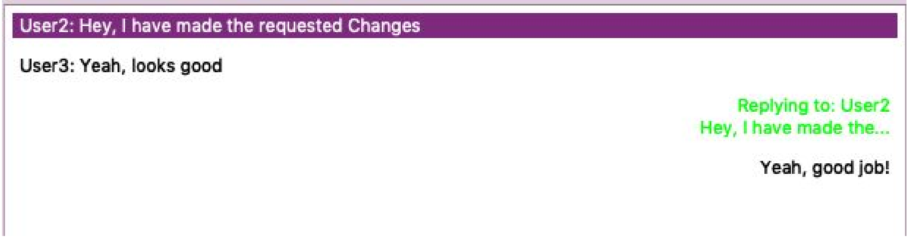
requested

User2: Hey, I have made the requested Changes

User3: Yeah, looks good

Yeah, good job!

- **Message Reply:** If a message is selected and then the send button is pressed the UI sends the selected message’s id as the reply_id to the server. This way replying to a specific message can be done. Clicking on the message can scroll you to the replied message.



User2: Hey, I have made the requested Changes

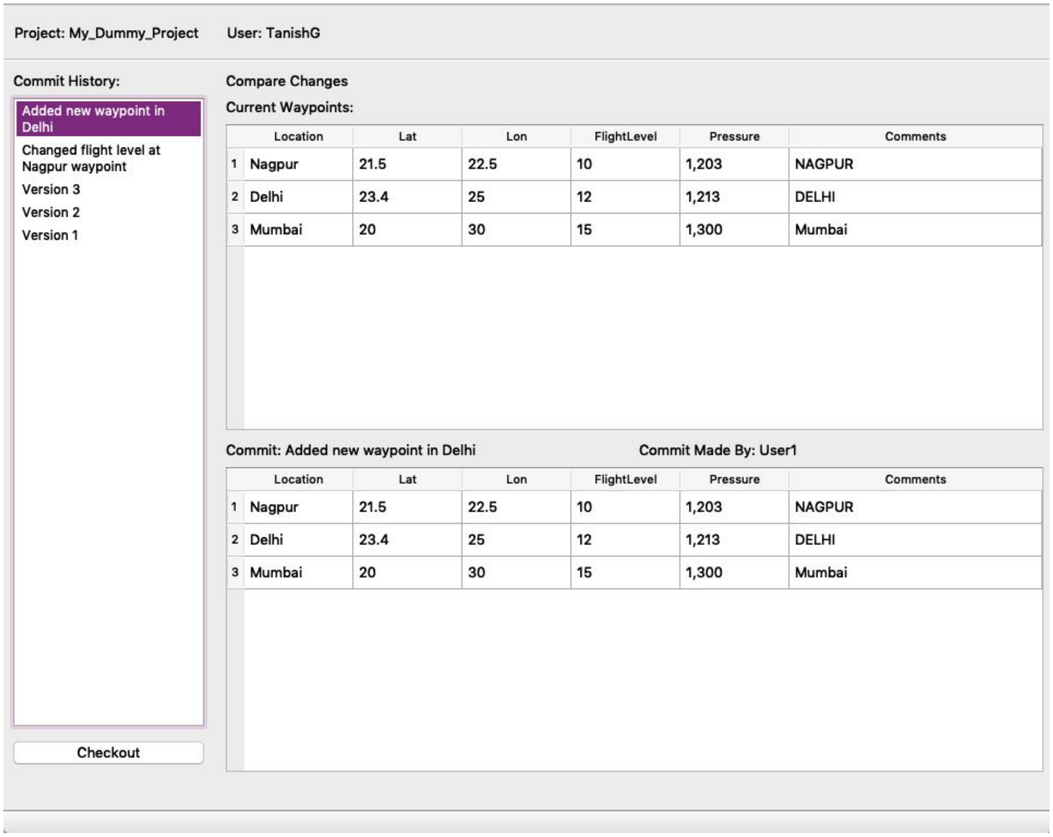
User3: Yeah, looks good

Replying to: User2
Hey, I have made the...

Yeah, good job!

Commit History Window:

The commit history section would be removed from the chat window and shifted to a separate window of its own.



Project: My_Dummy_Project User: TanishG

Commit History:

- Added new waypoint in Delhi
- Changed flight level at Nagpur waypoint
- Version 3
- Version 2
- Version 1

Compare Changes

Current Waypoints:

	Location	Lat	Lon	FlightLevel	Pressure	Comments
1	Nagpur	21.5	22.5	10	1,203	NAGPUR
2	Delhi	23.4	25	12	1,213	DELHI
3	Mumbai	20	30	15	1,300	Mumbai

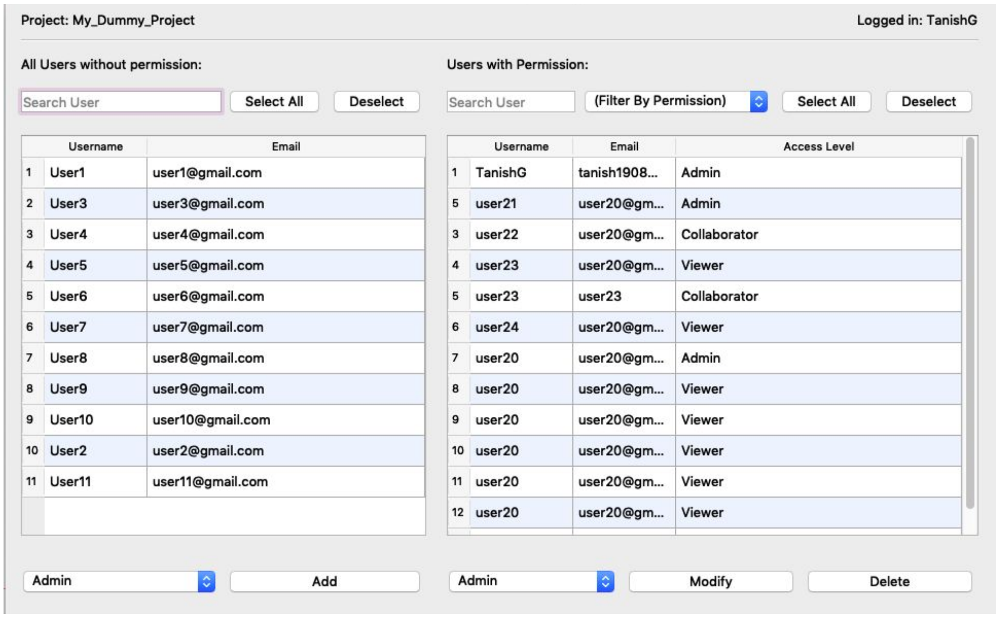
Commit: Added new waypoint in Delhi Commit Made By: User1

	Location	Lat	Lon	FlightLevel	Pressure	Comments
1	Nagpur	21.5	22.5	10	1,203	NAGPUR
2	Delhi	23.4	25	12	1,213	DELHI
3	Mumbai	20	30	15	1,300	Mumbai

This window will be using the existing APIs in mscolab server but would display the change in a better and cleaner format, showing the difference in data through tables between the current version and the selected commit version.

Admin Dashboard:

I would be developing a new window for the admin dashboard. The UI for the window is given below. Two API calls are made on window load: */get_users_without_permission* and */get_users_with_permission* and are used to populate the two tables.



Project: My_Dummy_Project

Logged in: TanishG

All Users without permission:

	Username	Email
1	User1	user1@gmail.com
2	User3	user3@gmail.com
3	User4	user4@gmail.com
4	User5	user5@gmail.com
5	User6	user6@gmail.com
6	User7	user7@gmail.com
7	User8	user8@gmail.com
8	User9	user9@gmail.com
9	User10	user10@gmail.com
10	User2	user2@gmail.com
11	User11	user11@gmail.com

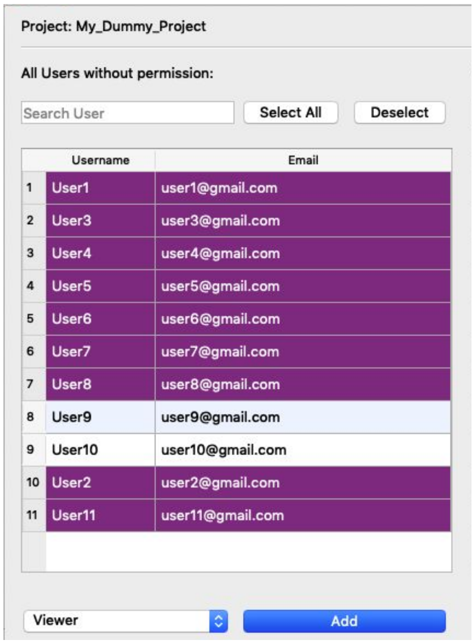
Admin

Users with Permission:

	Username	Email	Access Level
1	TanishG	tanish1908...	Admin
5	user21	user20@gm...	Admin
3	user22	user20@gm...	Collaborator
4	user23	user20@gm...	Viewer
5	user23	user23	Collaborator
6	user24	user20@gm...	Viewer
7	user20	user20@gm...	Admin
8	user20	user20@gm...	Viewer
9	user20	user20@gm...	Viewer
10	user20	user20@gm...	Viewer
11	user20	user20@gm...	Viewer
12	user20	user20@gm...	Viewer

Admin

- **Adding Users:** Multiple users can be selected at once from the table. An access level for these users is selected and clicking on the “add” button invokes */add_permissions* endpoint sending it the array of selected user ids.



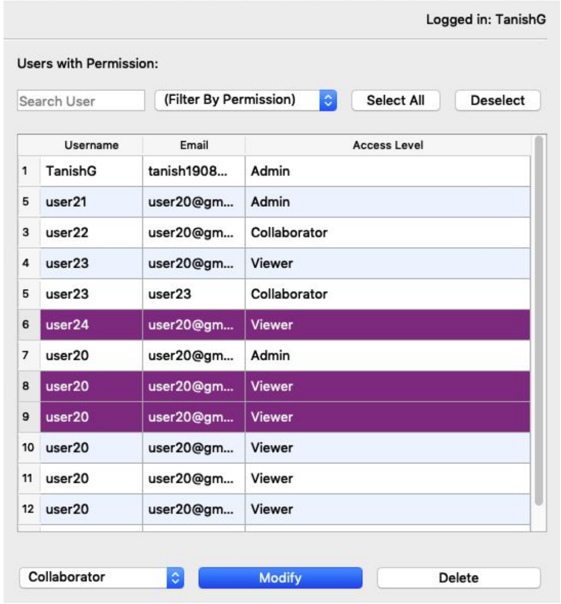
Project: My_Dummy_Project

All Users without permission:

	Username	Email
1	User1	user1@gmail.com
2	User3	user3@gmail.com
3	User4	user4@gmail.com
4	User5	user5@gmail.com
5	User6	user6@gmail.com
6	User7	user7@gmail.com
7	User8	user8@gmail.com
8	User9	user9@gmail.com
9	User10	user10@gmail.com
10	User2	user2@gmail.com
11	User11	user11@gmail.com

Viewer

- **Modifying or Deleting User Permissions:** Similar to adding users, multiple users can be selected from the added users table and their permissions can be modified or deleted in bulk. Invokes */modify_permissions* for modifying and */revoke_permissions* for deleting permissions.



	Username	Email	Access Level
1	TanishG	tanish1908...	Admin
5	user21	user20@gm...	Admin
3	user22	user20@gm...	Collaborator
4	user23	user20@gm...	Viewer
5	user23	user23	Collaborator
6	user24	user20@gm...	Viewer
7	user20	user20@gm...	Admin
8	user20	user20@gm...	Viewer
9	user20	user20@gm...	Viewer
10	user20	user20@gm...	Viewer
11	user20	user20@gm...	Viewer
12	user20	user20@gm...	Viewer

The logic for other buttons like search, select all/deselect, filter etc is handled in the frontend to make the process more quick and seamless.

Timeline

The following is my planned schedule as per GSoC's timeline. My work primarily consists of building 3 components. The most complex one being the new **flight path editing system** because it requires a lot of changes in the existing code base itself so new bugs can appear in different parts of the application dependent on this code. Hence, I will start out by working on the **Admin Dashboard**, a completely new component, then the **chat system** and finally when I am most familiarized with the whole codebase start working on the **flight path editing system**.

I will be working on all weekdays and if need arises I am also free to work on weekends. This includes all holidays. I have added a buffer period after finishing each component to complete any backlogs, refactoring code, fixing any new bugs and improving tests.

I will keep the last 1 and a half weeks free for bug fixing, increasing test coverage and improving documentation.

Time Span	Work
4th May	Accepted Student Proposals Announced
4th May - 1st June	Community Bonding Period

4th May - 5th May (2 days)	Setup: ➔ Decide schedule and mode of communication for weekly and emergency meetings. ➔ Setup a blog for GSoC.
6th May - 20th May (2 weeks)	Finalize proposed changes with mentors: ➔ Discuss with the mentors about any changes or improvements needed in the proposed design and draft a final design document. ➔ Solve some existing issues in version 1.9.2 of MSS. ➔ Improve the test coverage of mscolab.
21st May - 31st May (10 days)	Diving deep into the codebase: ➔ Get familiarized with the existing code base in depth. ➔ Make a rough note of all the functions and code that would require modifications after implementing the proposed ideas to catch errors early.
1st June	Official Coding Starts
1st June - 7th June (1 week)	Get Started with Admin Dashboard: ➔ Start with adding the dashboard window to MSUI. ➔ Implement the APIs for fetching users with and without project permissions in mscolab server. ➔ Implement APIs for adding, modifying and deleting the project permissions in mscolab server.
8th June - 14th June (1 week)	Continue Development of Admin Dashboard: ➔ Add button click listeners and connect the APIs with the frontend. ➔ Implement the filtering and search user logic in MSUI. ➔ Write new unit tests for the admin dashboard

	and improve existing tests.
15th June - 17th June (3 days)	Buffer Period to work on any backlogs, fix new bugs and improve unit tests.
18th June - 21st June (4 days)	Start Development of Chat System: ➔ Redesign the project chat window for mscolab in MSUI. ➔ Remove the commit history section from the window. ➔ Update Message Database Table to include the discussed attributes. ➔ Use python markdown package to convert messages into HTML and display using QTextEdit widget in the message box.
22nd June - 28th June (1 week)	Work on modifying the ChatManager Class: ➔ Develop the */delete_message* endpoint in mscolab server. ➔ Update *add_message* method in ChatManager class to handle saving images on the server and saving the reply_id if there is one. ➔ Update the *get_message* method in ChatManager class to handle retrieving the message attachment and reply_id if any. ➔ Update all the socket event handlers in SocketManager class to include the new data. ➔ Add a file picker popup for uploading images.
29th June - 3rd July	First Round of Evaluations
29th June - 5th July (1 week)	Work on connecting the chat window frontend to the mscolab server: ➔ Add button click listeners in the frontend and connect the APIs to it. ➔ Work on search message logic in the frontend. ➔ Create a new commit history window and connect it with existing APIs. ➔ Write unit tests

6th July - 8th July (3 days)	Buffer Period to work on any backlogs, code refactoring, fixing new bugs and improve unit tests.
9th July - 19th July (1.5 weeks)	Start work on the new flight path editing system: ➔ Update the mscolab main window UI to include buttons for the proposed changes. ➔ Update UserSetting Table in database to include *work_offline* attribute. ➔ Write endpoints in mscolab server to handle enabling and disabling of auto sync and work offline. ➔ Make changes in the *WaypointsTableModel* to handle work offline mode along with autosync. ➔ Change temporary_mscolab.ftml logic to handle a separate file for each project.
20th July - 26th July (1 week)	Connect the developed APIs to mscolab frontend: ➔ Update the get project details api (*/project_details*) to include the auto sync value for the project and the user's work offline mode value. ➔ Add click listeners for the auto sync and work offline mode to connect them with the APIs. ➔ Add necessary socket events to enable work offline for every user when auto sync is turned off. ➔ Add a new unit tests and update the existing ones to accommodate the new changes.
27th July - 31st July	Second Round of Evaluation
27th July - 2nd August (1 week)	Develop the Merge Window for merging local and server waypoint data: ➔ Add the *Merge Window* in MSUI to show up every time push or fetch buttons are clicked. ➔ Develop */fetch_waypoints* API to get the waypoints in the server .ftml file for the project. ➔ Add logic in Merge Window to fetch waypoints from the local project .ftml file and connect the */fetch_waypoints* API to populate

	the 2 waypoint tables. ➔ Add Logic for picking out rows from the 2 tables to add in a final waypoints table.
3rd Aug - 9th August (1 week)	Finish Work on Merge Window and start work on manual commit functionality: ➔ Add functions for updating local .ftml file for fetching waypoints. ➔ Modify the current save waypoints function to handle the new changes. ➔ Add click listeners to the merge window to connect these functions to the frontend. ➔ Develop a */commit_changes* endpoint to make to handle making commits to the server file.
10th August - 16th August (1 week)	Finish manual commit functionality and write tests: ➔ Add a dialog box popup for admins to enter a commit message when clicking the commit changes button. ➔ Connect the */commit_changes* API to the frontend. ➔ Write new unit tests and improve the existing ones.
17th August - 19th August (3 days)	Buffer Period to work on any backlogs, code refactoring, fixing new bugs and improve unit tests.
20th August - 23rd August(4 days)	Done with all the proposed functionality, work on testing and improving code performance. ➔ Implement integration testing. ➔ Find and fix more bugs if found. ➔ Find areas to improve code efficiency and refactor code. ➔ Document code and integrate documentation to that of MSS.
24th August - 31st August	Work Submission
24th August - 31st August (1 week)	➔ Ask the mentors for a more detailed review, and work on fixes covering code, documentation, tests etc. ➔ Project and documentation submission.

31st August - 7th September	Mentors Submit Final Evaluation
8th September	Final Results Announced.

Future Work

I plan to continue working on MSS after GSoC in my spare time. Some features which I hope to work on after GSoC are:

- **Mscolab Made the Primary UI:** After my work for GSoC is completed, mscolab would have really become a mature application and can be shifted from a secondary feature to a primary part of MSS, becoming the starting window for MSS.
- **Different File Support in Mscolab:** Currently mscolab uses a .ftml file to store the data in XML format. There are many different formats available which researchers use for geospatial data like geoJSON, .shp etc. Extending support for these files would broaden the use-case for mscolab.
- **Improving flight path views in MSS:** More views that allow researchers to visualise the flight path and atmospheric data in a 3D fashion would greatly help them analyse the flight path. Adding such a view in mscolab would require minimal modifications to mscolab itself apart from developing the view window.

Other Commitments

- If I am selected for GSoC, it will be my full-time commitment. My college would resume from the last week of July but that would not affect the number of hours I would work on my project. I would be working for 40 hours a week on average.
- Due to the ongoing COVID-19 pandemic, I am currently not aware of when my university exams for this semester will take place. My university exams last for a week and If they are shifted to a date which coincides with my GSoC timeline, I would be working a little less that week. However, I have given myself enough buffer time in my schedule and distributed my work well in my timeline to easily get back on track if such a situation arises.
- Python Software Foundation is the only organisation I am applying for in GSoC. I am only submitting this single proposal for the Mission Support System sub-org under Python Software Foundation.