CS3001- Software Engineering (Jan-May 2024)

Final Project Report (Milestone 6)

Group 21

- Isha Nayar : CS21B035
- Janapati Varshita Devi : CS21B036
- Kumar Kshitiz Singh : CS21B044
- Lagudu Sree Teja Vardhan : CS20B046
- Nisanth D : CS20B057

PROBLEM STATEMENT

Online support ticket system for the IITM BS degree program : Integration with Discourse and webhook.

The support team at the IITM BS degree program often gets overwhelmed with emails from students regarding queries and concerns. Your task is to create an online support ticketing system for the IITM BS degree program. Students can create a support ticket for a particular concern or query. Before they create a ticket, the system should also show a list of similar tickets, and allow users to like or +1 an already existing support ticket, so that duplicates are not created. This way popular concerns or queries can be prioritized by the support team.
After the support team addresses the concern, they can mark the ticket as resolved, and an appropriate notification should be sent to concerned users.

Another important feature of the ticketing system is dynamic FAQ updation. Many student concerns can be FAQs which will be useful for future students. If appropriate, the support query and response should be added to the FAQ section by support admins, and appropriately categorized, so that an updated FAQ will be readily available to students. The platform should allow users to enroll as students, support staff and admins.

a. ***Discourse Integration:*** *In addition to the creation and listing of tickets, your system should also have the provision to create a Discourse thread for each ticket. This will involve integrating the Discourse system with the IITM BS ticketing system. You can think of different rules and configurations to create/edit/modify Discourse threads (for example - the thread can initially be private, but then can be converted to a public topic by moderators (link)). You can also think of additional features like notifications when a thread which you created has received a reply, like etc.*

b. ***Webhooks Integration:*** *Certain tickets can be high priority and need to be addressed immediately. In such cases, the system has to integrate with webhooks to notify High Priority and Urgent tickets into GChat. This can then be used by higher authorities for handling escalations.*

LIST OF TABLES

MILESTONE 1:

User Stories

User Stories for discourse integration:

Primary Users : Students, Support team
Secondary Users : Admins

USER	USER STORY
STUDENT	As a student, I want to be able to create a discourse thread for each ticket in portal, So that I can effectively communicate with support staff and obtain additional assistance or insights.
	As a student, While creating a ticket, I want to be able to choose if I want a new thread or add it to an existing thread, So that it provides me flexibility to choose convenient options
	As a student, I want the ability to initiate a new discourse thread for a pre-existing ticket, So that I can start a conversation even after creating a ticket previously.
	As a student, I want the Discourse thread related to support ticket to be initially private, So that I can discuss my concern confidentially with the support team before it is made public.
	As a student, I want the ability to view all public discourse threads, So that I understand the issues other students are facing within the community.

STUDENT	As a student, I want the Discourse thread associated with my support ticket to be automatically closed or archived when the ticket is marked as resolved, So that discussions are concluded appropriately along with the resolution of the issue.
	As a student, I want to be able to reopen my closed thread, So that if I'm not satisfied with the solution, I can approach for more help.
	As a student, I want to know the status of my query through notification, So that I can know how long it will take to resolve.
	As a student, I want to be able to choose if my thread will be public or private, so that I can control the visibility of my posts and maintain my privacy.
SUPPORT STAFF	As a support staff I want the student to receive search results of similar discourse threads while creating a new ticket on the portal. So that repeated queries are avoided and I get to spend my time somewhere better.
	As a Support staff, I want to have the ability to merge threads of related tickets into a single thread, So that repeated queries on similar topics are consolidated, reducing redundancy and improving efficiency.

SUPPORT STAFF	As a Support staff, I want to notify the status of query to the student on discourse, So that they can know when the ticket will be resolved.
	As a Support staff, I want to get a notification if a thread has been created for a ticket, So that I can follow up on it further.
	As a Support staff, I want to create a Discourse thread for a support ticket, So that discussions related to the ticket can be easily managed and referenced.
	As a Support staff, I want to be able to close a thread, When the common issue in the thread is resolved.
	As a Support staff, I want to be able to be able to reopen a thread, When the issue can be solved further and better.
	As an Admin, I want to be able to see the most active thread, So that I can turn it into a FAQ documentation for students.
ADMIN	As an Admin, I want to have a creator access to a thread, So that I can delete any message that is offensive or irrelevant.

ADMIN	As an Admin, I want to have a thread to support staff mapping So that the same support staff can be allocated for similar requests.
	As an Admin, I want to have a mapping of the discourse threads and the corresponding tickets So that I can ensure orderly resolution and delete any message that is offensive or irrelevant.

WEBHOOKs INTEGRATION

Primary Users : Students, Support team
Secondary Users: Admin
Tertiary Users: Future students who are going to use the ticketing system, Software developers.

USER STORIES FOR WEBHOOK INTEGRATION

USER	**USER STORY**
STUDENT	As a student, I want to be able to have a faster mode of ticket resolution, So that my very important and high-priority tickets can be resolved sooner.
SUPPORT STAFF	As a support Staff, I want to be able to mark a ticket as a high priority, So that it will send a notification to the higher authorities via Gchat.

MILESTONE 2:

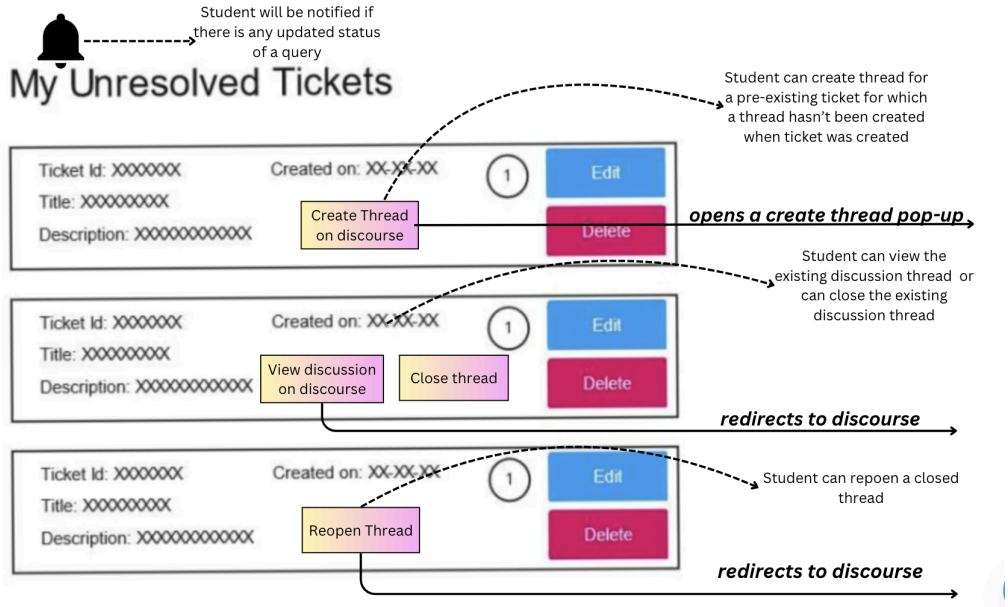
Making wireframes and Storyboard

WIREFRAMES BASED ON USER STORIES

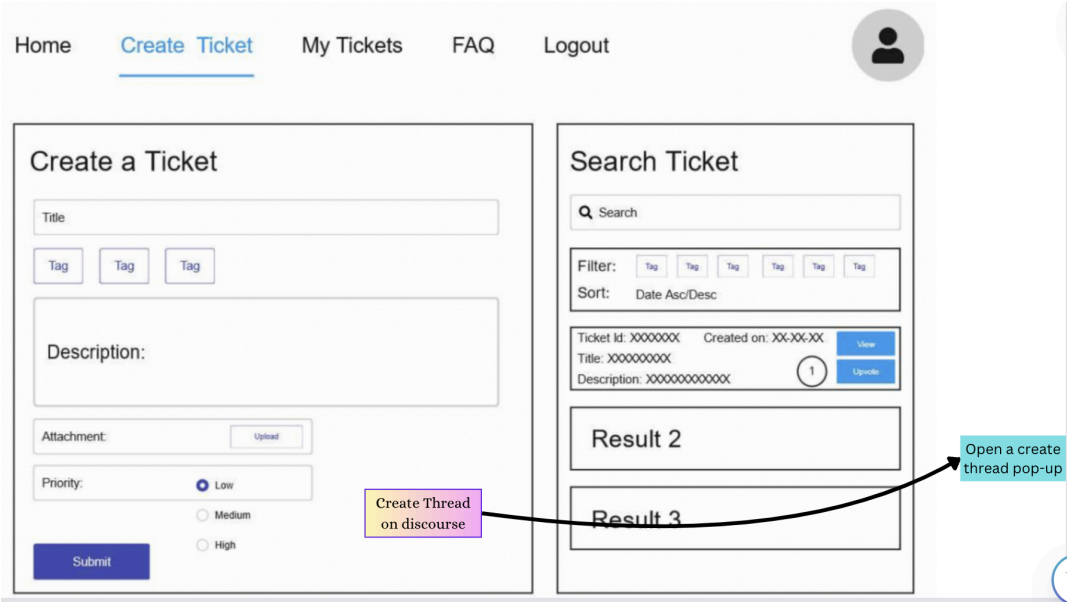
Discourse Integration

2.1.1] USER: STUDENT

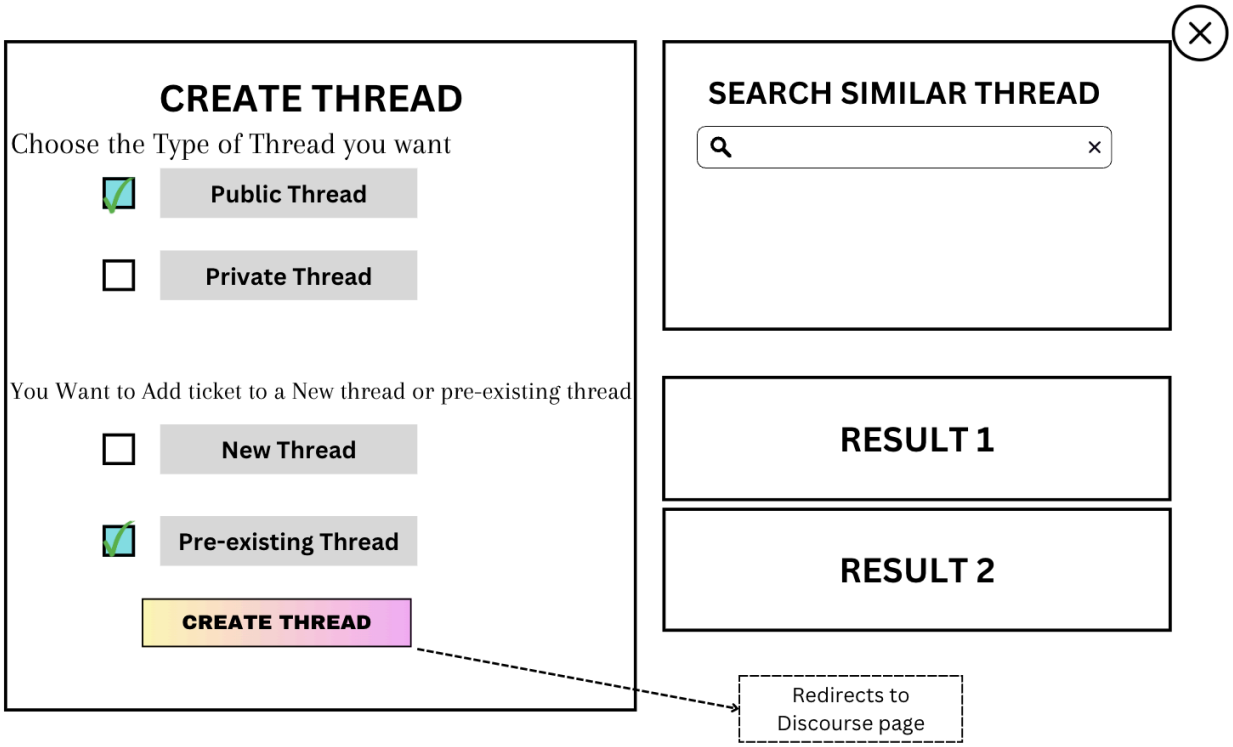
[1.1]Unresolved tickets page



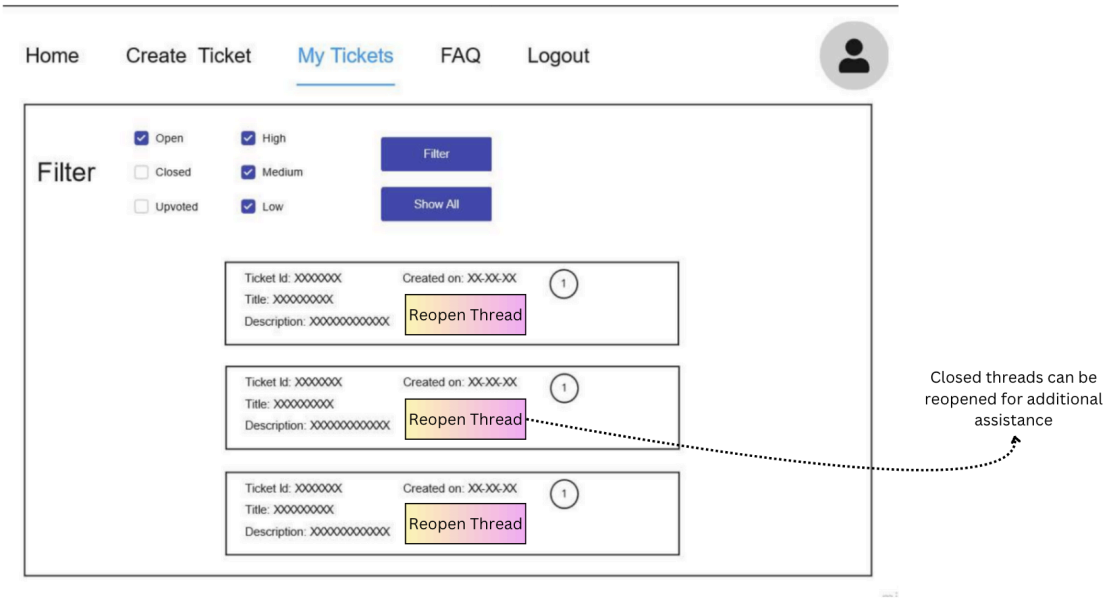
[2.2]Create ticket page



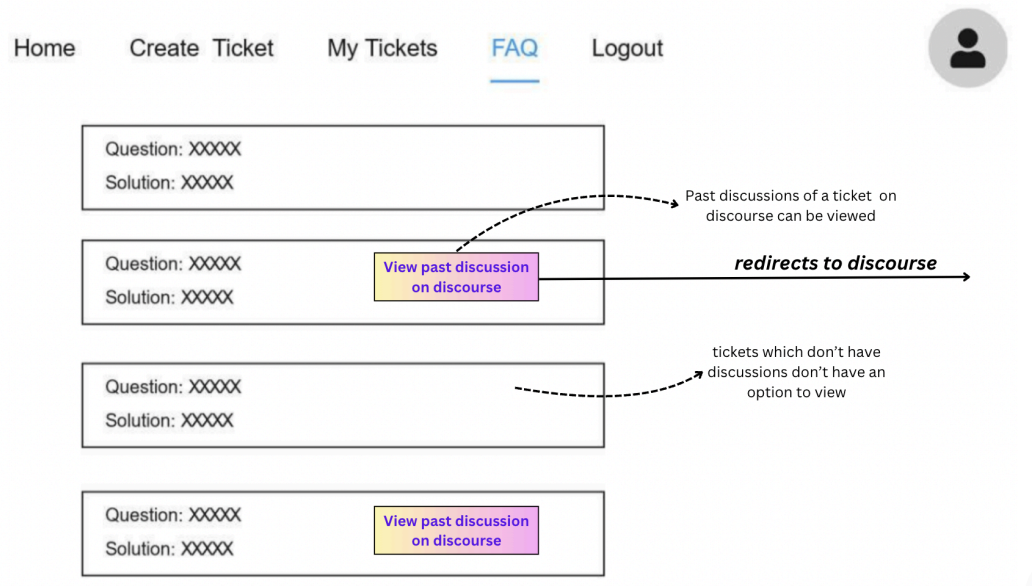
[2.3] Create thread page



[2.4]My tickets page

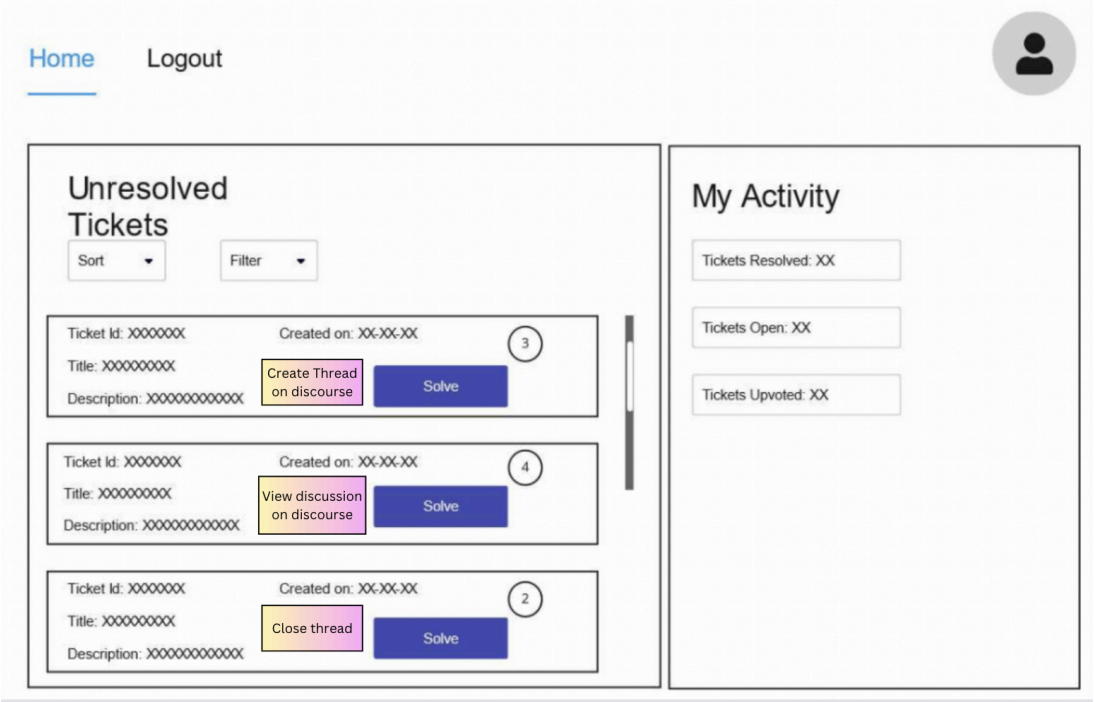


[2.5] FAQ Page



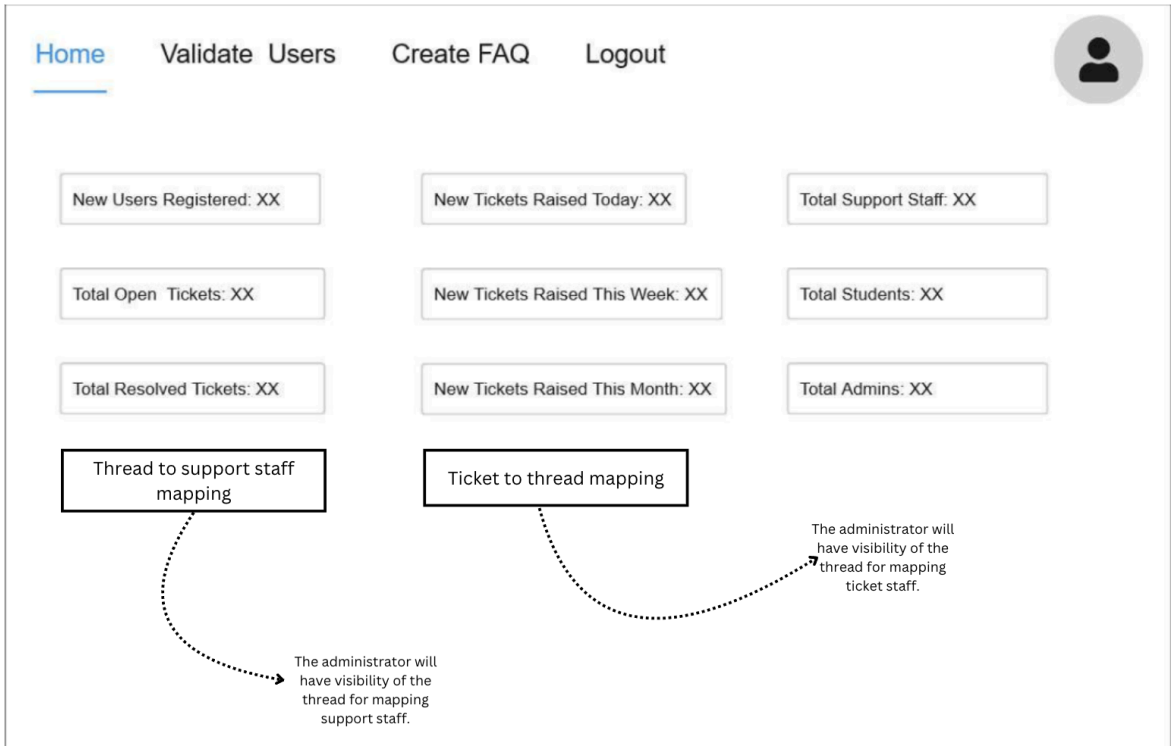
2.1.2] USER: Support Staff

[2.6] Home page



2.1.2] USER: Admin

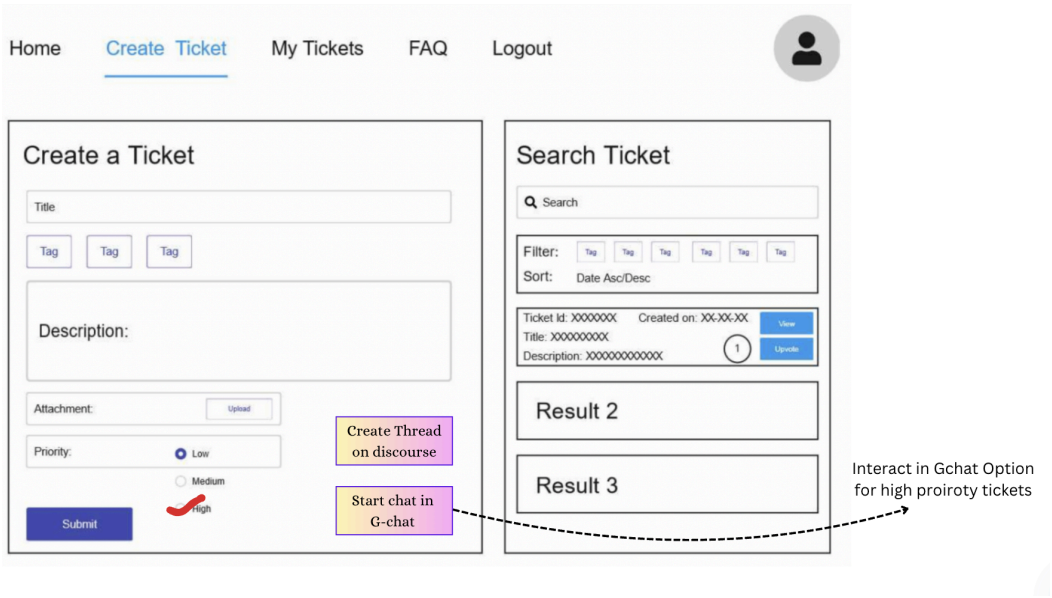
[2.7] Home page



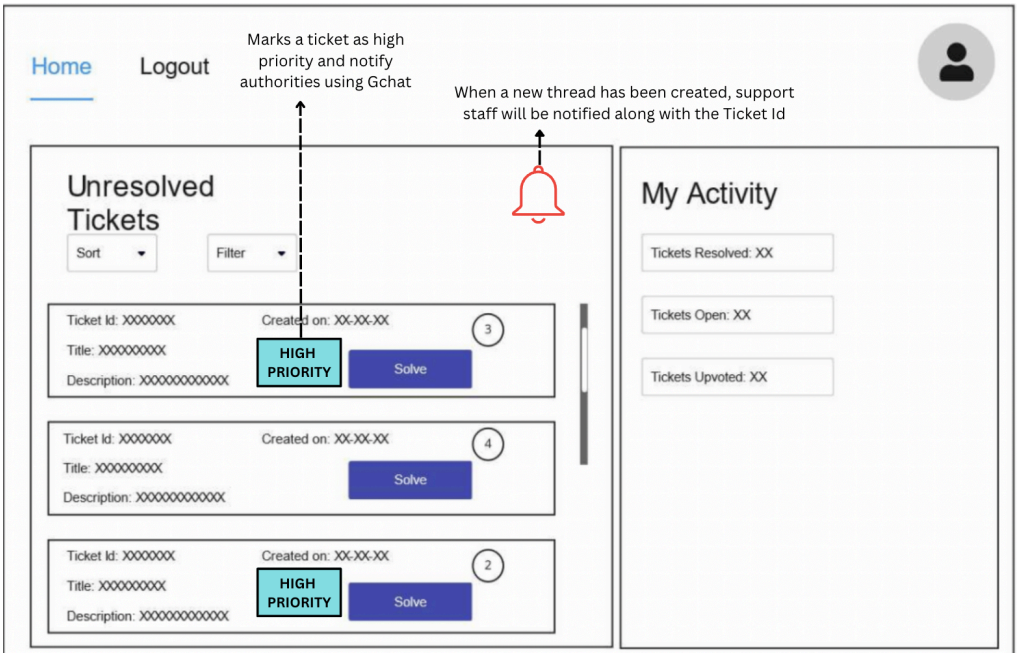
Webhook Integration

2.2.1] User: Student

[2.8] Create ticket and start Gchat

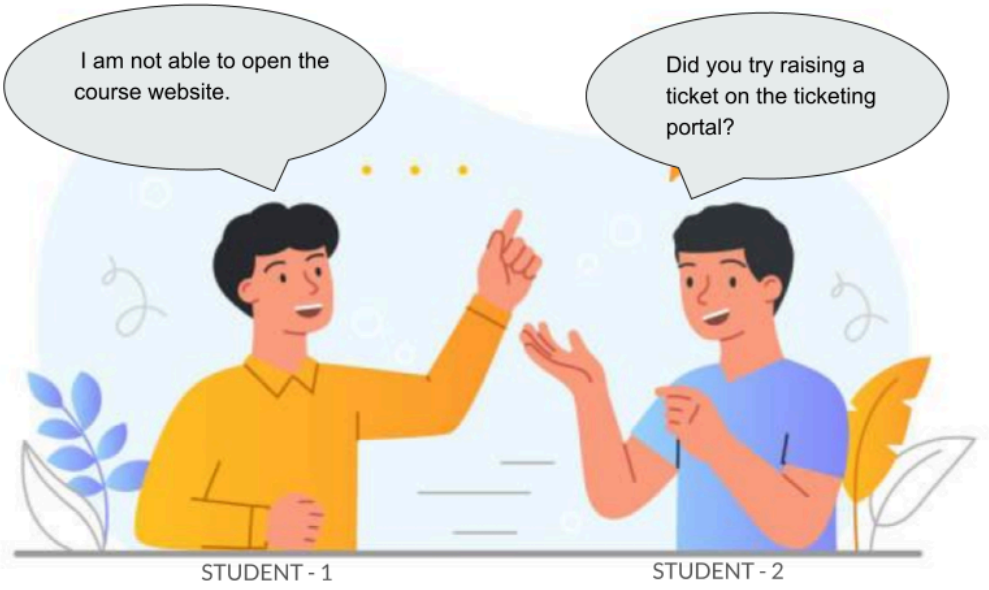


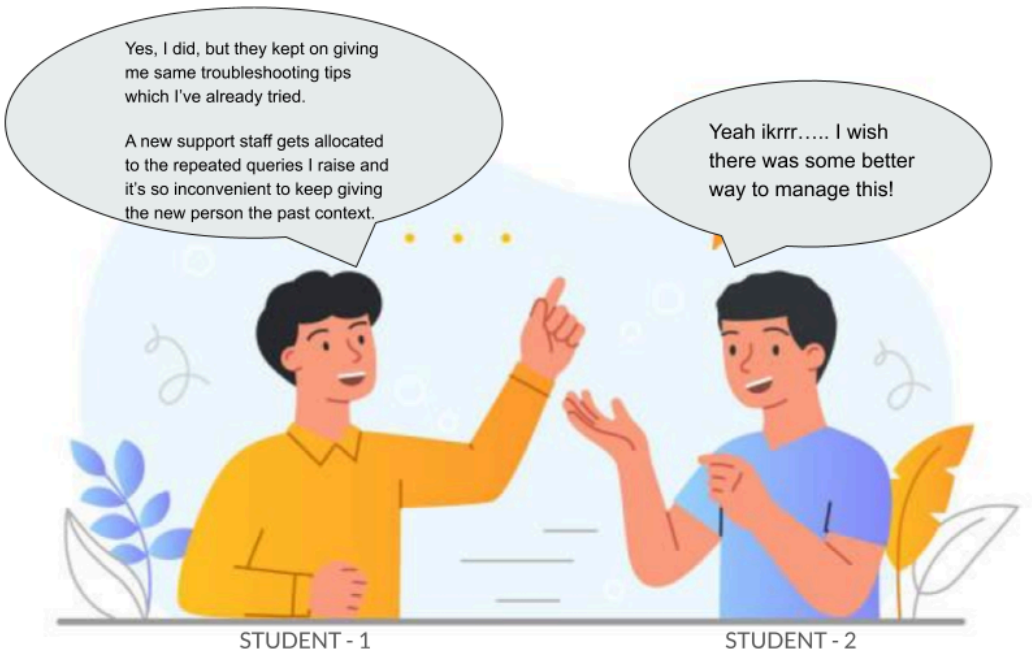
2.2.2] User: Support Staff



STORYBOARD

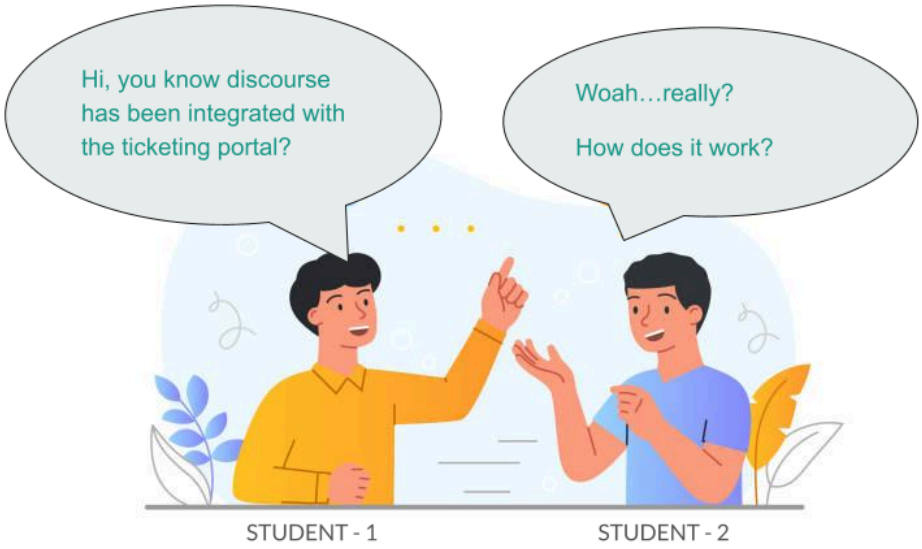
Let's have a look at 2 students discussing their problems…


I am not able to open the course website.
Did you try raising a ticket on the ticketing portal?
STUDENT - 1
STUDENT - 2

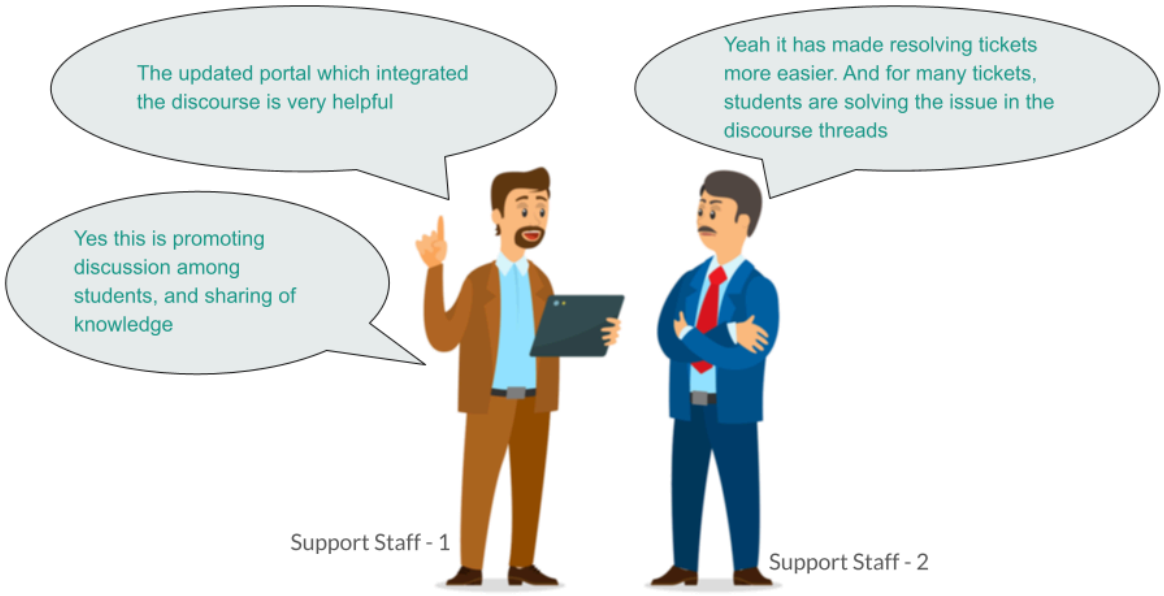

Yes, I did, but they kept on giving me same troubleshooting tips which I've already tried.
A new support staff gets allocated to the repeated queries I raise and it's so inconvenient to keep giving the new person the past context.
Yeah ikrrr….. I wish there was some better way to manage this!
STUDENT - 1
STUDENT - 2

Two support staffers in a discussion….

In an alternate scenario……..


Hi, you know discourse has been integrated with the ticketing portal?
Woah…really?
How does it work?
STUDENT - 1
STUDENT - 2


It helps me keeping all my queries related to the same ticket at one place, for the support staff to see.
And it's so convenient to converse with the support team this way
Wow! But what if the query is still unresolved?
The ticket can be escalated to higher authorities then.
STUDENT - 1
STUDENT - 2


The updated portal which integrated the discourse is very helpful
Yeah it has made resolving tickets more easier. And for many tickets, students are solving the issue in the discourse threads
Yes this is promoting discussion among students, and sharing of knowledge
Support Staff - 1
Support Staff - 2

MILESTONE 3:

Scheduling and Design

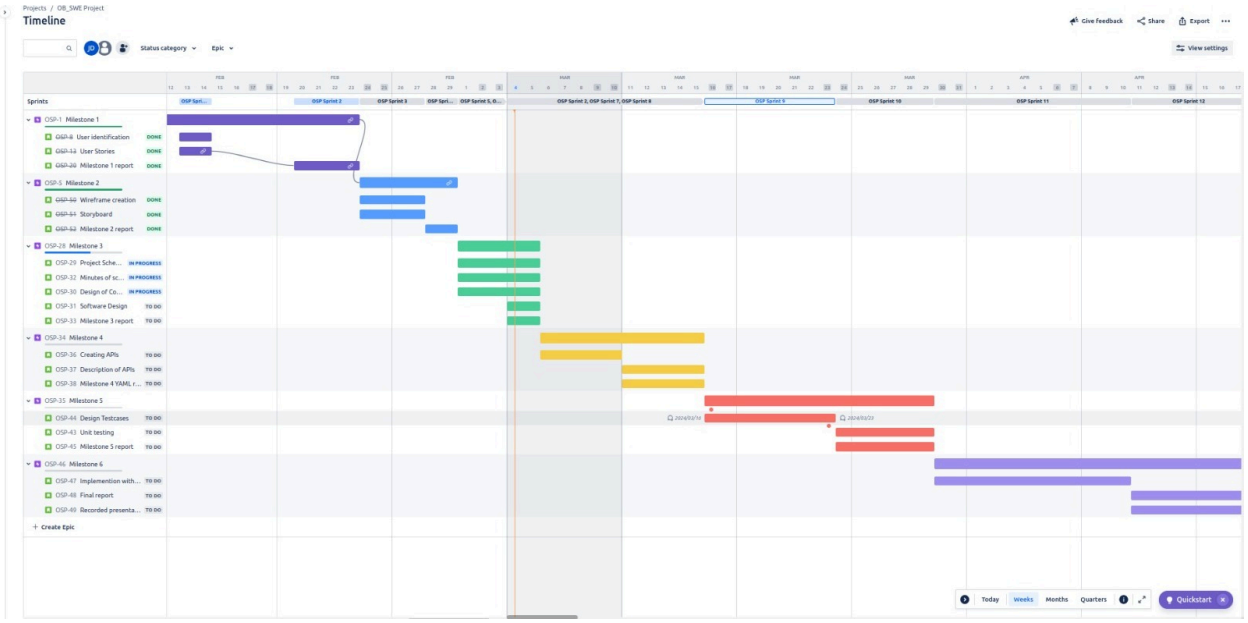
3.1 Project Schedule

3.1.1 Task Distribution

Milestone	Sub Task	Sprint	Assigned to
1- User Requirement	User identification	1	All
	User Stories	1	All
	Report	2	Kshitiz, Isha,Varshita
2-User Interface	Wireframes	3	Varshita,Isha
	Storyboards	3	Kshitiz,Nishanth,Teja Vardhan
	Report	4	All
3- Project scheduling	Project Scheduling	5	Isha,Varshita
	Component Design	5	All
	Software design	6	Nishanth,Teja Vardhan,Kshitiz
	Scrum meetings	6	Isha,Kshitiz
	Report	6	All
4-API	Design API	7	Varshita,Isha
	Code Review	8	Nishanth,Teja Vardhan,Kshitiz
	YAML Document	9	All
5-Testing	Test case Design	10	Nishanth,Teja Vardhan,Kshitiz
	Unit testing	11	Varshita,Isha
	Report	11	All
6-Submission	Frontend Design	12	Varshita,Isha,Kshitiz
	Demo	13	Nishanth,Teja Vardhan
	Final Report	13	All
	Presentation	14	All

3.1.2 Schedule using Gantt chart

We divided the milestones into sub-tasks using the SMART guidelines and scheduled our sprint in the form of Gantt chart using the JIRA software tool.The chart is shown below

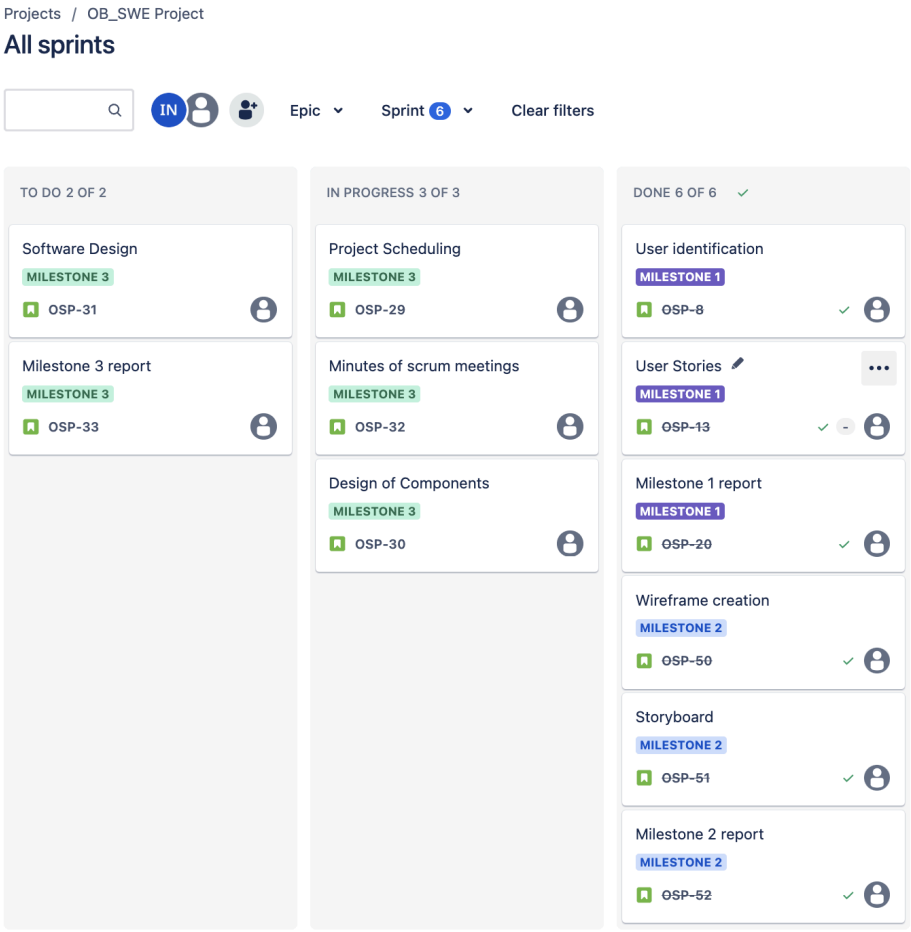


To see the zoom in image of the Gantt chart, click here.

3.1.3 Scrum board(Till Milestone 3)

Based on our Gantt chart and the sprint schedule, we've made the scrum board till Milestone 3. Our Milestone 1 and 2 is completed and Milestone 3 is in progress.

The Scrum board is shown below



3.2 Scrum Meetings

The scrum meetings are kept once every week to discuss the Milestones.

SCRUM MEETING FOR MILESTONE 1(User requirements)

AGENDA	DISCUSSION
• Discussing the overview of the project • Understanding the working of discourse and webhook • Identifying the users • Coming up with user stories	We discussed the problem statement and came up with ideas on how we can incorporate discourse and webhook in our ticketing system. All of us discussed and came up with user stories for the primary, secondary users. In the end, Kshitiz, Varshita,Isha compiled all the stories in our report. All of us were present for the scrum meeting.

SCRUM MEETING FOR MILESTONE 2(User Interface)

AGENDA	DISCUSSION
• Discussing what wireframes and storyboards are. • Creating a template for both which will be common throughout. • Distributing the task	We saw the basic template of the storyboard and wireframes and read through our user stories to note the modifications we'll be needing in the wireframes, We then divided the work. Isha and Varshita took the responsibility of creating wireframes and Kshitiz, Nishanth, Teja worked on the storyboard script. All of us were present for the scrum meeting.

SCRUM MEETING FOR MILESTONE 3(Project Scheduling)

AGENDA	DISCUSSION
• Discuss major milestones • Divide milestones into sub tasks with SMART guidelines. • Decide feasibility and deadlines. • Discuss major components for the project. • Discuss what is a class diagram. • Distribute tasks	During the meeting, we discussed major milestones and divided into sub tasks. The deadlines were set after assigning each task to a person and a sprint. The Gantt chart given to Varshita, Isha, Teja Scrum meetings and project scheduling to Isha. All of us were present for the scrum meeting.

3.3 Components for the project

The major components of the projects are:

1. Student view of the ticketing system.
2. Support Staff view of the ticketing system.
3. Admin view of the ticketing system.
4. Webhook API.

Components Description:

1. Student view of the ticketing system:

- Create Ticket page:
    - Users can create a discourse thread while raising a ticket.
    - Users can make the thread public or private.
    - Users can escalate a ticket using webhooks integration through gchat.
- Unresolved Tickets page:
    - Users can create a discourse thread for existing tickets if not done already.
    - Users can close or reopen an existing thread.
- Home page:
    - Users can see notifications in the notification panel.

2. Support Staff view of the ticketing system:

- Unresolved Tickets page:
    - Creation of a discourse thread option button for tickets without threads to take the discussion to discourse.
    - View the discussion on the discourse option button for tickets with threads already created.
    - Close the thread after the ticket has been solved on the discourse option button.
    - Escalate the ticket by marking it as high priority.
- Home page:
    - User can see notifications in the notification panel.

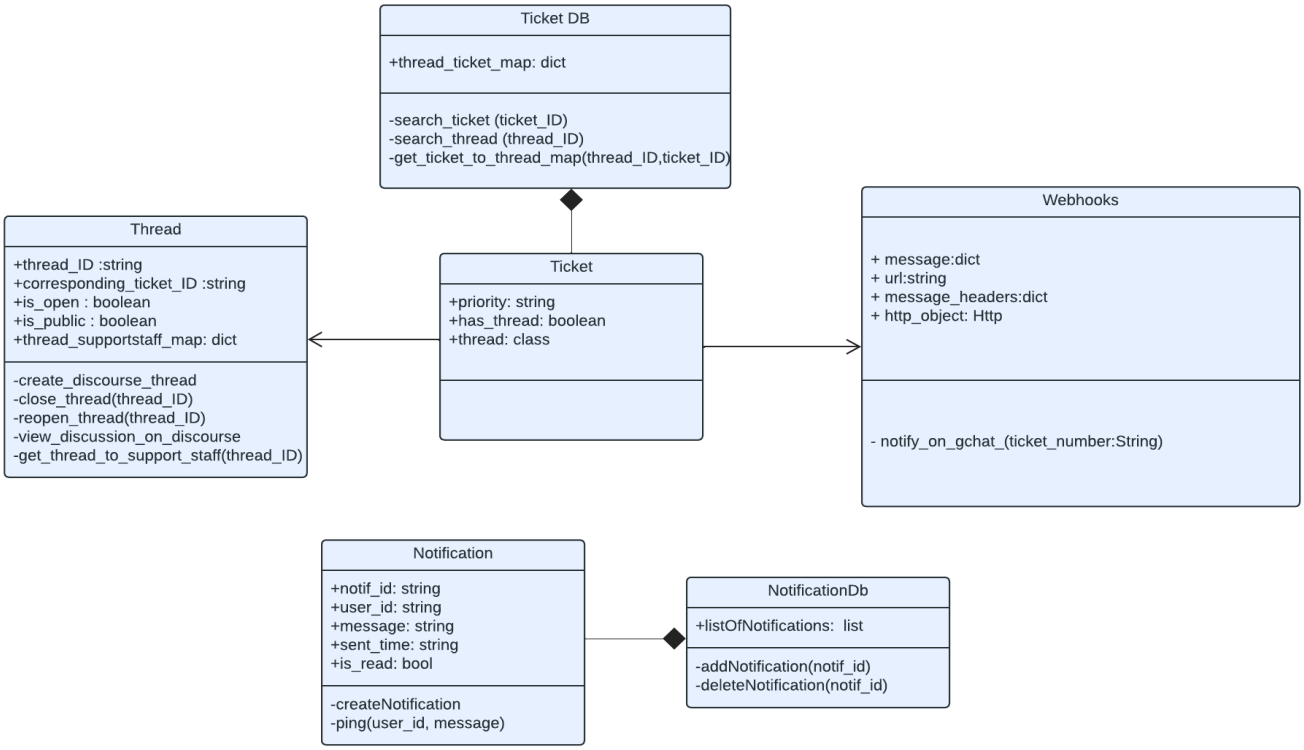
3. Admin view of the ticketing system:

- Home Page:
    - Thread to Support Staff Mapping View to see tickets assigned under each support staff
    - Ticket to Thread Mapping View shows the discourse thread link for each ticket.

4. Webhook API:

- Unresolved Tickets page
    - Support staff can notify higher authorities about an urgent issue via Gchat
- Create Ticket page
    - For high priority tickets, Students can notify support staff via Gchat.

3.4 Class Diagrams

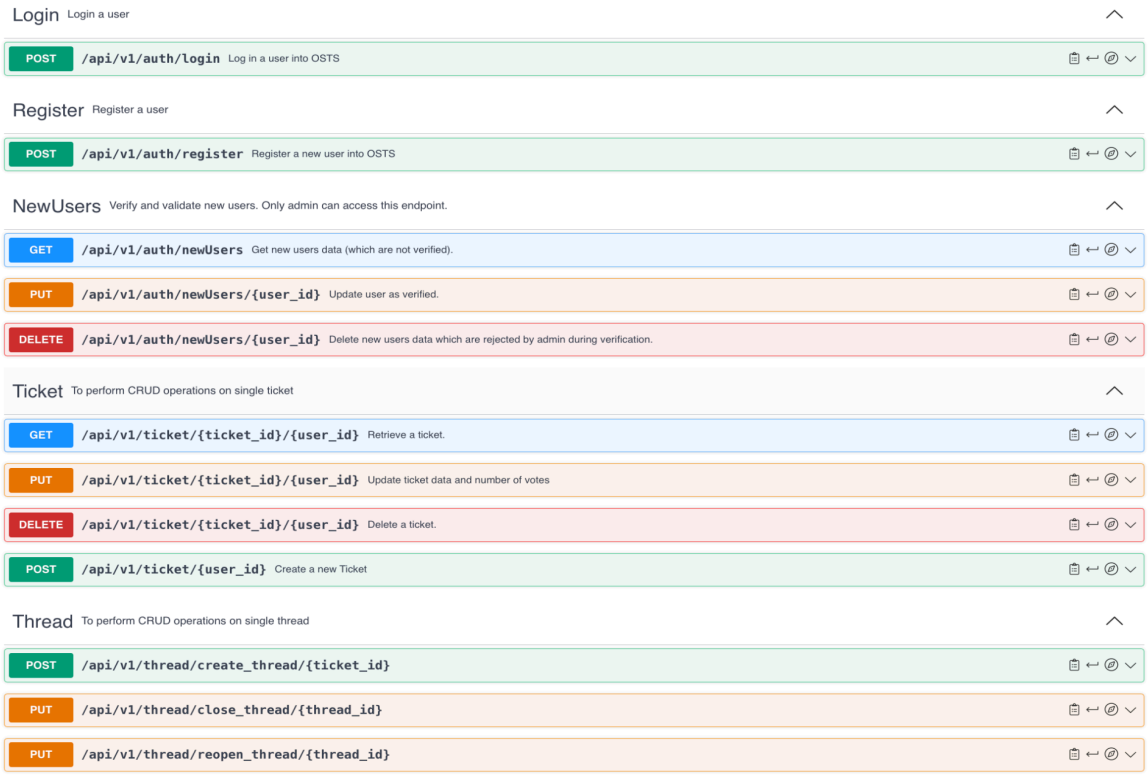


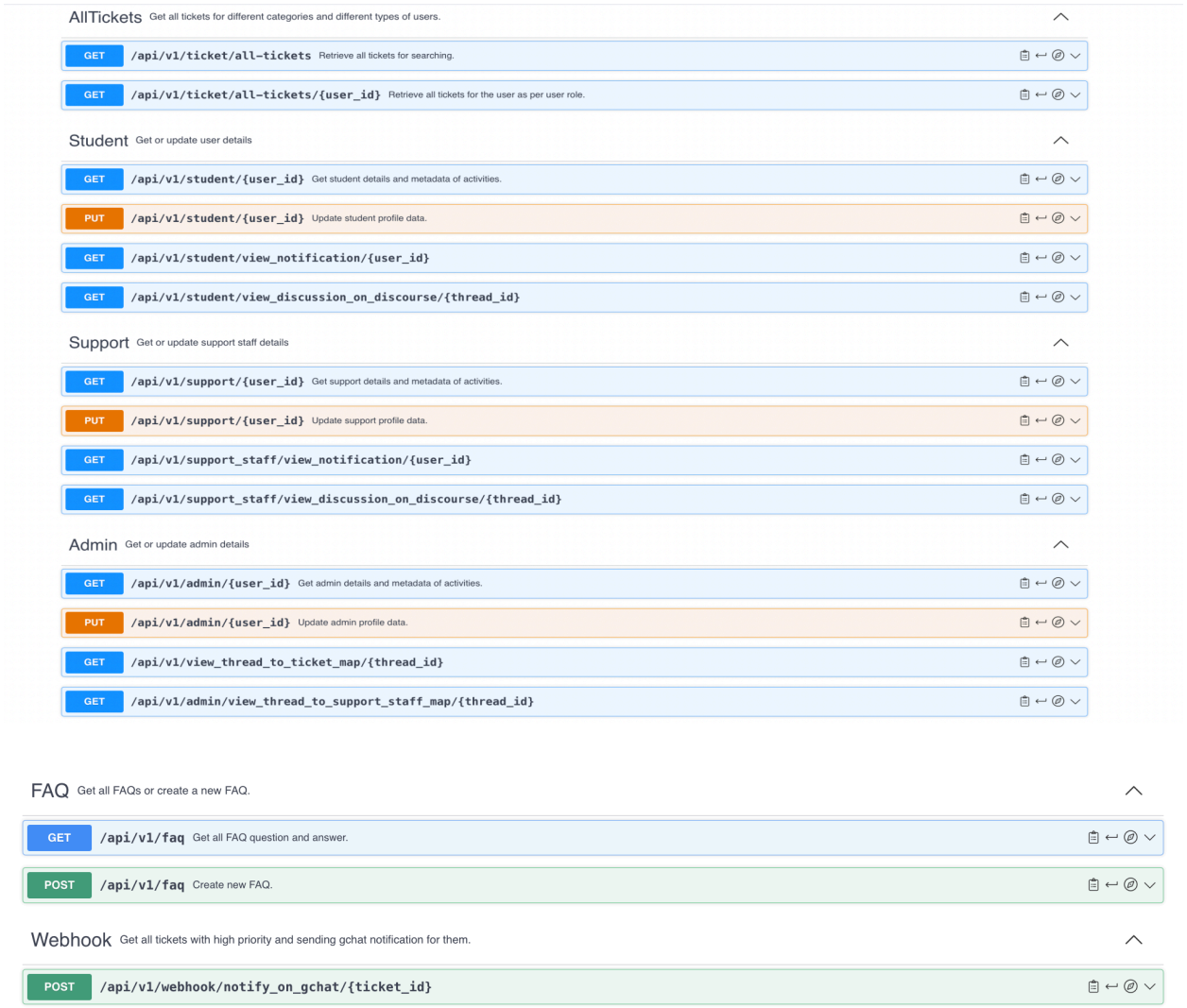
MILESTONE 4:

API Documentation

4. API Documentation

The API routes were defined and the document was created in ‘Swagger’. Below are the few screenshots of API routes





YAML file link: [click here]

MILESTONE 5:

Testing

5.1 App Testing

5.1.1 App Setup

Testing helps ensure that the app will work as expected for the end users.

Software projects which are tested properly and following standard practices, is a good indicator of the quality of the software. Unit tests test the functionality of an individual unit of code isolated from its dependencies. They are the first line of defense against errors and inconsistencies in the codebase.

For this project few of the unit tests are considered. These unit tests consist of testing API endpoints for discourse and webhooks integration for the ticketing system. To carry out testing, ‘PyTest’ python library is used.

The unit tests are divided into 3 major components

- Auth component testing
- Thread component testing

5.1.2 Sample Fixture

We are using the same fixture generated by the existing ticketing system code as part of our API testing. This fixture starts the app with test configuration and within app context the tests are carried out.

```python
from application import create_app
import pytest
from application.logger import logger

# --------------------  Constants  --------------------

# Please set following required constants to mimic a specific user role.

# STUDENT
student_user_id = "ccec26f5a52560cd22a2965287dc4ad9"
student_web_token = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6ImJvc

# SUPPORT
support_user_id = "a5997f803b4dfbdb0a7f17b012ca1697"
support_web_token = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6ImJvc

# ADMIN
admin_user_id = "3ad51db3c4defba9c7f1ca7549712e25"
admin_web_token = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6ImFiY2F

# --------------------  Code  --------------------

# before testing set current dir to `\code\backend`
@pytest.fixture(scope='module')
def test_client():
    flask_app = create_app(env_type="test")
    logger.info("Testing fixture set.")

    # Create a test client using the Flask application configured for testi
    with flask_app.test_client() as testing_client:
        # Establish an application context
        with flask_app.app_context():
            yield testing_client  # this is where the testing happens!
```

5.1.3 Sample Test

The test case code contains request URL, inputs, headers with user id and web token.The docstring explains test case importance.

The following figure shows sample test code

```python
def test_thread_generation_post_200_success(test_client):
    """
    GIVEN a Flask application configured for testing
    WHEN the '/api/v1/thread/create_thread/{ticket_id}' page is requested (POST)
    by student and ticket_id isnt mapped already
    THEN check that the response is 200
    """
    headers = {
        "Content-type": "application/json",
        "web_token": student_web_token,
        "user_id": student_user_id,
    }

    response = test_client.post(
        f"/api/{API_VERSION}/thread/create_thread/{ticket_id}",
        headers=headers,
    )
    response = response.get_json()
    assert response["status"] == 200
```

5.2 Test Description

5.2.1 Authorization Testing

Test: POST Request on Register Page

Description	GIVEN a Flask application configured for testing WHEN the '/api/v1/auth/register' page is requested (POST) with all correctly filled data fields for a new user THEN check that the response is 200 i.e. the account is created successfully
Page Being Tested	'/api/v1/auth/register'
Inputs	f"/api/{API_VERSION}/auth/register", json={ "first_name": "tej", "last_name": "v1", "email": random_email, "password": "1234", "retype_password": "1234",

	"role": "student", "discourse_id": random_id, "discourse_api_key": "1234",
Expected Output	Status code: 200
Actual Output	Status code: 200
Results	Passed

Test: POST Request on Register Page

Description	GIVEN a Flask application configured for testing WHEN the '/api/v1/auth/register' page is requested (POST) with existing discourse id THEN check that the response is 409 i.e. Discourse id already exists
Page Being Tested	'/api/v1/auth/register'
Inputs	f"/api/{API_VERSION}/auth/register", json={ "first_name": "tej", "last_name": "v1", "email": random_email, "password": "1234", "retype_password": "1234", "role": "student", "discourse_id": random_id, "discourse_api_key": "1234",
Expected Output	Status code: 409 # discourse id already exists
Actual Output	Status code: 409
Results	Passed

Test: POST Request on Register Page

Description	GIVEN a Flask application configured for testing WHEN the '/api/v1/auth/register' page is requested (POST) with unmatched API-key with discourse id THEN check that

	the response is 401 i.e. Discourse id does not matches with the API-key provided, incorrect API-key
Page Being Tested	'/api/v1/auth/register'
Inputs	f"/api/{API_VERSION}/auth/register", json={ "first_name": "tej", "last_name": "v1", "email": random_email, "password": "1234", "retype_password": "1234", "role": "student", "discourse_id": random_id, "discourse_api_key": "1234",
Expected Output	401 # discourse API Key invalid
Actual Output	Status code: 401
Results	Passed

Test: POST Request on Thread Page

Description	GIVEN a Flask application configured for testing WHEN the '/api/v1/thread/create_thread/{ticket_id}' page is requested (POST) by student and ticket_id is not mapped already THEN check that the response is 200
Page Being Tested	'/api/v1/thread/create_thread/{ticket_id}'
Inputs	f"/api/{API_VERSION}/thread/create_thread/{ticket_id}",headers=headers, test_client
Expected Output	Status code: 200
Actual Output	Status code: 200
Results	Passed

Test: POST Request on Thread Page

Description	GIVEN a Flask application configured for testing WHEN the '/api/v1/thread/create_thread/{ticket_id}' page is requested (POST) by student and ticket_id already exists THEN check that the response is 403
Page Being Tested	'/api/v1/thread/create_thread/{ticket_id}'
Inputs	f"/api/{API_VERSION}/thread/create_thread /{ticket_id}",headers=headers, test_client
Expected Output	Status code: 403
Actual Output	Status code: 403
Results	Passed

Test: PUT Request on Thread Page

Description	GIVEN a Flask application configured for testing WHEN the '/api/v1/thread/close_thread/{thread_id}' page is requested (PUT) by student and thread_id is valid THEN check that the response is 200
Page Being Tested	'/api/v1/thread/close_thread/{ticket_id}'
Inputs	f"/api/{API_VERSION}/thread/close_thread/{ ticket_id}",headers=headers, test_client
Expected Output	Status code: 200
Actual Output	Status code: 200
Results	Passed

Test: PUT Request on Thread Page

Description	GIVEN a Flask application configured for testing WHEN the '/api/v1/thread/close_thread/{thread_id}' page is requested (PUT) by student and thread_id is invalid THEN check that the response is 404
Page Being Tested	'/api/v1/thread/close_thread/{ticket_id}'
Inputs	f"/api/{API_VERSION}/thread/close_thread/{ticket_id}",headers=headers, test_client
Expected Output	Status code: 404
Actual Output	Status code: 404
Results	Passed

Test: PUT Request on Thread Page

Description	GIVEN a Flask application configured for testing WHEN the '/api/v1/thread/reopen_thread/{thread_id}' page is requested (PUT) by student and thread_id is closed already THEN check that the response is 200
Page Being Tested	'/api/v1/thread/reopen_thread/{ticket_id}'
Inputs	f"/api/{API_VERSION}/thread/reopen_thread/{ticket_id}",headers=headers, test_client
Expected Output	Status code: 200
Actual Output	Status code: 200
Results	Passed

Test: PUT Request on Thread Page

Description	GIVEN a Flask application configured for testing WHEN the '/api/v1/thread/reopen_thread/{thread_id}' page is requested (PUT) by student and thread_id is invalid or not closed THEN check that the response is 404
Page Being Tested	'/api/v1/thread/reopen_thread/{ticket_id}'
Inputs	f"/api/{API_VERSION}/thread/reopen_thread/{ticket_id}",headers=headers, test_client
Expected Output	Status code: 404
Actual Output	Status code: 404
Results	Passed

5.3 Final Test Summary

Screenshots of each test case code is attached below

Sample Screenshot - 1

```python
def test_register_page_with_fixture_post_200_success(test_client):
    """
    GIVEN a Flask application configured for testing
    WHEN the '/api/v1/auth/register' page is requested (POST)
    with all correctly filled data fields for a new user
    THEN check that the response is 200 i.e. the account is created successfully
    """

    random_email = f"tej{str(int(time.time()))}@gmail.com"
    random_id = f"{str(int(time.time()))}tej"

    response = test_client.post(
        f"/api/{API_VERSION}/auth/register",
        json={
            "first_name": "tej",
            "last_name": "v1",
            "email": random_email,
            "password": "1234",
            "retype_password": "1234",
            "role": "student",
            "discourse_id": random_id,
            "discourse_api_key": "1234",
        },
        headers=headers,
    )
    response = response.get_json()
    assert response["status"] == 200  # account created successfully
```

Sample Screenshot - 2

```python
def test_register_page_with_fixture_post_409_discourse_id_exists(test_client):
    """
    GIVEN a Flask application configured for testing
    WHEN the '/api/v1/auth/register' page is requested (POST)
    with existing discourse id
    THEN check that the response is 409 i.e. Discourse id already exists
    """

    random_email = f"tej{str(int(time.time()))}@gmail.com"

    response = test_client.post(
        f"/api/{API_VERSION}/auth/register",
        json={
            "first_name": "tej",
            "last_name": "v1",
            "email": random_email,
            "password": "1234",
            "retype_password": "1234",
            "role": "student",
            "discourse_id": "1210tej",
            "discourse_api_key": "1234",
        },
        headers=headers,
    )
    response = response.get_json()
    assert response["status"] == 409  # discourse id already exists
```

Sample Screenshot - 3

```python
def test_register_page_with_fixture_post_401_incorrect_APIKey(test_client):
    """
    GIVEN a Flask application configured for testing
    WHEN the '/api/v1/auth/register' page is requested (POST) with
    unmatched API-key with discourse id
    THEN check that the response is 401 i.e.
    Discourse id doesnot matches with the API-key provided, incorrect API-key
    """

    random_email = f"tej{str(int(time.time()))}@gmail.com"

    response = test_client.post(
        f"/api/{API_VERSION}/auth/register",
        json={
            "first_name": "tej",
            "last_name": "v1",
            "email": random_email,
            "password": "1234",
            "retype_password": "1234",
            "role": "student",
            "discourse_id": "nisanth",
            "discourse_api_key": "0000",
        },
        headers=headers,
    )
    response = response.get_json()
    assert response["status"] == 401  # discourse API Key invalid
```

Sample Screenshot - 4

```python
def test_thread_generation_post_200_success(test_client):
    """
    GIVEN a Flask application configured for testing
    WHEN the '/api/v1/thread/create_thread/{ticket_id}' page is requested (POST)
    by student and ticket_id isnt mapped already
    THEN check that the response is 200
    """
    headers = {
        "Content-type": "application/json",
        "web_token": student_web_token,
        "user_id": student_user_id,
    }

    response = test_client.post(
        f"/api/{API_VERSION}/thread/create_thread/{ticket_id}",
        headers=headers,
    )
    response = response.get_json()
    assert response["status"] == 200
```

Sample Screenshot -5

```python
def test_thread_closing_put_404_invalid_thread_id(test_client):
    """
    GIVEN a Flask application configured for testing
    WHEN the '/api/v1/thread/close_thread/{thread_id}' page is
    requested (PUT) by student and thread_id is invalid
    THEN check that the response is 404
    """
    headers = {
        "Content-type": "application/json",
        "web_token": student_web_token,
        "user_id": student_user_id,
    }

    response = test_client.put(
        f"/api/v1/thread/close_thread/{thread_id}",
        headers=headers,
    )
    response = response.get_json()
    assert response["status"] == 404
```

Sample Screenshot - 6

```python
def test_thread_reopening_put_404_invalid_thread_id(test_client):
    """
    GIVEN a Flask application configured for testing
    WHEN the '/api/v1/thread/reopen_thread/{thread_id}' page is
    requested (PUT) by student and thread_id is invalid or not closed
    THEN check that the response is 404
    """
    headers = {
        "Content-type": "application/json",
        "web_token": student_web_token,
        "user_id": student_user_id,
    }

    response = test_client.put(
        f"/api/v1/thread/reopen_thread/{thread_id}",
        headers=headers,
    )
    response = response.get_json()
    assert response["status"] == 404
```

Sample Screenshot - 7

```python
def test_thread_generation_post_200_success(test_client):
    """
    GIVEN a Flask application configured for testing
    WHEN the '/api/v1/thread/create_thread/{ticket_id}' page is requested (POST)
    by student and ticket_id isnt mapped already
    THEN check that the response is 200
    """
    headers = {
        "Content-type": "application/json",
        "web_token": student_web_token,
        "user_id": student_user_id,
    }

    response = test_client.post(
        f"/api/{API_VERSION}/thread/create_thread/{ticket_id}",
        headers=headers,
    )
    response = response.get_json()
    assert response["status"] == 200
```

Sample Screenshot - 8

```python
def test_thread_closing_put_404_invalid_thread_id(test_client):
    """
    GIVEN a Flask application configured for testing
    WHEN the '/api/v1/thread/close_thread/{thread_id}' page is
    requested (PUT) by student and thread_id is invalid
    THEN check that the response is 404
    """
    headers = {
        "Content-type": "application/json",
        "web_token": student_web_token,
        "user_id": student_user_id,
    }

    response = test_client.put(
        f"/api/v1/thread/close_thread/{thread_id}",
        headers=headers,
    )
    response = response.get_json()
    assert response["status"] == 404
```

Sample Screenshot - 9

```python
def test_thread_reopening_put_404_invalid_thread_id(test_client):
    """
    GIVEN a Flask application configured for testing
    WHEN the '/api/v1/thread/reopen_thread/{thread_id}' page is
    requested (PUT) by student and thread_id is invalid or not closed
    THEN check that the response is 404
    """
    headers = {
        "Content-type": "application/json",
        "web_token": student_web_token,
        "user_id": student_user_id,
    }

    response = test_client.put(
        f"/api/v1/thread/reopen_thread/{thread_id}",
        headers=headers,
    )
    response = response.get_json()
    assert response["status"] == 404
```

MILESTONE 6:

6. Implementation Details

6.1. Technologies Used

This project named “Discourse and webhooks integration to Online Support Ticket System (OSTS)” is built in Python and JavaScript. The wireframes for the project were built in **Canva**. The API doc is built in ‘Swagger’. The App testing is done with ‘PyTest’.Class diagram was made in **‘Figma’.**

Coding and GitHub operations were done using ‘Visual Studio Code’ and ‘Git’.

The backend server is built with Python and Flask. Common libraries used are summarized below.

Library/Framework/Language	Usage
Python	Core programming language for the project backend
Flask and its extensions	Micro web framework to create backend API server
SQLite	Backend Database
logging	Library to keep logs of data
smtp and email	Library to send email notifications
SQLAlchemy	Library to manage backend database transactions
httplib2 package	To Integrate webhooks
Json package	For sending the request to http

The frontend is built using Node and Vue in JavaScript. Common libraries used are summarized below.

Library/Framework/Language	Usage
JavaScript (Vue)	Core programming language for the project frontend and reactive components
Node	To create frontend server
VueLogger	Library to keep logs of data
FlashMessage	To display flash messages on screen for user
Router and Store	To keep important data in the frontend store and route different paths to components and views.
BootstrapVue	Library to style frontend web components

6.2. Instructions to Use App

The project sends notification mails whenever required and for security purposes, *'MailHog'* application is used in the backend. So, this application is currently hosted locally only. The *'backend.bat'* file, *'frontend.bat'* file and *'MailHog_windows_amd64*.exe' file are present in the *'code'* directory.

To start the software, there are three steps.

1. Start the 'MailHog_windows_amd64.exe' so that the emails sent during the usage of app, will be captured by 'MailHog' at http://127.0.0.1:8025/.

2. Then run the 'backend.bat' file. It starts the backend server which handles database manipulations and API functions. It runs at http://127.0.0.1:5000/

3. Then run the 'frontend.bat' file. It starts the frontend server which serves web pages for the frontend user.

Finally, visit the 'http://127.0.0.1:8080/home' page on the browser.

6.3 Issue Tracking and code review

We used VSCode as our IDE and git & gitHub as version control system. For each milestone, We had weekly meetings offline where we discussed challenges that we faced during the various phases of the project like requirement gathering, design and development, testing and deployment.

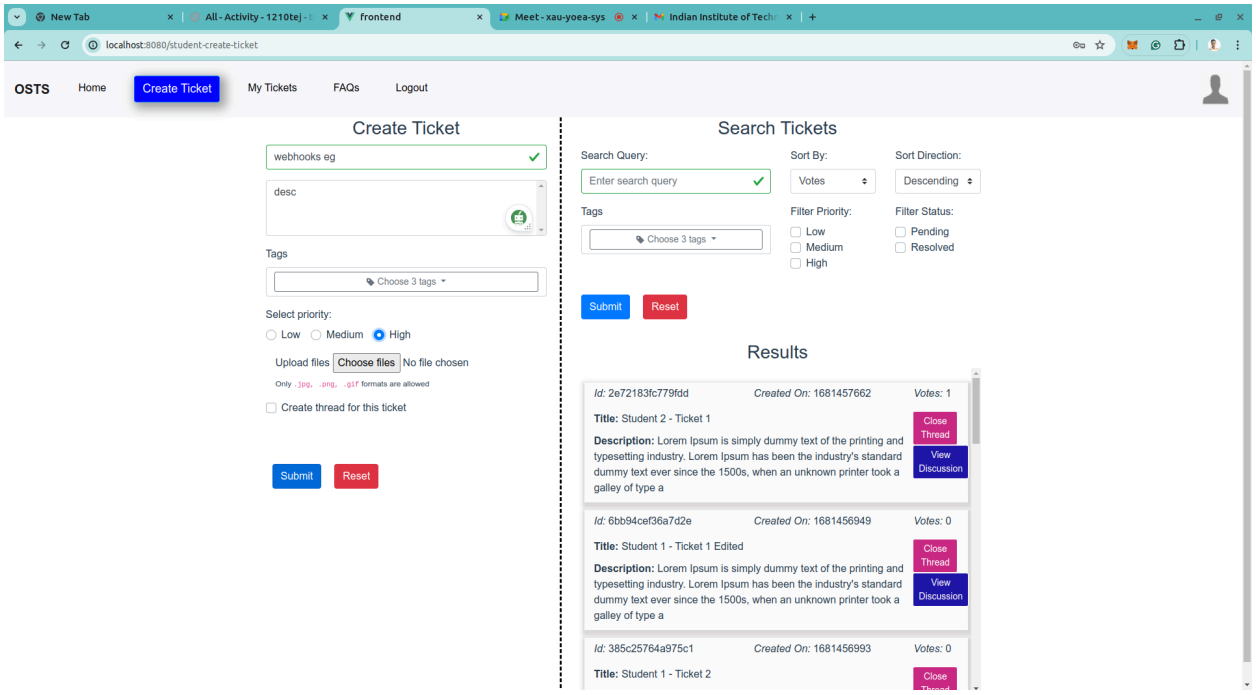
We used the jira website to track our progress sprint wise. We split the project into two parts: discourse integration and webhooks integration, where 3 people worked on discourse part and 2 people on webhooks integration. We faced issues while merging our two parts where we sat together and carefully debugged the issues.

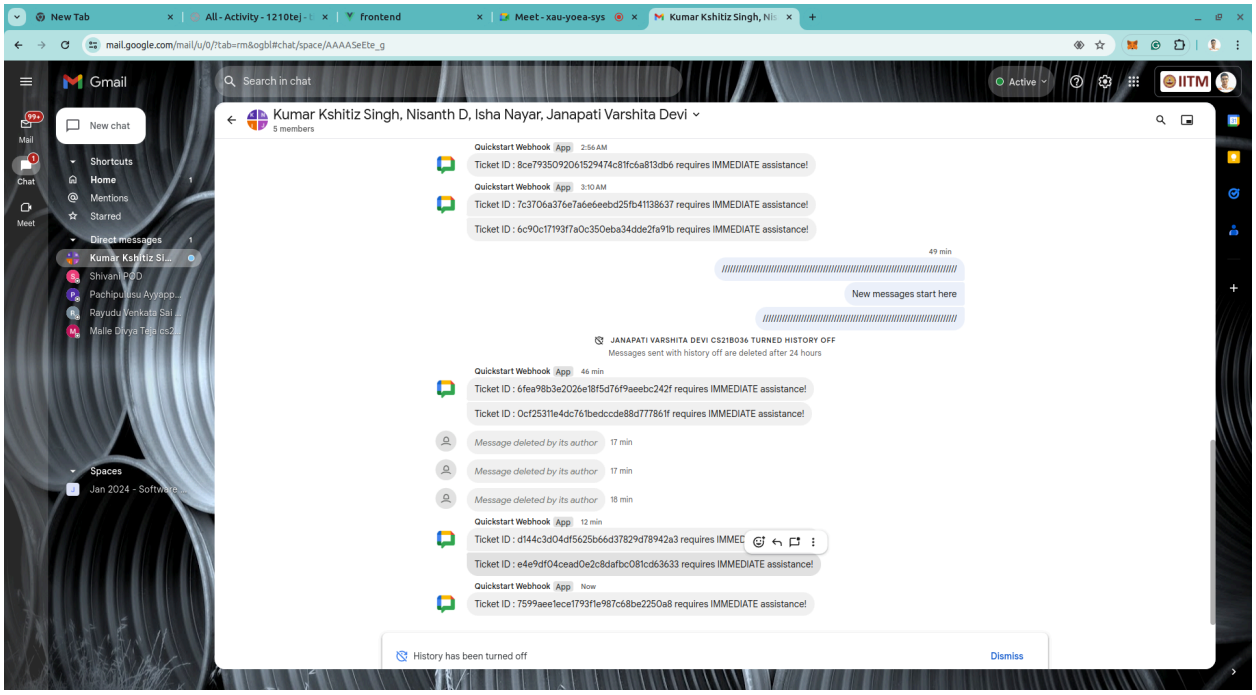
We faced some issues in integration of the frontend and backend part at the end, where we met offline in our department office and figured out the issues together. Meeting offline and whatsapp was our primary mode of communication.

6.4. Web App Screenshots

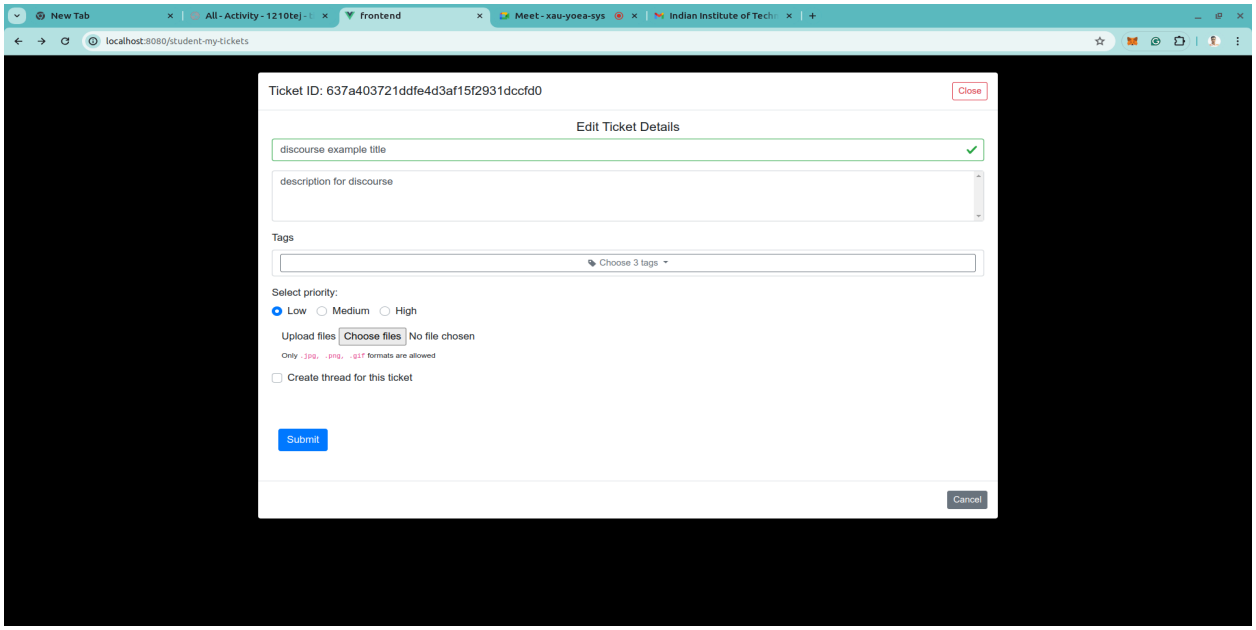
Here are the screenshots of Webpages and Gchat that represent the features that we have implemented.

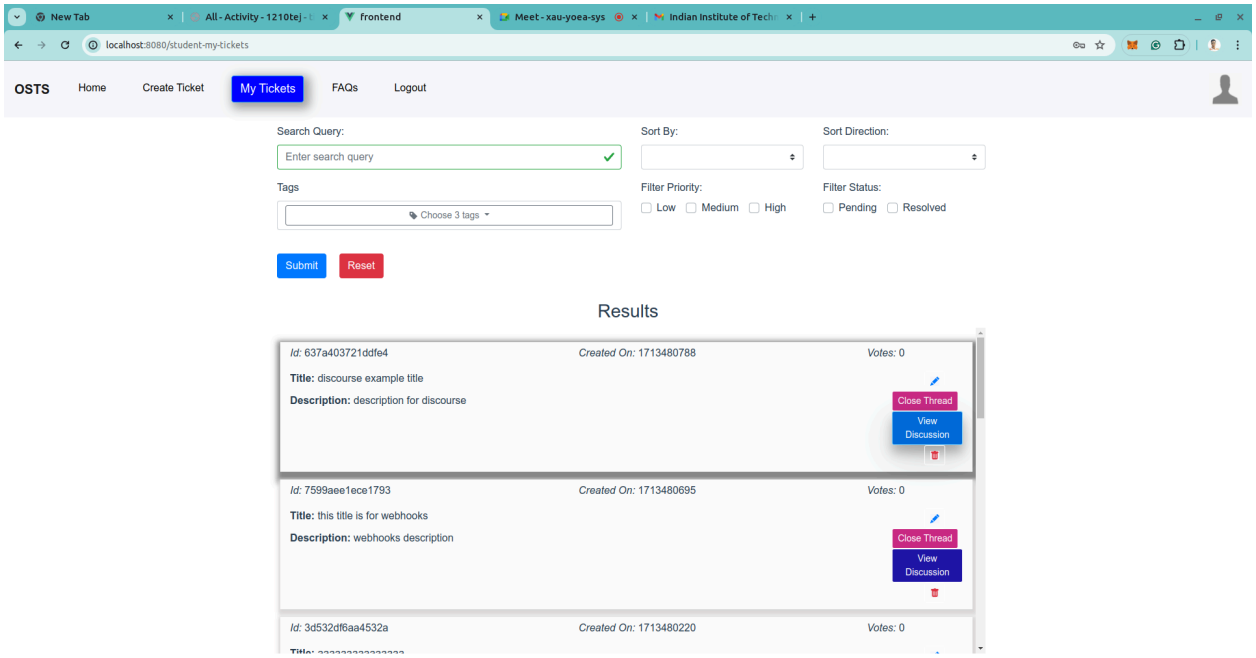
Webhook Integration:

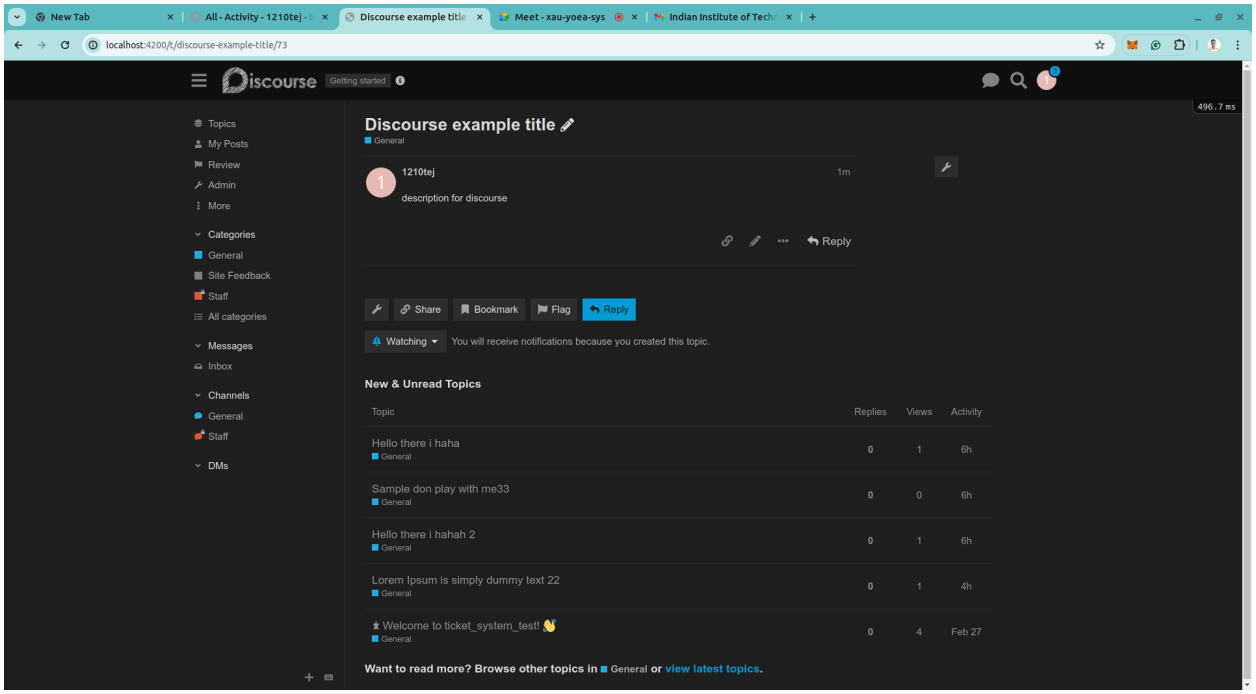



Gmail
Search in chat
Active
IITM
New chat
Mail
Chat
Meet
Shortcuts
Home
Mentions
Starred
Direct messages
Kumar Kshitiz Si...
Shivani POD
Pachipulusu Ayyapp...
Rayudu Venkata Sai ...
Malle Divya Teja cs2...
Spaces
Jan 2024 - Software ...
Kumar Kshitiz Singh, Nisanth D, Isha Nayar, Janapati Varshita Devi
5 members
Quickstart Webhook App 2:56 AM
Ticket ID : 8ce7935092061529474c81fc6a813db6 requires IMMEDIATE assistance!
Quickstart Webhook App 3:10 AM
Ticket ID : 7c3706a376e7a6e6eebd25fb41138637 requires IMMEDIATE assistance!
Ticket ID : 6c90c17193f7a0c350eba34dde2fa91b requires IMMEDIATE assistance!
49 min
New messages start here
JANAPATI VARSHITA DEVI CS21B036 TURNED HISTORY OFF
Messages sent with history off are deleted after 24 hours
Quickstart Webhook App 46 min
Ticket ID : 6fea98b3e2026e18f5d76f9aeebc242f requires IMMEDIATE assistance!
Ticket ID : 0cf25311e4dc761bedccde88d777861f requires IMMEDIATE assistance!
Message deleted by its author 17 min
Message deleted by its author 17 min
Message deleted by its author 18 min
Quickstart Webhook App 12 min
Ticket ID : e4e9df04cead0e2c8dafbc081cd63633 requires IMMEDIATE assistance!
Quickstart Webhook App Now
Ticket ID : 7599aee1ece1793f1e987c68be2250a8 requires IMMEDIATE assistance!
History has been turned off
Dismiss

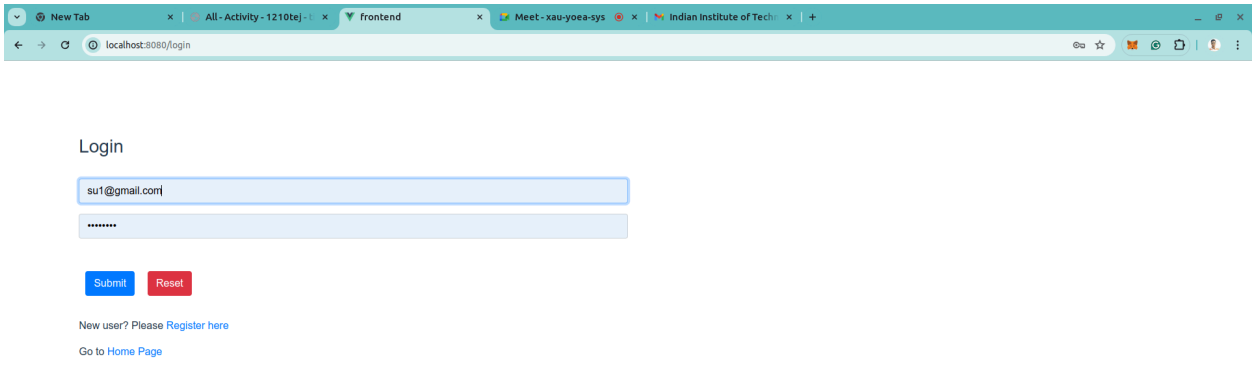
DISCOURSE INTEGRATION







SUPPORT STAFF LOGIN



SUPPORT STAFF HOME