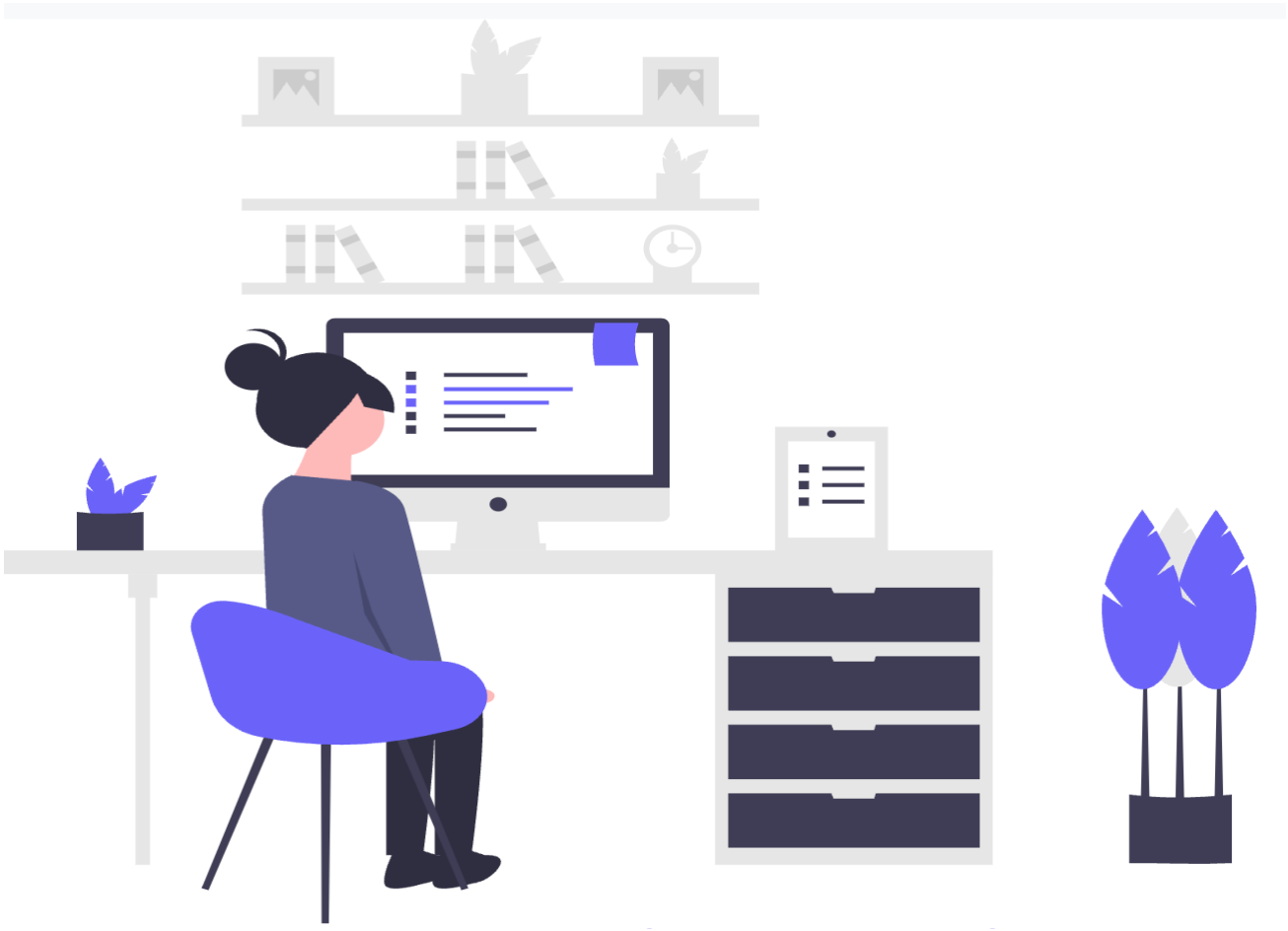
SUPPORTCENTRAL

GROUP PROJECT FOR SOFTWARE ENGINEERING



BY: TEAM FMM(GROUP NO.-2)

ABHISHEK KUMAR
KAUSTAV GOSWAMI
ANDIBOYINA MOURYA CHAKRADHAR NAGESH

Screencast Link

Table of Contents

1. CHAPTER-1: Problem Statement
2. CHAPTER-2: Milestone - 1
    1. TASKS
    2. 2.1 Identifying Primary, Secondary and Tertiary Users
    3. 2.2 Writing User Stories
        1. Students
        2. Support Staff
        3. Administrator
3. CHAPTER-3: Milestone-2
    1. TASKS
    2. 2.1 The Story
    3. 2.2 Storyboard
    4. 2.3 Wireframes Designed
4. CHAPTER-4: Milestone-3
    1. TASKS
    2. 4.1 Project Schedule
    3. 4.2 Gannt Chart for Project Schedule
    4. 4.3 Project Scheduling Tool
    5. 4.4 Components
        1. Login Management System
        2. Ticket Management System
        3. User Management System
        4. Tag Management System
        5. FAQ System
    6. 4.5 CLASS DIAGRAM
5. CHAPTER-5: Milestone-4
    1. TASKS
    2. 5.1 Login Manager API
        1. Error Codes
    3. 5.2 Role Manager API
        1. Error Codes
    4. 5.3 Ticket Manager API
        1. Error Codes
    5. 5.4 Response API for Ticket Manager
        1. Error Codes

    6. 5.5 Tag Manager API
        1. Error Codes
6. CHAPTER-6: Milestone-5
    1. TASKS
    2. 6.1 Login Manager API test details
        1. Output
    3. 6.2 Role Manager API test details
        1. Output
    4. 6.3 Ticket Manager API test details
        1. Output
    5. 6.4 Response API for Ticket Manager test details
    6. 6.5 Tag Manager API test details
        1. Output
7. CHAPTER-7: Milestone-6
    1. 7.1 API Code Reviews
        1. Login Management System
        2. Ticket Management System
        3. User Management System
        4. Tag Management System
    2. 7.2 VueJS Frontend Code Reviews
8. CHAPTER-8: Implementation Details
    1. 8.1 Technologies and Tools Used
    2. 8.2 Instructions to run application
        1. Frontend
        2. Backend (API)
    3. 8.3 Hosting Details
        1. Backend API Hosting
        2. Frontend Application Hosting
9. CHAPTER-9 Wireframes vs Actual Page

CHAPTER-1: Problem Statement

Online support ticket system for the IITM BSc degree program

The support team at the IITM BSc degree program often get overwhelmed with emails from students regarding queries and concerns. Your task is to create an online support ticketing system for the IITM BSc 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 update. 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.

Apart from these standard requirements, you can also think of other features which can add value to users.

CHAPTER-2: Milestone - 1

TASKS

Tasks for Milestone - 1.

1. Identifying Primary, Secondary and Tertiary Users
2. User Stories for the requirements based on **SMART** guidelines.

2.1 Identifying Primary, Secondary and Tertiary Users

- **Primary Users**
  1. Students
  2. Support Staff
  3. Administrators
- **Secondary Users**
  1. Developers and Technical Teams
  2. Data Analysts
- **Tertiary Users**
  1. Third Party apps

2.2 Writing User Stories

Students

1. As a student, I want to be able to create/login to my account, so that I can use this application to resolve my queries. [Implemented]
2. As a student, I want to be able to create ticket for any issue I'm having, so I can get help promptly . [Implemented]
   - As a student, I want to be able to create ticket with appropriate title and description for my issue, so that I can get help promptly.
   - As a student, I want to be able to create ticket with relevant tags for my issue, so that I can get help promptly.
3. As a student, I want to be able to search all the queries so that I can see similar existing queries.
   - As a student, I want to be able to search all the queries by mentioning the keywords, so that I can see similar existing queries. [Implemented]
   - As a student, I want to be able to search all the queries by selecting the tags, so that I can see similar existing queries. [Implemented]

    - As a student, I want to be able to search all the queries within a time-frame, so that I can see similar existing queries within that period.
4. As a student, I want to be able to know the status of my query through an email or notification, so that I can know when my query got resolved. [Implemented]
5. As a student, I want to be able to easily access the application from any device, so that I can get help whenever it's possible
6. As a student, I want to mark any of the responses to my query as an answer so that I can pin the answer at the top of my query. [Implemented]

Support Staff

1. As a member of support staff, I want to be able to create/login to my account, so that I can use this application to resolve the queries of the students. [Implemented]
2. As a member of support staff, I want to get a notification whenever someone creates a ticket of relevant tag/topic, so that I can try to quickly resolve their ticket as soon as possible. [Implemented]
3. As a support staff member, I want to be able to add important and resolved queries to FAQ's, so that students can get help without raising the query. [Implemented]
4. As a support staff member , I want to be able to mark an existing query as DUPLICATE if there is an already existing similar query so that I can reduce the clutter and redirect the visiting users to the original query. [Implemented]
5. As a support staff member, I want to mark a response to a query as an answer so that I pin the answer at the top of that query and let others know that query got resolved. [Implemented]
6. As a support staff member, I want to forward an incorrectly tagged ticket to the administrator, so that it can modified and resolved by the relevant support staff.

Administrator

1. As a member of support staff, I want to be able to create/login to my account, so that I can manage the working of this application. [Implemented]
2. As an admin, I want to be able to assign the type of query in relevant categories so that the queries are organized.
3. As an admin, I want to be create tags so that various tickets can get organized and relevant staff can get notified when a ticket is created. [Implemented]
4. As an admin, I want to be able to manage different roles for different users so that I can control the accessibility of various users.
5. As an admin, I want to mark a response to a query as an answer so that I can pin the answer at the top of that query and let others know that query got resolved. [Implemented]
6. As an admin, I want to be able to add important and resolved queries to FAQ's so that students can get help without raising the query. [Implemented]

7. As an admin, I want to be able delete any ticket/responses so that I can make this application safe and formal to work. [Implemented]

CHAPTER-3: Milestone-2

TASKS

1. Create a storyboard for the application - it can be a ppt or even a video. Embed the ppt/video in the pdf submission of this week.
2. Create low-fidelity wireframes for the identified user stories.
3. Apply usability design guidelines and heuristics discussed in lectures to come up with the wireframes.

2.1 The Story

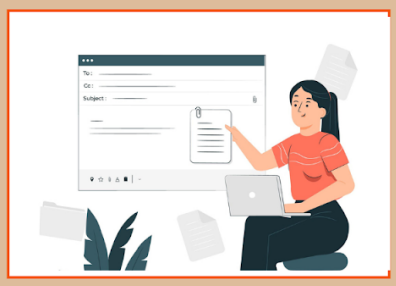
1. Vidya is preparing for Quiz-1 and she is stuck on a MLT topic and unable to understand it. She also understands that this topic is important for the Quiz-1.
2. She mails the doubts to online support team for clarification of the topic
3. It's been 5 hours but no reply from the support team. Vidya is now super stressed for the Quiz-1 also affecting other courses she has taken.
4. She now logins the Support Ticketing System of IITM and raised her queries about the topic.
5. The query has been assigned to the relevant support team for resolution
6. Within a very short time, Vidya has been notified that her query has been resolved. She totally understands the solution and likes the solution

2.2 Storyboard

STORYBOARD

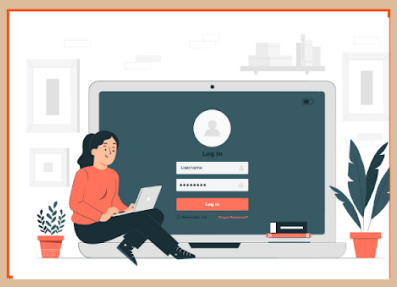
This storyboard contains a sequence of illustrations and includes descriptions. It illustrates that Online Support Ticketing System of IITM ensures immediate resolution of query as per compared to conventional resolution through mailing system

Vidya is preparing for Quiz-1 and she is stuck on a MLT topic and unable to understand it. She also understands that this topic is important for the Quiz-1.



She mails the doubts to online support team for clarification of the topic

Its been 5 hrs but no reply from support team. Vidya is now super stressed for the Quiz-1 also affecting other courses she has taken.



She now logins the Support Ticketing System of IITM and raised her queries about the topic.

The query has been assigned to the relevant support team for resolution

Within a very short time, Vidya has been notified that her query has been resolved. She totally understands the solution and likes the solution.

2.3 Wireframes Designed

1. **Student Perspective**
    1. Student Login Page
    2. Student Signup Page
    3. Student Dashboard Page
    4. Subject Page
    5. Create Ticket Page
    6. Ticket/Query View Page
2. **Support's Perspective
    1. Support Login Page (Same as Student)
    2. Support Signup Page
    3. Subject Page
    4. Query View for customized for Support
3. **Admin Perspective**
    1. Admin Login Page (Same as Student)
    2. Admin Dashboard
    3. Create Tag Page
    4. Role Edit/Delete Page
    5. Approve Staff Page
    6. Main Dashboard Page
    7. Subject Page
    8. Query View Page

CHAPTER-4: Milestone-3

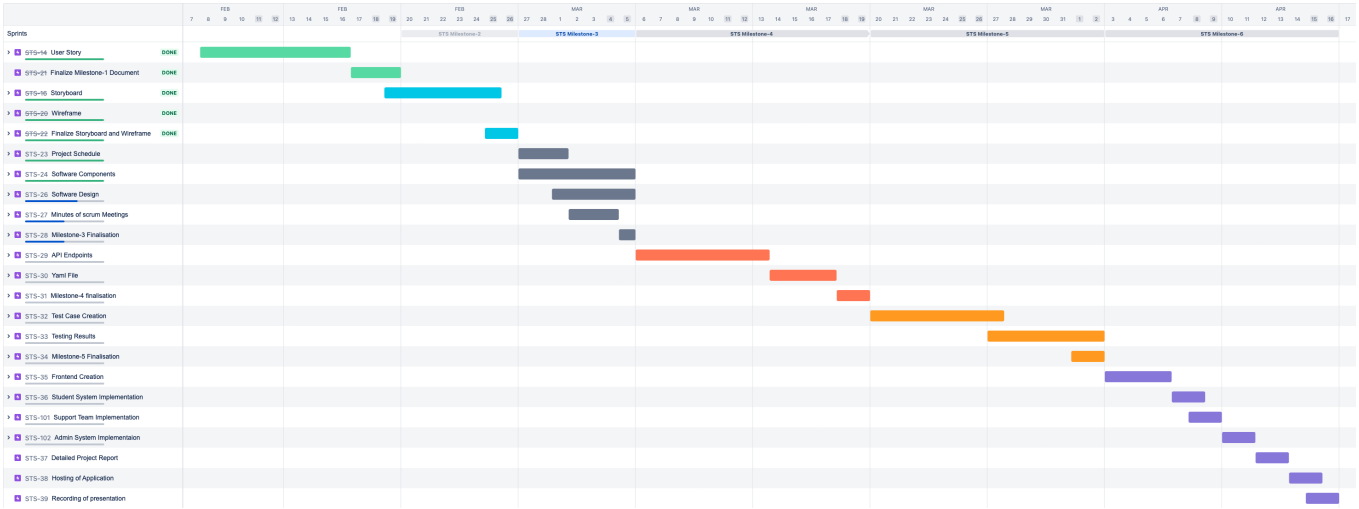
TASKS

1. **Project Schedule**: Come up with a schedule of your overall project - schedule for your sprints and iterations, timings of your scrum meetings etc. Trello board, Gantt chart - specifying your tasks and contributions.
2. **Project Scheduling Tools**: Which tools are you using? E.g. Pivotal Tracker, Jira etc.
3. **Design of Components**: Describe different components of your system.
4. **Software Design**: Basic class diagrams of your proposed system.
5. Details/Minutes of a few scrum meetings

4.1 Project Schedule

S.No.	Milestones	Sprints Schedule	Estimated Date of Completion	Actual Date of Completion
1.	Milestone-1	No Sprints	18.02.2023	18.02.2023
2.	Milestone-2	20.02.23-26.02.23	26.02.23	26.02.23
3.	Milestone-3	27.02.23-05.03.23	05.03.23	05.03.23
4.	Milestone-4	06.03.23-19.03.23	19.03.23	19.03.23
5.	Milestone-5	20.03.23-09.04.23	09.04.23	07.04.23
6.	Milestone-6	07.04.23-19.04.23	19.04.23	19.04.23

4.2 Gantt Chart for Project Schedule



4.3 Project Scheduling Tool

JIRA Software has been used as project scheduling tools and the following features have been utilized throughout the project:

- Preparation of Sprints in the Backlog Screen and scheduled based on the end date of Milestone submission
- Creation of Tasks and the relevant tasks were grouped under Epics
- Creation of Roadmaps based on submission dates of each Milestone
- Assigning Story Points to each task
- Using Kanban Board to track tasks in To-do, Progress and Completed Categories

4.4 Components

Login Management System

- Login manager manages the authentication and authorization process of users when they’re trying to access the software.
- It typically consists of a login screen where users are required to enter their credentials, such as username and password, to gain access to the system.
- Once the user is authenticated, the login manager grants them access to the support ticketing system and redirects them to the appropriate dashboard or homepage.
- The login manager also ensures that users are automatically logged out after a certain period of inactivity to prevent unauthorized access to the system
- The login manager is also responsible for registering students and support team.

Ticket Management System

- The ticket management component is responsible for managing student query from creation to resolution
- The ticket management component is essential in ensuring efficient and effective handling of student queries.
- The ticket management system enables the student to search, filter and create queries for the relevant topic
- When a student create a query in a particular category, the same is automatically assigned to the relevant support staff.
- The priority level of the ticket is based on the likes the query received from other students
- The ticket management system enables the support staff to view and resolve the ticket assigned to them
- It also enables the support staff to update the ticket status as FAQ.
- After resolution of the ticket, the system enables the student to mark the query as resolved to close the query

User Management System

- It is a vital component of an online support ticketing system that enables the management of different types of users with different levels of access and permissions. It provides a platform to manage and organize user accounts and their associated information.
- **Students**: The user management system allows students to create and manage their user accounts. Students can submit new support tickets and view the status of their existing tickets
- **Support team**: The user management system provides support team members with access to the software's ticket management .
  - The system assigns and manages tickets based on the support team member's role and permissions
  - Support team members can view and update ticket details.
- **Administrators**: The user management system enables administrators to manage the software's overall functionality, including user roles and permissions and ticket workflows.
  - Administrators can approve, modify, or delete support accounts and assign or revoke user permissions based on their roles and responsibilities
  - This also provides administrator access to Tag Management System.

Tag Management System

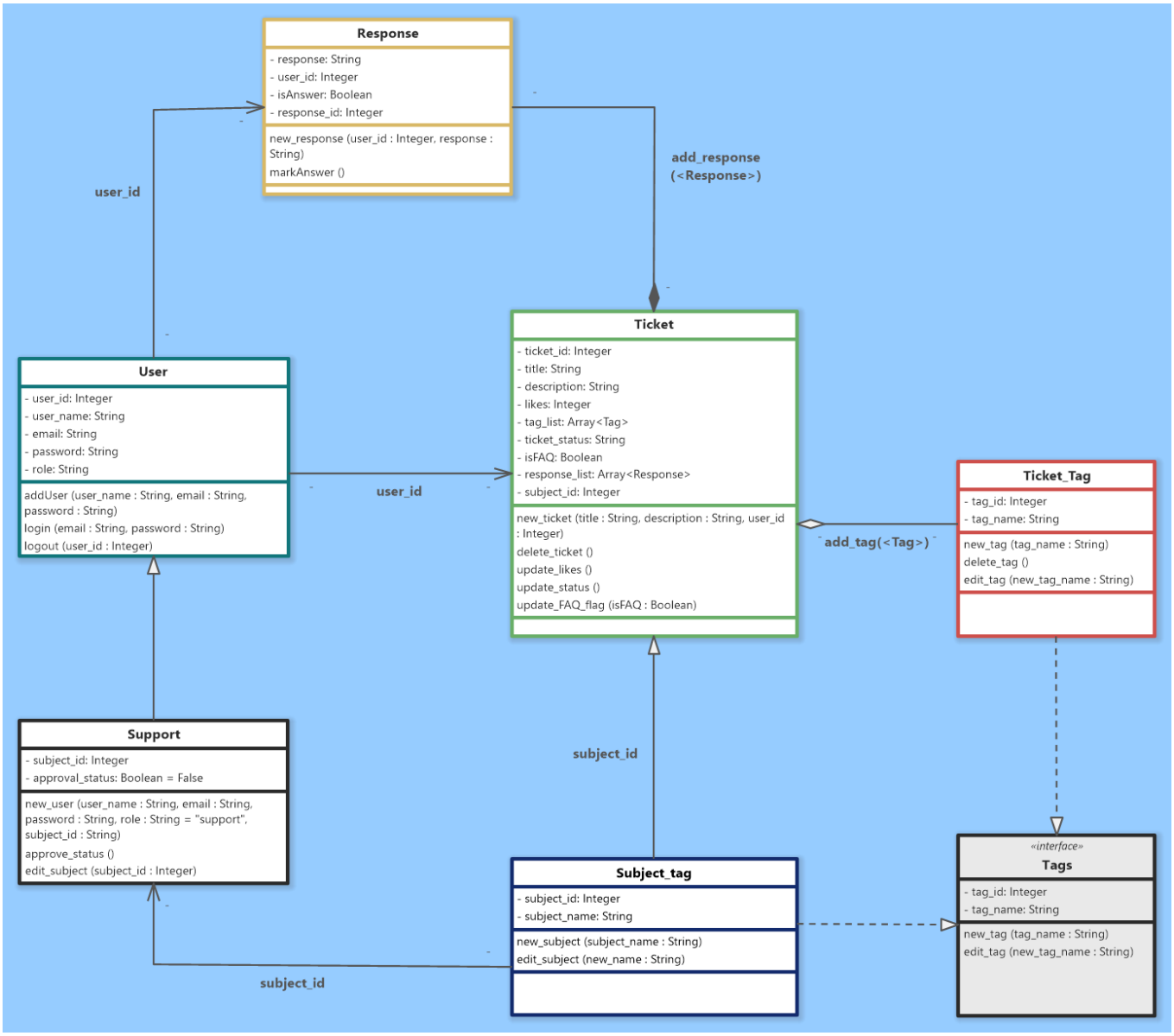
- The tag management component in a support ticketing software enables the administrator to create and manage tags that can be assigned to tickets.
- Tags are labels that are used to categorize and group similar tickets together

- It helps in automatic allocation of ticket to the relevant support staff.
- There are two types of tag which are categorized into subject tag and secondary tag.
- The subject tags relates to name of the subject/ course and this tag is automatically assigned during the creation of query.
- The secondary tag is assigned by the student during query submission.
- They allow support teams to easily identify and prioritize specific types of tickets
- It also allows the administrator to edit and delete the tags created.

FAQ System

- The FAQ (Frequently Asked Questions) system in an online support ticketing software allows administrators and support team members to update commonly asked questions and their corresponding answers as FAQ.
- Top 5 FAQ for all primary tags or course are populated on student's dashboard and whereas All FAQs of a particular course are shown when user clicks the particular course and FAQ tab in the course
- In addition to reducing the workload for the support team, the FAQ system also helps reduce student stress by providing student with immediate answers to their questions.

4.5 CLASS DIAGRAM



Details of Scrum Meetings can be viewed **here**

CHAPTER-5: Milestone-4

TASKS

1. Create and Describe API endpoints as per the problem statement.
2. Submit all the details of the API endpoints in a YAML file.

5.1 Login Manager API

1. `GET - /api/login/{email}`
    1. Operation to read user details.
    2. Fetches the details of the user with email as path parameter.
2. `POST - /api/register`
    1. Operation to Create a new User details. When `role=staff` `subject_id` is must but not required for `role=student`.
3. `PUT - /api/login/{email}`
    1. Operation to edit user details
    2. Usually used for changing password of the user.
4. `DELETE - /api/login/{delete}`
    1. Operation to delete user data from the database

Error Codes

Error Codes

Resource	Error Code	Message
LOGIN	USER001	Email is required and must be a non empty string.
LOGIN	USER002	Password is required and must be string with length>4.
LOGIN	USER003	Username is required and must be a non empty string.
LOGIN	USER004	Role is required and must be a non empty string.

5.2 Role Manager API

1. `GET - /api/role`
    1. Operation to filter staff details based on the status.
    2. Used to fetch details of staff members to display on the web page
2. `PUT - /api/role/{user_id}`
    1. Operation to change the status or subject-id of the staff.

3. `DELETE - /api/role/{user_id}`
    1. Operation to delete data of the Staff.

Error Codes

STAFF	STAFF001	Tag-id must be provided.
STAFF	STAFF002	Valid Subject-id is required.

5.3 Ticket Manager API

1. `GET - /api/subject/{subject_name}`
    1. Operation to get all the tickets that satisfies the parameter values.
    2. `FAQ`, `ResolvedStatus`,`limit` (to limit number of tickets in the data) and `search` keywords.
2. `POST - /api/subject/{tag_name}`
    1. Operation to Create a new ticket.
3. `PUT - /api/subject/ticket/{ticket_id}`
    1. Operation to Edit Ticket details
    2. The action variable can contain string `like`,`faq` or `notfaq`
4. `DELETE - /api/subject/ticket/{ticket_id}`
    1. Operation to Delete User details

Error Codes

TICKET	TICKET001	Either user_id or action type is missing.
TICKET	TICKET002	Ticket need to be resolved before marking as FAQ.
TICKET	TICKET003	Some data required for creating ticket is missing.
TICKET	TICKET004	Secondary tag entered is not available.
TICKET	TICKET005	A student cannot mark the ticket as FAQ.
TICKET	TICKET006	Invalid subject

5.4 Response API for Ticket Manager

1. `GET - /api/response/{ticket_id}`
    1. Operation to get all the responses replied for a ticket
2. `POST - /api/response/{ticket_id}`
    1. Operation to add a new response to the ticket.
3. `PUT - /api/reponse/{ticket_id}/{response_id}`

1. Operation to mark response as an answer to the ticket.

Error Codes

RESPONSE	RESPONE001	Invalid ticket id
RESPONSE	RESPONE002	Invalid response id
RESPONSE	RESPONE003	Either user id or response is missing in the body
RESPONSE	RESPONE004	'isAnswer' field is missing or invalid format
RESPONSE	RESPONE005	'ticket_status' field is missing or invalid format
RESPONSE	RESPONE006	Invalid value for ticket_status, it should be either 'resolved' or 'unresolved'

5.5 Tag Manager API

1. `GET - /api/tag/{tag_type}/{tag_id}`
    1. Operation to fetch details of a specific tag.
    2. The json object returned can contain two types of responses having either only subject data with `"sec_id":0 and "sec_name":null` or secondary tag detail with `"subject_id":0 and "subject_name":null,`
2. `GET - /api/tag/{tag_type}`
    1. Operation to Read all Tags
    2. Tag type can be `subject` or `secondary`
3. `POST - /api/tag/{tag_type}`
    1. Operation to create a new tag.
    2. The `tag_type` can be `subject` or `secondary`
4. `PUT - /api/tag/{tag_type}/{tag_id}`
    1. Operation to Edit Tag Name
5. `DELETE - /api/tag/{tag_type}/{tag_id}`
    1. Operation to Delete Tag details

Error Codes

TAG	TAG001	Tag_name should be non-empty string.
TAG	TAG002	Subject_name already exists.
TAG	TAG003	Secondary tag name already exists.
TAG	TAG004	Subject tag cannot be deleted.
TAG	TAG005	The url should contain secondary.

CHAPTER-6: Milestone-5

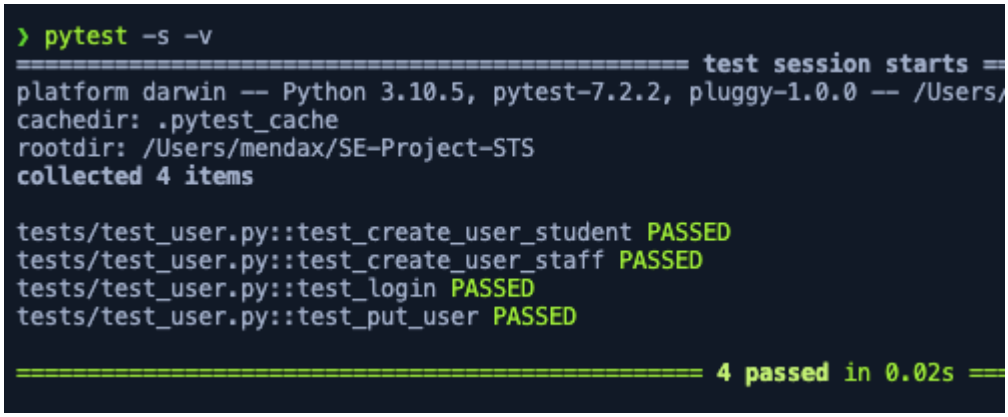
TASKS

1. Design and describe a few test cases.
2. The format should be:
    1. Page being tested
    2. Inputs
    3. Expected Output
    4. Actual Output
    5. Result - Success/Fail

6.1 Login Manager API test details

Test Scenario	Testing of User API		Tester	Abhishek		Date	March 29, 2023	
ID	**Test Case**	**Pre-condition**	**Test Steps**	**Test Data**	**Expected Output**	**Post-condition**	**Actual Output**	**Status**
User_01	Testing if student is succesfully registered	Server must be running	Create a dictionary of user details with keys as username, password, email and role	a. Valid Username format b. Valid Password length and format c. Valid email format d. Post request to User API class	a. response Status code should be 201 b. username of response created should match with entered data	Delete the user data	As expected	Pass
User_02	Testing to successfully register staff credentials	a. Server must be running b. A subject tag should already be created with corresponding subject_id	Create a dictionary of user details with keys as username, password, email, role and subject_id	a. Valid Username string format b. Valid Password length and string format c. Valid email in string format d. role should be mandatorily specified as staff e. valid subject_id of created subject tag d. Post request to User API class	a. response status code should be 201 b. username of response created should match with entered data c.email of response created should match with entered data	Delete the staff data	As expected	Pass
User_03	Testing for Login	a. Server must be running b. A valid email already registered	Entering email in the router url	a. GET Request to USER API Class with email	a. response status code should be 200	No post-conditions	As expected	Pass
User_04	Test for editing user password	a. Server must be running b. A valid email already registered	a. creating dictionary of password and role b. Entering the email in parameter of url	a. A valid password b. Valid role of either student or staff c. PUT request of User API class	a. response status code should be 202 b. new response password data should match with the entered data in dictionary	No post-conditions	As expected	Pass

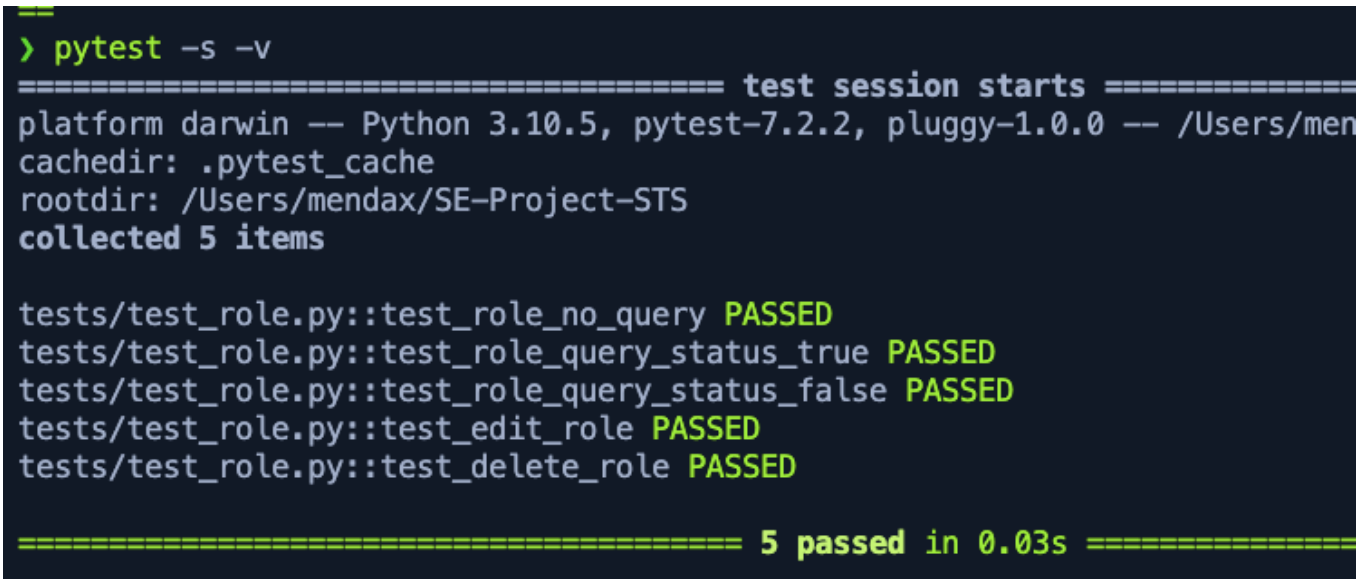
Output



6.2 Role Manager API test details

Test Scenario	Testing of Roles API		Tester	Kaustav Goswami		Date	March 30, 2023	
ID	**Test Case**	**Pre-condition**	**Test Steps**	**Test Data**	**Expected Output**	**Post-condition**	**Actual Output**	**Status**
Role_01	Testing to retrieve data about all the staff	a. Server must be running	GET request to Role API	No test data is required	a. Response status should be 200 b. Valid Staff data or empty list if no matching data is found	No post conditions	As expected	Pass
Role_02	Testing to retrieve data about all the approved staff	a. Server must be running	GET request to Role API with **status=1** in the query parameter	Correct value for status query parameter	a. Response status should be 200 b. Valid approved Staff data or empty list if no matching data is found	No post conditions	As expected	Pass
Role_03	Testing to retrieve data about all the pending approval of staff	a. Server must be running	GET request to Role API with **status=0** in the query parameter	Correct value for status query parameter	a. Response status should be 200 b. Valid pending approval Staff data or empty list if no matching data is found	No post conditions	As expected	Pass
Role_04	Testing to edit already existing staff data	a. Server must be running	a. Create dummy staff data dictionary with keys as username, password, email, **role='staff' and subject_id** b. Make a POST request to the Role API with the above dummy data c. Enter the user_id generated in the url parameter of PUT method of Role API and in the body pass **status=True**	a. Valid Staff data b. Valid update data for PUT request	a. 201 Response status for new staff data created b. 202 Response status for successful edit c. Response Body of PUT method should have **approved=True**	Delete the dummy data that is created	As expected	Pass
Role_05	Testing to delete already existing staff data	a. Server must be running	a. Create dummy staff data dictionary with keys as username, password, email, **role='staff' and subject_id** b. Make a POST request to the Role API with the above dummy data c. Enter the user_id generated in the url parameter of DELETE method of Role API	Valid Staff data	a. 201 Response status for new staff data created b. 200 Response status for successful delete	No post conditions	As expected	Pass

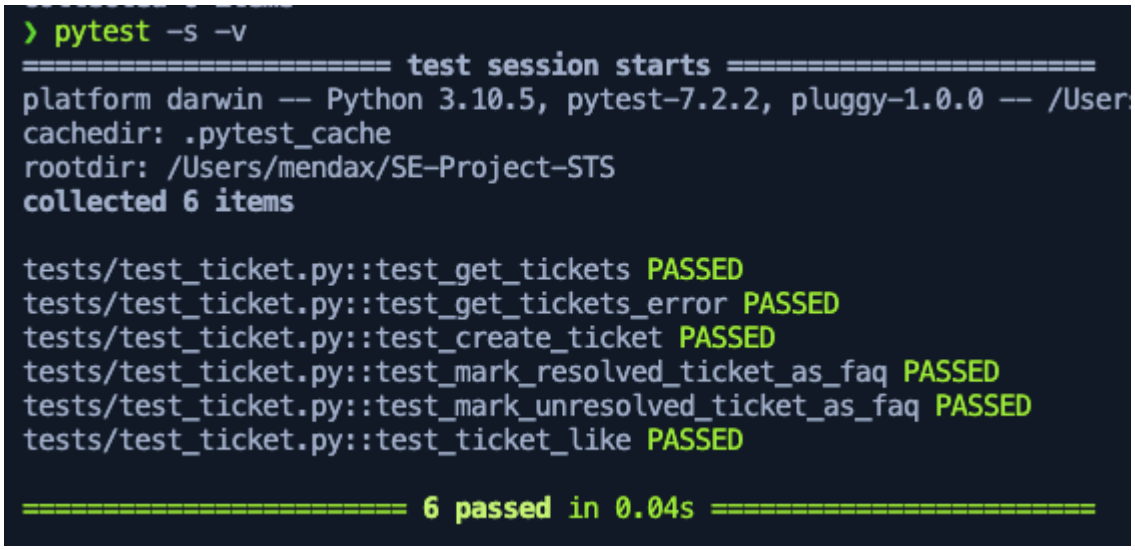
Output



6.3 Ticket Manager API test details

Test Scenario	Testing of Tag Manager API		Tester	Kaustav Goswami		Date	March 30, 2023	
ID	**Test Case**	**Pre-condition**	**Test Steps**	**Test Data**	**Expected Output**	**Post-condition**	**Actual Output**	**Status**
Tag_01	Testing to retrieve data about Subject tag	a. Server must be running b. A Subject tag must be created	GET request to Tag API with **tag_type=subject** in the url parameter	Valid tag_type	a. Response status should be 200 b. Valid Subject tag data or empty list if no matching data is found	No post condition	As expected	Pass
Tag_02	Testing to retrieve data about Secondary tag	a. Server must be running b. A Secondary tag must be created	GET request to Tag API with **tag_type=secondary** in the url parameter	Valid tag_type	a. Response status should be 200 b. Valid Secondary tag data or empty list if no matching data is found	No post condition	As expected	Pass
Tag_03	Testing to create a Subject Tag	a. Server must be running	a. Create dictionary with key as **tag_name** b. Make a POST request to Tag API with the above data and **tag_type=subject** in the url parameter	Unique subject tag name	a. 201 Response status for new tag data created b. The value in **subject_name** in the response body should match with the entered data	Delete the dummy data that is created	As expected	Pass
Tag_04	Testing to create a Secondary Tag	a. Server must be running	a. Create dictionary with key as **tag_name** b. Make a POST request to Tag API with the above data and **tag_type=secondary**in the url parameter	Unique secondary tag name	a. 201 Response status for new tag data created b. The value in **sec_name** in the response body should match with the entered data	Delete the dummy data that is created	As expected	Pass
Tag_05	Testing to edit a Subject Tag	a. Server must be running	a. Create dictionary with key as **tag_name** b. Make a POST request to Tag API with the above data and **tag_type=subject** in the url parameter c. Make a PUT request to Tag API with a different value in the **tag_name**	Unique subject tag name	a. 201 Response status for new tag data created b. The value in **subject_name** in the response body should match with the entered data c. The updated value of **subject_name** in the response body should match with the updated value given	Delete the dummy data that is created	As expected	Pass
Tag_06	Testing to edit a Secondary Tag	a. Server must be running	a. Create dictionary with key as **tag_name** b. Make a POST request to Tag API with the above data and **tag_type=secondary**in the url parameter c. Make a PUT request to Tag API with a different value in the **tag_name**	Unique secondary tag name	a. 201 Response status for new tag data created b. The value in **sec_name** in the response body should match with the entered data c. The updated value of **sec_name** in the response body should match with the updated value given	Delete the dummy data that is created	As expected	Pass

Output



6.4 Response API for Ticket Manager test details

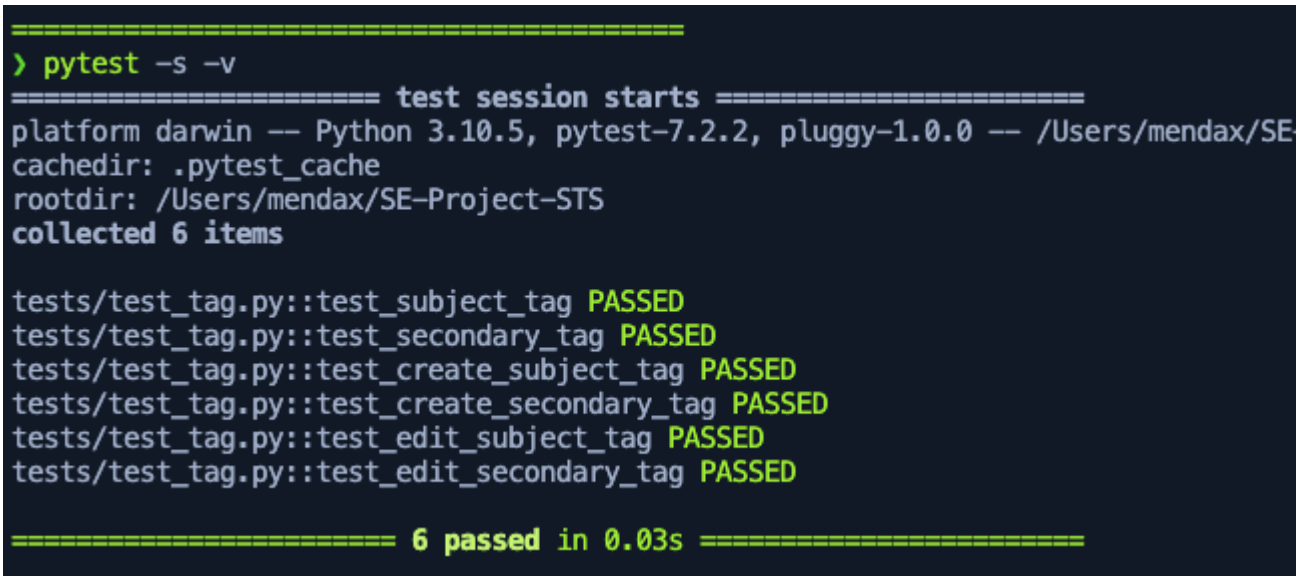
Test Scenario	Testing of Response API		Tester	ANDIBOYINA MOURYA CHAKRADHAR NAGESH		Date	April 2, 2023	
ID	Test Case	Pre-condition	Test Steps	Test Data	Expected Output	Post-condition	Actual Output	Status
Response_01	Testing to create a response	a. Server must be running b. Existing user_id c. Existing ticket_id	a. Create a dictionary of response details with keys as user_id and response b. POST request to Response API with ticket_id in POST url	a. Valid user_id b. Valid response format c. Valid existing ticket_id d. Post request to Response API class	a. response Status code should be 201 b. response should be created within entered ticket_id	Deleting the reponse	As expected	Pass
Response_02	Testing to get response	a. Server must be running b. Existing ticket_id c. Some responses to the given ticket_id	GET request to Response API	a. Valid ticket_id	a. response status code should be 200 b. The length of responses for the ticket_id should match as expected	No post conditions	As expected	Pass
Response_03	Testing to get response error	a. Server must be running b. Invalid ticket_id	GET request to Response API	a. Numerical invalid ticket_id	a. response status code should be 404 b. error code correspond to RESPONSE001	No post conditions	As expected	Pass
Response_04	Testing for different response creation error	a. Server must be running b. A invalid ticket_id c. Existing response to a ticket	a. Create a dictionary of response details with keys as user_id and response b. POST request to Response API with invalid ticket_id in url c. Creating dictionary with missing details	a. Numerical invalid ticket_id	a. response status code should be 404 for invalid ticket_id b. reponse status code for other requests should be 400 c. error codes as defined	No post-conditions	As expected	Pass
Response_05	Testing to marking response as Answer and resolving	a. Server must be running b. Existing ticket_id c. Existing response_id	a. Create a dictionary of payload with keys as isAnswer and ticket_status b. existing ticket_id and response _id in the PUT request to Response API class	a. isAnswer field should be boolean b. ticket_status should either be resolved or unresolved c. Valid ticket_id d. Valid response_id	response status code should be 200	No post-conditions	As expected	Pass
Response_06	Testing for different response editing error	a. Server must be running b. A invalid ticket_id c. Existing response to a ticket	a. Create a dictionary of payload with keys as isAnswer and ticket_status b. PUT request to Response API with invalid ticket_id or response_id in url c. Creating dictionary with missing details	a. Numerical invalid ticket_id	a. response status code should be 404 for invalid ticket_id or response_id b. reponse status code for other requests should be 400 c. error codes as defined	No post-conditions	As expected	Pass

```
❯ pytest -s -v
========================= test session starts =========================
platform darwin -- Python 3.10.5, pytest-7.2.2, pluggy-1.0.0 -- /Users/mendax/SE-Project-STS/.env/bin
cachedir: .pytest_cache
rootdir: /Users/mendax/SE-Project-STS
collected 10 items

tests/test_response.py::test_get_response PASSED
tests/test_response.py::test_get_response_error PASSED
tests/test_response.py::test_post_response PASSED
tests/test_response.py::test_post_response_error PASSED
tests/test_response.py::test_put_response {'isAnswer': True, 'ticket_status': 'resolved'}
PASSED
tests/test_response.py::test_put_response_error_1 {'isAnswer': False, 'ticket_status': 'unresolved'}
PASSED
tests/test_response.py::test_put_response_error_2 PASSED
tests/test_response.py::test_put_response_error_3 {'ticket_status': 'resolved'}
PASSED
tests/test_response.py::test_put_response_error_4 {'isAnswer': False}
PASSED
tests/test_response.py::test_put_response_error_5 {'isAnswer': False, 'ticket_status': 'hello'}
PASSED

========================= 10 passed in 0.04s =========================
```

6.5 Tag Manager API test details

Test Scenario	Testing of Tag Manager API		Tester	Kaustav Goswami		Date	March 30, 2023	
ID	**Test Case**	**Pre-condition**	**Test Steps**	**Test Data**	**Expected Output**	**Post-condition**	**Actual Output**	**Status**
Tag_01	Testing to retrieve data about Subject tag	a. Server must be running b. A Subject tag must be created	GET request to Tag API with **tag_type=subject** in the url parameter	Valid tag_type	a. Response status should be 200 b. Valid Subject tag data or empty list if no matching data is found	No post condition	As expected	Pass
Tag_02	Testing to retrieve data about Secondary tag	a. Server must be running b. A Secondary tag must be created	GET request to Tag API with **tag_type=secondary** in the url parameter	Valid tag_type	a. Response status should be 200 b. Valid Secondary tag data or empty list if no matching data is found	No post condition	As expected	Pass
Tag_03	Testing to create a Subject Tag	a. Server must be running	a. Create dictionary with key as **tag_name** b. Make a POST request to Tag API with the above data and **tag_type=subject** in the url parameter	Unique subject tag name	a. 201 Response status for new tag data created b. The value in **subject_name** in the response body should match with the entered data	Delete the dummy data that is created	As expected	Pass
Tag_04	Testing to create a Secondary Tag	a. Server must be running	a. Create dictionary with key as **tag_name** b. Make a POST request to Tag API with the above data and **tag_type=secondary**in the url parameter	Unique secondary tag name	a. 201 Response status for new tag data created b. The value in **sec_name** in the response body should match with the entered data	Delete the dummy data that is created	As expected	Pass
Tag_05	Testing to edit a Subject Tag	a. Server must be running	a. Create dictionary with key as **tag_name** b. Make a POST request to Tag API with the above data and **tag_type=subject** in the url parameter c. Make a PUT request to Tag API with a different value in the **tag_name**	Unique subject tag name	a. 201 Response status for new tag data created b. The value in **subject_name** in the response body should match with the entered data c. The updated value of **subject_name** in the response body should match with the updated value given	Delete the dummy data that is created	As expected	Pass
Tag_06	Testing to edit a Secondary Tag	a. Server must be running	a. Create dictionary with key as **tag_name** b. Make a POST request to Tag API with the above data and **tag_type=secondary**in the url parameter c. Make a PUT request to Tag API with a different value in the **tag_name**	Unique secondary tag name	a. 201 Response status for new tag data created b. The value in **sec_name** in the response body should match with the entered data c. The updated value of **sec_name** in the response body should match with the updated value given	Delete the dummy data that is created	As expected	Pass

Output



CHAPTER-7: Milestone-6

7.1 API Code Reviews

Login Management System

- **Code Review**: This system manages the authentication and authorization process of users when they're trying to access the software. It also takes care of registering new users and granting them access.
- **Issues Reported**:
  - Bug in password checking when new user registers [Resolved]
  - JWT token authorization is not implemented [Resolved]
  - User_id of the User data is not added in the response body. [Resolved]
  - Passwords saved in Database is not encrypted. [Resolved]

Ticket Management System

- **Code Review**: This component is responsible for managing student queries from creation to resolution.
- **Issues Reported**:
  - No error is thrown for wrong subject name [Resolved]
  - Anyone can mark a ticket as FAQ and unresolved ticket can be marked as FAQ [Resolved]
  - Some fields are missing in the ticket_api response body [Resolved]
  - Wrong Error codes for the thrown error message [Resolved]

User Management System

- **Code Review**: This component manages the different types of users with different levels of access and permissions
- **Issues Reported**:
  - Partial Staff Data is not being deleted from Database [Resolved]
  - JWT token authorization is not implemented [Resolved]
  - Some fields are missing in the Role_api response body [Resolved]

Tag Management System

- **Code Review**: This component enables the administrator to create and manage tags that can be assigned to tickets
- **Issues Reported**:
  - JWT token authorization is not implemented [Resolved]

- Subject Tags should not be allowed to delete [Resolved]

7.2 VueJS Frontend Code Reviews

- **Code Review**: This section points out the issues reported by the team in the GitHub.
- **Issues Reported**:

1. On clicking the Subject Title the user should be redirected to the subject dashboard where all the tickets of that subject will be displayed. [Resolved] Link
2. Unapproved staff can successfully login which shouldn't be allowed. [Resolved] Link
3. The staff can easily go to other subjects' dashboard which the staff is not assigned to. [Resolved] Link 1 Link 2
4. Support staff can view the tickets of other subjects after using the search bar [Resolved] Link
5. Subject dashboard and Student dashboard UI needs improvement [Resolved] Link
6. Anyone can post an empty response body i.e. any user can just click the Add Response button without writing anything in the response body and this will post it. [Resolved]link

Full commit details can be seen [here] (https://drive.google.com/file/d/1U5DQcah_qDti0CLAKBrzbe-gLCSLe7o-/view?usp=sharing)

CHAPTER-8: Implementation Details

8.1 Technologies and Tools Used

- VS Code as IDE for both API and frontend development
- Git and Github for collaborating on backend and frontend of the project
- JIRA for Project Management
- Vue CLI for development of frontend

8.2 Instructions to run application

Frontend

1. Clone the repository using this command

```git
git clone https://github.com/mendax-iitm/sts-frontend.git
```

2. Change the directory

```
cd sts-frontend
```

3. Run the command to install necessary packages and dependencies

```
npm install
```

4. Run the command to deploy the application

```
npm run serve
```

Backend (API)

1. Clone the repository using this command

```
git clone https://github.com/mourya96/SE-Project-STS.git
```

2. Change the directory to the project directory

```
cd SE-Project-STS
```

3. Run the command to setup necessary packages for API

```
sh local_setup.sh
```

4. Run the command to start the API

```
python3 app.py
```

8.3 Hosting Details

Backend API Hosting

- The Backend API for CRUD operations are hosted using `pythonanywhere` python based server.
- Python based Web server is created using Flask module on https://teamfmm.pythonanywhere.com.

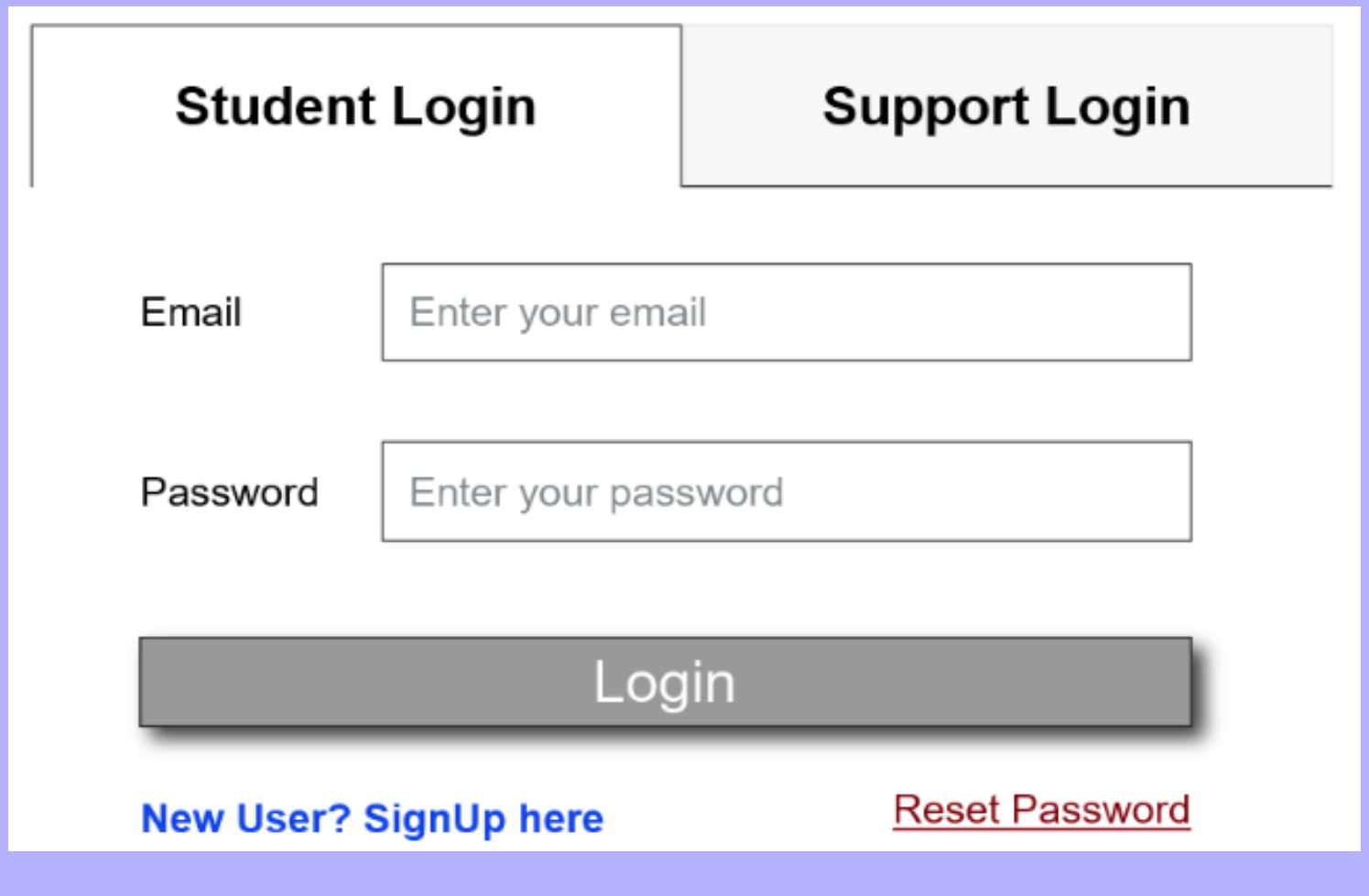
Frontend Application Hosting

- The frontend is made using Vue CLI and the production build is then deployed on `netlify` which can be accessed through the link https://loquacious-chimera-bab8d9.netlify.app/.

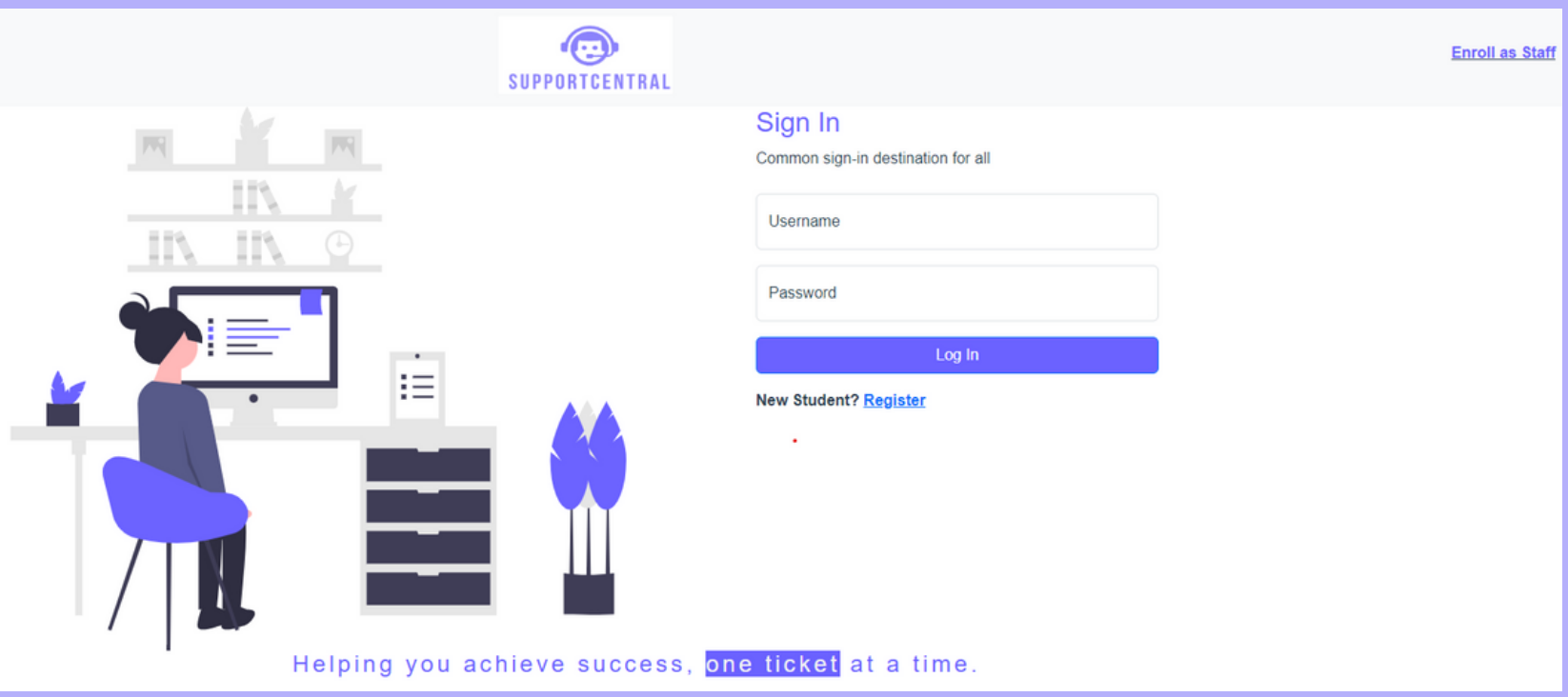
CHAPTER-9 Wireframes vs Actual Page

STUDENT LOGIN

Wireframe



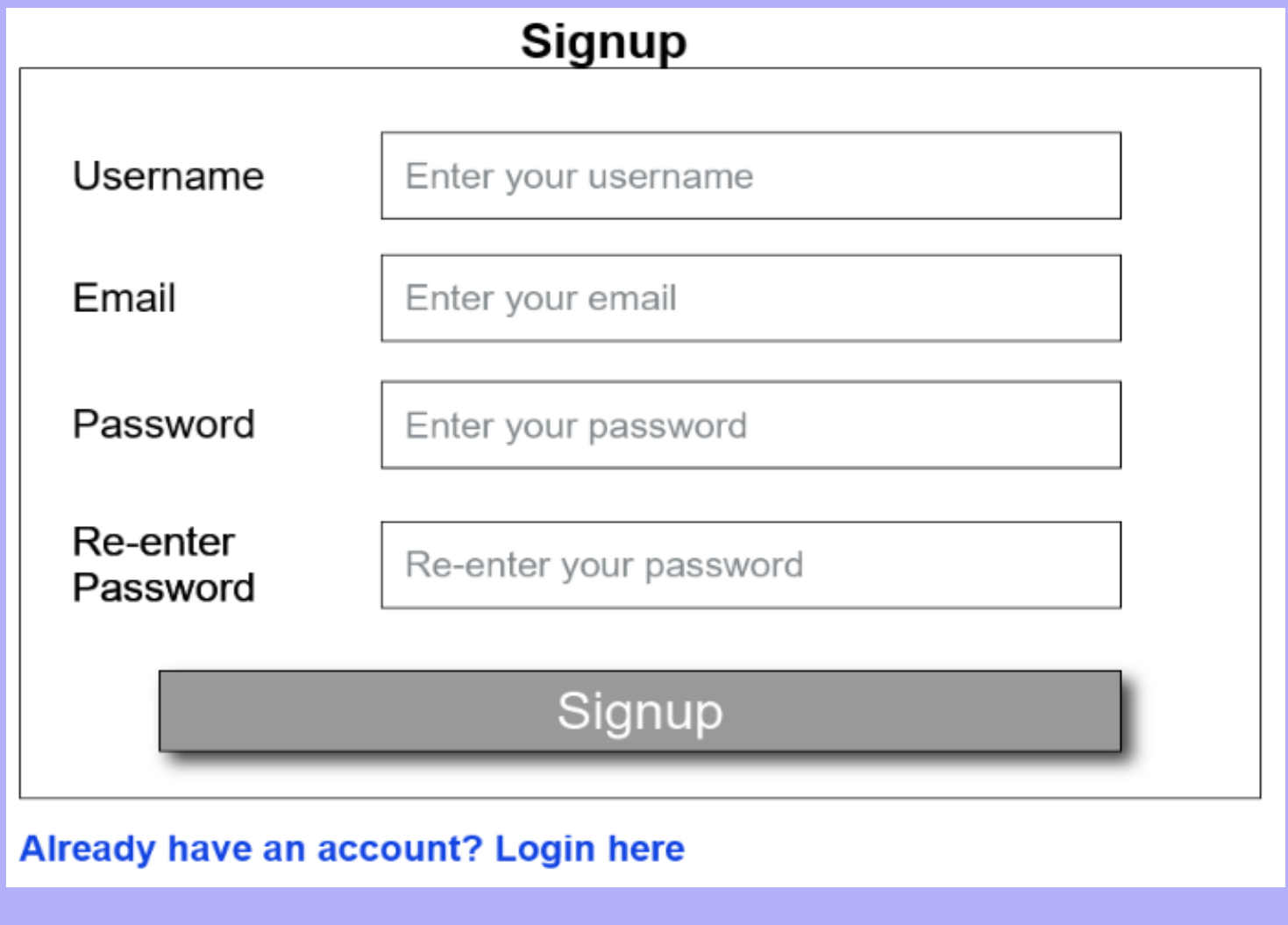
Actual



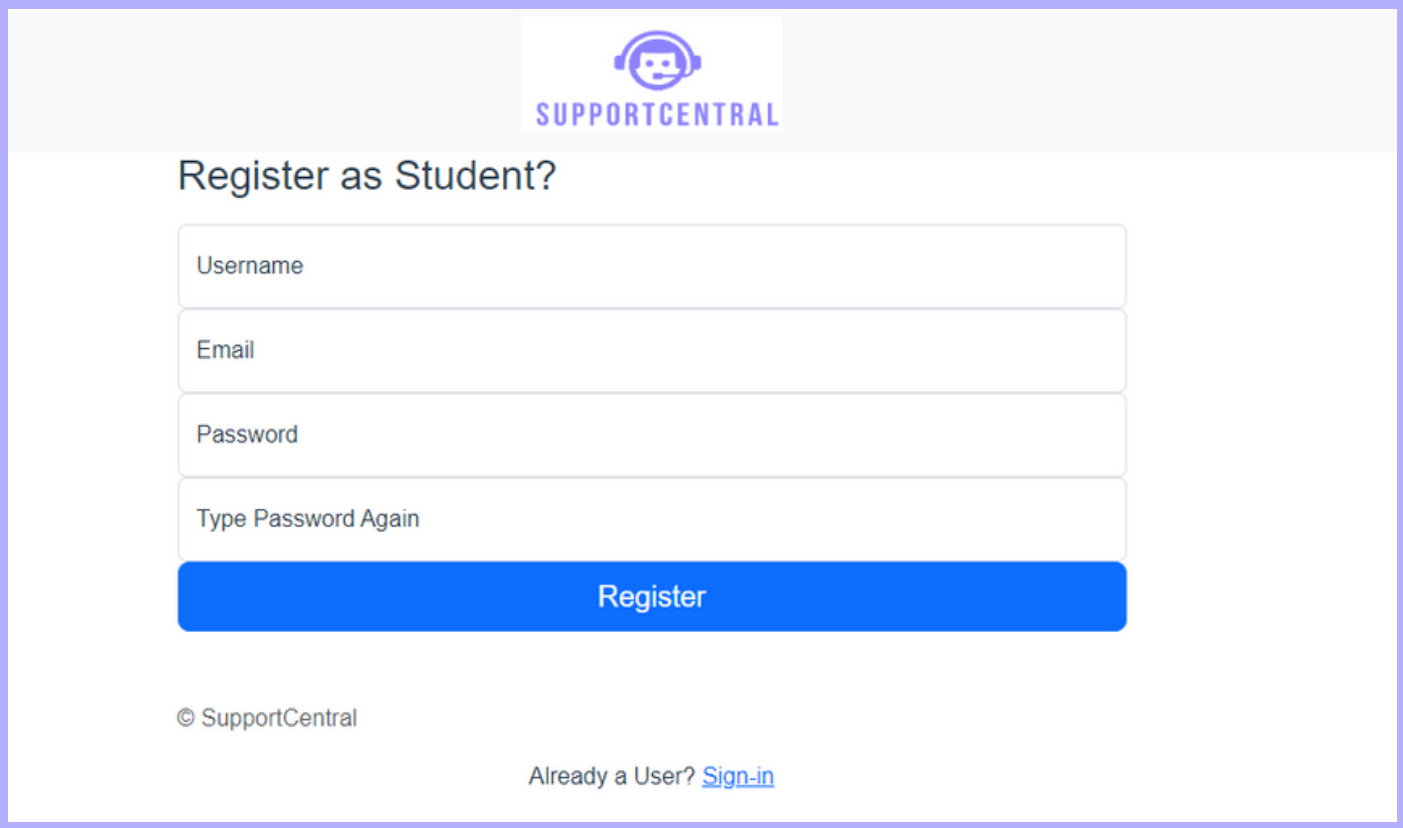
Note: The Login Page remains same for Student, Support and Admin

STUDENT SIGNUP

Wireframe



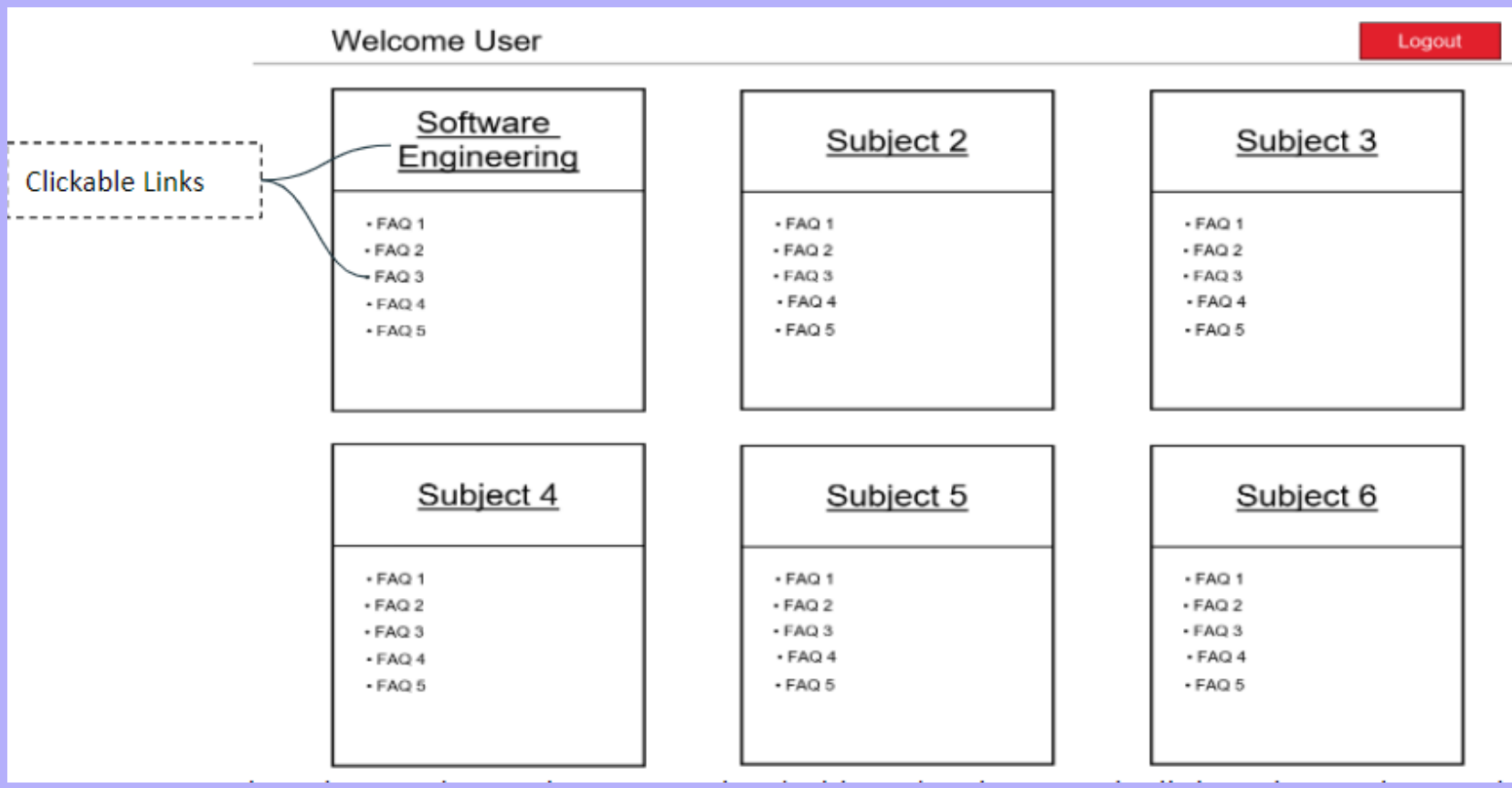
Actual



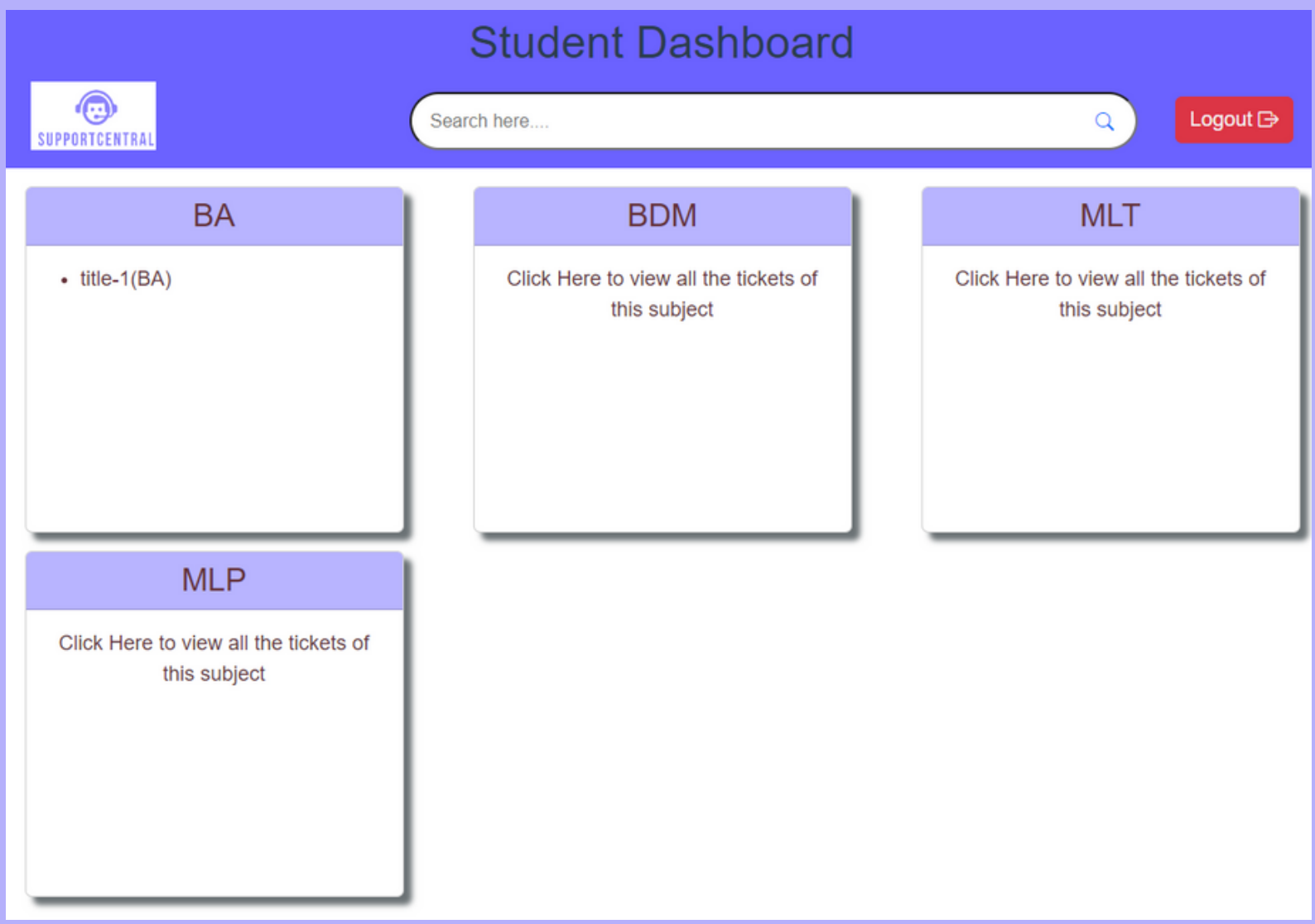
Note: The Admin Signup page is similar to Student Signup Page routed at some address

STUDENT DASHBOARD

Wireframe

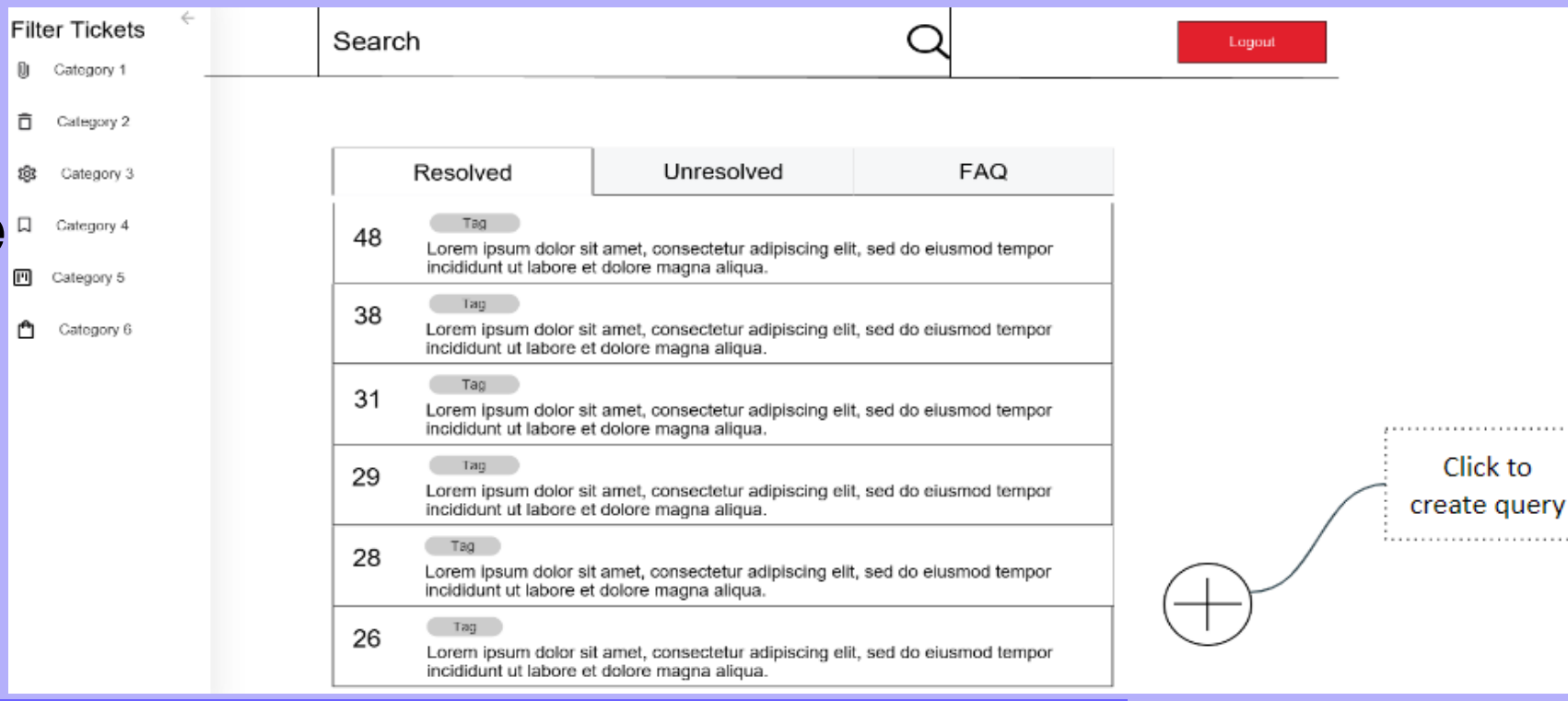


Actual

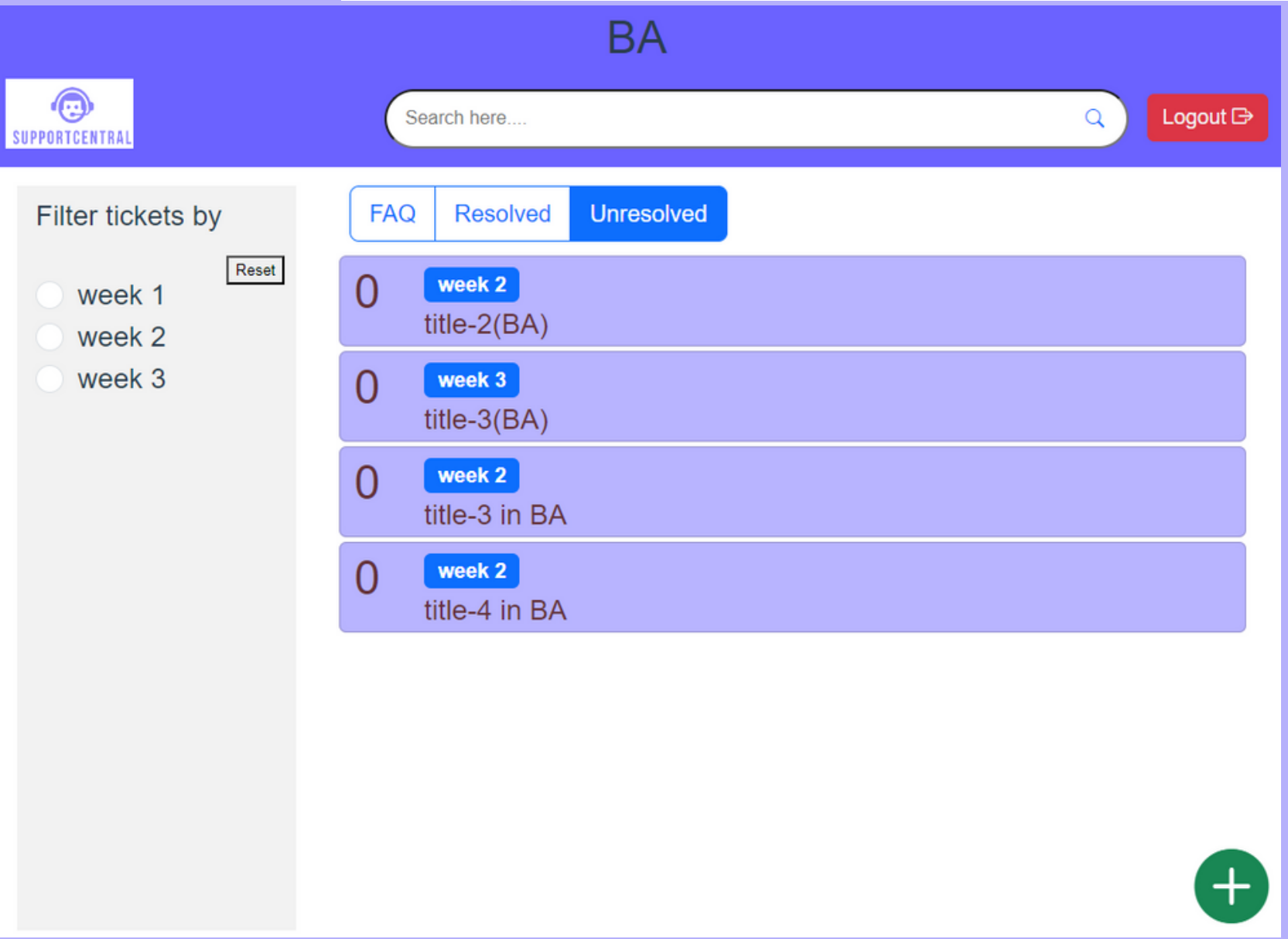


SUBJECT DASHBOARD

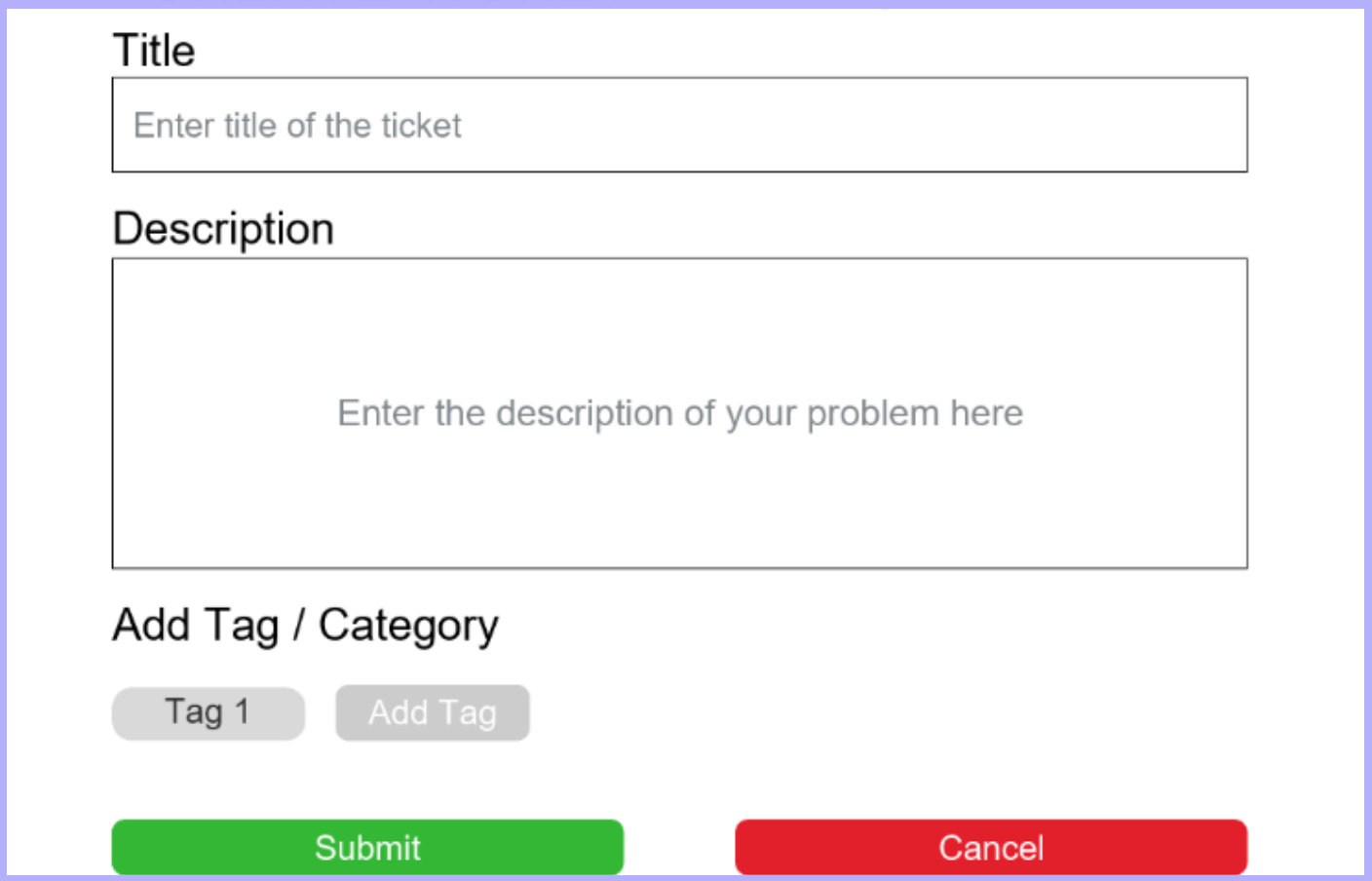
Wireframe



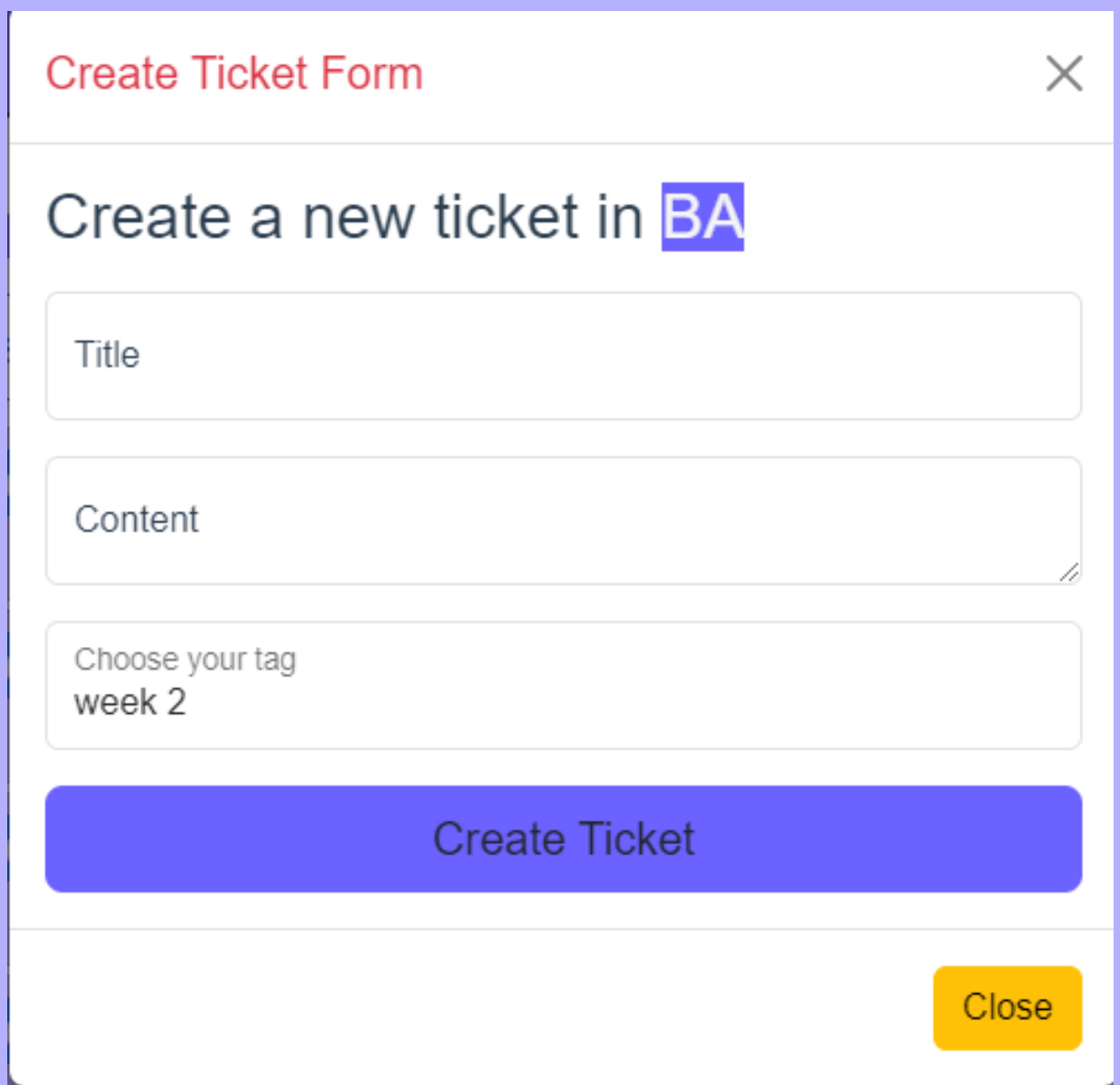
Actual



Wireframe

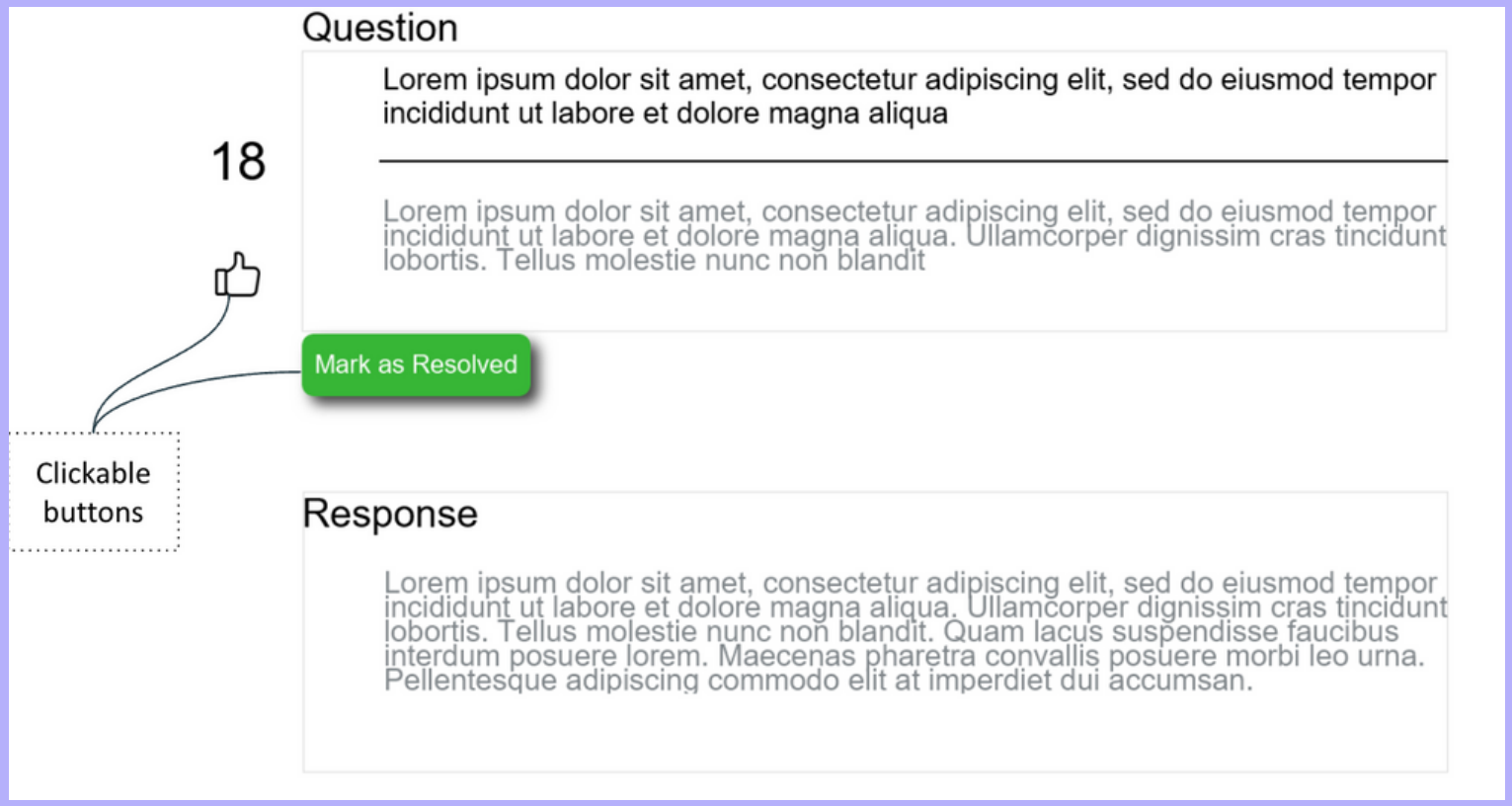


Actual

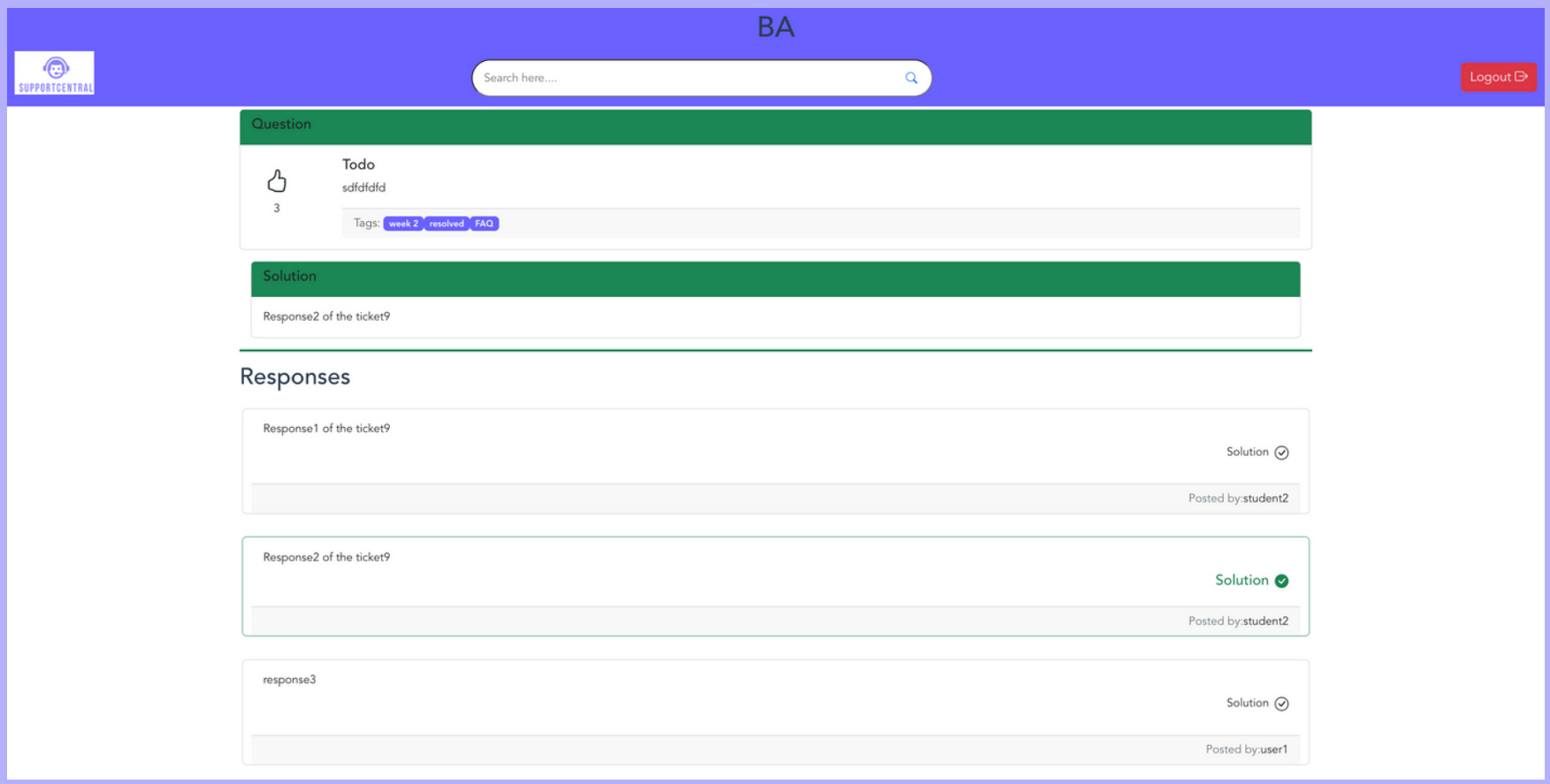


QUERY VIEW

Wireframe

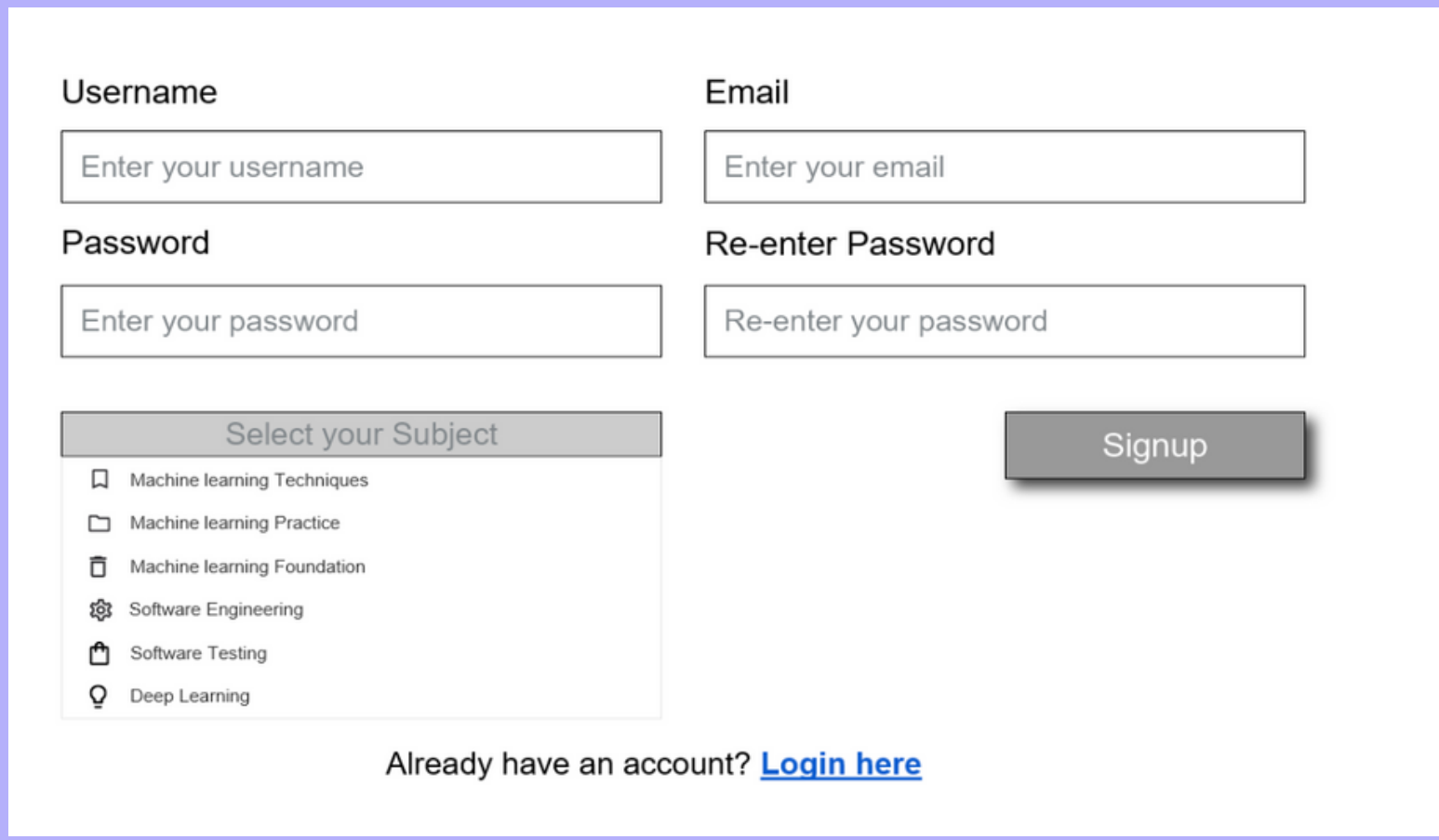


Actual

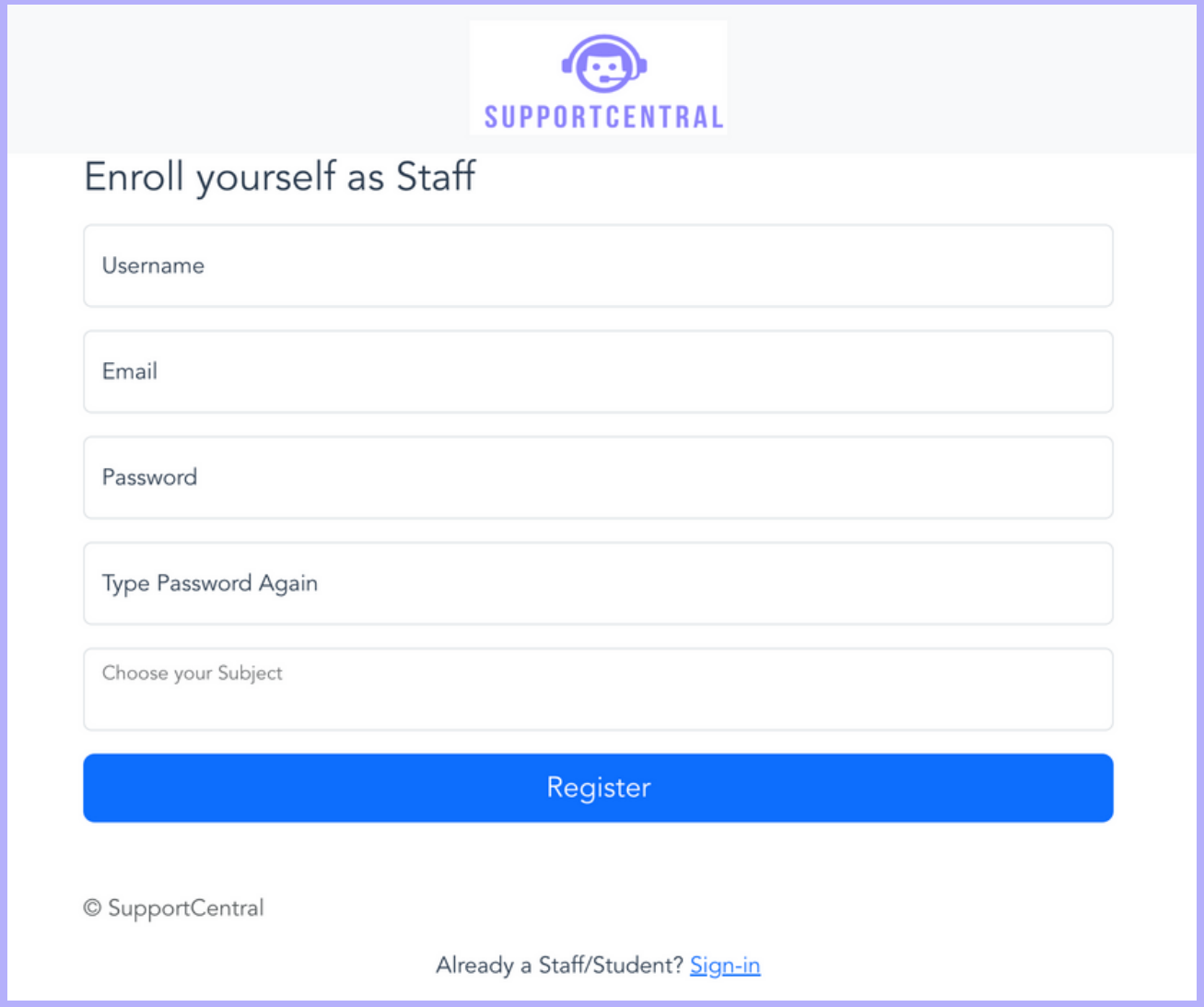


SUPPORT STAFF SIGNUP PAGE

Wireframe

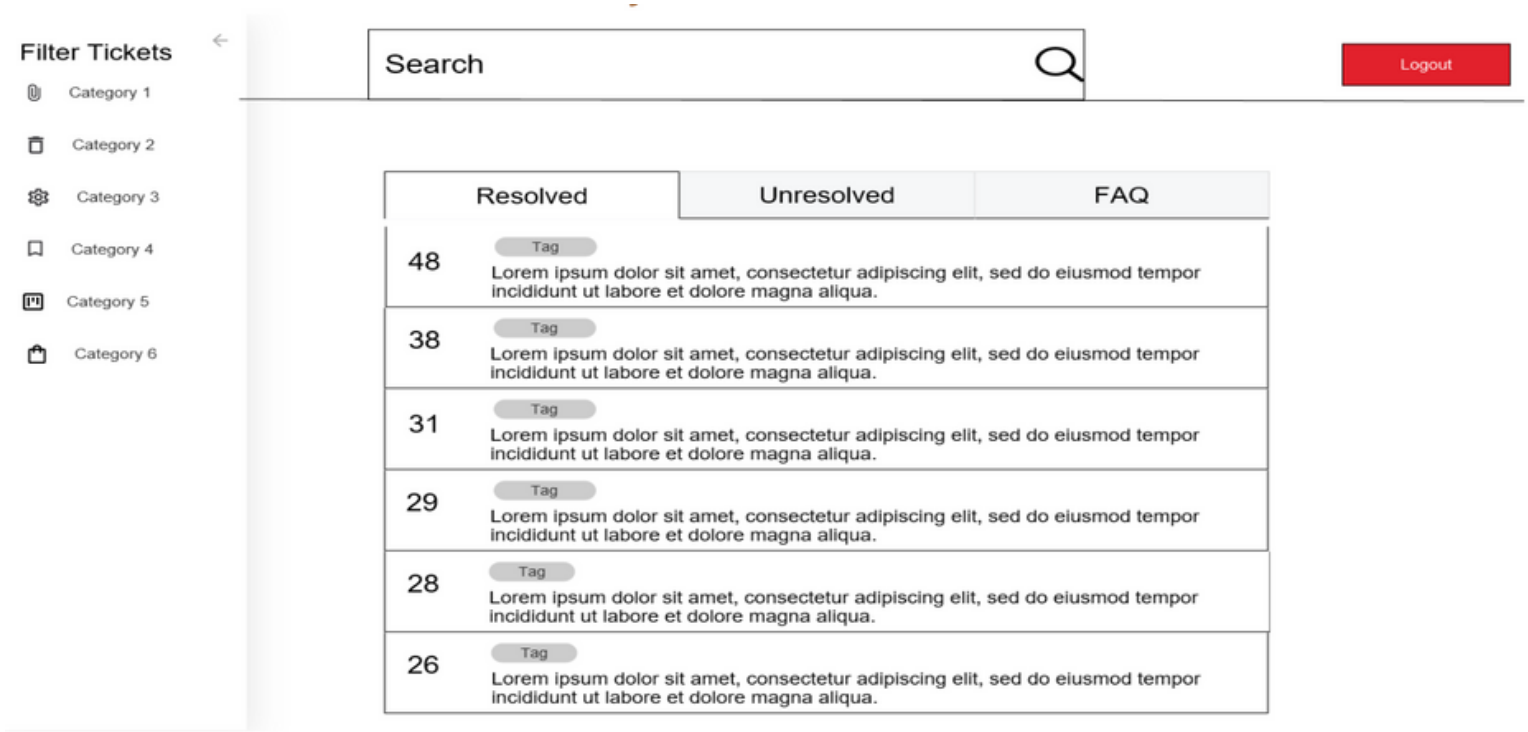


Actual

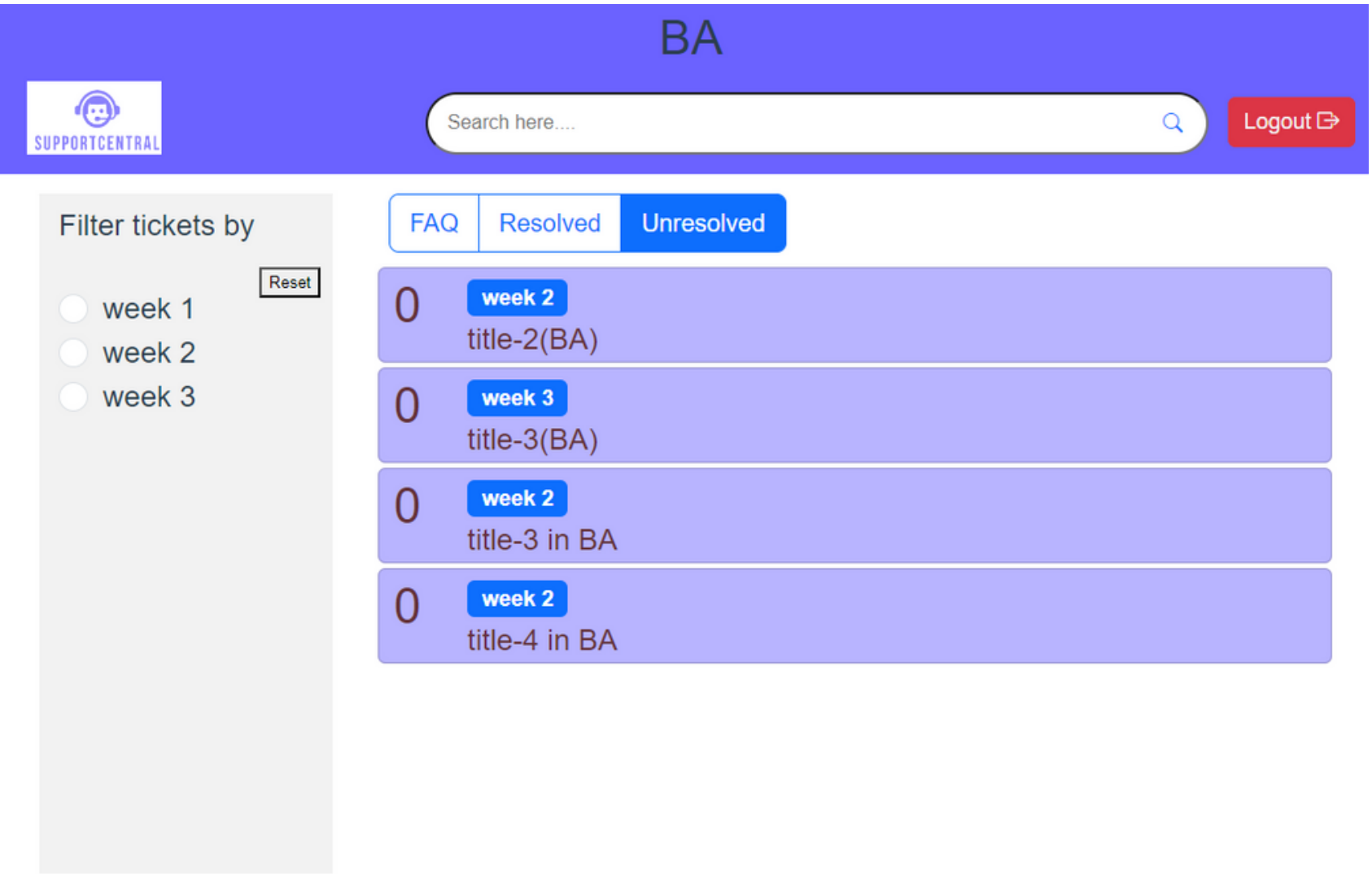


SUPPORT HOMEPAGE

Wireframe



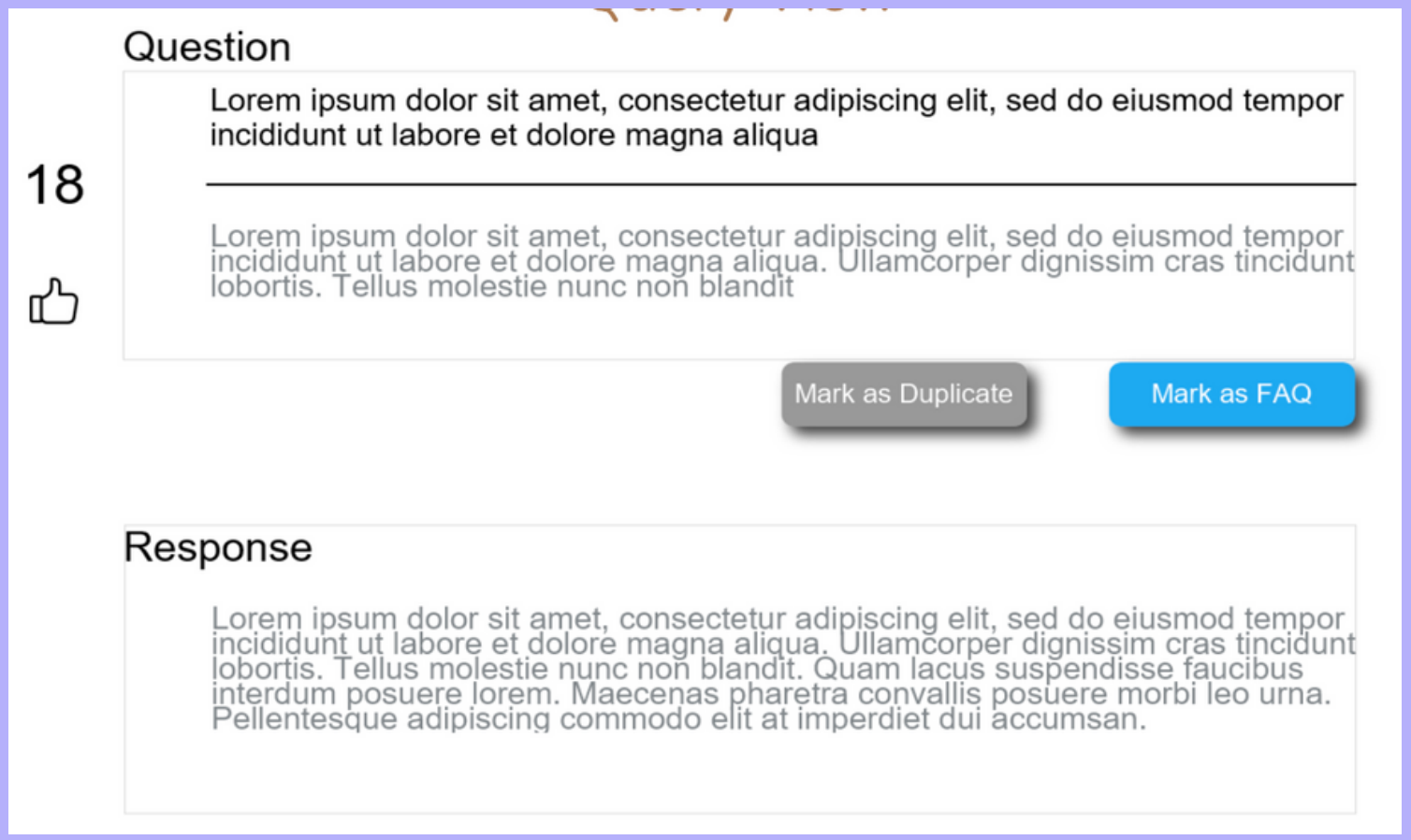
Actual



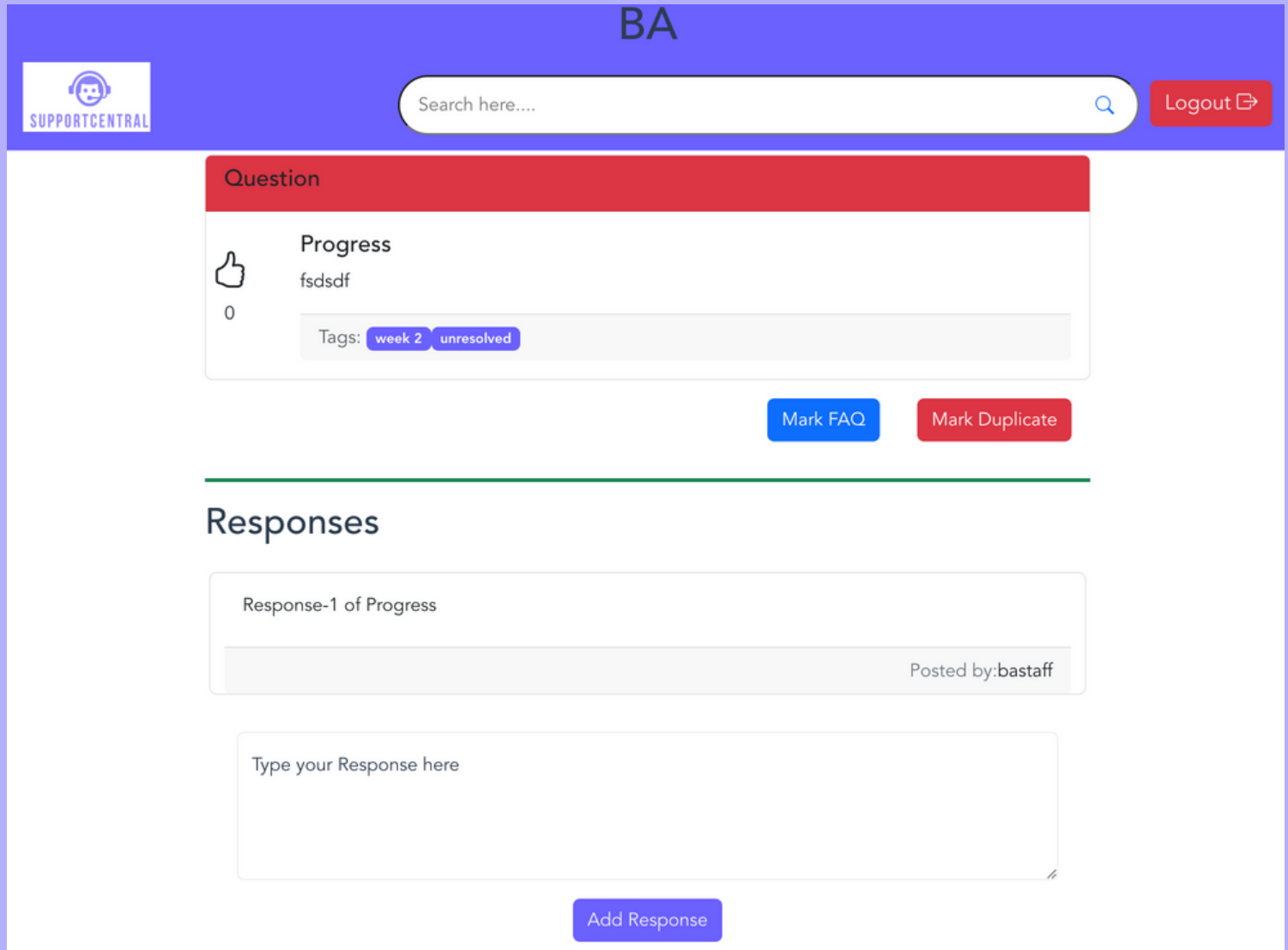
Note: The Subject Dashboard will be similar in case of admin.

SUPPORT QUERY VIEW

Wireframe

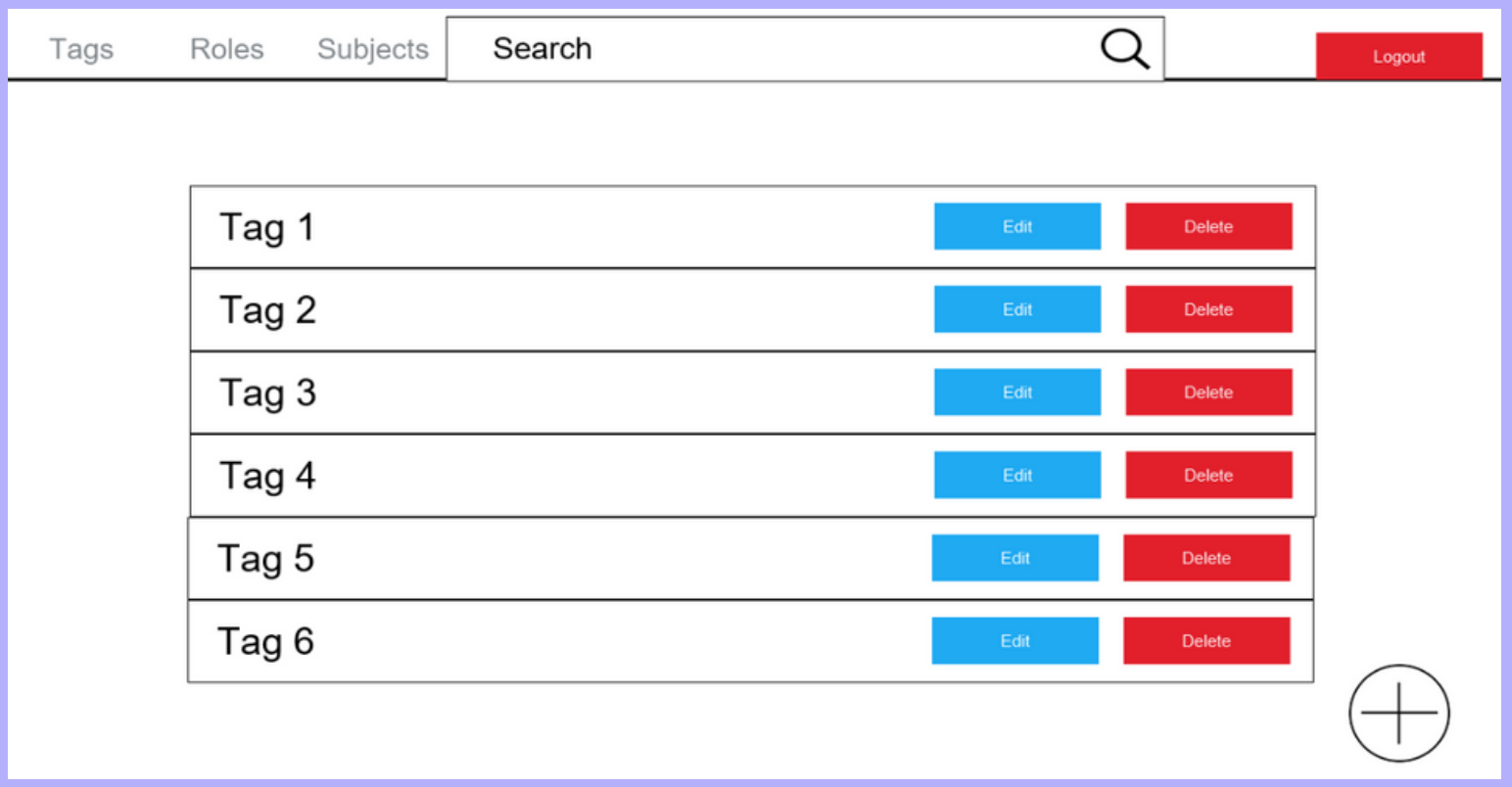


Actual

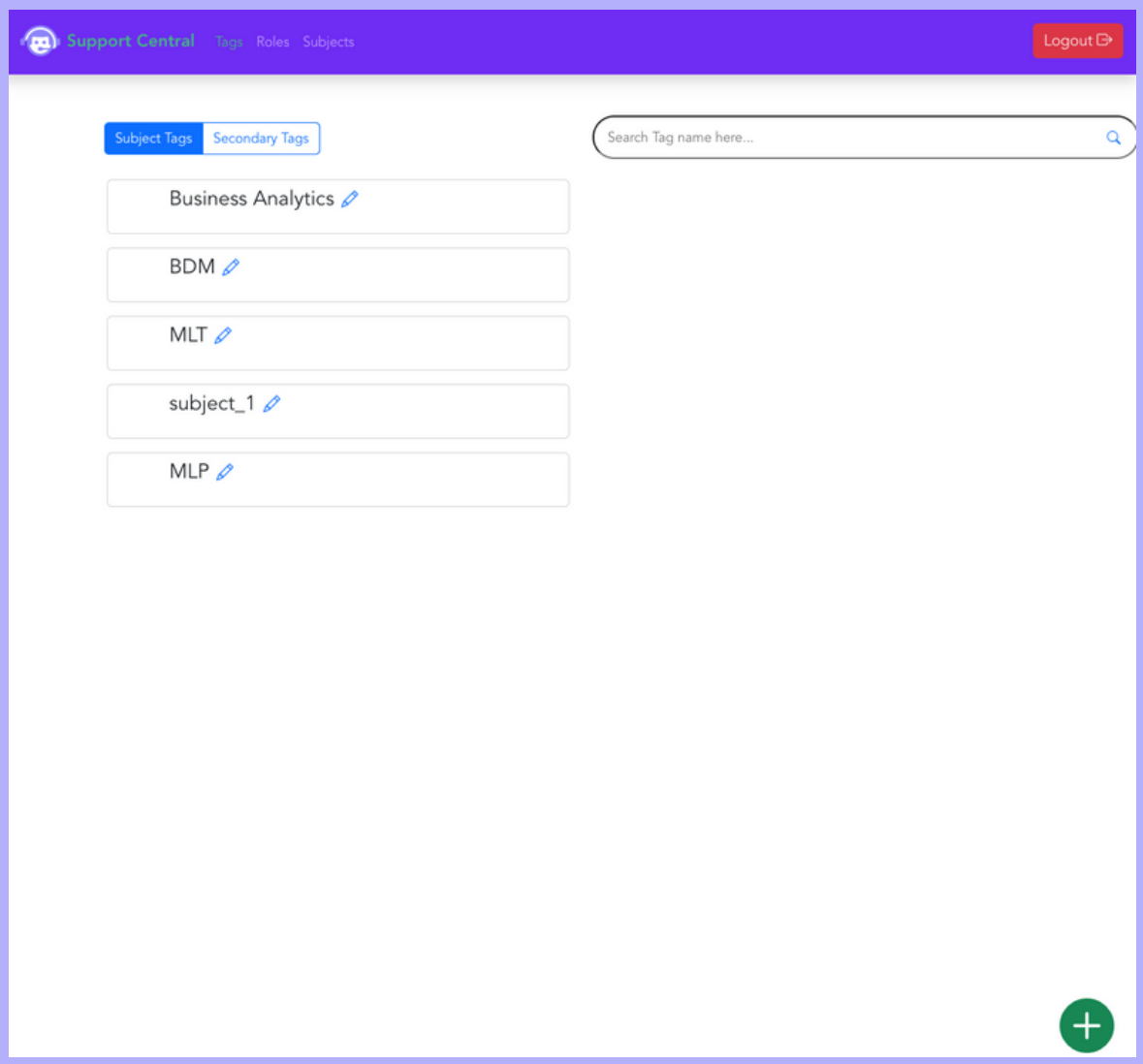


ADMIN DASHBOARD(TAG MANAGEMENT)

Wireframe

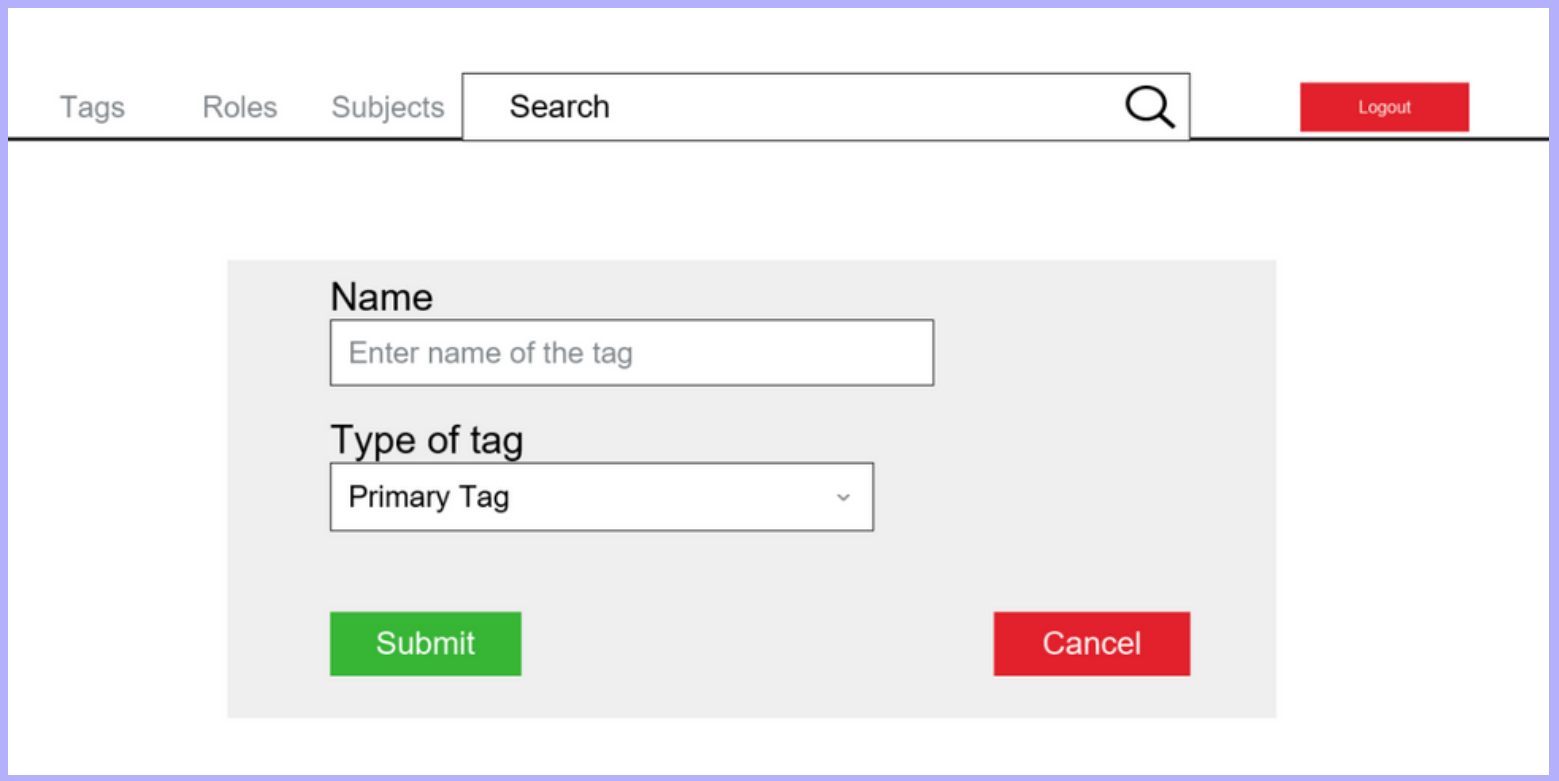


Actual

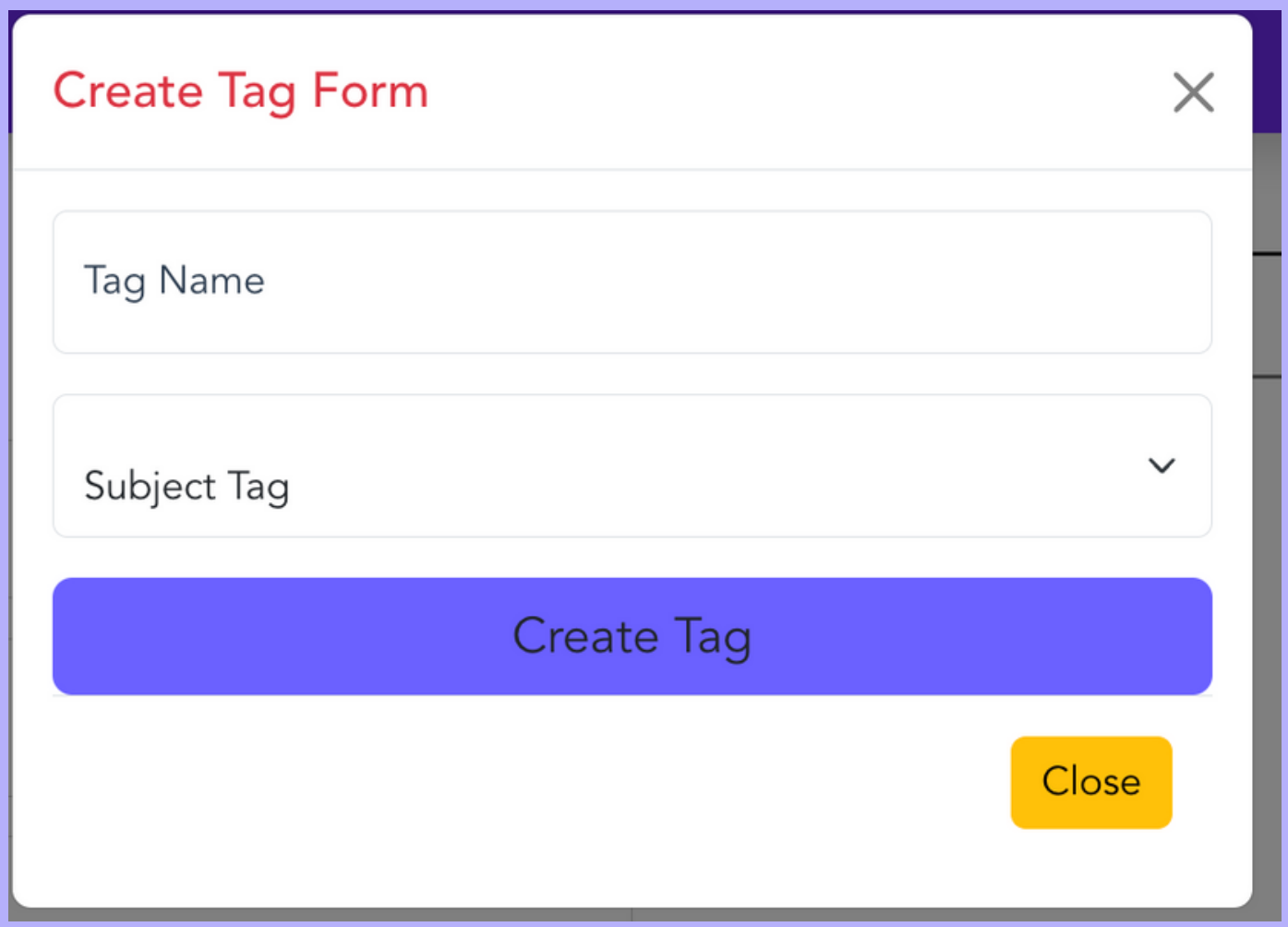


ADMIN DASHBOARD(CREATE TAG)

Wireframe

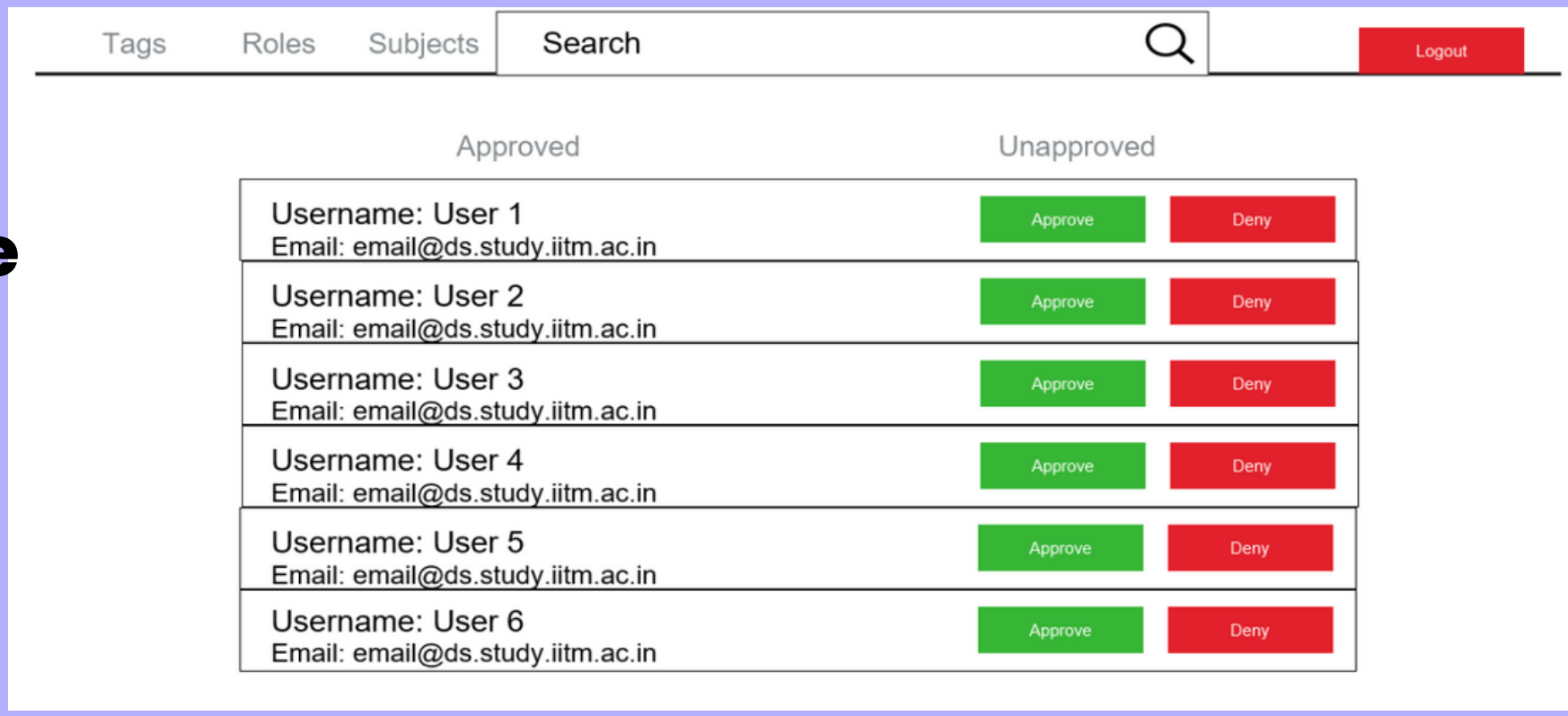


Actual

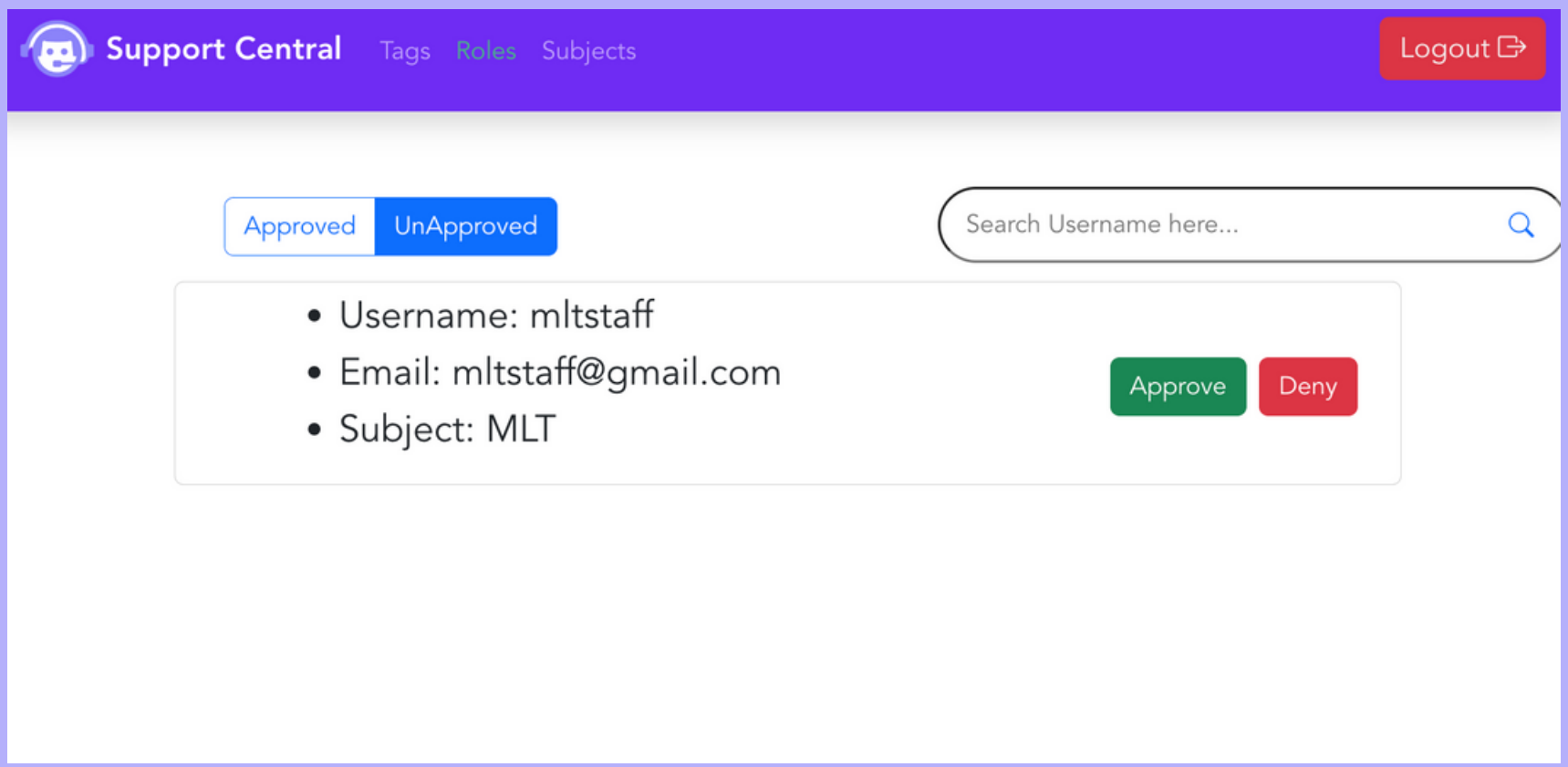


ADMIN DASHBOARD(ROLE MANAGEMENT)

Wireframe

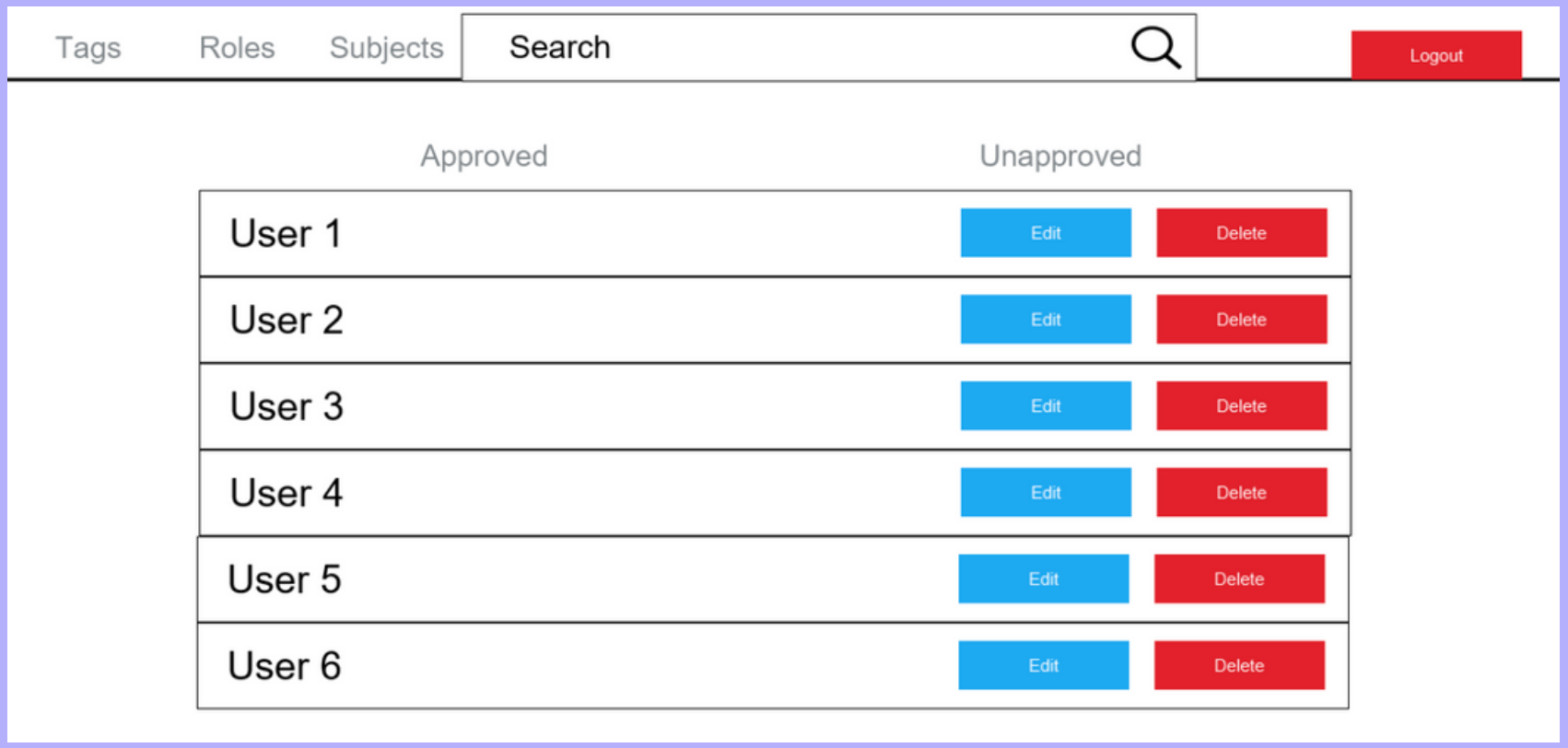


Actual

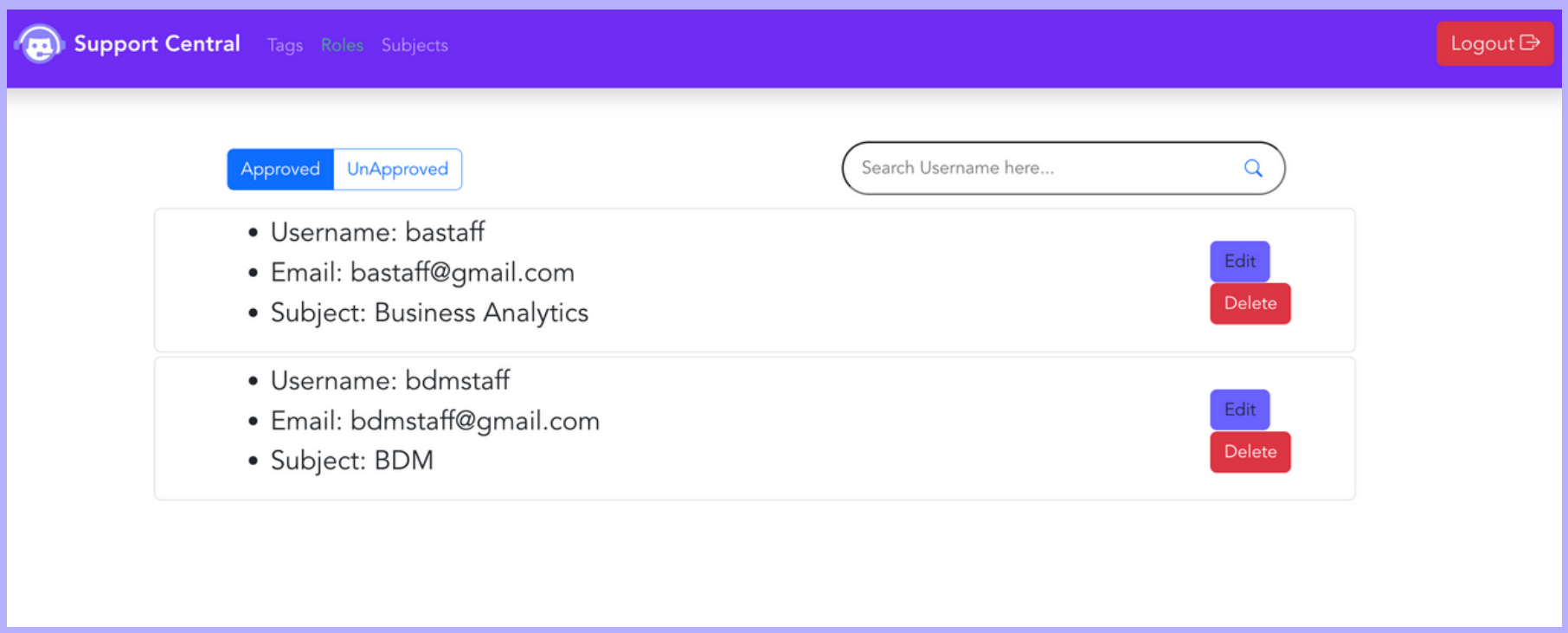


ADMIN DASHBOARD(ROLE MANAGEMENT)

Wireframe

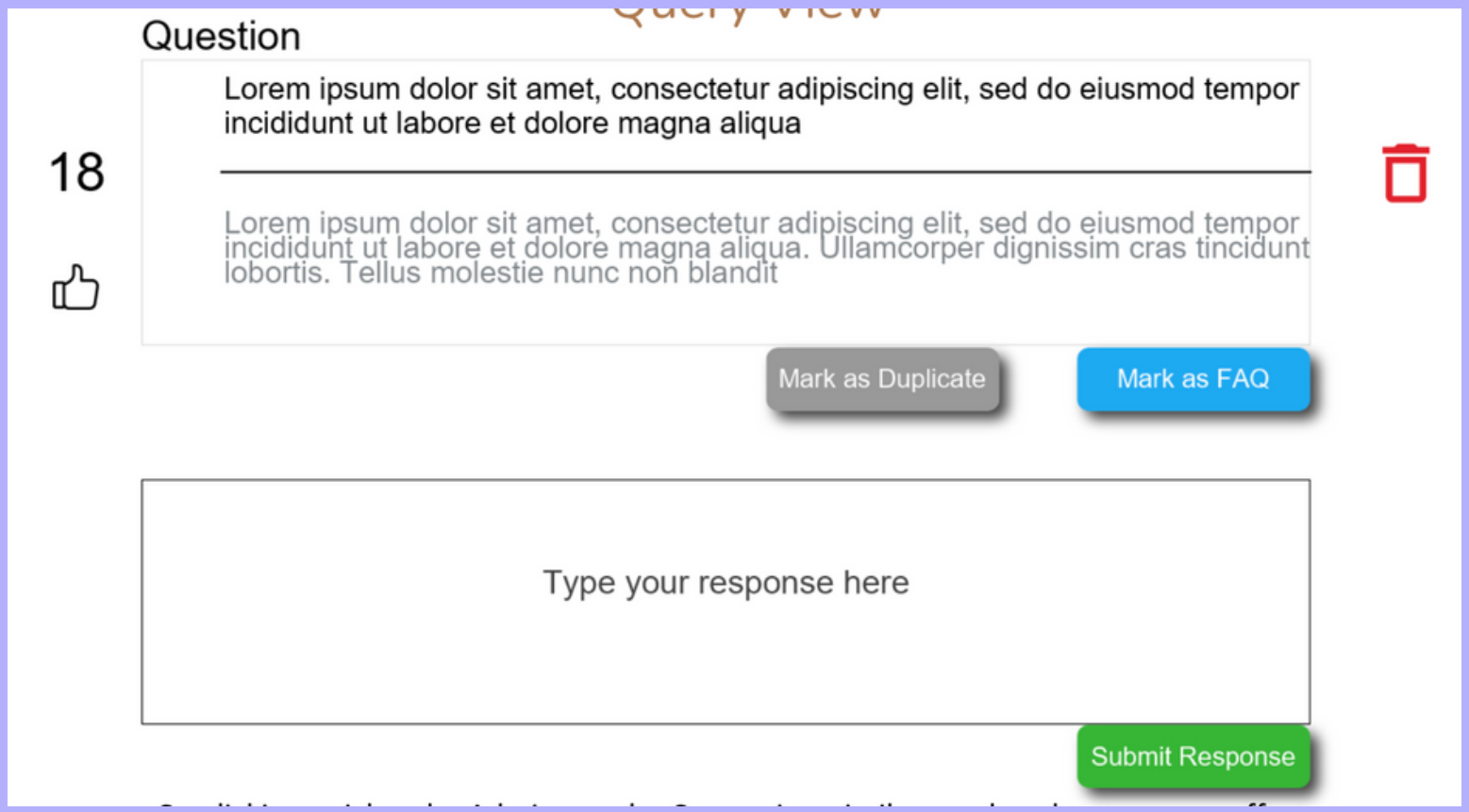


Actual



ADMIN QUERY VIEW

Wireframe



Actual