SOFTWARE ENGINEERING MILESTONE 1

SUBMITTED IN THE PARTIAL FULFILLMENT OF THE REQUIREMENTS OF THE COURSE:

BSCSS3001: Software Engineering

By:
Arya Bhattacharyya (21f2000436)
Varun Venkatesh (21f1000743)
Chirag Goel (21f2000540)

INDIAN INSTITUTE OF TECHNOLOGY, MADRAS
February, 2023

Identifying the various types of Users

Primary Users: Students, Support Staff, and Admins
Secondary Users: Managers
Tertiary Users:

- Software developers: A new feature request for the main IIT portal is highly upvoted. Thus Administrators/managers ask the software developers to implement that feature. Whilst they don't use the Support Desk, they are impacted by the use/decision of Primary and Secondary Users.
- The platform that hosts the website, the Internet Service Provider, etc.

User Stories

1. - **As a student**
   - I want to be able to create new support/query tickets
   - So that I can get help with my issues from the support team

2. - **As a student**
   - I want an edit option to edit my ticket post submission and before resolution
   - so that I can convey myself better if there is a need.

3. - **As a student**
   - I want the ability to delete a previously submitted ticket by me
   - so that I do not hold the queue up if my query has been resolved by me already.

4. - **As a student**
   - Before submitting my query ticket, I want the system to show if similar query tickets based on the title or content have been raised previously or are in the FAQ section
   - So that I can "+1" the relevant ticket or directly resolve it if already answered.

5. - **As a student**
   - I want to be able to "+1" existing tickets created by other students,
   - So that the support staff know how many students are facing a certain issue without creating a duplicate ticket

6. - **As a Support Agent**
   - I want the ability to sort queries based on their time of raising, the number of "+1"'s and also on the basis of reopened issues
   - So that I can strategise on which queries need to be given higher priority and need quicker resolution.

7. - **As a Support Agent**
   - I want to be able to mark a ticket as closed/resolved
   - So that other support staff can focus on other tickets pending resolution.

8. - **As a student**
   - I want to receive email notifications within 10 minutes of when my ticket has been responded to
   - So that I can be up to date on my ticket and take any steps that might be required

9. - **As a Support Agent**
   - I want to send automated email notifications to the student within 10 minutes of responding to a ticket,
   - So that the students are notified when a resolution is provided in a timely manner and take any steps that might be necessary

10. - **As a Support Agent**
    - I want the ability to suggest to the admin to add certain tickets (based on my recommendation) to the FAQ section
    - So that the support team doesn't have to answer the same queries repeatedly.

11. - **As an admin**
    - I want the ability to dynamically update the FAQ section based on recommendations from support agents
    - so that future students can resolve their queries by themselves.

12. - **As an admin**
    - I want the ability to categorise the FAQ section queries
    - so that students can easily find the queries based on their categories.

13. - **As an admin**
    - I want to create and assign different levels of permissions and roles like "student", "staff" and "admin"
    - So that appropriate information and actions are accessible to only those who are authorized to

14. - **As an admin**
    - I want to be able to enroll new people to the software by directly using their email IDs
    - So that they can use the platform for query raising and resolution and I do not have to add each user manually.

15. - **As a manager**
    - I want to be able to enroll an admin
    - So that they can enroll new users and assign appropriate roles/permissions.

16. - **As a student**
    - I want to rate the resolution provided by the support staff,
    - So that I can provide useful feedback to the support staff and improve future resolutions

17. - **As a Support Agent**
    - I want the ability to filter user tickets based on the username,
    - So that I can view the students ticket history at a glance and provide better resolution

18. - **As a student**
    - I want to have a ticket history view,
    - So that I can refer to my past tickets and resolutions incase I need to refer to them again in the future

19. - **As a manager**
    - I want to receive a notification when a ticket hasn't been answered by a support agent within 72 hours,
    - So that I can follow up on the status of the ticket with my support agents to provide effective and timely resolution to the students

20. - **As a manager**
    - I want to receive a notification whenever a support agent's average resolution time in the last 30 days is greater than 48 hours
    - So that I can address the agent and find a solution to the delayed response times

21. - **As a student**
    - I want to be able to reopen a ticket that's marked as closed,
    - So that I can receive further support incase i'm not happy with the resolution

22. **As a Support Agent**
    - I want the ability to flag a post as rude/offensive as per my discretion
    - So that they do not disturb the learning environment.

23. **As a Support Agent**
    - I want the ability to forward previously flagged offensive posts to the admin
    - So that they can take further action as deemed appropriate.

24. **As an admin**
    - I want to be able to remove a particular user by their username
    - So that if any student drops the degree programme or has disciplinary issues, we can revoke access to the platform.

25. **As a manager**
    - I want to be able to get the resolution times of queries
    - so that I can perform an analysis to understand the promptness of the query resolution process.

SOFTWARE ENGINEERING MILESTONE 2

SUBMITTED IN THE PARTIAL FULFILLMENT OF THE REQUIREMENTS OF THE COURSE:

BSCSS3001: Software Engineering

By:
Arya Bhattacharyya (21f2000436)
Varun Venkatesh (21f1000743)
Chirag Goel (21f2000540)

INDIAN INSTITUTE OF TECHNOLOGY, MADRAS
February, 2023

Storyboard 1

From the perspective of a support agent


Hi, Ethan! I'm Raj. I just joined in today as a support agent for IITM BS.
Hi, Raj! You look a bit worried... What's up? Is everything fine?
At IITM BS Degree Support Desk Office

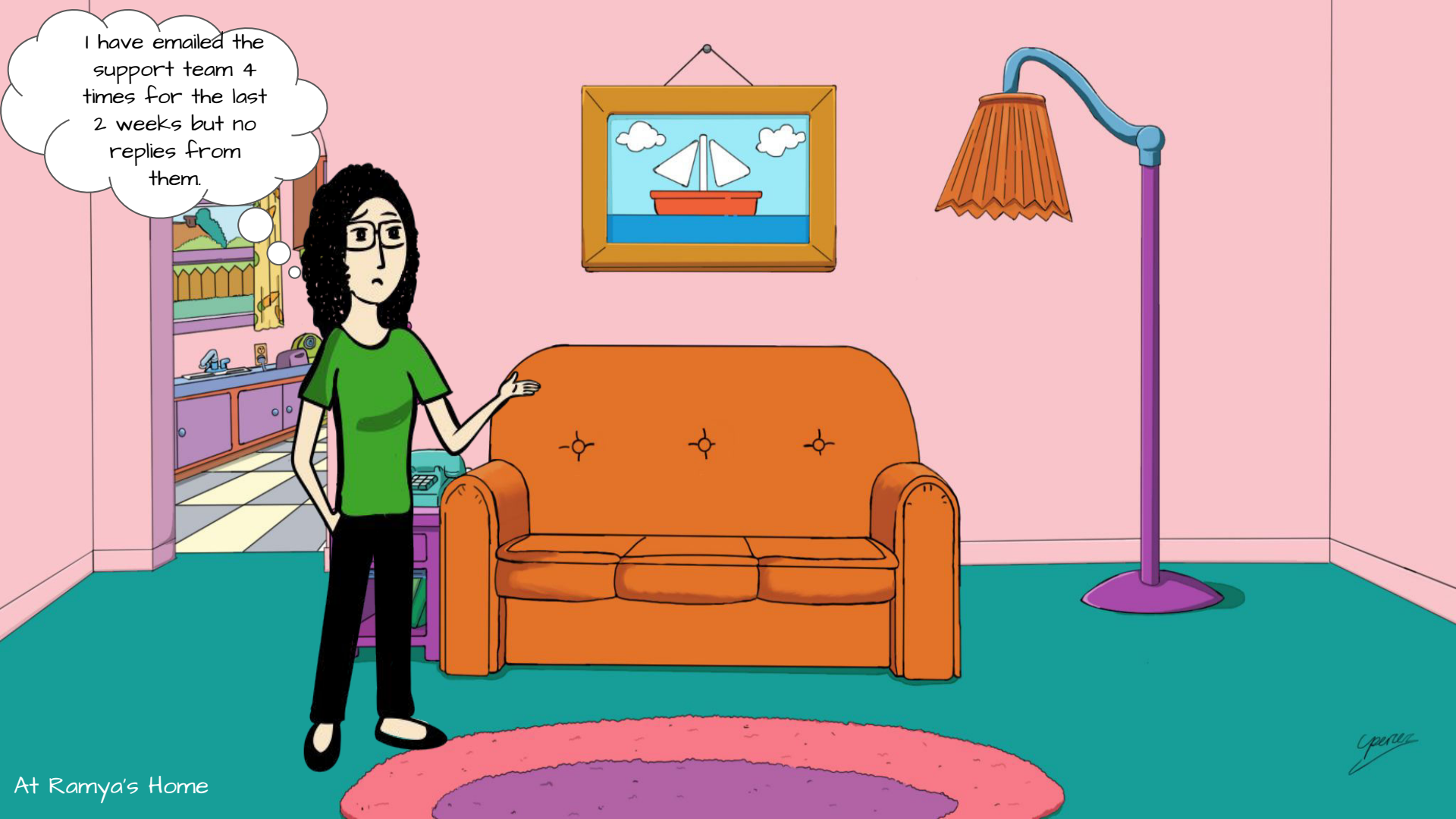

Yeah actually in my previous job as a support agent, I used to get extremely tired with tons of queries on email.
Is that all? Don't worry, we have got an amazing system in place here!
At IITM BS Degree Support Desk Office


This cool software by Sociogrammers easily organises the current unresolved queries, and it also has the option to filter them by various criteria!
Hi Raj,
Logout
Filter
Sort
Filter queries on the basis of various parameters
Sorting queries on different parameters
Topic Name
Flag
Description
Actions
Topic Name
Flag
Description
Actions
Reply
Mark Closed
Suggest for FAQ
On click the following prompt is shown
Are you sure you want to flag this post as offensive ?
Yes
No
Wow! This is great! I think I am going to enjoy working here as a support agent!
At IITM BS Degree Support Desk Office Presentation Room


Thanks Ethan for assuaging my fears. I'll get to work!
You're welcome, Raj! I am glad I could be of help.
At IITM BS Degree Support Desk Office

Storyboard 2

From the perspective of a student


I have emailed the support team 4 times for the last 2 weeks but no replies from them.
At Ramya's Home


But I still cannot access my courses properly. Is it just me or others are facing this too?.
At Ramya's Home


I am a bit worried about this. Let me message Saroj about this to ask her on what can be done..
At Ramya's Home


Hey Saroj! I have emailed the support team multiple times but not heard back! I can still not access my courses. What should I do?
At Ramya's Home


Hey Ramya! We have this support and query platform designed by Sociogrammers. You can post your query there and also check the FAQ section/ past queries to know if it has already been answered and if others are facing the same issue as you.
Woah! How did I not know about this? Anyways, let me try this out!
At Ramya's Home


A few moments later
From now on, I will use this platform to resolve queries. It also gives me the option to rate the resolution. Amazing!
Wow! This is great! I find that my query has already been resolved and is now in the FAQ section! This gives me more confidence and now I have no worries.
At Ramya's Home

Storyboard 3

From the perspective of a support agent

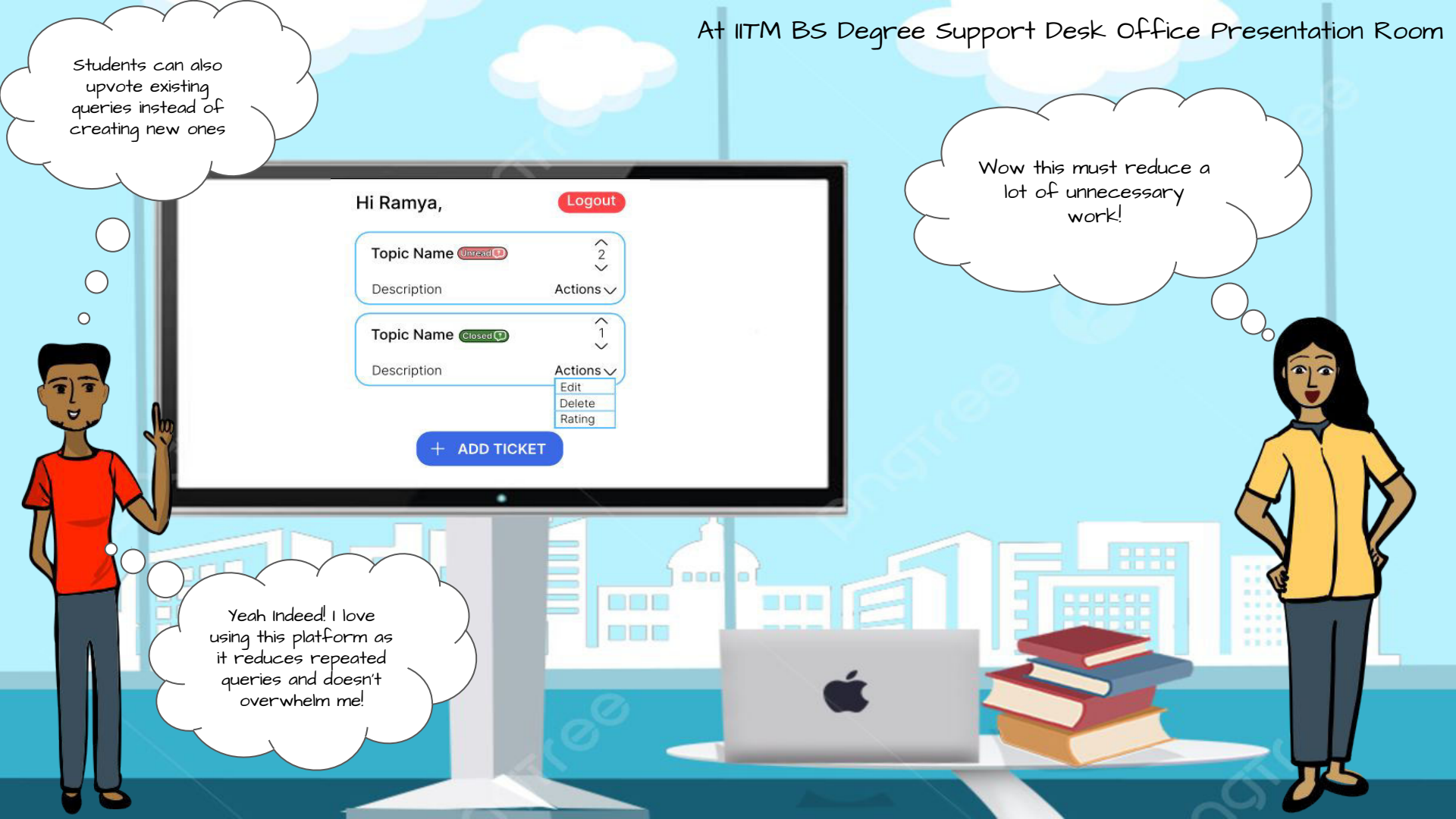

Hi I'm Priya. Today is my first day here. It's nice to meet you.
Hi, Priya! I'm Anil. Very happy to have you here. How are you?
At IITM BS Degree Support Desk Office

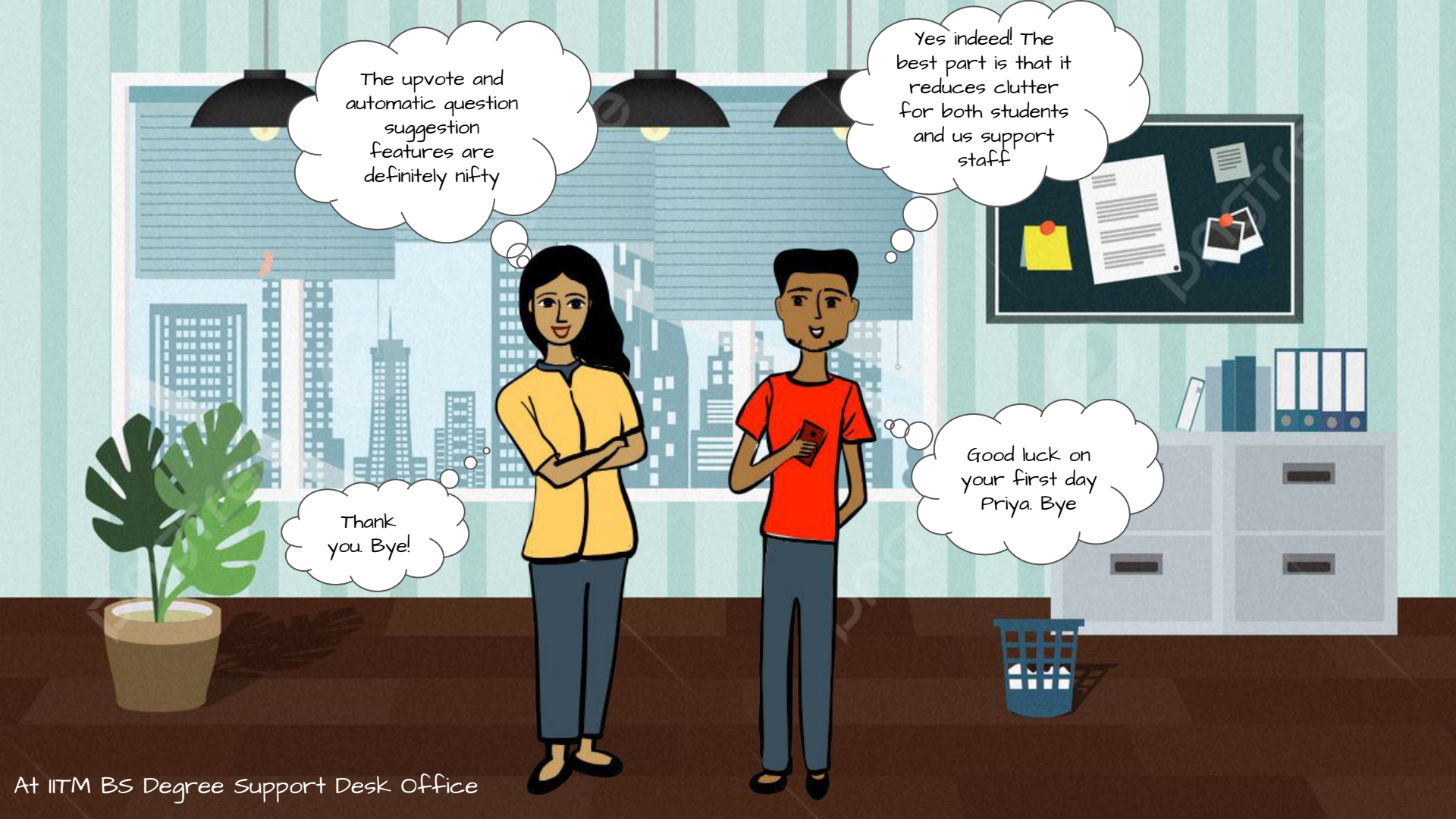

Oh I'm just a little worried.
Oh how come?
At IITM BS Degree Support Desk Office


You know how this job goes. It can get very monotonic and boring when most of our job is replying to the same query again and again
Oh you didn't know? We use The Sociogrammers support desk software here which makes our job very easy. Let me show you
At IITM BS Degree Support Desk Office


At IITM BS Degree Support Desk Office Presentation Room
When a user is typing their query, the software automatically searches for similar queries and suggests it to users
Id Card Error
These queries match well with your query. Consider up voting them if relevant, instead of creating a new query!
Id Card picture error
12
Wrong Name on Id Card
10
Id Card Address Change
8

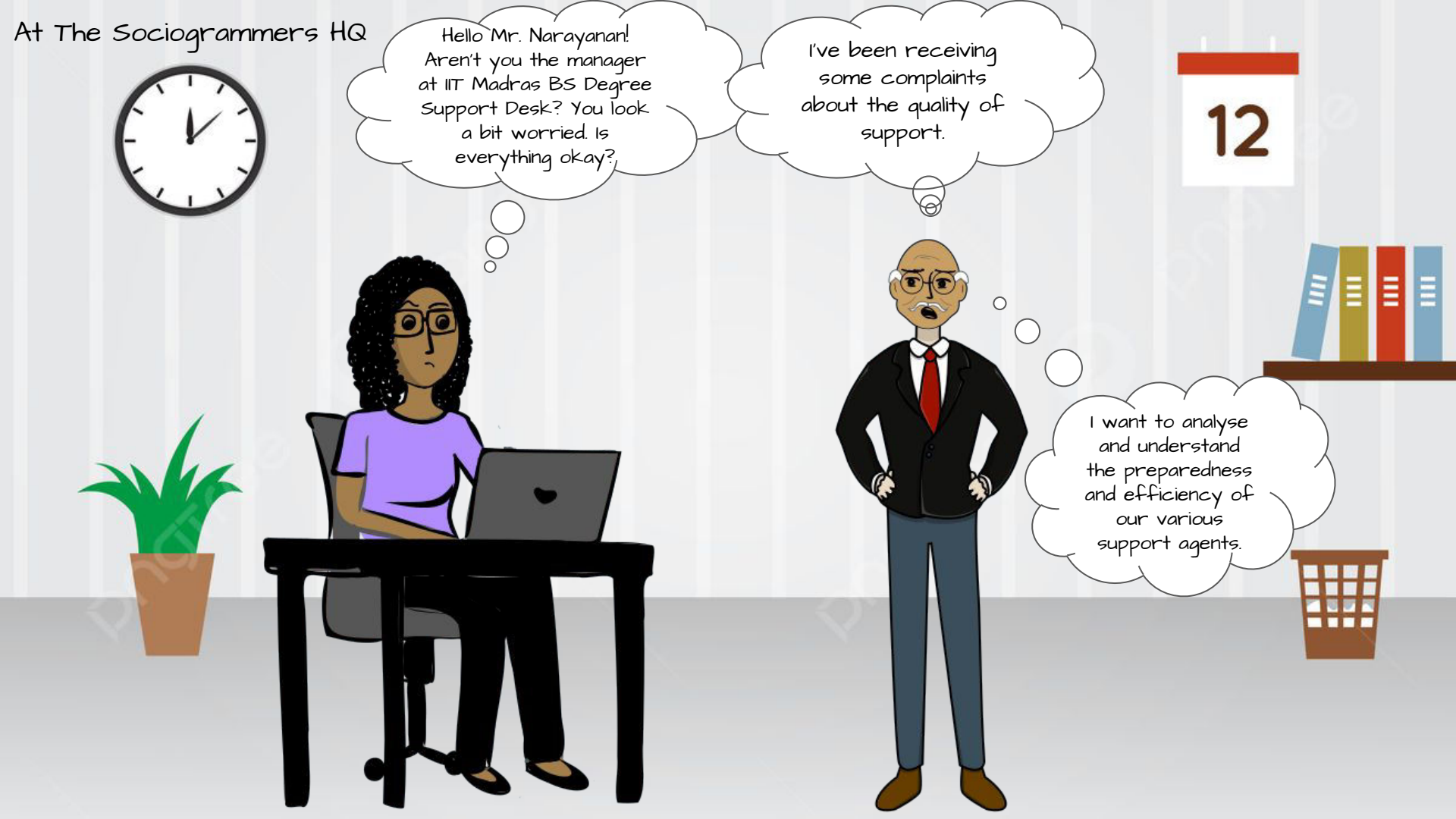
At IITM BS Degree Support Desk Office Presentation Room

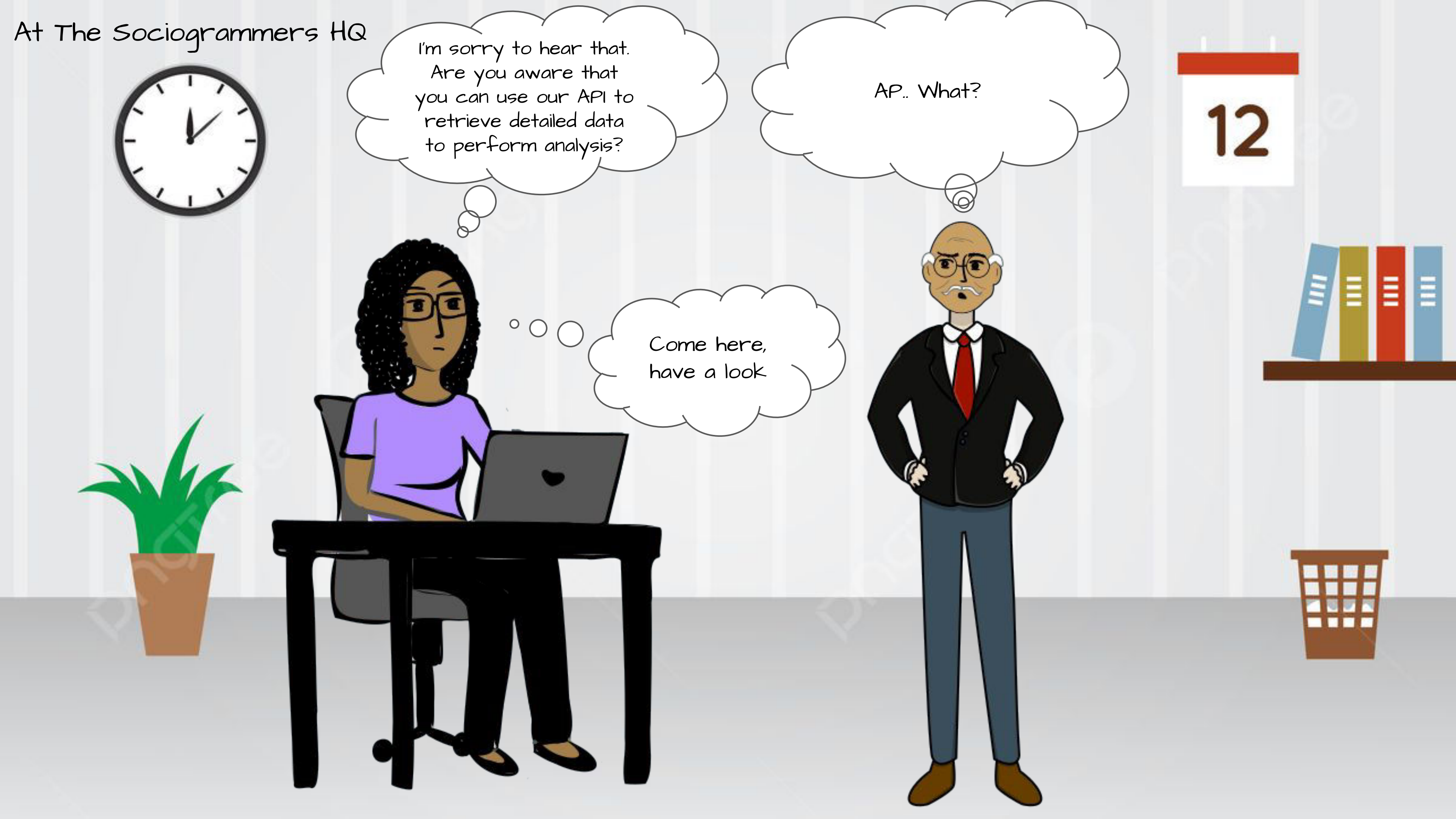



The upvote and automatic question suggestion features are definitely nifty
Yes indeed! The best part is that it reduces clutter for both students and us support staff
Thank you. Bye!
Good luck on your first day Priya. Bye
At IITM BS Degree Support Desk Office

Storyboard 4

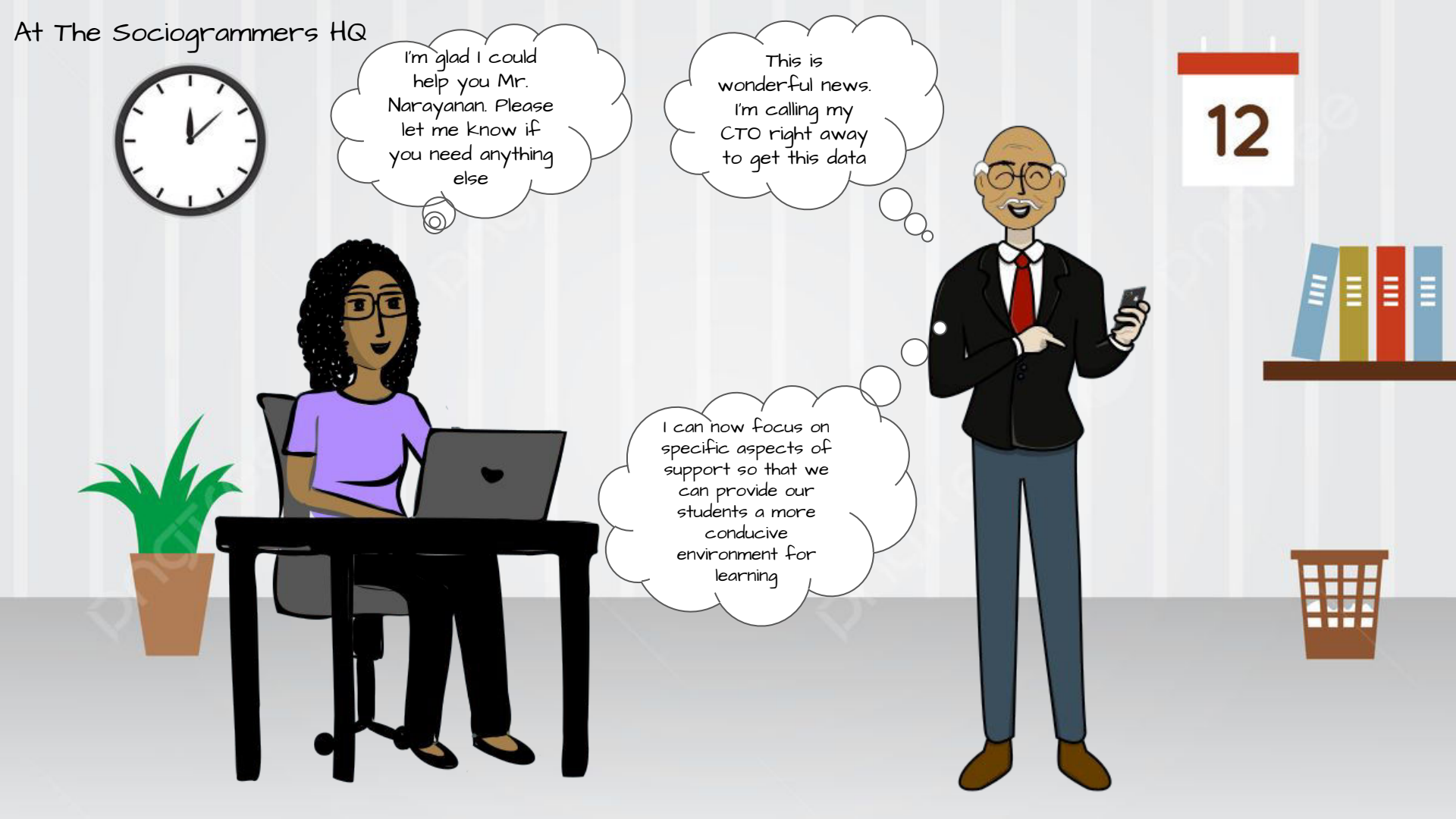
From the perspective of a manager


At The Sociogrammers HQ
Hello Mr. Narayanan! Aren't you the manager at IIT Madras BS Degree Support Desk? You look a bit worried. Is everything okay?
I've been receiving some complaints about the quality of support.
12
I want to analyse and understand the preparedness and efficiency of our various support agents.


At The Sociogrammers HQ
I'm sorry to hear that. Are you aware that you can use our API to retrieve detailed data to perform analysis?
AP.. What?
Come here, have a look

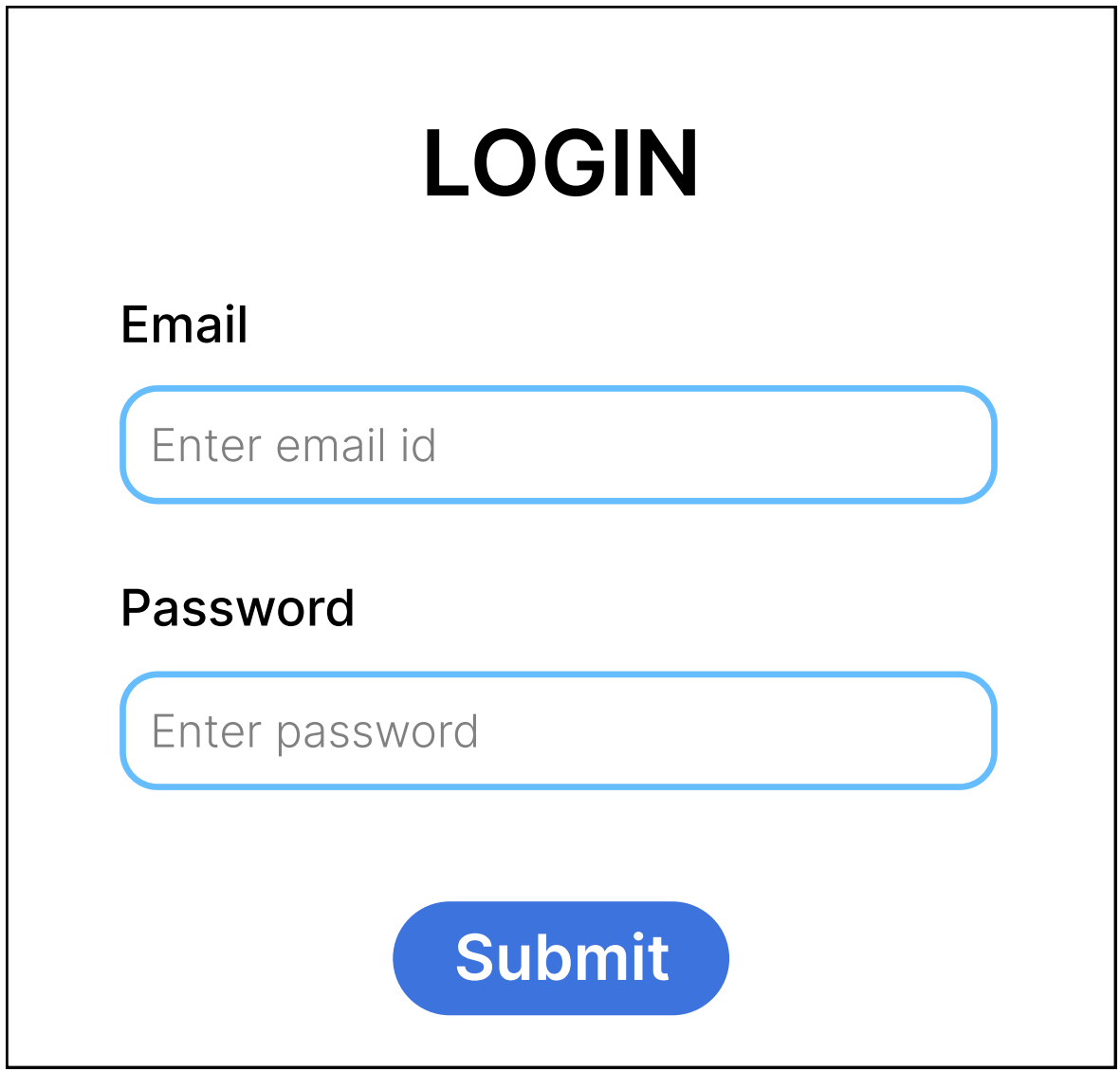
At The Sociogrammers HQ

At The Sociogrammers HQ

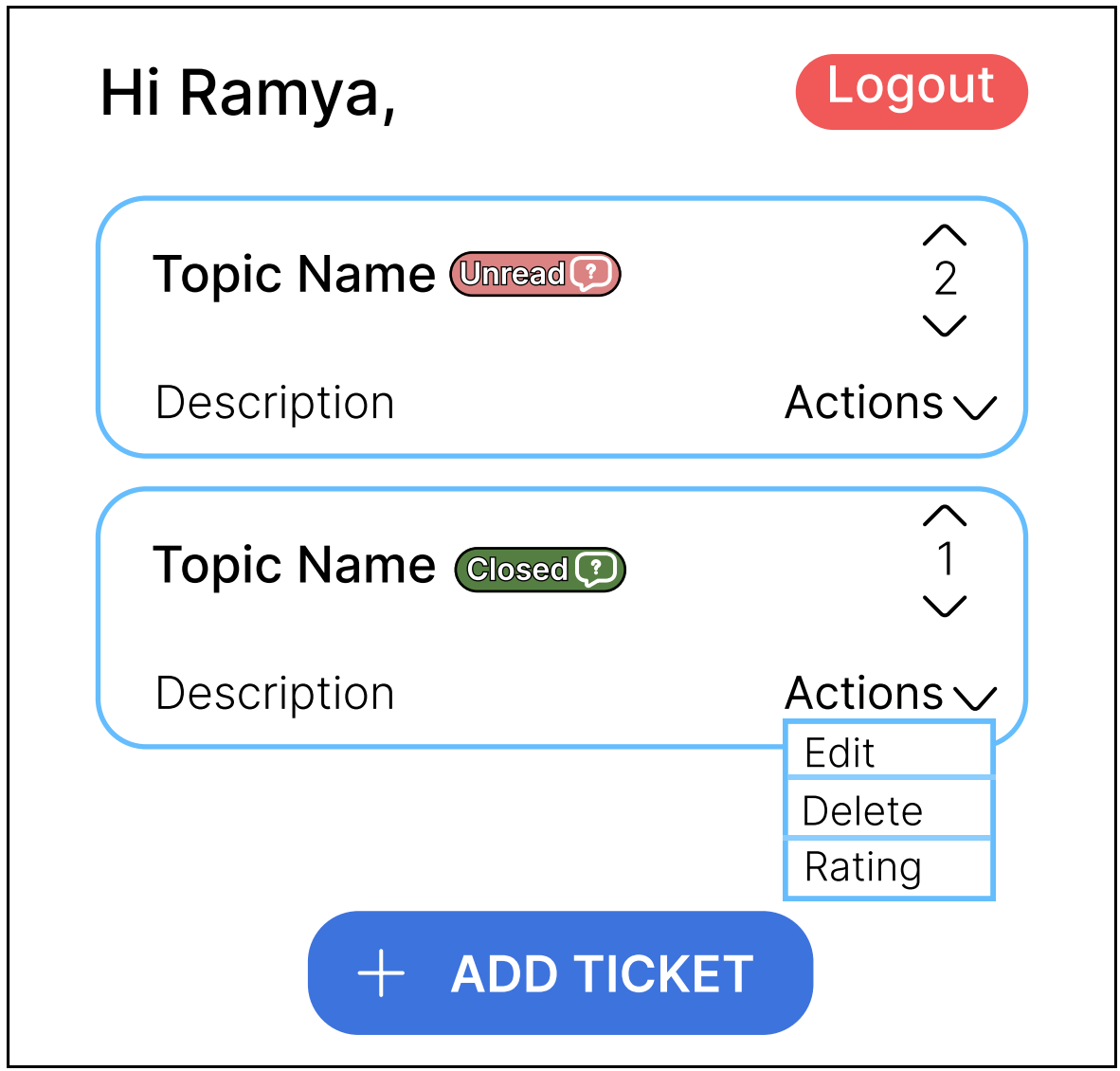


Wireframe

LOGIN PAGE



Student's View



Student’s View while posting a new query

If student types
a query similar to
existing queries

Hi Ramya,

Logout

Topic Name

Add Description

Submit

These queries match well with your query. Consider up voting them if relevant, instead of creating a new query!

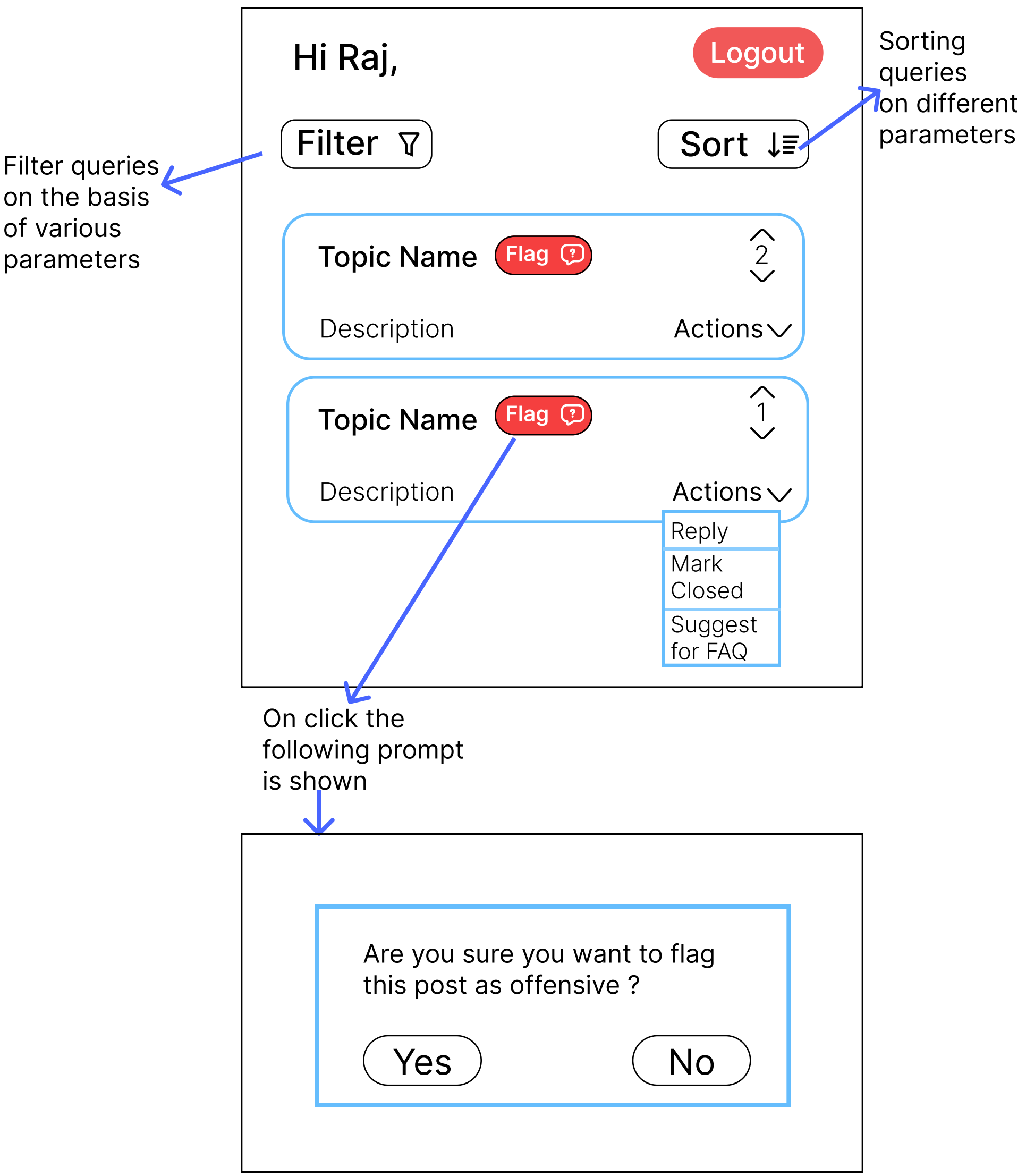
Id Card Error

Id Card picture error ^ 12

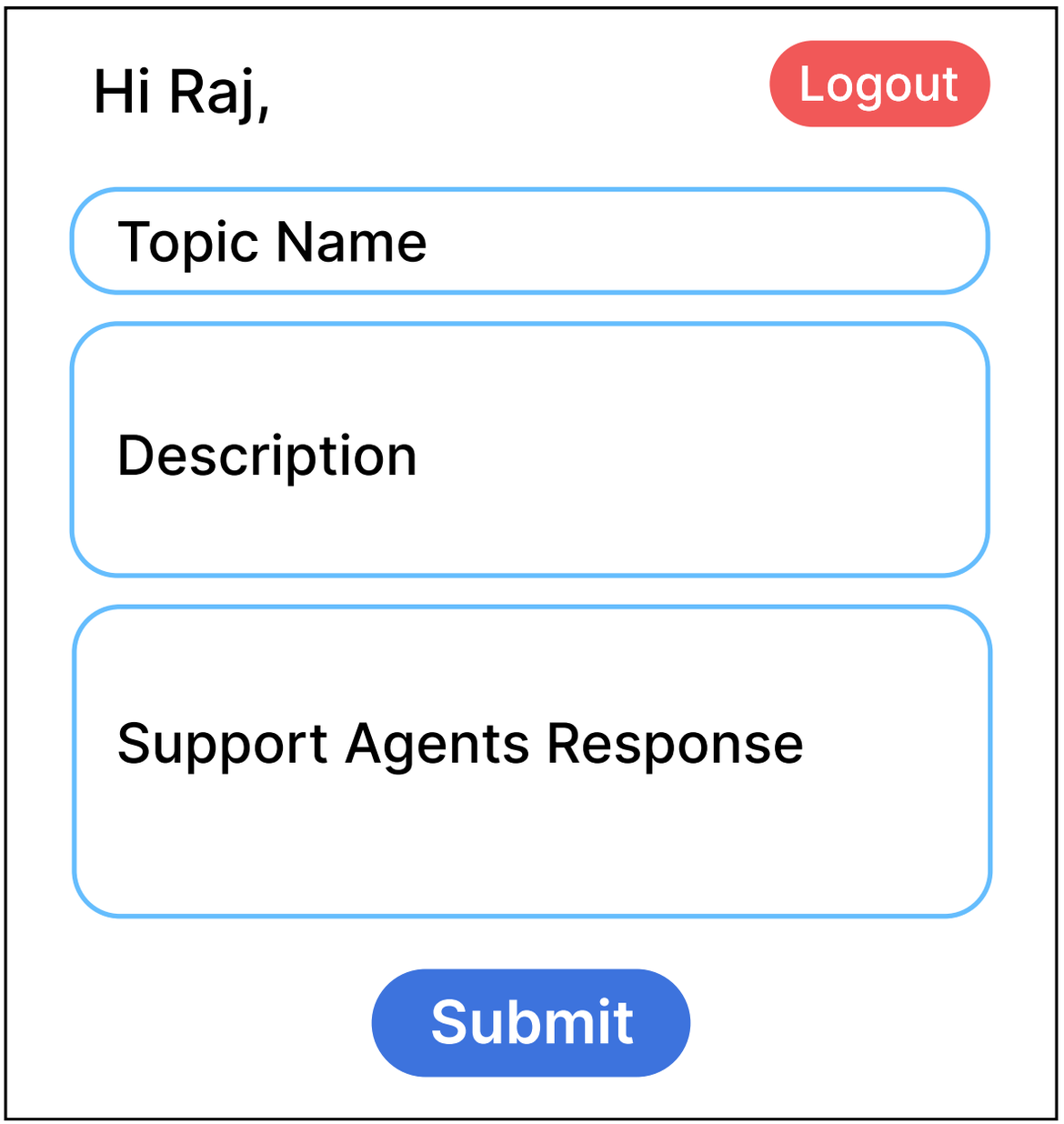
Wrong Name on Id Card ^ 10

Id Card Address Change ^ 8

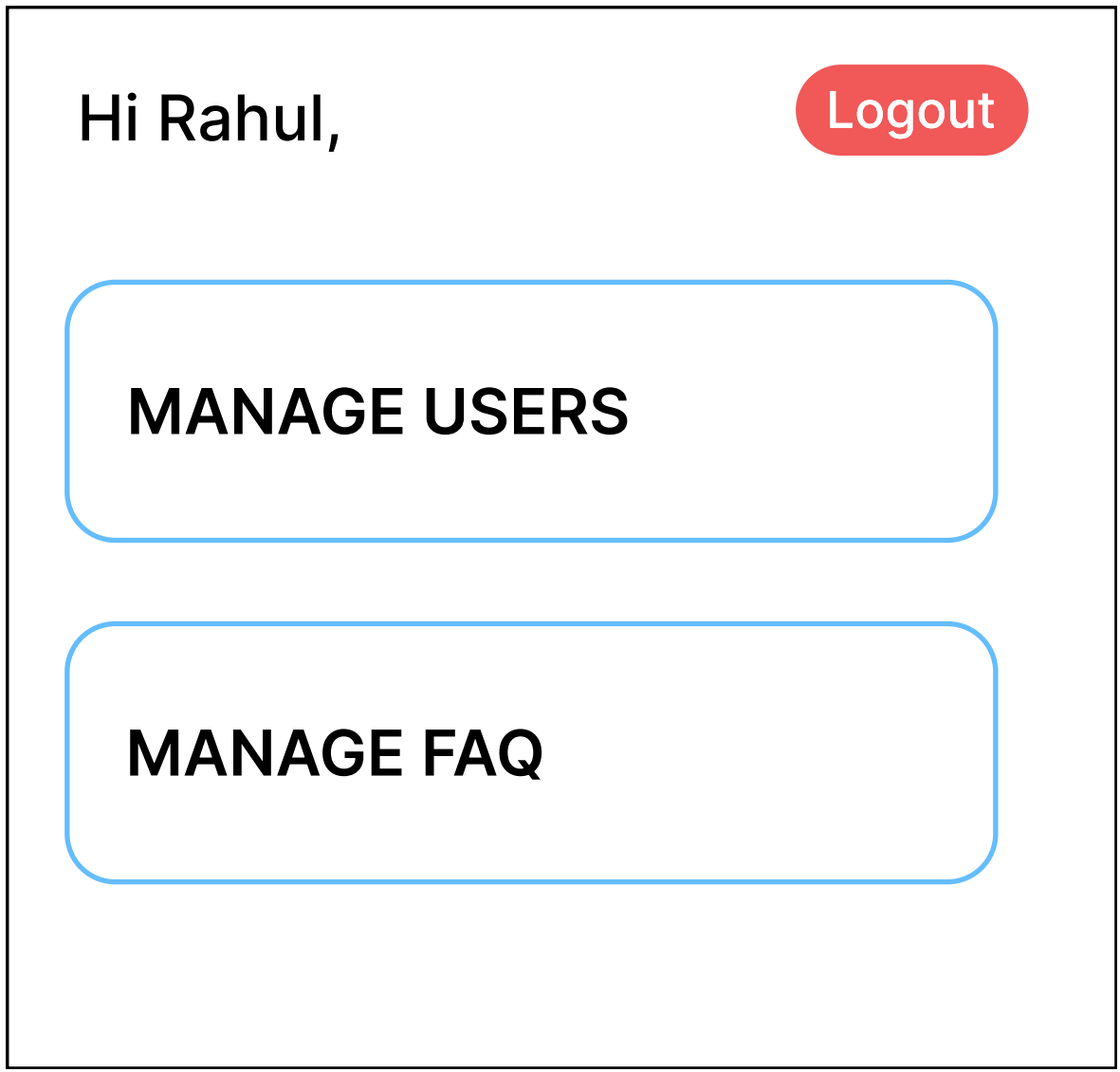
Support Agent View



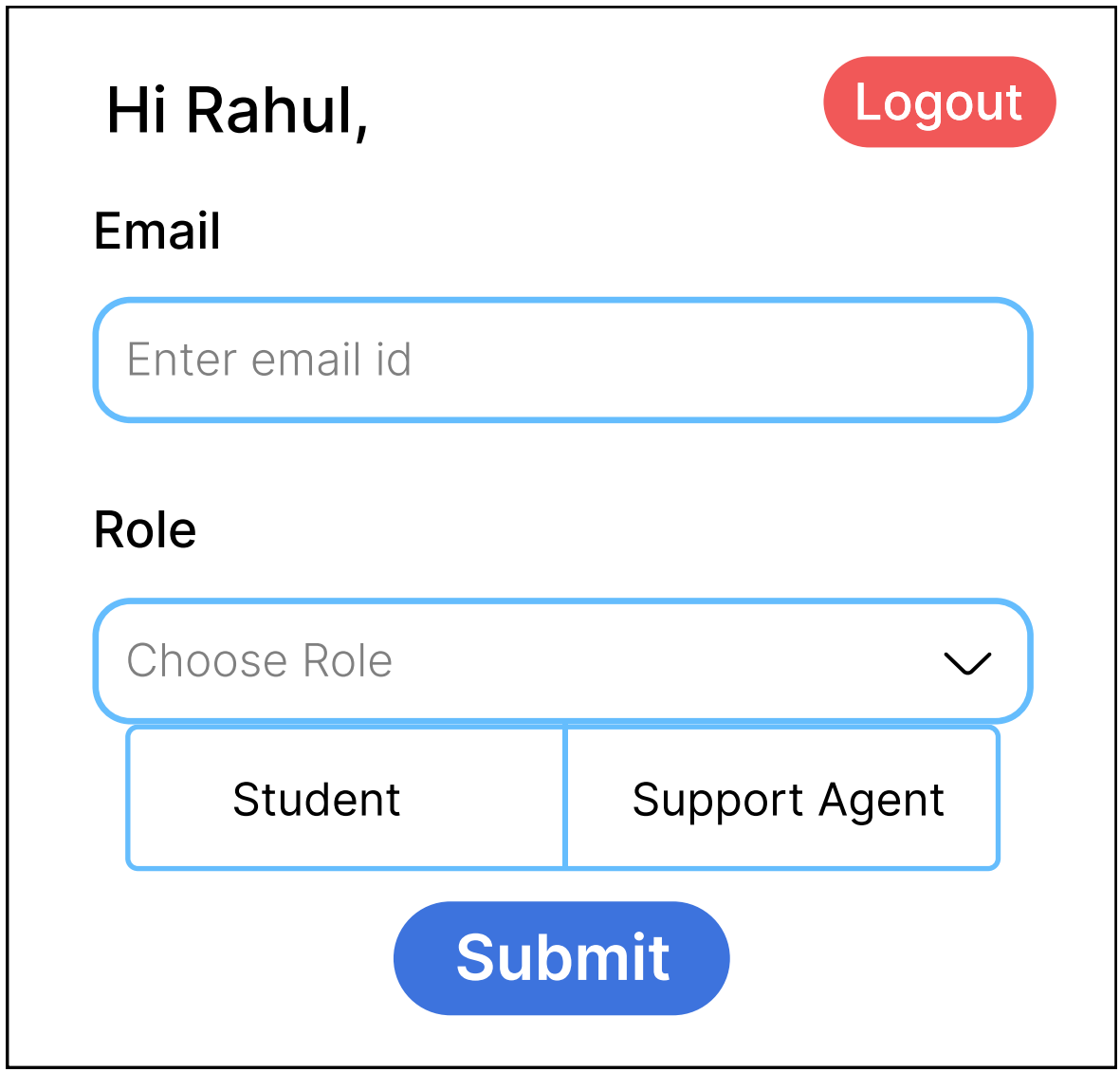
Support Agent’s Reply View



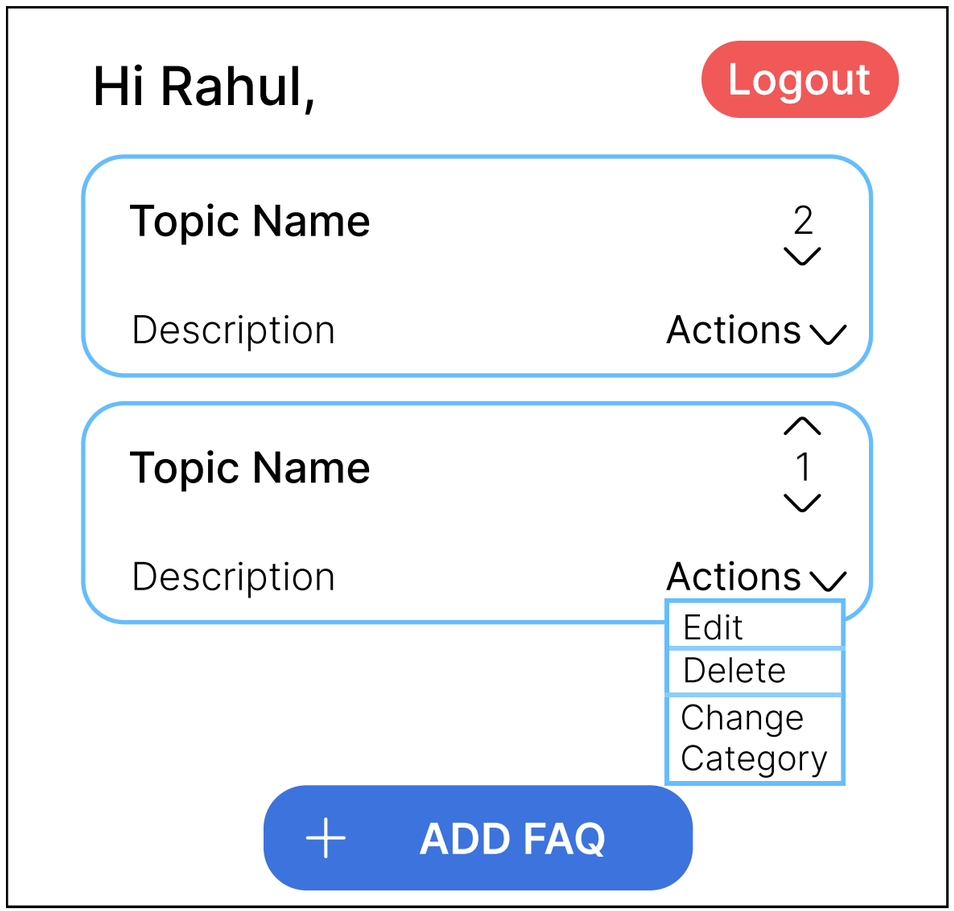
Admin’s View

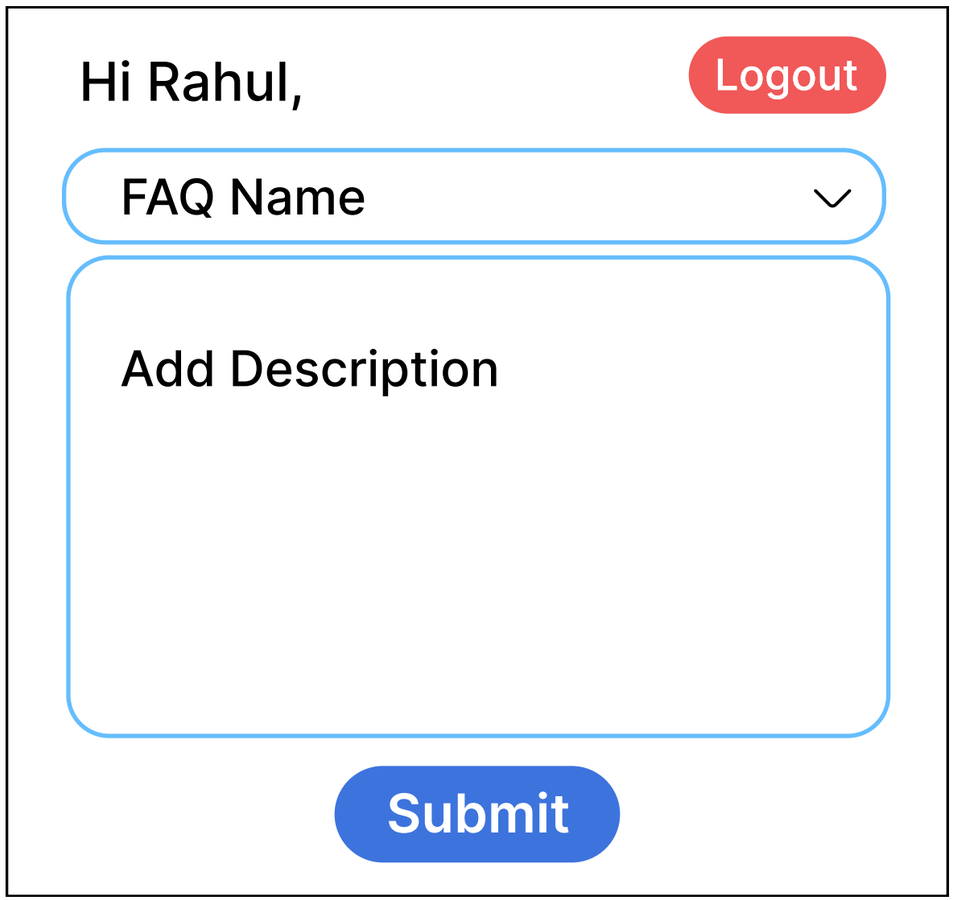


Admin’s View for adding Users



Admin’s view for FAQ





SOFTWARE ENGINEERING MILESTONE 3

SUBMITTED IN THE PARTIAL FULFILLMENT OF THE REQUIREMENTS OF THE COURSE:

BSCSS3001: Software Engineering

By:
Arya Bhattacharyya (21f2000436)
Varun Venkatesh (21f1000743)
Chirag Goel (21f2000540)

INDIAN INSTITUTE OF TECHNOLOGY, MADRAS
March, 2023

Design of Components

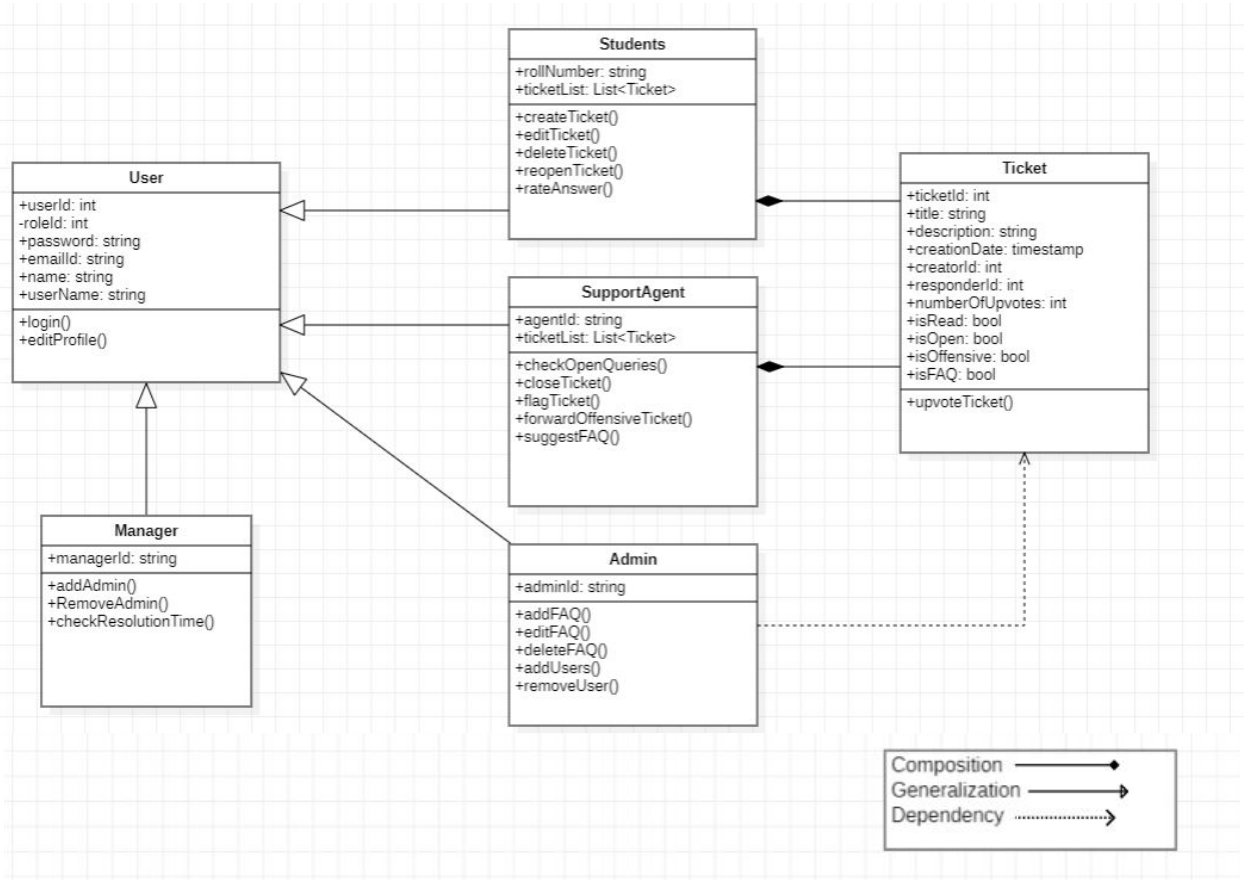
- **Student View(APIs/Export Jobs)**
    - Create a new ticket
    - Edit an existing ticket (Or upvote an existing one, rate the resolution if already responded to, reopen a closed ticket)
    - Delete an existing ticket
    - Student Dashboard (Read their tickets)
    - Similar queries must be retrieved from the database
    - Celery Task for Notification for query response (within 10 mins)

- **Support Agent View(APIs/Export Jobs)**
    - Retrieve active queries from the backend (Sorting can be taken care of in the frontend,filter tickets based on username to be done on frontend)
    - Edit an existing ticket (Mark as closed/resolved, mark a particular ticket to be added to FAQ section, flag a post as rude/offensive, forward offensive post to admin)
    - Celery task for notification on query resolution (Notification should reach to upvoted students as well)

- **Admin View**
    - Dynamically Update (CRUD) the FAQ section
    - CRUD for user management (Enroll new people using their email IDs, delete user by username)

- **Manager View**
    - CRUD on admins
    - Celery Task (Cron Job) to check for queries unanswered for 72 hours and send notification.
    - Celery Task (Cron Job) to list support agents whose average resolution time is more than 48 hours in the last 30 days (monthly cron job mainly)
    - Resolution times of different queries.

Class Diagram



Sprint Schedule

- **Sprint 1:** Identify the types of users
  - Date: 06/02/2023 - 10/02/2023
- **Sprint 2:** SMART User Stories
  - Date: 11/02/2023 - 16/02/2023
- **Sprint 3:** Vetting & Submission
  - Date: 17/02/2023 - 19/02/2023
- **Sprint 4:** Storyboarding, Wireframe, Applying Usability Principles to Wireframe, Vetting Submission, Final Submission
  - Date: 20/02/2023 - 26/02/2023
- **Sprint 5:** Jira Roadmap Setup, Design of Components, UML Diagrams, Vetting Submission, Final Submission
  - Date: 27/02/2023 - 05/03/2023
- **Sprint 6:** Database Schema, Database Models, User Class, Students Class (Code, Debug, Raise Issues & Code Review)
  - Date: 06/03/2023 - 12/03/2023
- **Sprint 7:** Ticket Class, SupportAgent Class, Admin Class, Manager Class (Code, Debug, Raise Issues & Code Review), OpenAPI 3.0 YAML for all endpoints
  - Date: 13/03/2023 - 19/03/2023
- **Sprint 8:** Design and Describe 15 test cases and perform unit testing in pytest
  - Date: 20/03/2023 - 26/03/2023
- **Sprint 9:** Design and Describe the remaining test cases and perform unit testing in pytest, Frontend Login and Signup Views
  - Date: 27/03/2023 - 02/04/2023
- **Sprint 10:** Finish Frontend, Integrate Frontend with Backend, Detailed report and record presentation
  - Date: 03/04/2023 - 16/04/202

SCRUM Meetings Schedule and Minutes

SCRUM Meetings: Every Monday, Wednesday, and Friday 19:00-20:30

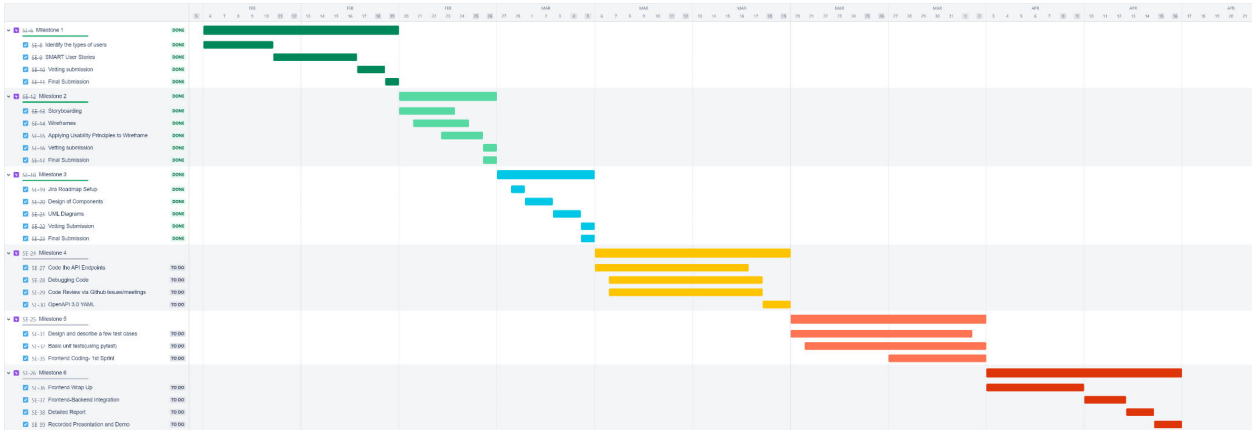
Minutes from Sprint 1 SCRUM Meetings:
Identified and discussed the different types of users (primary, secondary, and tertiary). Discussed a few User Stories associated with the aforementioned users and agreed to come up with at least 10 user stories each.

Minutes from Sprint 2 SCRUM Meetings:
Discussed our User Stories and applied SMART Guidelines to refine the User Stories.

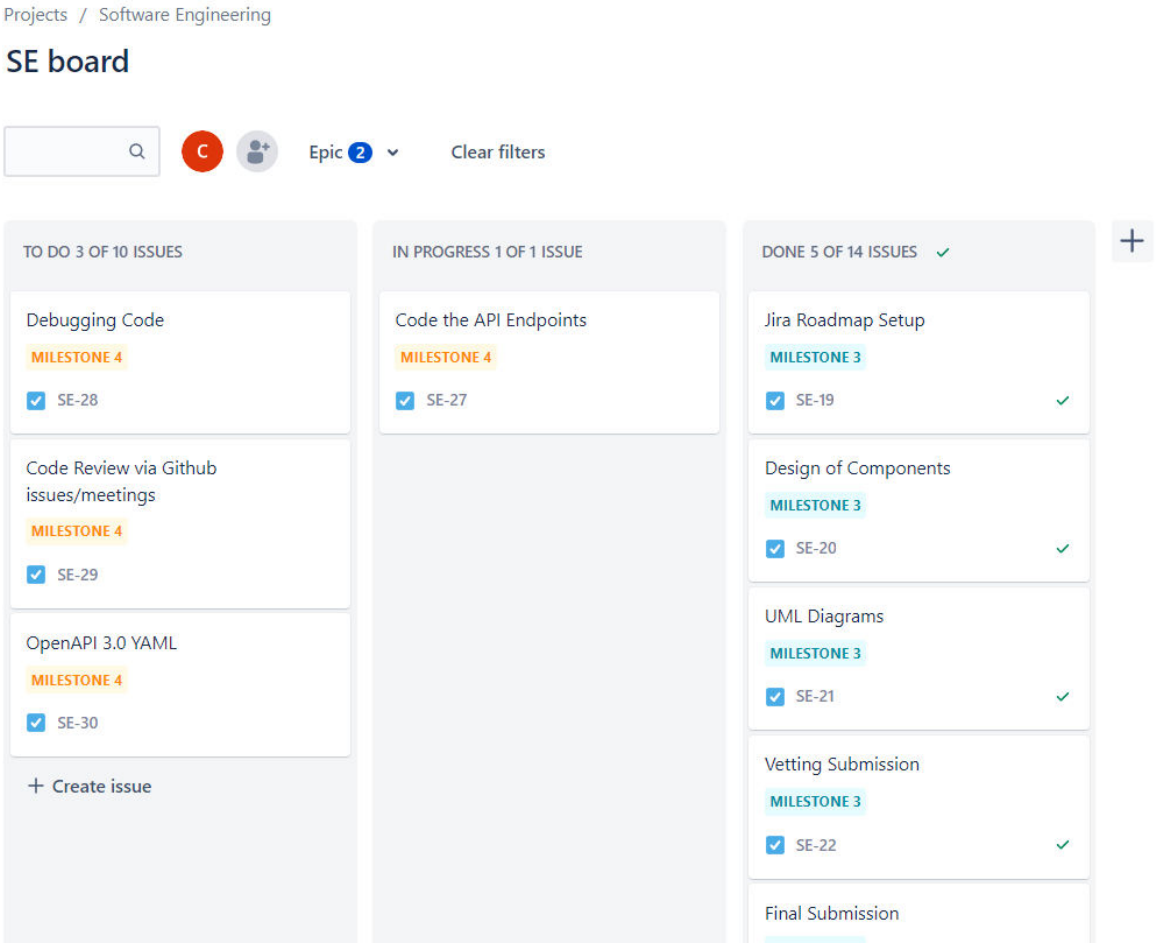
Minutes from Sprint 4 SCRUM Meetings:
Discussed the software to design the storyboards and wireframes. Also worked on the initial draft of our wireframe. Chirag was tasked to mainly refine the wireframe; Varun and Arya were tasked to come up with a few storylines for our storyboards to discuss in our next SCRUM meeting**.** Discussed our storylines for our storyboards and agreed on a few to take further. Applied design heuristics and found our present software inadequate to express the same. Thus decided to switch to a different software to design our wireframe.

Minutes from Sprint 5 SCRUM Meetings:
Came up with a schedule for our project and an appropriate project scheduling tool, which was decided to be **Jira**. We then designed the components on pen and paper and finally used StarUML to produce the digital version of the same.

GANTT Chart



(Partial) Kanban Board



SOFTWARE ENGINEERING MILESTONE 5

SUBMITTED IN THE PARTIAL FULFILLMENT OF THE REQUIREMENTS OF THE COURSE:

BSCSS3001: Software Engineering

By:
Arya Bhattacharyya (21f2000436)
Varun Venkatesh (21f1000743)
Chirag Goel (21f2000540)

INDIAN INSTITUTE OF TECHNOLOGY, MADRAS
April, 2023

A new item suggested by a support agent is approved/rejected by the admin for FAQ

Page being tested: http://127.0.0.1:5000/api/faq
Inputs:
- Request Method: POST
- JSON: { "category": "operational", "is_approved": false, "ticket_id": 2}
- Header: secret_authtoken: abcxyz

Expected Output:
- HTTP Status Code: 200
- JSON: {“message”: “FAQ item added successfully”}

Actual Output:
- HTTP Status Code: 200
- JSON: {“message”: “FAQ item added successfully”}

Result: Success

```python
def test_faq_authorized_role_post_valid_data():
    input_dict = { "category": "operational","is_approved": False, "ticket_id": 2}
    data = json.dumps(input_dict)
    header={"secret_authtoken":token_login_admin(), "Content-Type":"application/json"}
    request=requests.post(url_faq,data=data, headers=header)
    assert request.status_code==200
    assert request.json()['message']=="FAQ item added successfully"
    faq = FAQ.query.filter_by(ticket_id=2).first()
    assert input_dict["category"] == faq.category
    assert input_dict["is_approved"] == faq.is_approved
```

A new item suggested by a support agent is approved/rejected by the admin for FAQ but the request body has invalid data

Page being tested: http://127.0.0.1:5000/api/faq
Inputs:
- Request Method: POST
- Json body: { "category": "operational", "is_approved": “abcd”, "ticket_id": 2}
- Header: secret_authtoken: abcxyz

Expected Output:
- HTTP Status Code: 400
- JSON: {“message”: “is_approved must be boolean”}

Actual Output:
- HTTP Status Code: 400
- JSON: {“message”: “is_approved must be boolean”}

Result: Success

```
def test_faq_authorized_role_post_invalid_isapproved():
    input_dict = { "category": "operational","is_approved": "abs", "ticket_id": 2}
    data = json.dumps(input_dict)
    header={"secret_authtoken":token_login_admin(), "Content-Type":"application/json"}
    request=requests.post(url_faq,data=data, headers=header)
    assert request.status_code==400
    assert request.json()['message']=="is_approved must be boolean"
    assert FAQ.query.filter_by(ticket_id=input_dict["ticket_id"]).first() is None
```

An existing ticket in the FAQ is updated with a new category

Page being tested: http://127.0.0.1:5000/api/faq
Inputs:
- Request Method: PATCH
- Json body: { "category": "random","is_approved": false, "ticket_id": 2}
- Header: secret_authtoken: abcxyz

Expected Output:
- HTTP Status Code: 200
- JSON: {“message”: “FAQ item updated successfully”}

Actual Output:
- HTTP Status Code: 200
- JSON: {“message”: “FAQ item updated successfully”}

Result: Success

```
def test_faq_authorized_role_patch_valid_data():
    input_dict = { "category": "random","is_approved": False, "ticket_id": 1}
    data = json.dumps(input_dict)
    header={"secret_authtoken":token_login_admin(), "Content-Type":"application/json"}
    request=requests.patch(url_faq,data=data, headers=header)
    assert request.status_code==200
    assert request.json()['message']=="FAQ item updated successfully"
    faq = FAQ.query.filter_by(ticket_id=1).first()
    assert input_dict["category"] == faq.category
    assert input_dict["is_approved"] == faq.is_approved
```

A request is sent to to update a non-existing ticket in the FAQ

Page being tested: http://127.0.0.1:5000/api/faq
Inputs:
- Request Method: PATCH
- Json body: { "category": "random","is_approved": false, "ticket_id": 1000}
- Header: secret_authtoken: abcxyz

Expected Output:
- HTTP Status Code: 400
- JSON: {“message”: “ticket_id does not exist”}

Actual Output:
- HTTP Status Code: 400
- JSON: {“message”: “ticket_id does not exist”}

Result: Success

```python
def test_faq_authorized_role_patch_nonexistant_ticket_id():
    data = json.dumps({ "category": "operational","is_approved": False, "ticket_id": 10000})
    header={"secret_authtoken":token_login_admin(), "Content-Type":"application/json"}
    request=requests.patch(url_faq,data=data, headers=header)
    assert request.status_code==400
    assert request.json()['message']=="ticket_id does not exist"
    assert not FAQ.query.filter_by(ticket_id=10000).first()
```

A ticket is removed from the FAQ Table

Page being tested: http://127.0.0.1:5000/api/faq/2
Inputs:
- Request Method: DELETE
- Header: secret_authtoken: abcxyz

Expected Output:
- HTTP Status Code: 200
- JSON: {“message”: “FAQ item deleted successfully”}

Actual Output:
- HTTP Status Code: 200
- JSON: {“message”: “FAQ item deleted successfully”}

Result: Success

```python
def test_faq_authorized_role_delete_valid():
    header={"secret_authtoken":token_login_admin()}
    request=requests.delete(delete_url, headers=header)
    assert request.status_code==200
    assert request.json()['message']=="FAQ item deleted successfully"
    assert FAQ.query.filter_by(ticket_id=2).first() is None
```

Creating a new ticket by student

Page being tested: http://127.0.0.1:5000/api/ticket
Inputs:
- Request Method: POST
- JSON: { "title":"test1234", "description":"hi", "number_of_upvotes":13, "is_read":0, "is_open":1, "is_offensive":0, "is_FAQ":0 }
- Header: secret_authtoken: abcxyz

Expected Output:
- HTTP Status Code: 200
- JSON: {'message':'Ticket created successfully'}

Actual Output:
- HTTP Status Code: 200
- JSON: {'message':'Ticket created successfully'}

Result: Success

```python
def test_ticket_student_post():
    header={"secret_authtoken":token_login_student(),"Content-Type":"application/json"}
    data={
        "title":"test1234",
        "description":"hi",
        "number_of_upvotes":13,
        "is_read":0,
        "is_open":1,
        "is_offensive":0,
        "is_FAQ":0
        }
    data=json.dumps(data)
    response=requests.post(url_ticket,data=data,headers=header)
    assert response.status_code==200
    response_get=requests.get(url_ticket,headers=header)
    response_get=response_get.json()
    response_get=response_get['data']
    for i in response_get:
        if(i["title"]=="test1234"):
            assert i["description"]=="hi"
            assert i["number_of_upvotes"]==13
            assert i["is_read"]==0
            assert i["is_open"]==1
            assert i["is_offensive"]==0
            assert i["is_FAQ"]==0
```

Editing a ticket by student

Page being tested: http://127.0.0.1:5000/api/ticket
Inputs:

- Request Method: PATCH
- JSON: { "ticket_id":3,"title":"test" }
- Header: secret_authtoken: abcxyz

Expected Output:

- HTTP Status Code: 200
- JSON: {'message':'Ticket updated successfully'}

Actual Output:

- HTTP Status Code: 200
- JSON: {'message':'Ticket updated successfully'}

Result: Success

```python
def test_ticket_title_student_patch():
    header={"secret_authtoken":token_login_student(),"Content-Type":"application/json"}
    payload={
        "ticket_id":3,
        "title":"test",
    }
    payload=json.dumps(payload)
    response=requests.patch(url_ticket,data=payload,headers=header)
    assert response.status_code==200
    response_get=requests.get(url_ticket,headers=header)
    response_get=response_get.json()
    response_get=response_get['data']
    for i in response_get:
        if(i["ticket_id"]==3):
            assert i["title"]=="test"
```

Deleting a ticket by student if resolved

Page being tested: http://127.0.0.1:5000/api/ticket/3
Inputs:
- Request Method: DELETE
- Header: secret_authtoken: abcxyz

Expected Output:
- HTTP Status Code: 200
- JSON: {'message':'Ticket deleted successfully'}

Actual Output:
- HTTP Status Code: 200
- JSON: {'message':'Ticket deleted successfully'}

Result: Success

```python
def test_ticket_student_delete():
    url=url_ticket+"/3"
    header={"secret_authtoken":token_login_student(),"Content-Type":"application/json"}
    response=requests.delete(url,headers=header)
    assert response.status_code==200
    ticket=Ticket.query.filter_by(ticket_id=3).first()
    assert ticket==None
```

Upvoting ticket “+1” existing tickets created by other students

Page being tested: http://127.0.0.1:5000/api/ticketAll
Inputs:
- Request Method: PATCH
- JSON: { "ticket_id":2,"number_of_upvotes":146 }
- Header: secret_authtoken: abcxyz

Expected Output:
- HTTP Status Code: 200
- JSON: {“message”: “success”}

Actual Output:
- HTTP Status Code: 200
- JSON: {“message”: “success”}

Result: Success

```python
def test_ticket_all_number_of_upvotes():
    input_dict = { "number_of_upvotes": 146,"ticket_id": 2}
    data = json.dumps(input_dict)
    header={"secret_authtoken":token_login_admin(), "Content-Type":"application/json"}
    request=requests.patch(url_ticket_all,data=data, headers=header)
    assert request.status_code==200
    assert request.json()['message']=="success"
    ticket = Ticket.query.filter_by(ticket_id=input_dict["ticket_id"]).first()
    assert input_dict["number_of_upvotes"] == ticket.number_of_upvotes
    assert input_dict["is_read"] == ticket.is_read
```

Remove a particular user

Page being tested: http://127.0.0.1:5000/api/user/3
Inputs:
- Request Method: DELETE
- Header: secret_authtoken: abcxyz

Expected Output:
- HTTP Status Code: 200
- JSON: {'message':'User deleted successfully'}

Actual Output:
- HTTP Status Code: 200
- JSON: {'message':'User deleted successfully'}

Result: Success

```python
def test_user_admin_delete():
    url=url_user+"/8"
    header={"secret_authtoken":token_login_admin(),"Content-Type":"application/json"}
    response=requests.delete(url,headers=header)
    assert response.status_code==200
    user=User.query.filter_by(user_id=8).first()
    assert user==None
```

Add a particular user according to role by admin / manager

Page being tested: http://127.0.0.1:5000/api/user
Inputs:
- Request Method: POST
- JSON: {“email_id”:”test@test”,”role_id”:1}
- Header: secret_authtoken: abcxyz

Expected Output:
- HTTP Status Code: 200
- JSON: {'message':'User created successfully'}

Actual Output:
- HTTP Status Code: 200
- JSON: {'message':'User created successfully'}

Result: Success

```python
def test_user_admin_post():
    header={"secret_authtoken":token_login_admin(),"Content-Type":"application/json"}
    data={
        "email_id":"test@test",
        "role_id":1
    }
    data=json.dumps(data)
    response=requests.post(url_user,data=data,headers=header)
    assert response.status_code==200
    response_get=requests.get(url_user,headers=header)
    response_get=response_get.json()
    response_get=response_get['data']
    for i in response_get:
        if(i["email_id"]=="test@test"):
            assert i["role_id"]==1
```

An existing ticket is marked closed/resolved by support agent

Page being tested: http://127.0.0.1:5000/api/ticketAll
Inputs:

- Request Method: PATCH
- Json body: { "is_open": true, "ticket_id": 1}
- Header: secret_authtoken: abcxyz

Expected Output:

- HTTP Status Code: 200
- JSON: {“message”: “success”}

Actual Output:

- HTTP Status Code: 200
- JSON: {“message”: “success”}

Result: Success

```python
def test_ticket_all_patch():
    input_dict = { "is_open": True,"ticket_id": 1}
    data = json.dumps(input_dict)
    header={"secret_authtoken":token_login_support_agent(),
"Content-Type":"application/json"}
    request=requests.patch(url_ticket_all,data=data, headers=header)
    assert request.status_code==200
    assert request.json()['message']=="success"
    ticket =
Ticket.query.filter_by(ticket_id=input_dict["ticket_id"]).first()
    assert input_dict["number_of_upvotes"] == ticket.number_of_upvotes
    assert input_dict["is_read"] == ticket.is_read
```

A non-existing ticket is attempted to be marked closed/resolved by support agent

Page being tested: http://127.0.0.1:5000/api/ticketAll
Inputs:
- Request Method: PATCH
- Json body: { "is_open": true, "ticket_id": 10000}
- Header: secret_authtoken: abcxyz

Expected Output:
- HTTP Status Code: 404
- JSON: {“message”: “There is no such ticket by that ID”}

Actual Output:
- HTTP Status Code: 404
- JSON: {“message”: “There is no such ticket by that ID”}

Result: Success

```
def test_ticket_all_patch_ticket_not_found():
    input_dict = { "is_open": True,"ticket_id": 10000}
    data = json.dumps(input_dict)
    header={"secret_authtoken":token_login_support_agent(),
"Content-Type":"application/json"}
    request=requests.patch(url_ticket_all,data=data, headers=header)
    assert request.status_code==404
    assert request.json()['message']=="There is no such ticket by that ID"
```

An existing ticket is suggested for FAQ by support agent

Page being tested: http://127.0.0.1:5000/api/ticketAll
Inputs:
- Request Method: PATCH
- Json body: { "is_FAQ": true, "ticket_id": 1}
- Header: secret_authtoken: abcxyz

Expected Output:
- HTTP Status Code: 200
- JSON: {“message”: “success”}

Actual Output:
- HTTP Status Code: 200
- JSON: {“message”: “success”}

Result: Success

```
def test_ticket_all_patch():
    input_dict = { "is_FAQ": True,"ticket_id": 1}
    data = json.dumps(input_dict)
    header={"secret_authtoken":token_login_support_agent(),
"Content-Type":"application/json"}
    request=requests.patch(url_ticket_all,data=data, headers=header)
    assert request.status_code==200
    assert request.json()['message']=="success"
```

```
    ticket =
Ticket.query.filter_by(ticket_id=input_dict["ticket_id"]).first()
    assert input_dict["number_of_upvotes"] == ticket.number_of_upvotes
    assert input_dict["is_read"] == ticket.is_read
```

A non-existing ticket is attempted to be suggested for FAQ by support agent

Page being tested: http://127.0.0.1:5000/api/ticketAll
Inputs:
- Request Method: PATCH
- Json body: { "is_FAQ": true, "ticket_id": 10000}
- Header: secret_authtoken: abcxyz

Expected Output:
- HTTP Status Code: 404
- JSON: {“message”: “There is no such ticket by that ID”}

Actual Output:
- HTTP Status Code: 404
- JSON: {“message”: “There is no such ticket by that ID”}

Result: Success

```
def test_ticket_all_patch_ticket_not_found():
    input_dict = { "is_FAQ": True,"ticket_id": 10000}
    data = json.dumps(input_dict)
    header={"secret_authtoken":token_login_support_agent(),
"Content-Type":"application/json"}
    request=requests.patch(url_ticket_all,data=data, headers=header)
    assert request.status_code==404
    assert request.json()['message']=="There is no such ticket by that ID"
```

An existing ticket is marked as offensive by support agent

Page being tested: http://127.0.0.1:5000/api/ticketAll
Inputs:
- Request Method: PATCH
- Json body: { "is_offensive": true, "ticket_id": 1}
- Header: secret_authtoken: abcxyz

Expected Output:
- HTTP Status Code: 200
- JSON: {“message”: “success”}

Actual Output:
- HTTP Status Code: 200
- JSON: {“message”: “success”}

Result: Success

```
def test_ticket_all_patch():
    input_dict = { "is_offensive": True,"ticket_id": 1}
    data = json.dumps(input_dict)
```

```
    header={"secret_authtoken":token_login_support_agent(),
"Content-Type":"application/json"}
    request=requests.patch(url_ticket_all,data=data, headers=header)
    assert request.status_code==200
    assert request.json()['message']=="success"
    ticket =
Ticket.query.filter_by(ticket_id=input_dict["ticket_id"]).first()
    assert input_dict["number_of_upvotes"] == ticket.number_of_upvotes
    assert input_dict["is_read"] == ticket.is_read
```

A non-existing ticket is attempted to be marked as offensive by support agent

Page being tested: http://127.0.0.1:5000/api/ticketAll
Page being tested: http://127.0.0.1:5000/api/ticketAll
Inputs:

- Request Method: PATCH
- Json body: { "is_offensive": true, "ticket_id": 10000}
- Header: secret_authtoken: abcxyz

Expected Output:

- HTTP Status Code: 404
- JSON: {“message”: “There is no such ticket by that ID”}

Actual Output:

- HTTP Status Code: 404
- JSON: {“message”: “There is no such ticket by that ID”}

Result: Success

```
def test_ticket_all_patch_ticket_not_found():
    input_dict = { "is_offensive": True,"ticket_id": 10000}
    data = json.dumps(input_dict)
    header={"secret_authtoken":token_login_support_agent(),
"Content-Type":"application/json"}
    request=requests.patch(url_ticket_all,data=data, headers=header)
    assert request.status_code==404
    assert request.json()['message']=="There is no such ticket by that ID"
```

An existing ticket already marked as offensive is forwarded to admin by support agent

Page being tested: http://127.0.0.1:5000/api/flaggedPosts
Inputs:

- Request Method: POST
- Json body: { "creator_id": 1, "ticket_id": 1, “flagger_id”: 2}
- Header: secret_authtoken: abcxyz

Expected Output:
- HTTP Status Code: 200
- JSON: {“status”: “success”}

Actual Output:
- HTTP Status Code: 200
- JSON: {“status”: “success”}

Result: Success

```
def test_post_flaggedPost():
    header={"secret_authtoken":token_login_support_agent(),
"Content-Type":"application/json"}
    input_dict = { "ticket_id": 1, "creator_id": 1,"flagger_id": 2 }
    data = json.dumps(input_dict)
    request=requests.post(url = url_flaggedPosts, headers=header, data =
data)
    response = request.json()
    assert request.status_code == 200
    assert response["status"] == "success"
    header2 = {"secret_authtoken":token_login_admin(),
"Content-Type":"application/json"}
    request2 = requests.get(url = url_flaggedPosts, headers=header2)
    response2 = request2.json()
    for item in response2["data"]:
        if item["ticket_id"] == input_dict["ticket_id"]:
            assert item["creator_id"] == input_dict["creator_id"]
            assert item["flagger_id"] == input_dict["flagger_id"]
```

A non existing ticket not already marked as offensive is tried to be forwarded to admin by support agent

Page being tested: http://127.0.0.1:5000/api/flaggedPosts
Inputs:
- Request Method: POST
- Json body: { "creator_id": 1, "ticket_id": 10000, “flagger_id”: 2}
- Header: secret_authtoken: abcxyz

Expected Output:
- HTTP Status Code: 403
- JSON: {“message”: “The referenced ticket is not created by the referenced person/the ticket doesn’t exist in the first place.”}

Actual Output:
- HTTP Status Code: 403
- JSON: {“message”: “The referenced ticket is not created by the referenced person/the ticket doesn’t exist in the first place.”}

Result: Success

```
def test_post_flaggedPost_wrong_ticket_id():
    header={"secret_authtoken":token_login_support_agent(),
"Content-Type":"application/json"}
    input_dict = { "ticket_id": 10000, "creator_id": 1,"flagger_id": 2 }
    data = json.dumps(input_dict)
    request=requests.post(url = url_flaggedPosts, headers=header, data =
data)
    response = request.json()
    assert request.status_code == 403
    assert response["message"] == "The referenced ticket is not created by
the referenced person/ the ticket doesn't exist in the first place."
```

Manager obtaining resolution times of existing tickets

Page being tested: http://127.0.0.1:5000/api/getResolutionTimes
Inputs:

- Request Method: POST
- Json body: { "ticket_id": 2}
- Header: secret_authtoken: abcxyz

Expected Output:

- HTTP Status Code: 200
- JSON: {
  "data": [
  {
  "creation_time": "Fri, 10 Mar 2023 06:36:58 GMT",
  "days": 2,
  "microseconds": 583678,
  "resolution_time_datetime_format": "2 days, 21:40:12.583678",
  "response_time": "Fri, 10 Mar 2023 06:36:58 GMT",
  "seconds": 78012,
  "ticket_id": 2
  }
  ],
  "status": "success"
  }

Actual Output:

- HTTP Status Code: 200
- JSON: {
  "data": [
  {
  "creation_time": "Fri, 10 Mar 2023 06:36:58 GMT",
  "days": 2,
  "microseconds": 583678,
  "resolution_time_datetime_format": "2 days, 21:40:12.583678",
  "response_time": "Fri, 10 Mar 2023 06:36:58 GMT",

```
            "seconds": 78012,
            "ticket_id": 2
          }
        ],
        "status": "success"
      }
```


-

Result: Success

```
def test_getResolutionTimes_post():
    #Only checks if days, seconds, microseconds and ticket IDs match
    header={"secret_authtoken":token_login_manager(),
"Content-Type":"application/json"}
    input_dict = {"ticket_id": 2}
    data = json.dumps(input_dict)
    request=requests.post(url = url_getResolutionTimes,data = data,
headers=header)
    response = request.json()
    assert request.status_code == 200
    if isinstance(input_dict["ticket_id"], int):
        responses = Response.query.filter_by(ticket_id =
input_dict["ticket_id"]).all()
        responses = list(responses)
        ticket = Ticket.query.filter_by(ticket_id =
input_dict["ticket_id"]).first()
        a = {}
        response_times = []
        for thing in responses:
            if isinstance(thing.response_timestamp, datetime):
                #print("Here 1")
                response_times.append(thing.response_timestamp)
            elif isinstance(thing.response_timestamp, str):
                #print("Here 2")

response_times.append(datetime.strptime(thing.response_timestamp,'%Y-%m-%d
%H:%M:%S.%f'))
            response_time = max(response_times)
            a["creation_time"] = None
            if isinstance(ticket.creation_date, str):
                a["creation_time"] =
datetime.strptime(ticket.creation_date, '%Y-%m-%d %H:%M:%S.%f')
            elif isinstance(ticket.creation_date, datetime):
                a["creation_time"] = ticket.creation_date
```

```
            a["response_time"] = response_time
            a["resolution_time_datetime_format"] = a["response_time"] -
a["creation_time"]
            a["days"] = a["resolution_time_datetime_format"].days
            a["seconds"] = a["resolution_time_datetime_format"].seconds
            a["microseconds"] =
a["resolution_time_datetime_format"].microseconds
            a["resolution_time_datetime_format"] =
str(a["resolution_time_datetime_format"])
            a["creation_time"] = a["creation_time"]
            a["ticket_id"] = input_dict["ticket_id"]
            a["response_time"] = None
            a["resolution_time_datetime_format"] = None
            a["creation_time"] = None
        d = response["data"]
        for keys in a:
            if a[keys] is not None:
                assert a[keys] == d[keys]
    elif isinstance(input_dict["ticket_id"], list):
        data = []
        for item in input_dict["ticket_id"]:
            d = {}
            ticket = None
            ticket = Ticket.query.filter_by(ticket_id = item).first()
            if ticket is None:
                continue
            if isinstance(ticket.creation_date, str):
                d["creation_time"] =
datetime.strptime(ticket.creation_date, '%Y-%m-%d %H:%M:%S.%f')
            elif isinstance(ticket.creation_date, datetime):
                d["creation_time"] = ticket.creation_date
            responses = Response.query.filter_by(ticket_id = item).all()
            if ticket.is_open == False:
                responses = list(responses)
                response_times = []
                for thing in responses:
                    if isinstance(thing.response_timestamp, datetime):
                        response_times.append(thing.response_timestamp)
                    elif isinstance(thing.response_timestamp, str):
                        #print("Here 2")
```

```
response_times.append(datetime.strptime(thing.response_timestamp,'%Y-%m-%d
%H:%M:%S.%f'))

                response_time = max(response_times)
                d["response_time"] = response_time
                d["resolution_time_datetime_format"] = d["response_time"]
- d["creation_time"]
                d["days"] = d["resolution_time_datetime_format"].days
                d["seconds"] =
d["resolution_time_datetime_format"].seconds
                d["microseconds"] =
d["resolution_time_datetime_format"].microseconds
                d["response_time"] = d["response_time"]
                d["resolution_time_datetime_format"] =
str(d["resolution_time_datetime_format"])
                d["creation_time"] = d["creation_time"]
                d["ticket_id"] = item
                d["response_time"] = None
                d["resolution_time_datetime_format"] = None
                d["creation_time"] = None
                data.append(d)
        x = response["data"]
        for item in x:
            for thing in data:
                if item["ticket_id"] == thing["ticket_id"]:
                    for keys in thing:
                        if thing[keys] is not None:
                            assert thing[keys] == item[keys]
```

Manager trying to obtain resolution times of non-existing tickets

Page being tested: http://127.0.0.1:5000/api/getResolutionTimes
Inputs:

- Request Method: POST
- Json body: { "ticket_id": 1000}
- Header: secret_authtoken: abcxyz

Expected Output:

- HTTP Status Code: 404
- JSON: {“message”: “No such ticket exists by the given ticket ID.”}

Actual Output:

- HTTP Status Code: 404
- JSON: {“message”: “No such ticket exists by the given ticket ID.”}

Result: Success

```
def test_getResolutionTimes_post_wrong_ticket_id():
    header={"secret_authtoken":token_login_manager(),
"Content-Type":"application/json"}
    input_dict = {"ticket_id": 1000}
    data = json.dumps(input_dict)
    request=requests.post(url = url_getResolutionTimes,data = data,
headers=header)
    response = request.json()
    assert request.status_code == 404
    assert response["message"] == "No such ticket exists by the given
ticket ID."
```

Unit Tests on Celery tasks

Email Notification for a new response sent when inputs properly supplied

Page being tested: Not an API Endpoint. This is a celery task triggered internally when a new response is added
Inputs: (ticket_obj = {'title': 'Problems with my ID Card', 'ticket_id': 1, 'creator_id': 1, 'creator_email': 'redding.abba@dollstore.org'}, response_obj = {'responder_id': 2, 'response': 'test response', 'response_id': 17, 'responder_uname': 'chirag'})
Expected Output: 200
Actual Output: 200
Result: Success

```
#All Fields properly defined for Response Notification, whatever error you get will be from
def test_response_notfication_all_okay():
    ticket_obj = {'title': 'Problems with my ID Card', 'ticket_id': 1, 'creator_id': 1, 'creator_email': 'redding.abba@dollstore.org'}
    response_obj = {'responder_id': 2, 'response': 'test response', 'response_id': 17, 'responder_uname': 'chirag'}
    send_notification = chain(response_notification.s(ticket_obj = ticket_obj, response_obj=response_obj), send_email.s()).apply_async()
    assert send_notification.get() == 200
```

Email Notification for a new response not sent when inputs improperly specified

Page being tested: Not an API Endpoint. This is a celery task triggered internally when a new response is added
Inputs: (ticket_obj = {'title': 'Problems with my ID Card', 'ticket_id': 1, 'creator_id': 1}, response_obj = {'responder_id': 2, 'response': 'test response', 'response_id': 17, 'responder_uname': 'chirag'})

Expected Output: KeyError
Actual Output: KeyError
Result: Success

```python
#One Or more keys missing from expected input
def test_response_notification_inadequate_data_passed():
    ticket_obj = {'title': 'Problems with my ID Card', 'ticket_id': 1, 'creator_id': 1,}
    response_obj = {'responder_id': 2, 'response': 'test response', 'response_id': 17, 'responder_uname': 'chirag'}
    send_notification = chain(response_notification.s(ticket_obj = ticket_obj, response_obj=response_obj), send_email.s()).apply_async()
    with pytest.raises(KeyError):
        send_notification.get()
```

Manager is sent an email notification for open tickets created over 72 hours ago that are still unanswered

Page being tested: Not an API Endpoint. This is a celery task triggered internally by cron
Inputs: None (The output of the task is dependent on the state of the db not by any inputs)
Expected Output: Notification Sent
Actual Output: Notification Sent
Result: Success

```python
from datetime import datetime, timedelta
def test_unanswered_tickets_notification_email_sent_three_day_old_ticket_no_support_response(app):
    three_days_ago = datetime.now() - timedelta(hours=72)
    app = app(CeleryTesting)
    new_ticket = Ticket(title='test',
                        description = 'test desc',
                        creation_date = three_days_ago, creator_id=1,
                        number_of_upvotes=0, is_read=False, is_open=True,
                        is_offensive=False, is_FAQ=False)
    db.session.add(new_ticket)
    db.session.commit()
    assert unanswered_ticket_notification() == 'Notification Sent'
```

Manager is not sent an email notification for open tickets created over 72 hours ago that have been responded to by a support agent

Page being tested: Not an API Endpoint. This is a celery task triggered internally by cron
Inputs: None (The output of the task is dependent on the state of the db not by any inputs)
Expected Output: All Tickets Answered
Actual Output: All Tickets Answered
Result: Success

```python
def test_unanswered_tickets_notification_email_not_sent_as_ticket_open_but_response_from_agent(app):
    three_days_ago = datetime.now() - timedelta(hours=72)
    app = app(CeleryTesting)
    new_ticket = Ticket(title='test',
                        description = 'test desc',
                        creation_date = three_days_ago,
                        creator_id=1, number_of_upvotes=0,
                        is_read=False, is_open=True,
                        is_offensive=False, is_FAQ=False)
    new_response = Response(ticket_id=1, response='test', responder_id=2, response_timestamp=datetime.now())
    db.session.add(new_ticket)
    db.session.add(new_response)
    db.session.commit()
    assert unanswered_ticket_notification() == 'All Tickets Answered'
```

Manager is not sent an email notification as all tickets are marked as closed

Page being tested: Not an API Endpoint. This is a celery task triggered internally by cron
Inputs: None (The output of the task is dependent on the state of the db not by any inputs)
Expected Output: No Unresolved Tickets
Actual Output: No Unresolved Tickets
Result: Success

```python
def test_unanswered_tickets_notification_email_not_sent_ticket_is_not_open(app):
    three_days_ago = datetime.now() - timedelta(hours=72)
    app = app(CeleryTesting)
    new_ticket = Ticket(title='test',
                        description = 'test desc',
                        creation_date = three_days_ago,
                        creator_id=1, number_of_upvotes=0,
                        is_read=False, is_open=False,
                        is_offensive=False, is_FAQ=False)
    db.session.add(new_ticket)
    db.session.commit()
    assert unanswered_ticket_notification() == 'No Unresolved Tickets'
```

Manager is sent an email report of all the agents who have a resolution time of over 48 hours for tickets created in the past 30 days

Page being tested: Not an API Endpoint. This is a celery task triggered internally by cron
Inputs: None (The output of the task is dependent on the state of the db not by any inputs)
Expected Output: Email sent with details of agents with poor resolution time
Actual Output: Email sent with details of agents with poor resolution time
Result: Success

```python
def test_poor_resolution_time_email_sent_due_to_poor_performance(app):
    three_days_ago = datetime.now() - timedelta(hours=72)
    app = app(CeleryTesting)
    new_ticket = Ticket(title='test',
                        description = 'test desc',
                        creation_date = three_days_ago,
                        creator_id=1, number_of_upvotes=0,
                        is_read=False, is_open=False,
                        is_offensive=False, is_FAQ=False)
    new_response = Response(ticket_id=1, response='test', responder_id=2, response_timestamp=datetime.now())
    db.session.add(new_ticket)
    db.session.add(new_response)
    db.session.commit()
    assert poor_resolution_time() == 'Email sent with details of agents with poor resolution time'
```

Manager is not sent an email report since all agents have a resolution time under 48 hours for tickets created in the past 30 days

Page being tested: Not an API Endpoint. This is a celery task triggered internally by cron
Inputs: None (The output of the task is dependent on the state of the db not by any inputs)
Expected Output: All Agents have have a resolution time less than 48 hours in the past 30 days
Actual Output: All Agents have have a resolution time less than 48 hours in the past 30 days
Result: Success

```python
def test_poor_resolution_time_email_not_sent_due_to_good_performance(app):
    one_day_ago = datetime.now() - timedelta(hours=24)
    app = app(CeleryTesting)
    new_ticket = Ticket(title='test',
                        description = 'test desc',
                        creation_date = one_day_ago,
                        creator_id=1, number_of_upvotes=0,
                        is_read=False, is_open=False,
                        is_offensive=False, is_FAQ=False)
    new_response = Response(ticket_id=1, response='test', responder_id=2, response_timestamp=datetime.now())
    db.session.add(new_ticket)
    db.session.add(new_response)
    db.session.commit()
    assert poor_resolution_time() == 'All Agents have have a resolution time less than 48 hours in the past 30 days'
```

Test Results

Apart from the tests explicitly mentioned above, we had written many more tests. All of the tests passed. There were 6 warnings pertaining to soon to be deprecated methods used by sqlalchemy/celery. All the tests reside in the following directory: .\Intermediate Work\Code\backend\test

```
============================= test session starts ==============================
platform darwin -- Python 3.10.2, pytest-7.2.2, pluggy-1.0.0
rootdir: /Users/varun/Documents/soft-engg-project-jan-2023-group-1-1/Intermediate Work/Code/backend/test
collected 107 items

test_arya.py ........................................                    [ 39%]
test_chirag.py .......................                                   [ 62%]
test_varun.py ........................................                   [100%]

======================== 107 passed, 6 warnings in 9.80s =======================
```

SOFTWARE ENGINEERING
Final Report

SUBMITTED IN THE PARTIAL FULFILLMENT OF THE REQUIREMENTS OF THE COURSE:

BSCSS3001: Software Engineering

By:
Arya Bhattacharyya (21f2000436)
Varun Venkatesh (21f1000743)
Chirag Goel (21f2000540)

INDIAN INSTITUTE OF TECHNOLOGY, MADRAS
April, 2023

Report on work done in different milestones

- Milestone 1 :The group identified the different users of the application in terms of the primary, secondary and tertiary users. Post the same, user stories (following SMART guidelines) were written for the different users identified earlier. 25 user stories which had the mandatory features as suggested by the project and some additional features as deemed necessary were compiled and submitted before the deadline.

- Milestone 2 : Four storyboards (using Comicgen and Google Slides) were depicted using comic strips:
    - 2 storyboards were made from the perspective of a support agent
    - 1 storyboard was made from the perspective of a student
    - 1 storyboard was made from the perspective of a manager.

  Wireframes were designed for the application using the design heuristics that were learnt in the Software Engineering course. Essential usability principles that were incorporated in our wireframe was as follows:
    - Effectiveness: To ensure that users can carry out their work efficiently.
    - Efficiency: Each user can effectively carry out their task in a minimal number of steps.
    - Safety: Confirmation prompts to prevent any unwanted actions
    - Learnability: Useful tool-tips on mouse hover
    - Memorability: Easy to use, consistent interface, which helps the user to remember the same.

  Some design heuristics that were followed during this phase were:
    - Consistency: Layout of different pages are similar.
    - Clean and Functional Design: Clutter free easy to recognize design which focuses on features and functionality, with pleasant aesthetics.
    - Show Status: Students can see if support agents have viewed the ticket raised by them. Post that, on answering the ticket, the ticket is displayed as closed (which can be reopened by the student if required), again showing the status of their ticket.
    - Provide Help: Useful tool-tips on mouse hover.

- Milestone 3 : A rough project schedule was drawn upon keeping in mind the deadlines for each milestone and the amount of work that was required for the same. Sprints were planned for the different tasks.Jira was chosen as an appropriate project management tool and a Gantt Chart roadmap was plotted to keep track of the progress.Different components were designed as UML diagrams using the Star UML software to model the software system. Details of scrum meetings conducted up until this stage were presented as “minutes of the meet” in the milestone 3 submission.

- Milestone 4 : API endpoints were described in the YAML file (Using OpenAPI 3.0 specification) and the coding for the backend of the application was also commenced during this milestone. The API specification was supposed to be the first version of the backend, which we believed could be subsequently changed based on any additional requirements/dependencies.
- Milestone 5 : Extensive testing (100+ tests) was done using Pytest for the code written for the backend. Tests directly relating to user stories were documented in the Milestone 5 report using the format given in the instructions. All tests were passed. Frontend coding for the software was commenced in this milestone.
- Milestone 6 : Frontend was completed and fully integrated with the backend. Frontend-backend integration was tested manually for all the pages to ensure that the software was behaving as expected. The second version of the API documentation was completed based on the changes and new introductions to the API. Finally, a report was made for the completed project and a video demo was recorded for the working application.

Implementation Details of the Project

- **Technologies and tools used**
  - Technologies for the backend
    - Flask
    - Flask Restful (For creating API endpoints)
    - Flask SQLAlchemy
    - Pytest (Only for testing)
    - Swagger Editor (Only for API documentation)
    - Postman, Thunderclient (For checking API endpoints manually)
    - Mailgun - A Transactional email service provider is used to send relevant emails to the users
  - Technologies for the frontend
    - Vue 3 CLI
    - Javascript
    - Vue Router
    - Vuex Store
    - Bootstrap (for aesthetics and styling)
    - HTML and CSS
    - ESlint to help with debugging the frontend
    - Axios and Fetch API for requests.

- General technologies used
  - Github (for versioning, code management, tracking, reviewing, issues, etc)
  - Algolia Search is used in the backend and frontend to create a smooth search experience
  - Jira (for project management)
  - ComicGen (for creating comics for storyboarding)
  - Star UML (for creating UML diagrams)
- **Hosting**
  - Currently the application is not hosted publicly.

- **Instructions to run our application**
  - On Ubuntu/MAC OS:
    - Git clone the repository.
    - Change the directory to the “backend” directory inside the “Milestone-6-Final-Submission” directory using the command :
      cd ./Milestone-6-Final-Submission/Code/backend
    - Create a Python virtual environment using the command:
      python3 -m venv “<<Name of the virtual environment”>>
    - Activate the virtual environment using the command:
      source <<Name of the virtual environment>>/bin/activate
    - Install the requirements using the command :
      pip3 install -r requirements.txt
    - Now, on the terminal type the following command to start the flask server:
      python3 main.py
    - In the same “backend” directory, in a new terminal inside the virtual environment start the redis server by typing:
      redis-server
    - Similarly, start the celery worker in the “backend” directory inside the virtual environment by typing:
      celery -A main.celery worker -l info
    - Furthermore start the celery beat in the “backend” directory inside the virtual environment by typing:
      celery -A main.celery beat --max-interval 1 -l info
    - For the frontend of the application, we would require node js and npm. We recommend to install the latest LTS version of Node js from the following link : https://nodejs.org/en
    - Once Node js is installed and npm is working, open a new terminal and change the directory to the “Code” folder inside the “Milestone-6-Final-Submission” directory which we saw earlier.

- Once inside the directory, change the directory to “frontend” by using the command: cd ./frontend/
- Now, run the following command to install the necessary packages for the frontend: npm install.
- After successful installation of the required packages, serve the frontend using the command: npm run serve
- The frontend server would be available on the URL: localhost:8080
- Please note that for the email functionality and search functionality to work, you would need API keys. In order to secure these API keys, they aren’t part of the repository. You would need to request the keys from us

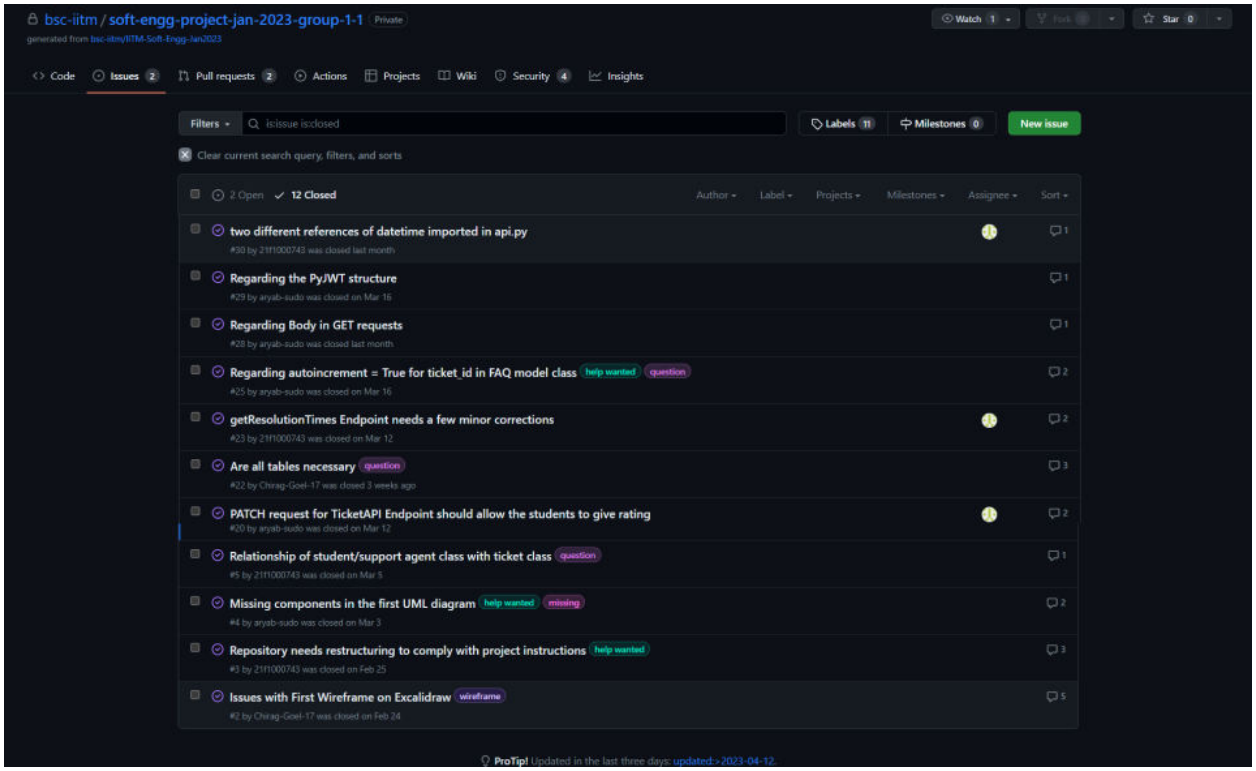
- On Windows:
  - Git clone the repository.
  - Change the directory to the “backend” directory inside the “Milestone-6-Final-Submission” directory using the command :
    cd .\Milestone-6-Final-Submission\Code\backend
  - Create a Python virtual environment using the command:
    python3 -m venv “<<Name of the virtual environment”>>
  - Activate the virtual environment using the command:
    .\<<Name of the virtual environment>>\Scripts\activate
    (You might have to enable Activate.ps1 on Windows, for which you can use the command (in a single line of the terminal):
    Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser {If you are still not able to activate your virtual environment, please visit these links to know more:
    a. https://docs.python.org/3/library/venv.html
    b. https://stackoverflow.com/questions/18713086/virtualenv-wont-activate-on-windows
  - Install the requirements using the command :
    pip install -r requirements.txt
  - Now, on the terminal type the following command to start the flask server:
    python main.py
  - In the same “backend” directory, in a new terminal inside the virtual environment start the redis server by typing:
    redis-server
  - Similarly, start the celery worker in the “backend” directory inside the virtual environment by typing:
    celery -A main.celery worker -l info

- Furthermore start the celery beat in the “backend” directory inside the virtual environment by typing:
  celery -A main.celery beat --max-interval 1 -l info
- For the frontend of the application, we would require node js and npm. We recommend to install the latest LTS version of Node js from the following link : https://nodejs.org/en
- Once Node js is installed and npm is working, open a new terminal and change the directory to the “Code” folder inside the “Milestone-6-Final-Submission” directory which we saw earlier.
- Once inside the directory, change the directory to “frontend” by using the command: cd .\frontend\
- Now, run the following command to install the necessary packages for the frontend: npm install.
- After successful installation of the required packages, serve the frontend using the command: npm run serve
- The frontend server would be available on the URL: localhost:8080

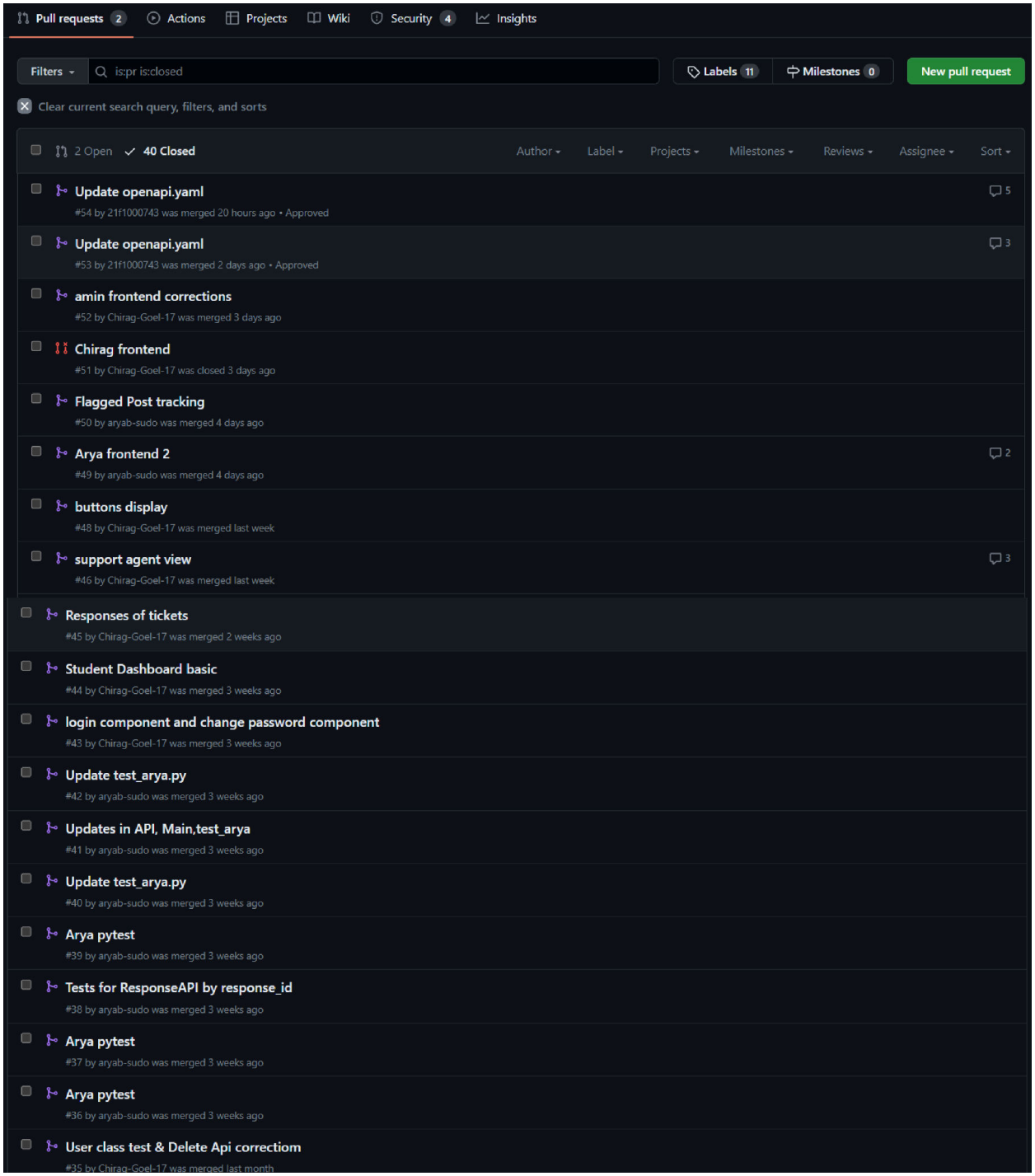
Code Review, Issue Reporting and Tracking

This was completely done on GitHub.

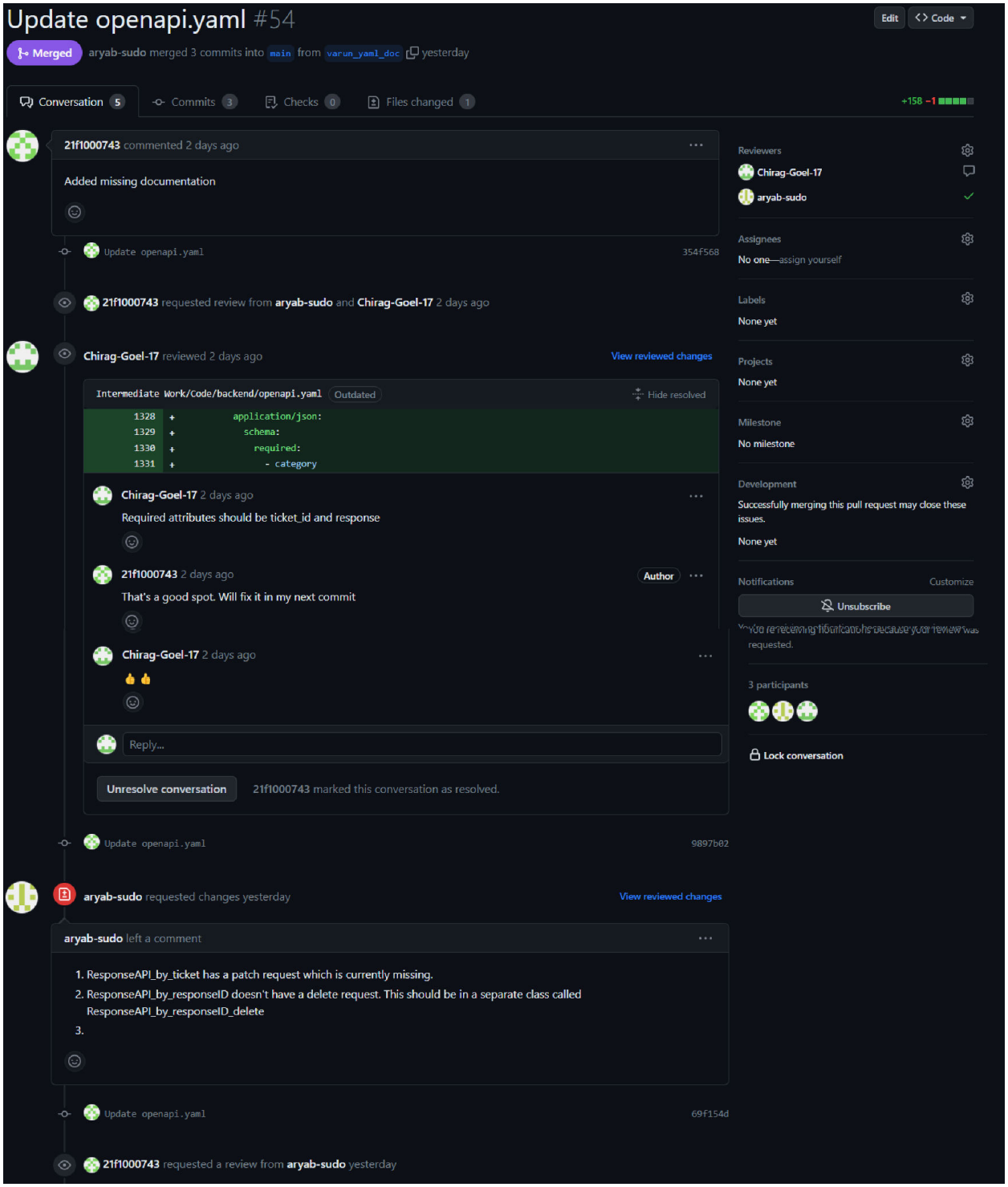
ISSUES:

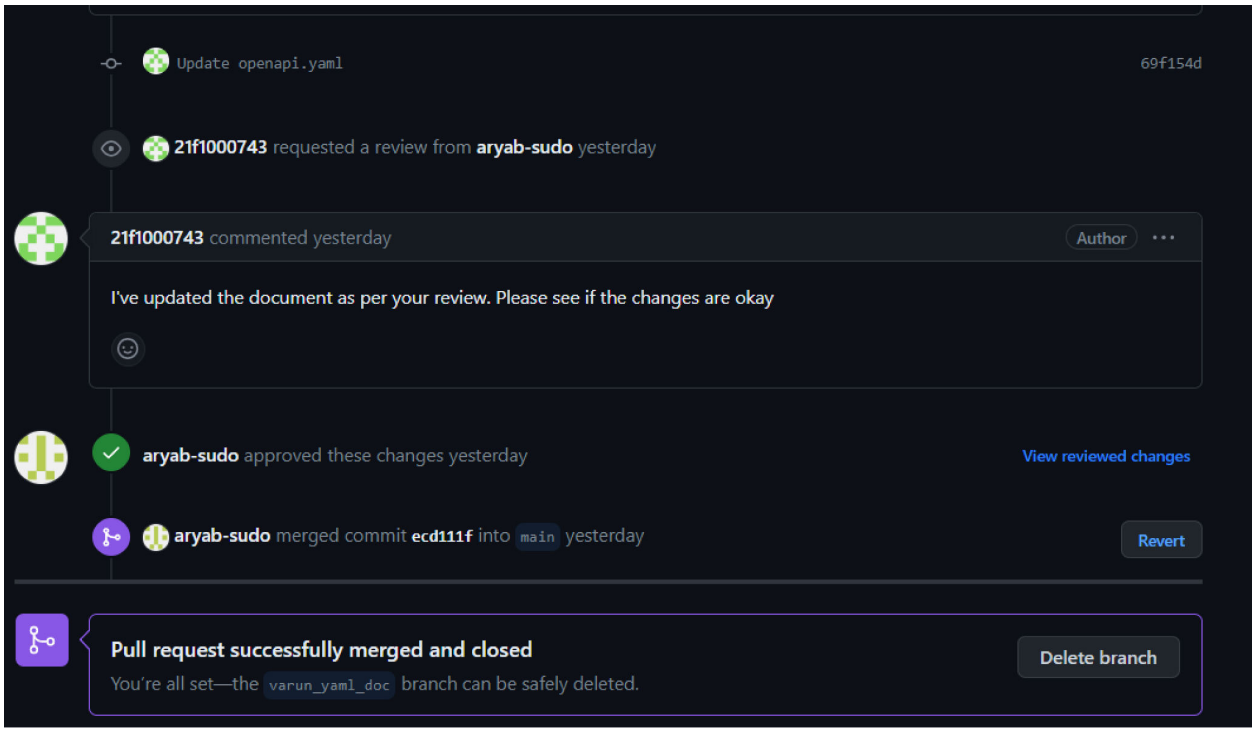


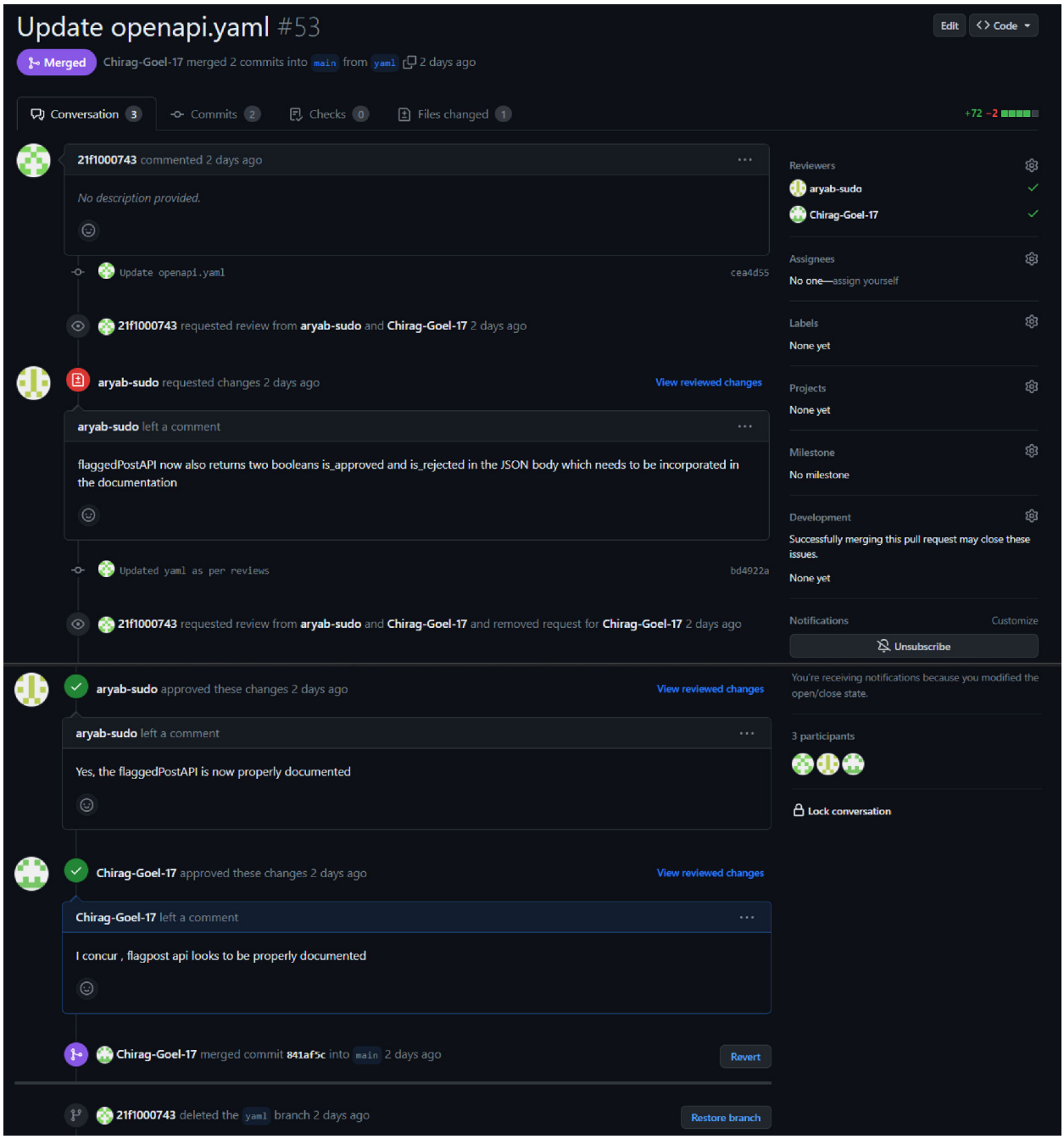
Pull Requests:

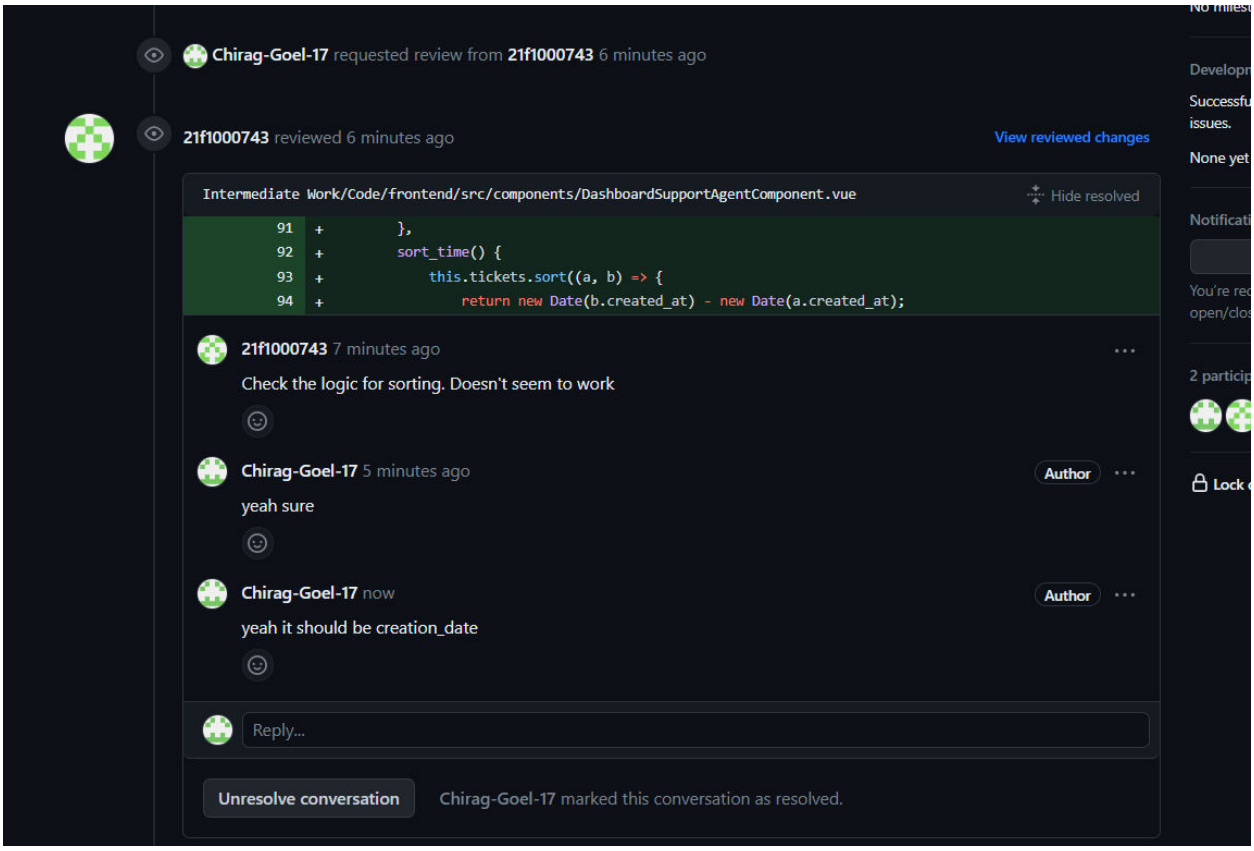


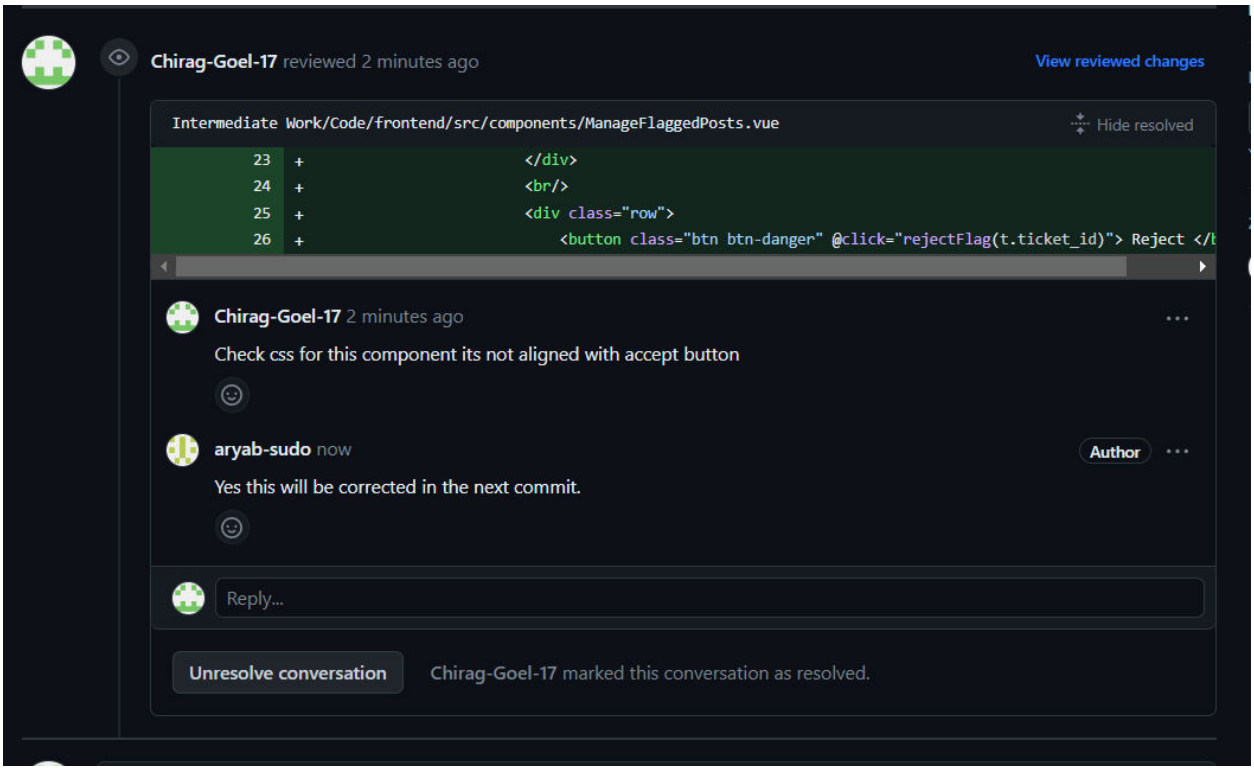
Code Reviews




Update openapi.yaml
69f154d
21f1000743 requested a review from aryab-sudo yesterday
21f1000743 commented yesterday
Author
I've updated the document as per your review. Please see if the changes are okay
aryab-sudo approved these changes yesterday
View reviewed changes
aryab-sudo merged commit ecd111f into main yesterday
Revert
Pull request successfully merged and closed
You're all set—the varun_yaml_doc branch can be safely deleted.
Delete branch


Update openapi.yaml #53
Edit
Code
Merged
Chirag-Goel-17 merged 2 commits into main from yaml 2 days ago
Conversation 3
Commits 2
Checks 0
Files changed 1
+72 −2
21f1000743 commented 2 days ago
No description provided.
Update openapi.yaml
cea4d55
21f1000743 requested review from aryab-sudo and Chirag-Goel-17 2 days ago
aryab-sudo requested changes 2 days ago
View reviewed changes
aryab-sudo left a comment
flaggedPostAPI now also returns two booleans is_approved and is_rejected in the JSON body which needs to be incorporated in the documentation
Updated yaml as per reviews
bd4922a
21f1000743 requested review from aryab-sudo and Chirag-Goel-17 and removed request for Chirag-Goel-17 2 days ago
aryab-sudo approved these changes 2 days ago
View reviewed changes
aryab-sudo left a comment
Yes, the flaggedPostAPI is now properly documented
Chirag-Goel-17 approved these changes 2 days ago
View reviewed changes
Chirag-Goel-17 left a comment
I concur , flagpost api looks to be properly documented
Chirag-Goel-17 merged commit 841af5c into main 2 days ago
Revert
21f1000743 deleted the yaml branch 2 days ago
Restore branch
Reviewers
aryab-sudo
Chirag-Goel-17
Assignees
No one—assign yourself
Labels
None yet
Projects
None yet
Milestone
No milestone
Development
Successfully merging this pull request may close these issues.
None yet
Notifications
Customize
Unsubscribe
You're receiving notifications because you modified the open/close state.
3 participants
Lock conversation


Chirag-Goel-17 requested review from 21f1000743 6 minutes ago
21f1000743 reviewed 6 minutes ago
View reviewed changes
Intermediate Work/Code/frontend/src/components/DashboardSupportAgentComponent.vue
Hide resolved
91 + },
92 + sort_time() {
93 + this.tickets.sort((a, b) => {
94 + return new Date(b.created_at) - new Date(a.created_at);
21f1000743 7 minutes ago
Check the logic for sorting. Doesn't seem to work
Chirag-Goel-17 5 minutes ago
Author
yeah sure
Chirag-Goel-17 now
Author
yeah it should be creation_date
Reply...
Unresolve conversation
Chirag-Goel-17 marked this conversation as resolved.


Chirag-Goel-17 reviewed 2 minutes ago
View reviewed changes
Intermediate Work/Code/frontend/src/components/ManageFlaggedPosts.vue
Hide resolved
23 +
24 +
25 +
26 + <button class="btn btn-danger" @click="rejectFlag(t.ticket_id)"> Reject </b
Chirag-Goel-17 2 minutes ago
Check css for this component its not aligned with accept button
aryab-sudo now
Author
Yes this will be corrected in the next commit.
Reply...
Unresolve conversation
Chirag-Goel-17 marked this conversation as resolved.