IIT MADRAS BS DEGREE

SOFTWARE ENGINEERING

COURSE PROJECT TITLE:- ONLINE SUPPORT TICKET SYSTEM

By:-
Dheeraj(21f1004378)

Vineeth(21f1004514)

Sharan(21f2000620)

Group Name:- Hyderabad360

- **Identifying User Requirements**.

  User Stories:-

  Milestone 1:-

  The identified users for the problem statement are Students, Support Staff and Admins. They are categorized as primary, secondary and tertiary users respectively.

  Student (Primary User):

  1. As a student, I want to be able to create a ticket for my specific concern or query, so that I can continue learning without any technical and
     administrative difficulties since this is an online degree.
  2. As a student, I want to be able to access a list of existing tickets and upvote the desired ticket, so that I can be notified when that query is answered.
  3. As a student, I want to be able to access the dynamic FAQ's page, so that I can see if my query has been addressed already.
  4. As a student, I want to be able to access the Important Links page, so that I can browse through content easily without the need for creating a query.
  5. As a student, I want to be able to close my tickets if they are no longer needed in order to reduce the work of the support staff.

Support staff (Secondary User):

1. As a member of the support staff, I would want to solve all the tickets which are addressed to me and notify the student post resolving, so that the student can clear his inhibitions and continue learning.
2. As a member of the support staff, I would like to be able to close tickets at any time in case they have already been solved in order to avoid working on solved issues again.

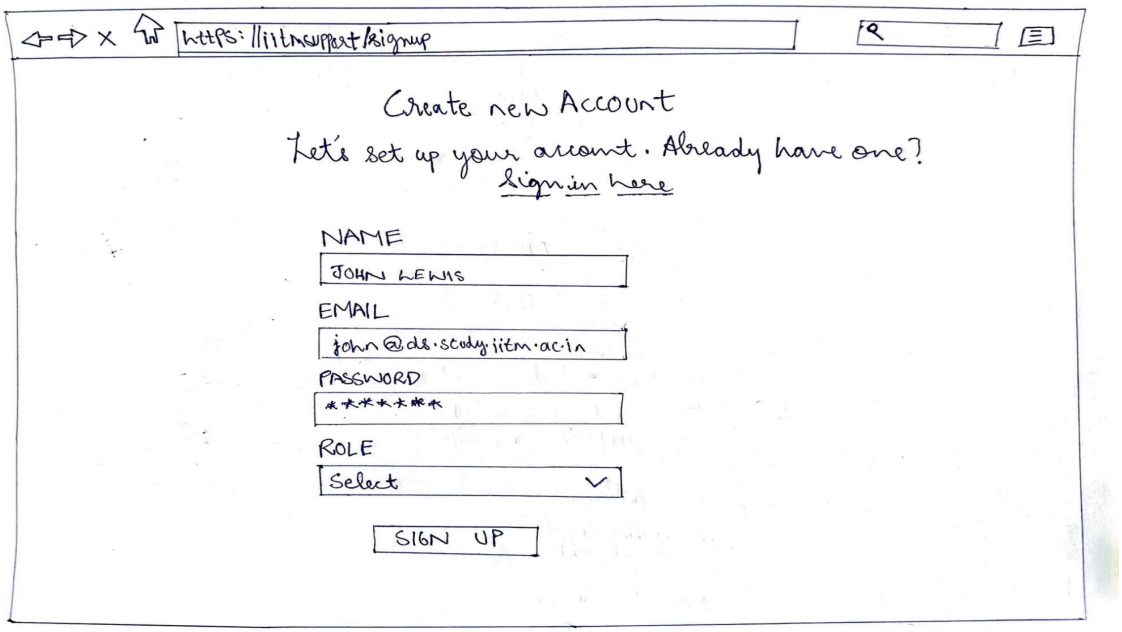
Support Admin (Tertiary User):

1. As an admin, I want to be able to create a dynamic FAQ list, so that it will be useful to future students and also decrease the burden of tickets on the support staff.
2. As a support admin, I want to be able to update the dynamic FAQ's page, so that the students can get their doubts clarified without raising a ticket.
3. As a support admin, I want to be able to update the important links page, so that the students can refer to various resources without raising a ticket.

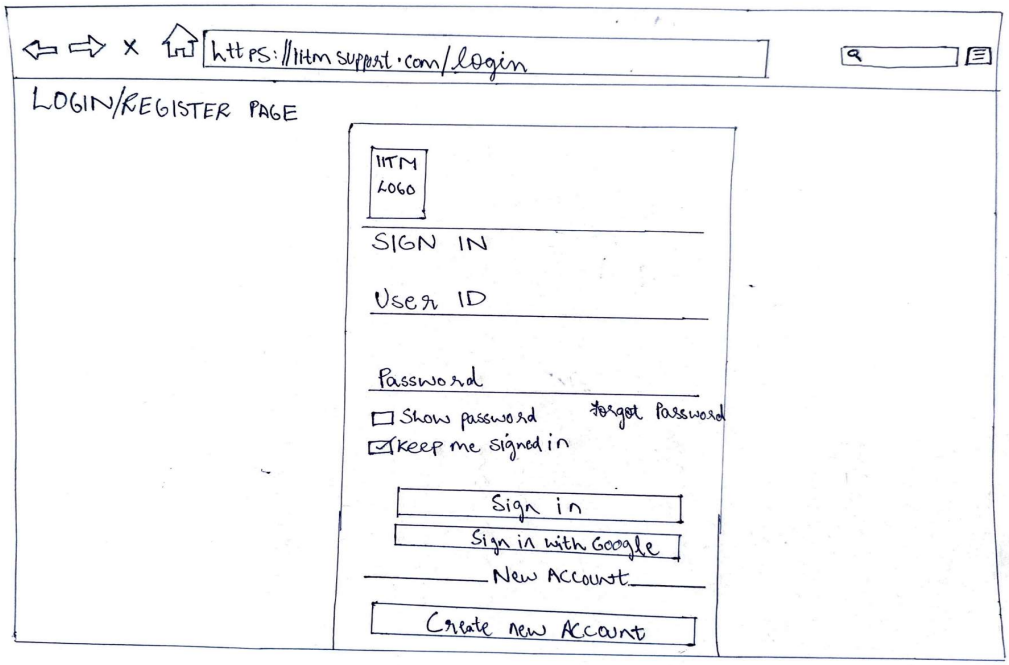
- **User Interfaces:**

This week, we came with the user interface design for the project. We looked at the user stories and made created wireframes to show the user interfaces. These interfaces are in the early stage and will be further refined as the project moves forward.

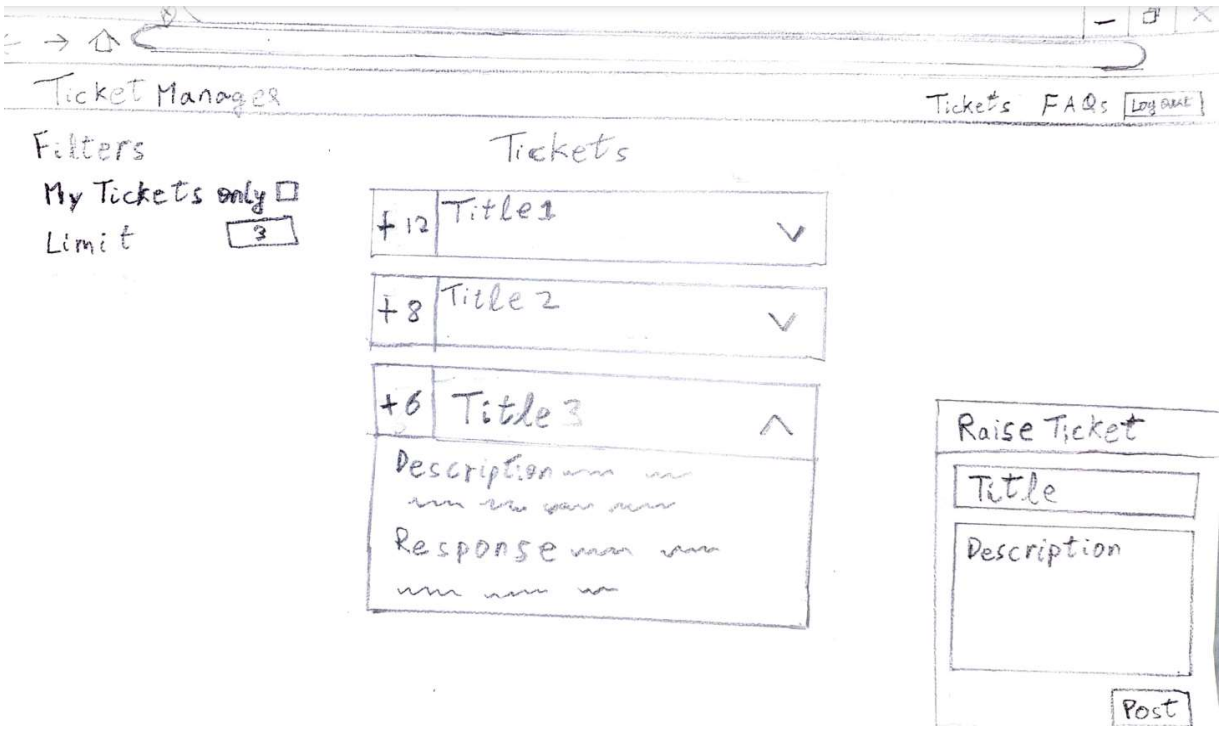
Signup Page:-



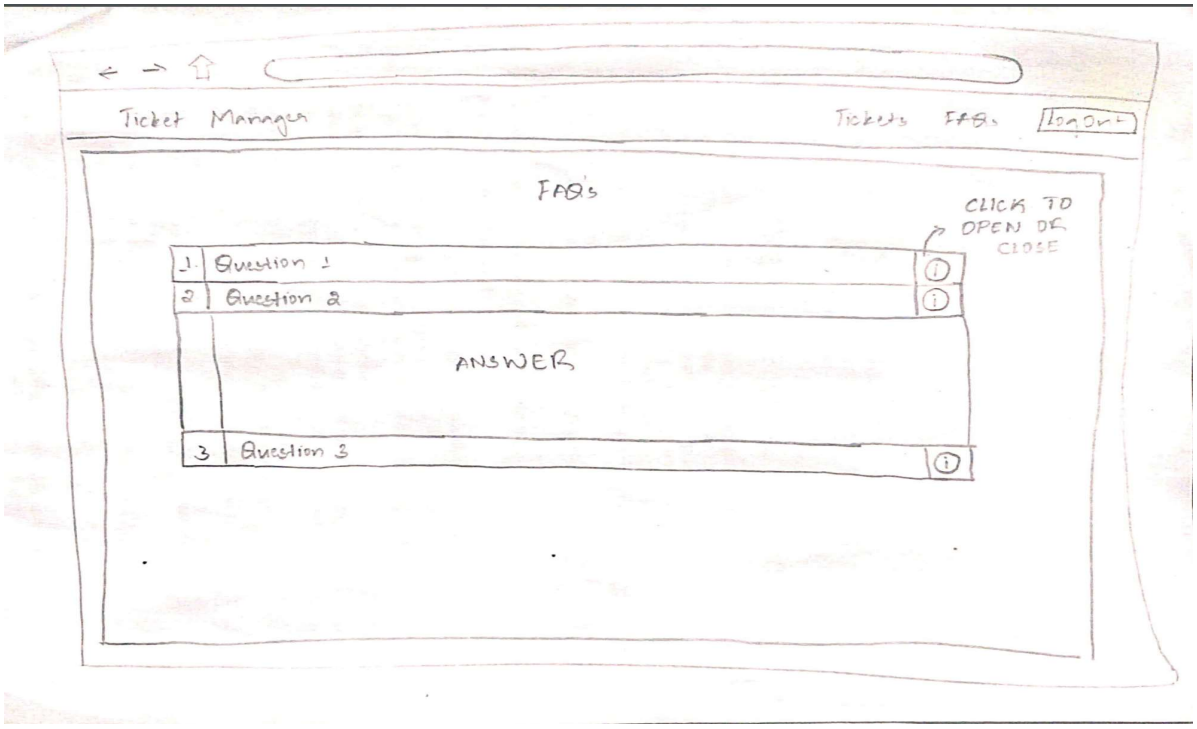
Login Page:-



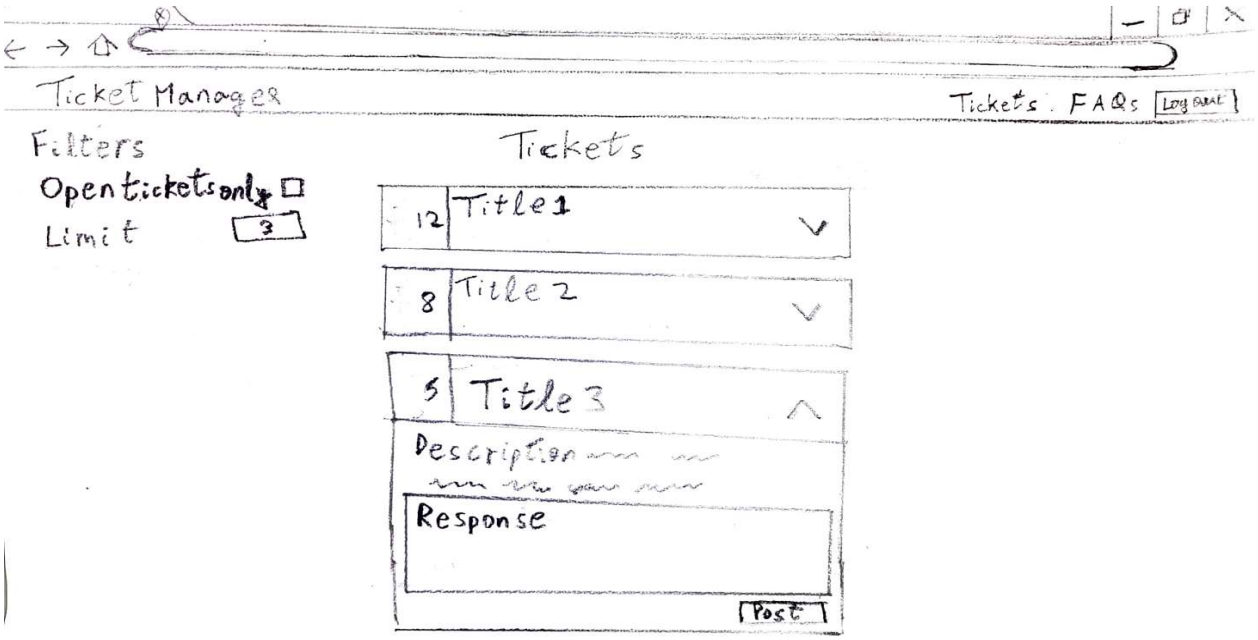
Tickets Page:- (Student View)



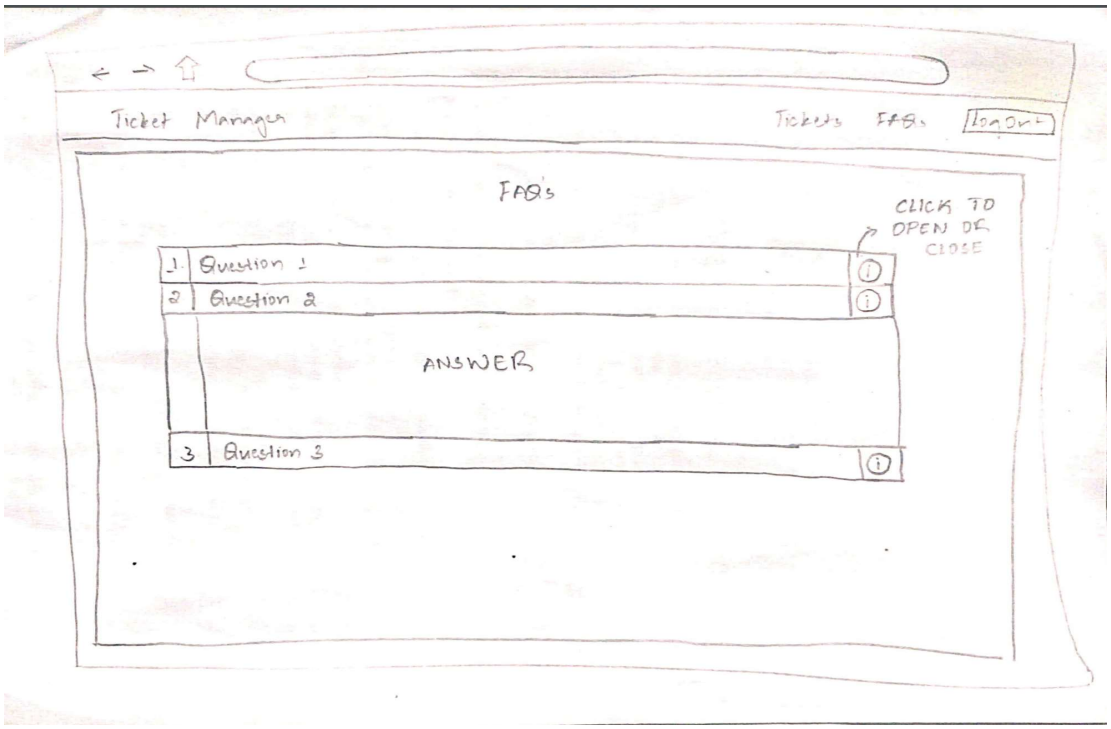
FAQ's Page:- (Student View)



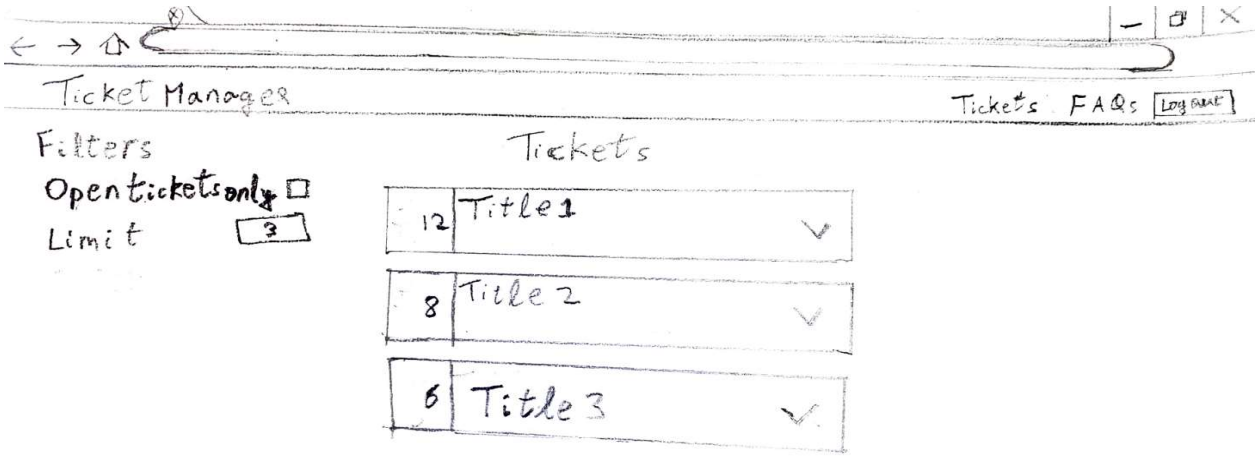
Tickets Page:- (Support Staff View)



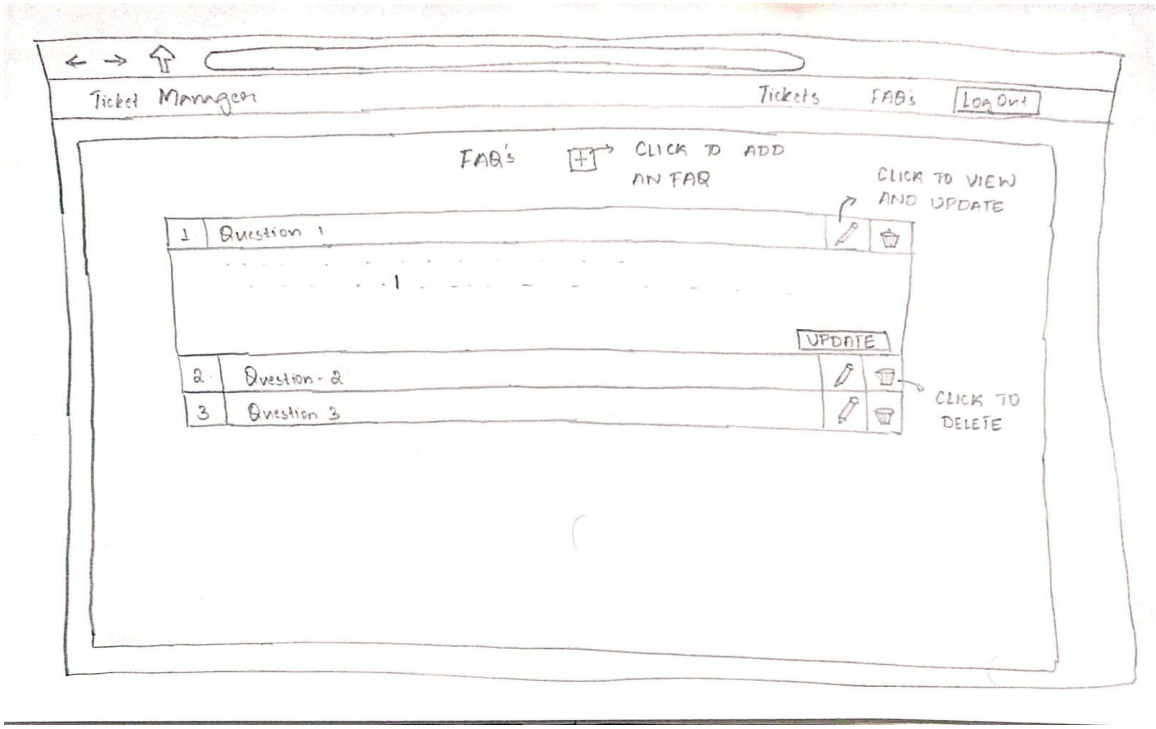
FAQ's Page:- (Support Staff View)



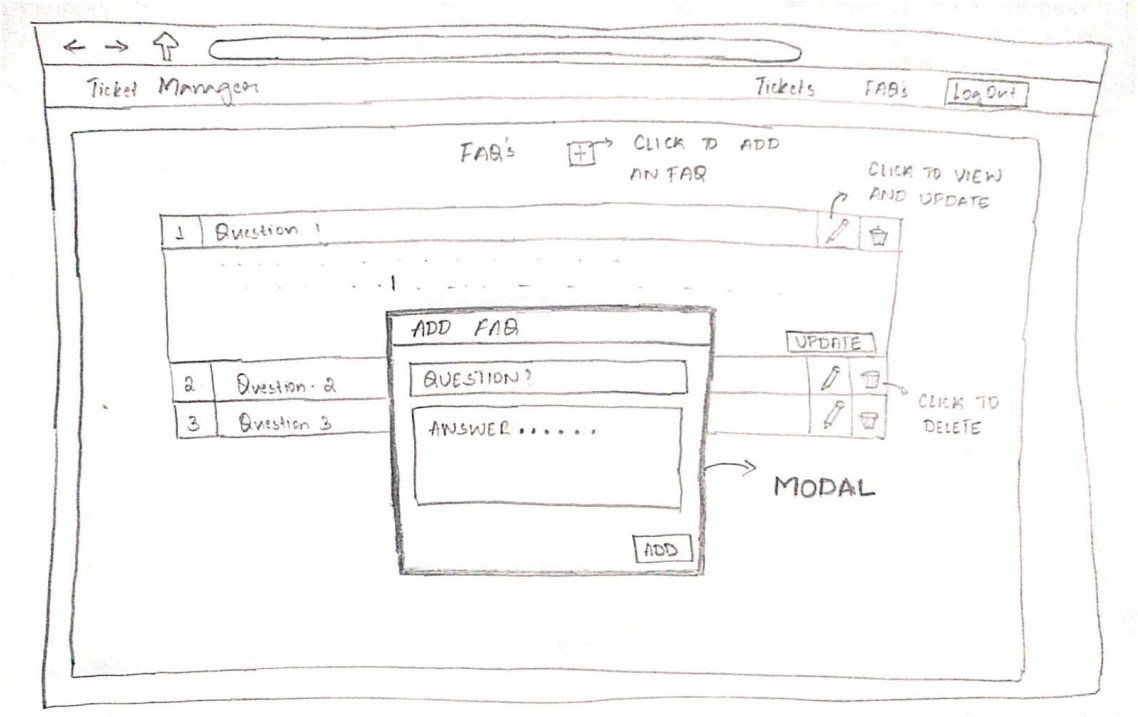
Tickets Page:- (Support Admin)



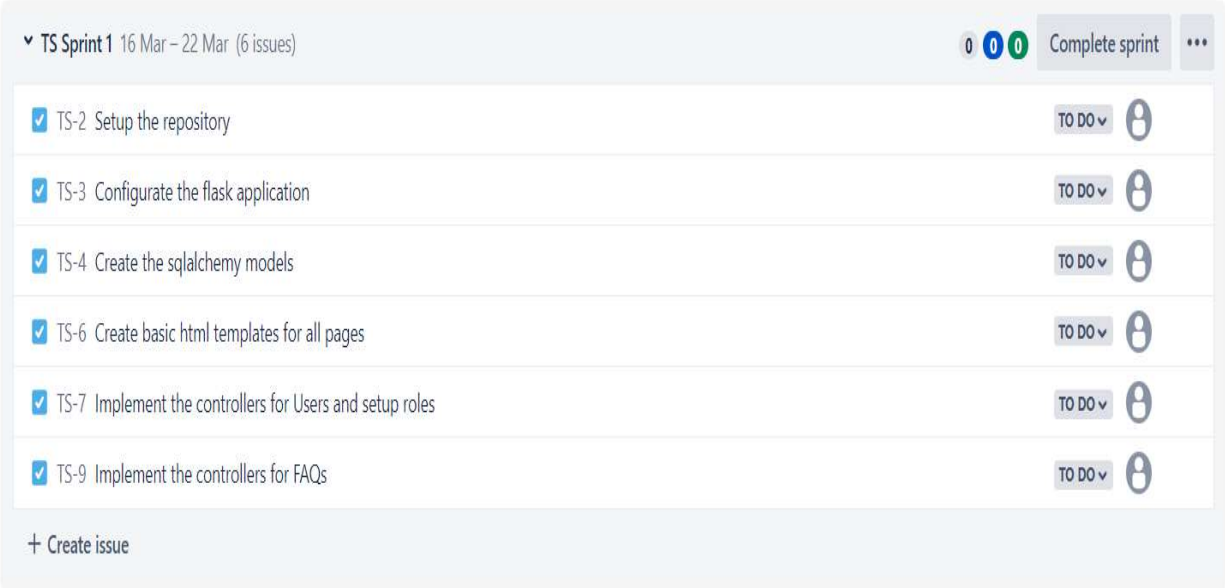
FAQ's Page:- (Support Admin)



Add FAQ Model:-

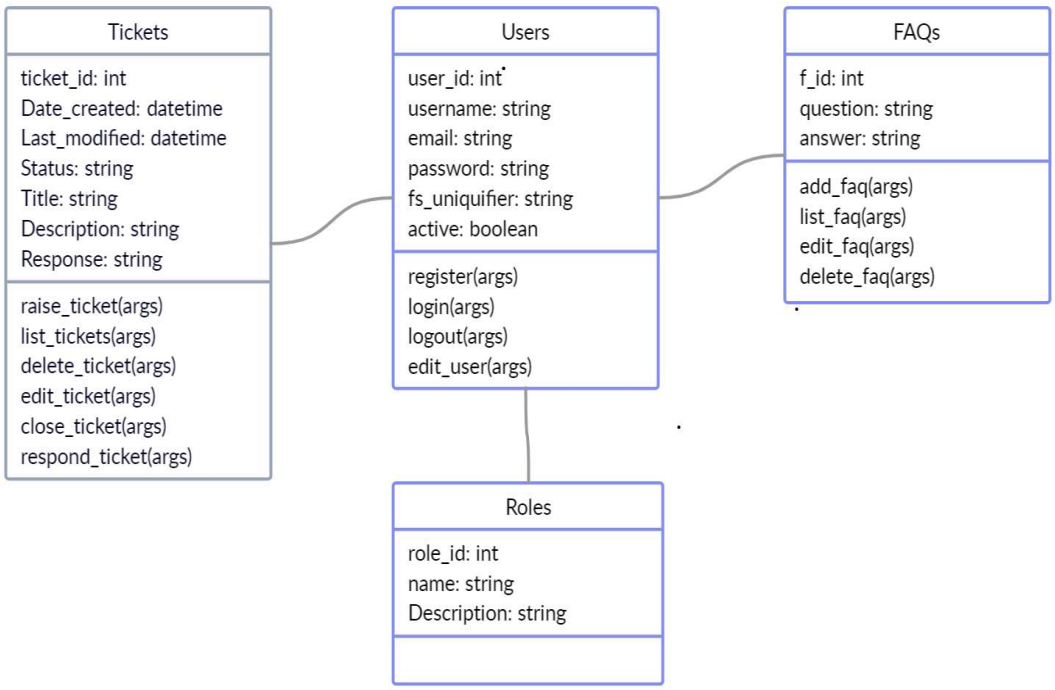


- **Scheduling and Design**
  The overall development will be split into multiple sprints. The tasks planned for the sprints are as follows.



The below screenshot is for Sprint 2 of the project lifecycle

The class diagram for the application is shown below. The different roles present in the application will determine the required access for the members.



- **API Endpoints:-**

  Below is the description of a few API Endpoints used in our project. The yaml file can be found in the repository.

  POST /tickets - Allows a student to create a new support ticket by providing a student ID, a title, and a description. Returns the ID of the created ticket, the student ID, title, description, upvotes, and status.

  GET /tickets - Allows a student to view a list of all existing support tickets and upvote them.

  GET /{ticket_id} - Allows a student or support staff member to view the details of a specific support ticket by providing the ticket ID. Returns the ID, student ID, title, description, upvotes, and status of the ticket.

  PUT /{ticket_id} - Allows a support staff member to update the status of a support ticket to either "open" or "closed". Requires a status parameter in the request body. Returns no content.

  DELETE /{ticket_id} - Allows a support staff member to delete a support ticket by providing the ticket ID. Returns no content.

  PUT /tickets/upvote - Allows a student to upvote a support ticket by providing the ticket ID. Returns no content.

  GET /tickets/upvote - Allows a student to view a list of the most upvoted support tickets, with a maximum number of tickets to return specified by the "limit" parameter. Returns the ID, student ID, title, description, upvotes, and status of the ticket.

  GET /faqs: Returns a list of all frequently asked questions.

GET /faqs/{id}: Returns a specific frequently asked question by ID.

POST /faqs: Creates a new frequently asked question.

PUT /faqs/{id}: Updates an existing frequently asked question by ID.

DELETE /faqs/{id}: Deletes a frequently asked question by ID.

POST /tickets/{ticket_id}/answer: Allows a support staff member to answer a ticket by providing the ticket ID and the answer details in the request body. Returns the updated ticket information.

PUT /tickets/{ticket_id}/answer/{answer_id}: Allows a support staff member to update an existing answer for a ticket by providing the ticket ID, answer ID, and the updated answer details in the request body. Returns the updated answer information.

- **Test Cases, Test Suite of the project:-**

  We used pytest script to execute these test cases on our project. The design and description of a few such test cases is mentioned below,

  test_ticket:

  Test Case 1: Creating a ticket and checking if it is returned with all the correct details
  Inputs: {"title": "Test Ticket", "description": "This is a test ticket"}
  Expected Output: status_code 201 and a JSON response with a ticket_id, title and description matching the inputs
  Actual Output: status_code 201 and a JSON response with a ticket_id, title and description matching the inputs
  Result: Success

Test Case 2: Updating a ticket and checking if it is returned with the correct updated details
Inputs: {"title": "Updated Test Ticket", "description": "This is an updated test ticket"}
Expected Output: status_code 200 and a JSON response with the ticket_id, title and description matching the updated inputs
Actual Output: status_code 200 and a JSON response with the ticket_id, title and description matching the updated inputs
Result: Success

test_vote:

Test Case 1: Adding an upvote to a ticket and checking if the upvote count is updated
Inputs: {"ticket_id": 1}
Expected Output: status_code 200 and a JSON response with the ticket_id and upvotes count updated by 1
Actual Output: status_code 200 and a JSON response with the ticket_id and upvotes count updated by 1
Result: Success

Test Case 2: Checking if the API returns all the tickets with their upvote count
Inputs: None
Expected Output: status_code 200 and a JSON response with all the tickets with their ticket_id, title, description, and upvotes count
Actual Output: status_code 200 and a JSON response with all the tickets with their ticket_id, title, description, and upvotes count
Result: Success

test_faq:

Test Case 1: Creating a new FAQ and checking if it is returned with all the correct details
Inputs: {"question": "Test Question", "answer": "This is a test answer"}

Expected Output: status_code 201 and a JSON response with an f_id, question and answer matching the inputs
Actual Output: status_code 201 and a JSON response with an f_id, question and answer matching the inputs
Result: Success

Test Case 2: Updating an FAQ and checking if it is returned with the correct updated details
Inputs: {"question": "Updated Test Question", "answer": "This is an updated test answer"}
Expected Output: status_code 200 and a JSON response with the f_id, question and answer matching the updated inputs
Actual Output: status_code 200 and a JSON response with the f_id, question and answer matching the updated inputs
Result: Success

- **Technologies and Tools used:-**

  We used the below technologies and tools for executing the Ticketing Software which is being currently described,

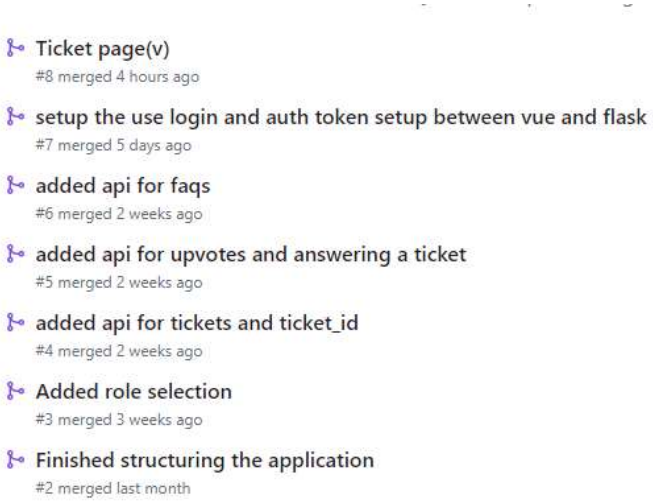
  1. Vue.js
  2. Flask
  3. SQL Alchemy
  4. HTML
  5. Bootstrap
  6. Visual Studio IDE

- **Instructions to run the application:-**
  1. Clone the github repository using Github CLI or download using as a zip file.
  2. Open the project in an IDE and setup the virtual environment, and also install the vue and axios packages with npm required for the frontend.

3. Install the required packages using the requirements.txt file in the project repository.
4. Run the app_run.sh and vue_run.sh scripts in seperate terminals to start the backend and frontend respectively.
5. Redirect to the localhost:8082 port for entering the landing page on your browser.

- **Issue Reporting and Tracking:-**

  This is the pull request history for our repository. The snippet below shows the sequence of such events.



- **Recorded Presentation of the working model:-**

  The below attached link is a working demonstration of the above discussed project.
  https://drive.google.com/file/d/1NK8Of1DU8m638H0PyD8vy58X1bx2t7it/view?usp=sharing

********** Thank You **********