Software Engineering Project : Group 14

Project report submitted to
Indian Institute of Technology, Madras
In partial fulfilment of the requirements for the course

BSCSS3001: Software Engineering

by

Tushar Supe (21F1003637)
Vaidehi Agarwal (21F1003880)

Online BSc in Programming and Data Science
Indian Institute of Technology
Madras 600 036 (India)
2023

Honour Code

I **Tushar Shrikrishna Supe** with roll no. **21F1003637** declare that I will not use any ideas, writings, code or work that is not my own or my groups with the intention of claiming it my or my group's work. For all the work that I will submit as part of this project, I will not share it outside my group with anybody directly or indirectly or upload it to any of the public forums on the internet.

I acknowledge that failing in any of the above constitutes plagiarism and in that case, the institute will take appropriate disciplinary action.

Sign: Tushar Supe

Date: 19-02-2023

I **Vaidehi Agarwal** with roll no. **21F1003880** declare that I will not use any ideas, writings, code or work that is not my own or my groups with the intention of claiming it my or my group's work. For all the work that I will submit as part of this project, I will not share it outside my group with anybody directly or indirectly or upload it to any of the public forums on the internet.

I acknowledge that failing in any of the above constitutes plagiarism and in that case, the institute will take appropriate disciplinary action.

Sign: Vaidehi Agarwal

Date: 19-02-2023

Acknowledgement

We have a great pleasure to express our deep sense of gratitude towards our respectful course instructors **Dr. Sridhar Iyer** (Department of Computer Science and Engineering & Inter-disciplinary Program in Educational Technology, IIT Bombay) and **Dr. Prajish Prasad** (Computer Science, FLAME University), for their generous encouragement throughout our project work. These words are formal thanks, which cannot express the real depth of feeling and affection that we have towards him.

We are also expressing our thanks towards our course support team members **Ankur Parmar** (MTech(CSE), IIT Bombay) and **Arup Chattopadhyay** (MTech, University of Calcutta) for their eminent guidance throughout our project work.

Tushar S Supe (21F1003637)

Vaidehi Agarwal (21F1003880)

Problem Statement

Title: Online support ticket system for the IITM BSc degree program

Description:

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 prioritised by the support team.

After the support team addresses the concern, they can mark the ticket as resolved, and an appropriate notification should be sent to concerned users. Another important feature of the ticketing system is dynamic FAQ updation. Many student concerns can be FAQs which will be useful for future students. If appropriate, the support query and response should be added to the FAQ section by support admins, and appropriately categorised, so that an updated FAQ will be readily available to students. The platform should allow users to enrol as students, support staff and admins. Apart from these standard requirements, you can also think of other features which can add value to users.

Contents

MILESTONE : 1
USER REQUIREMENTS

1. Various Users

1.1. Types of Users

The users can be categorised into three types mainly 'Primary', 'Secondary' and 'Tertiary'. The different types and users are summarised in the table below.

Identified Users

Category	Users	Remark
Primary	Students, Support Team	Student are primary users as they will create a ticket and then support team will resolve the ticket.
Secondary	System Software Developer, Admins	System developers will get users feedback and improve functionality. Admins will validate users and also create FAQs.
Tertiary	IIT Madras, App Hosting Platforms, Future Students.	IITM may fund the software development process. Hosting platform like Heroku, Replit will allot resources for the system.

1.2. User Stories

The common features for the support ticket system are listed below.

- Login, Logout, Register
- Create, Delete, Update Ticket
- View Tickets and Sort/Filter
- Vote Tickets
- Resolve Ticket
- View history of tickets
- View resolved tickets list
- Send Notifications
- Update FAQs

Based on these common features, the user stories for various users are defined with the help of SMART guidelines.

User Stories for Primary Users

User		Story
Student	[1]	As a student, I want to register, So that I can start using support ticket system
	[2]	As a student, I want to login, So that I can use support ticket system
	[3]	As a student, I want to logout, So that I can successfully sign out from support ticket system
	[4]	As a student, I want to update my profile, So that I can change my credentials whenever required.
	[5]	As a student, I want to change password, So that I can keep my account safe.
	[6]	As a student, I want to create a ticket, So that I can get help from support staff.
	[7]	As a student, I want a ticket deleting option, So that I can delete a ticket whenever I wish to delete.
	[8]	As a student, I want to see the list of similar tickets, So that I can avoid creating duplicate ticket.
	[9]	As a student, I want to ticket filtering option, So that I can see list of tickets based on the tags I have selected.
	[10]	As a student, I want to able to like or add +1 to existing ticket, So that I can prioritize my concern and avoid duplication.
	[11]	As a student, I want to receive a notification from time to time, So that I can get the information about the current status of my ticket.
	[12]	As a student, I want to receive a notification for ticket resolve, So that I can go to solution provided by support staff and carry out my tasks.

Support Staff	[1]	As a support staff, I want to sign up, So that I can start using support ticket system

	[2]	As a support staff, I want to login, So that I can use support ticket system
	[3]	As a support staff, I want to logout, So that I can stop using support ticket system
	[4]	As a support staff, I want to update my profile, So that I can change my information whenever required
	[5]	As a support staff, I want to see the list of tickets sorted as unresolved then resolved, So that I can differentiate between which needs to be answered and which has already been answered.
	[6]	As a support staff, I want to see highest priority unsolved concerns first, So that they can be answered first.
	[7]	As a support staff, I want to able to mark the ticket as resolved, So that a notification to the concerned student can be sent

User Stories for Secondary Users

User		Story
Admin	[1]	As a system admin, I want to see list of most voted concerns, So that I can convert them into FAQ for future students.
	[2]	As a system admin, I want to see student credentials, So that I can validate them while creating new accounts.
	[3]	As a system admin, I want to sign up, So that I can start using support ticket system
	[4]	As a system admin, I want to login, So that I can start monitoring support ticket system
	[5]	As a system admin, I want to logout, So that I can safely get out of support ticket system
	[6]	As a system admin, I want to see support staff credentials, So that I can validate them while creating new accounts.

System Developer	[1]	As a system software developer, I want to get feedback from users, So that I can improve the functionality as well as add extra features.
	[2]	As a system software developer, I want to get system performance report, So that I can track the performance and usability of the system.

User Stories for Tertiary Users

User		Story
IIT Madras R&D	[1]	As a IITM representative, I want to see the software performance and usability, So that I can fund the software development.
App Hosting Platform	[1]	As a 3rd party hosting platform, I want to know the software specifications, number of users and related metadata, So that I can allot proper resources for the system to work without any lag.

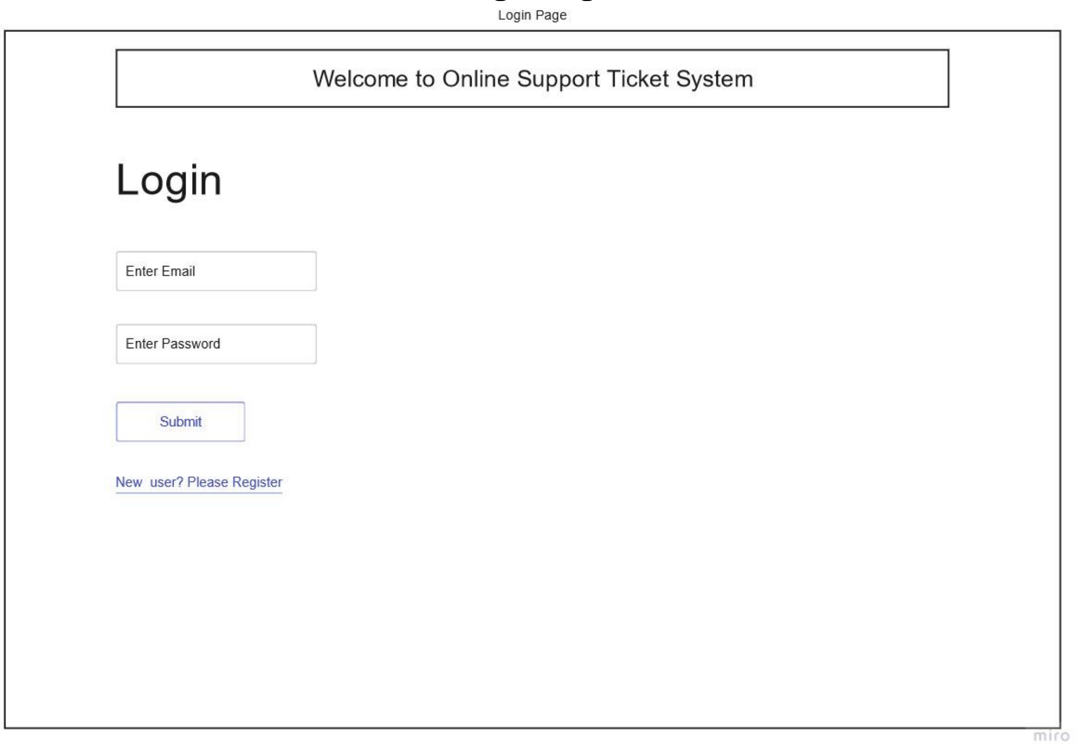
MILESTONE : 2
USER INTERFACES

2. Wireframes

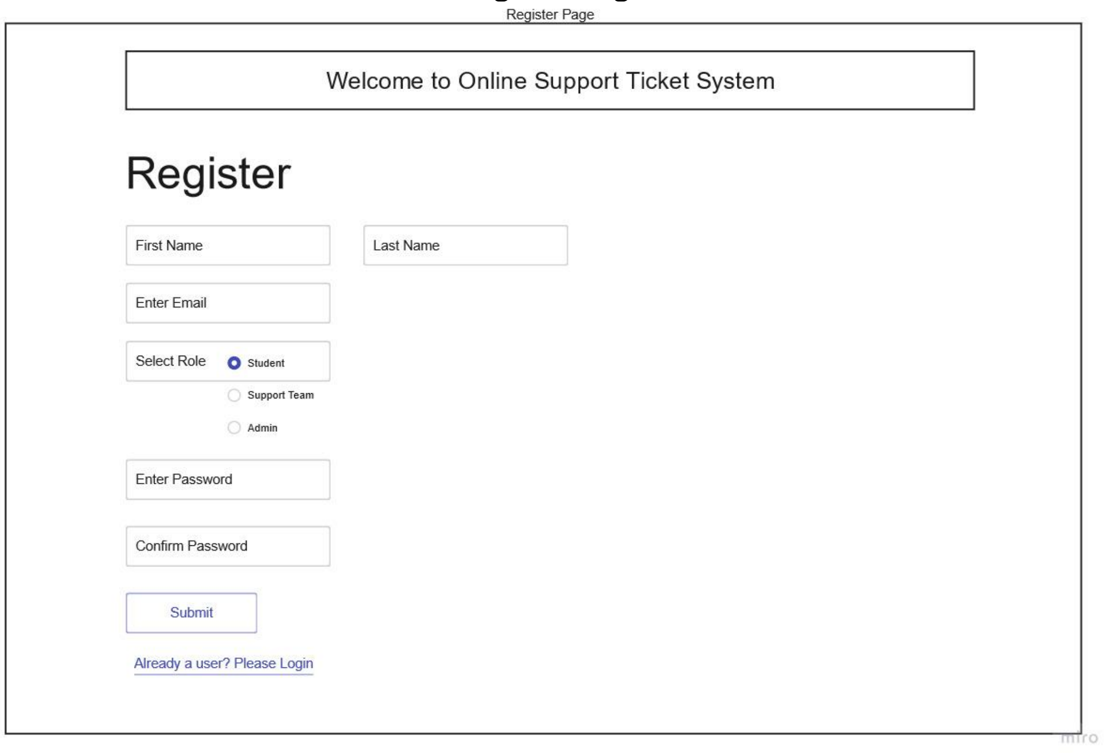
2.1. Low Fidelity Wireframes

2.1.1 Common

Login Page


Login Page
Welcome to Online Support Ticket System
Login
Enter Email
Enter Password
Submit
New user? Please Register
miro

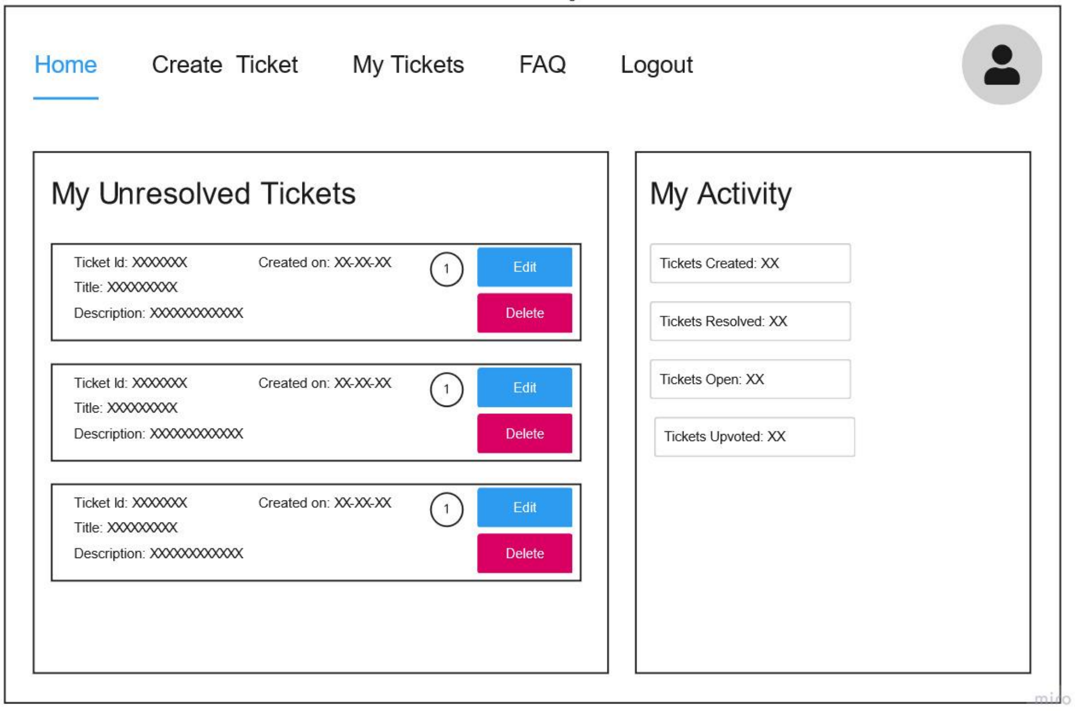
Register Page



2.1.2 User :- Student

Home Page

Student Home Page



My Unresolved Tickets

Ticket Id: XXXXXXX Created on: XX-XX-XX
Title: XXXXXXXXX
Description: XXXXXXXXXXXX

Ticket Id: XXXXXXX Created on: XX-XX-XX
Title: XXXXXXXXX
Description: XXXXXXXXXXXX

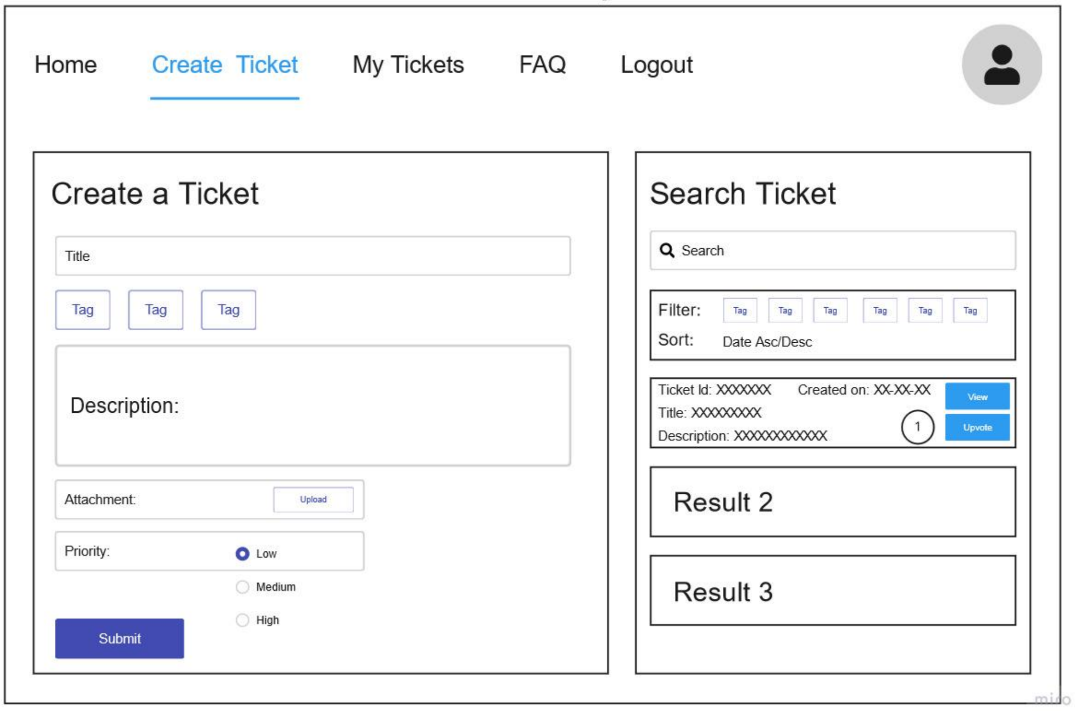
Ticket Id: XXXXXXX Created on: XX-XX-XX
Title: XXXXXXXXX
Description: XXXXXXXXXXXX

My Activity

Tickets Created: XX
Tickets Resolved: XX
Tickets Open: XX
Tickets Upvoted: XX

Create Ticket Page

Student Create Ticket Page



Create a Ticket

Title

Description:

Attachment:

Priority: [✓ Low] Medium High

Search Ticket

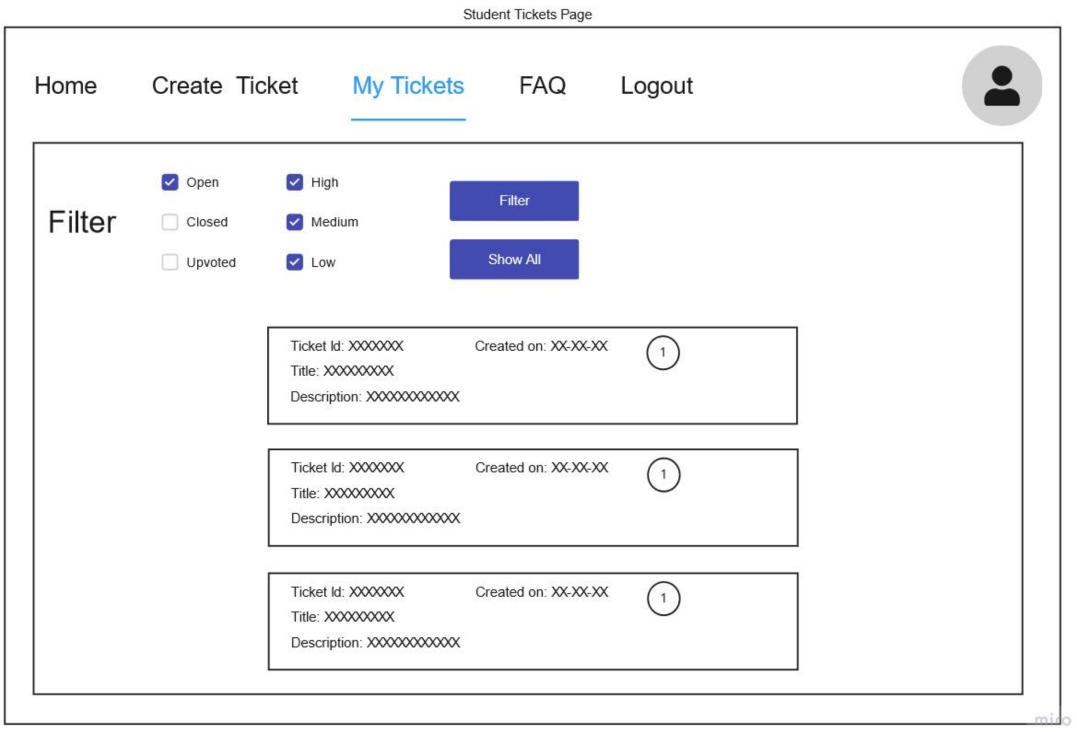
Filter:
Sort: Date Asc/Desc

Ticket Id: XXXXXXX Created on: XX-XX-XX
Title: XXXXXXXXX
Description: XXXXXXXXXXXX

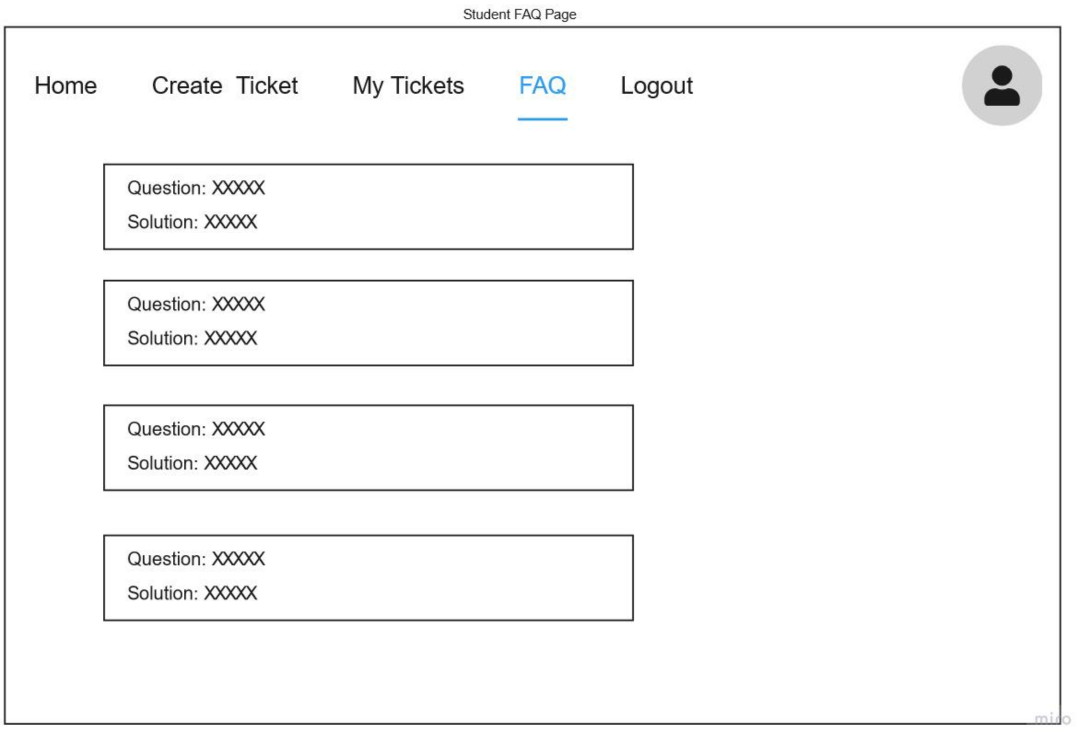
Result 2

Result 3

My Tickets Page

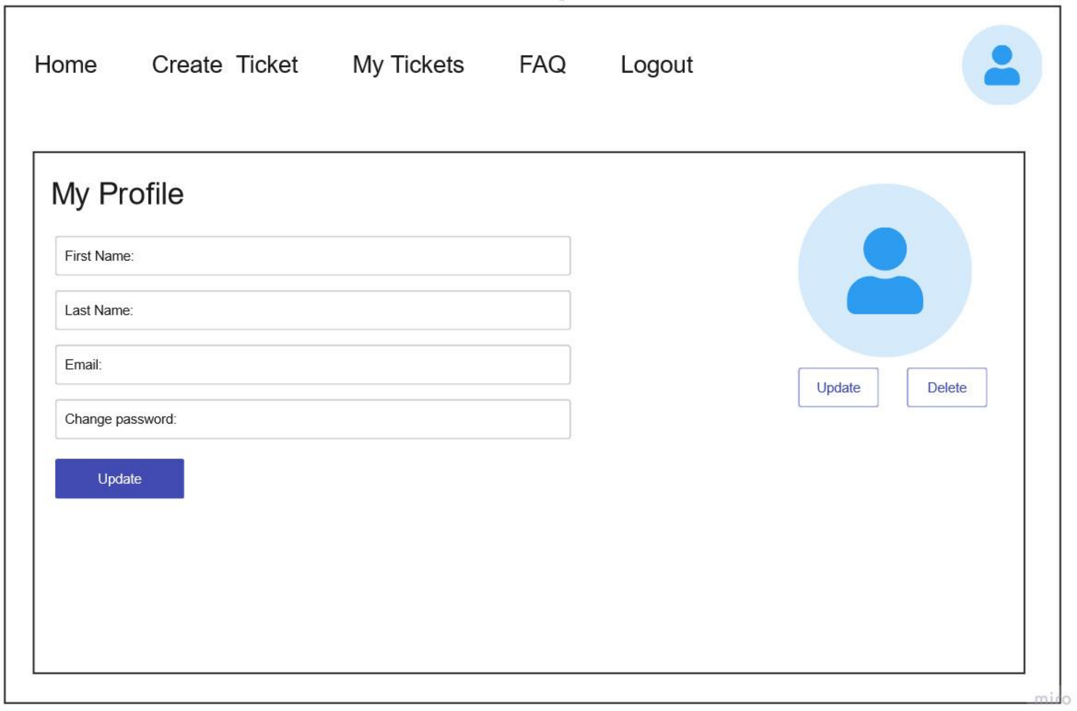


FAQ Page



Profile Page

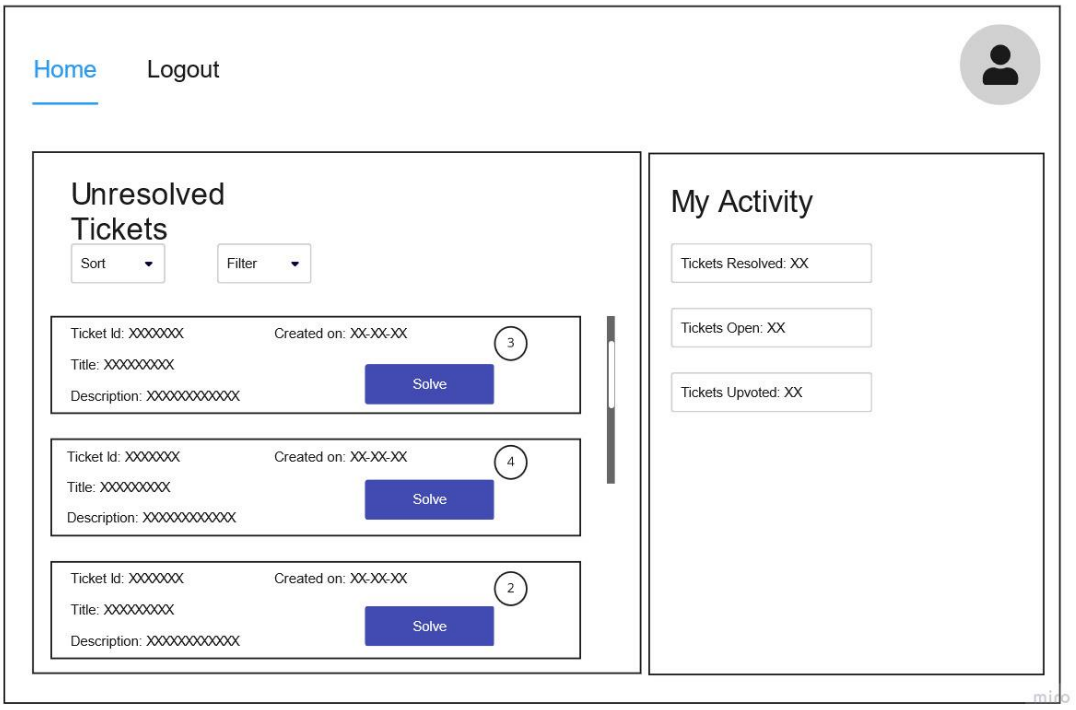
Student Profile Page



2.1.3 User :- Support Staff

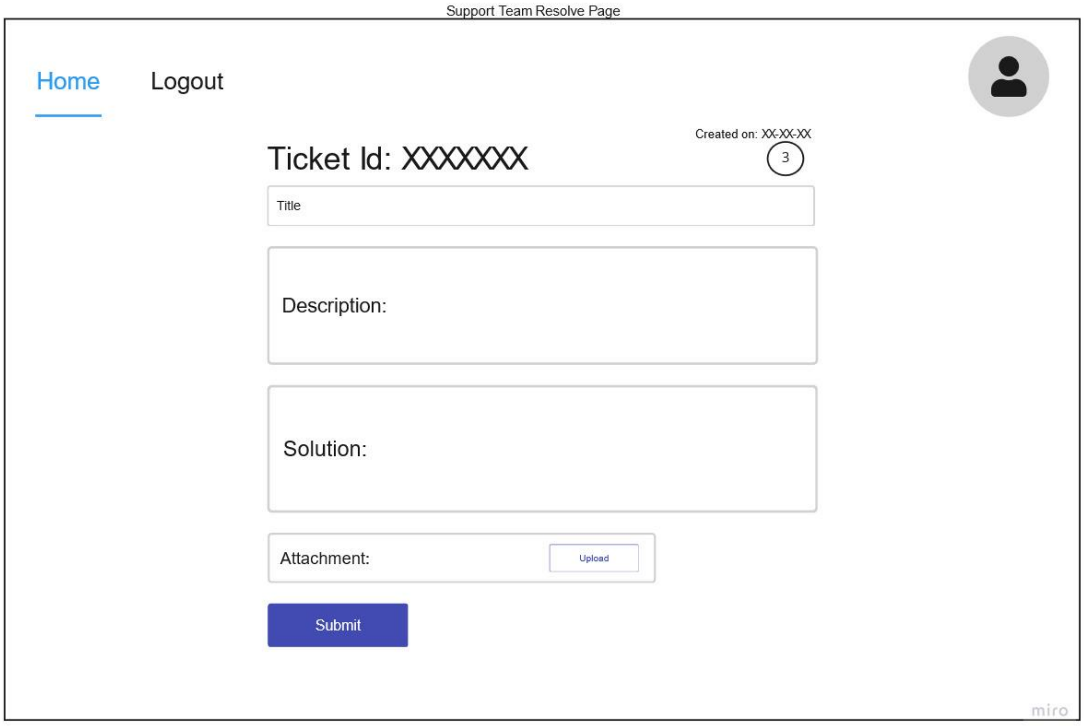
Home Page

Support Team Home Page



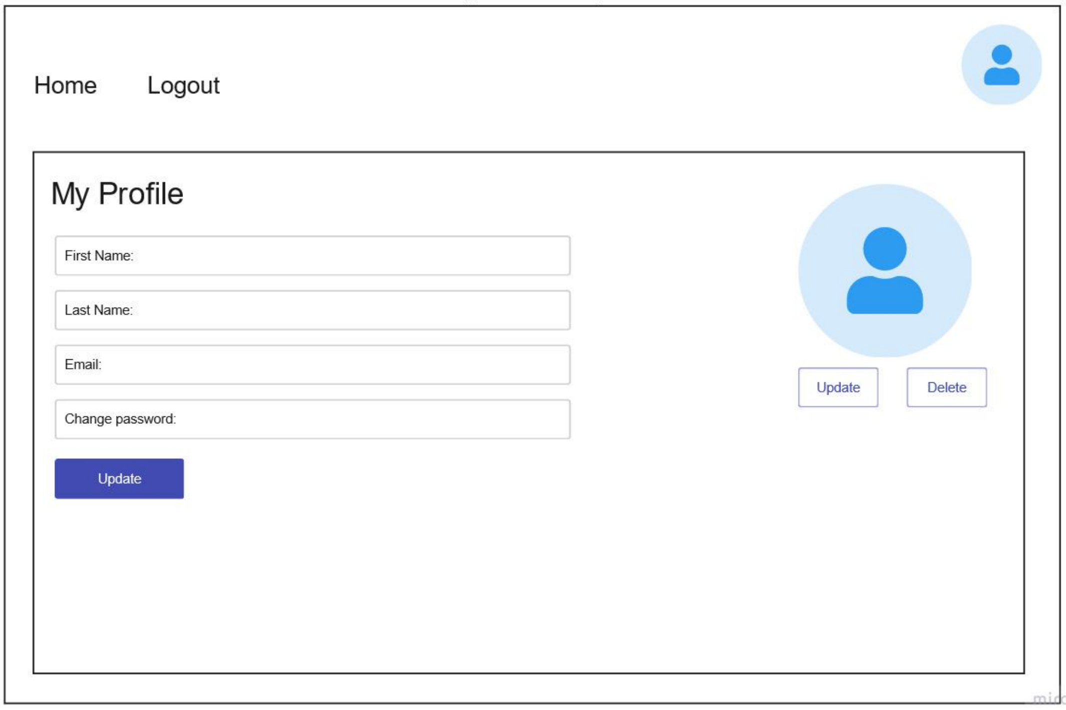
Ticket Resolve Page

Support Team Resolve Page



Profile Page

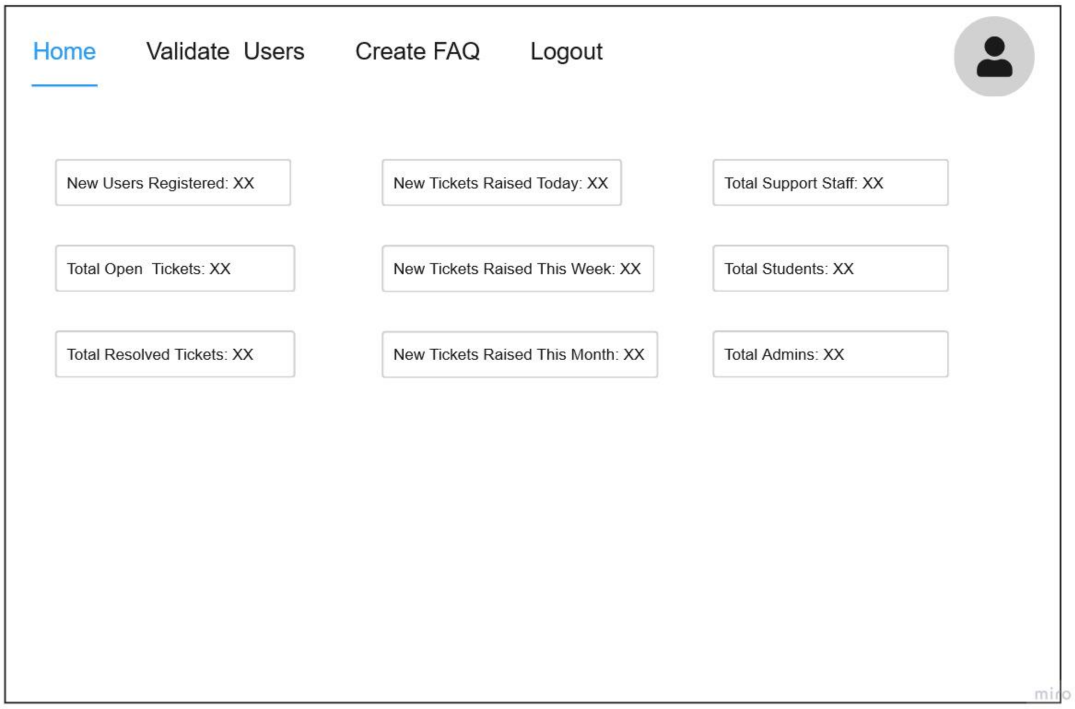
Support Team Profile Page



2.1.4 User :- Admin

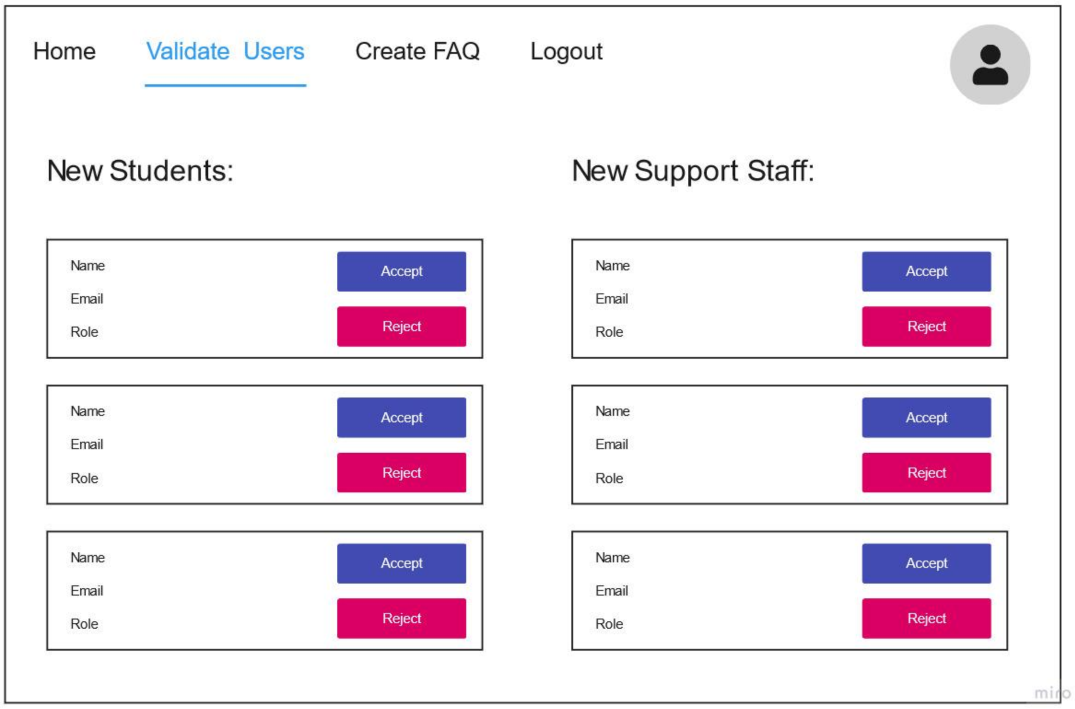
Home Page

Admin Home Page

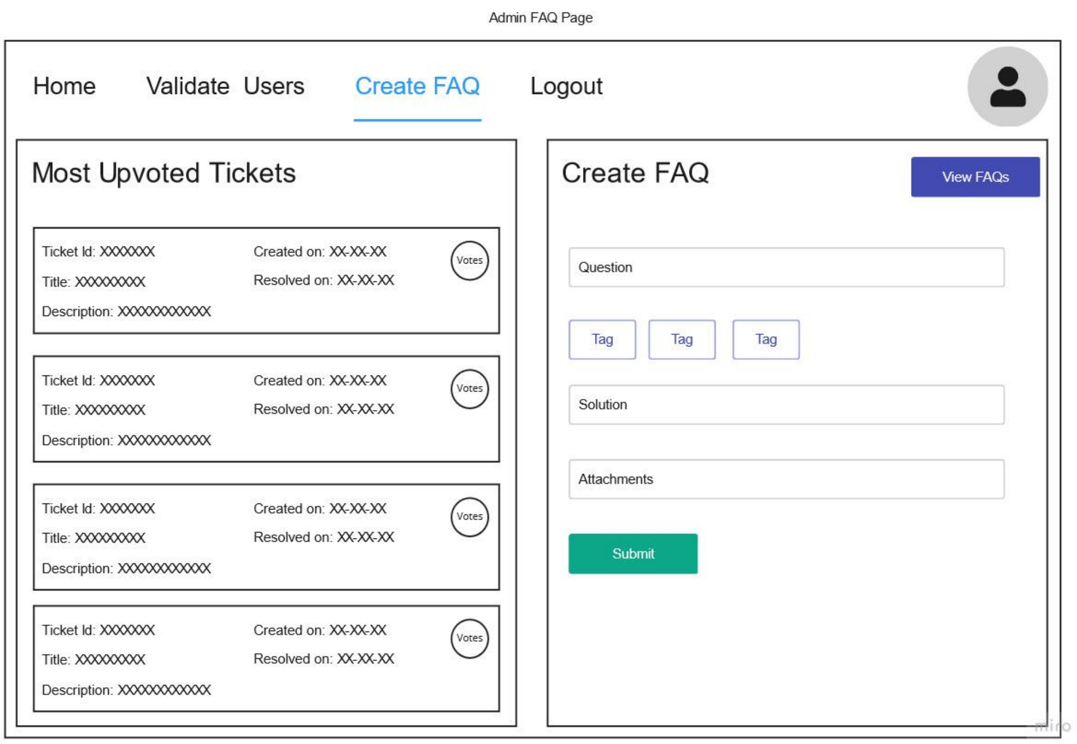


Validation Page

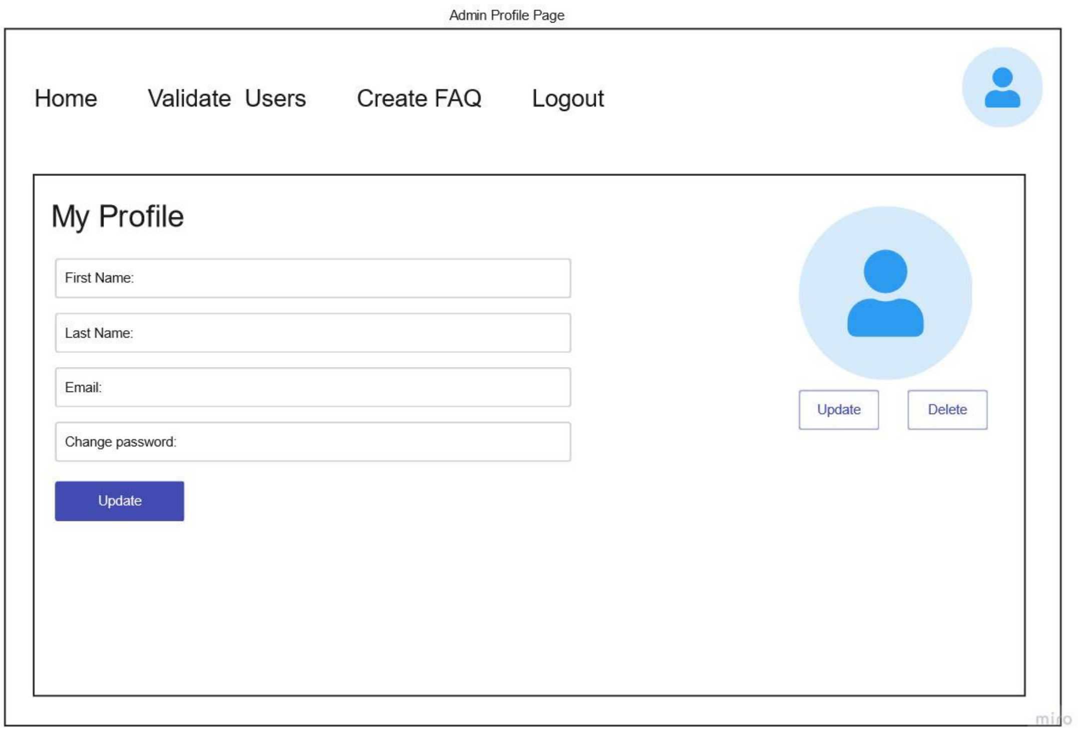
Admin Validation Page



FAQ Create Page



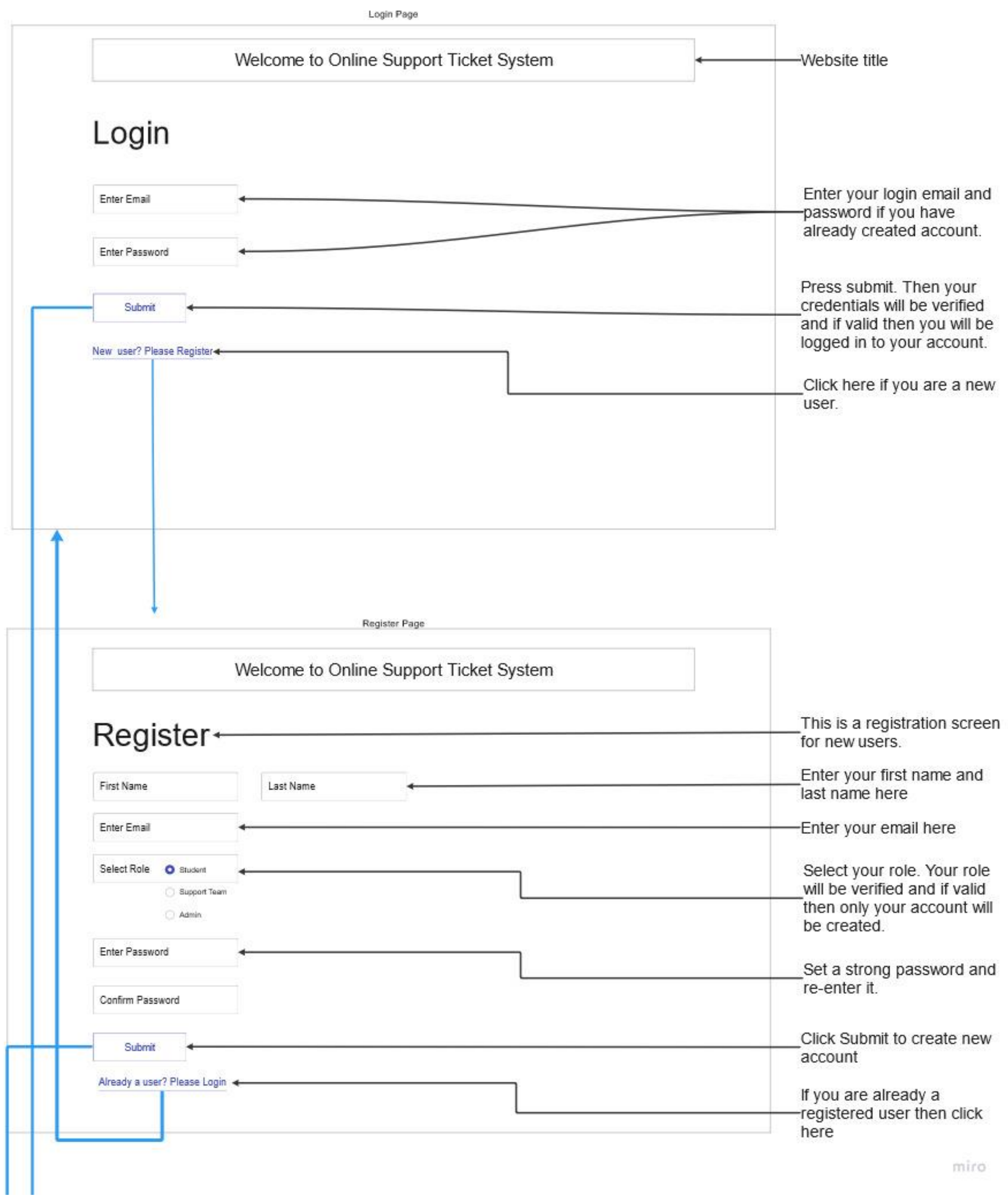
Profile Page



2.2. Storyboards for Users

2.2.1 Storyboard for Student

Storyboard for User: Student



Website title

Enter your login email and password if you have already created account.

Press submit. Then your credentials will be verified and if valid then you will be logged in to your account.

Click here if you are a new user.

This is a registration screen for new users.

Enter your first name and last name here

Enter your email here

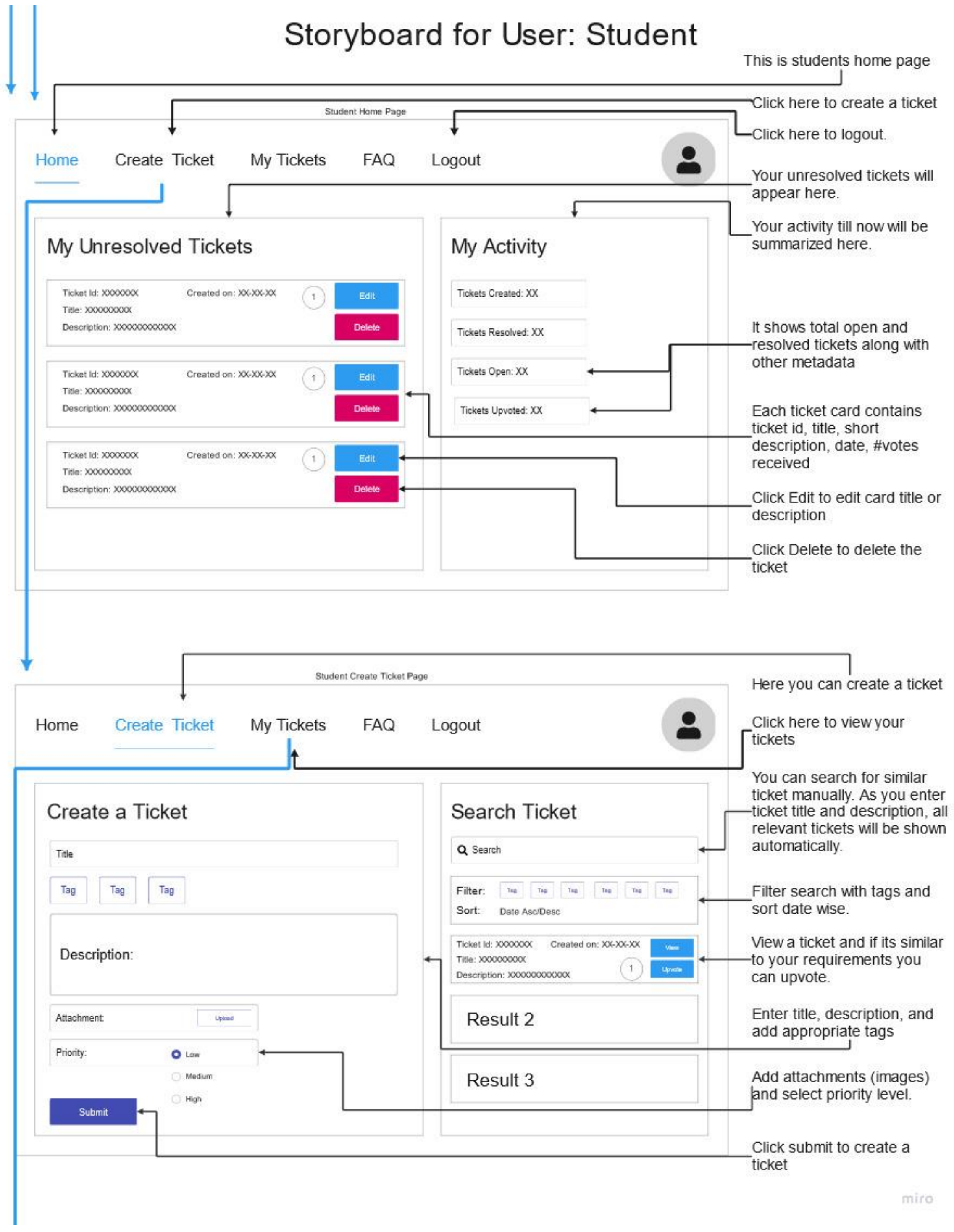
Select your role. Your role will be verified and if valid then only your account will be created.

Set a strong password and re-enter it.

Click Submit to create new account

If you are already a registered user then click here

Storyboard for User: Student



This is students home page

Click here to create a ticket

Click here to logout.

Your unresolved tickets will appear here.

Your activity till now will be summarized here.

It shows total open and resolved tickets along with other metadata

Each ticket card contains ticket id, title, short description, date, #votes received

Click Edit to edit card title or description

Click Delete to delete the ticket

Here you can create a ticket

Click here to view your tickets

You can search for similar ticket manually. As you enter ticket title and description, all relevant tickets will be shown automatically.

Filter search with tags and sort date wise.

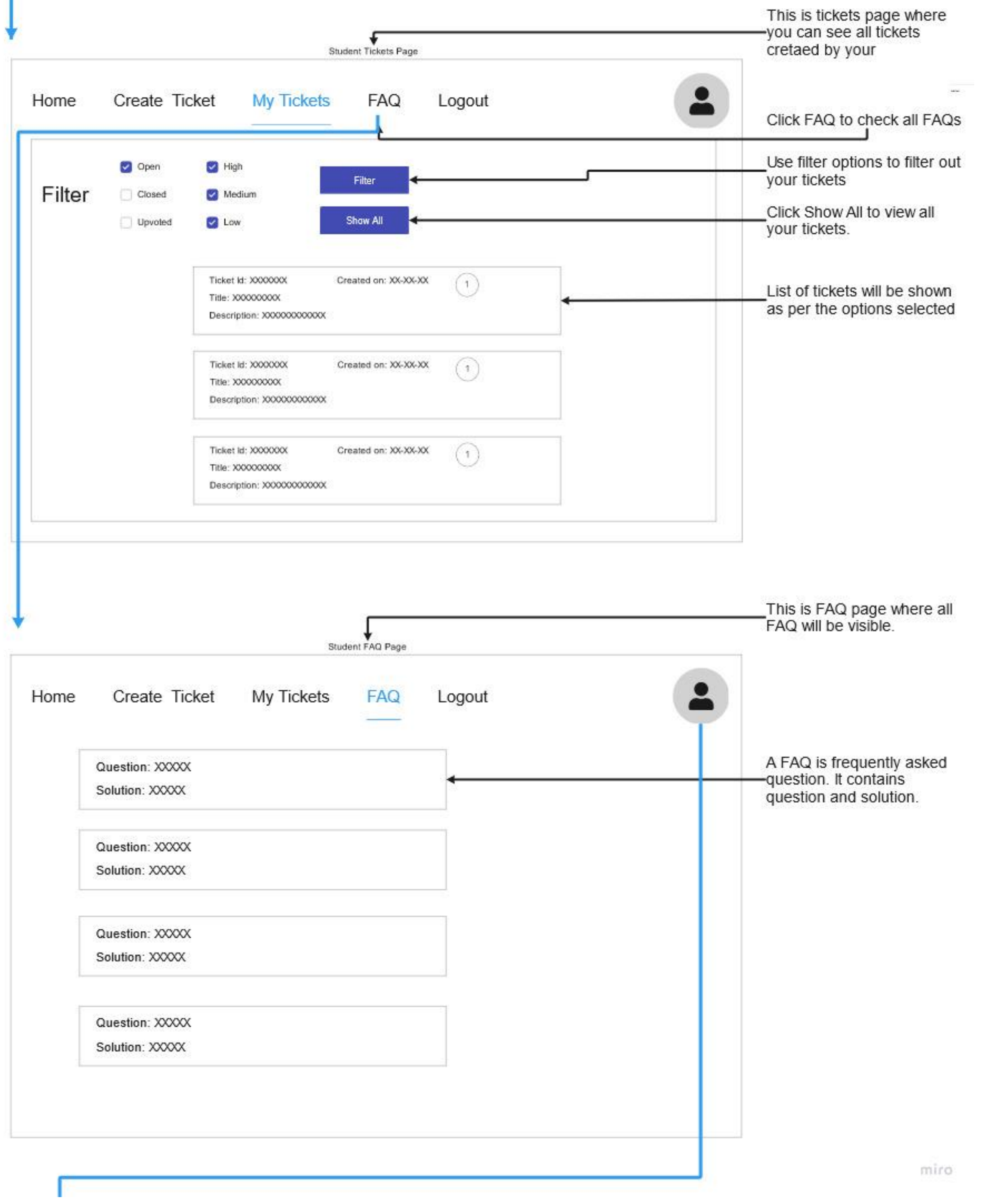
View a ticket and if its similar to your requirements you can upvote.

Enter title, description, and add appropriate tags

Add attachments (images) and select priority level.

Click submit to create a ticket

Storyboard for User: Student



This is tickets page where you can see all tickets cretaed by your

Click FAQ to check all FAQs

Use filter options to filter out your tickets

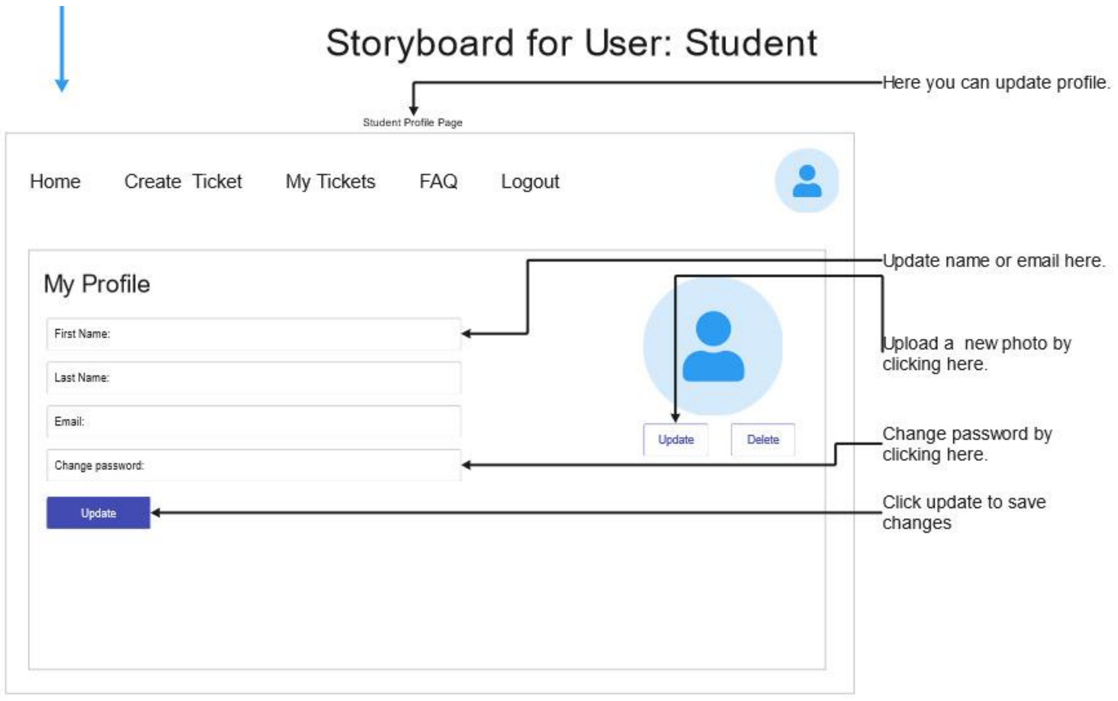
Click Show All to view all your tickets.

List of tickets will be shown as per the options selected

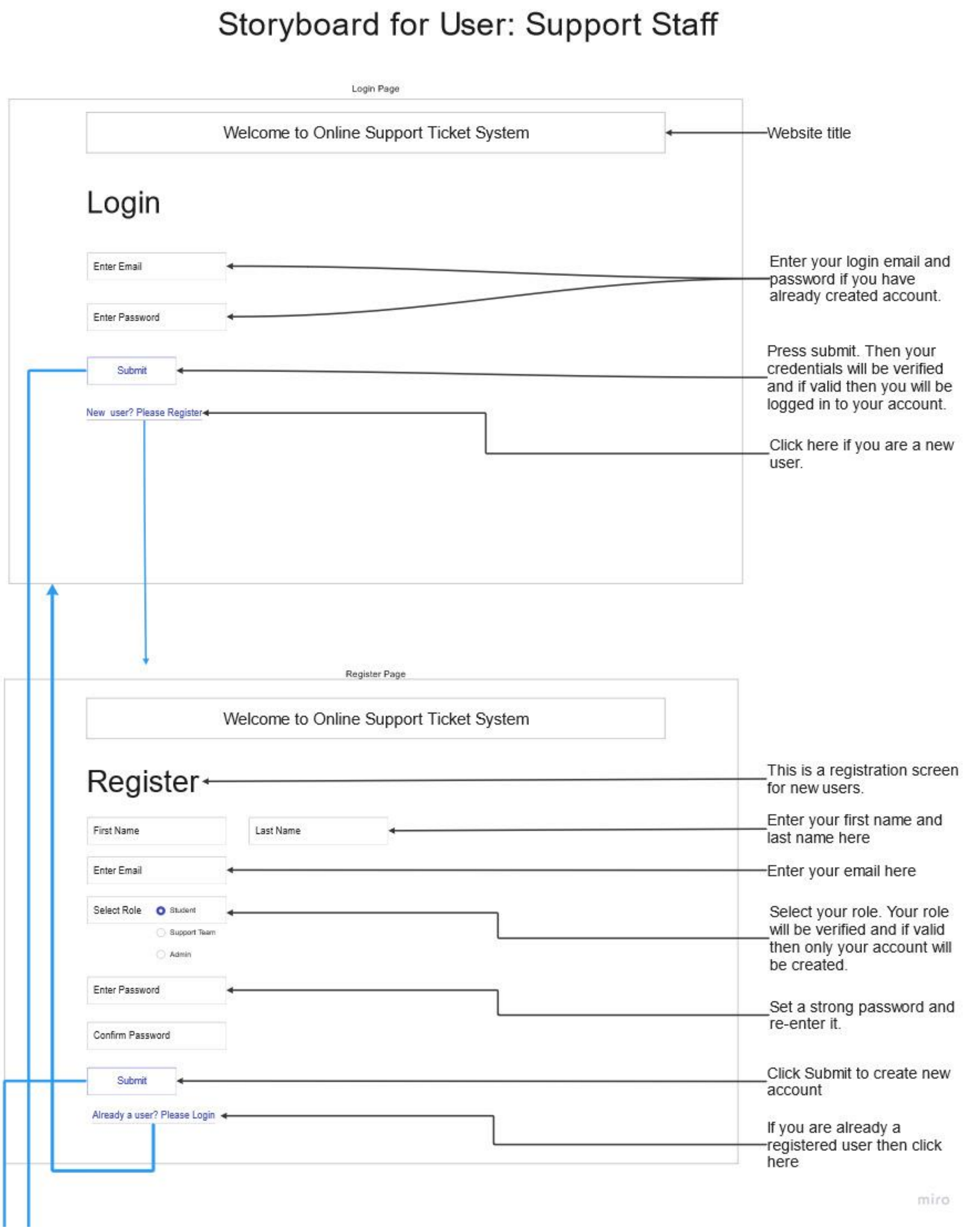
This is FAQ page where all FAQ will be visible.

A FAQ is frequently asked question. It contains question and solution.

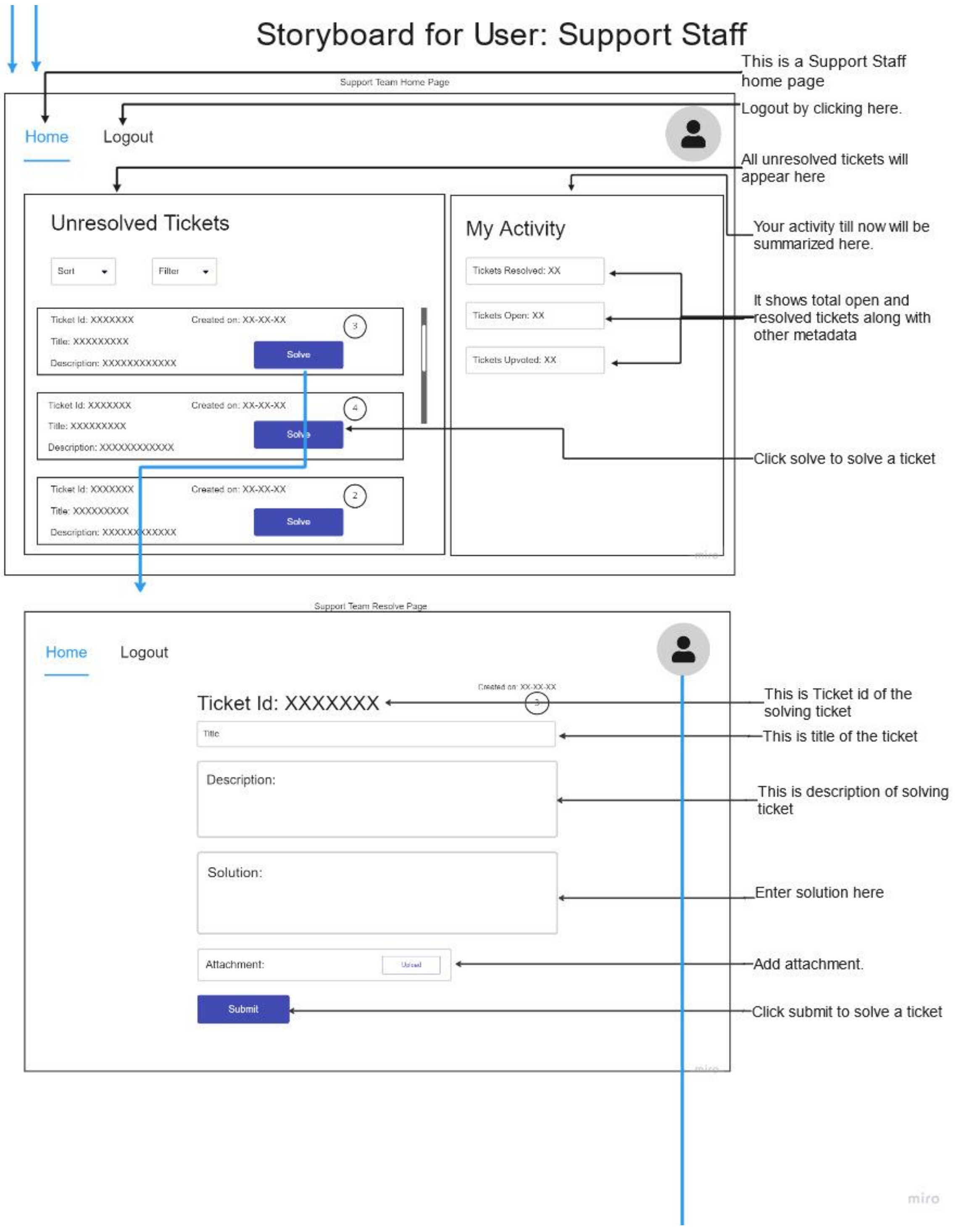
Storyboard for User: Student


Here you can update profile.
Student Profile Page
Home
Create Ticket
My Tickets
FAQ
Logout
My Profile
First Name:
Last Name:
Email:
Change password:
Update
Update
Delete
Update name or email here.
Upload a new photo by clicking here.
Change password by clicking here.
Click update to save changes

2.2.2 Storyboard for Support Staff

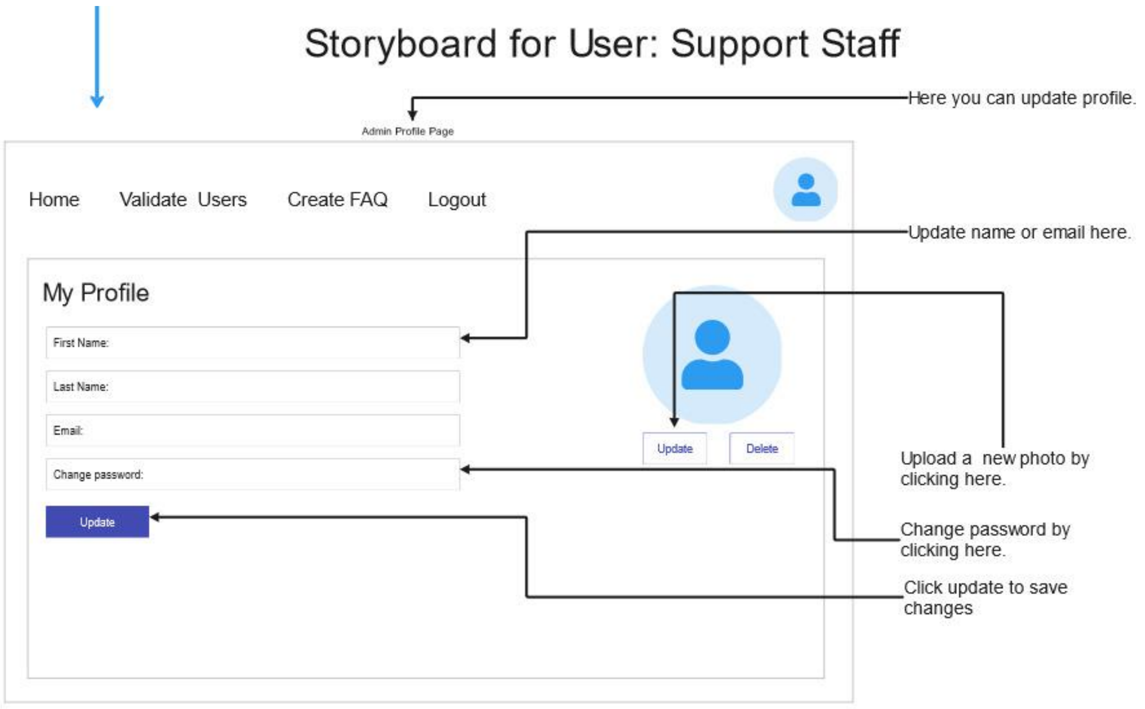

Storyboard for User: Support Staff
Login Page
Welcome to Online Support Ticket System
Website title
Login
Enter Email
Enter Password
Enter your login email and password if you have already created account.
Submit
Press submit. Then your credentials will be verified and if valid then you will be logged in to your account.
New user? Please Register
Click here if you are a new user.
Register Page
Welcome to Online Support Ticket System
Register
This is a registration screen for new users.
First Name
Last Name
Enter your first name and last name here
Enter Email
Enter your email here
Select Role
Student
Support Team
Admin
Select your role. Your role will be verified and if valid then only your account will be created.
Enter Password
Confirm Password
Set a strong password and re-enter it.
Submit
Click Submit to create new account
Already a user? Please Login
If you are already a registered user then click here
miro

Storyboard for User: Support Staff

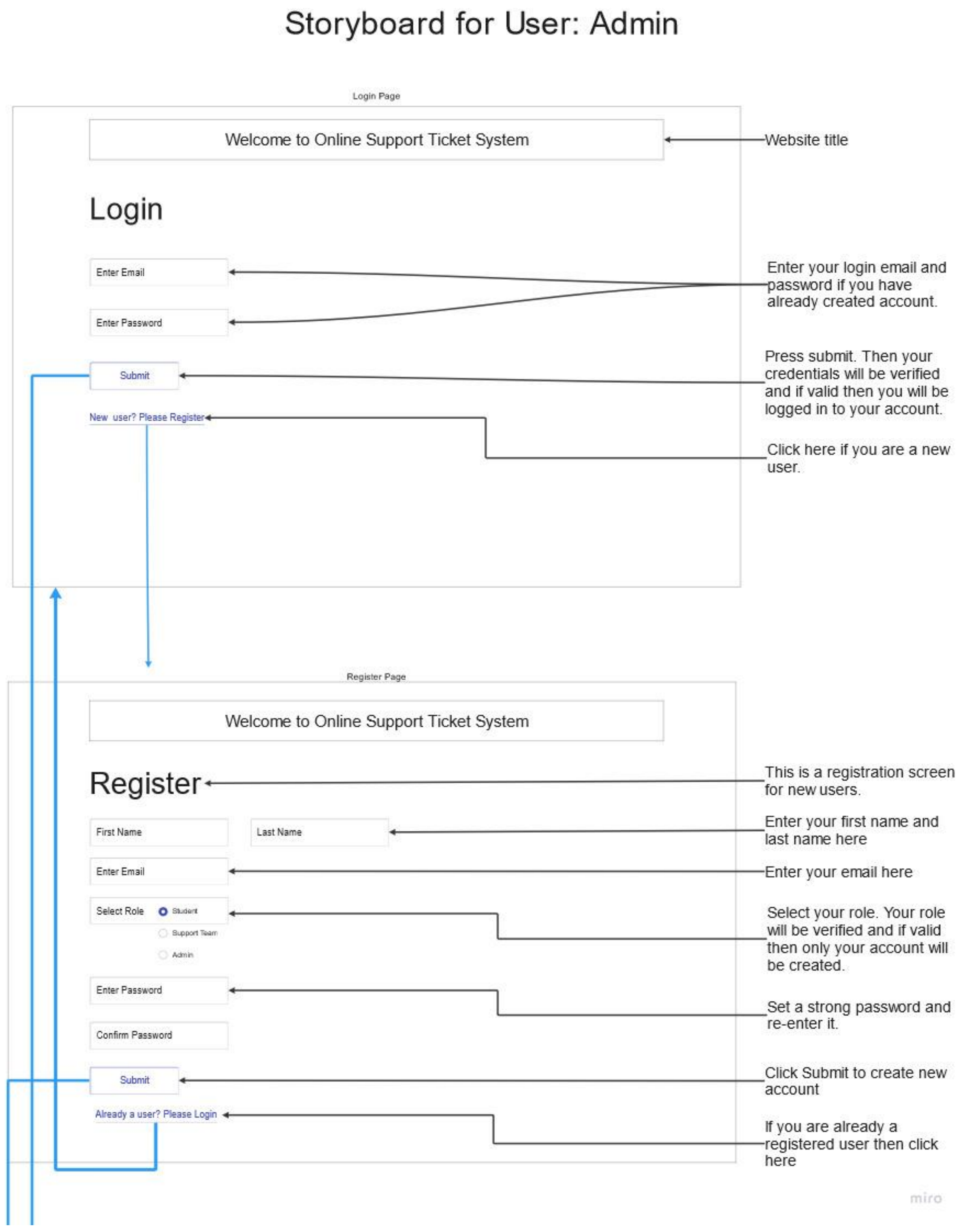


This is a Support Staff home page
Logout by clicking here.
All unresolved tickets will appear here
Your activity till now will be summarized here.
It shows total open and resolved tickets along with other metadata
Click solve to solve a ticket
Support Team Home Page
Home
Logout
Unresolved Tickets
Sort
Filter
Ticket Id: XXXXXXX
Created on: XX-XX-XX
Title: XXXXXXXXX
Description: XXXXXXXXXXXX
Solve
My Activity
Tickets Resolved: XX
Tickets Open: XX
Tickets Upvoted: XX
Support Team Resolve Page
Home
Logout
Ticket Id: XXXXXXX
Created on: XX-XX-XX
Title
Description:
Solution:
Attachment:
Upload
Submit
This is Ticket id of the solving ticket
This is title of the ticket
This is description of solving ticket
Enter solution here
Add attachment.
Click submit to solve a ticket

Storyboard for User: Support Staff


Here you can update profile.
Admin Profile Page
Home
Validate Users
Create FAQ
Logout
My Profile
First Name:
Last Name:
Email:
Change password:
Update
Update
Delete
Update name or email here.
Upload a new photo by clicking here.
Change password by clicking here.
Click update to save changes

2.2.3 Storyboard for Admin



Enter your login email and password if you have already created account.

Press submit. Then your credentials will be verified and if valid then you will be logged in to your account.

Click here if you are a new user.

This is a registration screen for new users.

Enter your first name and last name here

Enter your email here

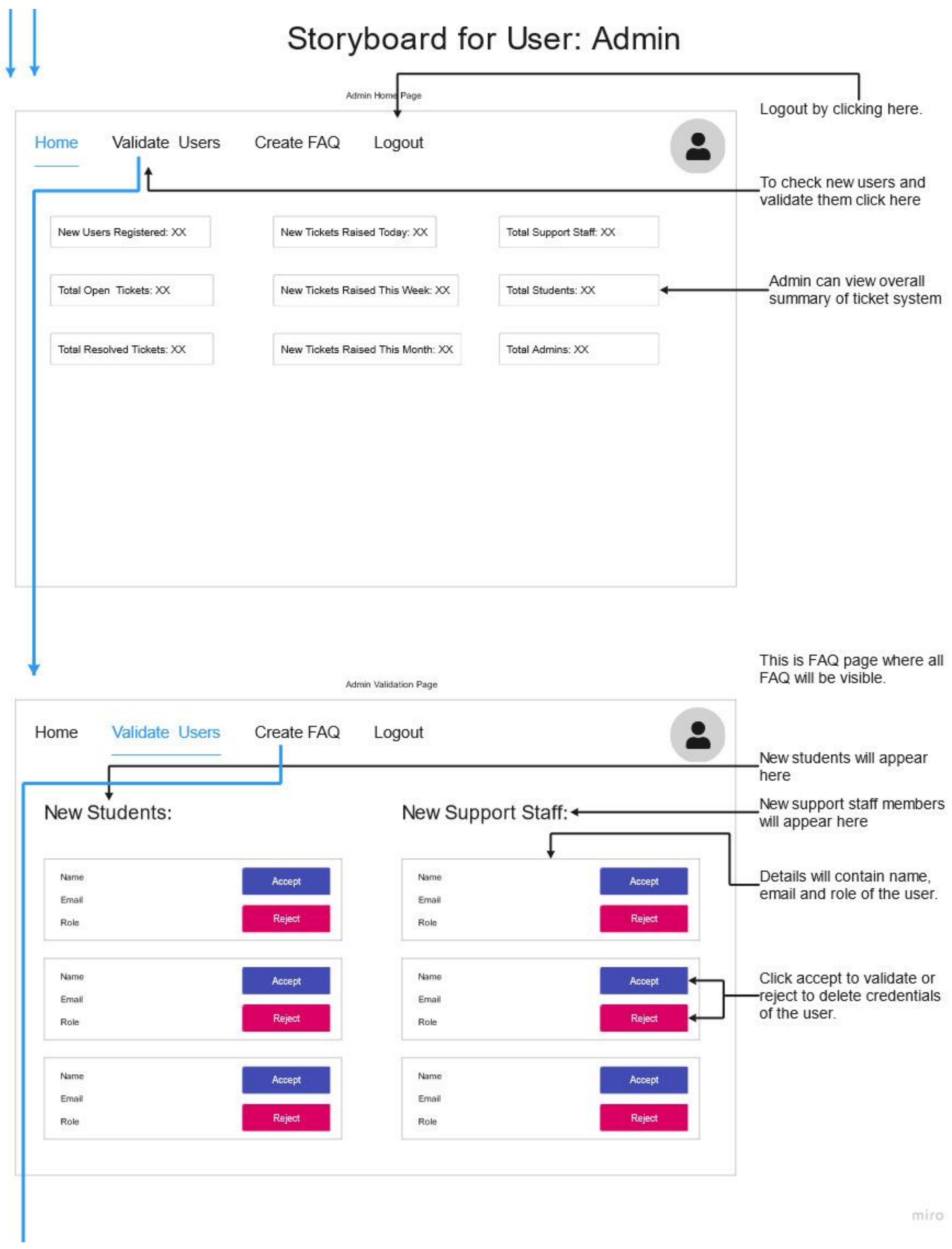
Select your role. Your role will be verified and if valid then only your account will be created.

Set a strong password and re-enter it.

Click Submit to create new account

If you are already a registered user then click here

Storyboard for User: Admin



Logout by clicking here.

To check new users and validate them click here

Admin can view overall summary of ticket system

This is FAQ page where all FAQ will be visible.

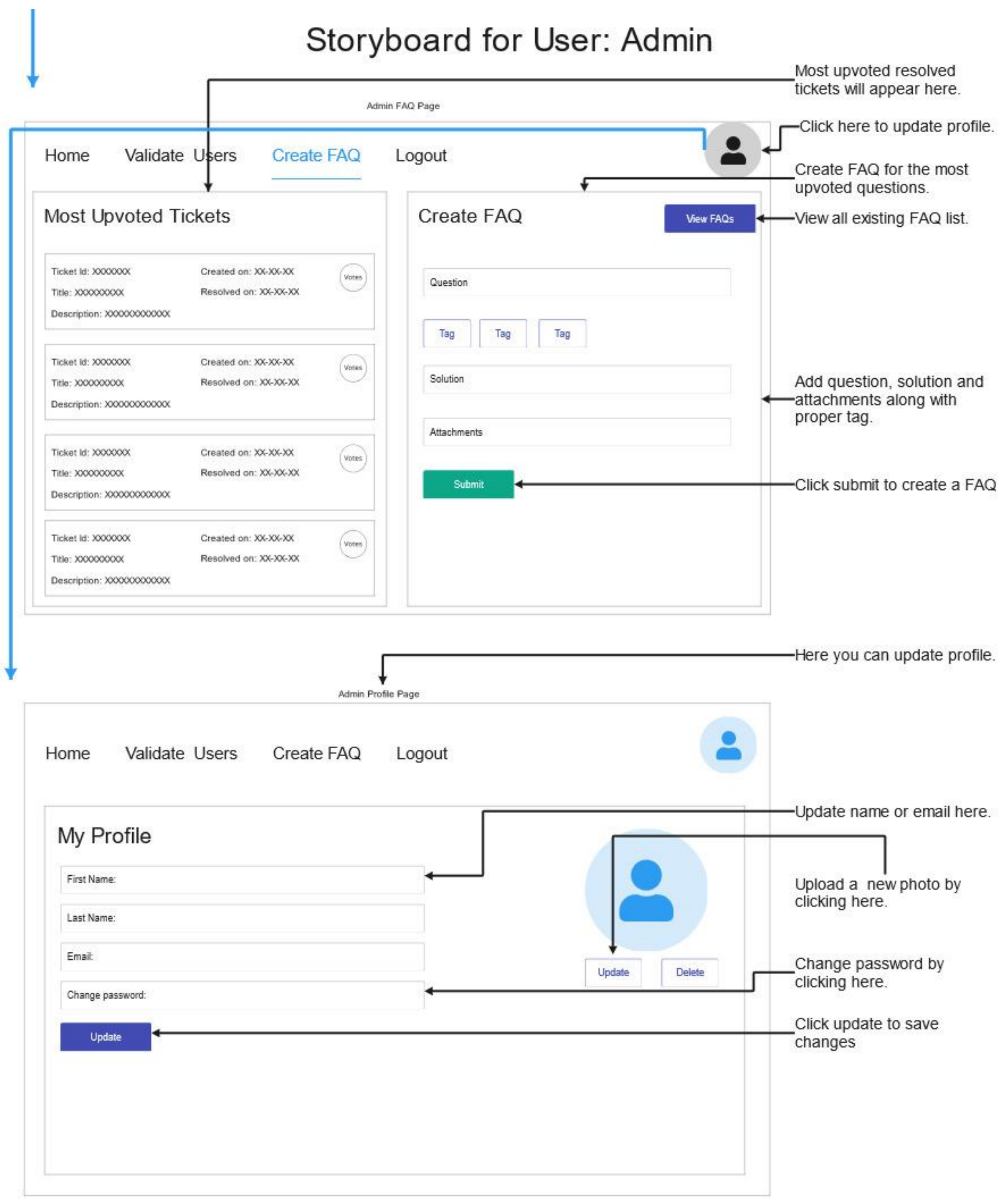
New students will appear here

New support staff members will appear here

Details will contain name, email and role of the user.

Click accept to validate or reject to delete credentials of the user.

Storyboard for User: Admin


Admin FAQ Page
Home
Validate Users
Create FAQ
Logout
Most Upvoted Tickets
Ticket Id: XXXXXXX
Title: XXXXXXXXX
Description: XXXXXXXXXXXX
Created on: XX-XX-XX
Resolved on: XX-XX-XX
Votes
Create FAQ
View FAQs
Question
Tag
Tag
Tag
Solution
Attachments
Submit
Most upvoted resolved tickets will appear here.
Click here to update profile.
Create FAQ for the most upvoted questions.
View all existing FAQ list.
Add question, solution and attachments along with proper tag.
Click submit to create a FAQ
Here you can update profile.
Admin Profile Page
Home
Validate Users
Create FAQ
Logout
My Profile
First Name:
Last Name:
Email:
Change password:
Update
Update
Delete
Update name or email here.
Upload a new photo by clicking here.
Change password by clicking here.
Click update to save changes

MILESTONE : 3
SCHEDULING AND DESIGN

3. Scheduling and Design

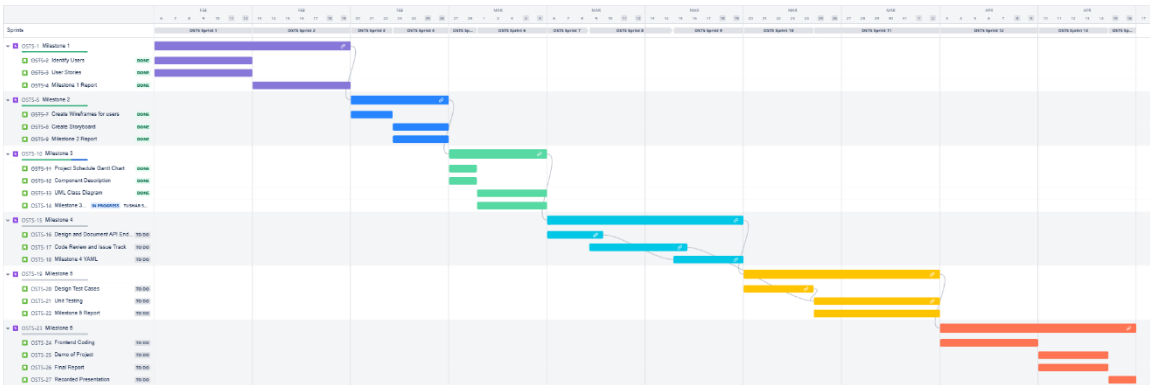
3.1. Project Schedule

3.1.1 Task Distribution

Milestone	Sub-Tasks	Sprint	Assigned To
1 – User Requirements	Identify Users	1	Tushar
	User Stories	1	Vaidehi
	Report	2	Tushar
2 – User Interfaces	Wireframe	3	Tushar
	Storyboard	4	Vaidehi
	Report	4	Tushar
3 – Scheduling	Project Schedule	5	Tushar
	Component Design	5	Vaidehi
	Class Diagram	6	Tushar
	Report	6	Tushar, Vaidehi
4 – API	Design API	7	Tushar, Vaidehi
	Code Review	8	Tushar, Vaidehi
	YAML Document	9	Tushar, Vaidehi
5 – Testing	Test Cases Design	10	Tushar
	Unit Testing	11	Vaidehi
	Report	11	Tushar, Vaidehi
6 – Submission	Frontend Design	12	Tushar, Vaidehi
	Demo	13	Tushar, Vaidehi
	Report	13	Tushar, Vaidehi
	Presentation	14	Tushar, Vaidehi

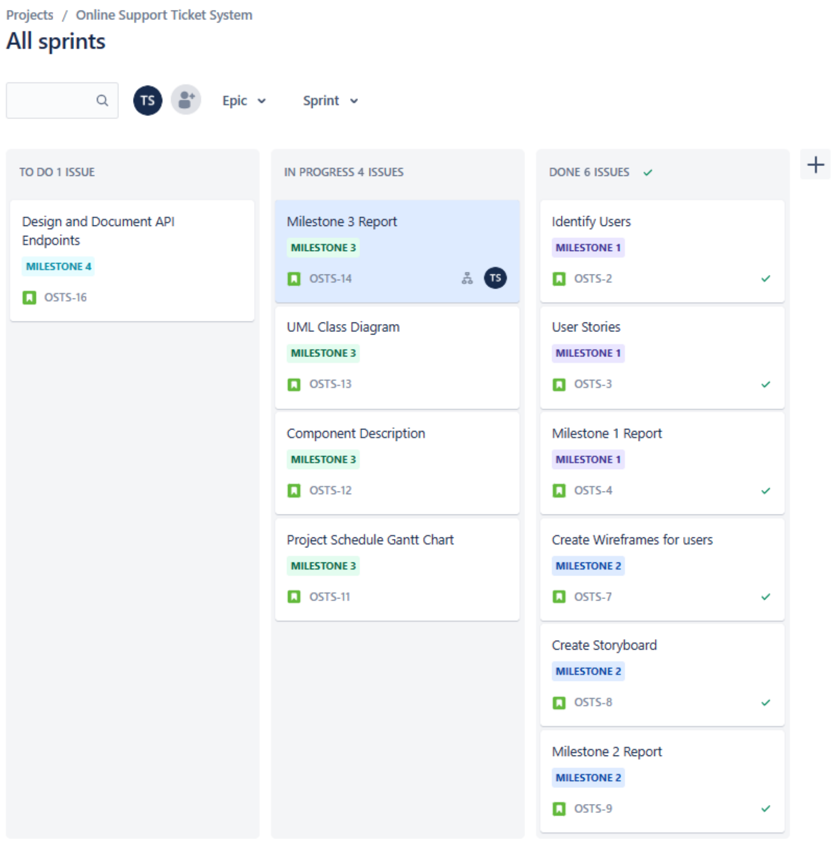
Each milestone is divided into sub tasks with SMART guidelines and assigned to each of the team member evenly. The sprints are schedules such that the dependency of each components is satisfied. The Gantt-Chart for the project schedule is shown as below. For the high resolution image of the chart, please click here

Gantt Chart of Schedule



The scrum board consists of 'To Do', 'In Progress' and 'Done' for the active sprint. The scrum board for the milestones till now (completed and in progress) is shown below.

Scrum Board (Till Milestone 3)



Projects / Online Support Ticket System

All sprints

TS Epic Sprint

TO DO 1 ISSUE

Design and Document API Endpoints
MILESTONE 4
OSTS-16

IN PROGRESS 4 ISSUES

Milestone 3 Report
MILESTONE 3
OSTS-14 TS

UML Class Diagram
MILESTONE 3
OSTS-13

Component Description
MILESTONE 3
OSTS-12

Project Schedule Gantt Chart
MILESTONE 3
OSTS-11

DONE 6 ISSUES ✓

Identify Users
MILESTONE 1
OSTS-2 ✓

User Stories
MILESTONE 1
OSTS-3 ✓

Milestone 1 Report
MILESTONE 1
OSTS-4 ✓

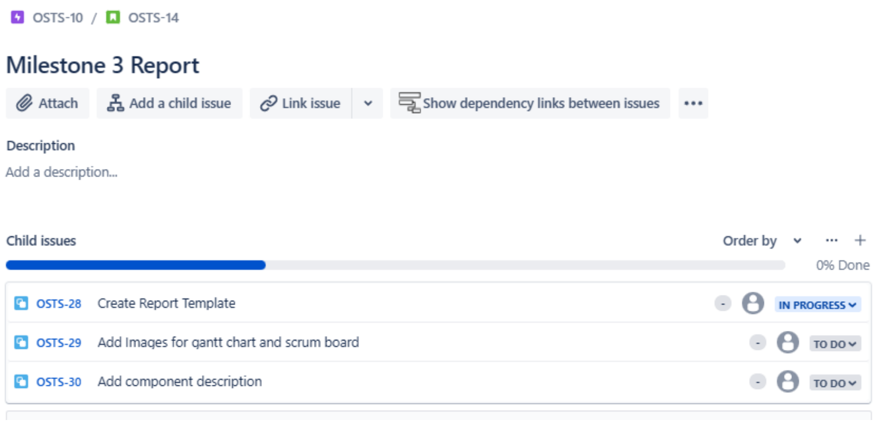
Create Wireframes for users
MILESTONE 2
OSTS-7 ✓

Create Storyboard
MILESTONE 2
OSTS-8 ✓

Milestone 2 Report
MILESTONE 2
OSTS-9 ✓

Each sub-task is divided into even smaller tasks when they are added to 'In Progress' tab. A sample is shown below.

Dividing Sub-Tasks


OSTS-10 / OSTS-14

Milestone 3 Report

Attach | Add a child issue | Link issue | Show dependency links between issues

Description

Add a description...

Child issues — Order by — 0% Done

OSTS-28	Create Report Template	IN PROGRESS
OSTS-29	Add Images for gantt chart and scrum board	TO DO
OSTS-30	Add component description	TO DO

3.2. Scrum Meetings

The sprint schedule is shown in Gantt-Chart. 'Jira' is used for scheduling the project. The scrum meetings are scheduled at the start of every sub-task at 9 PM to 10 PM (preferably on Saturday or Sunday).

The details of few scrum meeting are summarised below.

Scrum Meeting for User Identification

Agenda	Discussion
• Discuss project overview • Understand problem statement • Identify users • Distribute user stories	During the meeting, we discussed the project problem statement. We listed down exact requirements for the project as per the problem statement. Then we discussed the potential users and categorised them into different types. Then we divided the tasks. Tushar was tasked with generating report and the user stories for the admin. Vaidehi was tasked with user stories for student and support staff.

Scrum Meeting for User Interfaces

Agenda	Discussion
• Discuss what wireframes and storyboard means. • Create basic layout as a template for all wireframes • Discuss different options in wireframe. • Distribute tasks	During the meeting, we discussed the wireframe structure. Tushar was tasked with wireframes generation using ‘miro’ and Vaidehi was tasked with storyboarding. We discussed different options that should be placed in wireframes and how it will be connected to each other.

Scrum Meeting for Project Schedule

Agenda	Discussion
• Discuss major milestones • Divide milestones into sub tasks with SMART guidelines. • Decide feasibility and deadlines. • Discuss major components for the project. • Discuss what is class diagram. • Distribute tasks	During the meeting, we discussed major milestones and divided into sub tasks. The deadlines were set after assigning each task to a person and a sprint. The class diagram and Gantt chart were given to Tushar. Vaidehi decide to work on components description.

3.3. Components of the Project

The project is divided into 6 major components:

1. Student view - includes API and Frontend
2. Support view - includes API and Frontend
3. Admin view - includes API and Frontend
4. Ticket CRUD API
5. FAQ CRUD API
6. Authorization – includes user validation during login and register.

3.3.1 Components Description

The components and the short description of sub components is summarised as below.

Auth Components

- Login page (html template + Vue setup + bootstrap styling) (User can login using through email id and password)
- Register page (html template + Vue setup + bootstrap styling) (User need to fill in first and last name + unique email id + password + profile photo (optional))
- Frontend and Backend data validation for login/register (including validation of punctuations or any other symbol that can breach the security)
- Frontend store JWT and delete when expired (securing software with JWT)
- Backend Create JWT and verify for each request (Creating and verifying new and old requests)
- AuthAPI to handle login/register/logout (authenticated APIs to handle login + register + logout)
- Validate new users' registrations (sending information to the admin to accept the new user)
- Methods to send notifications (google chat webhook, email) (Once admin accept or decline the new user, sending notification about their status)

Ticket Components

- CRUD operations with Ticket API (Authenticated APIs for CRUD on Tickets)
- Multiple tickets request with API (Authenticated APIs for GET requests)
- Set up Cache for ticket request (Caching for data retrieval efficiency)
- Create Ticket page (html template + Vue setup + bootstrap styling) (User need to add title + description + priority + tags + add attachment which is optional)
- My tickets page (html template + Vue setup + bootstrap styling) (User can view all tickets or can use filters)

Student Components

- Home page (html template + Vue setup + bootstrap styling) (User can view unsolved tickets and their activity on the software)
- Update profile page (html template + Vue setup + bootstrap styling) (User can change their details except email)
- Search, sort, filter tickets for frontend (User can search, sort, filter to see specific tickets)
- Student API backend (Fetching Data and CRUD operations are implemented using authenticated APIs)

Support Staff Components

- Home page (html template + Vue setup + bootstrap styling) (User can see all the unresolved tickets and their activity + can sort and filter)
- My resolved tickets page (html template + Vue setup + bootstrap styling) (User can see all the tickets that they have solved)
- Resolve Ticket page (html template + Vue setup + bootstrap styling) (User need to add solution + attachment, if required and solve the ticket)
- Update profile and change password page (User can change their details except email)
- Backend API for support staff (Fetching Data and CRUD operations are implemented using authenticated APIs)

Admin Components

- Home page (User can see details of all the students, tickets, support staff and admins)
- Validate users page (User can accept or decline the request of new student or support staff)
- View most upvoted tickets (User can see most upvoted tickets and create FAQ)
- Update profile and change password page (User can change their details except email)
- AdminAPI for backend (Fetching Data and CRUD operations are implemented using authenticated APIs)

FAQ Components

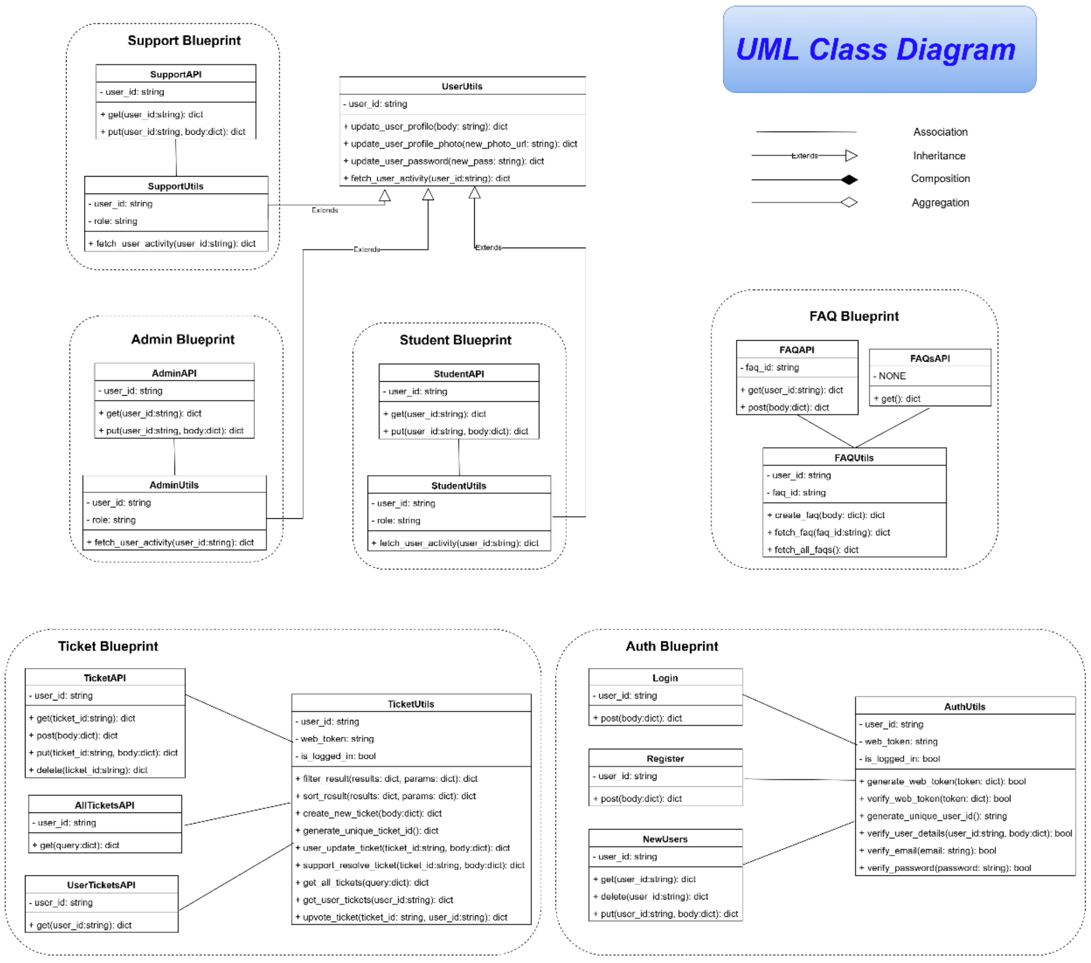
- View all FAQ page (html template + Vue setup + bootstrap styling) (See all the FAQs till now)
- Create FAQ page (html template + Vue setup + bootstrap styling) (Create FAQs for the most upvoted tickets)
- FAQ API for backend (Fetching Data and CRUD operations are implemented using authenticated APIs)

3.4. Class Diagram

Based on the above 6 major components, the UML Class diagram is created in diagrams.io tool. The components are grouped together as a blueprint. For example – *StudentAPI* (contains API end point methods like get, post, put, delete), *StudentUtils* (contains all supporting functions for student API) is grouped together as '*Student Blueprint*'.

The class diagram is shown below. For the high resolution image of the chart, please click here.

Class Diagram

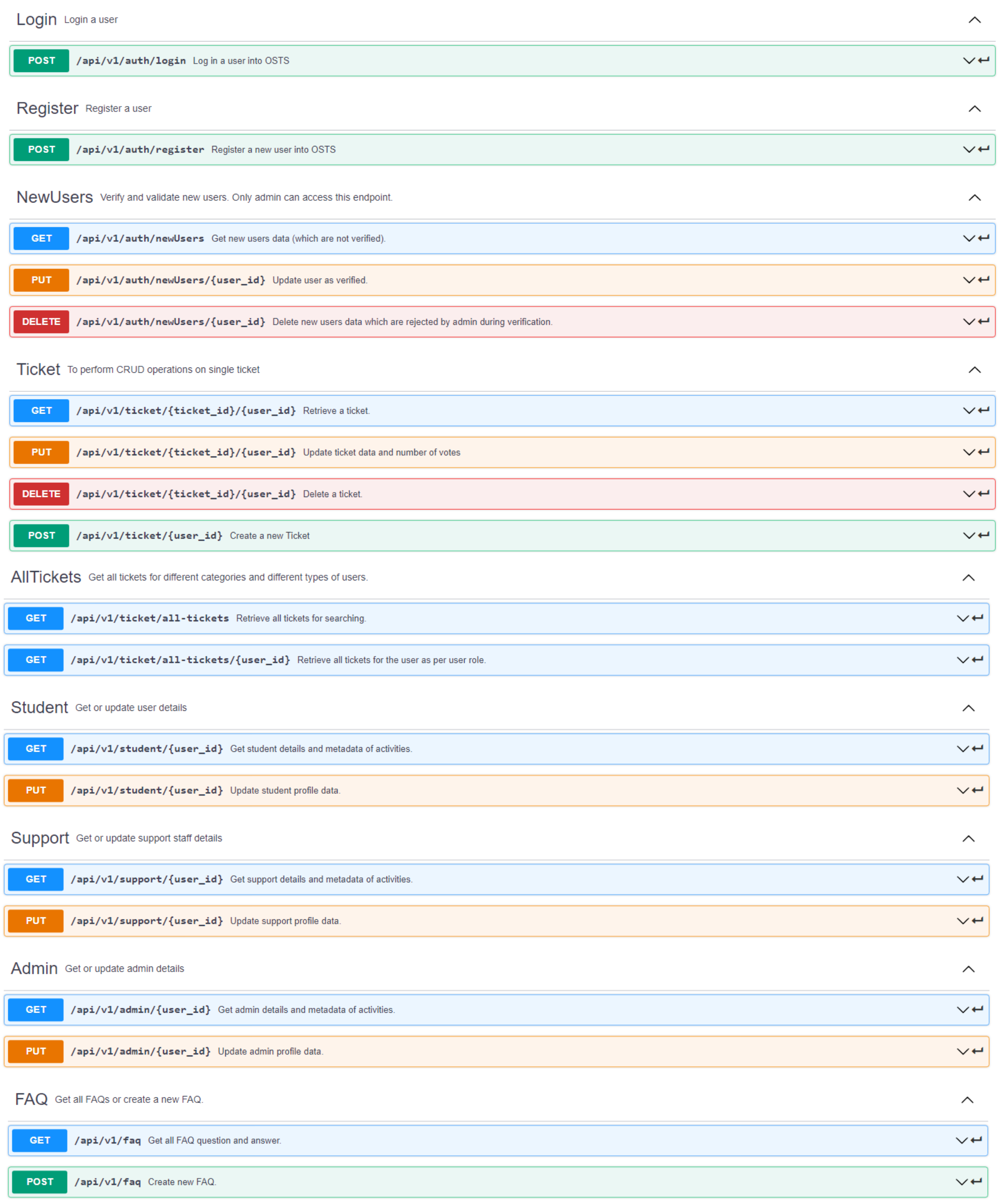

UML Class Diagram
Association
Extends
Inheritance
Composition
Aggregation
Support Blueprint
SupportAPI
- user_id: string
+ get(user_id:string): dict
+ put(user_id:string, body:dict): dict
SupportUtils
- user_id: string
- role: string
+ fetch_user_activity(user_id:string): dict
UserUtils
- user_id: string
+ update_user_profile(body: string): dict
+ update_user_profile_photo(new_photo_url: string): dict
+ update_user_password(new_pass: string): dict
+ fetch_user_activity(user_id:string): dict
Extends
Extends
Extends
Admin Blueprint
AdminAPI
- user_id: string
+ get(user_id:string): dict
+ put(user_id:string, body:dict): dict
AdminUtils
- user_id: string
- role: string
+ fetch_user_activity(user_id:string): dict
Student Blueprint
StudentAPI
- user_id: string
+ get(user_id:string): dict
+ put(user_id:string, body:dict): dict
StudentUtils
- user_id: string
- role: string
+ fetch_user_activity(user_id:string): dict
FAQ Blueprint
FAQAPI
- faq_id: string
+ get(user_id:string): dict
+ post(body:dict): dict
FAQsAPI
- NONE
+ get(): dict
FAQUtils
- user_id: string
- faq_id: string
+ create_faq(body: dict): dict
+ fetch_faq(faq_id:string): dict
+ fetch_all_faqs(): dict
Ticket Blueprint
TicketAPI
- user_id: string
+ get(ticket_id:string): dict
+ post(body:dict): dict
+ put(ticket_id:string, body:dict): dict
+ delete(ticket_id:string): dict
AllTicketsAPI
- user_id: string
+ get(query:dict): dict
UserTicketsAPI
- user_id: string
+ get(user_id:string): dict
TicketUtils
- user_id: string
- web_token: string
- is_logged_in: bool
+ filter_result(results: dict, params: dict): dict
+ sort_result(results: dict, params: dict): dict
+ create_new_ticket(body:dict): dict
+ generate_unique_ticket_id(): dict
+ user_update_ticket(ticket_id:string, body:dict): dict
+ support_resolve_ticket(ticket_id:string, body:dict): dict
+ get_all_tickets(query:dict): dict
+ get_user_tickets(user_id:string): dict
+ upvote_ticket(ticket_id: string, user_id:string): dict
Auth Blueprint
Login
- user_id: string
+ post(body:dict): dict
Register
- user_id: string
+ post(body:dict): dict
NewUsers
- user_id: string
+ get(user_id:string): dict
+ delete(user_id:string): dict
+ put(user_id:string, body:dict): dict
AuthUtils
- user_id: string
- web_token: string
- is_logged_in: bool
+ generate_web_token(token: dict): bool
+ verify_web_token(token: dict): bool
+ generate_unique_user_id(): string
+ verify_user_details(user_id:string, body:dict): bool
+ verify_email(email: string): bool
+ verify_password(password: string): bool

MILESTONE : 4
API DOCUMENTATION

4. API Documentation

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

API Routes



Login Login a user

POST `/api/v1/auth/login` Log in a user into OSTS

Register Register a user

POST `/api/v1/auth/register` Register a new user into OSTS

NewUsers Verify and validate new users. Only admin can access this endpoint.

GET `/api/v1/auth/newUsers` Get new users data (which are not verified).

PUT `/api/v1/auth/newUsers/{user_id}` Update user as verified.

DELETE `/api/v1/auth/newUsers/{user_id}` Delete new users data which are rejected by admin during verification.

Ticket To perform CRUD operations on single ticket

GET `/api/v1/ticket/{ticket_id}/{user_id}` Retrieve a ticket.

PUT `/api/v1/ticket/{ticket_id}/{user_id}` Update ticket data and number of votes

DELETE `/api/v1/ticket/{ticket_id}/{user_id}` Delete a ticket.

POST `/api/v1/ticket/{user_id}` Create a new Ticket

AllTickets Get all tickets for different categories and different types of users.

GET `/api/v1/ticket/all-tickets` Retrieve all tickets for searching.

GET `/api/v1/ticket/all-tickets/{user_id}` Retrieve all tickets for the user as per user role.

Student Get or update user details

GET `/api/v1/student/{user_id}` Get student details and metadata of activities.

PUT `/api/v1/student/{user_id}` Update student profile data.

Support Get or update support staff details

GET `/api/v1/support/{user_id}` Get support details and metadata of activities.

PUT `/api/v1/support/{user_id}` Update support profile data.

Admin Get or update admin details

GET `/api/v1/admin/{user_id}` Get admin details and metadata of activities.

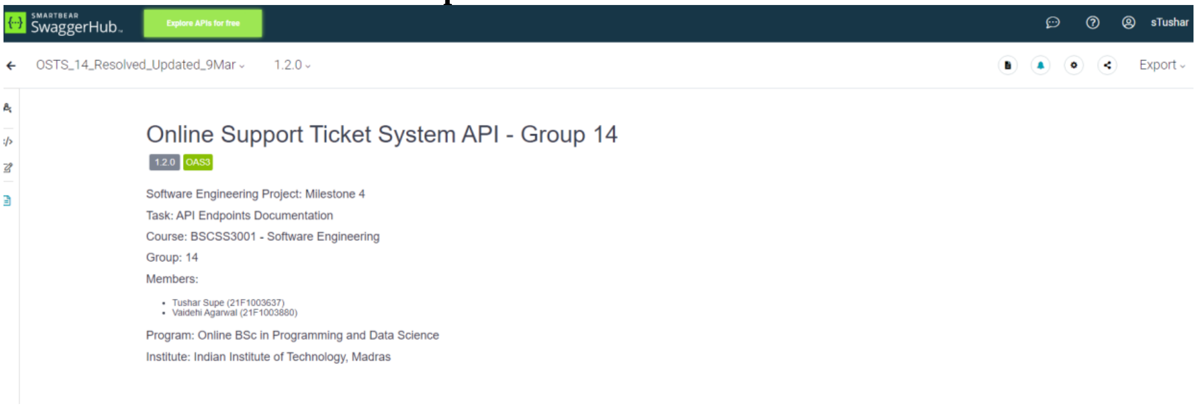
PUT `/api/v1/admin/{user_id}` Update admin profile data.

FAQ Get all FAQs or create a new FAQ.

GET `/api/v1/faq` Get all FAQ question and answer.

POST `/api/v1/faq` Create new FAQ.

Sample Screenshot of API Doc



Online Support Ticket System API - Group 14

1.2.0 OAS3

Software Engineering Project: Milestone 4

Task: API Endpoints Documentation

Course: BSCSS3001 - Software Engineering

Group: 14

Members:

- Tushar Supe (21F1003637)
- Vaidehi Agarwal (21F1003880)

Program: Online BSc in Programming and Data Science

Institute: Indian Institute of Technology, Madras

API Document Links:

Swagger API Doc: click here

YAML file link: click here

MILESTONE : 5
TEST CASES

5. Software Testing

5.1. Testing preparation

5.1.1 Setup

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

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

For this project few of the unit tests are considered. These unit tests consist of testing API endpoints for ticketing system, user management system as well as some common utility functions. To carry out testing, 'PyTest' python library is used. Required test fixture is built and sample database is built to start and facilitate these unit tests independent of other components.

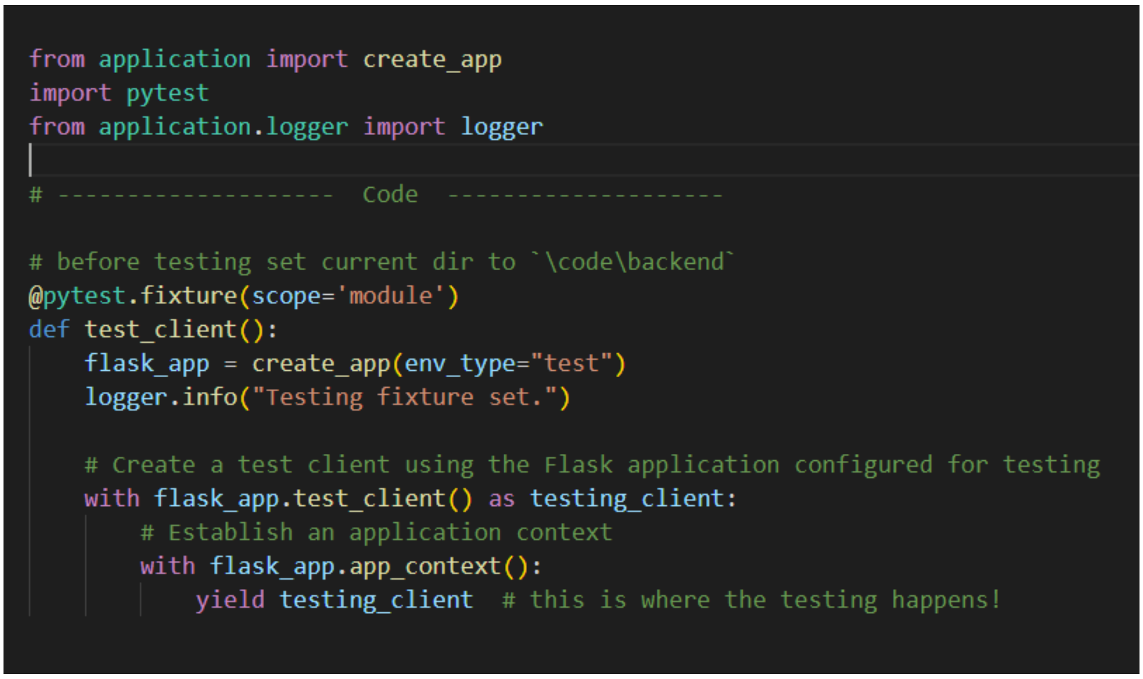
The unit tests for the project are divided into following major components:

- Auth component testing
- Users component testing
- Ticket component testing
- Common utility functions testing

5.1.2 Sample Fixture

The following figure shows sample fixture used for testing purpose. This fixture starts the app with test configuration and within app context the tests are carried out.

'Create App' Fixture



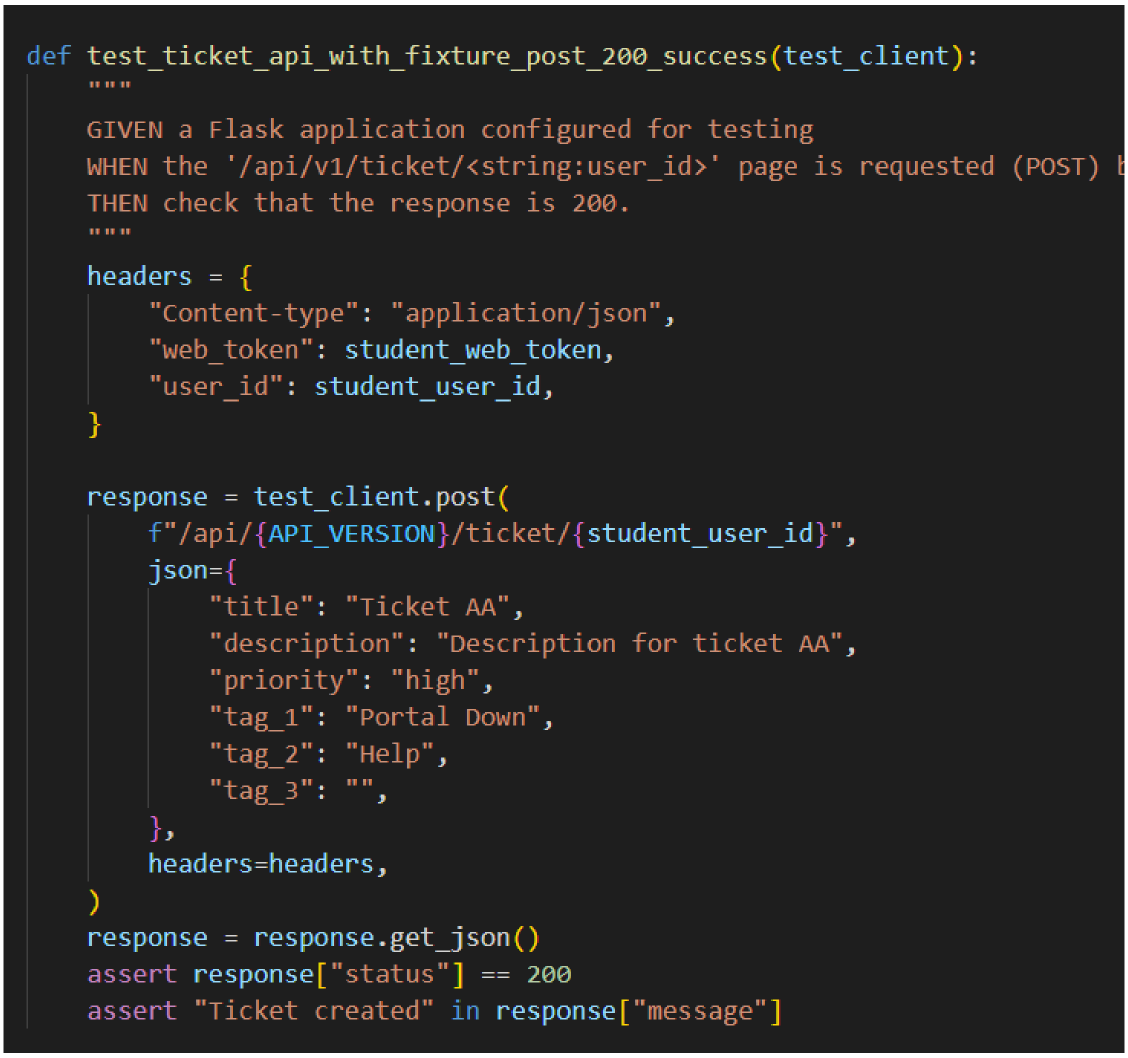
```python
from application import create_app
import pytest
from application.logger import logger

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

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

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

5.1.3 Sample Test

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

The following figure shows sample test code.

Sample Test Code



```python
def test_ticket_api_with_fixture_post_200_success(test_client):
    """
    GIVEN a Flask application configured for testing
    WHEN the '/api/v1/ticket/<string:user_id>' page is requested (POST) b
    THEN check that the response is 200.
    """
    headers = {
        "Content-type": "application/json",
        "web_token": student_web_token,
        "user_id": student_user_id,
    }

    response = test_client.post(
        f"/api/{API_VERSION}/ticket/{student_user_id}",
        json={
            "title": "Ticket AA",
            "description": "Description for ticket AA",
            "priority": "high",
            "tag_1": "Portal Down",
            "tag_2": "Help",
            "tag_3": "",
        },
        headers=headers,
    )
    response = response.get_json()
    assert response["status"] == 200
    assert "Ticket created" in response["message"]
```

5.2. Test Descriptions

5.2.1 Auth Component Testing

The tests are described as follows:

Test: GET Request on Register Page

Description	GIVEN a Flask application configured for testing WHEN the '/api/v1/auth/register' page is requested (GET) THEN check that the response is 405 i.e. method not allowed as no get method is defined for that endpoint
Page Being Tested	'/api/v1/auth/register'

Inputs	test_client, headers (headers contains user id and web token for valid request verification for all requests to API)
Expected Output	Status Code: 405 # method not allowed
Actual Output	Status Code: 405
Results	Pass

Test: POST Request on Register Page

Description	GIVEN a Flask application configured for testing WHEN the '/api/v1/auth/register' page is requested (POST) with empty data fields THEN check that the response is 400 i.e. bad request
Page Being Tested	'/api/v1/auth/register'

Inputs	test_client, headers,
	json= {
	"first_name": "",
	}

Expected Output	Status Code: 400 Message: first_name is empty or invalid
Actual Output	Status Code: 400 Message: first_name is empty or invalid
Results	Pass

Test: POST Request on Register Page

Description	GIVEN a Flask application configured for testing WHEN the '/api/v1/auth/register' page is requested (POST) with all correctly filled data fields for a new user THEN check that the response is 200 i.e. the account is created successfully
Page Being Tested	'/api/v1/auth/register'

Inputs	test_client, headers,
	json={
	"first_name": "tushar",
	"last_name": "",
	"email": tushar@gmail.com,
	"password": "1234",
	"retype_password": "1234",
	"role": "student",
	}

Expected Output	Status Code: 200 # account created successfully
Actual Output	Status Code: 200
Results	Pass

Test: POST Request on Register Page

Description	GIVEN a Flask application configured for testing

	WHEN the '/api/v1/auth/register' page is requested (POST) with already existing email id THEN check that the response is 409 i.e. Email already exists
Page Being Tested	'/api/v1/auth/register'

Inputs	test_client, headers,
	json={
	"first_name": "tushar",
	"last_name": "",
	"email": tushar@gmail.com,
	"password": "1234",
	"retype_password": "1234",
	"role": "student",
	}

Expected Output	Status Code: 409 # Email already exists
Actual Output	Status Code: 409
Results	Pass

Test: POST Request on Register Page

Description	GIVEN a Flask application configured for testing WHEN the '/api/v1/auth/register' page is requested (POST) with invalid or non-matching passwords THEN check that the response is 400.
Page Being Tested	'/api/v1/auth/register'

Inputs	test_client, headers,
	json={
	"first_name": "tushar",
	"last_name": "",
	"email": tushar@gmail.com,
	"password": "12345",
	"retype_password": "1234",
	"role": "student",
	}

Expected Output	Status Code: 400 # password not matching
Actual Output	Status Code: 400
Results	Pass

Test: POST Request on Login Page

Description	GIVEN a Flask application configured for testing WHEN the '/api/v1/auth/login' page is requested (POST) with empty fields THEN check that the response is 400.
Page Being Tested	'/api/v1/auth/ login'

Inputs	test_client, headers,
	json={
	"email": "tushar@gmail.com",
	"password": "",
	}

Expected Output	Status Code: 400 # empty fields, bad request Message: Password is empty
Actual Output	Status Code: 400 Message: Password is empty
Results	Pass

Test: POST Request on Login Page

Description	GIVEN a Flask application configured for testing WHEN the '/api/v1/auth/login' page is requested (POST) with wrong password THEN check that the response is 401
Page Being Tested	'/api/v1/auth/ login'

Inputs	test_client, headers,
	json={
	"email": "tushar@gmail.com",
	"password": "1234567",
	}

Expected Output	Status Code: 401 # unauthenticated
Actual Output	Status Code: 401
Results	Pass

Test: POST Request on Login Page

Description	GIVEN a Flask application configured for testing WHEN the '/api/v1/auth/login' page is requested (POST) with wrong email THEN check that the response is 404
Page Being Tested	'/api/v1/auth/ login'

Inputs	test_client, headers,
	json={
	"email": "tushar12345678@gmail.com",
	"password": "1234",
	}

Expected Output	Status Code: 404
Actual Output	Status Code: 404
Results	Pass

Test: POST Request on Login Page

Description	GIVEN a Flask application configured for testing WHEN the '/api/v1/auth/login' page is requested (POST) with correct user details THEN check that the response is 200 and user name is correct
Page Being Tested	'/api/v1/auth/ login'
Inputs	test_client, headers, json={ "email": "tushar_dummy@gmail.com", "password": "1234",

	}
Expected Output	Status Code: 200 # logged in successfully first_name: "dummy" # check users first name to verify if same user logged in
Actual Output	Status Code: 200 first_name: "dummy"
Results	Pass

Test: GET Request on New Users Page

Description	GIVEN a Flask application configured for testing WHEN the '/api/v1/auth/newUsers' page is requested (GET) with correct admin details THEN check that the response is 200.
Page Being Tested	'/api/v1/auth/newUsers'
Inputs	test_client, headers
Expected Output	Status Code: 200 Response Data Type: List
Actual Output	Status Code: 200 Response Data Type: List
Results	Pass

5.2.2 Common_utils Testing

Test: Backend Token Transfer Success

Description	GIVEN a Flask application configured for testing WHEN the '/api/v1/auth/newUsers' page is requested (GET) with valid token for admin THEN check that the response is 200
Page Being Tested	'/api/v1/auth/ newUsers' # any path can be chosen
Inputs	test_client, headers (headers contains token)
Expected Output	Status Code: 200
Actual Output	Status Code: 200
Results	Pass

Test: Backend Token Authentication

Description	GIVEN a Flask application configured for testing WHEN the '/api/v1/auth/newusers' page is requested (GET) with missing or invalid token THEN check that the response is 401
Page Being Tested	'/api/v1/auth/ newusers' # any path can be chosen
Inputs	test_client, headers (headers contains token)
Expected Output	Status Code: 401 Message: Token is empty or missing
Actual Output	Status Code: 401 Message: Token is empty or missing
Results	Pass

5.2.3 Users Component Testing

Test: GET Request on Student API

Description	GIVEN a Flask application configured for testing WHEN the '/api/v1/student/<string:user_id>' page is requested (GET) by student THEN check that the response is 200 and data contains student's personal data
Page Being Tested	'/api/v1/student/<string:user_id>'
Inputs	test_client, headers
Expected Output	Status Code: 200 user_id: student_user_id # related to current logged in student, refer screenshots first_name: "tushar"
Actual Output	Status Code: 200 user_id: student_user_id first_name: "tushar"
Results	Pass

Test: PUT Request on Student API

Description	GIVEN a Flask application configured for testing WHEN the '/api/v1/student/<string:user_id>' page is requested (PUT) by student to update details THEN check that the response is 200 and database contains updated data
Page Being Tested	'/api/v1/student/<string:user_id>'

Inputs	test_client, headers,
	json={
	"first_name": "tushar",
	"last_name": "supe",
	"email": "tushar@gmail.com",
	}

Expected Output	Status Code: 200 # fetch user with student_user_id from database and check its last_name last_name: "supe"
Actual Output	Status Code: 200 Auth.query.filter_by(user_id=student_user_id).first().last_name: "supe"
Results	Pass

Test: GET Request on Support API

Description	GIVEN a Flask application configured for testing WHEN the '/api/v1/support/<string:user_id>' page is requested (GET) by support

	THEN check that the response is 200 and data contains supports personal data
Page Being Tested	'/api/v1/support/<string:user_id>'
Inputs	test_client, headers
Expected Output	Status Code: 200 user_id: support_user_id # related to current logged in support member first_name: " support"
Actual Output	Status Code: 200 user_id: support_user_id first_name: " support"
Results	Pass

Test: GET Request on Admin API

Description	GIVEN a Flask application configured for testing WHEN the '/api/v1/admin/<string:user_id>' page is requested (GET) by admin THEN check that the response is 200 and data contains admins personal data
Page Being Tested	'/api/v1/admin/<string:user_id>'
Inputs	test_client, headers
Expected Output	Status Code: 200 user_id: admin_user_id # related to current logged in admin first_name: "admin"
Actual Output	Status Code: 200 user_id: admin_user_id first_name: "admin"
Results	Pass

5.2.4 Tickets Component Testing

Test: GET Request on Tickets API

Description	GIVEN a Flask application configured for testing WHEN the '/api/v1/ticket/all-tickets' page is requested (GET) by student THEN check that the response is 200 and data contains tickets details
Page Being Tested	'/api/v1/ticket/all-tickets'
Inputs	test_client, headers
Expected Output	Status Code: 200 Response Data Type: List
Actual Output	Status Code: 200 Response Data Type: List
Results	Pass

Test: GET Request on Tickets API

Description	GIVEN a Flask application configured for testing

	WHEN the '/api/v1/ticket/all-tickets' page is requested (GET) by user other than student THEN check that the response is 403 as that endpoint is only accessible to students
Page Being Tested	'/api/v1/ticket/all-tickets'
Inputs	test_client, headers
Expected Output	Status Code: 403 # No Access
Actual Output	Status Code: 403
Results	Pass

Test: GET Request on Tickets API

Description	GIVEN a Flask application configured for testing WHEN the '/api/v1/ticket/all-tickets/<string:user_id>' page is requested (GET) by student THEN check that the response is 200 and data contains tickets as per query
Page Being Tested	'/api/v1/ticket/all-tickets/<string:user_id>?filter_priority=&filter_status=pending&sortby=&sortdir=&filter_tags=’
Inputs	test_client, headers
Expected Output	Status Code: 200 Response Data Type: List All Ticket Status: Pending
Actual Output	Status Code: 200 Response Data Type: List All Ticket Status: Pending
Results	Pass

Test: GET Request on Tickets API by Student

Description	GIVEN a Flask application configured for testing WHEN the '/api/v1/ticket/all-tickets/<string:user_id>' page is requested (GET) by student THEN check that the response is 200 and data contains tickets as per query
Page Being Tested	'/api/v1/ticket/all-tickets/<string:user_id>filter_priority=low,medium&filter_status=&sortby=&sortdir=&filter_tags=’
Inputs	test_client, headers
Expected Output	Status Code: 200 Response Datatype: List All Tickets Priority: low or medium
Actual Output	Status Code: 200 Response Datatype: List All Tickets Priority: low or medium
Results	Pass

Test: GET Request on Tickets API by Support

Description	GIVEN a Flask application configured for testing WHEN the '/api/v1/ticket/all-tickets/<string:user_id>' page is requested (GET) by support THEN check that the response is 200 and data contains unresolved tickets as per query
Page Being Tested	'/api/v1/ticket/all-tickets/<string:user_id>?filter_priority=&filter_status=pending&sortby=created_on&sortdir=desc&filter_tags=’
Inputs	test_client, headers
Expected Output	Status Code: 200 Response Datatype: List Tickets sorted by date it created on: True All Tickets Status: Pending
Actual Output	Status Code: 200 Response Datatype: List Tickets sorted by date it created on: True All Tickets Status: Pending
Results	Pass

Test: GET Request on Tickets API by Admin

Description	GIVEN a Flask application configured for testing WHEN the '/api/v1/ticket/all-tickets/<string:user_id>' page is requested (GET) by admin THEN check that the response is 200 and data contains resolved tickets as per query and in descending order of votes by default
Page Being Tested	'/api/v1/ticket/all-tickets/<string:user_id>’
Inputs	test_client, headers
Expected Output	Status Code: 200 Response Datatype: List Tickets sorted by votes: True
Actual Output	Status Code: 200 Response Datatype: List Tickets sorted by votes: True
Results	Pass

Test: POST Request on Tickets API

Description	GIVEN a Flask application configured for testing WHEN the '/api/v1/ticket/<string:user_id>' page is requested (POST) by student to create a new ticket THEN check that the response is 200.
Page Being Tested	'/api/v1/ticket/<string:user_id>’
Inputs	test_client, headers, json={ "title": "Ticket AA", "description": "Description for ticket AA",

	"priority": "high", "tag_1": "Portal Down", "tag_2": "Help", "tag_3": "", }
Expected Output	Status Code: 200 Message: “Ticket created”
Actual Output	Status Code: 200 Message: “Ticket created”
Results	Pass

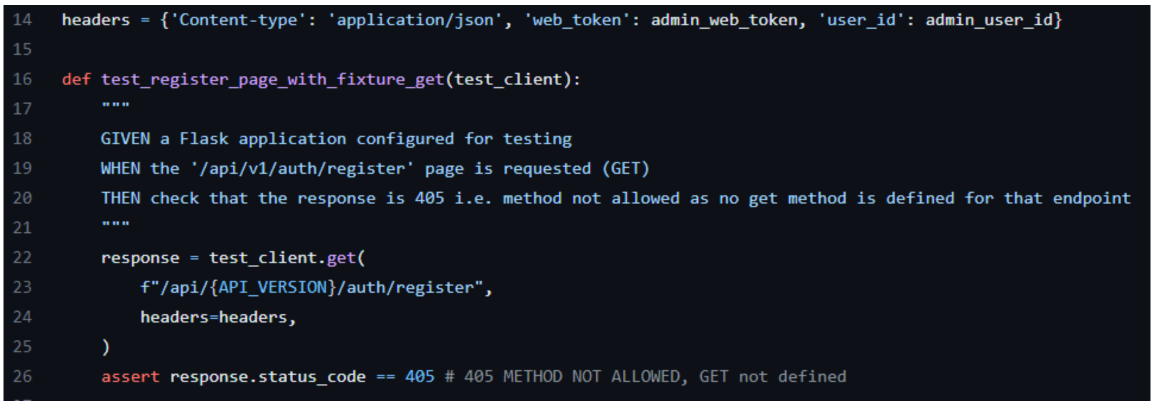
Test: GET Request on Tickets API

Description	GIVEN a Flask application configured for testing WHEN the '/api/v1/ticket/<string:ticket_id>/<string:user_id>' page is requested (GET) THEN check that the response is 200 and data contains ticket details
Page Being Tested	'/api/v1/ticket/<string:ticket_id>/<string:user_id>’
Inputs	test_client, headers
Expected Output	Status Code: 200 Ticket Id in Response: Ticket_id (sent in request URL, refer screenshot) Ticket_title: “This is ticket A”
Actual Output	Status Code: 200 Ticket Id in Response: Ticket_id Ticket_title: “This is ticket A”
Results	Pass

5.3. Final Test Summary

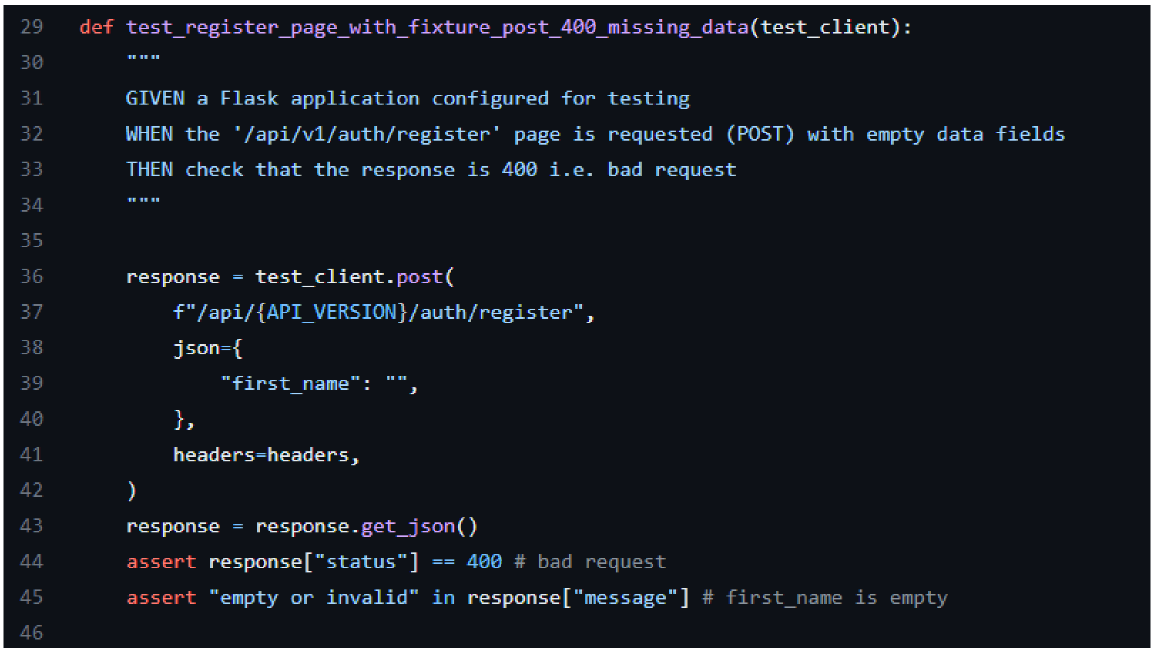
Sample screenshots of testcase code are attached below.

Sample Screenshots - 1



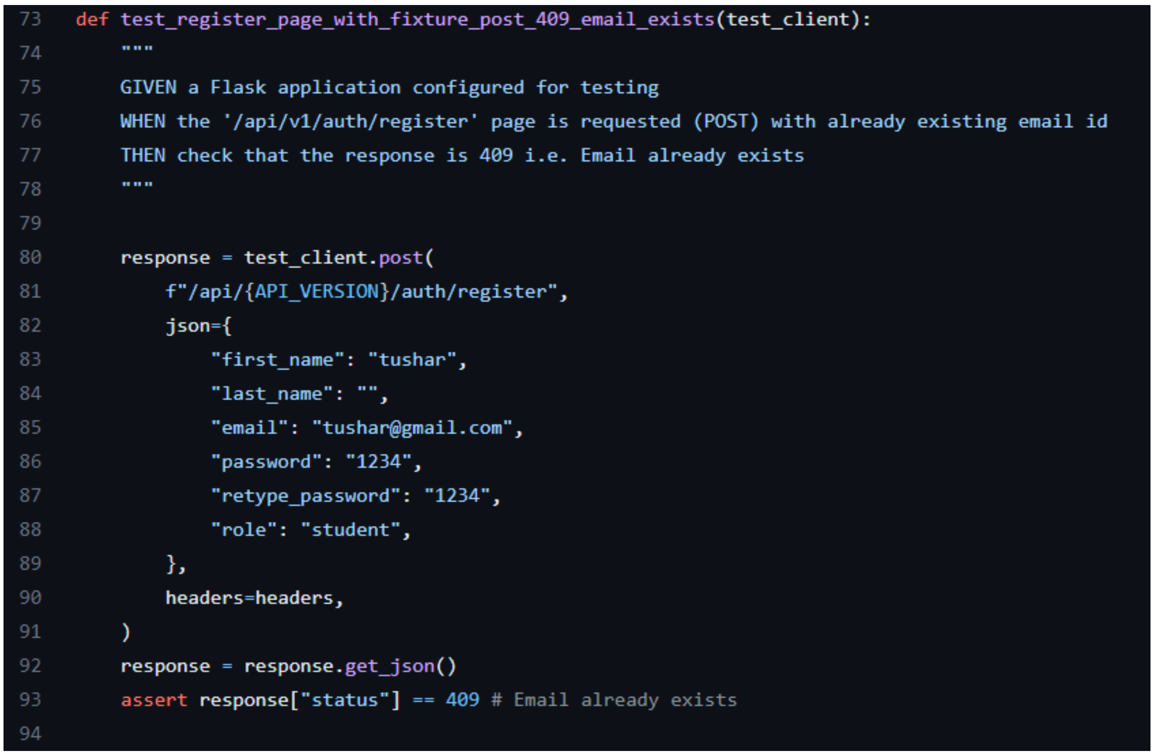
```
headers = {'Content-type': 'application/json', 'web_token': admin_web_token, 'user_id': admin_user_id}

def test_register_page_with_fixture_get(test_client):
    """
    GIVEN a Flask application configured for testing
    WHEN the '/api/v1/auth/register' page is requested (GET)
    THEN check that the response is 405 i.e. method not allowed as no get method is defined for that endpoint
    """
    response = test_client.get(
        f"/api/{API_VERSION}/auth/register",
        headers=headers,
    )
    assert response.status_code == 405 # 405 METHOD NOT ALLOWED, GET not defined
```

Sample Screenshots - 2



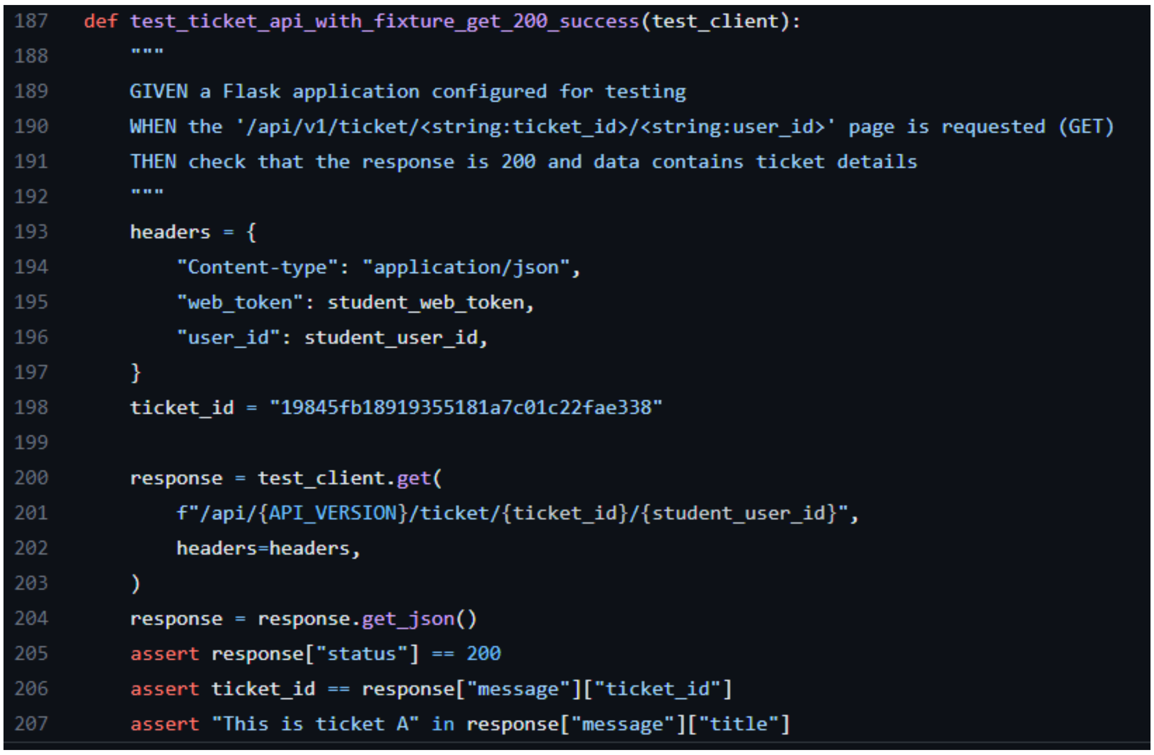
```python
def test_register_page_with_fixture_post_400_missing_data(test_client):
    """
    GIVEN a Flask application configured for testing
    WHEN the '/api/v1/auth/register' page is requested (POST) with empty data fields
    THEN check that the response is 400 i.e. bad request
    """

    response = test_client.post(
        f"/api/{API_VERSION}/auth/register",
        json={
            "first_name": "",
        },
        headers=headers,
    )
    response = response.get_json()
    assert response["status"] == 400 # bad request
    assert "empty or invalid" in response["message"] # first_name is empty

```

Sample Screenshots - 3



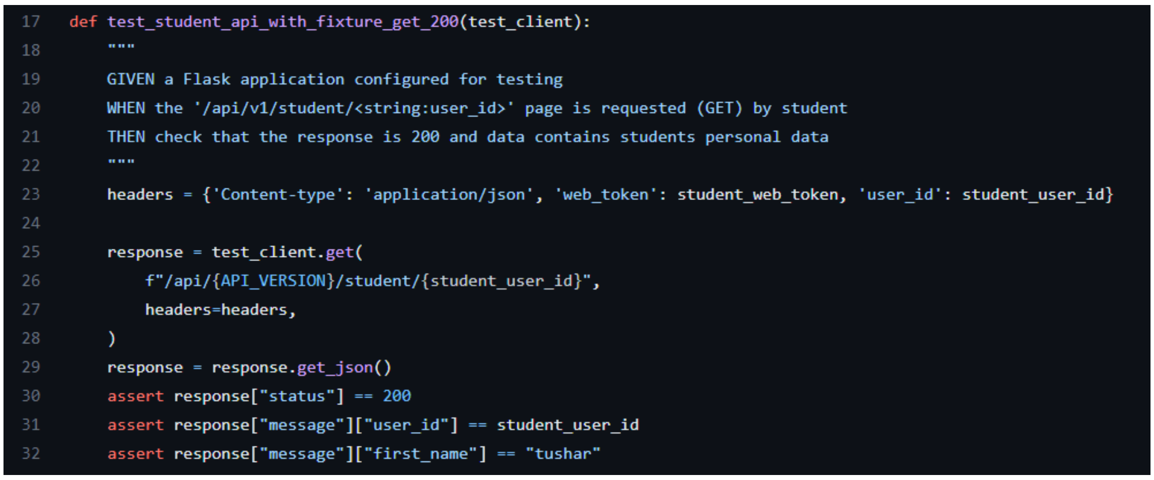
```python
def test_register_page_with_fixture_post_409_email_exists(test_client):
    """
    GIVEN a Flask application configured for testing
    WHEN the '/api/v1/auth/register' page is requested (POST) with already existing email id
    THEN check that the response is 409 i.e. Email already exists
    """

    response = test_client.post(
        f"/api/{API_VERSION}/auth/register",
        json={
            "first_name": "tushar",
            "last_name": "",
            "email": "tushar@gmail.com",
            "password": "1234",
            "retype_password": "1234",
            "role": "student",
        },
        headers=headers,
    )
    response = response.get_json()
    assert response["status"] == 409 # Email already exists

```

Sample Screenshots - 4



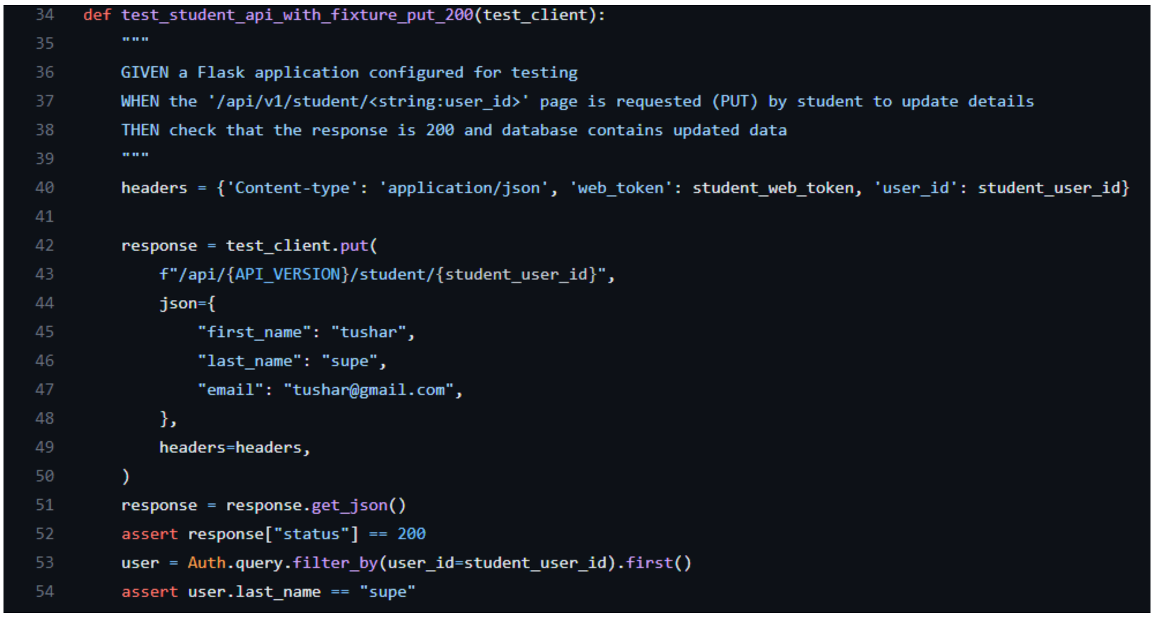
```python
def test_ticket_api_with_fixture_get_200_success(test_client):
    """
    GIVEN a Flask application configured for testing
    WHEN the '/api/v1/ticket/<string:ticket_id>/<string:user_id>' page is requested (GET)
    THEN check that the response is 200 and data contains ticket details
    """
    headers = {
        "Content-type": "application/json",
        "web_token": student_web_token,
        "user_id": student_user_id,
    }
    ticket_id = "19845fb18919355181a7c01c22fae338"

    response = test_client.get(
        f"/api/{API_VERSION}/ticket/{ticket_id}/{student_user_id}",
        headers=headers,
    )
    response = response.get_json()
    assert response["status"] == 200
    assert ticket_id == response["message"]["ticket_id"]
    assert "This is ticket A" in response["message"]["title"]
```

Sample Screenshots - 5



```python
def test_student_api_with_fixture_get_200(test_client):
    """
    GIVEN a Flask application configured for testing
    WHEN the '/api/v1/student/<string:user_id>' page is requested (GET) by student
    THEN check that the response is 200 and data contains students personal data
    """
    headers = {'Content-type': 'application/json', 'web_token': student_web_token, 'user_id': student_user_id}

    response = test_client.get(
        f"/api/{API_VERSION}/student/{student_user_id}",
        headers=headers,
    )
    response = response.get_json()
    assert response["status"] == 200
    assert response["message"]["user_id"] == student_user_id
    assert response["message"]["first_name"] == "tushar"
```

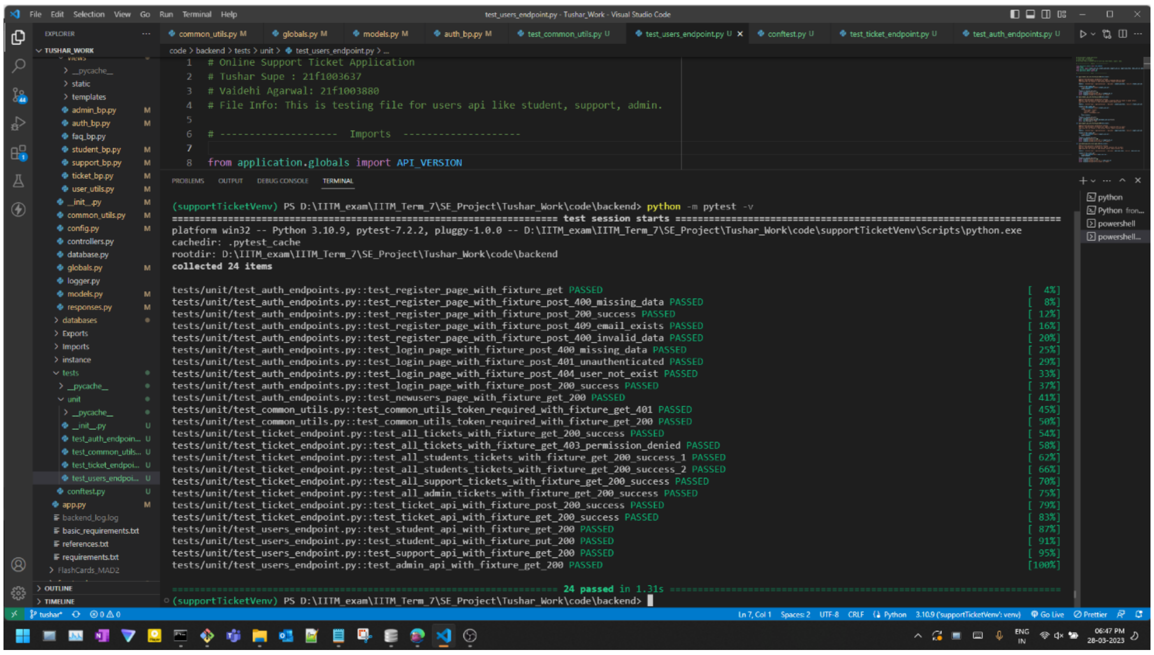
Sample Screenshots - 6



```python
def test_student_api_with_fixture_put_200(test_client):
    """
    GIVEN a Flask application configured for testing
    WHEN the '/api/v1/student/<string:user_id>' page is requested (PUT) by student to update details
    THEN check that the response is 200 and database contains updated data
    """
    headers = {'Content-type': 'application/json', 'web_token': student_web_token, 'user_id': student_user_id}

    response = test_client.put(
        f"/api/{API_VERSION}/student/{student_user_id}",
        json={
            "first_name": "tushar",
            "last_name": "supe",
            "email": "tushar@gmail.com",
        },
        headers=headers,
    )
    response = response.get_json()
    assert response["status"] == 200
    user = Auth.query.filter_by(user_id=student_user_id).first()
    assert user.last_name == "supe"
```

Following figure shows short summary of tests carried out.

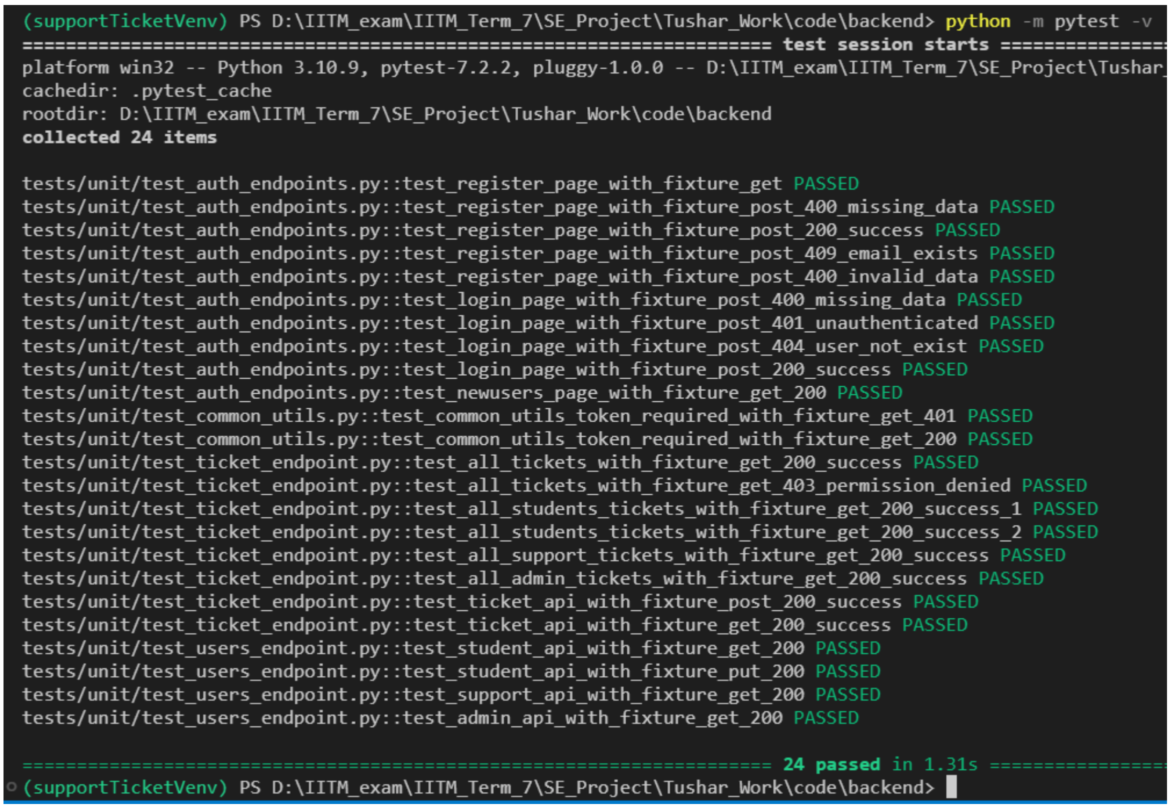
Tests Summary Screenshot



```python
# Online Support Ticket Application
# Tushar Supe : 21f1003637
# Vaidehi Agarwal: 21f1003880
# File Info: This is testing file for users api like student, support, admin.

# -------------------  Imports  -------------------

from application.globals import API_VERSION
```

```
(supportTicketVenv) PS D:\IITM_exam\IITM_Term_7\SE_Project\Tushar_Work\code\backend> python -m pytest -v
=================================================================== test session starts ===================================================================
platform win32 -- Python 3.10.9, pytest-7.2.2, pluggy-1.0.0 -- D:\IITM_exam\IITM_Term_7\SE_Project\Tushar_Work\code\supportTicketVenv\Scripts\python.exe
cachedir: .pytest_cache
rootdir: D:\IITM_exam\IITM_Term_7\SE_Project\Tushar_Work\code\backend
collected 24 items

tests/unit/test_auth_endpoints.py::test_register_page_with_fixture_get PASSED                                                               [  4%]
tests/unit/test_auth_endpoints.py::test_register_page_with_fixture_post_400_missing_data PASSED                                             [  8%]
tests/unit/test_auth_endpoints.py::test_register_page_with_fixture_post_200_success PASSED                                                  [ 12%]
tests/unit/test_auth_endpoints.py::test_register_page_with_fixture_post_409_email_exists PASSED                                             [ 16%]
tests/unit/test_auth_endpoints.py::test_register_page_with_fixture_post_400_invalid_data PASSED                                             [ 20%]
tests/unit/test_auth_endpoints.py::test_login_page_with_fixture_post_400_missing_data PASSED                                                [ 25%]
tests/unit/test_auth_endpoints.py::test_login_page_with_fixture_post_401_unauthenticated PASSED                                             [ 29%]
tests/unit/test_auth_endpoints.py::test_login_page_with_fixture_post_404_user_not_exist PASSED                                              [ 33%]
tests/unit/test_auth_endpoints.py::test_login_page_with_fixture_post_200_success PASSED                                                     [ 37%]
tests/unit/test_auth_endpoints.py::test_newusers_page_with_fixture_get_200 PASSED                                                           [ 41%]
tests/unit/test_common_utils.py::test_common_utils_token_required_with_fixture_get_401 PASSED                                               [ 45%]
tests/unit/test_common_utils.py::test_common_utils_token_required_with_fixture_get_200 PASSED                                               [ 50%]
tests/unit/test_ticket_endpoint.py::test_all_tickets_with_fixture_get_200_success PASSED                                                    [ 54%]
tests/unit/test_ticket_endpoint.py::test_all_tickets_with_fixture_get_403_permission_denied PASSED                                          [ 58%]
tests/unit/test_ticket_endpoint.py::test_all_students_tickets_with_fixture_get_200_success_1 PASSED                                         [ 62%]
tests/unit/test_ticket_endpoint.py::test_all_students_tickets_with_fixture_get_200_success_2 PASSED                                         [ 66%]
tests/unit/test_ticket_endpoint.py::test_all_support_tickets_with_fixture_get_200_success PASSED                                            [ 70%]
tests/unit/test_ticket_endpoint.py::test_all_admin_tickets_with_fixture_get_200_success PASSED                                              [ 75%]
tests/unit/test_ticket_endpoint.py::test_ticket_api_with_fixture_post_200_success PASSED                                                    [ 79%]
tests/unit/test_ticket_endpoint.py::test_ticket_api_with_fixture_get_200_success PASSED                                                     [ 83%]
tests/unit/test_users_endpoint.py::test_student_api_with_fixture_get_200 PASSED                                                             [ 87%]
tests/unit/test_users_endpoint.py::test_student_api_with_fixture_put_200 PASSED                                                             [ 91%]
tests/unit/test_users_endpoint.py::test_support_api_with_fixture_get_200 PASSED                                                             [ 95%]
tests/unit/test_users_endpoint.py::test_admin_api_with_fixture_get_200 PASSED                                                               [100%]

=================================================================== 24 passed in 1.31s ===================================================================
(supportTicketVenv) PS D:\IITM_exam\IITM_Term_7\SE_Project\Tushar_Work\code\backend>
```

Tests Summary



```
(supportTicketVenv) PS D:\IITM_exam\IITM_Term_7\SE_Project\Tushar_Work\code\backend> python -m pytest -v
===================================================================== test session starts ===============
platform win32 -- Python 3.10.9, pytest-7.2.2, pluggy-1.0.0 -- D:\IITM_exam\IITM_Term_7\SE_Project\Tushar_
cachedir: .pytest_cache
rootdir: D:\IITM_exam\IITM_Term_7\SE_Project\Tushar_Work\code\backend
collected 24 items

tests/unit/test_auth_endpoints.py::test_register_page_with_fixture_get PASSED
tests/unit/test_auth_endpoints.py::test_register_page_with_fixture_post_400_missing_data PASSED
tests/unit/test_auth_endpoints.py::test_register_page_with_fixture_post_200_success PASSED
tests/unit/test_auth_endpoints.py::test_register_page_with_fixture_post_409_email_exists PASSED
tests/unit/test_auth_endpoints.py::test_register_page_with_fixture_post_400_invalid_data PASSED
tests/unit/test_auth_endpoints.py::test_login_page_with_fixture_post_400_missing_data PASSED
tests/unit/test_auth_endpoints.py::test_login_page_with_fixture_post_401_unauthenticated PASSED
tests/unit/test_auth_endpoints.py::test_login_page_with_fixture_post_404_user_not_exist PASSED
tests/unit/test_auth_endpoints.py::test_login_page_with_fixture_post_200_success PASSED
tests/unit/test_auth_endpoints.py::test_newusers_page_with_fixture_get_200 PASSED
tests/unit/test_common_utils.py::test_common_utils_token_required_with_fixture_get_401 PASSED
tests/unit/test_common_utils.py::test_common_utils_token_required_with_fixture_get_200 PASSED
tests/unit/test_ticket_endpoint.py::test_all_tickets_with_fixture_get_200_success PASSED
tests/unit/test_ticket_endpoint.py::test_all_tickets_with_fixture_get_403_permission_denied PASSED
tests/unit/test_ticket_endpoint.py::test_all_students_tickets_with_fixture_get_200_success_1 PASSED
tests/unit/test_ticket_endpoint.py::test_all_students_tickets_with_fixture_get_200_success_2 PASSED
tests/unit/test_ticket_endpoint.py::test_all_support_tickets_with_fixture_get_200_success PASSED
tests/unit/test_ticket_endpoint.py::test_all_admin_tickets_with_fixture_get_200_success PASSED
tests/unit/test_ticket_endpoint.py::test_ticket_api_with_fixture_post_200_success PASSED
tests/unit/test_ticket_endpoint.py::test_ticket_api_with_fixture_get_200_success PASSED
tests/unit/test_users_endpoint.py::test_student_api_with_fixture_get_200 PASSED
tests/unit/test_users_endpoint.py::test_student_api_with_fixture_put_200 PASSED
tests/unit/test_users_endpoint.py::test_support_api_with_fixture_get_200 PASSED
tests/unit/test_users_endpoint.py::test_admin_api_with_fixture_get_200 PASSED

===================================================================== 24 passed in 1.31s ================
(supportTicketVenv) PS D:\IITM_exam\IITM_Term_7\SE_Project\Tushar_Work\code\backend> 
```

MILESTONE : 6
IMPLEMENTATION

6. Implementation Details

6.1. Technologies Used

This project named "***Online Support Ticket System (OSTS)***" is built in Python and JavaScript. The wireframes for the project were built in '***Miro***'. The API doc is built in '***Swagger***'. The API testing is done with '***Insomnia***'. The App testing is done with '***PyTest***'. Coding and GitHub operations were done using '***Visual Studio Code***' and '***Git***'.

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

Library/Framework/Language	**Usage**
Python	Core programming language for the project backend
Flask and its extensions	Micro web framework to create backend API server
SQLite	Backend Database
logging	Library to keep logs of data
smtp and email	Library to send email notifications
SQLAlchemy	Library to manage backend database transactions

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

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

6.2. Instructions to Use App

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

To start the software, there are three steps.

1. Start the 'MailHog_windows_amd64.exe' so that the emails sent during the usage of app, will be captured by '*MailHog*' at http://127.0.0.1:8025/.
2. Then run the 'backend.bat' file. It starts the backend server which handles database manipulations and API functions. It runs at http://127.0.0.1:5000/

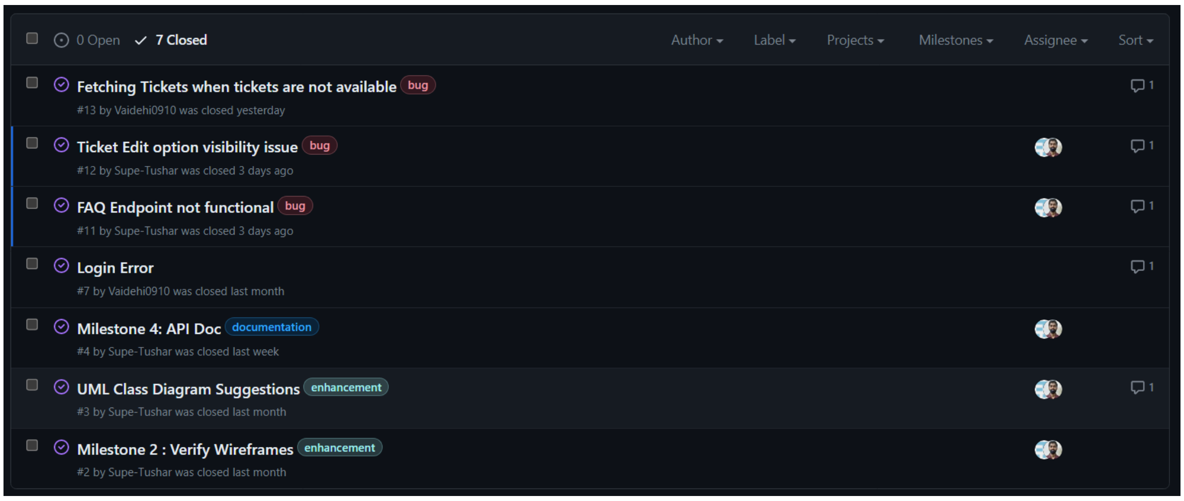
3. Then run the ‘frontend.bat’ file. It starts the frontend server which serves web pages for the frontend user.

Finally, visit ‘http://127.0.0.1:8080/home’ page on browser.

6.3. Code Review and Issue Tracking

We (there were only two of us) divided up the tasks for this project. The tasks were independent of one another because of how they were chosen. For instance, member B will work on the "Tickets API" if member A is working on the "Student API". There were merging problems when many components were introduced and used simultaneously, despite the fact that the tasks and coding components were independent. These issues were reported on GitHub by the relevant members, and the person in charge of resolving them updated and corrected the code. Various issues were created and resolved throughout the project.

Various Issues Which Were Created



Fetching Tickets when tickets are not available bug
#13 by Vaidehi0910 was closed yesterday

Ticket Edit option visibility issue bug
#12 by Supe-Tushar was closed 3 days ago

FAQ Endpoint not functional bug
#11 by Supe-Tushar was closed 3 days ago

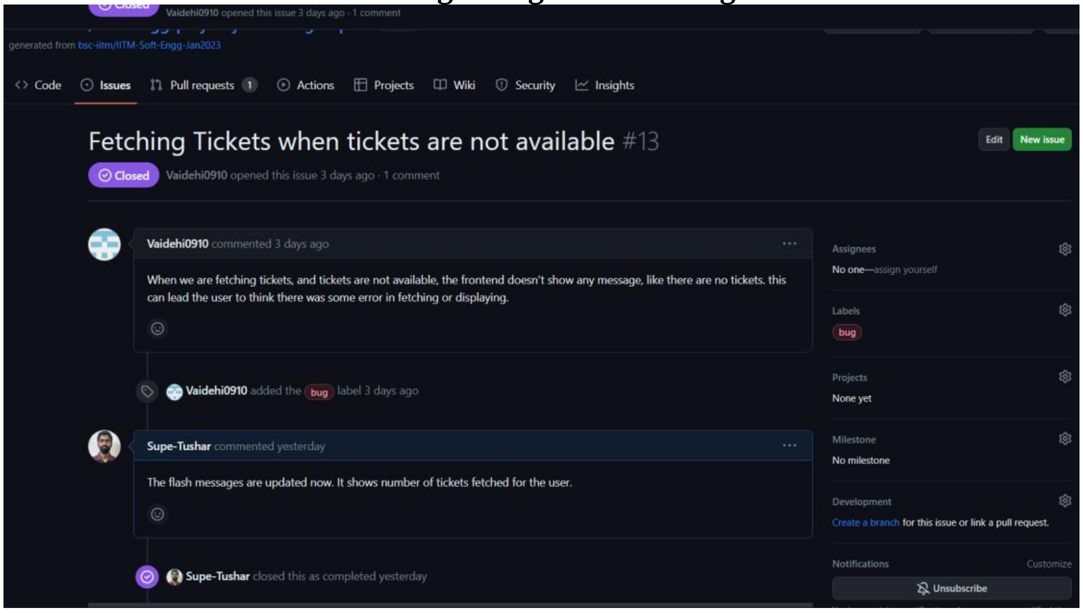
Login Error
#7 by Vaidehi0910 was closed last month

Milestone 4: API Doc documentation
#4 by Supe-Tushar was closed last week

UML Class Diagram Suggestions enhancement
#3 by Supe-Tushar was closed last month

Milestone 2 : Verify Wireframes enhancement
#2 by Supe-Tushar was closed last month

Issue Regarding Flash Messages



Fetching Tickets when tickets are not available #13

Closed Vaidehi0910 opened this issue 3 days ago · 1 comment

Vaidehi0910 commented 3 days ago

When we are fetching tickets, and tickets are not available, the frontend doesn't show any message, like there are no tickets. this can lead the user to think there was some error in fetching or displaying.

Vaidehi0910 added the bug label 3 days ago

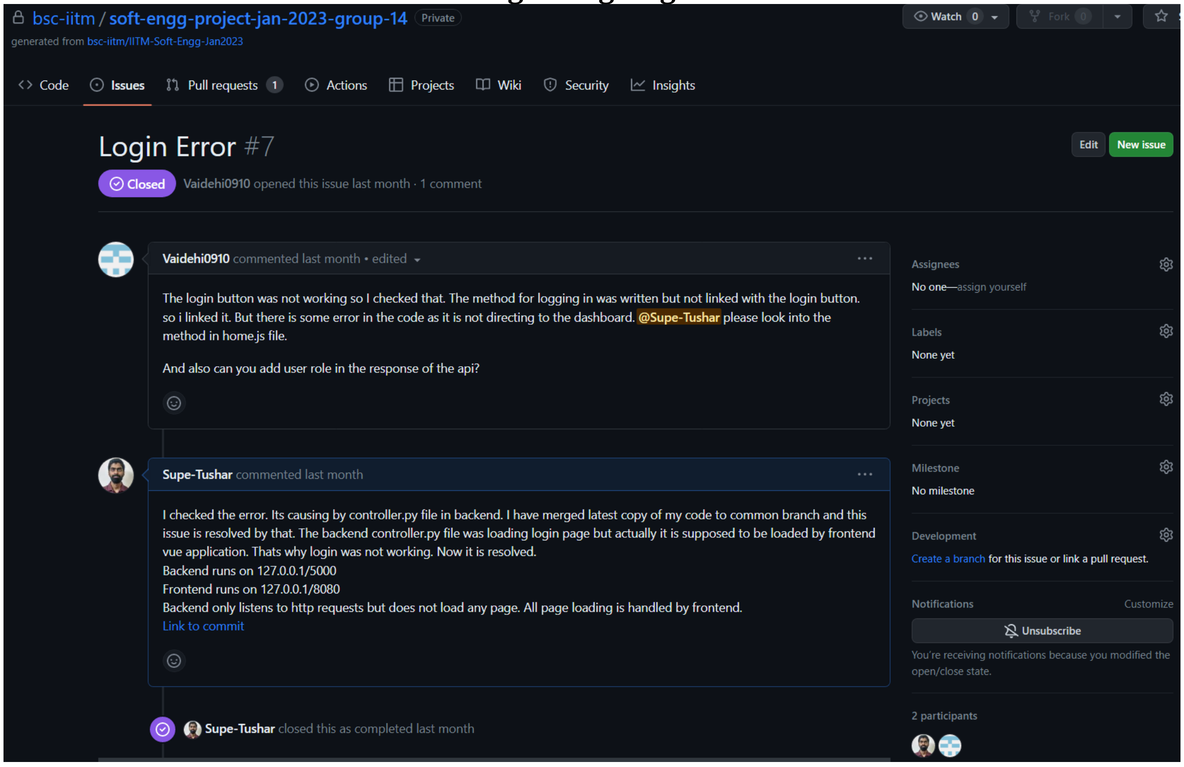
Supe-Tushar commented yesterday

The flash messages are updated now. It shows number of tickets fetched for the user.

Supe-Tushar closed this as completed yesterday

Assignees: No one—assign yourself
Labels: bug
Projects: None yet
Milestone: No milestone
Development: Create a branch for this issue or link a pull request.
Notifications Customize
Unsubscribe

Issue Regarding Login Error


bsc-iitm / soft-engg-project-jan-2023-group-14 Private
generated from bsc-iitm/IITM-Soft-Engg-Jan2023

Code · Issues · Pull requests 1 · Actions · Projects · Wiki · Security · Insights

Login Error #7

Closed Vaideh0910 opened this issue last month · 1 comment

Vaidehi0910 commented last month • edited

The login button was not working so I checked that. The method for logging in was written but not linked with the login button. so i linked it. But there is some error in the code as it is not directing to the dashboard. @Supe-Tushar please look into the method in home.js file.

And also can you add user role in the response of the api?

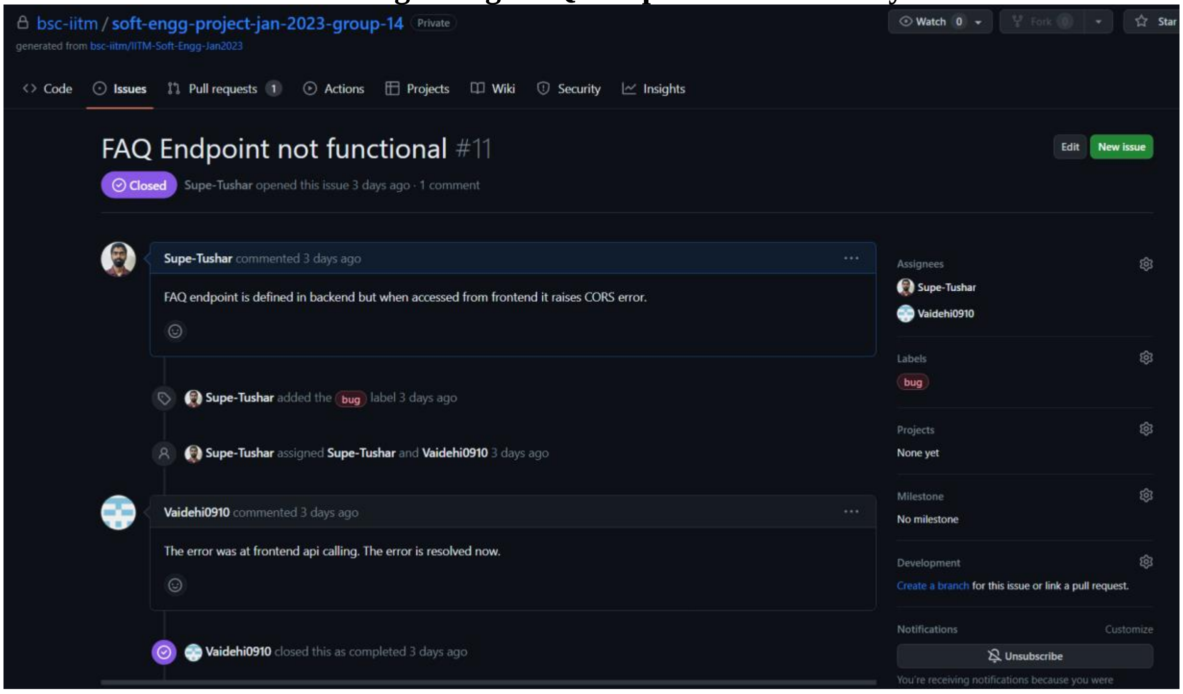
Supe-Tushar commented last month

I checked the error. Its causing by controller.py file in backend. I have merged latest copy of my code to common branch and this issue is resolved by that. The backend controller.py file was loading login page but actually it is supposed to be loaded by frontend vue application. Thats why login was not working. Now it is resolved.
Backend runs on 127.0.0.1/5000
Frontend runs on 127.0.0.1/8080
Backend only listens to http requests but does not load any page. All page loading is handled by frontend.
Link to commit

Supe-Tushar closed this as completed last month

Assignees: No one—assign yourself
Labels: None yet
Projects: None yet
Milestone: No milestone
Development: Create a branch for this issue or link a pull request.
Notifications: Unsubscribe. You're receiving notifications because you modified the open/close state.
2 participants

Issue Regarding FAQ Endpoint Functionality


bsc-iitm / soft-engg-project-jan-2023-group-14 Private
generated from bsc-iitm/IITM-Soft-Engg-Jan2023

Code · Issues · Pull requests 1 · Actions · Projects · Wiki · Security · Insights

FAQ Endpoint not functional #11

Closed Supe-Tushar opened this issue 3 days ago · 1 comment

Supe-Tushar commented 3 days ago

FAQ endpoint is defined in backend but when accessed from frontend it raises CORS error.

Supe-Tushar added the bug label 3 days ago

Supe-Tushar assigned Supe-Tushar and Vaidehi0910 3 days ago

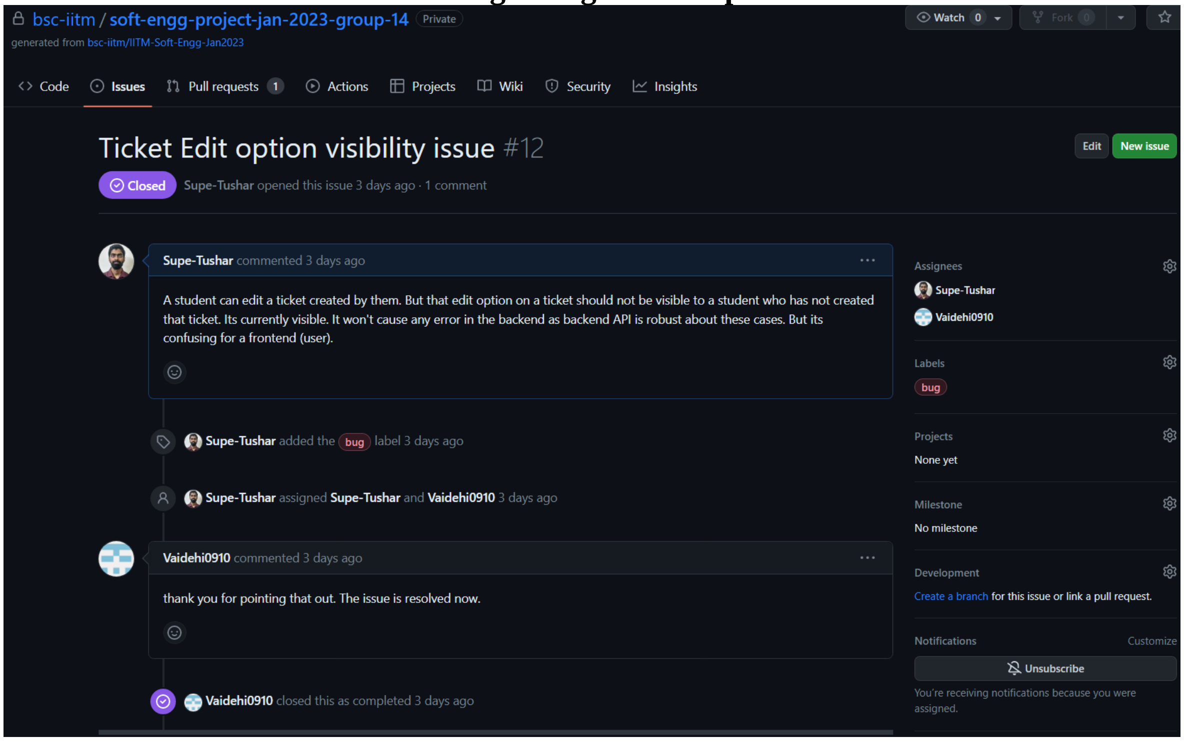
Vaidehi0910 commented 3 days ago

The error was at frontend api calling. The error is resolved now.

Vaidehi0910 closed this as completed 3 days ago

Assignees: Supe-Tushar, Vaidehi0910
Labels: bug
Projects: None yet
Milestone: No milestone
Development: Create a branch for this issue or link a pull request.
Notifications: Unsubscribe. You're receiving notifications because you were

Issue Regarding Ticket Options


bsc-iitm / soft-engg-project-jan-2023-group-14 Private

generated from bsc-iitm/IITM-Soft-Engg-Jan2023

Ticket Edit option visibility issue #12

Closed Supe-Tushar opened this issue 3 days ago · 1 comment

Supe-Tushar commented 3 days ago

A student can edit a ticket created by them. But that edit option on a ticket should not be visible to a student who has not created that ticket. Its currently visible. It won't cause any error in the backend as backend API is robust about these cases. But its confusing for a frontend (user).

Supe-Tushar added the bug label 3 days ago

Supe-Tushar assigned Supe-Tushar and Vaidehi0910 3 days ago

Vaidehi0910 commented 3 days ago

thank you for pointing that out. The issue is resolved now.

Vaidehi0910 closed this as completed 3 days ago

Assignees: Supe-Tushar, Vaidehi0910

Labels: bug

Projects: None yet

Milestone: No milestone

Development: Create a branch for this issue or link a pull request.

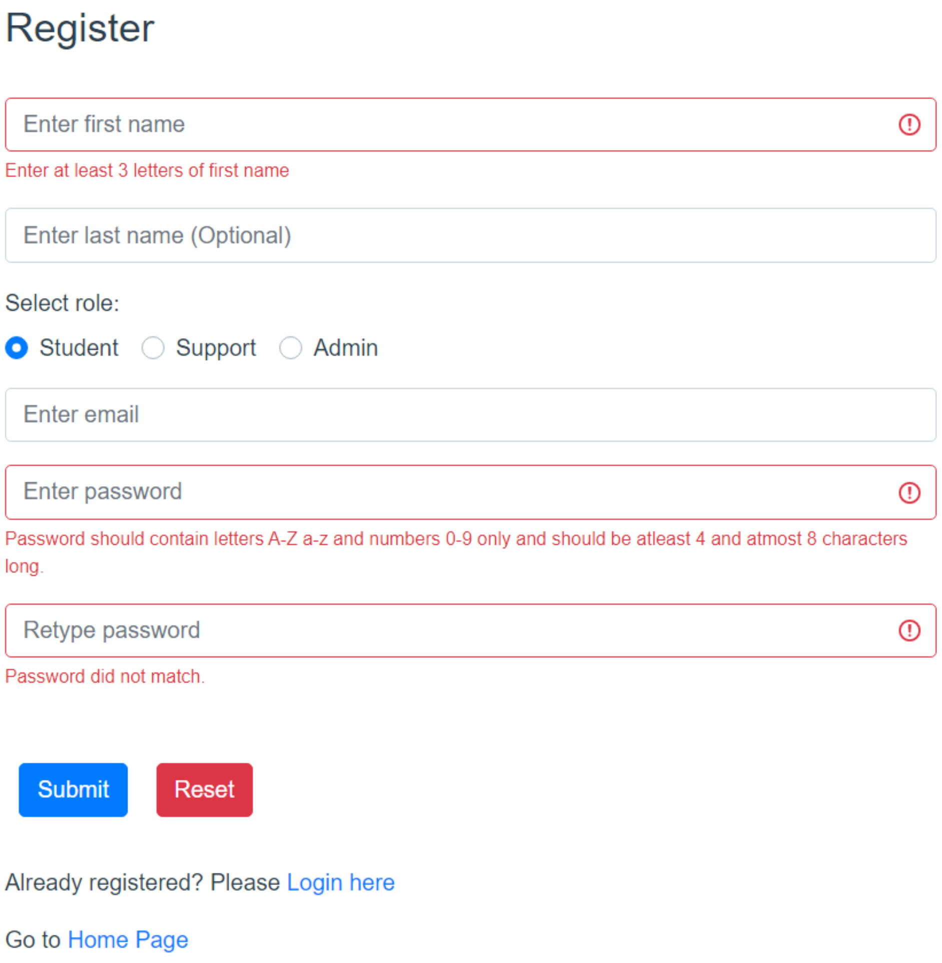
Notifications: You're receiving notifications because you were assigned.

For these issues, the member responsible for solving (coding part of that issue), went through the issue details, then updated the code and pushed the changes to their respective branch and then a pull request was created to merge code with 'common' branch. Every member stayed up-to-date with 'common' branch. Thus, issues were resolved.

6.4. Actual WebApp Screenshots

Few of the screenshots of webpages are attached below. The live demo and presentation video is available in GitHub repository in ‘milestone-6’ directory.

Register Page



Register

Enter first name

Enter at least 3 letters of first name

Enter last name (Optional)

Select role:

[✓ Student] Support Admin

Enter email

Enter password

Password should contain letters A-Z a-z and numbers 0-9 only and should be atleast 4 and atmost 8 characters long.

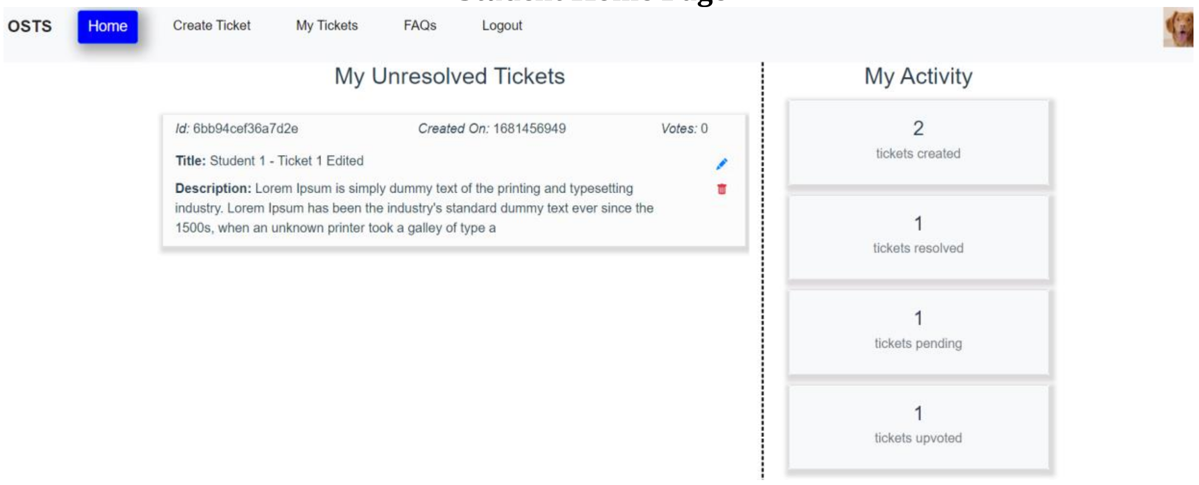
Retype password

Password did not match.

Already registered? Please Login here

Go to Home Page

Student Home Page



My Unresolved Tickets

Id: 6bb94cef36a7d2e *Created On:* 1681456949 *Votes:* 0

Title: Student 1 - Ticket 1 Edited

Description: Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type a

My Activity

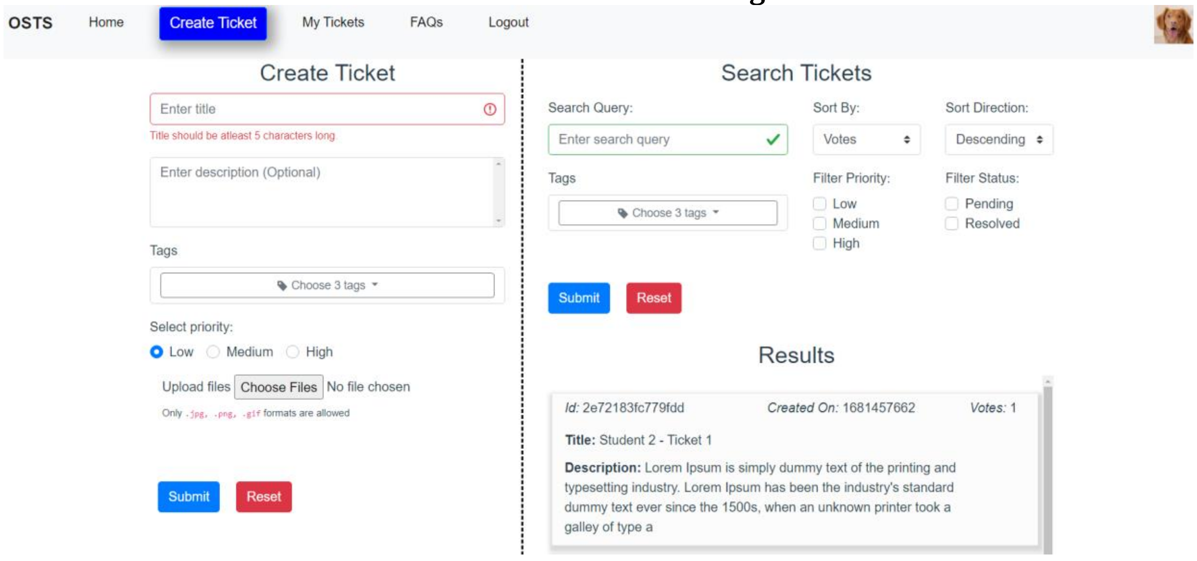
2 tickets created

1 tickets resolved

1 tickets pending

1 tickets upvoted

Student Create Ticket Page


OSTS | Home | Create Ticket | My Tickets | FAQs | Logout

Create Ticket

Enter title

Title should be atleast 5 characters long

Enter description (Optional)

Tags

Choose 3 tags

Select priority:

Low [✓] Medium High

Upload files Choose Files No file chosen

Only .jpg, .png, .gif formats are allowed

Submit Reset

Search Tickets

Search Query: Enter search query

Sort By: Votes

Sort Direction: Descending

Tags: Choose 3 tags

Filter Priority: Low, Medium, High

Filter Status: Pending, Resolved

Submit Reset

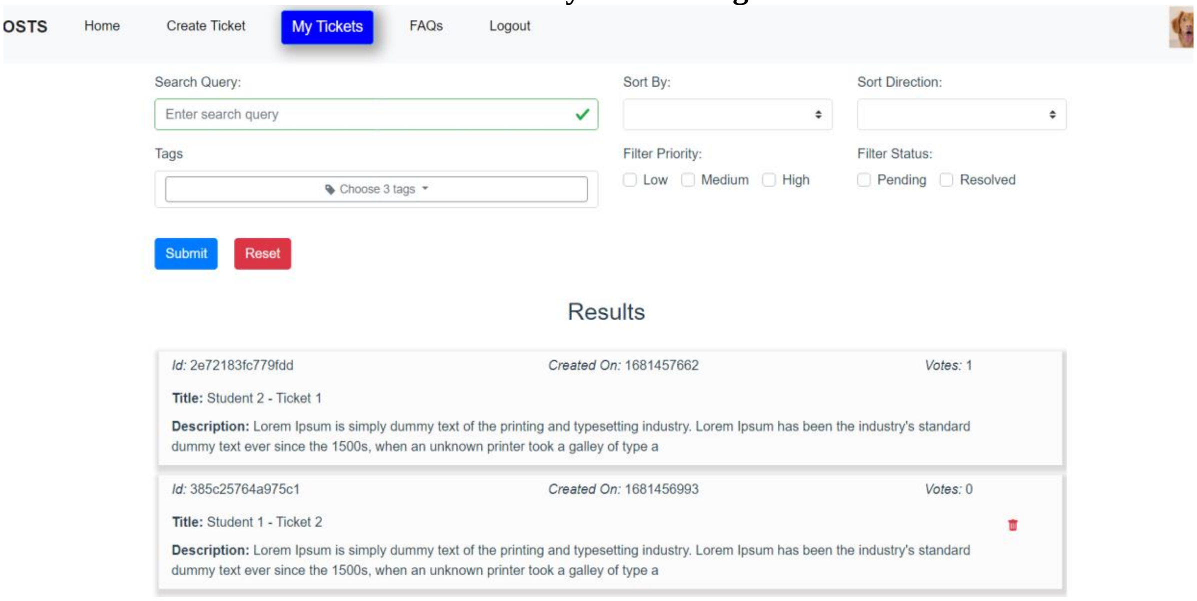
Results

Id: 2e72183fc779fdd *Created On:* 1681457662 *Votes:* 1

Title: Student 2 - Ticket 1

Description: Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type a

Student My Tickets Page


OSTS | Home | Create Ticket | My Tickets | FAQs | Logout

Search Query: Enter search query

Sort By:

Sort Direction:

Tags: Choose 3 tags

Filter Priority: Low, Medium, High

Filter Status: Pending, Resolved

Submit Reset

Results

Id: 2e72183fc779fdd *Created On:* 1681457662 *Votes:* 1

Title: Student 2 - Ticket 1

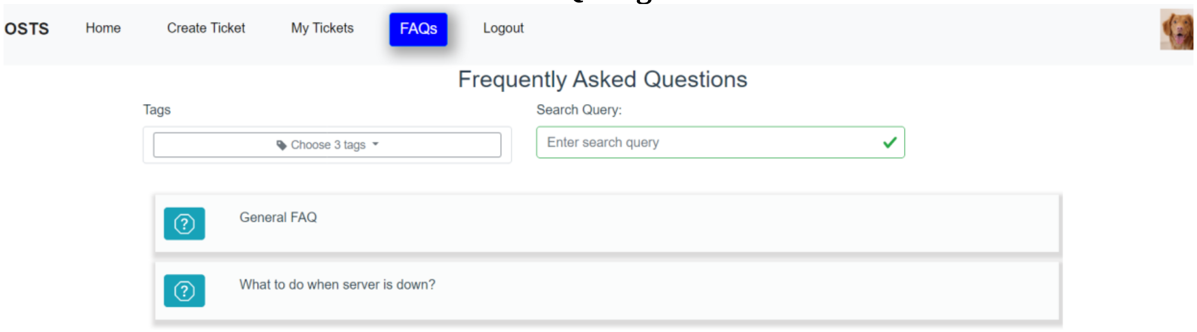
Description: Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type a

Id: 385c25764a975c1 *Created On:* 1681456993 *Votes:* 0

Title: Student 1 - Ticket 2

Description: Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type a

FAQ Page


OSTS | Home | Create Ticket | My Tickets | FAQs | Logout

Frequently Asked Questions

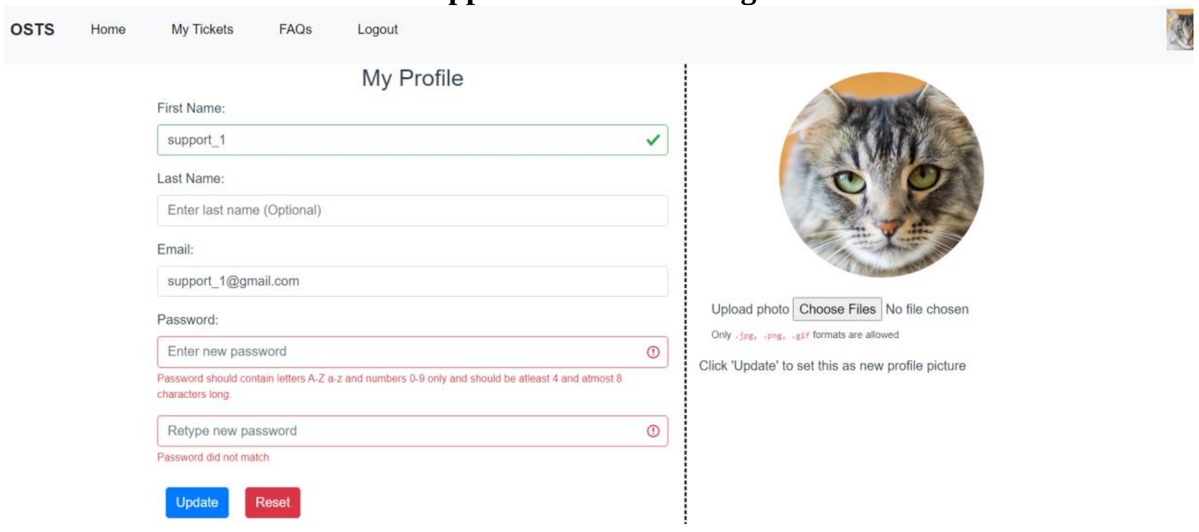
Tags: Choose 3 tags

Search Query: Enter search query

General FAQ

What to do when server is down?

Support User Profile Page


OSTS Home My Tickets FAQs Logout

My Profile

First Name:
support_1

Last Name:
Enter last name (Optional)

Email:
support_1@gmail.com

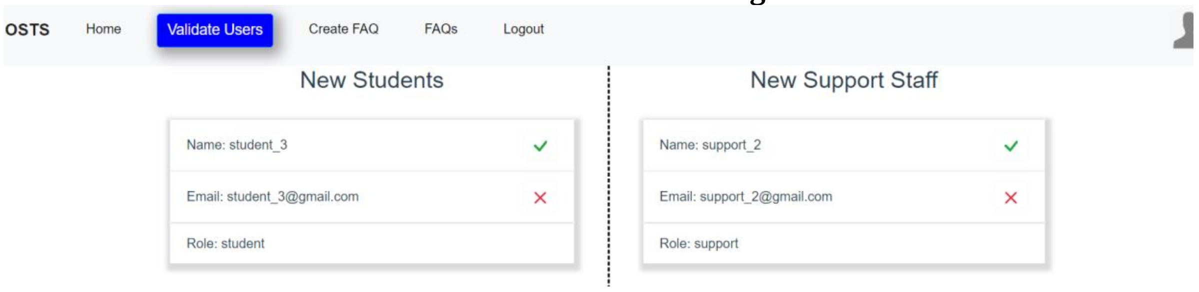
Password:
Enter new password
Password should contain letters A-Z a-z and numbers 0-9 only and should be atleast 4 and atmost 8 characters long.

Retype new password
Password did not match

Update Reset

Upload photo Choose Files No file chosen
Only .jpg, .png, .gif formats are allowed
Click 'Update' to set this as new profile picture

Admin Validate Users Page


OSTS Home Validate Users Create FAQ FAQs Logout

New Students

Name: student_3
Email: student_3@gmail.com
Role: student

New Support Staff

Name: support_2
Email: support_2@gmail.com
Role: support

References

[1] Software Engineering Project: Problem Statement

[2] YouTube Video: Storyboarding

[3] YouTube Video: Wireframes

[4] PyTest Online Resource: TestDriven