Software Engineering Project

Milestone 1

SUBMITTED IN THE PARTIAL FULFILLMENT OF THE

REQUIREMENTS OF THE COURSE:

By :- Group 10

HARSH Y MEHTA (21f1001295)

LEON B GEORGE (21f1000889)

Indian Institute of technology, Madra
Chennai, India - 600036

Milestone - 01

Software Engineering Project

Team Member 1 - Leon B George (21f1000889)
Team Member 2 - Harsh Y Mehta (21f1001295)

Primary users:

Students who need to submit support tickets for their concerns or queries.
Support staff who need to address and resolve the tickets submitted by students.
Admins who need to manage and monitor the ticketing system, as well as update the dynamic FAQ section.

Secondary users:

Support staff who may need to collaborate on certain support tickets.

Tertiary users:

Faculty members who may be consulted by the support staff to provide information or solutions for certain support tickets.

User stories:

As a student, I want to be able to create a support ticket for my concerns or queries, so that I can receive assistance from the support team.
As a student, I want to be able to view a list of similar tickets before creating a new one, and add a +1/like if the issue being addressed is the same as mine so that I can avoid creating duplicates and prioritize popular concerns.
As a student, I want to be able to track the status of my support ticket, so that I can know if it's being addressed or not.
As a student, I want to receive an automatic confirmation email after creating a support ticket, so that I know my query has been successfully submitted and acknowledged.

As a support staff member, I want to be able to mark a ticket as resolved, so that the student who submitted the ticket will receive an appropriate notification and the ticket can be closed.

As a support staff member, I want to be able to assign a ticket to another support staff member or escalate it to a higher authority, so that complex or urgent issues can be resolved efficiently.
As a support staff member, I want to be able to add internal notes or comments to a support ticket, so that I can provide context or updates to other staff members who may be working on the same issue.

As an admin, I want to be able to manage and monitor the ticketing system, so that I can ensure tickets are being addressed in a timely and effective manner.
As an admin, I want to be able to update the dynamic FAQ section with support queries and responses, so that future students can easily access relevant information.
As an admin, I want to be able to set priorities and deadlines for support tickets, so that urgent or critical issues can be addressed first.
As an admin, I want to be able to generate reports and analytics on the ticketing system, so that I can identify areas for improvement and measure the effectiveness of the support team.

Software Engineering Project

Milestone - 2

Topic - StoryBoard and WireFrame

By :- Group 10

LEON B GEORGE

(21f1000889)

HARSH Y MEHTA

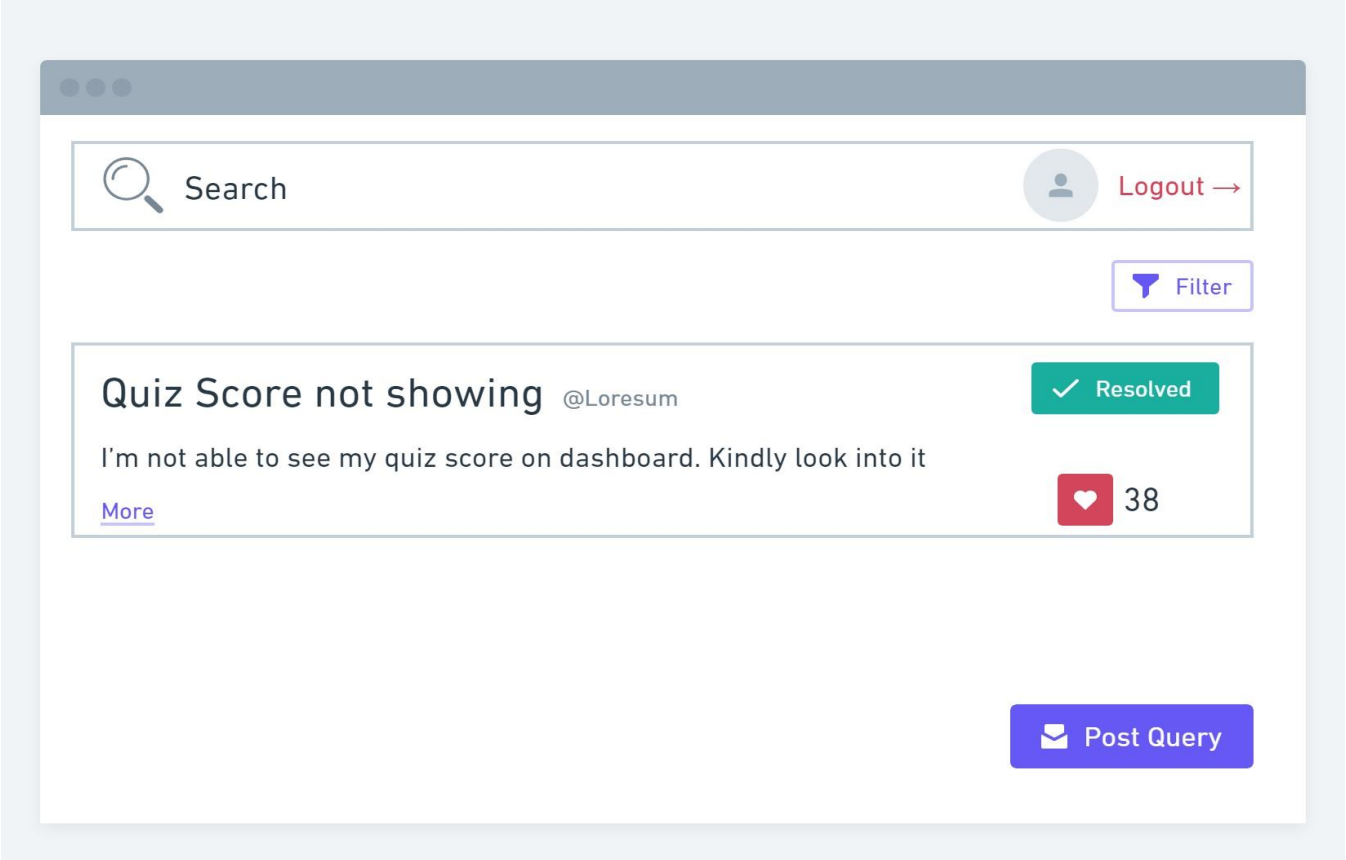
(21f1001295)

StoryBoard

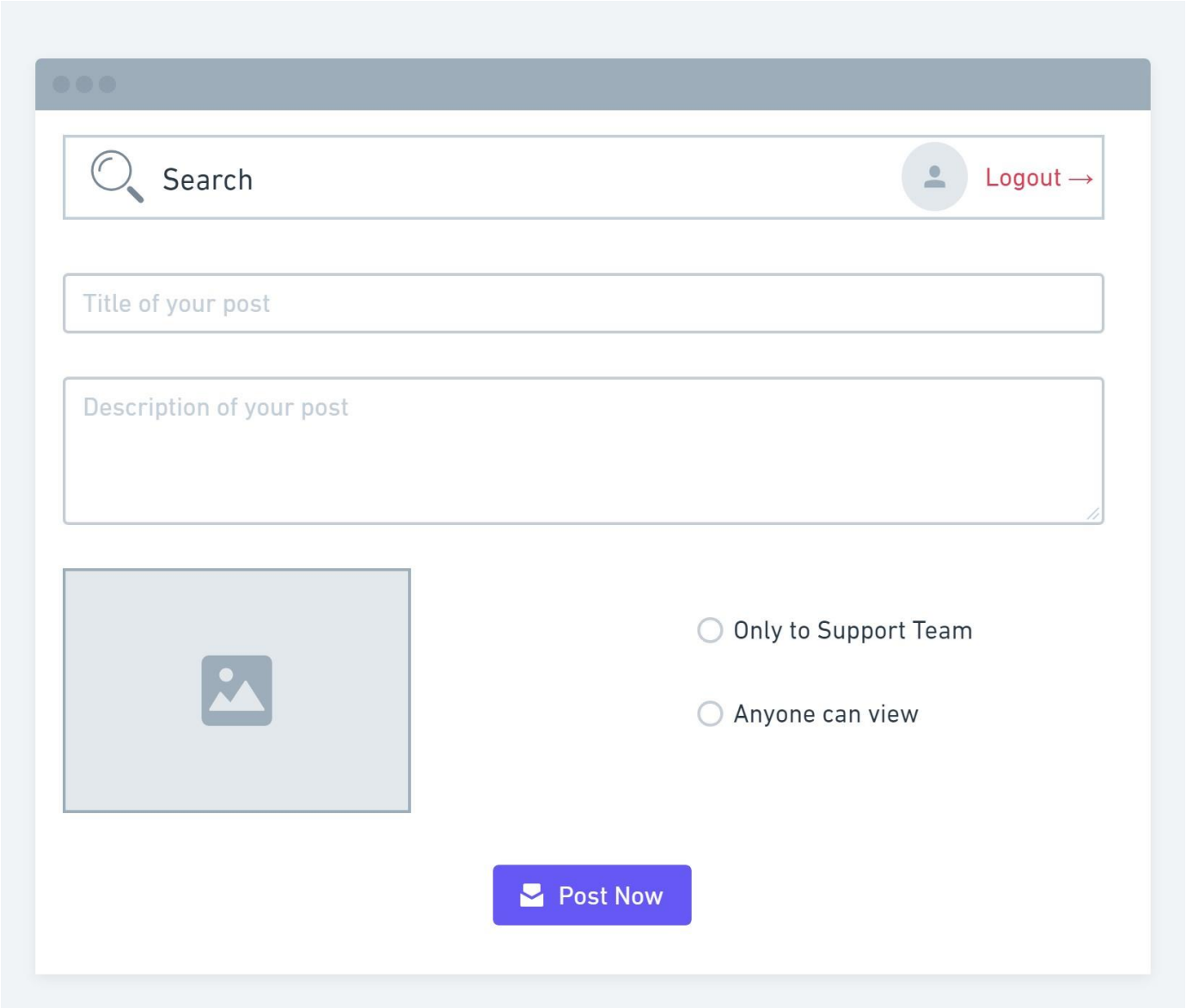
https://docs.google.com/presentation/d/1ajvOFoysgWLDR9utjFwY5S6azav502OQ/edit?usp=sharing&ouid=112736013081601353076&rtpof=true&sd=true

WireFrame

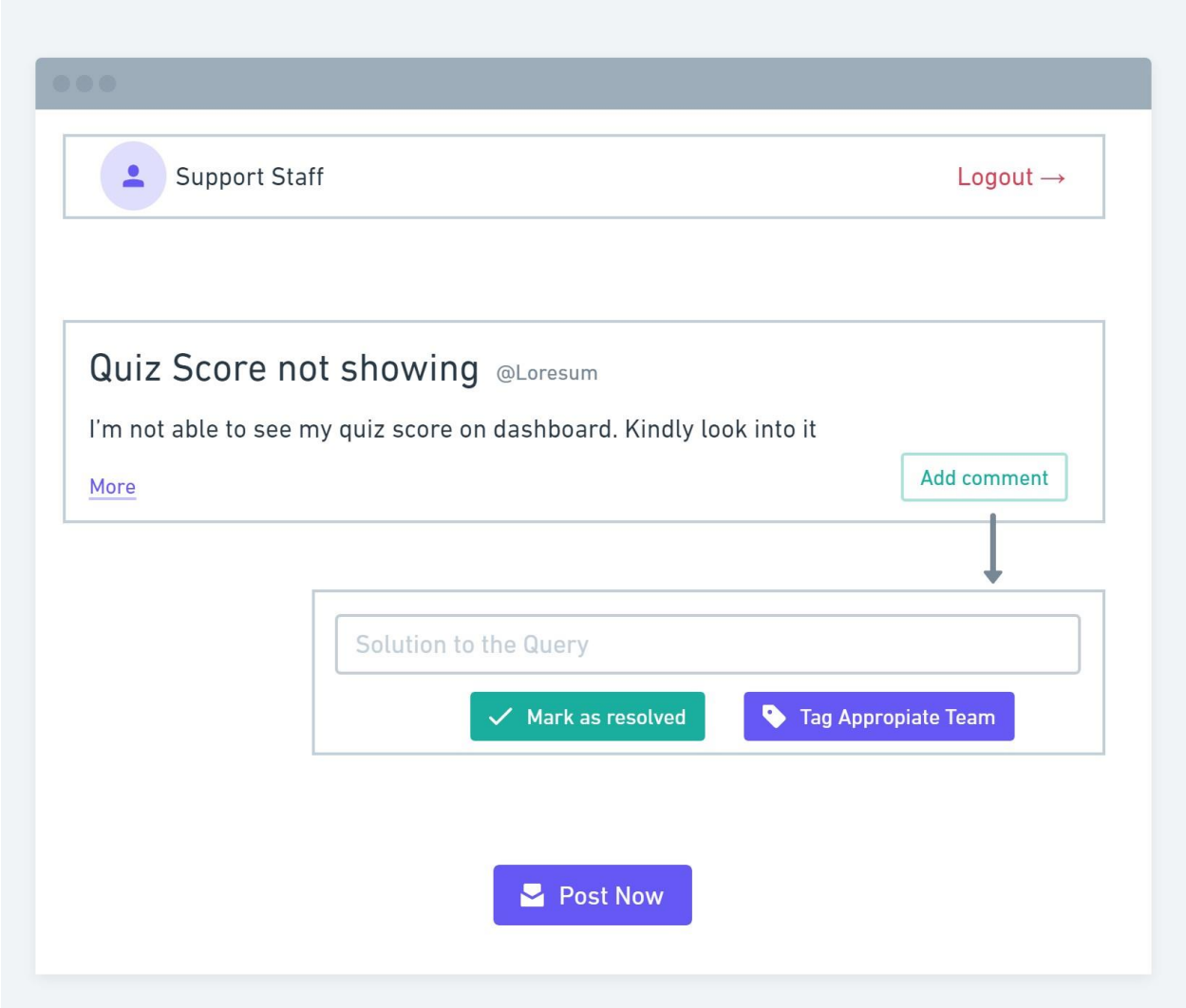
- **Students Perspective** - FAQ, Searching for similar Query before posting



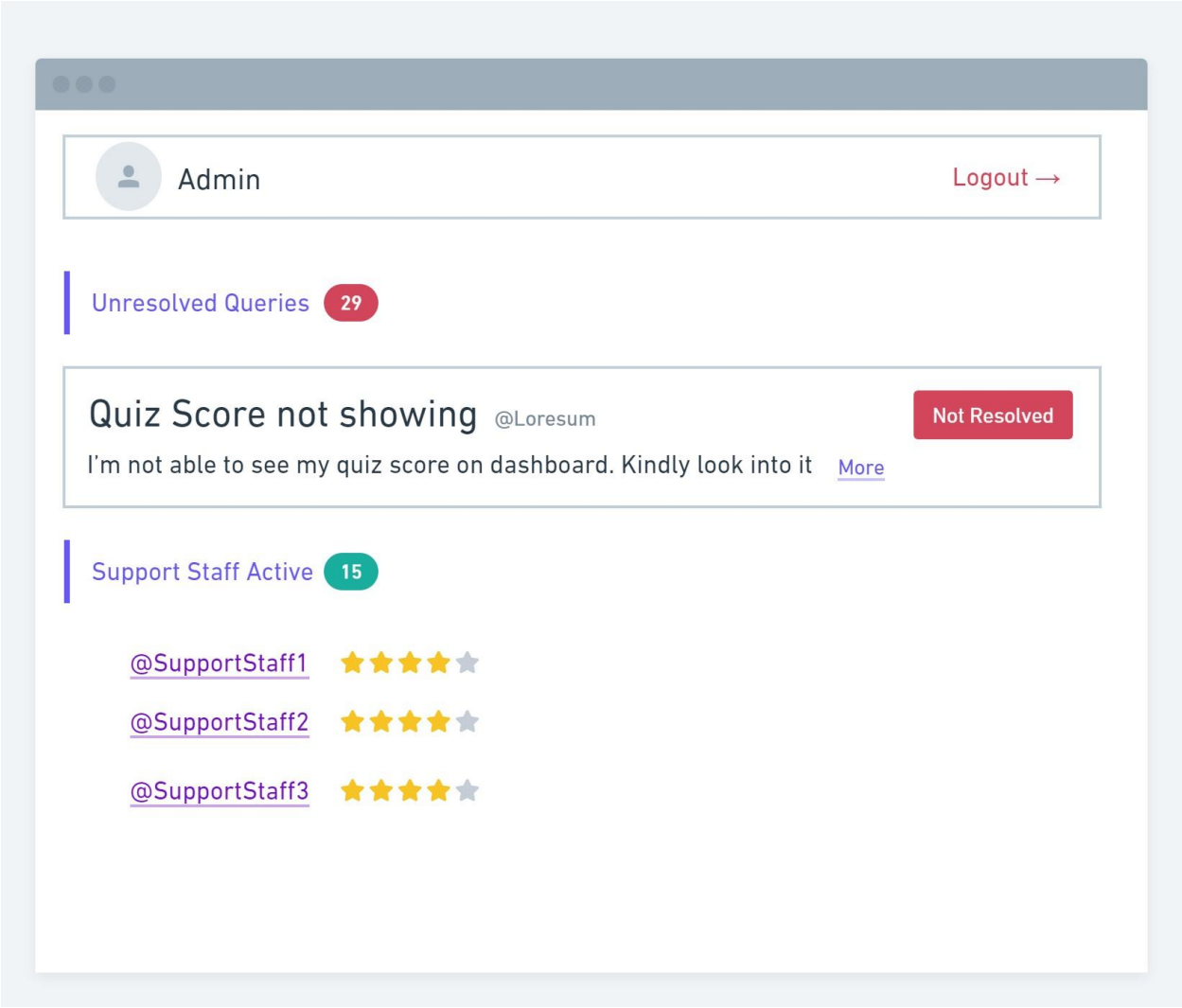
Students Perspective - Posting a Query.



Support Team Perspective - List of tickets raised in Descending order of likes and could further be filter to resolve urgent queries.



Admins Perspective - To see all the Unresolved Queries and Support Staff effort to resolve them.



Software Engineering Project

Milestone 3

SUBMITTED IN THE PARTIAL FULFILLMENT OF THE

REQUIREMENTS OF THE COURSE:

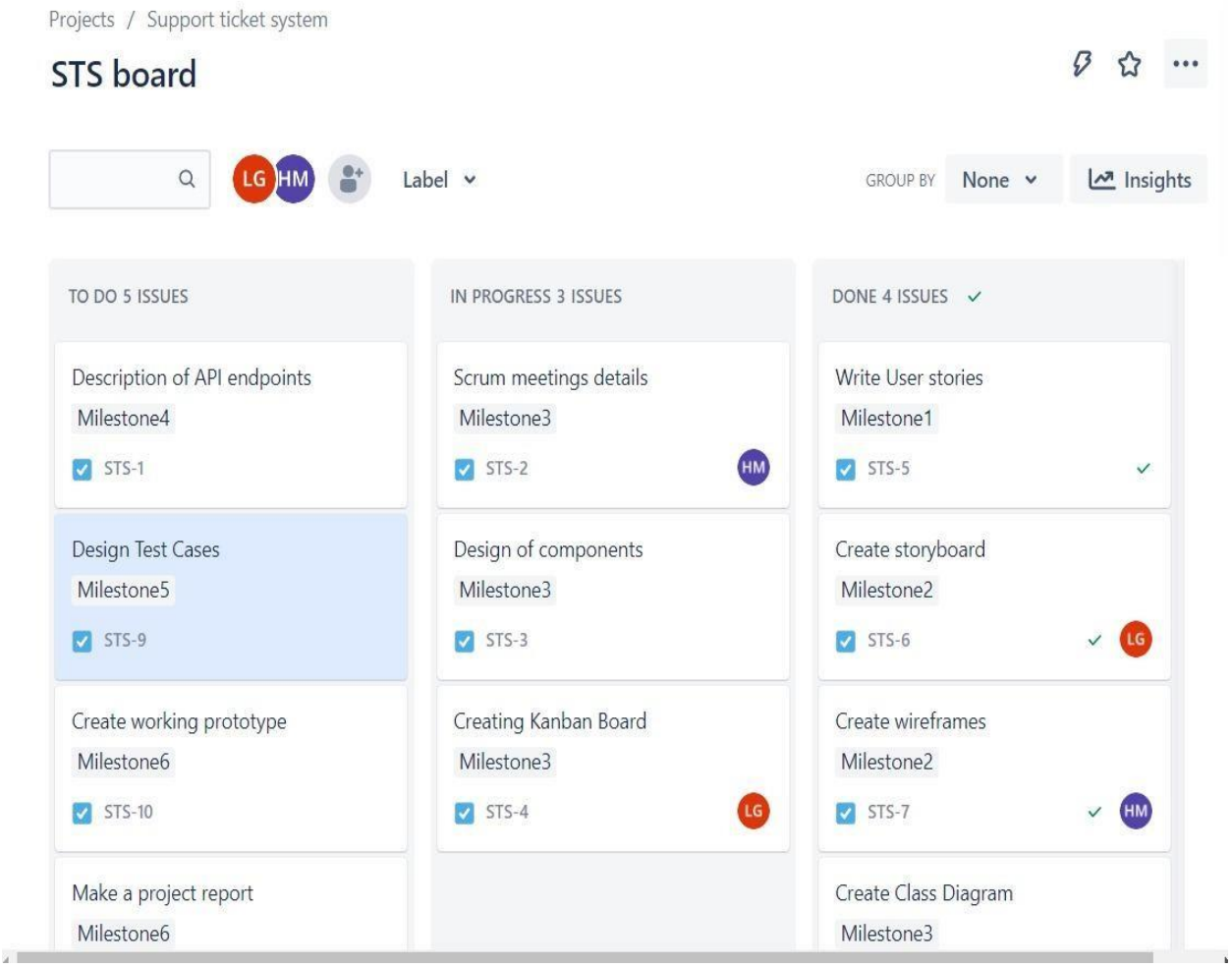
By :- Group 10

HARSH Y MEHTA (21f1001295)

LEON B GEORGE (21f1000889)

Indian Institute of technology, Madra
Chennai, India - 600036

Trello Board



Design of Components

Student View:

- Dashboard - to view other queries and/or check if similar query exist before raising it.
- New Ticket - to raise a ticket when needed and get resolution for it.
- Email Notification on raised tickets.
- Feedback after resolving their query.

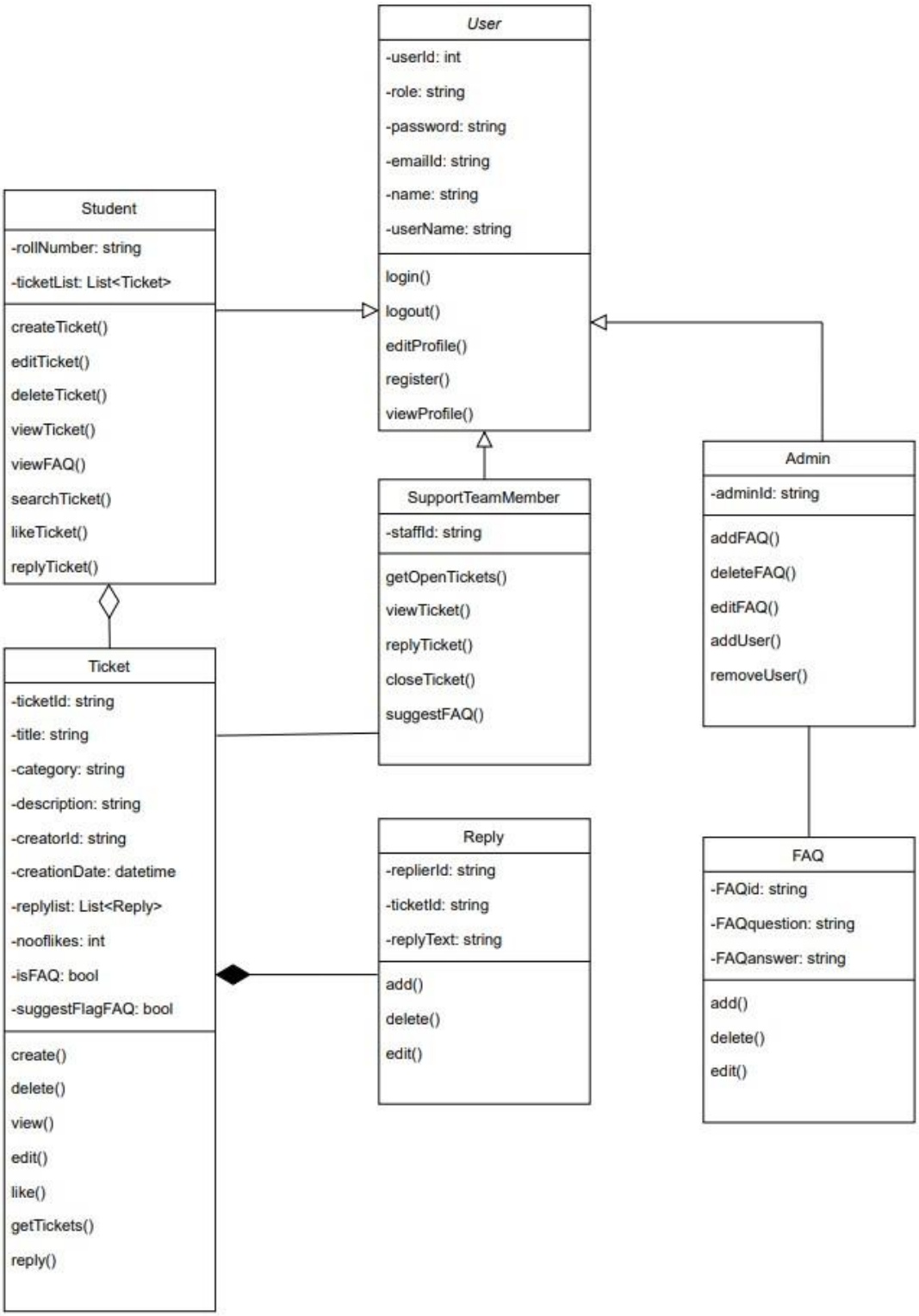
Support Staff View:

- View their assigned tickets.
- Escalate or tag appropriate support staff for quicker resolution.
- Ability to mark ticket as duplicate if similar query exist and completed upon giving a satisfactory solution.
- Filter and sort mechanism to attend to more important queries first when lot to query are assigned.

Admin View:

- Manage student and support staff accounts.
- View all feedback and rating for support staff and students based on which further action could be taken.
- Download ticket data in csv, pdf format to further analyze the queries.

Class Diagram



Sprint Schedule

Date marked in Red are deadlines

Milestone 1: User requirements. 12th Feb ’23

- **Sprint 1:**
  - Identifying users of this application (7th Feb '23).
  - User story (9th Feb ’23)

Milestone 2: User Interfaces 26th Feb ’23

- **Sprint 2:**
  - WireFrame (23rd Feb '23) ●

Sprint 3:
  - StoryBoard (23rd Feb ’23)

Milestone 3: Scheduling and Design 5th Mar ’23

- **Sprint 4:**
  - Project Schedule & Project Scheduling Tool (27th Feb '23)
- **Sprint 5:**
  - Class Diagram (2nd Mar ‘23)
  - Design of Components(4th Mar ‘23)

Milestone 4: API Endpoints 19th Mar ’23

- **Sprint 6:**
  - Description of API endpoints (15th Mar '23).

Milestone 5: Test cases, test suite of the project 2th April ’23 ●

Sprint 7:
  - Describing Test cases for Unit Testing (25th Mar ‘23)
- **Sprint 8:**
  - Unit Testing using Pytest (30th Mar’ 23)

<u>Milestone 6</u>: Test cases, test suite of the project 16th April ’23 ●

Sprint 7:

- ○ Complete implementation with prototype(8th April ’23)
- ○ Hosting ●

Sprint 8:

- ○ Detailed Report of the entire project, Technologies used. (12th April ‘23)
- ○ A short Video demonstration of the project.
- ○ A section describing code review, issue reporting and tracking using screenshots.

Scrum Meeting

Scrum Meeting 1

Date: 6th Feb ‘23
Time: 18:00 - 19:00 IST
Location: Virtual
Attendees: Leon B George, Harsh Y Mehta

Agenda: Brainstorm various Users of this application and user stories

Minutes:
The team brainstormed and discussed various users of the application (primary, secondary, tertiary) and user stories associated with the different user types, and agreed to come up with at least 5 user stories each before the next meeting. The team also discussed the importance of prioritizing user stories and how this would impact the development process.

Action Items:
- Each team member comes up with at least 5 user stories associated with the different user types.
- Assigned work to be completed before the deadline and compiled together.

Scrum Meeting 2

Date: 20th Feb‘23
Time: 20:00 - 21:00 IST
Location: Virtual
Attendees: Leon B George, Harsh Y Mehta

Agenda: Discuss about Wireframes and storyboard for the application.

Minutes:
The team discussed each User Story in detail and applied the SMART Guidelines (Specific, Measurable, Achievable, Relevant, Time-bound) to refine and improve them. The team looked at various example to get ideas on creating Wireframes. The team also discussed tools to use for WireFrame and Storyboard.

Action Items:
- Assigned team member for Wireframe had to create wireframe for each user type.
- Assigned team member for StoryBoard had to make for at least 2 User stories discussed earlier.

Scrum Meeting 3

Date: 2th Mar‘23
Time: 18:00 - 19:00 IST
Location: Virtual
Attendees: Leon B George, Harsh Y Mehta

Agenda: Scheduling tool and Design of Project.

Minutes:
The team discussed various Scheduling Tools and finalized Jira for tracking and assigning work to team members. They also discussed Design of Components based on users and user story.
Action Items:

- Assigned team member for Design of Components and class Diagram ● Assigned team member for setting up scheduling tool Jira.
- Assigned team members to document minutes of each meeting that happened till date.

Milestone 4

API documentation:

https://drive.google.com/file/d/109PM460y-COPEe8c4EVay2HjVPNV1Tam/view?usp=share_link

Software Engineering Project

Milestone 5

SUBMITTED IN THE PARTIAL FULFILLMENT OF THE

REQUIREMENTS OF THE COURSE:

By :- Group 10

HARSH Y MEHTA (21f1001295)

LEON B GEORGE (21f1000889)

Indian Institute of technology, Madra
Chennai, India - 600036

Test Cases

Test Case 1:

Page being tested : Registration page

Inputs : Username - test123, Name - Test, Email test@test.com , Password - 12345678

Expected output : User Registered Successful

Actual Output : User Registered Successful

Result : Success

Test Case 2:

Page being tested : Login page

Inputs : Email - test@test.com , Password - 12345678

Expected output : Logged in Successful

Actual Output : Logged in Successful

Result : Success

Test Case 3:

Page being tested : Support Ticket Creation

Inputs : Valid student details, ticket details(title and description)

Expected output :
Support ticket created, confirmation email sent

Actual Output : Ticket created Successfully

Result : Success

Test Case 4:

Page being tested : Similar Tickets List

Inputs : Query parameters

Expected output : List of similar tickets displayed

Actual Output : Return list of similar tickets displayed

Result : Success

Test Case 5:

Page being tested : Ticket Status Tracking

Inputs : Student's ticket ID

Expected output : Ticket Status (In Progress/Completed)

Actual Output : In Progress

Result : Success

Test Case 6:

Expected output :
Page being tested : Support Staff - Resolve

Inputs :Resolved ticket details

Marked Resolve and Email Notification

Actual Output : Ticket Marked as resolved

Result : Success

Test Case 7:

Page being tested : Support Staff (Assign/Reassign)

Inputs :Ticket ID, Support Staff to tag

Expected output : Notify ticket to tagged Staff

Actual Output : Notification sent

Result : Success

Test Case 8:

Page being tested : Admin Page (Generate reports)

Inputs : Date range, report type

Expected output : Email Report

Actual Output : Report sent to admin mail.

Result : Success

Expected output :

Test Case 9:

Page being tested : Admin Page (Set Priorities)

Inputs : Ticket ID, priority level

Set Deadline based on Priority

Actual Output : Priority set to the ticket

Result : Success

Test Case 10:

Page being tested : Admin (Update FAQ)

Inputs : FAQ Q&A Details

Expected output : Add FAQ Q&A

Actual Output : Q&A added successfully.

Result : Success

Test Case 11:

Page being tested : Ticket Page

Inputs : Reply text

Expected output : Adding reply to the ticket

Expected output :
Actual Output : Reply Submitted Successfully

Result : Success

Unit Testing using Pytest

FLASK API Implementation

app.py > User

```python
from flask import Flask, request, jsonify
from flask_sqlalchemy import SQLAlchemy
from flask_restful import Api, Resource
import jwt

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///users.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
app.config['SECRET_KEY'] = 'your_secret_key'
db = SQLAlchemy(app)
api = Api(app)

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(120), unique=True, nullable=False)
    password = db.Column(db.String(80), nullable=False)
    name = db.Column(db.String(120), nullable=True)
    username = db.Column(db.String(120), unique=True, nullable=False)
    role = db.Column(db.String(80), nullable=False)

    def __repr__(self):
        return f'<User {self.email}>'

class Ticket(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(120), nullable=False)
    description = db.Column(db.Text, nullable=False)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
    user = db.relationship('User', backref='tickets')
    resolved = db.Column(db.Boolean, default=False)

    def __repr__(self):
        return f'<Ticket {self.title}>'

class Reply(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    text = db.Column(db.Text, nullable=False)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
    user = db.relationship('User', backref='replies')
    ticket_id = db.Column(db.Integer, db.ForeignKey('ticket.id'), nullable=False)
```

app.py × pytesting.py users.db app copy.py

app.py > User

```python
class Reply(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    text = db.Column(db.Text, nullable=False)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
    user = db.relationship('User', backref='replies')
    ticket_id = db.Column(db.Integer, db.ForeignKey('ticket.id'), nullable=False)
    ticket = db.relationship('Ticket', backref='replies')

    def __repr__(self):
        return f'<Reply {self.id}>'

db.create_all()

class Register(Resource):
    def post(self):
        email = request.json.get("email")
        password = request.json.get("password")
        name = request.json.get("name")
        username = request.json.get("username")
        role = request.json.get("role")

        if not email or not password or not role:
            return {"message": "Email, password, and role are required"}, 400

        existing_user = User.query.filter_by(email=email).first()
        if existing_user:
            return {"message": "User with this email already exists"}, 400

        user = User(email=email, password=password, name=name, username=username, role=role)
        db.session.add(user)
        db.session.commit()

        return {"message": "User registered successfully"}, 201

class Login(Resource):
    def post(self):
        email = request.json.get("email")
        password = request.json.get("password")
```

app.py > User

```python
class Login(Resource):
    def post(self):
        email = request.json.get("email")
        password = request.json.get("password")

        if not email or not password:
            return {"message": "Email and password are required"}, 400

        user = User.query.filter_by(email=email).first()
        if not user or not password:
            return {"message": "Invalid email or password"}, 401

        token = jwt.encode({"user_id": user.id}, app.config['SECRET_KEY'], algorithm='HS256')
        print(token)
        return {"message": "Logged in successfully", "token": token}, 200

class CreateTicket(Resource):
    def post(self):
        auth_header = request.headers.get("Authorization")
        if not auth_header:
            return {"message": "Missing authorization header"}, 401

        token = auth_header.split(" ")[1]
        try:
            payload = jwt.decode(token, app.config['SECRET_KEY'], algorithms=['HS256'])
        except jwt.ExpiredSignatureError:
            return {"message": "Expired token"}, 401
        except jwt.InvalidTokenError:
            return {"message": "Invalid token"}, 401

        user_id = payload['user_id']
        title = request.json.get("title")
        description = request.json.get("description")

        if not title or not description:
            return {"message": "Title and description are required"}, 400

        ticket = Ticket(title=title, description=description, user_id=user_id)
        db.session.add(ticket)
```

```python
        ticket = Ticket(title=title, description=description, user_id=user_id)
        db.session.add(ticket)
        db.session.commit()

        return {"message": "Ticket created successfully", "ticket_id": ticket.id}, 201

class SubmitReply(Resource):
    def post(self, ticket_id):
        auth_header = request.headers.get("Authorization")
        if not auth_header:
            return {"message": "Missing authorization header"}, 401

        token = auth_header.split(" ")[1]
        try:
            payload = jwt.decode(token, app.config['SECRET_KEY'], algorithms=['HS256'])
        except jwt.ExpiredSignatureError:
            return {"message": "Expired token"}, 401
        except jwt.InvalidTokenError:
            return {"message": "Invalid token"}, 401

        user_id = payload['user_id']
        text = request.json.get("text")

        if not text:
            return {"message": "Text is required"}, 400

        ticket = Ticket.query.get(ticket_id)
        if not ticket:
            return {"message": "Ticket not found"}, 404

        reply = Reply(text=text, user_id=user_id, ticket_id=ticket_id)
        db.session.add(reply)
        db.session.commit()

        return {"message": "Reply submitted successfully"}, 201

class MarkTicketResolved(Resource):
    def put(self, ticket_id):
        auth_header = request.headers.get("Authorization")
```

(variable) config: Config

app.py > User

```python
class MarkTicketResolved(Resource):
    def put(self, ticket_id):
        auth_header = request.headers.get("Authorization")
        if not auth_header:
            return {"message": "Missing authorization header"}, 401

        token = auth_header.split(" ")[1]
        try:
            payload = jwt.decode(token, app.config['SECRET_KEY'], algorithms=['HS256'])
        except jwt.ExpiredSignatureError:
            return {"message": "Expired token"}, 401
        except jwt.InvalidTokenError:
            return {"message": "Invalid token"}, 401

        user_id = payload['user_id']
        ticket = Ticket.query.get(ticket_id)
        if not ticket:
            return {"message": "Ticket not found"}, 404

        ticket.resolved = True
        db.session.commit()

        return {"message": "Ticket marked as resolved"}, 200

api.add_resource(Register, "/api/register")
api.add_resource(Login, "/api/login")
api.add_resource(CreateTicket, "/api/tickets")
api.add_resource(SubmitReply, "/api/tickets/<int:ticket_id>/replies")
api.add_resource(MarkTicketResolved, "/api/tickets/<int:ticket_id>/resolve")

if __name__ == "__main__":
    app.run(debug=True)
```

Unit Testing

app.py | pytesting.py | users.db | app copy.py

pytesting.py > test_login_user

```python
import pytest
import requests

BASE_URL = "http://127.0.0.1:5000/api"

# Test data
test_user_data = {
    "email": "testuser@example.com",
    "password": "password",
    "name": "Test User",
    "username": "testuser",
    "user_id": "123456",
    "role": "student"
}

test_ticket_data = {
    "title": "Test Ticket",
    "description": "This is a test ticket."
}

test_reply_data = {
    "text": "This is a test reply."
}

def test_register_user():
    response = requests.post(f"{BASE_URL}/register", json=test_user_data)
    print(response.text, "Status code = ",response.status_code)
    assert response.status_code == 201
    assert "User registered successfully" in response.text

def test_login_user():
    response = requests.post(f"{BASE_URL}/login", json={"email": test_user_data["email"], "password": test_user_data["password"]})
    print(response.text, "Status code = ",response.status_code)
    assert response.status_code == 200
    assert "token" in response.json()

def test_create_ticket():
    login_response = requests.post(f"{BASE_URL}/login", json={"email": test_user_data["email"], "password": test_user_data["password"]})
    token = login_response.json()["token"]
    headers = {"Authorization": f"Bearer {token}"}
```

app.py | pytesting.py | users.db | app copy.py

pytesting.py > test_login_user

```python
    assert response.status_code == 200
    assert "token" in response.json()

def test_create_ticket():
    login_response = requests.post(f"{BASE_URL}/login", json={"email": test_user_data["email"], "password": test_user_data["password"]})
    token = login_response.json()["token"]
    headers = {"Authorization": f"Bearer {token}"}

    response = requests.post(f"{BASE_URL}/tickets", json=test_ticket_data, headers=headers)
    print(response.text, "Status code = ",response.status_code)
    assert response.status_code == 201
    assert "Ticket created successfully" in response.text

def test_submit_reply():
    login_response = requests.post(f"{BASE_URL}/login", json={"email": test_user_data["email"], "password": test_user_data["password"]})
    token = login_response.json()["token"]
    headers = {"Authorization": f"Bearer {token}"}

    # First, create a ticket to reply to
    create_ticket_response = requests.post(f"{BASE_URL}/tickets", json=test_ticket_data, headers=headers)
    ticket_id = create_ticket_response.json()["ticket_id"]

    # Submit a reply to the created ticket
    response = requests.post(f"{BASE_URL}/tickets/{ticket_id}/replies", json=test_reply_data, headers=headers)
    print(response.text, "Status code = ",response.status_code)
    assert response.status_code == 201
    assert "Reply submitted successfully" in response.text

def test_mark_ticket_resolved():
    login_response = requests.post(f"{BASE_URL}/login", json={"email": test_user_data["email"], "password": test_user_data["password"]})
    token = login_response.json()["token"]
    headers = {"Authorization": f"Bearer {token}"}

    # First, create a ticket to mark as resolved
    create_ticket_response = requests.post(f"{BASE_URL}/tickets", json=test_ticket_data, headers=headers)
    ticket_id = create_ticket_response.json()["ticket_id"]

    # Mark the created ticket as resolved
    response = requests.put(f"{BASE_URL}/tickets/{ticket_id}/resolve", headers=headers)
    print(response.text, "Status code = ",response.status_code)
```

pytesting.py > test_login_user

```python

def test_mark_ticket_resolved():
    login_response = requests.post(f"{BASE_URL}/login", json={"email": test_user_data["email"], "password": test_user_data["password"]})
    token = login_response.json()["token"]
    headers = {"Authorization": f"Bearer {token}"}

    # First, create a ticket to mark as resolved
    create_ticket_response = requests.post(f"{BASE_URL}/tickets", json=test_ticket_data, headers=headers)
    ticket_id = create_ticket_response.json()["ticket_id"]

    # Mark the created ticket as resolved
    response = requests.put(f"{BASE_URL}/tickets/{ticket_id}/resolve", headers=headers)
    print(response.text, "Status code = ",response.status_code)
    assert response.status_code == 200
    assert "Ticket marked as resolved" in response.text

#test_register_user()
#test_login_user()
#test_create_ticket()
#test_submit_reply()
test_mark_ticket_resolved()
```

Command Prompt (Running Pytest)

```
PS C:\Users\leonb\OneDrive\Desktop\IITMonlinedeg\MAD_finalproject_leon> py pytesting.py
{
    "message": "User registered successfully"
 Status code =  201
PS C:\Users\leonb\OneDrive\Desktop\IITMonlinedeg\MAD_finalproject_leon> py pytesting.py
{
    "message": "User with this email already exists"
}
Traceback (most recent call last):
  File "C:\Users\leonb\OneDrive\Desktop\IITMonlinedeg\MAD_finalproject_leon\pytesting.py", line 77, in <module>
    test_register_user()
  File "C:\Users\leonb\OneDrive\Desktop\IITMonlinedeg\MAD_finalproject_leon\pytesting.py", line 28, in test_register_user
    assert response.status_code == 201
PS C:\Users\leonb\OneDrive\Desktop\IITMonlinedeg\MAD_finalproject_leon> py pytesting.py
{
    "message": "Logged in successfully",
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjoxfQ.ftVWQD9oDs-8zTWaaTAq9SYq21WJeq6Y_pz5NY6MIIo"
 Status code =  200
PS C:\Users\leonb\OneDrive\Desktop\IITMonlinedeg\MAD_finalproject_leon> py pytesting.py
{
    "message": "Ticket created successfully",
    "ticket_id": 1
}
 Status code =  201
PS C:\Users\leonb\OneDrive\Desktop\IITMonlinedeg\MAD_finalproject_leon> py pytesting.py
{
    "message": "Reply submitted successfully"
}
 Status code =  201
PS C:\Users\leonb\OneDrive\Desktop\IITMonlinedeg\MAD_finalproject_leon> py pytesting.py
{
    "message": "Ticket marked as resolved"
}
 Status code =  200
PS C:\Users\leonb\OneDrive\Desktop\IITMonlinedeg\MAD_finalproject_leon>
```

Software Engineering Project

Milestone 6

SUBMITTED IN THE PARTIAL FULFILLMENT OF THE

REQUIREMENTS OF THE COURSE:

By :- Group 10

HARSH Y MEHTA (21f1001295)

LEON B GEORGE (21f1000889)

Indian Institute of technology, Madra
Chennai, India – 600036

Technologies and tools used

- Flask for Backend
- Vue for Frontend
- JWT module for security
- SQlite3 database

https://drive.google.com/drive/folders/1yENVDLVeEfGqxgJrok3zOXR9b__I4E5G?usp=share_link