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:

Page being tested : Support Staff - Resolve

Inputs :Resolved ticket details

Expected output : 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

Test Case 9:

Page being tested : Admin Page (Set Priorities)

Inputs : Ticket ID, priority level

Expected output : 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

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)
```

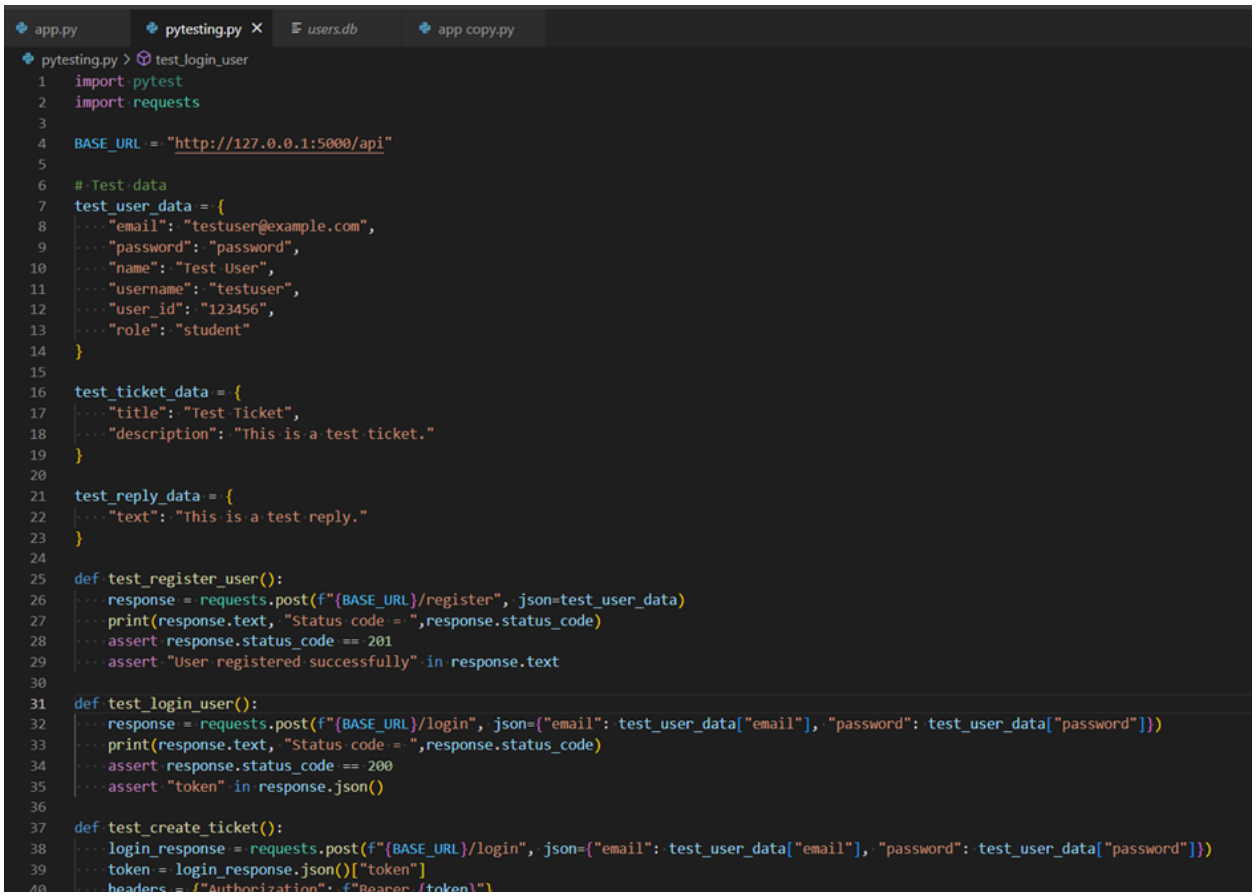
```python
        ticket = Ticket(title=title, description=description, user_id=user_id)
        db.session.add(ticket)
        db.session.commit()

        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

```
    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

```
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)