

# 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](mailto: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](mailto: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
1 from flask import Flask, request, jsonify
2 from flask_sqlalchemy import SQLAlchemy
3 from flask_restful import Api, Resource
4 import jwt
5
6 app = Flask(__name__)
7 app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///users.db'
8 app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
9 app.config['SECRET_KEY'] = 'your_secret_key'
10 db = SQLAlchemy(app)
11 api = Api(app)
12
13 class User(db.Model):
14     id = db.Column(db.Integer, primary_key=True)
15     email = db.Column(db.String(120), unique=True, nullable=False)
16     password = db.Column(db.String(80), nullable=False)
17     name = db.Column(db.String(120), nullable=True)
18     username = db.Column(db.String(120), unique=True, nullable=False)
19     role = db.Column(db.String(80), nullable=False)
20
21     def __repr__(self):
22         return f'<User {self.email}>'
23
24 class Ticket(db.Model):
25     id = db.Column(db.Integer, primary_key=True)
26     title = db.Column(db.String(120), nullable=False)
27     description = db.Column(db.Text, nullable=False)
28     user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
29     user = db.relationship('User', backref='tickets')
30     resolved = db.Column(db.Boolean, default=False)
31
32     def __repr__(self):
33         return f'<Ticket {self.title}>'
34
35 class Reply(db.Model):
36     id = db.Column(db.Integer, primary_key=True)
37     text = db.Column(db.Text, nullable=False)
38     user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
39     user = db.relationship('User', backref='replies')
40     ticket_id = db.Column(db.Integer, db.ForeignKey('ticket.id'), nullable=False)
```

```

app.py x pytesting.py users.db app copy.py
app.py > User
34
35 class Reply(db.Model):
36     id = db.Column(db.Integer, primary_key=True)
37     text = db.Column(db.Text, nullable=False)
38     user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
39     user = db.relationship('User', backref='replies')
40     ticket_id = db.Column(db.Integer, db.ForeignKey('ticket.id'), nullable=False)
41     ticket = db.relationship('Ticket', backref='replies')
42
43     def __repr__(self):
44         return f'<Reply {self.id}>'
45
46 db.create_all()
47
48 class Register(Resource):
49     def post(self):
50         email = request.json.get("email")
51         password = request.json.get("password")
52         name = request.json.get("name")
53         username = request.json.get("username")
54         role = request.json.get("role")
55
56         if not email or not password or not role:
57             return {"message": "Email, password, and role are required"}, 400
58
59         existing_user = User.query.filter_by(email=email).first()
60         if existing_user:
61             return {"message": "User with this email already exists"}, 400
62
63         user = User(email=email, password=password, name=name, username=username, role=role)
64         db.session.add(user)
65         db.session.commit()
66
67         return {"message": "User registered successfully"}, 201
68
69 class Login(Resource):
70     def post(self):
71         email = request.json.get("email")
72         password = request.json.get("password")
73

```

```

app.py > User
69 class Login(Resource):
70     def post(self):
71         email = request.json.get("email")
72         password = request.json.get("password")
73
74         if not email or not password:
75             return {"message": "Email and password are required"}, 400
76
77         user = User.query.filter_by(email=email).first()
78         if not user or not password:
79             return {"message": "Invalid email or password"}, 401
80
81         token = jwt.encode({"user_id": user.id}, app.config['SECRET_KEY'], algorithm='HS256')
82         print(token)
83         return {"message": "Logged in successfully", "token": token}, 200
84
85 class CreateTicket(Resource):
86     def post(self):
87         auth_header = request.headers.get("Authorization")
88         if not auth_header:
89             return {"message": "Missing authorization header"}, 401
90
91         token = auth_header.split(" ")[1]
92         try:
93             payload = jwt.decode(token, app.config['SECRET_KEY'], algorithms=['HS256'])
94         except jwt.ExpiredSignatureError:
95             return {"message": "Expired token"}, 401
96         except jwt.InvalidTokenError:
97             return {"message": "Invalid token"}, 401
98
99         user_id = payload['user_id']
100         title = request.json.get("title")
101         description = request.json.get("description")
102
103         if not title or not description:
104             return {"message": "Title and description are required"}, 400
105
106         ticket = Ticket(title=title, description=description, user_id=user_id)
107         db.session.add(ticket)

```



```

106         ticket = Ticket(title=title, description=description, user_id=user_id)
107         db.session.add(ticket)
108         db.session.commit()
109
110     return {"message": "Ticket created successfully", "ticket_id": ticket.id}, 201
111
112 class SubmitReply(Resource):
113     def post(self, ticket_id):
114         auth_header = request.headers.get("Authorization")
115         if not auth_header:
116             return {"message": "Missing authorization header"}, 401
117
118         token = auth_header.split(" ")[1]
119         try:
120             payload = jwt.decode(token, app.config['SECRET_KEY'], algorithms=['HS256'])
121         except jwt.ExpiredSignatureError:
122             return {"message": "Expired token"}, 401
123         except jwt.InvalidTokenError:
124             return {"message": "Invalid token"}, 401
125
126         user_id = payload['user_id']
127         text = request.json.get("text")
128
129         if not text:
130             return {"message": "Text is required"}, 400
131
132         ticket = Ticket.query.get(ticket_id)
133         if not ticket:
134             return {"message": "Ticket not found"}, 404
135
136         reply = Reply(text=text, user_id=user_id, ticket_id=ticket_id)
137         db.session.add(reply)
138         db.session.commit()
139
140     return {"message": "Reply submitted successfully"}, 201
141
142 class MarkTicketResolved(Resource):
143     def put(self, ticket_id):
144         auth_header = request.headers.get("Authorization")

```

app.py > User

```
141     ...
142     class MarkTicketResolved(Resource):
143     ... def put(self, ticket_id):
144     ...     auth_header = request.headers.get("Authorization")
145     ...     if not auth_header:
146     ...         return {"message": "Missing authorization header"}, 401
147     ...
148     ...     token = auth_header.split(" ")[1]
149     ...     try:
150     ...         payload = jwt.decode(token, app.config['SECRET_KEY'], algorithms=['HS256'])
151     ...     except jwt.ExpiredSignatureError:
152     ...         return {"message": "Expired token"}, 401
153     ...     except jwt.InvalidTokenError:
154     ...         return {"message": "Invalid token"}, 401
155     ...
156     ...     user_id = payload['user_id']
157     ...     ticket = Ticket.query.get(ticket_id)
158     ...     if not ticket:
159     ...         return {"message": "Ticket not found"}, 404
160     ...
161     ...     ticket.resolved = True
162     ...     db.session.commit()
163     ...
164     ...     return {"message": "Ticket marked as resolved"}, 200
165     ...
166
167     api.add_resource(Register, "/api/register")
168     api.add_resource(Login, "/api/login")
169     api.add_resource(CreateTicket, "/api/tickets")
170     api.add_resource(SubmitReply, "/api/tickets/<int:ticket_id>/replies")
171     api.add_resource(MarkTicketResolved, "/api/tickets/<int:ticket_id>/resolve")
172
173     if __name__ == "__main__":
174     ...     app.run(debug=True)
175
```

# Unit Testing

```
app.py  pytest.py X  users.db  app copy.py
pytest.py > test_login_user
1  import pytest
2  import requests
3
4  BASE_URL = "http://127.0.0.1:5000/api"
5
6  # Test data
7  test_user_data = {
8      ... "email": "testuser@example.com",
9      ... "password": "password",
10     ... "name": "Test User",
11     ... "username": "testuser",
12     ... "user_id": "123456",
13     ... "role": "student"
14 }
15
16 test_ticket_data = {
17     ... "title": "Test Ticket",
18     ... "description": "This is a test ticket."
19 }
20
21 test_reply_data = {
22     ... "text": "This is a test reply."
23 }
24
25 def test_register_user():
26     ... response = requests.post(f"{BASE_URL}/register", json=test_user_data)
27     ... print(response.text, "Status code = ", response.status_code)
28     ... assert response.status_code == 201
29     ... assert "User registered successfully" in response.text
30
31 def test_login_user():
32     ... response = requests.post(f"{BASE_URL}/login", json={"email": test_user_data["email"], "password": test_user_data["password"]})
33     ... print(response.text, "Status code = ", response.status_code)
34     ... assert response.status_code == 200
35     ... assert "token" in response.json()
36
37 def test_create_ticket():
38     ... login_response = requests.post(f"{BASE_URL}/login", json={"email": test_user_data["email"], "password": test_user_data["password"]})
39     ... token = login_response.json()["token"]
40     ... headers = {"Authorization": f"Bearer {token}"}
```

```

app.py  pytesting.py x  users.db  app copy.py
pytesting.py > test_login_user
34     assert response.status_code == 200
35     assert "token" in response.json()
36
37 def test_create_ticket():
38     login_response = requests.post(f"{BASE_URL}/login", json={"email": test_user_data["email"], "password": test_user_data["password"]})
39     token = login_response.json()["token"]
40     headers = {"Authorization": f"Bearer {token}"}
41
42     response = requests.post(f"{BASE_URL}/tickets", json=test_ticket_data, headers=headers)
43     print(response.text, "Status code = ", response.status_code)
44     assert response.status_code == 201
45     assert "Ticket created successfully" in response.text
46
47 def test_submit_reply():
48     login_response = requests.post(f"{BASE_URL}/login", json={"email": test_user_data["email"], "password": test_user_data["password"]})
49     token = login_response.json()["token"]
50     headers = {"Authorization": f"Bearer {token}"}
51
52     # First, create a ticket to reply to
53     create_ticket_response = requests.post(f"{BASE_URL}/tickets", json=test_ticket_data, headers=headers)
54     ticket_id = create_ticket_response.json()["ticket_id"]
55
56     # Submit a reply to the created ticket
57     response = requests.post(f"{BASE_URL}/tickets/{ticket_id}/replies", json=test_reply_data, headers=headers)
58     print(response.text, "Status code = ", response.status_code)
59     assert response.status_code == 201
60     assert "Reply submitted successfully" in response.text
61
62 def test_mark_ticket_resolved():
63     login_response = requests.post(f"{BASE_URL}/login", json={"email": test_user_data["email"], "password": test_user_data["password"]})
64     token = login_response.json()["token"]
65     headers = {"Authorization": f"Bearer {token}"}
66
67     # First, create a ticket to mark as resolved
68     create_ticket_response = requests.post(f"{BASE_URL}/tickets", json=test_ticket_data, headers=headers)
69     ticket_id = create_ticket_response.json()["ticket_id"]
70
71     # Mark the created ticket as resolved
72     response = requests.put(f"{BASE_URL}/tickets/{ticket_id}/resolve", headers=headers)
73     print(response.text, "Status code = ", response.status_code)

```

```

pytesting.py > test_login_user
61
62 def test_mark_ticket_resolved():
63     login_response = requests.post(f"{BASE_URL}/login", json={"email": test_user_data["email"], "password": test_user_data["password"]})
64     token = login_response.json()["token"]
65     headers = {"Authorization": f"Bearer {token}"}
66
67     # First, create a ticket to mark as resolved
68     create_ticket_response = requests.post(f"{BASE_URL}/tickets", json=test_ticket_data, headers=headers)
69     ticket_id = create_ticket_response.json()["ticket_id"]
70
71     # Mark the created ticket as resolved
72     response = requests.put(f"{BASE_URL}/tickets/{ticket_id}/resolve", headers=headers)
73     print(response.text, "Status code = ", response.status_code)
74     assert response.status_code == 200
75     assert "Ticket marked as resolved" in response.text
76
77 #test_register_user()
78 #test_login_user()
79 #test_create_ticket()
80 #test_submit_reply()
81 test_mark_ticket_resolved()
82

```

# Command Prompt (Running Pytest)

```
200
PS C:\Users\leonb\OneDrive\Desktop\IITMonlinedeg\WAD_finalproject_leon> py pytesting.py
{
  "message": "User registered successfully"
  Status code = 201
PS C:\Users\leonb\OneDrive\Desktop\IITMonlinedeg\WAD_finalproject_leon> py pytesting.py
{
  "message": "User with this email already exists"
}
Traceback (most recent call last):
  File "C:\Users\leonb\OneDrive\Desktop\IITMonlinedeg\WAD_finalproject_leon\pytesting.py", line 77, in <module>
    test_register_user()
  File "C:\Users\leonb\OneDrive\Desktop\IITMonlinedeg\WAD_finalproject_leon\pytesting.py", line 28, in test_register_user
    assert response.status_code == 201
PS C:\Users\leonb\OneDrive\Desktop\IITMonlinedeg\WAD_finalproject_leon> py pytesting.py
{
  "message": "Logged in successfully",
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjoxfQ.ftWwQD9oDs-8zTWaaTAq9SYq21WDeq6Y_pz5NY6MIIo"
  Status code = 200
PS C:\Users\leonb\OneDrive\Desktop\IITMonlinedeg\WAD_finalproject_leon> py pytesting.py
{
  "message": "Ticket created successfully",
  "ticket_id": 1
}
Status code = 201
PS C:\Users\leonb\OneDrive\Desktop\IITMonlinedeg\WAD_finalproject_leon> py pytesting.py
{
  "message": "Reply submitted successfully"
}
Status code = 201
PS C:\Users\leonb\OneDrive\Desktop\IITMonlinedeg\WAD_finalproject_leon> py pytesting.py
{
  "message": "Ticket marked as resolved"
}
Status code = 200
PS C:\Users\leonb\OneDrive\Desktop\IITMonlinedeg\WAD_finalproject_leon> []
```