Software Engineering Project: Milestone 5

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

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.

INDEX

MILESTONE 5:

TEST CASES

1.1 App Testing

1.1.1 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

1.1.2 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!
```

1.1.3 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

```
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"]
```

1.2 Test Descriptions

1.2.1 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

1.2.2 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

1.2.3 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

1.2.4 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

1.3 Final Test Summary

Sample screenshots of testcase code are attached below.

Sample Screenshots - 1

```
14  headers = {'Content-type': 'application/json', 'web_token': admin_web_token, 'user_id': admin_user_id}
15
16  def test_register_page_with_fixture_get(test_client):
17      """
18      GIVEN a Flask application configured for testing
19      WHEN the '/api/v1/auth/register' page is requested (GET)
20      THEN check that the response is 405 i.e. method not allowed as no get method is defined for that endpoint
21      """
22      response = test_client.get(
23          f"/api/{API_VERSION}/auth/register",
24          headers=headers,
25      )
26      assert response.status_code == 405 # 405 METHOD NOT ALLOWED, GET not defined
```

Sample Screenshots - 2

```
29  def test_register_page_with_fixture_post_400_missing_data(test_client):
30      """
31      GIVEN a Flask application configured for testing
32      WHEN the '/api/v1/auth/register' page is requested (POST) with empty data fields
33      THEN check that the response is 400 i.e. bad request
34      """
35
36      response = test_client.post(
37          f"/api/{API_VERSION}/auth/register",
38          json={
39              "first_name": "",
40          },
41          headers=headers,
42      )
43      response = response.get_json()
44      assert response["status"] == 400 # bad request
45      assert "empty or invalid" in response["message"] # first_name is empty
46
```

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"
```

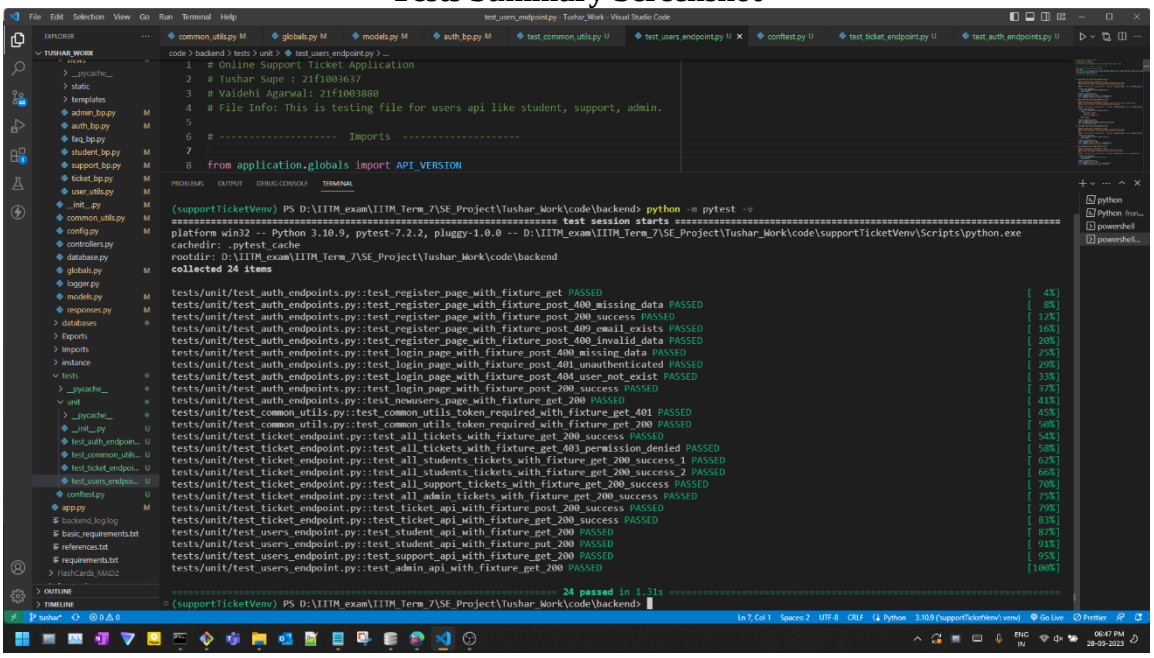
Sample Screenshots - 6

```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



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>
```

REFERENCES

[1] Software Engineering Project: Problem Statement

[2] PyTest Online Resource: TestDriven