Ticket System Integration with Discourse and Webhooks

A Project Report for the

Software Engineering

(Milestone 5)

Submitted By:

1. Aditya R	**21f1006862@ds.study.iitm.ac.in**
2. Ashutosh Kumar Barnwal	**21f1001709@ds.study.iitm.ac.in**
3. Kanishk Mishra	**21f1006627@ds.study.iitm.ac.in**
4. Nikhil Guru Venkatesh	**21f3000424@ds.study.iitm.ac.in**
5. Shubhankar Jaiswal	**21f1006828@ds.study.iitm.ac.in**
6. Sushobhan Bhargav	**22f1000948@ds.study.iitm.ac.in**
7. Utkarsh Kumar Yadav	**21f1006520@ds.study.iitm.ac.in**

IITM Online BS Degree Program,

Indian Institute of Technology, Madras, Chennai

Tamil Nadu, India, 600036

Contents

Milestone 5 - Test cases, test suite of the project

Introduction

These are the testing strategies and procedures for the application. Testing is a crucial aspect of software development to ensure the application's reliability, functionality, and security.

Testing Framework

The application's tests are written using the pytest framework, which is a widely used testing framework for Python applications. pytest offers a simple syntax for writing tests and provides powerful features for testing various aspects of the application.

5.1 App Testing on Discourse API:

Purpose: This test case verifies the functionality of the DiscourseSearchAPI class, which is responsible for searching Discourse topics and handling various operations related to Discourse posts.

5.1.1 Sample Fixture

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

soft-engg-project-jan-2024-se-jan-11 > Code > backend > tests > test_discourseAPIs.py > ...

```python
import pytest
from unittest.mock import patch
from flask import Flask
from flask_restful import Api
from application.api import DiscourseSearchAPI, DiscoursePostsAPI

@pytest.fixture
def app():
    app = Flask(__name__)
    api = Api(app)
    api.add_resource(DiscourseSearchAPI, '/api/discourse/search')
    api.add_resource(DiscoursePostsAPI, '/api/discourse/posts', '/api/discourse/posts/<int:post_id>')
    return app

@pytest.fixture
def client(app):
    return app.test_client()

```

5.1.2 Test for Searching Posts from Discourse

API being tested:

- http://127.0.0.1:5000/api/discourse/search

```python
class DiscourseSearchAPI(Resource):
    def get(self):
        query = request.args.get('q', '')
        discourse_url = 'http://localhost:4200/search.json'
        params = {'q': query}

        response = requests.get(discourse_url, params=params)

        if response.status_code == 200:
            return response.json(), 200
        else:
            return {'message': 'Error searching Discourse topics'}, response.status_code

```

Inputs:

- Request Method: GET

Expected Output:

- HTTP Status Code: 200
- JSON: {“message”: “Discourse posts searched successfully”}

Actual Output:

- HTTP Status Code: 200
- JSON: {“message”: “Discourse posts searched successfully”}

Result: Success

```python
def test_discourse_search_api(client):
    query = 'Test'
    with patch('requests.get') as mock_get:
        mock_get.return_value.status_code = 200
        mock_get.return_value.json.return_value = {"message": "Discourse posts searched successfully"}

        response = client.get(f'/api/discourse/search?q={query}')

        assert response.status_code == 200
        assert response.json == {"message": "Discourse posts searched successfully"}

```

5.1.3 Test for Retrieval of all Posts from Discourse

API being tested:

- http://127.0.0.1:5000/api/discourse/posts

```python
    # GET method to retrieve all posts from Discourse or a specific post
    def get(self, post_id=None):
        if post_id:
            discourse_url = f'http://127.0.0.1:4200/posts/{post_id}.json'
        else:
            discourse_url = 'http://127.0.0.1:4200/posts.json'

        response = requests.get(discourse_url)
        return response.json(), response.status_code

```

Inputs:

- Request Method: GET

Expected Output:

- HTTP Status Code: 200
- JSON: {“message”: “Discourse posts retrieved successfully”}

Actual Output:

- HTTP Status Code: 200
- JSON: {“message”: “Discourse posts retrieved successfully”}

Result: Success

```python
def test_discourse_posts_api_get(client):
    with patch('requests.get') as mock_get:
        mock_get.return_value.status_code = 200
        mock_get.return_value.json.return_value = {"message": "Discourse posts retrieved successfully"}

        response = client.get('/api/discourse/posts')

        assert response.status_code == 200
        assert response.json == {"message": "Discourse posts retrieved successfully"}

```

5.1.4 Test for Retrieval of Specific Post from Discourse

API being tested:

- http://127.0.0.1:5000/api/discourse/posts/<int:post_id>

```python
    # GET method to retrieve all posts from Discourse or a specific post
    def get(self, post_id=None):
        if post_id:
            discourse_url = f'http://127.0.0.1:4200/posts/{post_id}.json'
        else:
            discourse_url = 'http://127.0.0.1:4200/posts.json'

        response = requests.get(discourse_url)
        return response.json(), response.status_code

```

Inputs:

- Request Method: GET

Expected Output:

- HTTP Status Code: 200
- JSON: {“message”: “Discourse post retrieved successfully”}

Actual Output:

- HTTP Status Code: 200
- JSON: {“message”: “Discourse post retrieved successfully”}

Result: Success

```python
def test_discourse_posts_api_get_by_id(client):
    post_id = 23
    with patch('requests.get') as mock_get:
        mock_get.return_value.status_code = 200
        mock_get.return_value.json.return_value = {"message": "Discourse posts retrieved successfully"}

        response = client.get(f'/api/discourse/posts/{post_id}')

        assert response.status_code == 200
        assert response.json == {"message": "Discourse posts retrieved successfully"}

```

5.1.5 Test for Creating a New Post on Discourse

API being tested:

- http://127.0.0.1:5000/api/discourse/posts

```python
    # POST method to create a new post in Discourse
    def post(self):
        args = self.parser.parse_args()
        title = args['title']
        content = args['content']
        category = args.get('category')

        if category is not None:
            try:
                category = int(category)
            except ValueError:
                return {'message': 'Category must be an integer'}, 400

        discourse_url = 'http://127.0.0.1:4200/posts.json'
        headers = {
            'Api-Username': '22f1000948',
            'Api-Key': '614834da24a228d0f1e69c48c07ce122b8cb2fd461837c974f0fb9d7c17de6f3'
        }
        payload = {
            'title': title,
            'raw': content,
            'category': category
        }

        response = requests.post(discourse_url, headers=headers, json=payload)
        return response.json(), response.status_code

```

Inputs:

- Request Method: POST
- JSON: { 'title': 'Test API for creating Post on Discourse', 'content': 'Test API for creating content on Discourse', 'category': 4 }

Expected Output:

- HTTP Status Code: 200
- JSON: {“message”: “Discourse post created successfully”}

Actual Output:

- HTTP Status Code: 200
- JSON: {“message”: “Discourse post created successfully”}

Result: Success

```python
def test_discourse_posts_api_post(client):
    payload = {'title': 'Test API for creating Post on Discourse', 'content': 'Test API for creating content on Discourse', 'category': 4}
    with patch('application.api.requests.post') as mock_post:
        mock_post.return_value.status_code = 200
        mock_post.return_value.json.return_value = {"message": "Discourse post created successfully"}

        response = client.post('/api/discourse/posts', json=payload)

        assert response.status_code == 200
        assert response.json == {"message": "Discourse post created successfully"}

```

5.1.6 Test for Updating the Post on the Discourse

API being tested:

- http://127.0.0.1:5000/api/discourse/posts/<int:post_id>

```python
    # PUT method to update an existing post in Discourse
    def put(self, post_id):
        args = self.parser.parse_args()
        content = args['content']

        discourse_url = f'http://127.0.0.1:4200/posts/{post_id}.json'
        headers = {
            'Api-Username': '22f1000948',
            'Api-Key': '614834da24a228d0f1e69c48c07ce122b8cb2fd461837c974f0fb9d7c17de6f3'
        }
        payload = {
            'post': {
                'raw': content
            }
        }

        response = requests.put(discourse_url, headers=headers, json=payload)
        return response.json(), response.status_code

```

Inputs:

- Request Method: PUT
- JSON: { 'content': 'Test API for updating content on Discourse' }

Expected Output:

- HTTP Status Code: 200
- JSON: {“message”: “Discourse post updated successfully”}

Actual Output:

- HTTP Status Code: 200
- JSON: {“message”: “Discourse post updated successfully”}

Result: Success

```
62  def test_discourse_posts_api_put(client):
63      post_id = 23
64      payload = {'content': 'Test API for updating content on Discourse'}
65      with patch('application.api.requests.put') as mock_put:
66          mock_put.return_value.status_code = 200
67          mock_put.return_value.json.return_value = {"message": "Discourse post updated successfully"}
68
69          response = client.put(f'/api/discourse/posts/{post_id}', json=payload)
70
71          assert response.status_code == 200
72          assert response.json == {"message": "Discourse post updated successfully"}
```

5.2 App Testing on G-Chat Webhook:

Purpose: This test case verifies the functionality of the send_notification function, which sends notifications to Google Chat with a webhook endpoint for processing high/urgent priority tickets.

5.2.1 Sample Fixture

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

soft-engg-project-jan-2024-se-jan-11 > Code > backend > tests > test_GChat_webhook.py > ...

```
1  import pytest
2  from unittest.mock import patch
3  from application.routes import app, send_notification
4
5  @pytest.fixture
6  def client():
7      with app.test_client() as client:
8          yield client
9
```

5.2.2 Test for sending notifications

API being tested:

- http://127.0.0.1:5000/webhook

```python
def send_notification(ticket_title, ticket_description, webhook_url):
    message = {
        "text": f"New high priority/urgent ticket:\nTitle: {ticket_title}\nDescription: {ticket_description}"
    }
    response = requests.post(webhook_url, json=message)
    if response.status_code == 200:
        print("Notification sent successfully")
    else:
        print("Failed to send notification")

```

Inputs:

- Request Method: POST
- JSON: { "text": "New high priority/urgent ticket:\nTitle: Test Title\nDescription: Test Description" }

Expected Output:

- HTTP Status Code: 200
- JSON: {“message”: "Notification sent successfully"}

Actual Output:

- HTTP Status Code: 200
- JSON: {“message”: "Notification sent successfully"}

Result: Success

```python
def test_send_notification(client):
    with patch('application.routes.requests.post') as mock_post:
        mock_post.return_value.status_code = 200
        webhook_url = 'https://chat.googleapis.com/v1/spaces/AAAAVFgvcso/messages?key=AIzaSyDdI0hCZtE6vySjMm-WEfRq3CPzqKqqsHI&token=bJsKuUQVHQFNOLm
        send_notification("Test Title", "Test Description", webhook_url)

        mock_post.assert_called_once_with(
            webhook_url,
            json={"text": "New high priority/urgent ticket:\nTitle: Test Title\nDescription: Test Description"}
        )

```

5.2.3 Test for high-priority notifications

API being tested:

- http://127.0.0.1:5000/webhook

```python
@app.route('/webhook', methods=['POST'])
def webhook():
    data = request.json
    ticket_title = data.get('title')
    ticket_description = data.get('description')
    priority = data.get('priority')
    if priority == 'high' or priority == 'urgent':
        webhook_url = 'https://chat.googleapis.com/v1/spaces/AAAAVFgvcso/messages?key=AIzaSyDdI0hCZtE6vySjMm-WEfRq3CPzqKqqsHI&token=bJsKuUQVHQFNOLm
        send_notification(ticket_title, ticket_description, webhook_url)
    return '', 200

```

Inputs:

- Request Method: POST
- JSON: { 'title': 'Test Title', 'description': 'Test Description', 'priority': 'high' }

Expected Output:

- HTTP Status Code: 200

Actual Output:

- HTTP Status Code: 200

Result: Success

```
def test_webhook_high_priority(client):
    with patch('application.routes.requests.post') as mock_post:
        # Mock the JSON payload sent by the webhook
        mock_request = {'title': 'Test Title', 'description': 'Test Description', 'priority': 'high'}

        # Simulate the POST request to the webhook endpoint
        response = client.post('/webhook', json=mock_request)

        assert response.status_code == 200
        mock_post.assert_called_once()

```

5.2.4 Test for urgent-priority notifications

API being tested:

- http://127.0.0.1:5000/webhook

```
@app.route('/webhook', methods=['POST'])
def webhook():
    data = request.json
    ticket_title = data.get('title')
    ticket_description = data.get('description')
    priority = data.get('priority')
    if priority == 'high' or priority == 'urgent':
        webhook_url = 'https://chat.googleapis.com/v1/spaces/AAAAVFgvcso/messages?key=AIzaSyDdI0hCZtE6vySjMm-WEfRq3CPzqKqqsHI&token=bJsKuUQVHQFNOLm
        send_notification(ticket_title, ticket_description, webhook_url)
    return '', 200

```

Inputs:

- Request Method: POST
- JSON: { 'title': 'Test Title', 'description': 'Test Description', 'priority': 'urgent' }

Expected Output:

- HTTP Status Code: 200

Actual Output:

- HTTP Status Code: 200

Result: Success

```python
def test_webhook_urgent_priority(client):
    with patch('application.routes.requests.post') as mock_post:
        # Mock the JSON payload sent by the webhook
        mock_request = {'title': 'Test Title', 'description': 'Test Description', 'priority': 'urgent'}

        # Simulate the POST request to the webhook endpoint
        response = client.post('/webhook', json=mock_request)

        assert response.status_code == 200
        mock_post.assert_called_once()

```

5.3 Test Results:

All test cases have passed successfully.

```
============================= test session starts ==============================
platform linux -- Python 3.10.12, pytest-7.2.2, pluggy-1.4.0
rootdir: /mnt/d/SE_Project/soft-engg-project-jan-2024-se-jan-11/Code/backend
collected 8 items

tests/test_GChat_webhook.py ...                                          [ 37%]
tests/test_discourseAPIs.py .....                                        [100%]

============================== 8 passed in 1.54s ===============================
```

5.4 Test Environment

- **Language**: Python
- **Testing Framework**: pytest
- **Dependencies**: Flask, patch, requests

Conclusion

Testing is an essential aspect of the development process to ensure the application's reliability and functionality. By following the outlined test cases and procedures, we can ensure that the application meets its requirements and functions correctly in various scenarios.