Milestone-5 Report

By Team-FMM
Group No.2

Testing of User API

User API Test Details

Test Scenario	Testing of User API		Tester	Abhishek		Date	March 29, 2023	
ID	**Test Case**	**Pre-condition**	**Test Steps**	**Test Data**	**Expected Output**	**Post-condition**	**Actual Output**	**Status**
User_01	Testing if student is succesfully registered	Server must be running	Create a dictionary of user details with keys as username, password, email and role	a. Valid Username format b. Valid Password length and format c. Valid email format d. Post request to User API class	a. response Status code should be 201 b. username of response created should match with entered data	Delete the user data	As expected	Pass
User_02	Testing to successfully register staff credentials	a. Server must be running b. A subject tag should already be created with corresponding subject_id	Create a dictionary of user details with keys as username, password, email, role and subject_id	a. Valid Username string format b. Valid Password length and string format c. Valid email in string format d. role should be mandatorily specified as staff e. valid subject_id of created subject tag d. Post request to User API class	a. response status code should be 201 b. username of response created should match with entered data c.email of response created should match with entered data	Delete the staff data	As expected	Pass
User_03	Testing for Login	a. Server must be running b. A valid email already registered	Entering email in the router url	a. GET Request to USER API Class with email	a. response status code should be 200	No post-conditions	As expected	Pass
User_04	Test for editing user password	a. Server must be running b. A valid email already registered	a. creating dictionary of password and role b. Entering the email in parameter of url	a. A valid password b. Valid role of either student or staff c. PUT request of User API class	a. response status code should be 202 b. new response password data should match with the entered data in dictionary	No post-conditions	As expected	Pass

Pytest Functions and Results

```python
def test_create_user_student(client):
    # Set up test data
    new_user = {'username': 'testuser', 'password': 'testpassword',
                'email': 'testuser@gmail.com', 'role': 'student'}
    email = new_user['email']
    # Make POST request to create new user
    response = client.post('/api/register', json=new_user)

    # Check status code of response
    assert response.status_code == 201

    # Check response body for correct username
    assert json.loads(response.data)['username'] == 'testuser'
    response = client.delete(f'/api/login/{email}')


def test_create_user_staff(client):
    # Set up test data
    new_staff = {'username': 'teststaff', 'password': 'testpass',
                 'email': 'teststaff@gmail.com', 'role': 'staff', 'subject_id': 1}
    email_staff = new_staff['email']
    # Make POST request to create new user
    response = client.post('/api/register', json=new_staff)
    client.delete(f'/api/login/{email_staff}')
    # Check status code of response
    assert response.status_code == 201

    # Check response body for correct username
    assert json.loads(response.data)['username'] == 'teststaff'
    assert json.loads(response.data)['email'] == 'teststaff@gmail.com'


def test_login(client):
    email = 'user1@gmail.com'
    response = client.get(f'/api/login/{email}')
    assert response.status_code == 200


def test_put_user(client):
    email = 'user1@gmail.com'
    response = client.put(
        f'/api/login/{email}', json={'password': 'user1pass', 'role': 'student'})
    assert response.status_code == 202
    assert json.loads(response.data)['password'] == 'user1pass'
```

```
❯ pytest -s -v
============================================ test session starts ==
platform darwin -- Python 3.10.5, pytest-7.2.2, pluggy-1.0.0 -- /Users/
cachedir: .pytest_cache
rootdir: /Users/mendax/SE-Project-STS
collected 4 items

tests/test_user.py::test_create_user_student PASSED
tests/test_user.py::test_create_user_staff PASSED
tests/test_user.py::test_login PASSED
tests/test_user.py::test_put_user PASSED

============================================ 4 passed in 0.02s ===
```

Testing of TicketManager API

Ticket Manager API Test Details

Test Scenario	Testing of Ticket Manager API		Tester	Abhishek		Date	March 29, 2023	
ID	Test Case	Pre-condition	Test Steps	Test Data	Expected Output	Post-condition	Actual Output	Status
Ticket_01	Testing to create a ticket	Server must be running	Create a dictionary of ticket details with keys as title, description, secopndary_tag and user_id	a. Valid and unique title b. Valid description format c. Valid secondary_tag d. Post request to Ticket Manager API class	a. response Status code should be 201 b. title of response created should match with entered data	Delete the ticket data	As expected	Pass
Ticket_02	Testing to mark unresolved ticket as FAQ	a. Server must be running b. A user_id corresponding to staff must be created c. A unresolved ticket must be created	Create a dictionary of action details with keys as action and user_id	a. Valid action as faq b. Valid user_id corresponding to staff c.Put request to Ticket manager API class	a. response status code should be 400 b. error code must correspond to TICKET002	No post conditions	As expected	Pass
Ticket_03	Testing to mark resolved ticket as FAQ	a. Server must be running b. A user_id corresponding to staff must be created c. A ticket should be resolved for testing	Create a dictionary of action details with keys as action and user_id	a. Valid action as faq b. Valid user_id corresponding to staff c.Put request to Ticket manager API class	a. response status code should be 200 b. isFAQ should be marked as True for the ticket	Changing the isFAQ to False	As expected	Pass
Ticket_04	Testing for liking the ticket	a. Server must be running b. A valid user_id	a.Create a dictionary of action details with keys as action and user_id b. Make PUT request to Ticket Manager API Class and check the response c. Again make PUT request using same data and check if the like is decreased.	a. Valid action value as like b. Valid user_id already registered c.Put request to Ticket manager API class	a. response status code should be 200 b. After first PUT request like should be increased c. After second PUT request like should be decreased	No post-conditions	As expected	Pass
Ticket_05	Testing to search tickets	a. Server must be running b. Existing subject tag c. Existing secondary tag	a. Create a dictionary of paramter queries with keys as limit, TagName and search b. existing subject_tag in the GET request alongwith parameter	a. Valid numerical limit b. Valid secondary tag name c. Valid string format in search value d. valid subject tag name	response status code should be 200	No post-conditions	As expected	Pass

Pytest Functions and Results

```python
def test_get_tickets(client):
    params = {
        "limit": 3,
        "TagName": "week 1",
        "search": "titl"
    }
    url = "/api/subject/subject_1"
    res = client.get(url, query_string=params)
    json_data = json.loads(res.data)
    pprint(json_data)
    assert res.status_code == 200
    assert len(json_data) <= params["limit"]
    for obj in json_data:
        assert obj['subject_name'] == "subject_1"
        assert obj["sec_name"] == params['TagName']
        assert params['search'].lower() in obj['title'].lower()


def test_get_tickets_error(client):
    url = "/api/subject/NotASubject"
    res = client.get(url)
    assert res.status_code == 404
    json_data = json.loads(res.data)
    assert json_data['error_code'] == 'TICKET006'


def test_create_ticket(client):
    # Set up test data
    new_ticket = {
        "title": "title-ba",
        "description": "desc-ba",
        "secondary_tag": "week-1",
        "user_id": 1
    }
    tag_name = "BA"
    # Make POST request to create new ticket
    response = client.post(f'/api/subject/{tag_name}', json=new_ticket)

    # Check status code of response
    assert response.status_code == 201

    # Check response body for correct username      You, 5 days ago • test
    ticket_obj = json.loads(response.data)
    ticket_id = ticket_obj['ticket_id']
    client.delete(f'/api/subject/ticket/{ticket_id}')
    assert ticket_obj['title'] == 'title-ba'
```

```python
def test_mark_resolved_ticket_as_faq(client):
    # Preconditions-corresponding to ticket_id=1, the ticket_Sta
    action_detail = {
        "action": "faq",
        "user_id": 4
    }

    ticket_id = 1

    response = client.put(
        f'/api/subject/ticket/{ticket_id}', json=action_detail)
    ticket_obj = json.loads(response.data)
    assert response.status_code == 200
    assert ticket_obj['isFAQ'] == True
    obj = Ticket.query.filter_by(ticket_id=ticket_id).first()
    obj.isFAQ = False
    db.session.commit()


def test_mark_unresolved_ticket_as_faq(client):
    # Preconditions-corresponding to ticket_id=2, the ticket_Sta
    action_detail = {
        "action": "faq",
        "user_id": 4
    }

    ticket_id = 2
    response = client.put(
        f'/api/subject/ticket/{ticket_id}', json=action_detail)
    error = json.loads(response.data)['error_code']
    assert response.status_code == 400
    assert error == "TICKET002"


def test_ticket_like(client):
    action_detail = {
        "action": "like",
        "user_id": 1
    }
    ticket_id = 1

    response = client.put(
        f'/api/subject/ticket/{ticket_id}', json=action_detail)
    ticket_obj = json.loads(response.data)
      You, 6 minutes ago • Uncommitted changes
    assert response.status_code == 200
    assert ticket_obj['likes'] == 1
    response = client.put(
        f'/api/subject/ticket/{ticket_id}', json=action_detail)
    ticket_obj = json.loads(response.data)
    assert response.status_code == 200
    assert ticket_obj['likes'] == 0
```

```
❯ pytest -s -v
========== test session starts ==========
platform darwin -- Python 3.10.5, pytest-7.2.2, pluggy-1.0.0 -- /User
cachedir: .pytest_cache
rootdir: /Users/mendax/SE-Project-STS
collected 6 items

tests/test_ticket.py::test_get_tickets PASSED
tests/test_ticket.py::test_get_tickets_error PASSED
tests/test_ticket.py::test_create_ticket PASSED
tests/test_ticket.py::test_mark_resolved_ticket_as_faq PASSED
tests/test_ticket.py::test_mark_unresolved_ticket_as_faq PASSED
tests/test_ticket.py::test_ticket_like PASSED

========== 6 passed in 0.04s ==========
```

Testing of Tag Manager API

Tag Manager API Test Details

Test Scenario	Testing of Tag Manager API		Tester	Kaustav Goswami		Date	March 30, 2023	
ID	**Test Case**	**Pre-condition**	**Test Steps**	**Test Data**	**Expected Output**	**Post-condition**	**Actual Output**	**Status**
Tag_01	Testing to retrieve data about Subject tag	a. Server must be running b. A Subject tag must be created	GET request to Tag API with **tag_type=subject** in the url parameter	Valid tag_type	a. Response status should be 200 b. Valid Subject tag data or empty list if no matching data is found	No post condition	As expected	Pass
Tag_02	Testing to retrieve data about Secondary tag	a. Server must be running b. A Secondary tag must be created	GET request to Tag API with **tag_type=secondary** in the url parameter	Valid tag_type	a. Response status should be 200 b. Valid Secondary tag data or empty list if no matching data is found	No post condition	As expected	Pass
Tag_03	Testing to create a Subject Tag	a. Server must be running	a. Create dictionary with key as **tag_name** b. Make a POST request to Tag API with the above data and **tag_type=subject** in the url parameter	Unique subject tag name	a. 201 Response status for new tag data created b. The value in **subject_name** in the response body should match with the entered data	Delete the dummy data that is created	As expected	Pass
Tag_04	Testing to create a Secondary Tag	a. Server must be running	a. Create dictionary with key as **tag_name** b. Make a POST request to Tag API with the above data and **tag_type=secondary**in the url parameter	Unique secondary tag name	a. 201 Response status for new tag data created b. The value in **sec_name** in the response body should match with the entered data	Delete the dummy data that is created	As expected	Pass
Tag_05	Testing to edit a Subject Tag	a. Server must be running	a. Create dictionary with key as **tag_name** b. Make a POST request to Tag API with the above data and **tag_type=subject** in the url parameter c. Make a PUT request to Tag API with a different value in the **tag_name**	Unique subject tag name	a. 201 Response status for new tag data created b. The value in **subject_name** in the response body should match with the entered data c. The updated value of **subject_name** in the response body should match with the updated value given	Delete the dummy data that is created	As expected	Pass
Tag_06	Testing to edit a Secondary Tag	a. Server must be running	a. Create dictionary with key as **tag_name** b. Make a POST request to Tag API with the above data and **tag_type=secondary**in the url parameter c. Make a PUT request to Tag API with a different value in the **tag_name**	Unique secondary tag name	a. 201 Response status for new tag data created b. The value in **sec_name** in the response body should match with the entered data c. The updated value of **sec_name** in the response body should match with the updated value given	Delete the dummy data that is created	As expected	Pass

Pytest Functions and Results

```python
def test_subject_tag(client):
    # Make a GET request to Tag API
    response = client.get(f'/api/tag/subject/1')
    # Check status of response
    assert response.status_code == 200


def test_secondary_tag(client):
    # Make a GET request to Tag API
    response = client.get(f'/api/tag/secondary/1')
    # Check status of response      Kaustav-Goswami9, 5 days ag
    assert response.status_code == 200


def test_create_subject_tag(client):
    # Set up test data
    new_tag = {'tag_name': 'MAD-1'}
    # Make POST request to create new tag
    response = client.post('/api/tag/subject', json=new_tag)

    # Check status code of response
    assert response.status_code == 201

    # Check response body for correct tag-name
    data = json.loads(response.data)
    assert data['subject_name'] == 'MAD-1'
    # Remove the Test data from DB
    db.session.delete(Subject_Tag.query.filter_by(
        subject_name=data['subject_name']).first())
    db.session.commit()


def test_create_secondary_tag(client):
    # Set up test data
    new_tag = {'tag_name': 'Quiz-1'}
    # Make POST request to create new tag
    response = client.post('/api/tag/secondary', json=new_tag)

    # Check status code of response
    assert response.status_code == 201

    # Check response body for correct tag-name
    data = json.loads(response.data)
    assert data['sec_name'] == 'Quiz-1'
    # Remove the Test data from DB
    client.delete(f'/api/tag/secondary/{data["sec_id"]}')
```

```python
def test_edit_subject_tag(client):
    # Set up test data
    new_tag = {'tag_name': 'MAD-2'}
    # Make POST request to create new tag
    response = client.post('/api/tag/subject', json=new_tag)

    # Check status code of response
    assert response.status_code == 201

    data = json.loads(response.data)
    # Set up edit data
    edit_data = {'tag_name': 'mad-2'}
    # Make PUT request to edit that tag
    response = client.put(f'/api/tag/subject/{data["subject_id"]}',
                          json=edit_data)

    # Check status code of response
    assert response.status_code == 202

    data = json.loads(response.data)
    assert data['subject_name'] == 'mad-2'
    # Remove the Test data from DB
    db.session.delete(Subject_Tag.query.filter_by(
        subject_name=data['subject_name']).first())
    db.session.commit()


def test_edit_secondary_tag(client):
    # Set up test data
    new_tag = {'tag_name': 'Quiz-2'}
    # Make POST request to create new tag
    response = client.post('/api/tag/secondary', json=new_tag)

    # Check status code of response
    assert response.status_code == 201

    data = json.loads(response.data)
    # Set up edit data
    edit_data = {'tag_name': 'quiz-2'}
    # Make PUT request to edit that tag
    response = client.put(f'/api/tag/secondary/{data["sec_id"]}',
                          json=edit_data)

    # Check status code of response
    assert response.status_code == 202

    data = json.loads(response.data)
    assert data['sec_name'] == 'quiz-2'
    # Remove the Test data from DB
    client.delete(f'/api/tag/secondary/{data["sec_id"]}')
```

```
==========================================
❯ pytest -s -v
========== test session starts ==========
platform darwin -- Python 3.10.5, pytest-7.2.2, pluggy-1.0.0 -- /Users/mendax/SE
cachedir: .pytest_cache
rootdir: /Users/mendax/SE-Project-STS
collected 6 items

tests/test_tag.py::test_subject_tag PASSED
tests/test_tag.py::test_secondary_tag PASSED
tests/test_tag.py::test_create_subject_tag PASSED
tests/test_tag.py::test_create_secondary_tag PASSED
tests/test_tag.py::test_edit_subject_tag PASSED
tests/test_tag.py::test_edit_secondary_tag PASSED

========== 6 passed in 0.03s ==========
```

Testing of Roles Manager API

Roles API Test Details

Test Scenario	Testing of Roles API		Tester	Kaustav Goswami		Date	March 30, 2023	
ID	**Test Case**	**Pre-condition**	**Test Steps**	**Test Data**	**Expected Output**	**Post-condition**	**Actual Output**	**Status**
Role_01	Testing to retrieve data about all the staff	a. Server must be running	GET request to Role API	No test data is required	a. Response status should be 200 b. Valid Staff data or empty list if no matching data is found	No post conditions	As expected	Pass
Role_02	Testing to retrieve data about all the approved staff	a. Server must be running	GET request to Role API with **status=1** in the query parameter	Correct value for status query parameter	a. Response status should be 200 b. Valid approved Staff data or empty list if no matching data is found	No post conditions	As expected	Pass
Role_03	Testing to retrieve data about all the pending approval of staff	a. Server must be running	GET request to Role API with **status=0** in the query parameter	Correct value for status query parameter	a. Response status should be 200 b. Valid pending approval Staff data or empty list if no matching data is found	No post conditions	As expected	Pass
Role_04	Testing to edit already existing staff data	a. Server must be running	a. Create dummy staff data dictionary with keys as username, password, email, **role='staff' and subject_id** b. Make a POST request to the Role API with the above dummy data c. Enter the user_id generated in the url parameter of PUT method of Role API and in the body pass **status=True**	a. Valid Staff data b. Valid update data for PUT request	a. 201 Response status for new staff data created b. 202 Response status for successful edit c. Response Body of PUT method should have **approved=True**	Delete the dummy data that is created	As expected	Pass
Role_05	Testing to delete already existing staff data	a. Server must be running	a. Create dummy staff data dictionary with keys as username, password, email, **role='staff' and subject_id** b. Make a POST request to the Role API with the above dummy data c. Enter the user_id generated in the url parameter of DELETE method of Role API	Valid Staff data	a. 201 Response status for new staff data created b. 200 Response status for successful delete	No post conditions	As expected	Pass

Pytest Functions and Results

```python
def test_role_no_query(client):
    # Make a GET request to Role API
    response = client.get('/api/role')
    # Check status of response
    assert response.status_code == 200


def test_role_query_status_true(client):
    # Make a GET request to Role API
    response = client.get('/api/role?status=1')
    # Check status of response
    assert response.status_code == 200


def test_role_query_status_false(client):
    # Make a GET request to Role API
    response = client.get('/api/role?status=0')
    # Check status of response
    assert response.status_code == 200

    Kaustav-Goswami9, 5 days ago • Updated Test cases for Role API
def test_edit_role(client):
    # Dummy Staff data
    staff_data = {"username": "dummy_satff", "email": "dummy_satff@gmail.com",
                  "password": "abcd3", "role": "staff", 'subject_id': 1}
    # Create a new Staff account
    response = client.post('/api/register', json=staff_data)
    # Check for response status code
    assert response.status_code == 201

    # Extract user-id and make a PUT request
    staff_data = json.loads(response.data)
    response = client.put(f"/api/role/{staff_data['user_id']}",
                          json={'status': True})
    # Check for response status code
    assert response.status_code == 202
    # Check for response data
    assert json.loads(response.data)['approved'] == True

    # Remove Dummy data from DB
    client.delete(f"/api/role/{staff_data['user_id']}")


def test_delete_role(client):
    # Dummy Staff data
    staff_data = {"username": "dummy_satff", "email": "dummy_satff@gmail.com",
                  "password": "abcd3", "role": "staff", 'subject_id': 1}
    # Create a new Staff account
    response = client.post('/api/register', json=staff_data)
    # Check for response status code
    assert response.status_code == 201

    # Extract user-id and make a DELETE request
    staff_data = json.loads(response.data)
    response = client.delete(f"/api/role/{staff_data['user_id']}")
    # Check for response status code
    assert response.status_code == 200
```

```
==
❯ pytest -s -v
========================================= test session starts ==============
platform darwin -- Python 3.10.5, pytest-7.2.2, pluggy-1.0.0 -- /Users/men
cachedir: .pytest_cache
rootdir: /Users/mendax/SE-Project-STS
collected 5 items

tests/test_role.py::test_role_no_query PASSED
tests/test_role.py::test_role_query_status_true PASSED
tests/test_role.py::test_role_query_status_false PASSED
tests/test_role.py::test_edit_role PASSED
tests/test_role.py::test_delete_role PASSED

========================================= 5 passed in 0.03s ================
```

Testing of Response API

Response API Test Details

Test Scenario	Testing of Response API		Tester	ANDIBOYINA MOURYA CHAKRADHAR NAGESH		Date	April 2, 2023	
ID	Test Case	Pre-condition	Test Steps	Test Data	Expected Output	Post-condition	Actual Output	Status
Response_01	Testing to create a response	a. Server must be running b. Existing user_id c. Existing ticket_id	a. Create a dictionary of response details with keys as user_id and response b. POST request to Response API with ticket_id in POST url	a. Valid user_id b. Valid response format c. Valid existing ticket_id d. Post request to Response API class	a. response Status code should be 201 b. response should be created within entered ticket_id	Deleting the reponse	As expected	Pass
Response_02	Testing to get response	a. Server must be running b. Existing ticket_id c. Some responses to the given ticket_id	GET request to Response API	a. Valid ticket_id	a. response status code should be 200 b. The length of responses for the ticket_id should match as expected	No post conditions	As expected	Pass
Response_03	Testing to get response error	a. Server must be running b. Invalid ticket_id	GET request to Response API	a. Numerical invalid ticket_id	a. response status code should be 404 b. error code correspond to RESPONSE001	No post conditions	As expected	Pass
Response_04	Testing for different response creation error	a. Server must be running b. A invalid ticket_id c. Existing response to a ticket	a. Create a dictionary of response details with keys as user_id and response b. POST request to Response API with invalid ticket_id in url c. Creating dictionary with missing details	a. Numerical invalid ticket_id	a. response status code should be 404 for invalid ticket_id b. reponse status code for other requests should be 400 c. error codes as defined	No post-conditions	As expected	Pass
Response_05	Testing to marking response as Answer and resolving	a. Server must be running b. Existing ticket_id c. Existing response_id	a. Create a dictionary of payload with keys as isAnswer and ticket_status b. existing ticket_id and response _id in the PUT request to Response API class	a. isAnswer field should be boolean b. ticket_status should either be resolved or unresolved c. Valid ticket_id d. Valid response_id	response status code should be 200	No post-conditions	As expected	Pass
Response_06	Testing for different response editing error	a. Server must be running b. A invalid ticket_id c. Existing response to a ticket	a. Create a dictionary of payload with keys as isAnswer and ticket_status b. PUT request to Response API with invalid ticket_id or response_id in url c. Creating dictionary with missing details	a. Numerical invalid ticket_id	a. response status code should be 404 for invalid ticket_id or response_id b. reponse status code for other requests should be 400 c. error codes as defined	No post-conditions	As expected	Pass

Pytest Functions and Results

```python
def test_get_response(client):
    url = "http://127.0.0.1:5500/api/response/1"
    res = client.get(url)
    json_data = json.loads(res.data)
    pprint(json_data)
    assert res.status_code == 200
    assert json_data['ticket_id'] == 1


def test_get_response_error(client):
    url = "http://127.0.0.1:5500/api/response/9999"
    res = client.get(url)
    json_data = json.loads(res.data)
    pprint(json_data)
    assert res.status_code == 404
    assert json_data['error_code'] == 'RESPONSE001'


def test_post_response(client):
    url = "http://127.0.0.1:5500/api/response/1"
    payload = {
        "user_id": 1,
        "response": "Test Response"
    }
    res = client.post(url, json=payload)
    json_data = json.loads(res.data)
    assert res.status_code == 201
    assert json_data['ticket_id'] == 1


def test_post_response_error(client):
    # 404 status code test
    url = "http://127.0.0.1:5500/api/response/9999"
    payload = {
        "user_id": 1,
        "response": "Test Response"
    }
    res = client.post(url, json=payload)
    json_data = json.loads(res.data)
    assert res.status_code == 404
    assert json_data['error_code'] == 'RESPONSE001'
    # 400 status code test
    url = "http://127.0.0.1:5500/api/response/1"
    payload = {
        "response": "Test Response"
    }
    res = client.post(url, json=payload)
    json_data = json.loads(res.data)
    assert res.status_code == 400
    assert json_data['error_code'] == 'RESPONSE003'

    url = "http://127.0.0.1:5500/api/response/1"
    payload = {
        "user_id": 1
    }
    res = client.post(url, json=payload)
    json_data = json.loads(res.data)
    assert res.status_code == 400
    assert json_data['error_code'] == 'RESPONSE003'
```

```python
def test_put_response(client):
    url = "http://127.0.0.1:5500/api/response/1/4"
    payload = {
        "isAnswer": True,
        "ticket_status": "resolved"
    }
    res = client.put(url, json=json.dumps(payload))
    json_data = json.loads(res.data)
    assert res.status_code == 200


def test_put_response_error_1(client):
    url = "http://127.0.0.1:5500/api/response/9999/4"
    payload = {
        "isAnswer": False,
        "ticket_status": "unresolved"
    }
    res = client.put(url, json=json.dumps(payload))
    json_data = json.loads(res.data)
    assert res.status_code == 404
    assert json_data['error_code'] == 'RESPONSE001'


def test_put_response_error_2(client):
    url = "http://127.0.0.1:5500/api/response/1/999"
    payload = {
        "isAnswer": False,
        "ticket_status": "unresolved"
    }
    res = client.put(url, json=json.dumps(payload))
    json_data = json.loads(res.data)
    assert res.status_code == 404
    assert json_data['error_code'] == 'RESPONSE002'


def test_put_response_error_3(client):
    url = "http://127.0.0.1:5500/api/response/1/4"
    payload = {
        "ticket_status": "resolved"
    }
    res = client.put(url, json=json.dumps(payload))
    json_data = json.loads(res.data)
    assert res.status_code == 400
    assert json_data['error_code'] == 'RESPONSE004'


def test_put_response_error_4(client):
    url = "http://127.0.0.1:5500/api/response/1/4"
    payload = {
        "isAnswer": False,
    }
    res = client.put(url, json=json.dumps(payload))
    json_data = json.loads(res.data)
    assert res.status_code == 400
    assert json_data['error_code'] == 'RESPONSE005'
```

```python
def test_put_response_error_5(client):
    url = "http://127.0.0.1:5500/api/response/1/4"
    payload = {
        "isAnswer": False,
        "ticket_status": "hello"
    }
    res = client.put(url, json=json.dumps(payload))
    json_data = json.loads(res.data)
    assert res.status_code == 400
    assert json_data['error_code'] == 'RESPONSE006'
```

```
❯ pytest -s -v
========== test session starts ==========
platform darwin -- Python 3.10.5, pytest-7.2.2, pluggy-1.0.0 -- /Users/mendax/SE-Project-STS/.env/bir
cachedir: .pytest_cache
rootdir: /Users/mendax/SE-Project-STS
collected 10 items

tests/test_response.py::test_get_response PASSED
tests/test_response.py::test_get_response_error PASSED
tests/test_response.py::test_post_response PASSED
tests/test_response.py::test_post_response_error PASSED
tests/test_response.py::test_put_response {'isAnswer': True, 'ticket_status': 'resolved'}
PASSED
tests/test_response.py::test_put_response_error_1 {'isAnswer': False, 'ticket_status': 'unresolved'}
PASSED
tests/test_response.py::test_put_response_error_2 PASSED
tests/test_response.py::test_put_response_error_3 {'ticket_status': 'resolved'}
PASSED
tests/test_response.py::test_put_response_error_4 {'isAnswer': False}
PASSED
tests/test_response.py::test_put_response_error_5 {'isAnswer': False, 'ticket_status': 'hello'}
PASSED

========== 10 passed in 0.04s ==========
```