MILESTONE 5

# Test Cases

# Introduction

This document outlines the test cases for the GenAI-enhanced programming learning environment project, aimed at integrating Generative AI into educational platforms like the SEEK portal used in the IITM BS degree program. The project seeks to leverage GenAI to enhance learner engagement, provide effective feedback, and support problem-solving activities. Given the importance of ensuring reliable and seamless integration of GenAI technologies, this document provides a comprehensive set of test cases designed to validate the functionality, performance, and reliability of the system components, including APIs, GenAI interactions, and user interface elements. These test cases serve as a roadmap for systematically evaluating the application's capabilities and ensuring a high-quality user experience.

# Testing Framework

The testing framework for this project is built upon Python’s pytest, a robust and flexible testing tool known for its simplicity and scalability. The framework covers a wide range of tests, including unit tests to verify individual functions, integration tests to ensure components work together as expected, and system tests to validate the application as a whole. Each test case is meticulously designed to assess specific functionalities, such as API responses, data processing, GenAI model integration, and user interactions. The framework also incorporates continuous integration practices to automate testing, allowing for rapid identification and resolution of issues during development. This structured approach ensures that the system not only meets functional requirements but also maintains performance standards under various conditions.

# Test Cases

## 1. Home API

```python
import pytest
from flask import Flask, jsonify
from flask.testing import FlaskClient
from application import app, db
from application.models import User, Course, Assignment, Announcement, Lecture, CourseDocs
from flask_bcrypt import Bcrypt

bcrypt = Bcrypt()

@pytest.fixture
def client():
    app.config['TESTING'] = True
    with app.test_client() as client:
        with app.app_context():
            db.create_all()
        yield client
        with app.app_context():
            db.drop_all()

def test_home_api(client: FlaskClient):
    response = client.post('/')
    assert response.status_code == 200
    assert response.json == {"message": "Welcome to the App"}
```

**Test Case:** Home API Welcome Message

- ***API Endpoint:*** */*
- ***Request Method:*** POST
- ***Inputs:*** None
- ***Expected Output:***
  - HTTP Status Code: 200
  - JSON: {"message": "Welcome to the App"}
- ***Actual Output:***
  - HTTP Status Code: 200
  - JSON: {"message": "Welcome to the App"}

# 2. Registration API

```python
def test_register_new_user(client: FlaskClient):
    data = {
        "username": "iitmbsstudent",
        "password": "iitmbs123",
        "email": "student@ds.study.iitm.ac.in",
        "mob": "1234567890",
        "name": "Soft Engg Student", ##############################################
        "courses": ["python"]
    }
    response = client.post('/register', json=data)
    assert response.status_code == 201
    assert response.json == {"message": "User created successfully", "code": 201}

def test_register_existing_user(client: FlaskClient):
    # First, register a user
    data = {
        "username": "iitmbsstudent",
        "password": "iitmbs123",
        "email": "student@ds.study.iitm.ac.in",
        "mob": "1234567890",
        "name": "Soft Engg Student"
    }
    client.post('/register', json=data)
    response = client.post('/register', json=data)
    assert response.status_code == 400
    assert response.json == {"error": "This username is already taken", "code": 400}
```

**Test Case:** Register New User

- ***API Endpoint:*** /register
- ***Request Method:*** POST
- ***Expected Output:***
  - HTTP Status Code: 201
  - JSON: {"message": "User created successfully", "code": 201}
- ***Actual Output:***
  - HTTP Status Code: 201
  - JSON: {"message": "User created successfully", "code": 201}
- ***Result:*** Success

**Test Case:** Register Existing User

- ***API Endpoint:*** /register
- ***Request Method:*** POST
- ***Expected Output:***
  - HTTP Status Code: 400
  - JSON: {"error": "This username is already taken", "code": 400}
- ***Actual Output:***
  - HTTP Status Code: 400
  - JSON: {"error": "This username is already taken", "code": 400}
- ***Result:*** Success

# 3. Login API

```python
def test_valid_login(client: FlaskClient):
    data = {
        "username": "iitmbsstudent",
        "password": "iitmbs123",
        "email": "student@ds.study.iitm.ac.in",
        "mob": "1234567890",
        "name": "Soft Engg Student"
    }
    client.post('/register', json=data)

    # Now, test login
    login_data = {
        "username": "iitmbsstudent",
        "password": "iitmbs123"
    }
    response = client.post('/login', json=login_data)
    assert response.status_code == 200
    assert "token" in response.json

def test_invalid_login(client: FlaskClient):
    login_data = {
        "username": "invalidiitmuser",
        "password": "wrongiitmbspassword"
    }
    response = client.post('/login', json=login_data)
    assert response.status_code == 400
    assert response.json == {"error": "Invalid credentials", "code": 400}
```

**Test Case:** Valid Login

- ***API Endpoint:*** /login
- ***Request Method:*** POST
- ***Expected Output:***
  - HTTP Status Code: 200
  - JSON: {"token": "<jwt_token>", "code": 200}
- ***Actual Output:***
  - HTTP Status Code: 200
  - JSON: {"token": "<jwt_token>", "code": 200}
- ***Result:*** Success

**Test Case:** Invalid Login

- ***API Endpoint:*** /login
- ***Request Method:*** POST
- ***Expected Output:***
  - HTTP Status Code: 400
  - JSON: {"error": "Invalid credentials", "code": 400}
- ***Actual Output:***
  - HTTP Status Code: 400
  - JSON: {"error": "Invalid credentials", "code": 400}
- ***Result:*** Success

# 4. Dashboard API

```python
def test_dashboard_api(client: FlaskClient):
    # We are assuming user with id 1 exists
    response = client.get('/dashboard')
    assert response.status_code == 200
    assert "dashboard" in response.json
```

**Test Case:** Fetch Dashboard Data

- ***API Endpoint:*** /dashboard
- ***Request Method:*** GET
- ***Expected Output:***
  - HTTP Status Code: 200
  - {"dashboard": {"user": {"id": 1,"name": "Test User"},"deadlines": [...],"announcements": [...]},"code": 200}
- ***Actual Output:***
  - HTTP Status Code: 200
  - JSON: Matches expected output
- ***Result:*** Success

# 5. Study API

```python
def test_study_api(client: FlaskClient):
    response = client.get('/study')
    assert response.status_code == 200
    assert "study" in response.json
```

**Test Case:** Fetch Study Content

- ***API Endpoint:*** /study
- ***Request Method:*** GET
- ***Expected Output:***
  - HTTP Status Code: 200
  - {"study": [{"course_id": 1,"course_name": "Course Name","course_description": "Course Description"}],"code": 200}
- ***Actual Output:***
  - HTTP Status Code: 200
  - JSON: Matches expected output
- ***Result:*** Success

```python
def test_create_new_course(client: FlaskClient):
    data = {
        "course_name": "Python",
        "course_description": "Python is a Programming Langyage.",
        "start_date": "2024-05-28",
        "end_date": "2024-09-01"
    }
    response = client.post('/study', json=data)
    assert response.status_code == 201
    assert response.json == {"message": "Course created successfully", "code": 201}
```

**Test Case:** Create New Course

- ***API Endpoint:*** /study
- ***Request Method:*** POST
- ***Expected Output:***
  - HTTP Status Code: 201
  - JSON: {"message": "Course created successfully", "code": 201}
- ***Actual Output:***
  - HTTP Status Code: 201
  - JSON: {"message": "Course created successfully", "code": 201}
- ***Result:*** Success

# 6. Lectures API

```python
def test_fetch_lectures(client: FlaskClient):
    # We are assuming course with id 1 exists
    response = client.get('/study/lectures')
    assert response.status_code == 200
    assert "lectures" in response.json
```

**Test Case:** Fetch Lectures

- ***API Endpoint:*** /study/lectures
- ***Request Method:*** GET
- ***Expected Output:***
  - HTTP Status Code: 200
  - {"lectures": [{"lecture_id": 1,"lecture_title": "Lecture Title","lecture_link": "http://lecture.link","lecture_date": "2024-01-15","description": "Lecture description"}],"code": 200}
- ***Actual Output:***
  - HTTP Status Code: 200
  - JSON: Matches expected output
- ***Result:*** Success

```python
def test_create_new_lecture(client: FlaskClient):
    data = {
        "lecture_title": "Introduction to Python",
        "lecture_link": "https://www.youtube.com/watch?v=8ndsDXohLMQ&list=PLZ2ps__7DhBb2cXAu5PevO_mzgS3Fj3Fs",
        "lecture_date": "2024-05-28",
        "lecture_description": "Introductory lecture to Python Programming Language"
    }
    response = client.post('/study/lectures', json=data)
    assert response.status_code == 201
    assert response.json == {"message": "Lecture created successfully", "code": 201}
```

**Test Case:** Create New Lecture

- ***API Endpoint:*** /study/lectures
- ***Request Method:*** POST
- ***Expected Output:***
  - HTTP Status Code: 201
  - JSON: {"message": "Lecture created successfully", "code": 201}
- ***Actual Output:***
  - HTTP Status Code: 201
  - JSON: {"message": "Lecture created successfully", "code": 201}
- ***Result:*** Success

# 7. Profile API

```python
def test_fetch_user_profile(client: FlaskClient):
    response = client.get('/profile')
    assert response.status_code == 200
    assert "profile" in response.json
```

**Test Case:** Fetch User Profile

- ***API Endpoint:*** /profile
- ***Request Method:*** GET
- ***Expected Output:***
  - HTTP Status Code: 200
  - {"profile": {"first_name": "Test","last_name": "User","email": "test@example.com","mob": "1234567890","subjects_taken": ["Course1", "Course2"]},"code": 200}
- ***Actual Output:***
  - HTTP Status Code: 200
  - JSON: Matches expected output
- ***Result:*** Success

# 8. Course Documents API

```python
def test_fetch_course_documents(client: FlaskClient):
    # We are assuming course with id 1 exists
    response = client.get('/study/course_docs')
    assert response.status_code == 200
    assert "course_docs" in response.json
```

**Test Case:** Fetch Course Documents

- ***API Endpoint:*** /study/course_docs
- ***Request Method:*** GET
- ***Expected Output:***
  - HTTP Status Code: 200
  - {"course_docs": [{"document_name": "Document 1","document_link": "http://document1.link"}],"code": 200}
- ***Actual Output:***
  - HTTP Status Code: 200
  - JSON: Matches expected output
- ***Result:*** Success

```python
def test_add_new_document(client: FlaskClient):
    data = {
        "doc_title": "Transcript",
        "doc_link": "https://drive.google.com/file/d/1kYbJ8wTIkokMO3rZz0RZLVnRYCaMhBsJ/view?usp=sharing"
    }
    response = client.post('/study/course_docs', json=data)
    assert response.status_code == 201
    assert response.json == {"message": "Document created successfully", "code": 201}
```

**Test Case:** Add New Document

- ***API Endpoint:*** /study/course_docs
- ***Request Method:*** POST
- ***Expected Output:***
  - HTTP Status Code: 201
  - JSON: {"message": "Document created successfully", "code": 201}
- ***Actual Output:***
  - HTTP Status Code: 201
  - JSON: {"message": "Document created successfully", "code": 201}
- ***Result:*** Success

# 9. Practice Assignments API

```
139 def test_fetch_practice_assignments(client: FlaskClient):
140     response = client.get('/study/practice')
141     assert response.status_code == 200
142     assert "practice_assignments" in response.json
```

**Test Case:** Fetch Practice Assignments

- ***API Endpoint:*** /study/practice
- ***Request Method:*** GET
- ***Expected Output:***
  - HTTP Status Code: 200
  - {"practice_assignments": [{"assignment_id": 1,"course_id": 1,"title": "Practice Assignment 1","description": "Description","due_date": "2024-03-01","created_at": "2024-01-10"}],"code": 200}
- ***Actual Output:***
  - HTTP Status Code: 200
  - JSON: Matches expected output
- ***Result:*** Success

# 10. Graded Assignments API

```
144 def test_fetch_graded_assignments(client: FlaskClient):
145     response = client.get('/study/graded')
146     assert response.status_code == 200
147     assert "graded_assignments" in response.json
```

**Test Case:** Fetch Graded Assignments

- ***API Endpoint:*** /study/graded
- ***Request Method:*** GET
- ***Expected Output:***
  - HTTP Status Code: 200
  - {"graded_assignments": [{"assignment_id": 1,"course_id": 1,"title": "Graded Assignment 1","description": "Description","due_date": "2024-02-01","score": 90,"max_score": 100,"created_at": "2024-01-05"}],"code": 200}
- ***Actual Output:***
  - HTTP Status Code: 200
  - JSON: Matches expected output
- ***Result:*** Success

# 11. Execute Code API

```python
def test_execute_python_code(client: FlaskClient):
    code = {
        "code": "print('Hello, World!')"
    }
    response = client.post('/execute', json=code)
    assert response.status_code == 200
    assert response.json['output'] == "Hello, World!\n"
```

**Test Case:** Execute Python Code

- ***API Endpoint:*** /execute
- ***Request Method:*** POST
- ***Expected Output:***
  - HTTP Status Code: 200
  - JSON: {"output": "Hello, World!\n", "code": 200}
- ***Actual Output:***
  - HTTP Status Code: 200
  - JSON: {"output": "Hello, World!\n", "code": 200}
- ***Result:*** Success

# 12. GeminiLLM Class Initialisation

```python
@pytest.fixture
def gemini_llm():
    return GeminiLLM(model_name='gemini-pro')

def test_initialize_gemini_llm(gemini_llm):
    assert gemini_llm.model_name == "gemini-pro"
    assert gemini_llm.model is not None
```

**Test Case:** Initialise GeminiLLM with Valid Model Name

- ***Function:*** __init__
- ***Expected Outcome:***
  - The GeminiLLM instance is initialized successfully.
  - self.model_name is set to "gemini-pro".
  - self.model is an instance of GenerativeModel.
- ***Result:*** Success

# 13. Generate RAG Prompt

```python
def test_generate_rag_prompt():
    query = "What is the course content for Lecture 1?"
    context = "Lecture 1 covers the basics of Python programming, including variables, data types, and control structures."
    prompt = generate_rag_prompt(query=query, context=context)
    expected_prompt = (
        "You are V.Chatty an AI assistant for Video Lectures. Your role is to provide accurate and helpful information "
        "about the Lectures.Analyze student's queries and retrieve relevant information to the course content and video "
        "in the the database.Provide concise and informative response to the student.Use this information to offer insights "
        "and statistics when answering questions. Please strictly don't give answer to any of the question that asks you to "
        "give a solution to a programming assignment,When responding to queries, prioritize official IIT Madras policies and guidelines.
        "Strictly don't go outside the scope of video "
        "If a query falls outside your knowledge base, politely direct the Student to the appropriate representative."
        "QUESTION: 'What is the course content for Lecture 1?'"
        "CONTEXT: 'Lecture 1 covers the basics of Python programming, including variables, data types, and control structures.'"
        "Response: \n"
    )
    assert prompt == expected_prompt
```

**Test Case:** Generate RAG Prompt with Valid Query and Context

- ***Function:*** generate_rag_prompt
- ***Expected Output:***
    - You are V.Chatty an AI assistant for Video Lectures. Your role is to provide accurate and helpful information about the Lectures. Analyze student's queries and retrieve relevant information to the course content and video in the database. Provide concise and informative response to the student. Use this information to offer insights and statistics when answering questions. Please strictly don't give answer to any of the question that asks you to give a solution to a programming assignment, When responding to queries, prioritize official IIT Madras policies and guidelines. Strictly don't go outside the scope of video If a query falls outside your knowledge base, politely direct the Student to the appropriate representative.

    - QUESTION: 'What is the course content for Lecture 1?'
    - CONTEXT: 'Lecture 1 covers the basics of Python programming, including variables, data types, and control structures.'

    - Response:
- ***Result:*** Success

# 14. Generate Answer

```
39  def test_generate_answer(gemini_llm):
40      prompt = "Explain the basics of Python programming covered in Lecture 1."
41      answer = gemini_llm._call(prompt=prompt)
42      assert isinstance(answer, str)
43      assert len(answer) > 0
```

**Test Case:** Generate Answer with Valid Prompt

- ***Function:*** generate_answer
- ***Expected Outcome:***
  - The function returns a string containing the response generated by the GeminiLLM.
  - Output: A non-empty string response that is relevant to the prompt.
- ***Result:*** Success

# 15. Process Additional Data

```
45  def test_process_additional_data():
46      faiss_vector_store = process_additional_data()    "faiss": Unknown word.
47      assert isinstance(faiss_vector_store, FAISS)    "faiss": Unknown word.
```

**Test Case:** Process PDF, Web, and YouTube Data

- ***Function:*** process_additional_data
- ***Expected Outcome:***
  - PDF content is extracted and stored as a Document object.
  - Web pages are loaded and stored as Document objects.
  - YouTube video information is loaded and stored as Document objects.
  - Texts are split into smaller chunks.
  - FAISS vector store is created with embeddings from all sources.
  - Output: An instance of FAISS vector store containing embeddings from PDF, web, and YouTube documents.
- ***Result:*** Success

# 16. VideoChat Class Initialization

```
49  def test_videochat_initialization():    "videochat": Unknown word.
50      vc = VideoChat()
51      assert vc.faiss_vector_store is not None    "faiss": Unknown word.
52      assert isinstance(vc.gemini_llm, GeminiLLM)
53      assert isinstance(vc.embedding_function, HuggingFaceEmbeddings)
```

**Test Case:** Initialise VideoChat with Processed Data

- ***Function:*** __init__
- ***Expected Outcome:***
  - self.faiss_vector_store is initialized with processed data.
  - self.gemini_llm is an instance of GeminiLLM.
  - self.embedding_function is an instance of HuggingFaceEmbeddings.
- ***Result:*** Success

# 17. Chat Functionality

```
55  def test_chat_functionality():
56      vc = VideoChat()
57      user_input = "Explain the concept of Python lists."
58      answer = vc.chat(user_input=user_input)
59      assert isinstance(answer, str)
60      assert len(answer) > 0
```

**Test Case:** Chat with User Input

- ***Function:*** chat
- ***Expected Outcome:***
  - User input is processed to form a coherent string.
  - Relevant documents are retrieved based on user input.
  - A RAG prompt is generated using the user query and document context.
  - The GeminiLLM generates a response to the prompt.
  - Output: A string containing the response to the user query.
- ***Result:*** Success

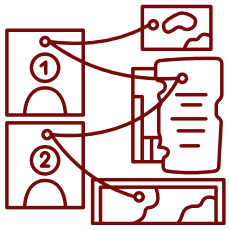
```
PS C:\Users\satis\Desktop\noGenAI\soft-engg-project-may-2024-se-may-27\Code\backend\application> pytest testcases.py
============================================= test session starts =============================================
platform win32 -- Python 3.12.4, pytest-7.4.4, pluggy-1.0.0
rootdir: C:\Users\satis\Desktop\noGenAI\soft-engg-project-may-2024-se-may-27\Code\backend\application
plugins: anyio-4.2.0
collected 16 items

testcases.py ................                                                                    [100%]

============================================== warnings summary ===============================================
testcases.py::test_valid_login
  C:\Users\satis\Desktop\noGenAI\soft-engg-project-may-2024-se-may-27\Code\backend\application\controllers.py:47: DeprecationWarning: datetime.datetime.utcn
ow() is deprecated and scheduled for removal in a future version. Use timezone-aware objects to represent datetimes in UTC: datetime.datetime.now(datetime.U
TC).
    first_login_time=datetime.utcnow()

testcases.py::test_valid_login
  C:\Users\satis\Desktop\noGenAI\soft-engg-project-may-2024-se-may-27\Code\backend\application\token_validation.py:23: DeprecationWarning: datetime.datetime
.utcnow() is deprecated and scheduled for removal in a future version. Use timezone-aware objects to represent datetimes in UTC: datetime.datetime.now(datet
ime.UTC).
    token=jwt.encode({'id':user.id, 'role': user.role, 'exp': datetime.datetime.utcnow() + datetime.timedelta(minutes=100)},app.config['SECRET_KEY'], algori
thm="HS256" )

-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
========================================== 16 passed, 2 warnings in 5.22s ==========================================
PS C:\Users\satis\Desktop\noGenAI\soft-engg-project-may-2024-se-may-27\Code\backend\application>
```



# Conclusion

In conclusion, the testing framework and test cases detailed in this document provide a comprehensive approach to validating the integration of GenAI into the SEEK learning environment. Through rigorous testing, we aim to ensure that the application is reliable, efficient, and capable of delivering a seamless and engaging learning experience. By systematically covering all aspects of the application, from individual components to the overall system, we strive to identify and mitigate potential issues early in the development process. This commitment to quality assurance lays the foundation for successful deployment and future enhancements of the GenAI-enhanced learning platform, ultimately transforming how learners interact with educational content and utilize AI-driven insights for improved learning outcomes.