Introduction to Software Engineering

Software Testing

The student team is required to complete the **Software Testing** documentation for the assigned course project, following the attached template.

Software Engineering Department
Faculty of Information and Technology
University of Science

Table of Contents

Software Testing

Objectives

This document focus on the following topics:

- ✓ Completing the Software Testing document with the following sections:
  - Test Plan
  - Test Cases
- ✓ Understanding the Software Testing document.

1 Member Contribution Assessment

ID	Name	Contribution (%)	Signature
23127069	Nguyen Minh Khoi	100	[illegible]
23127359	Vo Tran Quoc Duy	100	[illegible]
23127367	Ha Thu Hoang	100	[illegible]
23127455	Truong Cong Thien Phu	100	[illegible]
23127511	Vo Minh Tuan	100	[illegible]

2 Test plan

Testing Techniques

To ensure thorough test coverage, the team will apply the following testing techniques:

- **Black-Box Testing**: This will be our primary approach. We will test the functionality of the API endpoints without any knowledge of the internal code structure. Testers will focus exclusively on the inputs and outputs, verifying that the API behaves as described in the documentation. This includes testing valid and invalid data inputs to check for correct responses and error handling.
- **API Testing**: We will use API testing tools to send requests directly to the various endpoints (GET, POST, PUT, DELETE). This technique allows us to validate the core business logic, data processing, and integration between different parts of the system. We will verify HTTP status codes, response bodies, and headers against the expected results defined in our test cases.
- **Role-Based Access Control (RBAC) Testing**: Given the different user roles in the system (Manager, Learning Advisor, Teacher), this technique is critical. We will perform tests to confirm that users can only access the endpoints and perform the actions permitted for their specific role. This involves attempting to access restricted endpoints with unauthorized user tokens to ensure the API correctly returns 403 Forbidden errors.

Test Objects

The testing will be performed on the following functions and documents of the system:

Functions (API Endpoints)

The core functionality is encapsulated in the API endpoints. Each of the following route files represents a testable object:

- auth_route.py (Authentication and Authorization)
- account_route.py (Account Management)

- employee_route.py (Employee Management)
- student_route.py (Student and Enrolment Management)
- course_route.py (Course Management)
- contract_route.py (Contract Management)
- class_route.py (Class Management)
- room_route.py (Room Management)
- checkin_route.py (Staff Check-in/out)
- dashboard_route.py (Statistical Dashboards)

Documents

The following documents are also objects of the testing process, as their accuracy and clarity are essential for quality assurance:

- **API Documentation**: Will be validated to ensure it accurately reflects the behavior of the implemented endpoints.
- **Test Cases**: Will be executed and updated with results.
- **Bug Reports**: Will be created to document any defects found during testing.

Testing Tools

- **Postman**: Will be used as the primary tool for manual API testing. It will allow us to construct and send HTTP requests, manage different testing environments, and inspect responses to validate them against the expected outcomes.
- **Base URL**: http://127.0.0.1:5000/

3 Test cases

3.1 List of test cases

Seq	Test case	Target	Description
1	Authorisation Test Cases	auth.py	Verifies the functionality of user login, logout, and password reset.
2	Account Test Cases	account_route.py	Verifies the CRUD functionality for user accounts.
3	Employee Test Cases	employee_route.py	Verifies the CRUD functionality for employee profiles.
4	Student Test Cases	student_route.py	Verifies the CRUD functionality for student profiles and enrolments.
5	Course Test Cases	course_route.py	Verifies the CRUD functionality for course records.
6	Contract Test Cases	contract_route.py	Verifies the CRUD functionality for student contracts.
7	Class Test Cases	class_route.py	Verifies the CRUD functionality for class records.
8	Check-in/out Test Cases	checkin_route.py	Verifies the functionality for staff checking in and out.

9	Dashboard Test Cases	dashboard_route .py	Verifies the retrieval of statistical data for managers.

3.2 Test case specifications

3.2.1 Test case 1

Test case	AUTH-01-TC-001: Successful Login
Related Use case	User Authentication
Context	A user with valid credentials attempts to log in to the system to receive an access token.
Input Data	**URL**: POST http://127.0.0.1:5000/auth/login **Body**: { "username": "chris.wilson", "password": "password123" }
Expected Output	**Status**: 200 OK **Body**: { "access_token": "<JWT_TOKEN>" }
Test steps	1. Set method to POST. 2. Set URL to {{baseURL}}/auth/login. 3. Set Body to raw JSON with valid credentials. 4. Send request.
Actual Output	Similar to Expected Output
Result	*Passed*

3.2.2 Test case 2

Test case	CON-001-TC-001: Successful Contract Creation
Related Use case	Add a new student contract.
Context	A Learning Advisor, who is authenticated and authorised, attempts to create a new contract for an existing student enrolling in a specific course. All provided data is valid and unique.

Input Data	**URL**: POST http://127.0.0.1:5000/contract/learningadvisor/add
	Headers: Authorization: Bearer {{laToken}}
	Body: json{"student_id": "STU001", "course_id": "C001", "course_date": "2025-07-04", "tuition_fee": 32000000, "start_date": "2025-08-07", "end_date": "2026-02-07"}

Expected Output	**Status**: 201 Created **Body**: The new contract object.

Test steps	1. Set method to POST. 2. Set URL to {{baseURL}}/contract/learningadvisor/add. 3. Set Authorisation header to Bearer {{laToken}}. 4. Set Body to raw JSON with valid, unique data. 5. Send request.
Actual Output	Similar to Expected Output
Result	*Passed*

3.2.3 Test case 3

Test case	EMP-01-TC-001: Successful Employee Creation
Related Use case	Add a new employee profile.
Context	A Manager creates a new employee profile with valid and unique data.

Input Data	**URL**: POST http://127.0.0.1:5000/employee/add
	Headers: Authorization: Bearer {{managerToken}}
	Body: { "full_name": "New Employee", "email": "new.employee@example.com", "role": "Teacher", "teacher_status": "Available" }

Expected Output	**Status**: 201 Created **Body**: The new employee object.
Test steps	1. Set method to POST. 2. Set URL to {{baseURL}}/employee/add. 3. Set Authorisation header to Bearer {{managerToken}}. 4. Set Body to raw JSON with valid data.5. Send request.
Actual Output	Similar to Expected Output
Result	*Passed*

3.2.4 Test case 4

Test case	STU-003-TC-001: Successful Student Creation
Related Use case	Add a new student profile.
Context	A Learning Advisor creates a new student profile with valid and unique data.
Input Data	**URL**: POST http://127.0.0.1:5000/student/learningadvisor/add **Headers**: Authorization: Bearer {{laToken}}

	Body: { "fullname": "New Student", "contact_info": "0901234567 - Mother", "date_of_birth": "2010-05-20" }
Expected Output	**Status**: 201 Created **Body**: The new student object.
Test steps	1. Set method to POST. 2. Set URL to {{baseURL}}/student/learningadvisor/add. 3. Set Authorisation header to Bearer {{laToken}}. 4. Set Body to raw JSON with valid data. 5. Send request.
Actual Output	Similar to Expected Output
Result	*Passed*

3.2.5 Test case 5

Test case	CLS-001-TC-001: Successful Class Creation
Related Use case	Add a new course.
Context	A Learning Advisor creates a new course with valid data.

Input Data	**URL**: POST http://127.0.0.1:5000/course/learningadvisor/add
	Headers: Authorization: Bearer {{laToken}}
	Body: { "name": "IELTS Foundation", "duration": 6, ... }

Expected Output	**Status**: 201 Created**Body**: The new course object.
Test steps	1. Set method to POST. 2. Set URL to {{baseURL}}/course/learningadvisor/add. 3. Set Authorisation header to Bearer {{laToken}}. 4. Set Body to raw JSON with valid data. 5. Send request.
Actual Output	Similar to Expected Output
Result	*Passed*

3.2.6 Test case 6

Test case	ACC-001-TC-001: Successful Account Creation
Related Use case	Add a new account.
Context	A admin attempts to create a new contract for an existing employee. All provided data is valid and unique.

Input Data	**URL**: POST http://127.0.0.1:5000/account/add **Headers**: N/A **Body**: json { "employee_id": "EM001", "username": "HTH", "password": "123456" }
Expected Output	**Status**: 201 Created **Body**: The new account object.
Test steps	1. Set method to POST. 2. Set URL to {{baseURL}}/account /add. 3. Set Body to raw JSON with valid, unique data for a new account linked to an existing employee (e.g., EM001). 4. Send request.
Actual Output	Similar to Expected Output
Result	*Passed*

3.2.7 Test case 7

Test case	CLS-001-TC-001: Successful Class Creation
Related Use case	Add a new class.
Context	A Learning Advisor creates a new class for a course they manage.

Input Data	**URL**: POST http://127.0.0.1:5000/class/learningadvisor/add
	Headers: Authorization: Bearer {{laToken}}
	Body: { "course_id": "C002", "term": 1, ... }

Expected Output	**Status**: 201 Created **Body**: The new class object.
Test steps	1. Set method to POST. 2. Set URL to {{baseURL}}/class/learningadvisor/add. 3. Set Authorisation header to Bearer {{laToken}}. 4. Set Body to raw JSON with valid data.5. Send request.
Actual Output	Similar to Expected Output
Result	*Passed*

3.2.8 Test case 8

Test case	CHK-001-TC-001: Successful Staff Check-in
Related Use case	Staff member checks in for the day.
Context	A Learning Advisor checks in on time. The system should record the check-in and set their status to "Checked In".
Input Data	**URL**: POST http://127.0.0.1:5000/checkin/in **Body**: { "employee_id": "EM005" }
Expected Output	**Status**: 201 Created **Body**: A JSON object with the new check-in record.
Test steps	1. Set method to POST. 2. Set URL to {{baseURL}}/checkin/in. 3. Set Body to raw JSON with a valid employee_id. 4. Send request between 09:00 and 09:15.
Actual Output	Similar to Expected Output
Result	*Passed*

3.2.9 Test case 9

Test case	DSH-001-TC-001: Successful Statistics Retrieval
Related Use case	View statistics.
Context	A Manager requests the overview statistics from the dashboard.
Input Data	**URL**: GET http://127.0.0.1:5000/dashboard/statistics **Headers**: Authorization: Bearer {{managerToken}}
Expected Output	**Status**: 200 OK **Body**: A JSON object containing totals for employees, students, revenue, etc.
Test steps	1. Set method to GET. 2. Set URL to {{baseURL}}/dashboard/statistics. 3. Set Authorisation header to Bearer {{managerToken}}. 4. Send request.
Actual Output	Similar to Expected Output
Result	*Passed*

3.2.10 Test case 10

Test case	STU-005-TC-001: Teacher Gets Students by Class
Related Use case	View students in a class.
Context	A Teacher requests the list of students for a class they are assigned to teach.
Input Data	**URL**: GET http://127.0.0.1:5000/student/teacher/?id=CLS001 **Headers**: Authorization: Bearer {{teacherToken}}
Expected Output	**Status**: 200 OK **Body**: A JSON array of student profiles.
Test steps	1. Set method to GET. 2. Set URL to {{baseURL}}/student/teacher/?id=CLS001. 3. Set Authorisation header to Bearer {{teacherToken}}. 4. Send request.
Actual Output	Similar to Expected Output
Result	*Passed*