ACADEMIA PORTAL – TESTING REPORT

Github Link : https://github.com/AkiBatra25/Academia_Portal

Submitted By: Akshat Batra (IMT2023025), Harshita Bansal(IMT2023035)

1. Introduction

The Academia Portal is a multi-threaded TCP client–server system implemented in C as part of the Operating Systems Lab and Software Engineering coursework. It supports three user roles—Admin, Faculty, and Student—and provides authentication, course registration, user management, and course catalog functionalities.

This Testing Report presents the testing strategy, test cases, and results applied across different levels of the system, including unit testing, module testing, integration testing, system testing, white-box testing, black-box testing, and branch/path coverage assessment.

---

2. Testing Objectives

The main objectives of testing the Academia Portal are:

1. To verify correctness of functional modules (authentication, enrollment, user and course management).
2. To ensure system stability and expected behavior under different workflows.
3. To detect boundary errors, invalid inputs, and incorrect operations.
4. To validate key modules using white-box testing with **branch coverage** and **path coverage**.
5. To confirm that the complete client–server system behaves correctly under real-world scenarios.

---

3. Testing Methods Used

3.1 White-Box Testing

White-box testing was performed on internal C functions, mainly:

- authenticate()
- enroll_course()
- unenroll_course()

White-box techniques used:

- **Branch Coverage**: Ensuring every decision (true/false) is executed.
- **Path Coverage**: Validating major execution paths such as success, failure, and invalid states.

Example:
authenticate() was tested for:

- Valid credentials (success path)
- Wrong password (failure path)
- Wrong role (rejection path)
- Non-existing user (search failure path)

3.2 Black-Box Testing

End-to-end testing was conducted without inspecting internal source code.
This included:

- Testing login flows
- Testing course enrollment and unenrollment
- Testing admin operations
- Testing invalid menu choices
- Testing the behavior of incorrect credentials

Black-box testing validates functionality as experienced by real users.

---

4. Levels of Testing

4.1 Unit Testing

Unit tests were created for the smallest testable components, especially:

Function: authenticate()

Test cases included:

- Valid Admin login
- Invalid password
- Valid Student login
- Correct password but wrong role
- Non-existing username

These tests helped achieve **branch and path coverage** for this function.

4.2 Module Testing

Modules were tested independently:

Admin Module

- Adding users (student/faculty)
- Viewing lists
- Blocking and activating accounts
- Modifying user details

Student Module

- Viewing all courses
- Enrolling in a course
- Unenrolling
- Viewing enrolled courses

Faculty Module

- Adding a new course
- Removing a course
- Updating course details
- Viewing offering courses

Module testing verified that grouped functionalities work correctly before integration.

4.3 Integration Testing

Integration testing checked interactions between modules.

Examples:

- Admin adds a student → that student can immediately log in.
- Faculty adds a course → students can view and enroll in it.
- Enrollment updates both courses.dat seat count and enrollments.dat records.
- Unenrollment correctly removes records and updates seat counts.

This confirms smooth data flow across the system.

4.4 System Testing

System testing validated the **entire client–server system** as one unit.

Scenarios tested:

- Full login → perform operation → logout (all three roles)
- Invalid login attempts return correct error messages
- Invalid menu entries prompt correct error-handling

- Server handles multiple clients (multi-threading)
- Clean exit from the application

This ensures the whole application meets the requirements.

---

5. Test Case Summary

Below is a representative summary of test cases used during testing:

Test Case ID	Description	Input	Expected Output
TC-Auth-01	Valid Admin Login	Username = admin1, Password = admin1pwd	Login Successful
TC-Auth-02	Wrong Password	admin1 / wrongpass	Invalid Credentials
TC-Auth-03	Wrong Role	admin1 logging in as student	Login Rejected
TC-Stu-01	Student Enrollment	Enroll in valid course	Enrollment Successful
TC-Stu-02	Duplicate Enrollment	Enroll again	Error: Already Enrolled
TC-Stu-03	Unenrollment	Unenroll valid course	Unenrollment Successful
TC-Fac-01	Add Course	Faculty enters course name + seat count	Course Added
TC-Admin-01	Add Student	Admin inputs valid username + password	Student Added Successfully

These test cases reflect both functional and robustness tests.

---

6. Branch and Path Coverage

Coverage analysis focused on the most critical functions:

authenticate():

- All conditional branches (role match, password match, username match) executed.
- Paths covered:
    - Successful login
    - Wrong role
    - Wrong password
    - User not found

enroll_course():

- Branches tested:

    - Already enrolled
    - Course full
    - Course does not exist
    - Enrollment success

unenroll_course():

- Branches tested:
    - Student not enrolled
    - Successful unenrollment

Result:
Core functions achieve required **branch coverage** and **path coverage** for SE deliverable.

---

7. Automated Testing

A set of shell scripts simulate real user interaction with the client. These scripts are in the /tests folder:

- test_invalid_login.sh
- test_student_enroll_unenroll.sh
- test_admin_add_student.sh
- test_authenticate (C-unit test)

These scripts automate end-to-end black-box testing for consistency and repeatability.

---

8. Conclusion

Testing validates that:

- All major functionalities work as expected.
- User flows for Admin, Faculty, and Student roles behave correctly.
- Data consistency is maintained across concurrent operations.
- Core logic meets branch and path coverage requirements.
- The system behaves as intended in real usage scenarios.

Thus, the Academia Portal meets the testing criteria for the Software Engineering course.
