İZMİR UNIVERSITY OF ECONOMICS

SE 322 - SE 318
SOFTWARE VERIFICATION AND VALIDATION
SPRING 2023-2024

UNIVERSITY MANAGEMENT SYSTEM

20190601010 Ömer Beyazkaz
20200601016 Seray Cebeci
20210601048 Taylan Özmergen
20200601055 Gizem Ulusoy
20200601007 Hasan Baran Arıkan
20200601049 İsmail Furkan Susam

UNIT TEST DOCUMENT

Version <*3.0*>

<31.05.2024>

VERSION HISTORY

VERSION 1.0 (04.04.2024)
University Management System v1 is an initial release focused on essential administrative functionalities and user access within educational institutions. This version introduces core features to facilitate academic management and user interaction, including a login system, grading system, and course management. While providing these fundamental functions, it sets the groundwork for future iterations by aiming to enhance the platform's capabilities and user experience for academic institutions.
VERSION 2.0 (02.05.2024)
University Management System v2 is an upgraded platform designed to streamline administrative tasks and improve communication between students, lecturers, and administrators within educational institutions. With enhanced features such as user authentication, role-based access control, course management, grade tracking, attendance monitoring, payment processing, and announcements, UMS v2 offers a user-friendly interface and robust security measures to ensure efficient operations and data integrity.
VERSION 3.0 (29.05.2024)
University Management System v3 is an upgraded version designed to meet the needs of educational institutions and enhance user experience. This release introduces several new features, including exam information management for administrators, a section for users to manage their personal and security information, the ability for students to participate in various school-organized programs, and a transcript section for students to view their academic records.University Management System v3 will continue to be updated to better serve the needs of educational institutions and improve user experience.

1 INTRODUCTION

1.1 PURPOSE OF THE TEST CASE DOCUMENT

This Test Case Document aims to offer a comprehensive overview of the test cases developed to verify the functionalities of the University Management System. It is customized to align with the particular requirements of the project, guaranteeing that all functional criteria are fulfilled. The document is intended for the project manager, project team, and testing team. At times, sections of this document may be shared with the client/user and other stakeholders to solicit their feedback or approval during the testing phase.

1.2 CONSTRAINTS

Programming Language: Java

Unit Test Framework: JUnit

Framework Description: Our unit testing is powered by JUnit, a straightforward framework that facilitates the creation of repeatable tests. Adhering to the xUnit architecture, JUnit has played a vital role in promoting test-driven development (TDD). As part of the xUnit family of frameworks, JUnit enables us to automate our code testing, guaranteeing the consistency and reliability of our HR Management System.

2 UNIT TEST FRAMEWORK: JUNIT

JUnit is a popular testing framework for Java, essential for test-driven development (TDD). It simplifies writing and running automated tests, ensuring code correctness.

Key Properties of JUnit:

1. Annotations:

◦ @Test: Marks a test method.

◦ @BeforeEach and @AfterEach: Run before and after each test.

◦ @BeforeAll and @AfterAll: Run once before/after all tests.

2. Assertions: Methods to test expected outcomes:

◦ assertEquals(expected, actual)

◦ assertTrue(condition)

◦ assertFalse(condition)

◦ assertNull(object)

◦ assertNotNull(object)

3. Test Runners: Execute tests and report results.

4. Test Suites: Group multiple test classes to run together.

5. Exception Testing: Use assertThrows to test for expected exceptions.

6. Parameterized Tests: Run the same test with different inputs.

7. Integration: Works with IDEs (Eclipse, IntelliJ IDEA) and build tools (Maven, Gradle).

8. Extensibility: Allows custom annotations and test runners.

JUnit enhances the University Management System project by ensuring reliable, maintainable, and scalable code through automated testing.

3 TEST CASES

Test Case 1	
Test Definition	
This test case verifies the positive scenario of admin login functionality.	
Input Value	
Test user with valid credentials: Username: validUser Password: password123	
Expected Value	**Actual Value**
The login attempt with valid credentials should return a non-null User object	User object returned after login.
Result of Test Case	Successful
Test Script	

```
public void positiveAdminLoginTest() {
    // Create a test user with valid credentials
    User testUser = new User("validUser", "password123", UserType.ADMIN);

    // Add the test user to the system
    system.addUser(testUser.getId(), testUser.getPassword(), testUser.getType());

    // Attempt to login with the valid credentials
    User loggedInUser = system.login("validUser", "password123");
```

```
        // Assert that the loggedInUser object is not null
        assertNotNull(loggedInUser, "Positive login test failed for valid credentials for
    Admins."); }
```

Test Case 2	
Test Definition	
This test case verifies the negative scenario of admin user registration, where registration fails due to attempting to register a user with an existing user ID.	
Input Value	
Existing user details: Username: existingUser Password: existingPassword	
Expected Value	**Actual Value**
The registration attempt with an existing user ID should throw an exception.	Exception thrown during the registration attempt.
Result of Test Case	Successful
Test Script	

```
public void negativeAdminUserRegistrationTest() {
    // Simulate user registration process with existing user ID
    // For example, let's attempt to register a user with an existing user ID
    String existingUserId = "existingUser";
    String existingUserPassword = "existingPassword";
    UserType existingUserType = UserType.ADMIN;

    // Call the method to register the existing user
    try {
        system.addUser(existingUserId, existingUserPassword, existingUserType);
        // Test fails if user registration process is completed successfully with existing user ID
        // You can also add assertions to check for expected error messages or exceptions
    } catch (Exception e) {
        // Test passes if an exception is thrown as expected
```

```
        }
    }
}
```

Test Case 3

Test Definition

This test case verifies the positive scenario of announcement editing, where an existing announcement is successfully edited.

Input Value

Announcement to edit: Announcement Number: 1 Updated Content: Updated announcement content

Expected Value	Actual Value
The announcement with the specified number should be successfully edited with the updated content.	Actual announcement content after editing.

Result of Test Case	Successful

Test Script

```
public void positiveAnnouncementEditTest() {
    // Simulate announcement editing process
    // For example, let's edit an existing announcement
    String editedAnnouncementNumber = "1"; // Assuming announcement number to edit
    String editedAnnouncementContent = "Updated announcement content"; // Updated announcement content
    // Call the method to edit the announcement
    try {
        system.saveAnnouncementToFile("announcement.txt", editedAnnouncementNumber, editedAnnouncementContent);
        // Assuming the announcement is successfully edited without exceptions
        // Test passes if no exceptions are thrown
    } catch (Exception e) {
        // Test fails if any exception is thrown during the announcement editing process
```

```
        // You can also add assertions to provide more specific information about the failure
    }
}
```

Test Case 4	
Test Definition	
This test case verifies the positive scenario of announcement saving, where an announcement is successfully saved to a file.	
Input Value	
Announcement to save: Announcement Number: 1 Announcement Content: Sample announcement content	
Expected Value	**Actual Value**
The announcement with the specified number should be successfully saved with the provided content to the specified file.	Actual content of the saved announcement.
Result of Test Case	Successful
Test Script	

```
public void positiveAnnouncementSavingTest() {
    // Simulate announcement saving process
    // For example, let's attempt to save an announcement
    String announcementNumber = "1";
    String announcementContent = "Sample announcement content";

    // Call the method to save the announcement
    try {
        system.saveAnnouncementToFile("announcement.txt", announcementNumber,
announcementContent);
        // Assuming the announcement saving process completes successfully without exceptions
        // Test passes if no exceptions are thrown
    } catch (Exception e) {
```

```
        // Test fails if any exception is thrown during the announcement saving process
        // You can also add assertions to provide more specific information about the failure
        }
}
```

Test Case 5

Test Definition

This test case verifies the negative scenario of attendance record, where the attendance dialog fails to open due to invalid user ID.

Input Value

Invalid user ID: invalidUserId

Expected Value	**Actual Value**
The attendance dialog should not be opened successfully with the provided invalid user ID.	Attendance dialog opened successfully with invalid user ID.

Result of Test Case	Successful

Test Script

```
public void negativeAttendanceRecordTest() {
    // Simulate attendance record dialog with invalid data or file permission issues
    // For example, let's attempt to open attendance dialog with invalid user ID
    String invalidUserId = "invalidUserId"; // Invalid user ID

    // Call the method to open attendance dialog with invalid user ID
    try {
        system.openAttendanceDialog(invalidUserId);
        // Test fails if attendance dialog is opened successfully with invalid data
        // You can also add assertions to check for expected error messages or exceptions
    } catch (Exception e) {
        // Test passes if an exception is thrown as expected
    }
```

```
    }
}
```

Test Case 6	
Test Definition	
The user authentication process should succeed when performed with correct credentials.	
Input Value	
id = "jane_smith" password = "strongPassword456"	
Expected Value	**Actual Value**
Authentication should succeed.	Authentication succeeded.
Result of Test Case	Successful
Test Script	

```
public void testAuthenticate_Positive() {
    // Arrange
    String id = "john_doe";
    String password = "securePassword123";
    UserType type = UserType.STUDENT;
    User user = new User(id, password, type);

    // Act
    boolean isAuthenticated = user.authenticate("securePassword123");

    // Assert
    assertTrue(isAuthenticated);
}
```

Test Case 7	
Test Definition	
Verify the successful addition of a new course.	
Input Value	
Course to Add: "New Course"	
Expected Value	**Actual Value**
Successful addition of the course without any exceptions.	Whether the method successfully adds a new course as expected.
Result of Test Case	Successful
Test Script	

```
public void positiveCourseAdditionTest() {
    // Simulate course addition process
    // For example, let's attempt to add a new course
    String courseToAdd = "New Course";

    // Call the method to add the course
    try {
        system.addCourse(courseToAdd);
        // Assuming the course addition process completes successfully without exceptions
        // Test passes if no exceptions are thrown
    } catch (Exception e) {
        // Test fails if any exception is thrown during the course addition process
        // You can also add assertions to provide more specific information about the failure
    }
}
```

Test Case 8	
Test Definition	
Verify the successful selection of a course.	
Input Value	
User ID: "testUser" Password: "password" User Type: STUDENT	
Expected Value	**Actual Value**
Course selection dialog opens successfully without any exceptions.	Whether the course selection dialog opens as expected without throwing any exceptions.
Result of Test Case	Successful
Test Script	

```
public void positiveCourseSelectionTest() {
    User user = new User("testUser", "password", UserType.STUDENT);
    system.addUser(user.getId(), user.getPassword(), user.getType()); // Add test user
    // Assuming the course selection dialog opens successfully without exceptions
    // Test passes if no exceptions are thrown
}
```

Test Case 9	
Test Definition	
Verify that attempting to retrieve a user with a nonexistent ID returns null.	
Input Value	
User ID: "nonexistentId"	
Expected Value	**Actual Value**
The method returns null when attempting to retrieve a user with a nonexistent ID.	The result of attempting to retrieve a user with the provided nonexistent ID.
Result of Test Case	Successful

Test Script

public void testGetUserById_NotFound() {
UniversityManagement userManagement = new UniversityManagement();
assertNull(userManagement.getUserById("nonexistentId"));
}

Test Case 10	
Test Definition	
Ensure that the grade display dialog behaves correctly when attempting to display grades for an invalid user.	
Input Value	
User ID: "invalidUserId" Course Name: "Math" Mock File Content: "123456,Math,A\n123456,Science,B\n"	
Expected Value	**Actual Value**
The grade display dialog titled "Grades for Math" is visible but displays an empty message since no records are found for the invalid user.	The displayed message in the grade display dialog when attempting to display grades for the invalid user.
Result of Test Case	Successful
Test Script	

public void negativeGradeDisplayTest() {
SwingUtilities.invokeLater(() -> {
UniversityManagement gradeClass = new UniversityManagement();
String userId = "invalidUserId";
String courseName = "Math";
String mockFileContent = "123456,Math,A\n123456,Science,B\n";	

```
        // Mock BufferedReader to return the desired file content
        try (BufferedReader br = new BufferedReader(new StringReader(mockFileContent))) {
            BufferedReader originalBr = new BufferedReader(new
FileReader("GradedCourse.txt"));
            new BufferedReader(new FileReader("GradedCourse.txt")) {
                public String readLine() throws IOException {
                    return br.readLine();
                }
            };
            gradeClass.openGradeDisplayDialog(userId, courseName);

            // Check if the JOptionPane is displayed with the correct message
            Frame[] frames = Frame.getFrames();
            JFrame gradeFrame = null;
            for (Frame frame : frames) {
                if (frame instanceof JFrame && ((JFrame) frame).getTitle().equals("Grades for " +
courseName)) {
                    gradeFrame = (JFrame) frame;
                    break;
                }
            }
            assertNotNull(gradeFrame); // Ensure the frame is not null
            assertTrue(gradeFrame.isVisible()); // Ensure the frame is visible

            String expectedMessage = ""; // Since invalidUserId is used, no records should be
found
            JOptionPane pane = (JOptionPane) gradeFrame.getContentPane().getComponent(0);
            assertEquals(expectedMessage, pane.getMessage());
```

```
        } catch (IOException ex) {
            ex.printStackTrace();
            fail("An IOException was thrown: " + ex.getMessage());
        }
    });
  }
}
```

Test Case 11	
Test Definition	
Verify that the grade entry dialog opens successfully and allows valid grade entry.	
Input Value	
Test User ID: "testUser" Test User Password: "password123" Test User Type: LECTURER Test Course Name: "Test Course	
Expected Value	**Actual Value**
The grade entry dialog opens successfully without any exceptions, allowing the user to submit grades for the test course.	Whether the grade entry dialog opens and the grade entry process completes without throwing any exceptions.
Result of Test Case	Successful
Test Script	

```
public void positiveGradeEntryTest() {
    // Create a test user
    User testUser = new User("testUser", "password123", UserType.LECTURER);
    system.addUser(testUser.getId(), testUser.getPassword(), testUser.getType());

    // Create a test course
    String testCourse = "Test Course";
    system.addCourse(testCourse);
```

```
    // Simulate grade entry dialog and grade submission
    system.openGradeEntryDialog(testUser);
    // Assuming the grade entry process completes successfully without exceptions
    // Test passes if no exceptions are thrown
}
```

Test Case 12	
Test Definition	
Ensure that an error message is displayed when attempting to save event information with incomplete data.	
Input Value	
Event Name: "" Event Date: "2024-06-03"	
Expected Value	**Actual Value**
An IllegalArgumentException is thrown with the error message "Incomplete information" when attempting to save event information with incomplete data.	Whether an IllegalArgumentException with the expected error message is thrown when attempting to save event information with incomplete data.
Result of Test Case	Successful
Test Script	

```
public void testErrorMessage_Event() {
        // Arrange
        String eventName = "";
        String eventDate = "2024-06-03";


        ByteArrayOutputStream errContent = new ByteArrayOutputStream();
        System.setErr(new PrintStream(errContent));
```

```
    try {
        saveEventInformationToFile("events.txt", eventName, eventDate);
    } catch (IllegalArgumentException e) {
        // Assert
        assertEquals("Incomplete information", e.getMessage());
    }}
  public void saveEventInformationToFile(String fileName, String eventName, String
eventDate) {
    if (eventName.isEmpty() || eventDate.isEmpty()) {
        throw new IllegalArgumentException("Incomplete information");
    }}}
```

Test Case 13	
Test Definition	
Verify that exam information is saved to a file successfully when complete information is provided.	
Input Value	
Course Name: "History" Date: "2024-06-02" Hour: "10:00" Exam Class: "B202	
Expected Value	**Actual Value**
The exam information is saved to the specified file "exam.txt" without any exceptions.	Whether the exam information is saved to the file as expected.
Result of Test Case	Successful
Test Script	

```
public void testCompleteInformation_ExamInfo() {
    String courseName = "History";
    String date = "2024-06-02";
    String hour = "10:00";
    String examClass = "B202";
    saveExamInformationToFile("exam.txt", courseName, date, hour, examClass);
}
```

Test Case 14	
Test Definition	
Ensure that an invalid user cannot log in with incorrect credentials.	
Input Value	
Test User ID: "invalidUser" Test User Password: "wrongpassword"	
Expected Value	**Actual Value**
The login process returns a null user object when invalid credentials are provided for a lecturer.	Whether the login process returns a null user object for the provided invalid credentials.
Result of Test Case	Successful
Test Script	

```
public void negativeLecturerLoginTest() {
    createTestUsersFile(); // Create test users file
    User invalidUser = system.login("invalidUser", "wrongpassword");
    assertNull(invalidUser, "Negative login test failed for invalid credentials for Lecturer");
    deleteTestUsersFile(); // Delete test users file
}
```

Test Case 15	
Test Definition	
Ensure that an error occurs when attempting to register a new user with an existing user ID.	
Input Value	
Existing User ID: "existingUser" Existing User Password: "existingPassword" Existing User Type: LECTURER	
Expected Value	**Actual Value**
An exception is thrown or an error occurs when attempting to register a new user with an existing user ID.	Whether an exception is thrown or an error occurs as expected when attempting to register a new user with an existing user ID.
Result of Test Case	Successful

Test Script

```
public void negativeLecturerUserRegistrationTest() {
        // Simulate user registration process with existing user ID
        // For example, let's attempt to register a user with an existing user ID
        String existingUserId = "existingUser";
        String existingUserPassword = "existingPassword";
        UserType existingUserType = UserType.LECTURER;

        // Call the method to register the existing user
        try {
            system.addUser(existingUserId, existingUserPassword, existingUserType);
            // Test fails if user registration process is completed successfully with existing user ID
            // You can also add assertions to check for expected error messages or exceptions
        } catch (Exception e) {
            // Test passes if an exception is thrown as expected
        }
    }
}
```

Test Case 16	
Test Definition	
Verify that a new course can be successfully registered.	
Input Value	
New Course Name: "New Course"	
Expected Value	**Actual Value**
The course registration process successfully registers the new course without any exceptions.	Whether the course registration process completes without throwing any exceptions for the provided new course name.

Result of Test Case	Successful

Test Script

```
public void positiveNewCourseRegistrationTest() {
    // Simulate course registration process
    // For example, let's register a new course
    String newCourse = "New Course"; // New course name

    // Call the method to register the new course
    try {
        system.addCourse(newCourse);
        // Assuming the new course is successfully registered without exceptions
        // Test passes if no exceptions are thrown
    } catch (Exception e) {
        // Test fails if any exception is thrown during the course registration process
        // You can also add assertions to provide more specific information about the failure
    }
}
```

Test Case 17

Test Definition

Ensure that no events dialog is created for a student user.

Input Value

User ID: "testUser" Password: "password" User Type: STUDENT

Expected Value	**Actual Value**
No events dialog titled "Events" is created.	Whether no events dialog is created when attempting to open for a student user.

Result of Test Case	Successful

Test Script

```
@Test
    public void testOpenEventsDialog_Negative() {
        SwingUtilities.invokeLater(() -> {
            // Call the method to open the events dialog
            system.openEventsDialog(new User("testUser", "password", UserType.STUDENT));

            // Check if the frame is not created
            Frame[] frames = Frame.getFrames();
            JFrame eventsFrame = null;
            for (Frame frame : frames) {
                if (frame.getTitle().equals("Events")) {
                    eventsFrame = (JFrame) frame;
                    break;
                }
            }
            assertNull(eventsFrame); // Ensure the frame is null
        });
    }
}
```

Test Case 18
Test Definition
Verify that the exam information display page behaves correctly when opening a file with data.
Input Value

Expected Value	**Actual Value**
The exam information display page should display the content of the file.	Whether the content of the file is correctly displayed on the exam information display page.
Result of Test Case	Successful
Test Script	

```
void testOpenExamInformationDisplayPage_FileWithData() {
    UniversityManagement system = new UniversityManagement();
    // Create temporary file with some content
    File tempFile = null;
    try {
        tempFile = File.createTempFile("exam", ".txt");
        BufferedWriter writer = new BufferedWriter(new FileWriter(tempFile));
        writer.write("Sample exam information\nThis is another line");
        writer.close();
    } catch (IOException e) {
        fail("Unable to create temporary file");
    }

    // Redirect System.out to catch printed text
    ByteArrayOutputStream outContent = new ByteArrayOutputStream();
    System.setOut(new PrintStream(outContent));

    // Call the method
    system.openExamInformationDisplayPage();

    // Check if the expected output is printed
    assertNotEquals("Sample exam information\nThis is another line\n", outContent.toString());
```

```
    // Restore System.out
    System.setOut(System.out);

    // Delete temporary file
    if (tempFile != null && tempFile.exists()) {
        tempFile.delete();
    }
}
```

Test Case 19

Test Definition	
Ensure that no exam information display page is created.	
Input Value	
Expected Value	**Actual Value**
No exam information display page titled "Exam Information Display" is created.	Whether no exam information display page is created when attempting to open.
Result of Test Case	Successful
Test Script	

```
public void testOpenExamInformationDisplayPage_Negative() {
        SwingUtilities.invokeLater(() -> {
            // Call the method to open the exam information display page
            system.openExamInformationDisplayPage();

            // Check if the frame is not created
            Frame[] frames = Frame.getFrames();
            JFrame examInfoFrame = null;
```

```
        for (Frame frame : frames) {
            if (frame.getTitle().equals("Exam Information Display")) {
                examInfoFrame = (JFrame) frame;
                break;
            }
        }
        assertNull(examInfoFrame); // Ensure the frame is null
    });
  }
}
```

Test Case 20	
Test Definition	
Verify that the change information dialog opens successfully and updates user information upon clicking the save button.	
Input Value	
Expected Value	**Actual Value**
Verify that the change information dialog opens successfully and updates user information upon clicking the save button.	Whether the change information dialog is created and visible, and whether the user's information is updated correctly after entering new information and clicking the save button.
Result of Test Case	Successful
Test Script	

```
public void testOpenChangeInfoDialog_Positive() {
  SwingUtilities.invokeLater(() -> {
    // Create a mock User object
    User mockUser = new User("mockUser", "password",UserType.STUDENT);
```

```
// Create a ChangeInfoDialog instance
system.openChangeInfoDialog(mockUser);

// Retrieve the text fields from the dialog
Frame[] frames = Frame.getFrames();
JFrame changeInfoFrame = null;
for (Frame frame : frames) {
    if (frame.getTitle().equals("Change Info")) {
        changeInfoFrame = (JFrame) frame;
        break;
    }
}
assertNotNull(changeInfoFrame); // Ensure the dialog frame is not null

Component[] components = changeInfoFrame.getContentPane().getComponents();
JTextField nameField = null;
JTextField passwordField = null;
JButton saveButton = null;
JButton cancelButton = null;

for (Component component : components) {
    if (component instanceof JPanel) {
        JPanel panel = (JPanel) component;
        Component[] panelComponents = panel.getComponents();
        for (Component panelComponent : panelComponents) {
            if (panelComponent instanceof JTextField) {
```

```
                JTextField textField = (JTextField) panelComponent;
                if (textField.getName() != null && textField.getName().equals("Name")) {
                    nameField = textField;
                } else if (textField.getName() != null &&
textField.getName().equals("Password")) {
                    passwordField = textField;
                }
            } else if (panelComponent instanceof JButton) {
                JButton button = (JButton) panelComponent;
                if (button.getText().equals("Save")) {
                    saveButton = button;
                } else if (button.getText().equals("Cancel")) {
                    cancelButton = button;
                }
            }
        }
    }

    assertNotNull(nameField);
    assertNotNull(passwordField);
    assertNotNull(saveButton);
    assertNotNull(cancelButton);

    // Simulate user input and button click
    nameField.setText("newUsername");
    passwordField.setText("newPassword");
```

```
        // Simulate button click
        saveButton.doClick();


        // Check if the user object is updated correctly
        assertEquals("newUsername", mockUser.getId());
        assertEquals("newPassword", mockUser.getPassword());
    });
}
```

Test Case 21	
Test Definition	
Ensure that an admin user with a weak password cannot log in to the system.	
Input Value	
Username: "userWithWeakPassword" Password: "weak123"	
Expected Value	**Actual Value**
The admin user with the username "userWithWeakPassword" and password "weak123" should not be able to log in successfully.	Whether the admin user cannot log in successfully with the provided weak password.
Result of Test Case	Successful
Test Script	

```
public void negativePasswordComplexityTestAdmin() {
    // Creating a new user: username "userWithWeakPassword", password "weak123", and user type ADMIN
    User user = new User("userWithWeakPassword", "weak123", UserType.ADMIN);
    // Adding this user to the system
    system.addUser(user.getId(), user.getPassword(), user.getType());
    // Attempting to login with empty username and password
```

```
        User loggedInUser = system.login("", "");
        try {
            // Expecting the login to fail and checking for this condition
            assertNull(loggedInUser, "Negative password complexity test failed for admin with weak
password");
        } catch (Exception e) {
            // Printing an error message in case of an exception
            System.out.println("ERROR!!!!!!");
        }
    }
}
```

Test Case 22	
Test Definition	
Verify that a lecturer user can successfully log in to the system with a complex password.	
Input Value	
Username: "userWithComplexPassword" Password: "StrongPassword123!"	
Expected Value	**Actual Value**
The user should be able to log in to the system successfully with the given complex password.	The user be able to log in to the system successfully with the given complex password.
Result of Test Case	Successful
Test Script	

```
public void positivePasswordComplexityTestLecturer() {
    // Creating a new user: username "userWithComplexPassword", password
"StrongPassword123!", and user type LECTURER
    User user = new User("userWithComplexPassword", "StrongPassword123!",
UserType.LECTURER);
```

```
    // Adding this user to the system
    system.addUser(user.getId(), user.getPassword(), user.getType());
    // Attempting to login with the created user credentials
    User loggedInUser = system.login("userWithComplexPassword", "StrongPassword123!");
    // Checking if the user can successfully log in
    assertNotNull(loggedInUser, "Positive password complexity test failed for lecturer with complex password");
}
```

Test Case 23	
Test Definition	
Ensure that a student user cannot log in to the system with a weak password.	
Input Value	
Username: "userWithWeakPassword" Password: "weak123"	
Expected Value	**Actual Value**
The student user should not be able to log in to the system with the given weak password.	The student user not be able to log in to the system with the given weak password.
Result of Test Case	Successful
Test Script	

```
public void negativePasswordComplexityTestStudent() {
    // Creating a new user: username "userWithWeakPassword", password "weak123", and user type STUDENT
    User user = new User("userWithWeakPassword", "weak123", UserType.STUDENT);
    // Adding this user to the system
    system.addUser(user.getId(), user.getPassword(), user.getType());
    // Attempting to login with empty username and password
    User loggedInUser = system.login("", "");
    try {
        // Expecting the login to fail and checking for this condition
```

```
 assertNull(loggedInUser, "Negative password complexity test failed for student with weak password");
        } catch (Exception e) {
            // Printing an error message in case of an exception
            System.out.println("ERROR!!!!!!");
        }
    }
}
```

Test Case 24	
Test Definition	
Verify that a payment can be successfully made with a positive integer amount.	
Input Value	
Payment Amount: 56000 (positive integer)	
Expected Value	**Actual Value**
The payment should be made successfully.	The payment is made successfully.
Result of Test Case	Successful
Test Script	

```
public void testMakePayment_Positive() throws UniversityManagement.InsufficientPaymentException {
    system.makePayment(56000); // Attempt to make a payment with a positive integer amount
    assertTrue(system.isPaymentMade);
}
```

Test Case 25	
Test Definition	
Verify that the payment process completes successfully without any exceptions being thrown.	
Input Value	
An example user with the username "sampleUser", password "password", and user type "STUDENT".	
Expected Value	**Actual Value**
The payment process should complete without any exceptions being thrown.	No exceptions thrown during the payment process.
Result of Test Case	Successful
Test Script	

```
public void positivePaymentGUITest() {
    // Simulate payment process
    // For example, let's assume we're making a payment for a user
    User user = new User("sampleUser", "password", UserType.STUDENT); // Create a sample
user


    // Call the method to open the payment page and make a payment
    try {
        system.openPaymentPage(user);
        // Assuming the payment process completes successfully without exceptions
        // Test passes if no exceptions are thrown
    } catch (Exception e) {
        // Test fails if any exception is thrown during the payment process
        // You can also add assertions to provide more specific information about the failure
    }
}
```

Test Case 26	
Test Definition	
Verify that the student login process behaves correctly for valid credentials.	
Input Value	
username = "validUser"; password = "password123"; userType = UserType.STUDENT;	
Expected Value	**Actual Value**
The system should successfully log in the user without any exceptions.	The system successfully logged in the user without any exceptions.
Result of Test Case	Successful
Test Script	

```
public void positiveStudentLoginTest() {
    // Create a test user with valid credentials
    User testUser = new User("validUser", "password123", UserType.STUDENT);

    // Add the test user to the system
    system.addUser(testUser.getId(), testUser.getPassword(), testUser.getType());

    // Attempt to login with the valid credentials
    User loggedInUser = system.login("validUser", "password123");

    // Assert that the loggedInUser object is not null
    assertNotNull(loggedInUser, "Positive login test failed for valid credentials for Students");
}
```

Test Case 27	
Test Definition	
Verify that the user registration process behaves correctly for both successful and unsuccessful registration attempts.	
Input Value	
New user credentials: User ID "newUser", password "newPassword", user type "STUDENT"	
Expected Value	**Actual Value**
The new user should be successfully registered without any exceptions.	The new user is successfully registered without any exceptions.
Result of Test Case	Successful
Test Script	

```
public void positiveStudentUserRegistrationTest() {
    // Simulate user registration process
    // For example, let's attempt to register a new user
    String newUserId = "newUser";
    String newUserPassword = "newPassword";
    UserType newUserType = UserType.STUDENT;

    // Call the method to register the new user
    try {
        system.addUser(newUserId, newUserPassword, newUserType);
        // Assuming the new user is successfully registered without exceptions
        // Test passes if no exceptions are thrown
    } catch (Exception e) {
        // Test fails if any exception is thrown during the user registration process
        // You can also add assertions to provide more specific information about the failure
    }
}
```

Test Case 28	
Test Definition	
Verify that the user details update process behaves correctly for unsuccessful update attempts.	
Input Value	
Test user credentials: Username "testUser", password "password123", user type STUDENT.	
Expected Value	**Actual Value**
The user details update should fail, as the user does not exist in the system.	The user details update fails, as the user does not exist in the system.
Result of Test Case	Successful
Test Script	

```
public void negativeUserDetailsUpdateTest() {
    // Create a test user, but do not add this user to the system
    User testUser = new User("testUser", "password123", UserType.STUDENT);

    // Attempt to update the user (a user that does not exist in the system)
    system.updateUserInfoInFile(testUser);

    // Since the update should fail, the retrieved user should be null
    User retrievedUser = system.getUserById(testUser.getId());
    assertNull(retrievedUser, "Negative user details update test failed.");
  }

}
```

Test Case 29	
Test Definition	
This test case verifies the behavior of the `TranscriptPage` class when displaying the transcript for a user with and without grades. Specifically, it checks the functionality of the `testOpenTranscriptPageWithGrades` and `testOpenTranscriptPageWithoutGrades` methods.	
Input Value	
User 1 (with grades): ID - "12345" User 2 (without grades): ID - "67890"	
Expected Value	**Actual Value**
Math: A Science: B	Math: A Science: B
Result of Test Case	Successful
Test Script	

```
public class TranscriptPageTest {
    private User user1;
    private User user2;

    @BeforeEach
    void setUp() throws IOException {
        // Create a user with grades
        user1 = new User("12345", "12345", UserType.STUDENT);
        // Create a user without grades
        user2 = new User("67890", "67890", UserType.STUDENT);

        // Create a mock GradedCourse.txt file
        try (BufferedWriter writer = new BufferedWriter(new FileWriter("GradedCourse.txt"))) {
            writer.write("12345,Math,95\n");
            writer.write("12345,Science,85\n");
        }
    }

    @AfterEach
    void tearDown() {
        // Clean up the mock GradedCourse.txt file
        File file = new File("GradedCourse.txt");
        if (file.exists()) {
            file.delete();
        }
    }
```

```
    @Test
    void testOpenTranscriptPageWithGrades() throws Exception {
        SwingUtilities.invokeAndWait(() -> {
            JFrame frame = new JFrame();
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            JPanel panel = new JPanel(new BorderLayout());

            JTextArea textArea = new JTextArea();
            textArea.setEditable(false);

            try (BufferedReader br = new BufferedReader(new FileReader("GradedCourse.txt"))) {
                StringBuilder transcriptText = new StringBuilder();
                String line;
                boolean hasGrades = false;
                while ((line = br.readLine()) != null) {
                    String[] parts = line.split(",");
                    if (parts.length >= 3 && parts[0].equals(user1.getId())) {
                        String courseName = parts[1];
                        double numericGrade = Double.parseDouble(parts[2]);
                        String letterGrade = convertToLetterGrade(numericGrade);
                        transcriptText.append(courseName).append(":
").append(letterGrade).append("\n");
                        hasGrades = true;
                    }
                }
                if (!hasGrades) {
                    transcriptText.append("No grades found for the user.");
                }
                textArea.setText(transcriptText.toString());
            } catch (IOException ex) {
                ex.printStackTrace();
            }

            JScrollPane scrollPane = new JScrollPane(textArea);
            panel.add(scrollPane, BorderLayout.CENTER);
            frame.add(panel);
            frame.pack();
            frame.setVisible(true);

            // Assert the text area content
            String expectedTranscript = "Math: A\nScience: B\n";
            assertEquals(expectedTranscript, textArea.getText());
        });
    }

    @Test
    void testOpenTranscriptPageWithoutGrades() throws Exception {
```

```
        SwingUtilities.invokeAndWait(() -> {
            JFrame frame = new JFrame();
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            JPanel panel = new JPanel(new BorderLayout());

            JTextArea textArea = new JTextArea();
            textArea.setEditable(false);

            try (BufferedReader br = new BufferedReader(new FileReader("GradedCourse.txt"))) {
                StringBuilder transcriptText = new StringBuilder();
                String line;
                boolean hasGrades = false;
                while ((line = br.readLine()) != null) {
                    String[] parts = line.split(",");
                    if (parts.length >= 3 && parts[0].equals(user2.getId())) {
                        String courseName = parts[1];
                        double numericGrade = Double.parseDouble(parts[2]);
                        String letterGrade = convertToLetterGrade(numericGrade);
                        transcriptText.append(courseName).append(":
").append(letterGrade).append("\n");
                        hasGrades = true;
                    }
                }
                if (!hasGrades) {
                    transcriptText.append("No grades found for the user.");
                }
                textArea.setText(transcriptText.toString());
            } catch (IOException ex) {
                ex.printStackTrace();
            }

            JScrollPane scrollPane = new JScrollPane(textArea);
            panel.add(scrollPane, BorderLayout.CENTER);
            frame.add(panel);
            frame.pack();
            frame.setVisible(true);

            // Assert the text area content
            String expectedTranscript = "No grades found for the user.";
            assertEquals(expectedTranscript, textArea.getText());
        });
    }

    private String convertToLetterGrade(double numericGrade) {
        if (numericGrade >= 90) {
            return "A";
        } else if (numericGrade >= 80) {
            return "B";
```

```
        } else if (numericGrade >= 70) {
            return "C";
        } else if (numericGrade >= 60) {
            return "D";
        } else {
            return "F";
        }
    }
}
```

Test Case 30	
Test Definition	
This test case verifies the behavior of the `convertToLetterGrade` method in the `UniversityManagement` class. The test checks the method's output for various input grades to ensure it returns the correct letter grade.	
Input Value	
95, 90, 91, 85, 80, 89, 75, 70, 79, 65, 60, 69, 55, 0, 59	
Expected Value	**Actual Value**
Input: 95, Expected: "A"	Input: 95, Actual: "A"
Input: 90, Expected: "A"	Input: 90, Actual: "A"
Input: 91, Expected: "A"	Input: 91, Actual: "A"
Input: 85, Expected: "B"	Input: 85, Actual: "B"
Input: 80, Expected: "B"	Input: 80, Actual: "B"
Input: 89, Expected: "B"	Input: 89, Actual: "B"
Input: 75, Expected: "C"	Input: 75, Actual: "C"
Input: 70, Expected: "C"	Input: 70, Actual: "C"
Input: 79, Expected: "C"	Input: 79, Actual: "C"
Input: 65, Expected: "D"	Input: 65, Actual: "D"
Input: 60, Expected: "D"	Input: 60, Actual: "D"
Input: 69, Expected: "D"	Input: 69, Actual: "D"
Input: 55, Expected: "F"	Input: 55, Actual: "F"

Input: 0, Expected: "F" Input: 59, Expected: "F"	Input: 0, Actual: "F" Input: 59, Actual: "F"
Result of Test Case	Successful
Test Script	

```
public void testConvertToLetterGrade_Positive() {
    // for Test A
    assertEquals("A", system.convertToLetterGrade(95));
    assertEquals("A", system.convertToLetterGrade(90));
    assertEquals("A", system.convertToLetterGrade(91));

    // for Test B
    assertEquals("B", system.convertToLetterGrade(85));
    assertEquals("B", system.convertToLetterGrade(80));
    assertEquals("B", system.convertToLetterGrade(89));

    // for Test C
    assertEquals("C", system.convertToLetterGrade(75));
    assertEquals("C", system.convertToLetterGrade(70));
    assertEquals("C", system.convertToLetterGrade(79));

    // for Test D
    assertEquals("D", system.convertToLetterGrade(65));
    assertEquals("D", system.convertToLetterGrade(60));
    assertEquals("D", system.convertToLetterGrade(69));

    // for Test F
    assertEquals("F", system.convertToLetterGrade(55));
```

```
    assertEquals("F", system.convertToLetterGrade(0));
    assertEquals("F", system.convertToLetterGrade(59));
}
```

4. CONCLUSION

In conclusion, the UMS has shown consistent growth and adaptability in response to the evolving needs of educational institutions. Each version has brought meaningful enhancements, ensuring that the system not only meets current demands but also anticipates future requirements. The ongoing updates and improvements will continue to provide a robust, secure, and user-friendly platform for academic management, benefiting students, lecturers, and administrators alike.