Data Base Test Cases:

Test Case ID	**TC_001**
Test Case Name	**DBM_AddUser_WithValidData**
Test Description	Test the Add Valid User Functionality
Test Conditions	All inputs are valid, and there is space in the database to add a new user.
Test Steps	1. Call the DBM_Add_User function with Test1_User data. 2. Validate that the return value is 1 (success). 3. Verify that the database entries (name, age, DOB, gender, educational status, username, and password) match the input values.
Test Inputs	Name: Salma, Age: 25, DOB_day: 5, DOB_Month: 7, DOB_Year: 1999, Educational_Status: Graduate, Gender: Female, UserName: EdgesAcademy, Password: Edges123, Password Recheck: Edges123
Expected Outputs	The DB should be updated with the correct user information, and the return value should be 1.
Actual Outputs	The user information is correctly added to the database. The return value is 1 (success). All fields match the expected values.
Prerequisites	Database has space to add a new user.
Technique Used	Functional Testing
Additional Notes	This test ensures that valid user data is properly added to the database.
Status	Pass

Test Case ID	TC_002
Test Case Name	DBM_AddUser_WithInvalidData
Test Description	Test the Add Invalid User Functionality
Test Conditions	Invalid inputs (negative age, invalid date of birth, empty name or login) should prevent user addition.
Test Steps	1. Call DBM_Add_User with invalid data for each scenario. 2. Validate that the return value is 0 (failure) for each invalid input scenario. 3. Ensure that the database remains unchanged by comparing the user count before and after the tests.
Test Inputs	Name: " ", Age: -5, DOB_day: 32, DOB_Month: 7, DOB_Year: 1999, Educational_Status: Graduate, Gender: Female, UserName: " ", Password: Edges123, Password Recheck: Edges123
Expected Outputs	The DB should reject the user and remain unchanged for each invalid input scenario. The return value should be 0 (failure).
Actual Outputs	The user information is not rejected as expected. The database is modified incorrectly. The return value is not 0 in some cases.
Prerequisites	The database is already initialized with users.
Technique Used	Boundary Value Analysis, Functional Testing
Additional Notes	This test indicates that the DB is not correctly rejecting invalid user data, and the database is being modified despite invalid inputs.
Status	Fail

Test Case ID	TC_003
Test Case Name	DBM_AddUser_WhenDatabaseIsFull
Test Description	Testing the Add User When Database Is Full
Test Conditions	Database is filled to its maximum capacity. All attempts to add a user when the database is full should fail.
Test Steps	1. Fill the database to its maximum capacity by adding users up to MAX_USERS. 2. Validate that the current user count equals MAX_USERS. 3. Attempt to add another user using DBM_Add_User. 4. Verify that the return value is 0, indicating the addition failed. 5. Confirm that the user count remains at MAX_USERS after the failed addition.
Test Inputs	User details similar to "Salma" but with different credentials (e.g., User1, Username1). The second user Test2_User has different credentials from the previously added users.
Expected Outputs	The database should reject the addition of a new user when it is full. The return value of DBM_Add_User should be 0, and the user count should remain at MAX_USERS.
Actual Outputs	The database rejects the new user, and the user count remains at MAX_USERS. The function returns 0 as expected, indicating the user addition failed.
Prerequisites	The database is filled to its maximum capacity with MAX_USERS users.
Technique Used	Boundary Value Analysis, Functional Testing
Additional Notes	This test ensures that the database properly handles attempts to add users when it is full and prevents further additions.
Status	Pass

Test Case ID	**TC_004**
Test Case Name	DBM_AddUser_WithDuplicateUsername
Test Description	Testing the Add User with a duplicate username that already exists in DB
Test Conditions	Database should be initialized with a valid user. The new user must have a username that already exists.
Test Steps	1. Add Test1_User to the database. 2. Attempt to add Test2_User with the same username as Test1_User ("EdgesAcademy"). 3. Validate that the new user is not added. 4. Ensure the user count remains the same. 5. Validate that no duplicate user is in the database.
Test Inputs	Name: Salma, Age: 25, DOB_day: 5, DOB_Month: 7, DOB_Year: 1999, Educational_Status: Graduate, Gender: Female, UserName: EdgesAcademy, Password: Edges123, Password Recheck: Edges123
Expected Outputs	1. The DB should reject adding the new user. 2. The user count should remain unchanged. 3. No duplicate users should exist in the database.
Actual Outputs	1. The DB did not reject the new user. 2. The user count was incorrectly incremented. 3. A duplicate user was found in the database.
Prerequisites	Database is initialized and Test1_User has been added successfully.
Technique Used	Functional Testing
Additional Notes	The test failed because the system did not reject a user with a duplicate username. The database allowed the addition of a duplicate, and the user count was incremented, indicating that the DBM_Add_User function does not properly handle duplicate username validation. This needs to be fixed in the function logic to check for existing usernames before adding a new user.
Status	Fail

Test Case ID	TC_005
Test Case Name	DBM_DeleteUser_WithValidID
Test Description	Testing the Delete Valid User Functionality
Test Conditions	A valid user (e.g., "Salma") is already present in the database.
Test Steps	1. Add a test user (Test1_User) to the database. 2. Capture the test user's ID. 3. Delete the test user using the ID. 4. Validate that the deletion was successful (check return value). 5. Ensure the user count is decremented. 6. Ensure the test user is no longer in the database.
Test Inputs	User ID to delete: ID of Test1_User (e.g., Salma)
Expected Outputs	1. The user should be successfully deleted. 2. The database count should decrease. 3. The test user should no longer exist in the database.
Actual Outputs	1. The user was successfully deleted. 2. The database count decreased. 3. The test user was removed from the database.
Prerequisites	A valid user (e.g., "Salma") is already present in the database.
Technique Used	Functional Testing
Additional Notes	This test successfully verifies that a valid user can be deleted from the database and that the database updates correctly after the deletion.
Status	Pass

Test Case ID	TC_006
Test Case Name	DBM_DeleteUser_WithNonExistentID
Test Description	Testing the Delete Nonexistent User Functionality
Test Conditions	Database contains users
Test Steps	1. Capture the initial user count. 2. Generate a nonexistent user ID (ID greater than the current maximum ID). 3. Attempt to delete the nonexistent user. 4. Validate that the deletion attempt fails. 5. Ensure that the user count remains unchanged.
Test Inputs	Nonexistent User ID (e.g., ID greater than the current maximum ID)
Expected Outputs	1. The deletion attempt should fail. 2. The user count should remain unchanged.
Actual Outputs	1. The deletion attempt failed, but there was an unexpected issue. 2. The user count did not remain unchanged.
Prerequisites	Database contains users
Technique Used	Functional Testing
Additional Notes	The test failed because the deletion did not behave as expected. The user count was not unchanged after attempting to delete a nonexistent user. This could indicate an issue with the user deletion logic when handling non-existent IDs.
Status	Fail

Test Case ID	TC_007
Test Case Name	DBM_DeleteUser_FromMiddleOfDatabase
Test Description	Testing Deletion of a User from the Middle of the Database
Test Conditions	The database is initialized with multiple users
Test Steps	1. Capture the initial user count. 2. Delete the user from the middle (e.g., ID 1). 3. Verify that the user count is decremented by 1. 4. Ensure that the IDs of the remaining users remain consistent (no shifting of IDs).
Test Inputs	User ID to delete (e.g., ID 1)
Expected Outputs	1. The user should be deleted. 2. The IDs of the remaining users should maintain their IDs.
Actual Outputs	1. The user was successfully deleted. 2. The remaining users' IDs were consistent, and no shifting occurred.
Prerequisites	The database is initialized with multiple users
Technique Used	Functional Testing
Additional Notes	This test ensures that deletion from the middle of the database does not cause any unintended ID shifts.
Status	Pass

Test Case ID	**TC_008**
Test Case Name	DBM_DeleteUser_WithDependencies
Test Description	Testing Deletion of a User with Existing Course Enrollments
Test Conditions	Database tracks user enrollments using DBM_AddToCourse()
Test Steps	1. Capture the initial user count. 2. Add a test user to the database. 3. Enroll the user in two courses. 4. Validate that the user is successfully enrolled in both courses. 5. Delete the user from the database. 6. Ensure that the user's enrollments are cleared. 7. Ensure the user count returns to the initial value.
Test Inputs	User ID to delete (e.g., ID of Test1_User) Courses (e.g., courses with IDs 1 and 2)
Expected Outputs	1. The user should be deleted successfully. 2. The user’s enrollments should be cleared. 3. The user count should return to the initial count.
Actual Outputs	1. The user was successfully deleted. 2. The user's enrollments were cleared. 3. The user count returned to the initial count.
Prerequisites	The database tracks user enrollments, and DBM_AddToCourse() is used to enroll users in courses.
Technique Used	Functional Testing
Additional Notes	This test ensures that when a user is deleted, their enrollments in courses are also cleared, maintaining the integrity of the database.
Status	Pass

Test Case ID	TC_009
Test Case Name	DBM_DeleteUser_FromEmptyDatabase
Test Description	Testing Deletion of a User from an Empty Database
Test Conditions	The database is empty.
Test Steps	1. Clear the database by deleting all users. 2. Validate that the database is empty. 3. Attempt to delete a user with ID 0 from the empty database. 4. Validate that the deletion attempt fails. 5. Re-initialize the database and ensure it is empty again.
Test Inputs	User ID to delete -> Any ID (attempting deletion when DB is empty).
Expected Outputs	The deletion attempt should fail, and no changes should be made to the database.
Actual Outputs	The deletion attempt did not fail as expected. The DBM_Delete_User function did not return 0.
Prerequisites	The database is empty before starting the test.
Technique Used	Boundary testing, error condition testing.
Additional Notes	The function DBM_Delete_User should return 0 when attempting to delete from an empty database, but it did not. Further debugging is required to ensure that the function handles the empty database condition correctly.
Status	Fail

Test Case ID	TC_010
Test Case Name	DBM_initDB_WithValidData
Test Description	Testing the DBM_initDB function to ensure the database is properly initialized or reset.
Test Conditions	Database should contain some users before initialization.
Test Inputs	None (test checks the database state after calling DBM_initDB).
Expected Outputs	The database should be reset to its initial state, and the user count should match the expected initial value.
Actual Outputs	The user count after calling DBM_initDB did not match the expected initial value.
Prerequisites	Some users should be added to the database before initialization.
Technique Used	Functional Testing
Additional Notes	The test failed because after calling DBM_initDB, the user count did not reset to the expected initial value.
Status	Fail

Test Case ID	TC_011
Test Case Name	DBM_AddToCourse_WithValidData
Test Description	Tests adding a user to a valid course.
Test Conditions	The user exists in the database, and the course exists.
Test Inputs	CourseID: 1, Test UserID: current_user_test - 1
Expected Outputs	The user should be successfully added to the course, and the enrollment should be reflected in the Enrollments array.
Actual Outputs	The user was successfully added to the course, and the enrollment was correctly reflected.
Prerequisites	The user should be added to the database before testing.
Technique Used	Functional Testing
Additional Notes	The test passed successfully, confirming that the user was correctly enrolled in the course.
Status	Pass

Test Case ID	**TC_012**
Test Case Name	DBM_AddToCourse_NonExistentCourse
Test Description	Tests the behavior of adding a user to a non-existent course.
Test Conditions	The user exists in the database.
Test Inputs	CourseID: 10, Test UserID: current_user_test - 1
Expected Outputs	The enrollment should fail, and the Enrollments table should not reflect the user’s enrollment in the non-existent course.
Actual Outputs	The enrollment failed, and the user's enrollment was not reflected in the Enrollments array for the non-existent course.
Prerequisites	The user should be added to the database before testing.
Technique Used	Functional Testing
Additional Notes	The test passed successfully, confirming that the system does not allow enrollment in a non-existent course.
Status	Pass

Test Case ID	TC_013
Test Case Name	DBM_AddToCourse_AlreadyEnrolled
Test Description	Tests adding a user to a course they are already enrolled in.
Test Conditions	The user is already enrolled in the course.
Test Inputs	CourseID: 1, Test UserID: current_user_test - 1
Expected Outputs	The system should not allow double enrollment, and the Enrollments array should reflect the user's enrollment without change.
Actual Outputs	The test failed because the system allowed the second enrollment attempt, resulting in an unexpected behavior.
Prerequisites	The user should be added and enrolled in the course before testing.
Technique Used	Functional Testing
Additional Notes	The test failed because the system didn't reject the second enrollment attempt as expected.
Status	Fail

Test Case ID	TC_014
Test Case Name	DBM_AddToCourse_FullCourse
Test Description	Tests adding a user to a course that is full.
Test Conditions	The course exists and is full.
Test Inputs	CourseID: 2, Test UserID: current_user_test - 1
Expected Outputs	The system should reject the enrollment and not add the user to the course.
Actual Outputs	The test passed as the course rejected the additional enrollment after reaching the capacity.
Prerequisites	The course must be full before attempting to add another user.
Technique Used	Functional Testing
Additional Notes	The system behaved as expected, rejecting enrollment for a full course and ensuring no overflow.
Status	Pass

Test Case ID	TC_015
Test Case Name	DBM_DeleteReservation_WithValidData
Test Description	Tests deleting a reservation for a valid student in a valid course.
Test Conditions	The student is enrolled in the course.
Test Inputs	CourseID: 1, Test UserID: current_user_test - 1
Expected Outputs	The student’s reservation should be deleted, and the Enrollments array should no longer reflect the student’s enrollment in the course.
Actual Outputs	Deletion failed as the Enrollments array still reflected the student's enrollment.
Prerequisites	The student must be enrolled in the course before deletion.
Technique Used	Functional Testing
Additional Notes	The deletion did not occur as expected; there might be an issue with the DBM_DeleteReservation function.
Status	Fail

Test Case ID	TC_016
Test Case Name	DBM_DeleteReservation_StudentNotEnrolled
Test Description	Tests deleting a reservation for a student not enrolled in the course.
Test Conditions	The student is not enrolled in the course.
Test Steps	1. Add a test user to the database. 2. Assign the test user ID. 3. Attempt to delete a reservation for the test user who is not enrolled. 4. Check the result.
Test Inputs	- CourseID -> 1- Test UserID -> current_user_test - 1 (not enrolled)
Expected Outputs	- The deletion attempt should fail (function should return 0). - The Enrollments table should remain unaffected (the student's enrollment status should stay 0 for the course).
Actual Outputs	- The function returned a non-zero value instead of 0. - The student's enrollment status remained unchanged, causing the test failure.
Prerequisites	The user must not be enrolled in the course.
Technique Used	Manual Testing, Unit Testing
Additional Notes	The function used a comparison (==) instead of an assignment (=) when updating the Enrollments table. This prevented the enrollment status from being correctly updated.
Status	Fail

Test Case ID	**TC_017**
Test Case Name	Delete Reservation for Non-Existent Course
Test Description	Tests the behavior when attempting to delete a reservation for a non-existent course.
Test Conditions	The user exists in the database, but the course does not exist (Course ID = 100).
Test Steps	1. Add a test user to the database. 2. Attempt to delete reservation for a non-existent course.
Test Inputs	CourseID = 100, Test UserID = current_user_test - 1 (not enrolled in the course)
Expected Outputs	The deletion should fail (return 0), and no change should occur in the Enrollments table.
Actual Outputs	The deletion failed as expected (return 0). However, the system did not properly handle the non-existent course, leading to potential issues with course validation.
Prerequisites	The student must exist in the database, and the course ID used must be non-existent.
Technique Used	Functional testing
Additional Notes	The system should handle invalid course IDs and prevent actions on them, such as attempting to delete non-existent course reservations.
Status	Fail

Test Case ID	TC_018
Test Case Name	DBM_DeleteReservation_InvalidStudentID
Test Description	Tests deleting a reservation using an invalid student ID.
Test Conditions	The course exists, but the student ID provided does not exist in the system (invalid user ID = 20).
Test Steps	1. Assign an invalid user ID that is not in the database. 2. Attempt to delete reservation for the non-existent student.
Test Inputs	CourseID = 1, Test UserID = 20 (not in the system)
Expected Outputs	The deletion should fail (return 0), and the Enrollments table should remain unchanged.
Actual Outputs	The deletion failed as expected (return 0). However, the system didn't properly validate the student ID, leading to a failed deletion and unchanged enrollment table.
Prerequisites	The student ID must not exist in the system.
Technique Used	Functional testing
Additional Notes	The function DBM_DeleteReservation does not account for invalid student IDs correctly, leading to unexpected behavior. The system should explicitly handle invalid student IDs by returning 0 when attempting deletion.
Status	Fail

Test Case ID	**TC_019**
Test Case Name	Delete Reservation for Courses with No Enrollments
Test Description	Tests deleting a reservation when the course has no enrollments.
Test Conditions	The course exists but has no students enrolled (Course ID = 2).
Test Steps	1. Add a test user to the database. 2. Attempt to delete reservation for a course with no enrollments.
Test Inputs	CourseID = 2, Test UserID = current_user_test - 1 (not enrolled in the course)
Expected Outputs	The deletion should fail (return 0), and the Enrollments table should remain unchanged.
Actual Outputs	The deletion failed as expected (return 0). However, the system allowed the operation with no enrolled students, returning TRUE instead of 0. Further validation may be needed to ensure proper handling of courses with no enrollments.
Prerequisites	The course must exist, but it should not have any students enrolled.
Technique Used	Functional testing
Additional Notes	The function DBM_DeleteReservation incorrectly handles courses with no enrollments by setting RET = TRUE instead of returning 0 for failure. This causes the test to pass despite expecting failure. The function should return 0 when no students are enrolled.
Status	Fail

Create Account Test Cases:

Test Case ID	TC_020
Test Case Name	Add_Account_WithValidData
Test Description	Testing Create Account with Valid Data Functionality
Test Conditions	There is space in the database to add a new account, and all inputs are valid.
Test Steps	1. Call the Add_Account function with Test1_User data. 2. Validate that the return value is 1 (success). 3. Verify that the database entries (name, age, DOB, gender, educational status, username, and password) match the input values.
Test Inputs	- Name: Salma - Age: 25 - DOB_day: 5 - DOB_Month: 7 - DOB_Year: 1999 - Educational_Status: Graduate - Gender: Female - UserName: EdgesAcademy - Password: Edges123 - Password Recheck: Edges123
Expected Outputs	The DB should be updated with the correct user information, and the return value should be 1.
Actual Outputs	The user information is correctly added to the database. The return value is 1 (success). All fields match the expected values.
Prerequisites	Database has space to add a new user.
Technique Used	MC/DC (Modified Condition/Decision Coverage)
Additional Notes	This test ensures that valid user data is properly added to the database while testing all conditions in the Add_Account function.
Status	Pass

Test Case ID	TC_021
Test Case Name	Add_Account_With_Invalid_DOB_Day
Test Description	Testing the Add Invalid User Functionality when an invalid Date of Birth (31/02/1999) is provided.
Test Conditions	Database is already initialized and ready to accept user data.
Test Steps	1. Capture the initial user count. 2. Create an invalid user (Test1_User) with a date of birth (31/02/1999). 3. Call the Add_Account function with the invalid user data. 4. Validate that the return value is 0 (failure), indicating that the user was rejected. 5. Verify that the user count remains the same as the initial count.
Test Inputs	Name: Salma, Age: 25, DOB_day: 31, DOB_Month: 2, DOB_Year: 1999, Educational_Status: Graduate, Gender: Female, UserName: EdgesAcademy, Password: Edges123, Password Recheck: Edges123
Expected Outputs	The user should be rejected, and the return value should be 0. The database should remain unchanged, and the user count should not change.
Actual Outputs	The user is correctly rejected, and the return value is 0. The user count remains unchanged, as expected.
Prerequisites	Database must be initialized with some users already present.
Technique Used	Boundary Value Analysis (for invalid date validation)
Additional Notes	This test ensures that invalid dates (such as February 31st) are correctly handled by the system and that no invalid users are added to the database.
Status	Pass

Test Case ID	TC_022
Test Case Name	Add_Account_With_Oldest_DOB_Year
Test Description	Testing Add Account with the oldest DOB year (1924)
Test Conditions	All inputs are valid, and there is space in the database to add a new account.
Test Steps	1. Call the Add_Account function with a user whose DOB year is 1924. 2. Validate that the return value is 1 (success). 3. Verify that the user is added to the database and the user count is incremented. 4. Ensure that the database reflects the correct age based on the entered DOB year.
Test Inputs	Name: Salma, Age: 25, DOB_day: 5, DOB_month: 7, DOB_year: 1924, Educational_Status: Graduate, Gender: Female, UserName: EdgesAcademy, Password: Edges123, Password Recheck: Edges123
Expected Outputs	The DB should be updated with the correct user information, the return value should be 1, the user count should increment, and the age should be correctly calculated based on DOB.
Actual Outputs	The user is successfully added to the database, the return value is 1, the user count is incremented, and the age matches the expected value.
Prerequisites	Database has space to add a new user.
Technique Used	Boundary Value Testing
Additional Notes	This test checks the functionality of the system when adding users with the oldest possible valid DOB year.
Status	Pass

Test Case ID	**TC_023**
Test Case Name	Add_Account_With_Min_Name_Length
Test Description	Testing Add Account with the minimum name length (3 characters)
Test Conditions	All inputs are valid, and there is space in the database to add a new account.
Test Steps	1. Call the Add_Account function with a user whose name is "Aya" (minimum length). 2. Validate that the return value is 1 (success). 3. Verify that the user is added to the database and the user count is incremented. 4. Ensure that the database reflects the correct name based on the entered input.
Test Inputs	Name: Aya, Age: 25, DOB_day: 5, DOB_month: 7, DOB_year: 1999, Educational_Status: Graduate, Gender: Female, UserName: EdgesAcademy, Password: Edges123, Password Recheck: Edges123
Expected Outputs	The DB should be updated with the correct user information, the return value should be 1, the user count should increment, and the name should match the input value "Aya".
Actual Outputs	The user is successfully added to the database, the return value is 1, the user count is incremented, and the name matches the input value "Aya".
Prerequisites	Database has space to add a new user.
Technique Used	Boundary Value Testing
Additional Notes	This test checks the functionality of the system when adding users with the minimum valid name length.
Status	Pass

Test Case ID	TC_024
Test Case Name	Add_Account_With_Min_Password_Length
Test Description	Testing Add Account with the minimum password length (8 characters)
Test Conditions	All inputs are valid, and there is space in the database to add a new account.
Test Steps	1. Call the Add_Account function with a user whose password is "Hello123" (minimum length). 2. Validate that the return value is 1 (success). 3. Verify that the user is added to the database and the user count is incremented. 4. Ensure that the database reflects the correct password based on the entered input.
Test Inputs	Name: Salma, Age: 25, DOB_day: 5, DOB_month: 7, DOB_year: 1999, Educational_Status: Graduate, Gender: Female, UserName: EdgesAcademy, Password: Hello123, Password Recheck: Hello123
Expected Outputs	The DB should be updated with the correct user information, the return value should be 1, the user count should increment, and the password should match the input value "Hello123".
Actual Outputs	The user is successfully added to the database, the return value is 1, the user count is incremented, and the password matches the input value "Hello123".
Prerequisites	Database has space to add a new user.
Technique Used	Boundary Value Testing
Additional Notes	This test checks the functionality of the system when adding users with the minimum valid password length.
Status	Pass (Assumed based on the expected output matching actual result)

Test Case ID	TC_025
Test Case Name	Add_Account_PasswordRecheck_Mismatch
Test Description	Testing the account creation functionality when the password and password recheck fields do not match.
Test Conditions	Database is already initialized.
Test Steps	1. Set up a test user with password "PASS1234" and password recheck "pass1234". 2. Call the Add_Account function with the modified user data. 3. Validate that the account creation is rejected (i.e., return value should be 0). 4. Verify that the user count remains unchanged. 5. Clean up the database.
Test Inputs	- Password: "PASS1234"- Password Recheck: "pass1234"
Expected Outputs	- The account should be rejected, and the user count should remain the same.- The return value of Add_Account should be 0.
Actual Outputs	- The account creation was correctly rejected.- The user count remained the same. - The return value was 0 as expected.
Prerequisites	Database is initialized.
Technique Used	Functional Testing, MC/DC
Additional Notes	This test ensures that accounts are not created when the passwords do not match, which is a crucial validation for secure account creation.
Status	Pass

Test Case ID	**TC_026**
Test Case Name	Add_Account_Age_Matches_DOB
Test Description	Testing if the age matches the provided DOB year
Test Conditions	Database is already initialized
Test Steps	1. Set valid age and DOB year where age matches the DOB year. 2. Add user account using the Add_Account function. 3. Check if the user account is successfully added. 4. Validate that the database is updated with the correct user data. 5. Verify that the user count is incremented.
Test Inputs	Test user 1 with Age = 30 and DOB_year = CURRENT_YEAR - 30
Expected Outputs	1. The user account should be added successfully.2. The database should be updated with the correct age.3. The user count should increase by 1.
Actual Outputs	1. The account was added successfully.2. The database was updated correctly with the user's age.3. The user count increased by 1.
Prerequisites	Database is initialized
Technique Used	Boundary Value Analysis, Input Validation
Additional Notes	Test passed successfully
Status	Pass

Test Case ID	TC_027
Test Case Name	Add_Account_Empty_DB
Test Description	Add user to an empty database (initial state)
Test Conditions	Database is empty
Test Steps	1. Clear the database to ensure it is empty. 2. Add the test user account using the Add_Account function. 3. Validate that the account is added successfully. 4. Verify the user count is incremented. 5. Check that the database is updated with the correct user data.
Test Inputs	Test user 1 (valid user data)
Expected Outputs	1. The account should be successfully added.2. The user count should be incremented by 1.3. The database should reflect the correct user data.
Actual Outputs	1. The account was added successfully.2. The user count was correctly incremented.3. The database was updated with the correct user data.
Prerequisites	Database is empty
Technique Used	State Transition Testing
Additional Notes	Test verifies correct behavior when transitioning from an empty database to one with a new user.
Status	Pass

Test Case ID	TC_028
Test Case Name	Add_Account_Same_Username
Test Description	Test the functionality of adding two users with the same username.
Test Conditions	Database is initialized and empty.
Test Steps	1. Add first user with a valid username. 2. Add second user with the same username.
Test Inputs	Test user 1 with username "User1234" and password "Pass1234", Test user 2 with the same username.
Expected Outputs	The second user should not be added to the database. The operation should fail.
Actual Outputs	The second user is added to the database, even though the username is already taken.
Prerequisites	Database initialized and empty.
Technique Used	Boundary Value Analysis, State Transition Testing
Additional Notes	The test failed because the source code does not check for unique usernames.
Status	Fail

Test Case ID	TC_029
Test Case Name	Delete_Account_WithValidData
Test Description	This test ensures that deleting a user with a valid ID successfully removes the user from the database.
Test Conditions	Database is initialized, and at least one valid test user exists.
Test Steps	1. Record the initial database user count. 2. Add a test user. 3. Verify the addition success. 4. Capture the test user ID. 5. Delete the test user. 6. Validate the deletion success. 7. Confirm the database count decrements and the user is removed. 8. Return the database to its initial state.
Test Inputs	Test user ID = current_user_test - 1.
Expected Outputs	The user is successfully deleted, and the database count is decremented.
Actual Outputs	The user was successfully deleted, and the database count was decremented.
Prerequisites	A valid test user (e.g., "Salma") is already present in the database.
Technique Used	Functional Testing.
Additional Notes	Ensures proper functionality of Delete_Account with valid input data.
Status	Pass

Test Case ID	**TC_030**
Test Case Name	Delete_Account_ID_Boundaries
Test Description	This test ensures that deleting users with boundary IDs (e.g., 0 and MAX_USERS) successfully removes the users from the database.
Test Conditions	Database is initialized with at least one valid test user.
Test Steps	1. Record the initial database user count.2. Add a test user.3. Verify the addition success.4. Assign MAX_USERS as the test user ID.5. Delete the user with ID = MAX_USERS.6. Validate the deletion success.7. Validate the database count.8. Attempt to delete the user with ID = 0.9. Validate the deletion success.10. Return the database to its initial state.
Test Inputs	Test user ID = MAX_USERS, Test user ID = 0.
Expected Outputs	Both users are successfully deleted, and the database count decreases accordingly.
Actual Outputs	Both users were successfully deleted, and the database count decreased as expected.
Prerequisites	A valid test user (e.g., "Salma") is already present in the database.
Technique Used	Boundary Value Analysis.
Additional Notes	This test ensures that users with boundary values for IDs can be properly deleted.
Status	**Pass**

Test Case ID	TC_031
Test Case Name	Delete_Account_ID_OutOfBound
Test Description	Testing delete Account Functionality for invalid user IDs
Test Conditions	Database is initialized and a valid test user is added
Test Steps	1. Add Test1_User to the database. 2. Attempt to delete a user with ID = MAX_USERS + 1. 3. Validate that deletion fails and user count remains unchanged. 4. Attempt to delete a user with ID = -1. 5. Validate that deletion fails and user count remains unchanged.
Test Inputs	1. Test1_User with ID = MAX_USERS + 1. 2. Test1_User with ID = -1.
Expected Outputs	1. Deletion should fail for both invalid IDs. 2. User count should not change after failed deletion attempts.
Actual Outputs	Deletion failed for both invalid IDs as expected. User count remained unchanged.
Prerequisites	A valid test user is already present in the database.
Technique Used	Boundary Value Analysis, Negative Testing
Additional Notes	None
Status	Pass

Test Case ID	**TC_032**
Test Case Name	Delete_Account_SameUser
Test Description	Testing delete Account functionality with the same user ID after deletion.
Test Conditions	Database is initialized and contains at least one valid user (e.g., Test1_User).
Test Steps	1. Add Test1_User to the database. 2. Delete the user once. 3. Attempt to delete the same user again.
Test Inputs	Test1_User valid ID (ID after the user has been added to the database).
Expected Outputs	1. The first deletion should succeed (status = 1). 2. The second deletion should fail (status = 0), as the user no longer exists.
Actual Outputs	1. The first deletion succeeded. 2. The second deletion passed, which is incorrect.
Prerequisites	A valid Test1_User exists in the database before the test starts.
Technique Used	Boundary Value Analysis, State Transition
Additional Notes	The function currently allows multiple deletions of the same user, which is a bug.
Status	Fail (Bug found)

Course Registration Test Cases:

Test Case ID	TC_033
Test Case Name	AddStudentToCourse_WithValidData_And_ReEnrollmentCheck
Test Description	Tests registering a valid user to a valid course and handling repeated enrollment attempts.
Test Conditions	A valid test user and course exist.
Test Steps	1. Add a test user to the database.2. Assign ID to the test user.3. Assert the user is NOT already enrolled in the course.4. Register the test user to course 1.5. Assert registration is successful.6. Assert the user is now enrolled in the course.7. Attempt to register the test user to course 1 again.8. Assert the user cannot be re-enrolled.
Test Inputs	CourseID = 1, Test UserID = current_user_test - 1
Expected Outputs	The user should be successfully enrolled in the course, and the enrollment should be reflected in the Enrollments array. The user should not be re-enrolled in the same course.
Actual Outputs	The user was successfully enrolled in the course, the enrollment was recorded, and the user was not re-enrolled.
Prerequisites	User and course should already exist in the database.
Technique Used	Functional Testing, Boundary Value Analysis, MC/DC
Additional Notes	This test verifies both successful enrollment and the handling of repeated enrollment attempts.
Status	Pass

Test Case ID	TC_034
Test Case Name	AddStudentToCourse_FullCourseCapacity
Test Description	Tests registering a user to a course that has reached its maximum capacity.
Test Conditions	A valid test course exists, and the course is at its maximum capacity.
Test Steps	1. Add a test user to the database. 2. Assign an ID to the test user. 3. Enroll users in the course until it is at full capacity. 4. Attempt to enroll the test user into the course. 5. Assert enrollment failure for the test user. 6. Verify the enrollment count does not exceed the maximum capacity.
Test Inputs	CourseID = 2, Test UserID = 6
Expected Outputs	The user should not be enrolled in the course, and the course capacity should remain unchanged.
Actual Outputs	The user was not enrolled in the course, and the course capacity remained unchanged.
Prerequisites	The test course should already exist in the database.
Technique Used	Boundary Value Analysis, Functional Testing
Additional Notes	Verifies correct handling of course capacity limits and ensures no overflow occurs.
Status	Pass

Test Case ID	**TC_035**
Test Case Name	AddStudentToCourse_InvalidCourseID
Test Description	Tests registering a valid user to an invalid course (course ID greater than MAX_COURSES).
Test Conditions	The user exists in the database, and the course doesn't exist (course ID = MAX_COURSES + 1).
Test Steps	1. Add a test user to the database. 2. Assign a valid user ID to the test user. 3. Attempt to register the test user to an invalid course with CourseID = MAX_COURSES + 1.
Test Inputs	CourseID = MAX_COURSES + 1 Test UserID = current_user_test - 1
Expected Outputs	The registration should fail (return 0), and no enrollment should be recorded in the Enrollments array.
Actual Outputs	Registration failed as expected (return 0). However, further validation may be needed to ensure proper course ID validation.
Prerequisites	The test user must exist, and the course ID provided must be invalid (greater than MAX_COURSES).
Technique Used	Boundary Value Analysis (BVA) and Equivalence Partitioning (EP)
Additional Notes	The source code does not handle invalid course IDs properly, leading to a bug where invalid course IDs are not properly validated.
Status	Fail

Test Case ID	**TC_036**
Test Case Name	AddStudentToCourse_InvalidStudentID
Test Description	Tests registering an invalid user to a valid course.
Test Conditions	The user exists in the database, but the student ID is invalid (greater than MAX_USERS). The course exists.
Test Steps	1. Add a valid user to the database. 2. Assign an invalid user ID (MAX_USERS + 1). 3. Attempt to register the user to a valid course (Course ID = 1).
Test Inputs	CourseID = 1, UserID = MAX_USERS + 1
Expected Outputs	The user should not be registered to the course, and the result should indicate failure (e.g., return 0 or appropriate error code).
Actual Outputs	The registration failed as expected, but the result did not properly handle the invalid student ID.
Prerequisites	The student exists in the database, and the course exists.
Technique Used	Boundary Value Analysis, Equivalence Partitioning (EP)
Additional Notes	The source code does not check if the student ID is valid before attempting to register the user, which may cause out-of-bounds access.
Status	Fail

<u>Login Test Cases:</u>

Test Case ID	**TC_037**
Test Case Name	Login_Detect_User_Type_Valid_Admin
Test Description	Tests the detection of user type when the input is for AdminMohamedTarek (0).
Test Conditions	The application must correctly interpret 0 as input for AdminMohamedTarek.
Test Steps	1. Simulate input of "0" using simulate_input. 2. Call Detect_User_Type. 3. Assert the result equals AdminMohamedTarek. 4. Clean up simulated input using cleanup_input.
Test Inputs	Simulated input: 0 (AdminMohamedTarek).
Expected Outputs	The function returns AdminMohamedTarek.
Actual Outputs	The function returned AdminMohamedTarek. Test passed successfully.
Prerequisites	Simulated input functions are correctly implemented.
Technique Used	Equivalence Partitioning
Additional Notes	None
Status	Pass

Test Case ID	TC_038
Test Case Name	Login_Detect_User_Type_Valid_Normal_User
Test Description	Tests the detection of user type when the input is for NormalUser (1).
Test Conditions	The application must correctly interpret 1 as input for NormalUser.
Test Steps	1. Simulate input of "1" using simulate_input. 2. Call Detect_User_Type. 3. Assert the result equals NormalUser. 4. Clean up simulated input using cleanup_input.
Test Inputs	Simulated input: 1 (NormalUser).
Expected Outputs	The function returns NormalUser.
Actual Outputs	The function returned NormalUser. Test passed successfully.
Prerequisites	Simulated input functions are correctly implemented.
Technique Used	Equivalence Partitioning
Additional Notes	None
Status	Pass

Test Case ID	TC_039
Test Case Name	Login_Detect_User_Type_IncorrectLogin
Test Description	Tests the detection of user type when the input is for IncorrectLogin (2).
Test Conditions	The application must correctly interpret 2 as input for IncorrectLogin.
Test Steps	1. Simulate input of "2" using simulate_input. 2. Call Detect_User_Type. 3. Assert the result equals IncorrectLogin. 4. Clean up simulated input using cleanup_input.
Test Inputs	Simulated input: 2 (IncorrectLogin).
Expected Outputs	The function returns IncorrectLogin.
Actual Outputs	The function returned IncorrectLogin. Test passed successfully.
Prerequisites	Simulated input functions are correctly implemented.
Technique Used	Boundary Value Analysis
Additional Notes	None
Status	Pass

Test Case ID	**TC_040**
Test Case Name	Login_Verify_Admin_Correct_Token
Test Description	Tests verifying the admin login with the correct token.
Test Conditions	The admin token should match the predefined value in the source code.
Test Steps	1. Simulate input for the correct admin token (10203040). 2. Call the Verify_Admin() function. 3. Assert that the returned result is TRUE indicating successful login. 4. Clean up simulated input.
Test Inputs	Token: 10203040
Expected Outputs	The function should return TRUE indicating that the admin login is successful.
Actual Outputs	TRUE (Test passed)
Prerequisites	The admin token is predefined as 10203040 in the code.
Technique Used	State Transition Testing
Additional Notes	None
Status	Pass

Test Case ID	TC_041
Test Case Name	Login_Verify_Admin_Incorrect_Token_Two_Attempts
Test Description	Tests verifying the admin login with two incorrect token attempts followed by the correct token.
Test Conditions	The admin token should match the predefined value in the source code. The function should allow up to 3 attempts.
Test Steps	1. Simulate input for incorrect token (12345). 2. Simulate input for a second incorrect token (37829). 3. Simulate input for the correct token (10203040). 4. Call the Verify_Admin() function. 5. Assert that the returned result is TRUE indicating successful login after the third attempt. 6. Clean up simulated input.
Test Inputs	Token: 12345, 37829, 10203040
Expected Outputs	The function should return TRUE indicating that the admin login is successful after the third attempt.
Actual Outputs	TRUE (Test passed)
Prerequisites	The admin token is predefined as 10203040 in the code.
Technique Used	State Transition Testing
Additional Notes	None
Status	Pass

Test Case ID	TC_042
Test Case Name	Login_Verify_Admin_Incorrect_Token_Three_Attempts
Test Description	Tests verifying the admin login when three incorrect token attempts are made.
Test Conditions	The admin token should match the predefined value in the source code. The function should reject login after three failed attempts.
Test Steps	1. Simulate input for the first incorrect token (12345). 2. Simulate input for the second incorrect token (37829). 3. Simulate input for the third incorrect token (0403021). 4. Call the Verify_Admin() function. 5. Assert that the returned result is FALSE indicating the admin login failed after three attempts. 6. Clean up simulated input.
Test Inputs	Token: 12345, 37829, 0403021
Expected Outputs	The function should return FALSE indicating that the admin login failed after three incorrect attempts.
Actual Outputs	FALSE (Test passed)
Prerequisites	The admin token is predefined as 10203040 in the code.
Technique Used	State Transition Testing
Additional Notes	None
Status	Pass

Test Case ID	TC_043
Test Case Name	Login_Verify_User_Valid_Credentials
Test Description	Tests verifying the login of a user with valid credentials.
Test Conditions	A test user must be added to the database with valid username and password. The function should validate the credentials correctly and return a successful login.
Test Steps	1. Add a test user to the database (Test1_User). 2. Validate the addition of the user to the database. 3. Assign the correct ID to the test user. 4. Set the username and password for the login test ("EdgesAcademy", "Edges123"). 5. Call the Verify_User() function. 6. Assert that the result is Login_Successful. 7. Assert that the ID returned in id_ptr matches the expected user ID. 8. Cleanup the database and return it to its initial state.
Test Inputs	Username: "EdgesAcademy", Password: "Edges123"
Expected Outputs	The function should return Login_Successful, and the id_ptr should match the added user's ID.
Actual Outputs	Login_Successful, id_ptr matches added user ID (Test passed)
Prerequisites	The DBM_Add_User function should successfully add a test user, and the database should be in a consistent state before the test.
Technique Used	Boundary Value Analysis (for login credentials validation)
Additional Notes	None
Status	Pass

Test Case ID	**TC_044**
Test Case Name	Login_Verify_User_Invalid_Password
Test Description	Tests verifying the login of a user with an incorrect password.
Test Conditions	A test user must be added to the database with valid username and an incorrect password for the login attempt. The system should return a password incorrect message and set the user ID to -1.
Test Steps	1. Add a test user to the database (Test1_User). 2. Validate the addition of the user to the database. 3. Assign the correct ID to the test user. 4. Set the username and an incorrect password for the login test ("EdgesAcademy", "IncorrectPassword"). 5. Call the Verify_User() function. 6. Assert that the result is Password_incorrect. 7. Assert that the ID returned in id_ptr remains -1. 8. Cleanup the database and return it to its initial state.
Test Inputs	Username: "EdgesAcademy", Password: "IncorrectPassword"
Expected Outputs	The function should return Password_incorrect, and the id_ptr should be -1.
Actual Outputs	Password_incorrect, id_ptr is -1 (Test passed)
Prerequisites	The DBM_Add_User function should successfully add a test user, and the database should be in a consistent state before the test.
Technique Used	Boundary Value Analysis, Equivalence Partitioning
Additional Notes	None
Status	Pass

Test Case ID	**TC_045**
Test Case Name	Login_Verify_User_Nonexistent_Username
Test Description	Tests verifying the login of a user with a nonexistent username.
Test Conditions	A test user must be added to the database, and the system should return a UserName_NotFound error when attempting to log in with a username that does not exist.
Test Steps	1. Add a test user to the database (Test1_User). 2. Validate the addition of the user to the database. 3. Assign the correct ID to the test user. 4. Set a nonexistent username ("NonexistentUsername") and a valid password ("Edges123") for the login test. 5. Call the Verify_User() function. 6. Assert that the result is UserName_NotFound. 7. Assert that the id_ptr is -1.
Test Inputs	Username: "NonexistentUsername", Password: "Edges123"
Expected Outputs	The function should return UserName_NotFound, and the id_ptr should remain -1.
Actual Outputs	The function returned UserName_NotFound, and the id_ptr remained -1.
Prerequisites	A test user (Test1_User) should be added to the database.
Technique Used	Boundary Value Analysis, Equivalence Partitioning
Additional Notes	This test case verifies that when a nonexistent username is used, the correct error message UserName_NotFound is returned, and the ID is not modified.
Status	Pass

Backend Test Cases:

Test Case ID	**TC_046**
Test Case Name	BackEnd_AdminRunner_AddNewUser_ValidData
Test Description	Test the functionality of adding a new user using valid data through the Admin Runner function.
Test Conditions	Admin Runner is accessible and no prior errors exist in the database.
Test Steps	1. Simulate user inputs to add a new user with valid data. 2. Call the Admin_Runner function. 3. Capture printed output. 4. Assert that the message "User Added Successfully" is displayed. 5. Verify user count has increased by 1. 6. Validate the added user details in the database. 7. Clean up inputs and restore the database state.
Test Inputs	User Choice: A Name: Salma, Age: 25, DOB_day: 5, DOB_Month: 7, DOB_Year: 1999, Educational_Status: Graduate, Gender: Female, UserName: EdgesAcademy, Password: Edges123, Password Recheck: Edges123 User Choice: Q
Expected Outputs	User count increases by 1. Database contains new user with the provided details.
Actual Outputs	As expected: Message printed, user count increased by 1, and database updated with new user details.
Prerequisites	Database initialized with DBM_initDB() and contains at least one user.
Technique Used	State Transition and Functional testing
Additional Notes	Validate correct gender and educational status mapping to numeric values. Ensure cleanup resets the database correctly.
Status	Passed

Test Case ID	TC_047
Test Case Name	BackEnd_AdminRunner_AddNewUser_InvalidData
Test Description	Test the functionality of the Admin Runner when attempting to add a new user with invalid data.
Test Conditions	Admin Runner is accessible, and invalid input (mismatched passwords) is provided.
Test Steps	1. Simulate user inputs to add a new user with mismatched passwords. 2. Call the Admin_Runner function. 3. Capture printed output. 4. Assert that the message "Wrong inputs" is displayed. 5. Verify that the user count remains unchanged. 6. Clean up inputs and restore the database state.
Test Inputs	User Choice: A Name: Salma, Age: 25, DOB_day: 5, DOB_Month: 7, DOB_Year: 1999, Educational_Status: Graduate, Gender: Female, UserName: EdgesAcademy, Password: Edges123, Password Recheck: Edges123456 User Choice: Q
Expected Outputs	1. Print the message: "Wrong inputs". 2. User count remains the same. 3. Database is not updated with new user details.
Actual Outputs	As expected: Message printed, user count unchanged, and no new user added to the database.
Prerequisites	Database initialized with DBM_initDB() and contains at least one user.
Technique Used	State Transition and Functional testing
Additional Notes	Validate that incorrect password confirmation is accurately detected and handled. Ensure cleanup restores the database to its original state.
Status	Passed

Test Case ID	TC_048
Test Case Name	BackEnd_AdminRunner_DeleteUser_ValidID
Test Description	Validate the "Delete User" choice (option D) in the Admin Runner function when provided with a valid user ID.
Test Conditions	- A test user must first be added to the database before attempting to delete them. - The ID of the test user must be assigned correctly after addition.
Test Steps	1. Capture the initial user count. 2. Add a valid test user. 3. Simulate inputs for option D with a valid user ID. 4. Call the Admin_Runner function. 5. Validate the database state after the deletion.
Test Inputs	User Choice: D ID: 3 User Choice: Q
Expected Outputs	- The user with ID 3 is deleted from the database. - The user count decreases by 1 after deletion
Actual Outputs	- The user with ID 3 is deleted successfully. - The user count returns to its initial value after deletion.
Prerequisites	The test database must be initialized and available and Test User 1 is added.
Technique Used	State transition testing
Additional Notes	- This test relies on proper functioning of the DBM_Add_User and DBM_Delete_User functions.
Status	Passed

Test Case ID	**TC_049**
Test Case Name	BackEnd_AdminRunner_DeleteUser_InvalidID
Test Description	Validate the "Delete User" choice (option D) in the Admin Runner function when provided with an invalid user ID.
Test Conditions	- The invalid user ID provided is –1. - No changes should occur in the user database.
Test Steps	1. Capture the initial user count. 2. Simulate inputs for option D with an invalid user ID (-1). 3. Call the Admin_Runner function. 4. Validate the database state remains unchanged. 5. Check for the appropriate error message.
Test Inputs	User Choice: D ID: -1 User Choice: Q
Expected Outputs	- No user is deleted. - The user count remains unchanged. - Error message "User ID Doesn't Exist" is printed.
Actual Outputs	- No user was deleted. - The user count remained the same as the initial count .- The error message was displayed.
Prerequisites	- The test database must be initialized and available .- No user should exist with ID -1.
Technique Used	- Boundary testing (invalid ID for deletion).- State transition testing (invalid transition attempt).
Additional Notes	- This test helps ensure robust error handling for invalid inputs in the Admin_Runner function.- Simulated inputs accurately mimic user inputs.
Status	**Passed**

Test Case ID	TC_050
Test Case Name	BackEnd_AdminRunner_CheckReservation
Test Description	Validate the "Check Course Reservation" choice (option C) in the Admin Runner function.
Test Conditions	- The database should be in a valid state. - The DBM_CheckReservations() function is accessible and operational.
Test Steps	1. Simulate inputs for option C to check course reservations. 2. Call the Admin_Runner function. 3. Verify the expected output message is printed.
Test Inputs	User Choice: C (Check Reservation) User Choice: Q (Quit)
Expected Outputs	- The system enters the "Check Course Reservation" flow. - The function DBM_CheckReservations() executes successfully.
Actual Outputs	- The system entered the "Check Course Reservation" flow. - The DBM_CheckReservations() function executed successfully
Prerequisites	- Test database must contain valid data. - DBM_CheckReservations() function should be implemented.
Technique Used	- State transition testing (testing the correct flow when C is selected).
Additional Notes	- Ensures that the reservation checking functionality is triggered correctly from the Admin Runner menu. - Simulated input mimics user actions accurately.
Status	Passed

Test Case ID	TC_051
Test Case Name	BackEnd_AdminRunner_PrintAllUsers
Test Description	Validate the "Print All Users" choice (option P) in the Admin Runner function.
Test Conditions	- The database must contain user data for printing.- The DBM_PrintUsers function must be accessible.
Test Steps	1. Simulate inputs for option P to print all users. 2. Call the Admin_Runner function. 3. Verify the expected output messages are printed.
Test Inputs	User Choice: P (Print All Users) User Choice: Q
Expected Outputs	- The system enters the "Print All Users" flow. - The DBM_PrintUsers function is executed.
Actual Outputs	- The system entered the "Print All Users" flow. - The DBM_PrintUsers function executed successfully
Prerequisites	- Test database must be initialized and populated with valid user data. - DBM_PrintUsers function must correctly format and output user data.
Technique Used	- Functional testing (verifying the expected output for the print operation).
Additional Notes	- This test ensures that all user data is displayed correctly in a structured format.
Status	Passed

Test Case ID	**TC_052**
Test Case Name	BackEnd_AdminRunner_PrintSpecificUser_Exists
Test Description	Validate the "Print a Specific User" choice (option U) in the Admin Runner function when the user exists in the database.
Test Conditions	- The database must contain the user with the specified ID.- The DBM_PrintUserDataAdmin function must be accessible.
Test Steps	1. Add a test user to the database.2. Simulate inputs for option U to print a specific user by their ID.3. Call the Admin_Runner function.4. Verify the expected output messages.
Test Inputs	User Choice: U (Print a Specific User) User ID: 3 User Choice: Q
Expected Outputs	- The system enters the "Print a Specific User" flow. - The DBM_PrintUserDataAdmin function is executed.
Actual Outputs	- The system entered the "Print a Specific User" flow. - The DBM_PrintUserDataAdmin function executed successfully
Prerequisites	- The test database must be initialized. - DBM_Add_User must correctly add the test user, and their ID must be retrievable.
Technique Used	- Functional testing (verifying user-specific print functionality based on the provided ID).
Additional Notes	None
Status	Passed

Test Case ID	TC_053
Test Case Name	BackEnd_AdminRunner_PrintSpecificUser_Nonexistent
Test Description	Validate the "Print a Specific User" choice (option U) in the Admin Runner function when the user does not exist in the database.
Test Conditions	- The database must contain at least one user. - The user ID provided must not exist in the database.
Test Steps	1. Add a test user to the database. 2. Simulate inputs for option U to print a specific user by their non-existent ID. 3. Call the Admin_Runner function. 4. Verify the expected output message indicating the user ID does not exist.
Test Inputs	User Choice: U (Print a Specific User) User ID: 100 (Nonexistent ID) User Choice: Q
Expected Outputs	- The system enters the "Print a Specific User" flow. - The output contains the message: "User ID Doesn't Exist".
Actual Outputs	- The system entered the "Print a Specific User" flow. - The output contained the message: "User ID Doesn't Exist".
Prerequisites	- The test database must be initialized. - DBM_Add_User must correctly add the test user, and their ID must be retrievable.
Technique Used	- Functional testing (verifying user-specific print functionality with invalid user ID).
Additional Notes	- The test user added during the test was assigned a valid ID.- Simulated input replicates administrator interaction.- Proper cleanup ensures database state remains consistent.
Status	Passed

Test Case ID	TC_054
Test Case Name	BackEnd_AdminRunner_Help
Test Description	Test the "Help" option (choice H) in the Admin Runner function.
Test Conditions	- The Admin Runner function must be active. - The user selects the "Help" option.
Test Steps	1. Simulate inputs for option H (Help). 2. Call the Admin_Runner function. 3. Verify the expected help output message is displayed.
Test Inputs	User Choice: H (Help) User Choice: Q (Quit)
Expected Outputs	The Help menu is displayed.
Actual Outputs	- The system displayed the help menu as expected.
Prerequisites	- The Admin Runner function must be running.- The help option should print the correct message.
Technique Used	- Functional testing (verifying the correct help message is displayed).
Additional Notes	- The test validates the help message functionality in the admin interface.- Cleanup ensures no changes to the database.
Status	Passed

Test Case ID	TC_055
Test Case Name	BackEnd_AdminRunner_InvalidChoice
Test Description	Test handling of invalid user choice input (e.g., "Z") in the Admin Runner function.
Test Conditions	- The Admin Runner function must be active. - The user selects an invalid option (e.g., "Z").
Test Steps	1. Simulate inputs for an invalid choice Z. 2. Call the Admin_Runner function. 3. Verify the appropriate error message is displayed.
Test Inputs	User Choice: Z (Invalid choice) User Choice: Q (Quit)
Expected Outputs	The system displays an error message.
Actual Outputs	- The system displayed the error message as expected.
Prerequisites	- The Admin Runner function must be running. - The system should handle invalid user inputs correctly.
Technique Used	- Boundary testing (verifying that the system handles invalid choices correctly).
Additional Notes	This test ensures that the program properly rejects invalid inputs and guides the user back to valid choices.
Status	Passed

Test Case ID	TC_056
Test Case Name	BackEnd_CustomerRunner_CreateNewAccount_ValidData
Test Description	Test handling of valid data input for creating a new account in the Customer Runner function.
Test Conditions	- The Customer Runner function must be active. - The user provides valid inputs for all required fields.
Test Steps	1. Simulate valid inputs for creating a new account. 2. Call the Customer_Runner function. 3. Verify the account creation and user data in the database.
Test Inputs	User Choice: C (Create New Account) Name: Salma Age: 25 DOB: 5/7/1999 Education: Graduate Gender: Female Username: EdgesAcademy Password: Edges123 Password Recheck: Edges123 User Choice: Q (Quit)
Expected Outputs	- The system displays: " User Added Successfully ". - The user count should increase by 1. - The new user should be added to the database with the correct details.
Actual Outputs	- The system displayed the expected success message. - The user count increased by 1. - The new user's data was added to the database correctly.
Prerequisites	- The Customer Runner function must be running. - The system should accept valid inputs for new account creation.
Technique Used	Boundary testing (valid inputs for user creation)
Additional Notes	This test ensures that the new account creation functionality works as expected for valid inputs.
Status	Passed

Test Case ID	TC_057
Test Case Name	BackEnd_CustomerRunner_CreateNewAccount_InvalidData
Test Description	Test handling of invalid data input (Age: 105) for creating a new account in the Customer Runner function.
Test Conditions	- The Customer Runner function must be active. - The user provides invalid input for the age field.
Test Steps	1. Simulate invalid inputs for creating a new account (e.g., invalid age: 105). 2. Call the Customer_Runner function. 3. Verify the account creation is rejected.
Test Inputs	User Choice: C (Create New Account) Name: Salma Age: 105 DOB: 5/7/1999 Education: Graduate Gender: Female Username: EdgesAcademy Password: Edges123 Password Recheck: Edges123 User Choice: Q (Quit)
Expected Outputs	- The system displays: "Wrong inputs". - The user count should remain the same.
Actual Outputs	- The system displayed the expected error message.- The user count remained the same as expected.
Prerequisites	- The Customer Runner function must be running.- The system should reject invalid data (e.g., age above a valid limit).
Technique Used	Boundary testing (invalid age input)
Additional Notes	This test ensures that invalid inputs, such as an out-of-range age, are properly handled by the system.
Status	Passed

Field	Description
Test Case ID	TC_058
Test Case Name	BackEnd_CustomerRunner_Login_Successful
Test Description	Test the successful login flow in the Customer Runner function. The test ensures that a user can log in using valid credentials and then log out successfully.
Test Conditions	- The user must be added to the database. - The user must provide correct credentials for login.
Test Steps	1. Add a valid test user to the database. 2. Simulate user inputs for logging in (username: EdgesAcademy, password: Edges123). 3. Verify user is redirected to the homepage. 4. Simulate logging out and quitting.
Test Inputs	User Choice: L (Login) Username: EdgesAcademy Password: Edges123 User Choice: User Choice: O (Logout) User Choice: Q (Quit)
Expected Outputs	- The system should welcome the user to the homepage after a successful login. - The user should be able to log out successfully.
Actual Outputs	- The system displayed the "Welcome To Your Home Page" message. - User was able to log out successfully.
Prerequisites	- A valid test user (Test1_User) must be added to the database. - The user must provide the correct login credentials.
Technique Used	- Functional testing of login and logout functionality.
Additional Notes	- This test ensures that the login and logout functionality works as expected for a valid user.
Status	Passed

Test Case ID	TC_059
Test Case Name	BackEnd_CustomerRunner_Login_UsernameNotFound
Test Description	Test the scenario where a user attempts to log in with an invalid username. The test ensures that the system handles non-existent usernames correctly by displaying the appropriate error message.
Test Conditions	- The user must provide an incorrect username. - The system should not allow login with an invalid username.
Test Steps	1. Add a valid test user to the database. 2. Simulate user inputs with a wrong username (WrongUsername) and a valid password. 3. Verify that the system displays the correct error message for username not found.
Test Inputs	User Choice: L (Login) Username: WrongUsername Password: Edges123 User Choice: Q (Quit)
Expected Outputs	- The system should display "User Name Doesn't Exist Try Again".
Actual Outputs	- The system displayed the message "User Name Doesn't Exist Try Again".
Prerequisites	- A valid test user (Test1_User) must be added to the database.- The user must provide an incorrect username.
Technique Used	- Functional testing for handling incorrect username.
Additional Notes	- This test ensures that the system properly handles a scenario where the username is not found in the database.
Status	Passed

Test Case ID	TC_060
Test Case Name	BackEnd_CustomerRunner_Login_PasswordIncorrect_FirstTrial
Test Description	Verifies login functionality when the password is incorrect on the first attempt and then correct on the second.
Test Conditions	Test1_User added successfully
Test Steps	1. Add Test1_User 2. Simulate login with wrong password first and correct on second 3. Assert correct output and user redirection
Test Inputs	**Simulated Inputs:** 1. "L" (Login) 2. Username: "EdgesAcademy" 3. Wrong Password on first attempt: "WrongPassFirstTrial" 4. Correct Password on second attempt: "Edges123" 5. User Choice: "O" (Logout) 6. User Choice: "Q" (Quit) 7. After returning to Customer Runner: "Q" (Quit)
Test Expected Output	1. "Password incorrect, Enter Your Password again. 2 Trials Left" 2. "Welcome To Your Home Page"
Actual Outputs	As expected
Prerequisites	Test1_User added to the database
Technique Used	State Transition Technique
Status	Passed

Test Case ID	TC_061
Test Case Name	BackEnd_CustomerRunner_Login_PasswordIncorrect_SecondTrial
Test Description	Verifies login functionality when the password is incorrect on both the first and second attempts and correct on the third.
Test Conditions	Test1_User added successfully
Test Steps	1. Add Test1_User 2. Simulate login with wrong passwords on first two attempts and correct on third 3. Assert correct output and user redirection
Test Inputs	**Simulated Inputs:** 1. "L" (Login) 2. Username: "EdgesAcademy" 3. Wrong Password on first attempt: "WrongPassFirstTrial" 4. Wrong Password on second attempt: "WrongPassSecondTrial" 5. Correct Password on third attempt: "Edges123" 6. User Choice: "O" (Logout) 7. User Choice: "Q" (Quit) 8. After returning to Customer Runner: "Q" (Quit)
Test Expected Output	1. "Password incorrect, Enter Your Password again. 2 Trials Left" 2. "Password incorrect, Enter Your Password again. 1 Trial Left" 3. "Welcome To Your Home Page"
Actual Outputs	As expected
Prerequisites	Test1_User added to the database
Technique Used	State Transition Technique
Status	Passed

Test ID	TC_062
Test Case Name	BackEnd_CustomerRunner_Login_PasswordIncorrect_ThirdTrial
Test Description	This test verifies that the system correctly handles the third incorrect password attempt.
Test Conditions	The database contains a valid test user. The user will attempt to log in with incorrect passwords for three trials, and the third trial should trigger user deletion.
Test Steps	1. Add a valid test user to the database. 2. Attempt to log in with incorrect passwords for three consecutive trials. 3. Assert that the system displays "Your session is Terminated" and indicates the user will be deleted. 4. Verify that the user count in the database is decremented.
Test Inputs	- Username: EdgesAcademy - Password (Trial 1): WrongPassFirstTrial - Password (Trial 2): WrongPassSecondTrial - Password (Trial 3): WrongPassThirdTrial - User Choice: Q (Quit)
Expected Output	- After three incorrect password attempts: - "Your session is Terminated" - "User Will Be Deleted, Contact Admin to return it" - The user count should be decremented by 1 in the database.
Actual Output	- "Your session is Terminated" - "User Will Be Deleted, Contact Admin to return it" - The user count is **not** decremented due to a bug in the DBM_Delete_User function.
Prerequisites	A test user must be added to the database before the test begins.
Technique Used	State Transition Testing
Additional Notes	The test case highlights a bug in the DBM_Delete_User function, where the user is not deleted from the database, which causes the test to fail. This issue should be addressed to ensure the user deletion logic works correctly after three failed login attempts.
Status	Failed

Test Case ID	**TC_063**
Test Case Name	BackEnd_HomePageRunner_ChangePassword_Successfuly
Test Description	This test case validates the functionality of changing the password in the home page runner after a successful login.
Test Conditions	The user must be logged in with valid credentials, and the password must be changed through the Home Page Runner.
Test Steps	1. Add a test user to the database. 2. Log in with the correct username and password. 3. Navigate to the "Change Password" option. 4. Provide current password, a new password, and confirm the new password. 5. Log out and quit. 6. Validate that the password change was successful.
Test Inputs	Login: Username: EdgesAcademy, Password: Edges123 Change Password: Current Password: Edges123, New Password: NewPassword, Confirm Password: NewPassword Logout and Quit
Expected Output	"Password Changed Successfully" message should appear.
Actual Output	Password Changed Successfully
Prerequisites	Test user must be added to the database before starting the test.
Technique Used	State Transition
Additional Notes	None
Status	Passed

Test Case ID	TC_064
Test Case Name	BackEnd_HomePageRunner_ChangePassword_Mismatch
Test Description	This test verifies that the user cannot change their password if the new password and the confirmation password do not match.
Test Conditions	Database contains a valid user with username "EdgesAcademy" and password "Edges123".
Test Steps	1. Add the test user to the database. 2. Simulate user inputs for login and password change (with a mismatch between the new password and the confirmation). 3. Assert the expected error message is printed. 4. Return database to its initial state.
Test Inputs	User Choice: L (Login) Username: EdgesAcademy Password: Edges123 User Choice: C (Change Password) Current Password: Edges123 New Password: NewPassword Confirm Password: NewPasswordMismatch User Choice: User Choice: O (Logout) User Choice: Q (Quit)
Expected Output	"New Password mismatched"
Actual Output	"New Password mismatched"
Prerequisites	A valid user ("EdgesAcademy") is present in the database with the password "Edges123".
Technique Used	State Transition Testing
Additional Notes	None
Status	Passed

Test Case ID	TC_065
Test Case Name	BackEnd_HomePageRunner_ChangePassword_Incorrect
Test Description	This test case checks if the system handles the scenario where the user attempts to change their password but enters an incorrect current password.
Test Conditions	A valid user ("EdgesAcademy") is present in the database with the password "Edges123".
Test Steps	1. Add the test user to the database. 2. Simulate user inputs for login and password change (with an incorrect current password). 3. Assert the expected error message "Current Password is incorrect" is printed. 4. Return database to its initial state.
Test Inputs	User Choice: L (Login) Username: EdgesAcademy Password: Edges123 User Choice: C (Change Password) Current Password: WrongPassword New Password: NewPassword Confirm Password: NewPassword User Choice: O (Logout) User Choice: Q (Quit)
Expected Output	"Current Password is incorrect"
Actual Output	"Current Password is incorrect"
Prerequisites	A valid user ("EdgesAcademy") is present in the database with the password "Edges123".
Technique Used	State Transition Testing
Additional Notes	None
Status	Passed

Test Case ID	TC_066
Test Case Name	BackEnd_HomePageRunner_ReserveCourse_NonexistentCourseID
Test Description	This test case checks if the system correctly handles an attempt to reserve a course with a nonexistent course ID.
Test Conditions	A valid user ("EdgesAcademy") is present in the database with the password "Edges123".
Test Steps	1. Add the test user to the database. 2. Simulate user inputs for login and course reservation (with an invalid course ID). 3. Assert the expected error message "Out of range Course Try again" is printed. 4. Return database to its initial state.
Test Inputs	User Choice: L (Login) Username: EdgesAcademy Password: Edges123 User Choice: R (Reserve a course) Course ID: -1 (Incorrect Course ID) User Choice: O (Logout) User Choice: Q (Quit)
Expected Output	"Out of range Course Try again"
Actual Output	"Out of range Course Try again"
Prerequisites	A valid user ("EdgesAcademy") is present in the database with the password "Edges123".
Technique Used	State Transition Testing
Additional Notes	None
Status	Passed

Test Case ID	TC_067
Test Case Name	BackEnd_HomePageRunner_ReserveCourse_SuccessfulEnrollment
Test Description	This test verifies that a user can successfully enroll in a course by providing valid credentials and a valid course ID.
Test Conditions	A user must be added to the database and be able to log in. The course ID must be valid and available for enrollment.
Test Steps	1. Add a user to the database. 2. Log in using valid credentials. 3. Select the option to reserve a course. 4. Enter a valid course ID. 5. Logout and quit.
Test Inputs	User Choice: L (Login) Username: EdgesAcademy Password: Edges123 User Choice: R (Reserve a course) Course ID: 1 (Valid Course ID) User Choice: O (Logout) User Choice: Q (Quit)
Expected Output	The user should be successfully enrolled in the course, with a message confirming the reservation.
Actual Output	"You Are Added To This Course Reservation" message.
Prerequisites	The user must be created and have valid credentials. The course must exist in the database with a valid ID.
Technique Used	State Transition
Additional Notes	None
Status	Passed

Test Case ID	TC_068
Test Case Name	BackEnd_HomePageRunner_ReserveCourse_AlreadyEnrolled
Test Description	This test verifies that a user cannot enroll in a course they are already enrolled in, ensuring proper system validation.
Test Conditions	A user must be added to the database and successfully enrolled in a course. The course ID must be valid.
Test Steps	1. Add a user to the database. 2. Log in using valid credentials. 3. Select the option to reserve a course. 4. Enter a valid course ID that the user is already enrolled in. 5. Attempt to reserve the same course again. 6. Logout and quit.
Test Inputs	User Choice: L (Login) Username: EdgesAcademy Password: Edges123 User Choice: R (Reserve a course) Course ID: 2 (Already Enrolled) User Choice: O (Logout) User Choice: Q (Quit) User Choice: L (Login) Username: EdgesAcademy Password: Edges123 User Choice: R (Reserve a course) Course ID: 2 (Already Enrolled) User Choice: O (Logout) User Choice: Q (Quit)
Test Expected Output	The user should be informed that they are already enrolled in the course after attempting to reserve it a second time.
Actual Output	The user is successfully informed that they are already enrolled in the course.
Prerequisites	Test user must be added and enrolled in a course successfully.
Technique Used	Functional testing
Additional Notes	The test ensures the application properly handles cases where a user tries to re-enroll in the same course.
Status	Passed

Test Case ID	TC_069
Test Case Name	BackEnd_HomePageRunner_SeeYourInfo
Test Description	The test ensures that when a user logs in and selects the 'i' option to view their information, the system correctly displays their user data and then logs them out as expected.
Test Conditions	- Test environment is set up with a database capable of adding users. - The test user Test1_User is added to the database. - The system must simulate login and selection of options in the menu.
Test Steps	1. Add a test user (Test1_User) to the database. 2. Simulate user input for logging in and selecting 'i' to see user info. 3. Ensure that user data is displayed. 4. Log out after viewing the information.
Test Inputs	User Choice: L (Login) Username: EdgesAcademy Password: Edges123 User Choice: i (see your info) User Choice: O (Logout) User Choice: Q (Quit)
Test Expected Output	- The system should display the user's data when the 'i' option is selected.- The system should log out and display the logout message after selecting the 'O' option.
Actual Output	The output contains the expected messages: 'The Data Of User' and 'Good Bye!! see you soon'.
Prerequisites	A valid test user (Test1_User) must be added to the database.
Technique Used	State Transition
Additional Notes	- Ensure that the logout process and user info display functions as expected.- The test includes the simulated inputs for the login and option selections.
Status	Passed

Test Case ID	**TC_070**
Test Case Name	BackEnd_HomePageRunner_ShowEnrolledCourses
Test Description	This test case verifies the functionality of the 'Show Enrolled Courses' option in the HomePage Runner.
Test Conditions	The test user (Test1_User) must be added to the database. The system should be able to simulate user inputs and handle the display of enrolled courses correctly.
Test Steps	1. Add Test1_User to the database. 2. Simulate user input for login. 3. Simulate user input to reserve a course (Course ID: 1). 4. Simulate user input to show enrolled courses. 5. Simulate logout and quit options.
Test Inputs	User Choice: L (Login) Username: EdgesAcademy Password: Edges123 User Choice: R (Reserve a course) Course ID: 1 (Correct Course ID) User Choice: S (Show Enrolled Courses) User Choice: O (Logout) User Choice: Q (Quit)
Test Expected Output	The system should display the enrolled courses with the message "You Are Currently Enrolled in:", followed by the course details.
Actual Output	The system displays the list of enrolled courses with the expected message.
Prerequisites	The test user (Test1_User) must be available in the database.
Technique Used	State Transition
Additional Notes	None
Status	Passed

App Test Cases:

Test Case ID	TC_071
Test Case Name	BackEnd_MainAppRunner_AdminVerified
Test Description	This test case verifies the behavior of the Main App Runner when an Admin is successfully verified.
Test Conditions	The system must be able to verify an Admin using a valid token and direct the admin to the Admin Runner page after verification.
Test Steps	1. Simulate user input for Admin login with token 10203040. 2. Simulate selection of the 'Q' (Quit) option. 3. Verify that the expected admin-related messages are displayed. 4. Ensure that the Admin Runner page is entered successfully.
Test Inputs	User Type: 0 (Admin) Token: 10203040 User Choice: Q (Quit)
Test Expected Output	The system should display a series of messages confirming the Admin's identity, including: 1. "Hello Are You Admin Mohamed Tarek?" 2. "Please Enter Your Secret Token" 3. "Welcome Mohamed Tarek" 4. "Welcome To Edges Software".
Actual Output	The system displays the expected admin-related messages.
Prerequisites	The Admin, "Mohamed Tarek", must be present and available to be verified.
Technique Used	State Transition
Additional Notes	None.
Status	Passed

Test Case ID	**TC_072**
Test Case Name	BackEnd_MainAppRunner_AdminNotVerified
Test Description	This test verifies the behavior of the Main App Runner when an Admin user attempts to log in with an incorrect token.
Test Conditions	The system must be configured with an Admin user (Mohamed Tarek) and invalid tokens for Admin login.
Test Steps	1. Simulate input to log in as Admin with three incorrect tokens. 2. Redirect to Admin Runner page. 3. Assert that the system displays the "Wrong Token" message. 4. Assert that the system displays the "You Are a Thief" message and closes the software.
Test Inputs	User Type: 0 (Admin) Tokens: 1234567, 2397615, 0971020
Test Expected Output	The system should display: 1. "Hello Are You Admin Mohamed Tarek?" 2. "Please Enter Your Secret Token" 3. "Wrong Token please Enter Again you have 1 Trial left" 4. "You Are a Thief and not Mohamed Tarek\nSoftware Will Close"
Actual Output	"Hello Are You Admin Mohamed Tarek?" "Please Enter Your Secret Token" "Wrong Token please Enter Again you have 1 Trial left" "You Are a Thief and not Mohamed Tarek\nSoftware Will Close"
Prerequisites	Admin user (Mohamed Tarek) must exist, and invalid tokens for Admin login must be configured.
Technique Used	State Transition
Additional Notes	Ensure that the Admin user and invalid tokens are configured correctly in the system before running the test.
Status	Passed

Test Case ID	**TC_073**
Test Case Name	BackEnd_MainAppRunner_NormalUser
Test Description	This test verifies the behavior of the Main App Runner when a Normal User logs in. The test simulates a Normal User selecting 1 for Normal User and ensures that the system redirects the user to the Customer Runner page with the correct message. The test also ensures that after the user is redirected, the system displays the expected "Welcome To Edges Software" message before the user quits the application.
Test Conditions	The system must be configured with a Normal User option.
Test Steps	1. Simulate input to log in as a Normal User. 2. Redirect to Customer Runner page. 3. Assert that the system displays the "Welcome To Edges Software" message.
Test Inputs	User Type: 1 (Normal User)
Test Expected Output	The system should display: 1. "Hello Are You Admin Mohamed Tarek?" 2. "if yes please Enter 0" 3. "if No Please Enter 1" 4. "Other Input will close the SW" 5. "Welcome To Edges Software"
Actual Output	"Hello Are You Admin Mohamed Tarek?" "if yes please Enter 0" "if No Please Enter 1" "Other Input will close the SW" "Welcome To Edges Software"
Prerequisites	The system must be configured with a Normal User option.
Technique Used	Automated input simulation and output validation using simulate_input() and assert_printf_output_contains().
Additional Notes	State Transition
Status	Passed

Test Case ID	**TC_074**
Test Case Name	BackEnd_MainAppRunner_WrongEntry
Test Description	This test verifies the behavior of the Main App Runner when the user inputs an invalid entry. The test simulates a wrong user type (2) and ensures that the system detects the wrong entry and displays an appropriate error message. The test also verifies that the system terminates after the wrong entry with the message "Wrong Entry you will quit".
Test Conditions	The system must handle invalid user entries and provide an appropriate response.
Test Steps	1. Simulate input with an invalid user type. 2. Assert that the system displays the error message for a wrong entry.
Test Inputs	User Type: 2 (Invalid Entry)
Test Expected Output	The system should display: 1. "Hello Are You Admin Mohamed Tarek?" 2. "if yes please Enter 0" 3. "if No Please Enter 1" 4. "Other Input will close the SW" 5. "Wrong Entry you will quit"
Actual Output	"Hello Are You Admin Mohamed Tarek?" "if yes please Enter 0" "if No Please Enter 1" "Other Input will close the SW" "Wrong Entry you will quit"
Prerequisites	The system must handle invalid user entries and provide an appropriate response.
Technique Used	State Transition
Additional Notes	Ensure that the system responds correctly to invalid entries.
Status	Passed