EDGES
OUR EDGE IS YOUR SUCCESS

Report Test Cases For Application "EDGES Reservation software"

Created By	Bassam Ashraf
Reviewed By	

Test Function (Add Account):

Test ID	Test Case	Technique	Status
1	Add Account Functionality	Equivalence Partitioning	Passed

- **Description of Test Case:**
  Testing the Add Account Functionality with valid inputs.
- **Pre-requisites:**
  There is a User Assigned to DB.
- **Test Inputs:**
  - **Name:** Ahmed
  - **Age:** 26
  - **DOB_day:** 01
  - **DOB_Month:** 02
  - **DOB_Year:** 1999
  - **Educational_Status:** Masters_Student
  - **Gender:** Male
  - **UserName:** EdgesAcademy
  - **Password:** Edges123
  - **Password Recheck:** Edges123
- **Expected Results:**
  The DB should be updated with the previous inputs correctly.
- **Actual Results:**
  The DB was updated with the correct inputs.
- **Reason:**
  All inputs are in the correct format.
- **Bug:**
  None

Test ID	Test Case	Technique	Status
2	Add Account with (Invaild name)	Boundary Value Analysis	Passed

- **Description of Test Case:**
  Testing the Add Account Functionality by adding an invalid name.
- **Pre-requisites:**
  There is a User Assigned to DB.
- **Test Inputs:**
  - **Name:** “Al” (Invalid, 2 chars)
  - **Name:** “ThisIs33CharactersLong12345678901” (Invalid, 33 chars)
  - **Name:** “” (Leave it Empty)
- **Expected Results:**
  The function should reject the input and not add the user to the DB.
- **Actual Results:**
  The function rejected the input and did not add the user.
- **Reason:**
  The name length is below the minimum valid length (3-32 characters).
- **Bug:**
  None

Test ID	Test Case	Technique	Status
3	Add Account with (vaild name)	Boundary Value Analysis	Failed

- **Description of Test Case:**
  Testing the Add Account Functionality by adding a valid name.
- **Pre-requisites:**
  There is a User Assigned to DB.
- **Test Inputs:**
  1. First Input:
     - **Name:** “AhmedFIRST “   (Valid, 10 chars)
  2. Second Input:
     - **Name:** “Ali”   (Valid, 3 chars)
  3. Third input:
     - **Name:** “ThisIsExactly32CharactersLong123” (Valid, 32 chars) Failed
- **Expected Results:**
  The DB should be updated with the previous inputs correctly.
- **Actual Results:**
  The DB was updated with the correct inputs but at name length (32 chars) didn’t update.
- **Reason:**
  The name length is within the valid range (3-32 characters).
- **Bug:**
  The name length is 32 but the string is not added to DB see bug report for more details.

Test ID	Test Case	Technique	Status
4	Add Account with (Invalid Age)	Boundary Value Analysis	Passed

- **Description of Test Case**
  Testing the Add Account Functionality by adding an invalid age.

- **Pre-requisites**
  - There is a User Assigned to DB.

- **Test Inputs**
  1. First Input:
     - **Age:** -1 (Invalid, below the minimum valid range)

  2.Second Input:
     - **Age:** 101 (Invalid, above the maximum valid range)

- **Expected Results**
  The DB should not be updated with the previous inputs.

- **Actual Results**
  - The function rejected both input sets and did not update the DB.
  - For Age = -1: The function correctly identified the age as invalid and rejected the input.
  - For Age = 101: The function correctly identified the age as invalid and rejected the input.

- **Reason**
  The age values -1 and 101 are outside the valid range (0–100). The function correctly validated the age field and prevented invalid data from being added to the database.

- **Bug:**
  None

Test ID	Test Case	Technique	Status
5	Add Account with (valid Age)	Boundary Value Analysis	Failed

- **Description of Test Case**

  Testing the Add Account Functionality by adding valid age values within the acceptable range (0–100).

- **Pre-requisites**

  There is a User Assigned to DB.

- **Test Inputs:**
  1. First Input:
     - **Age:** “26 “ (Valid)
  2. Second Input:
     - **Age:** “0” (Valid) Failed
  3. Third input:
     - **Age:** “100” (Valid) Failed

- **Expected Results**

  The DB should be updated with the previous inputs.

- **Actual Results**
  - For Age = 26: The function correctly added the user to the DB.
  - For Age = 0: The function did not add the user to the DB (Bug Identified).
  - For Age = 100: The function did not add the user to the DB (Bug Identified).

- **Reason**

  The age values 26, 0, and 100 are all within the valid range (0–100). However:
  - The age 0 is technically valid but may be considered meaningless in real-world scenarios.
  - The age 100 is valid but may be considered unrealistic in certain contexts.

- **Bug**
  - Age = 0: The user was not added to the DB, even though the age is technically valid. This indicates a potential issue with handling edge cases at the lower boundary.
  - Age = 100: The user was not added to the DB, even though the age is technically valid. This indicates a potential issue with handling edge cases at the upper boundary.

Test ID	Test Case	Technique	Status
6	Add Account with (Invalid Gender)	Boundary Value Analysis	Passed

- **Description of Test Case**

  Testing the Add Account Functionality by adding invalid gender values.

- **Pre-requisites**

  There is a User Assigned to DB.

- **Test Inputs:**
  1. First Input:
     - **Gender:** “DEFAULT_Gender “ (Invalid)
  2. Second Input:
     - **Gender:** “-1” (Invalid)
  3. Third input:
     - **Gender:** “5” (Invalid)

- **Expected Results**

  The DB should not be updated with the previous inputs.

- **Actual Results**
  - For Gender = **DEFAULT_Gender**: The function rejected the input and did not update the DB.
  - For Gender = **-1**: The function rejected the input and did not update the DB.
  - For Gender = **5**: The function rejected the input and did not update the DB.

- **Reason**
  The gender field is validated with valid range for gender inputs.
- **Bug:**
  None

Test ID	Test Case	Technique	Status
7	Add Account with (valid Gender)	Boundary Value Analysis	Passed

- **Description of Test Case**
  Testing the Add Account Functionality by adding valid gender values (Male and Female).
- **Pre-requisites**
  There is a User Assigned to DB.
- **Test Inputs:**
  1. First Input:
     - **Gender:** "Male "   (Valid)
  2. Second Input:
     - **Gender:** "Female"   (Valid)
- **Expected Results**
  The DB should be updated with the previous inputs.
- **Actual Results**
  - For Gender = **Male**: The function correctly added the user to the DB.
  - For Gender = **Female**: The function correctly added the user to the DB.
- **Reason**
  The gender values Male and Female are valid and meet the required format. The function correctly validated the gender field and updated the database accordingly.
- **Bug:**
  None

Test ID	Test Case	Technique	Status
8	Add Account with (invaild educational status)	Boundary Value Analysis	Passed

- ## Description of Test Case

  Testing the Add Account Functionality by adding invalid educational status values.

- ## Pre-requisites

  There is a User Assigned to DB.

- ## Test Inputs:

  1. First Input:
     - **Educational Status:** "DEFAULT_Gender "   (Invalid)
  2. Second Input:
     - **Educational Status:** "-1"   (Invalid)
  3. Third input:
     - **Educational Status:** "8" (Invalid)

- ## Expected Results

  The DB should not be updated with the previous inputs.

- ## Actual Results

  - For Educational Status = **DEFAULT_Status**: The function rejected the input and did not update the DB.
  - For Educational Status = **-1**: The function rejected the input and did not update the DB.
  - For Educational Status = **8**: The function rejected the input and did not update the DB.

- ## Reason

  The educational status values **DEFAULT_Status**, **-1**, and **8** are outside the valid range for educational status inputs. The function correctly validated the educational status field and prevented invalid data from being added to the database.

- **Bug:**
  None

Test ID	Test Case	Technique	Status
9	Add Account with (vaild educational status)	Boundary Value Analysis	Passed

- **Description of Test Case**
  Testing the Add Account Functionality by adding valid educational status values (**Student**, **Faculty_Student**, **Graduate**, **Masters_Student**, **PHD_Student**, and **PHD_Holder**).

- **Pre-requisites**
  There is a User Assigned to DB.

- **Test Inputs:**
  1. First Input:
     - **Educational Status:** “Student “   (valid)
  2. Second Input:
     - **Educational Status:** “Faculty_Student”   (valid)
  3. Third input:
     - **Educational Status:** “Graduate” (Valid)
  4. Fourth input:
     - **Educational Status:** “Masters_Student” (Valid)
  5. Fifth input:
     - **Educational Status:** “PHD_Student” (Valid)
  6. Sixth input:
     - **Educational Status:** “PHD_Holder” (Valid)

- **Expected Results**
  The DB should be updated with the previous inputs.

- ## Actual Results
  - For Educational_Status = **Student**: The function correctly added the user to the DB.
  - For Educational_Status = **Faculty_Student**: The function correctly added the user to the DB.
  - For Educational_Status = **Graduate:** The function correctly added the user to the DB.
  - For Educational_Status = **Masters_Student:** The function correctly added the user to the DB.
  - For Educational_Status = **PHD_Student:** The function correctly added the user to the DB.
  - For Educational_Status = **PHD_Holder:** The function correctly added the user to the DB.
- ## Reason

  All educational status values (**Student**, **Faculty_Student, Graduate, Masters_Student, PHD_Student,** and **PHD_Holder**) are valid and meet the required format. The function correctly validated the educational status field and updated the database accordingly.
- ## Bug:

  None

Test ID	Test Case	Technique	Status
10	Add Account with (invaild username)	Boundary Value Analysis	Passed

- ## Description of Test Case
  Testing the Add Account Functionality by adding invalid usernames that do not meet the required length range (8–32 characters).

- ## Pre-requisites
  There is a User Assigned to DB.

- ## Test Inputs:
  1. First Input:
     - **Username:** “ThisIs33CharactersLong12345678901 “ (Invalid, 33 chars)
  2. Second Input:
     - **Username:** “ABCDEFG”   (Invalid, 7 chars)
  3. Third input:
     - **Username:** “”   (Leave it Empty)

- ## Expected Results
  The DB should not be updated with the previous inputs.

- ## Actual Results
  - For UserName = **ThisIs33CharactersLong12345678901**: The function rejected the input and did not update the DB.
  - For UserName = **ABCDEFG**: The function rejected the input and did not update the DB.
  - For UserName = **""**: The function rejected the input and did not update the DB.

- **Reason**
  The username values **ThisIs33CharactersLong12345678901**, **ABCDEFG**, and **""** are outside the valid range for usernames (8–32 characters). The function correctly validated the username field and prevented invalid data from being added to the database.

- **Bug:**
  None

Test ID	Test Case	Technique	Status
11	Add Account with (vaild username)	Boundary Value Analysis	Failed

- **Description of Test Case**
  Testing the Add Account Functionality by adding valid usernames that meet the required length range (8–32 characters).

- **Pre-requisites**
  There is a User Assigned to DB.

- **Test Inputs**
  1. First Input:
     - ➢ UserName: **AdminUser1** (Valid, 10 characters)
  2. Second Input:
     - ➢ UserName: **ABCDEFGH** (Valid, 8 characters)
  3. Third Input:
     - ➢ UserName: **ThisIsExactly32CharactersLong123** (Valid, 32 characters) Failed

- ## Expected Results

  The DB should be updated with the previous inputs.

- ## Actual Results

  - For UserName = **AdminUser1**: The function correctly added the user to the DB.
  - For UserName = **ABCDEFGH**: The function correctly added the user to the DB.
  - For UserName = **ThisIsExactly32CharactersLong123**: The function failed to add the user to the DB.

- ## Reason

  The username values **AdminUser1** and **ABCDEFGH** are within the valid range and were successfully added to the database. However, the username **ThisIsExactly32CharactersLong123** (32 characters) was not added due to a bug in handling the maximum boundary value.

- ## Bug:

  The username length is 32 but the string is not added to DB see bug report for more details.

Test ID	Test Case	Technique	Status
12	Add Account with (invaild password length)	Boundary Value Analysis	Passed

- **Description of Test Case**

  Testing the Add Account Functionality by adding invalid password lengths that do not meet the required range (8–32 characters).

- **Pre-requisites**

  There is a User Assigned to DB.

- **Test Inputs**

  1. First Input:
     - Password: **ThisIs33CharactersLong12345678901** (Invalid, 33 characters)
  2. Second Input:
     - Password: **ABCDEFG** (Invalid, 7 characters)
  3. Third Input:
     - Password: **""** (Empty string, 0 characters)

- **Expected Results**

  The DB should not be updated with the previous inputs

- **Actual Results**

  - For Password = **ThisIs33CharactersLong12345678901**: The function rejected the input and did not update the DB.
  - For Password = **ABCDEFG**: The function rejected the input and did not update the DB.
  - For Password = **""**: The function rejected the input and did not update the DB.

- **Reason**

  The password values **ThisIs33CharactersLong12345678901**, **ABCDEFG**, and **""** are outside the valid range for passwords (8–32 characters). The function correctly validated the password field and prevented invalid data from being added to the database.

- **Bug**

  None.

Test ID	Test Case	Technique	Status
13	Add Account with (vaild password length)	Boundary Value Analysis	Failed

- **Description of Test Case**

  Testing the Add Account Functionality by adding valid password lengths that meet the required range (8–32 characters).

- **Pre-requisites**

  There is a User Assigned to DB.

- **Test Inputs**

  1. First Input:
     - Password: **Edges12345** (Valid, 10 characters)
  2. Second Input:
     - Password: **ABCDEFGH** (Valid, 8 characters)
  3. Third Input:
     - Password: **ThisIsExactly32CharactersLong123** (Valid, 32 characters) Failed

- ## Expected Results

  The DB should be updated with the previous inputs.

- ## Actual Results

  - For Password = **Edges12345**: The function correctly added the user to the DB.
  - For Password = **ABCDEFGH**: The function correctly added the user to the DB.
  - For Password = **ThisIsExactly32CharactersLong123**: The function failed to add the user to the DB.

- ## Reason

  The password values **Edges12345** and **ABCDEFGH** are within the valid range and were successfully added to the database. However, the password **ThisIsExactly32CharactersLong123** (32 characters) was not added due to a bug in handling the maximum boundary value.

- ## Bug:

  The password length is 32 but the string is not added to DB see bug report for more details.

Test ID	Test Case	Technique	Status
14	Add Account with (invaild password recheck length)	Boundary Value Analysis	Passed

- **Description of Test Case**

  Testing the Add Account Functionality by adding invalid password recheck lengths that do not meet the required range (8–32 characters).

- **Pre-requisites**

  There is a User Assigned to DB.

- **Test Inputs**

  4. First Input:
     - ➢ Password: **ThisIs33CharactersLong12345678901** (Invalid, 33 characters)
  5. Second Input:
     - ➢ Password: **ABCDEFG** (Invalid, 7 characters)
  6. Third Input:
     - ➢ Password: **""** (Empty string, 0 characters)

- **Expected Results**

  The DB should not be updated with the previous inputs

- **Actual Results**
  - ➢ For Password recheck = **ThisIs33CharactersLong12345678901**: The function rejected the input and did not update the DB.
  - ➢ For Password recheck = **ABCDEFG**: The function rejected the input and did not update the DB.
  - ➢ For Password recheck = **""**: The function rejected the input and did not update the DB.

- **Reason**

  The password recheck values **ThisIs33CharactersLong12345678901**, **ABCDEFG**, and **""** are outside the valid range for passwords (8–32 characters). The function correctly validated the password field and prevented invalid data from being added to the database.

- **Bug**

  None.

Test ID	Test Case	Technique	Status
15	Add Account with (vaild password recheck length)	Boundary Value Analysis	Failed

- **Description of Test Case**

  Testing the Add Account Functionality by adding valid password recheck lengths that meet the required range (8–32 characters).

- **Pre-requisites**

  There is a User Assigned to DB.

- **Test Inputs**

  4. First Input:
     - Password recheck: **Edges12345** (Valid, 10 characters)
  5. Second Input:
     - Password recheck: **ABCDEFGH** (Valid, 8 characters)
  6. Third Input:
     - Password recheck: **ThisIsExactly32CharactersLong123** (Valid, 32 characters) Failed

- ## Expected Results
  The DB should be updated with the previous inputs.

- ## Actual Results
  - For Password recheck = **Edges12345**: The function correctly added the user to the DB.
  - For Password recheck = **ABCDEFGH**: The function correctly added the user to the DB.
  - For Password recheck = **ThisIsExactly32CharactersLong123**: The function failed to add the user to the DB.

- ## Reason
  The password recheck values **Edges12345** and **ABCDEFGH** are within the valid range and were successfully added to the database. However, the password **ThisIsExactly32CharactersLong123** (32 characters) was not added due to a bug in handling the maximum boundary value.

- ## Bug:
  The password recheck length is 32 but the string is not added to DB see bug report for more details.

Test ID	Test Case	Technique	Status
16	Add Account with (Password and Password Recheck Do Not Match)	Decision Table Testing	Passed

- **Description of Test Case**

  Testing the Add Account Functionality by ensuring that the function rejects inputs where the password and password recheck do not match.

- **Pre-requisites**

  There is a User Assigned to DB.

- **Test Inputs**
  - Password: **Edges123**
  - Password Recheck: **differentPass**

- **Expected Results**

  The DB should not be updated with the previous inputs.

- **Actual Results**

  The function rejected the input and did not update the DB.

- **Reason**

  The password (**Edges123**) and password recheck (**differentPass**) do not match. The function correctly validated the mismatch and prevented invalid data from being added to the database.

- **Bug**

  None.

Test ID	Test Case	Technique	Status
17	Add Account with (Password and Password Recheck Match)	Decision Table Testing	Passed

- **Description of Test Case**

  Testing the Add Account Functionality by ensuring that the function accepts inputs where the password and password recheck match.

- **Pre-requisites**

  There is a User Assigned to DB.

- **Test Inputs**

  - Password: **Edges123**
  - Password Recheck: **Edges123**

- **Expected Results**

  The DB should be updated with the previous inputs.

- **Actual Results**

  The function correctly added the user to the DB.

- **Reason**

  The password **(Edges123)** and password recheck **(Edges123)** match. The function correctly validated the inputs and updated the database accordingly.

- **Bug**

  None.

Test ID	Test Case	Technique	Status
18	Add Account (Adding Extra User Above Maximum Users - 5)	Equivalence Partitioning	Failed

- **Description of Test Case**
  Testing the Add Account Functionality by attempting to add a sixth user when the database already contains the maximum number of users (5).
- **Pre-requisites**
  There are already 5 users assigned to the database.
- **Test Inputs**
  - Fill DB with vaild 5 user **Edges123**
  - Then try to add user number 6 with vaild data Failed
- **Expected Results**
  - The DB should not be updated with the sixth user's input.
  - Reason: The database can hold only 5 users, and adding a sixth user exceeds the capacity.
- **Actual Results**
  - The first five users were successfully added to the DB.
  - For the sixth user:
    - The function returned **TRUE**, indicating success.
    - However, the sixth user was not actually added to the DB.
- **Reason**
  The database has a fixed capacity of 5 users. When attempting to add a sixth user, the function should return **FALSE** to indicate that the operation failed due to exceeding the maximum capacity. However, the function incorrectly returned **TRUE**, even though the sixth user was not added to the database.

- **Bug**

  The function **Add_Account** returns **TRUE** even if the user is not added to the database

---

Test Function (Delete Account):

Test ID	Test Case	Technique	Status
19	Delete Account (Valid and Invalid User ID)	Boundary Value Analysis	Failed

- **Description of Test Case**

  Verifies the correct behavior of the account deletion functionality for both valid and invalid user IDs. The test ensures that:

  1. Invalid user IDs are rejected and return **FALSE**.
  2. Valid user IDs are accepted, return **TRUE**, and result in the corresponding user being removed from the database.

- **Pre-requisites**

  There are at least 3 users assigned to the database.

- **Test Inputs**

  1. **Invalid User IDs:**
     - ➢ ID = -1 (Below minimum valid range)
     - ➢ ID = 6 (Above maximum valid range, assuming there are only 5 users)
  2. **Valid User IDs:**
     - ➢ ID = 0 (First user)
     - ➢ ID = 2 (Middle user)
     - ➢ ID = 4 (Last user)

- ## Expected Results

  1. For invalid IDs (**-1** and **6**):
     - The function should return **FALSE**.
     - No user should be deleted from the database.
  2. For valid IDs (**0**, **2**, and **4**):
     - The function should return **TRUE**.
     - The corresponding user should be deleted from the database.

- ## Actual Results

  1. For invalid ID = **-1**:
     - The function returned **FALSE**.
     - No user was deleted from the database.
  2. For invalid ID = **6**:
     - The function returned **TRUE**.
     - The user with ID = **5** was incorrectly deleted from the database.
  3. For valid IDs (**0**, **2**, and **4**):
     - The function returned **TRUE**.
     - The corresponding users were correctly deleted from the database.

- ## Reason

  - For invalid ID = **-1**: The function correctly identified the ID as invalid and did not delete any user.
  - For invalid ID = **6**: The function incorrectly returned **TRUE** and deleted the user with ID = **5**. This indicates a bug in the function's validation logic for out-of-range IDs.
  - For valid IDs (**0**, **2**, and **4**): The function correctly identified the IDs as valid, deleted the corresponding users, and updated the database.

- **Bug**
  Bug Description: When attempting to delete a user with an invalid ID (**6**), the function returned **TRUE** and incorrectly deleted the user with ID = **5**.

Test Function (Get Number of Days):

Test ID	Test Case	Technique	Status
20	Get number of days (months with 31 days (non-leap year and leap year))	Equivalence Partitioning	Passed

- **Description of Test Case**
  Verifies the correct behavior of the **Get_Number_Of_Days** function for all months with 31 days, including both non-leap years and leap years.

- **Pre-requisites**
  None.

- **Test Inputs**
  1. Months with 31 Days (Non-Leap Year - 2023):
     - January (Month = 1)
     - March (Month = 3)
     - May (Month = 5)
     - July (Month = 7)
     - August (Month = 8)
     - October (Month = 10)
     - December (Month = 12)
  2. Months with 31 Days (Leap Year - 2024):
     - January (Month = 1)
     - March (Month = 3)

- May (Month = 5)
- July (Month = 7)
- August (Month = 8)
- October (Month = 10)
- December (Month = 12)

- **Expected Results**

  The **Get_Number_OfDays** function should return **31** for all months with 31 days, regardless of whether the year is a leap year or not.

- **Actual Results**

  The function behaves as expected for all inputs.

- **Reason**

  All tested months have 31 days, and the function correctly returns **31** for each of these months. The behavior is consistent across both non-leap years (e.g., 2023) and leap years (e.g., 2024), as leap year rules do not affect months with 31 days.

- **Bug**

  None.

Test ID	Test Case	Technique	Status
21	Get number of days (months with 30 days (non-leap year and leap year))	Equivalence Partitioning	Passed

- **Description of Test Case**

  Verifies the correct behavior of the **Get_Number_OfDays** function for all months with 30 days, including both non-leap years and leap years.

- **Pre-requisites**

  None.

- ## Test Inputs

  1. Months with 30 Days (Non-Leap Year - 2023):
     - April (Month = 4)
     - June (Month = 6)
     - September (Month = 9)
     - November (Month = 11)
  2. Months with 30 Days (Leap Year - 2024):
     - April (Month = 4)
     - June (Month = 6)
     - September (Month = 9)
     - November (Month = 11)

- ## Expected Results

  The **Get_Number_OfDays** function should return **30** for all months with 30 days, regardless of whether the year is a leap year or not.

- ## Actual Results

  The function behaves as expected for all inputs.

- ## Reason

  All tested months have 30 days, and the function correctly returns **30** for each of these months. The behavior is consistent across both non-leap years (e.g., 2023) and leap years (e.g., 2024), as leap year rules do not affect months with 30 days.

- ## Bug

  None

Test ID	Test Case	Technique	Status
22	Get number of days (February in Non-Leap and Leap Years)	Equivalence Partitioning	Passed

- **Description of Test Case**

  Verifies the correct behavior of the **Get_Number_OfDays** function for the month of February in both non-leap years and leap years.

- **Pre-requisites**

  None.

- **Test Inputs**

  1. Non-Leap Year:
     - ➢ Month: February (Month = 2)
     - ➢ Year: 2023 (non-leap year)
  2. Leap Year:
     - ➢ Month: February (Month = 2)
     - ➢ Year: 2024 (Leap year)

- **Expected Results**

  1. For Non-Leap Year (e.g., 2023):
     - ➢ The function should return **28** days for February.
  2. For Leap Year (e.g., 2024):
     - ➢ The function should return **29** days for February.

- **Actual Results**

  1. For Non-Leap Year (2023):
     - ➢ February: Returned **28**.
  2. For Leap Year (2024):
     - ➢ February: Returned **29**.

- **Reason**

The month of February has 28 days in a non-leap year and 29 days in a leap year. The function correctly identifies whether the given year is a leap year or not using the following logic:

  - A year is a leap year if it is divisible by 4 but not divisible by 100, unless it is also divisible by 400.
  - Based on this logic, the function returns the correct number of days for February.

- **Bug**

  None.

Test ID	Test Case	Technique	Status
23	Get number of days (Invalid Days)	Equivalence Partitioning	Passed

- **Description of Test Case**

  Verifies the correct behavior of the **Get_Number_OfDays** function when invalid inputs (e.g., out-of-range months or invalid years) are provided.

- **Pre-requisites**

  None.

- **Test Inputs**

  1. Invalid Months:
     - Month = 0 (Below minimum valid range)
     - Month = 13 (Above maximum valid range)
  2. Invalid Years:
     - Year = -1 (Negative year, invalid)
     - Year = 0 (Invalid year, no calendar year starts at 0)

- **Expected Results**
  - The **Get_Number_OfDays** function should return **0** for all invalid inputs.
  - Reason: Invalid months and years do not correspond to any valid calendar entries.
- **Actual Results**

  The function behaves as expected for all inputs.
- **Reason**

  The function correctly identifies invalid inputs (out-of-range months or invalid years) and returns **0**. This behavior ensures that invalid inputs do not produce incorrect results or cause undefined behavior.
- **Bug**

  None.

Test Function (Show Student Courses):

Test ID	Test Case	Technique	Status
24	Show Student Courses (ID Input is Invalid)	Equivalence Partitioning	Passed

- **Description of Test Case**

  Verifies the correct behavior of the **ShowStudentCourses** function when an invalid user ID is provided as input. The function should handle invalid IDs gracefully and display an appropriate error message.
- **Pre-requisites**
  1. There is at least one user assigned to the database.
  2. Users are enrolled in specific courses:
     - User with ID = **-1** is enrolled in courses **1 (Standard_Diploma)** and **2 (AUTOSAR_Diploma)**.

- User with ID = **0** is enrolled in courses **3 (ARM_Diploma)** and **4 (RTOS_Diploma)**.
- User with ID = **6** is enrolled in courses **5 (Testing_Diploma)** and **6 (EmbeddedLinux_Diploma)**.

Test Inputs

1. Invalid User IDs:
    - ID = **-1** (Below minimum valid range)
    - ID = **6** (Above maximum valid range)
    - ID = **0** (Within invalid range)

Expected Results

- For invalid IDs (**-1**, **0** and **6**):
    - The function should print **"Invalid User Id"**.
    - No course information should be displayed.

Actual Results

The function behaves as expected for all inputs.

Reason

The function correctly identifies invalid user IDs (**-1** and **6**) and handles them by printing **"Invalid User Id"**. For valid user IDs (e.g., **0**), the function retrieves and displays the enrolled courses as expected.

Bug

None.

Test ID	Test Case	Technique	Status
25	Show Student Courses (ID Input is valid)	Boundary Value Analysis	Passed

- ## Description of Test Case

  Verifies the correct behavior of the **ShowStudentCourses** function when a valid user ID is provided as input. The function should display the courses enrolled by the specified user.

- ## Pre-requisites

  1. There is at least one user assigned to the database.
  2. Users are enrolled in specific courses:
     - ➢ User with ID = **1** is enrolled in courses **1 (Standard_Diploma)** and **2 (AUTOSAR_Diploma)**.
     - ➢ User with ID = **2** is enrolled in courses **3 (ARM_Diploma)** and **4 (RTOS_Diploma)**.
     - ➢ User with ID = **5** is enrolled in courses **5 (Testing_Diploma)** and **6 (EmbeddedLinux_Diploma)**.

- ## Test Inputs

  1. Valid User IDs:
     - ➢ ID = **1**
     - ➢ ID = **2**
     - ➢ ID = **5**

- ## Expected Results

  1. For ID = **1**:
     - ➢ The function should display: **"Standard_Diploma"** and **"AUTOSAR_Diploma"**.
  2. For ID = **2**:

    - The function should display: **"ARM_Diploma"** and **"RTOS_Diploma"**.
3. For ID = **5**:
    - The function should display: **"Testing_Diploma"** and **"EmbeddedLinux_Diploma"**.

- ## Actual Results
    - For ID = **1**:
        - Displayed: **"Standard_Diploma"** and **"AUTOSAR_Diploma"**.
    - For ID = **2**:
        - Displayed: **"ARM_Diploma"** and **"RTOS_Diploma"**.
    - For ID = **5**:
        - Displayed: **"Testing_Diploma"** and **"EmbeddedLinux_Diploma"**.
- ## Reason

  The function correctly identifies valid user IDs (**1**, **2**, and **5**) and retrieves the list of courses enrolled by each user. The output matches the expected results, confirming that the function behaves as intended for valid inputs.
- ## Bug

  None.

Test ID	Test Case	Technique	Status
26	Show Student Courses (Valid User with No Registered Courses)	Equivalence Partitioning	Failed

- ## Description of Test Case

  Verifies the correct behavior of the **ShowStudentCourses** function when a valid user ID is provided, but the user has not registered for any courses. The function should handle this scenario gracefully by displaying an appropriate message.

- ## Pre-requisites

  - There is at least one user assigned to the database.
  - A user with ID = **1** exists in the database but is not enrolled in any courses.

- ## Test Inputs

  - User with ID = **1** is assigned to the database.
  - No courses are registered for this user.

- ## Expected Results

  - The function should print: **“You Are Currently Enrolled in:”** and no courses should be displayed.
  - Ideally, the function should also print a message like: **“You Are Currently Not Registered to Any Courses Yet”** to inform the user that they have no registered courses.

- ## Actual Results

  - The function printed: **“You Are Currently Enrolled in:** “and no courses were displayed.
  - However, the function did not print a message indicating that the user is not registered for any courses.

- ### Reason

  The function correctly identifies the valid user ID and attempts to display the list of registered courses. Since the user has no registered courses, the list is empty. However, the function does not provide additional feedback to indicate that the user has no registered courses, leading to a suboptimal user experience.

- ### Bug

  The function does not print a message like **"You Are Currently Not Registered to Any Courses Yet"** when there are no registered courses for the user. But print **"You Are Currently Enrolled in:"**

Test Function (Add Student To Courses):

Test ID	Test Case	Technique	Status
27	Add Student to Course (Valid User and Valid Course)	Decision Table Testing	Passed

- ### Description of Test Case

  Verifies the correct behavior of the **AddStudentToCourse** function when valid inputs (user ID and course ID) are provided. The function should successfully enroll the user in the specified course and return **Enrolled**.

- ### Pre-requisites

  - ➢ There is at least one user assigned to the database.
  - ➢ User with ID = **1** exists in the database and is not yet enrolled in any courses.

- ### Test Inputs

  1. User ID = **1**
  2. Course IDs:
     - ➢ Course ID = **1** (Standard_Diploma)
     - ➢ Course ID = **2** (AUTOSAR_Diploma)

- ## Expected Results

  1. For Course ID = **1**:
     - The function should return **Enrolled**.
     - The user should be successfully enrolled in the course **Standard_Diploma**.
  2. For Course ID = **2**:
     - The function should return **Enrolled**.
     - The user should be successfully enrolled in the course **AUTOSAR_Diploma**.

- ## Actual Results

  The function behaves as expected for all inputs.

- ## Reason

  The function correctly identifies the valid user ID (**1**) and valid course IDs (**1** and **2**). It checks that the user is not already enrolled in the specified courses and successfully enrolls the user. The return value **Enrolled** confirms that the operation was successful.

- ## Bug

  None

Test ID	Test Case	Technique	Status
28	Add Student to Course (Valid User and Valid Course)	Boundary Value Analysis	Failed

- ## Description of Test Case

  Verifies the correct behavior of the **AddStudentToCourse** function when an invalid user ID is provided as input. The function should reject invalid user IDs and not add any courses to them.

- ## Pre-requisites
  - There is at least one user assigned to the database.
  - Users with invalid IDs (**-1**, **0**, and **6**) are assigned to the database but are not valid users.
- ## Test Inputs
  1. Invalid User IDs:
     - ID = **-1** (Below minimum valid range)
     - ID = **0** (Invalid, out of range)
     - ID = **6** (Above maximum valid range)
  2. Valid Course IDs:
     - Course ID = **1** (Standard_Diploma)
     - Course ID = **2** (AUTOSAR_Diploma)
- ## Expected Results
  1. For all invalid user IDs (**-1**, **0**, and **6**):
     - The function should not add any courses to these IDs.
     - The function should return a value indicating failure (e.g., **FALSE** or **InvalidUserID**).
     - No runtime errors or system crashes should occur.
- ## Actual Results
  1. For ID = **-1** and **0**:
     - The system crashed (Runtime Error).
  2. For ID = **6**:
     - The function returned **Enrolled** even though the user ID is invalid.
     - The course was incorrectly added to the database.

Reason

1. For IDs **-1** and **0**:
   - The system crashed due to improper handling of invalid user IDs. This indicates a lack of validation for out-of-range user IDs.
2. For ID **6**:
   - The function incorrectly returned **Enrolled** and added the course to the database, despite the user ID being invalid. This suggests that the function does not properly validate whether the user ID falls within the valid range of existing users.

Bug

- The function **AddStudentToCourse** returns **Enrolled** even if the user ID is invalid (e.g., **6**).
- The system crashes (Runtime Error) when invalid user IDs such as **-1** or **0** are provided.

Test ID	Test Case	Technique	Status
29	Add Student to Course (Valid User and InValid Course)	Equivalence Partitioning	Failed

Description of Test Case

Verifies the correct behavior of the **AddStudentToCourse** function when valid user IDs are provided but invalid course IDs are used. The function should reject invalid course IDs and not add any courses to the users.

Pre-requisites

- There is at least one user assigned to the database.
- Users with valid IDs (**1** and **2**) exist in the database.

- ## Test Inputs
  1. Valid User IDs:
     - ➢ User ID = **1**
     - ➢ User ID = **2**
  2. Invalid Course IDs:
     - ➢ Course ID = **0** (Below minimum valid range)
     - ➢ Course ID = **7** (Above maximum valid range)
- ## Expected Results
  1. For all invalid course IDs (**0** and **7**):
     - ➢ The function should not add any courses to the users.
     - ➢ The function should return a value indicating failure (e.g., **FALSE** or **InvalidCourseID**).
     - ➢ No runtime errors or system crashes should occur.
- ## Actual Results
  1. For Course ID = **0**:
     - ➢ The system crashed (Runtime Error).
  2. For Course ID = **7**:
     - ➢ The function returned **CapacityCompleted**, even though the course ID is invalid.
     - ➢ The system did not crash but incorrectly indicated that the operation was successful.
- ## Reason
  1. For Course ID = **0**:
     - ➢ The system crashed due to improper handling of invalid course IDs. This indicates a lack of validation for out-of-range course IDs.

2. For Course ID = **7**:

   ➢ The function incorrectly returned **CapacityCompleted** and did not validate whether the course ID falls within the valid range of existing courses. This suggests that the function does not properly check for invalid course IDs before proceeding.

- **Bug**
  ➢ The function **AddStudentToCourse** returns **CapacityCompleted** even if the course ID is invalid (e.g., **7**).
  ➢ The system crashes (Runtime Error) when an invalid course ID such as **0** is provided.

Test ID	Test Case	Technique	Status
30	Add Student to Course (Valid User and Valid Course) 3 scenarios	Equivalence Partitioning	Passed

- **Description of Test Case**

Verifies the correct behavior of the **AddStudentToCourse** function when all inputs are valid. The function should handle scenarios such as:

1. Enrolling a user in a course for the first time.
2. Attempting to re-enroll the same user in the same course.
3. Enrolling a new user in a course that has already reached its maximum capacity.

- **Pre-requisites**

1. There is at least one user assigned to the database.
2. Maximum users allowed: 5.
3. Maximum students per course: 5.
4. Valid course IDs: **1 (Standard_Diploma)**, **2 (AUTOSAR_Diploma)**, **3 (ARM_Diploma)**, **4 (RTOS_Diploma)**, **5 (Testing_Diploma)**, **6 (EmbeddedLinux_Diploma)**.

- ## Test Inputs

1. First Scenario:
   - ➢ User ID = **1**
   - ➢ Course ID = **1** (Standard_Diploma)
2. Second Scenario:
   - ➢ User ID = **1**
   - ➢ Course ID = **1** (Standard_Diploma) (Attempt to re-enroll the same user)
3. Third Scenario:
   - ➢ Course ID = **1** (Standard_Diploma) is set to full capacity (**NumberOfEnrolledStudents = 5**).
   - ➢ User ID = **2**

- ## Expected Results

1. For the first registration (User ID = **1**, Course ID = **1**):
   - ➢ The function should return **Enrolled**.
   - ➢ The user should be successfully enrolled in the course.
2. For the second registration attempt (User ID = **1**, Course ID = **1**):
   - ➢ The function should return **AlreadyEnrolled**.
   - ➢ The user should not be enrolled again in the same course.
3. For the third registration attempt (User ID = **2**, Course ID = **1** with full capacity):
   - ➢ The function should return **CapacityCompleted**.
   - ➢ The user should not be enrolled due to the course reaching its maximum capacity.

- ## Actual Results

1. For the first registration (User ID = **1**, Course ID = **1**):

        - Returned: **Enrolled**.
        - The user was successfully enrolled in the course.
2. For the second registration attempt (User ID = **1**, Course ID = **1**):
    - Returned: **AlreadyEnrolled**.
    - The user was not enrolled again in the same course.
3. For the third registration attempt (User ID = **2**, Course ID = **1** with full capacity):
    - Returned: **CapacityCompleted**.
    - The user was not enrolled due to the course reaching its maximum capacity.

- ## Reason

The function correctly identifies and handles the following scenarios:

1. First Enrollment: The user is not yet enrolled in the course, so the function successfully enrolls them and returns **Enrolled**.
2. Re-Enrollment: The user is already enrolled in the course, so the function prevents duplicate enrollment and returns **AlreadyEnrolled**.
3. Full Capacity: The course has reached its maximum capacity, so the function prevents further enrollments and returns **CapacityCompleted**.

- ## Bug

None.

Test Function (Detect User Type):

Test ID	Test Case	Technique	Status
31	Valid Admin Input	Equivalence Partitioning	Passed

- **Description of Test Case**

  Tests if the function **Detect_User_Type** correctly identifies an admin user when valid input corresponding to the admin user type is provided.

- **Pre-requisites**

  None.

- **Test Inputs**

  User_Type = **0** (Corresponds to **AdminMohamedTarek** in the enum).

- **Expected Results**

  - The function should return **AdminMohamedTarek**.
  - Reason: The input value **0** matches the value for **AdminMohamedTarek** in the enum.

- **Actual Results**

  The function returned **AdminMohamedTarek**.

- **Reason**

  The function **Detect_User_Type** correctly identifies the input value **0** as corresponding to the admin user type (**AdminMohamedTarek**) and returns the expected result. The behavior aligns with the expected functionality.

- **Bug**

  None.

Test ID	Test Case	Technique	Status
32	Valid Normal User Input	Equivalence Partitioning	Passed

- ## Description of Test Case
  Tests if the function **Detect_User_Type** correctly identifies a normal user when valid input corresponding to the normal user type is provided.

- ## Pre-requisites
  None.

- ## Test Inputs
  User_Type = **1** (Corresponds to **NormalUser** in the enum).

- ## Expected Results
  - The function should return **NormalUser**.
  - Reason: The input value **1** matches the value for **NormalUser** in the enum.

- ## Actual Results
  The function returned **NormalUser**.

- ## Reason
  The function **Detect_User_Type** correctly identifies the input value **1** as corresponding to the normal user type (**NormalUser**) and returns the expected result. The behavior aligns with the expected functionality.

- ## Bug
  None.

Test ID	Test Case	Technique	Status
33	Valid IncorrectLogin	Equivalence Partitioning	Passed

- **Description of Test Case**
  Tests if the function **Detect_User_Type** correctly identifies an input corresponding to **IncorrectLogin**. The function should return **IncorrectLogin** when the input matches the value for **IncorrectLogin** in the enum.

- **Pre-requisites**
  None.

- **Test Inputs**
  User_Type = **2** (Corresponds to **IncorrectLogin** in the enum).

- **Expected Results**
  - The function should return **IncorrectLogin**.
  - Reason: The input value **2** matches the value for **IncorrectLogin** in the enum.

- **Actual Results**
  The function returned **IncorrectLogin**.

- **Reason**
  The function **Detect_User_Type** correctly identifies the input value **2** as corresponding to the **IncorrectLogin** type and returns the expected result. The behavior aligns with the expected functionality.

- **Bug**
  None.

Test ID	Test Case	Technique	Status
34	Invalid Input (Negative Number)	Equivalence Partitioning	Passed

- **Description of Test Case**

  Tests if the function **Detect_User_Type** correctly handles invalid input in the form of a negative number. Any input other than **0** (AdminMohamedTarek) or **1** (NormalUser) should be treated as invalid and return **IncorrectLogin**.

- **Pre-requisites**

  None.

- **Test Inputs**

  User_Type = **-1** (Invalid input).

- **Expected Results**

  - The function should return **IncorrectLogin**.
  - Reason: Any input other than **0** (AdminMohamedTarek) or **1** (NormalUser) is treated as invalid.

- **Actual Results**

  The function returned **IncorrectLogin**.

- **Reason**

  The function **Detect_User_Type** correctly identifies the input value **-1** as invalid and returns **IncorrectLogin**. The behavior aligns with the expected functionality, as any input outside the valid range (**0** or **1**) is treated as invalid.

- **Bug**

  None.

Test ID	Test Case	Technique	Status
35	Non-Numeric Input Handling	Equivalence Partitioning	Failed

- ## Description of Test Case

  Tests if the function **Detect_User_Type** correctly handles non-numeric input (e.g., characters or strings). The function should reject invalid input and return **IncorrectLogin**.

- ## Pre-requisites

  None.

- ## Test Inputs

  User_Type = **"abc"** (non-numeric input).

- ## Expected Results

  - The function should return **IncorrectLogin**.
  - Reason: Non-numeric input should be rejected, and the function should handle such cases gracefully.

- ## Actual Results

  - The function returned **AdminMohamedTarek** instead of **IncorrectLogin**.
  - Debugging revealed that the **scanf()** function left **User_Type** uninitialized or set it to **0** due to the invalid input. This caused the function to incorrectly interpret the input as **AdminMohamedTarek**.

- ## Reason

  - The function uses **scanf("%d", &User_Type)** to read user input. If the input is non-numeric, **scanf()** fails to read an integer and leaves **User_Type** uninitialized or retains its previous value (defaulting to **0**). This leads to incorrect behavior, as **0** corresponds to **AdminMohamedTarek**.
  - Additionally, failing to handle invalid input can cause undefined behavior, such as infinite loops or corrupted data.

- ## Bug

  - When **scanf()** receives non-numeric input, it does not assign a valid value to **User_Type**. Instead, it may leave **User_Type** uninitialized or retain its default value (**0**), leading to incorrect results.
  - The lack of error handling for **scanf()** causes the function to misinterpret invalid input.

Test Function (Detect User Type):

Test ID	Test Case	Technique	Status
36	Valid Token on First Attempt	Equivalence Partitioning	Passed

- ### Description of Test Case

Tests if the function **Verify_Admin** accepts a valid token (**10203040**) on the first attempt. The function should return **TRUE** when the input matches the predefined **SECRET_ADMIN_TOKEN**.

- ### Pre-requisites

  None.

- ### Test Inputs

  Token = **10203040** (valid token, matching **SECRET_ADMIN_TOKEN**).

- ### Expected Results

  - The function should return **TRUE**.
  - Reason: The input matches the predefined **SECRET_ADMIN_TOKEN**.

- ### Actual Results

  The function returned **TRUE**.

- ### Reason

  The function **Verify_Admin** correctly identifies the input value **10203040** as matching the **SECRET_ADMIN_TOKEN** and returns the expected result. The behavior aligns with the expected functionality.

- ### Bug

  None.

Test ID	Test Case	Technique	Status
37	Valid Token on Second Attempt	Equivalence Partitioning	Passed

- **Description of Test Case**

Tests if the function **Verify_Admin** accepts a valid token (**10203040**) on the second attempt after an invalid token (**99999999**) is entered on the first attempt. The function should return **TRUE** when the input matches the predefined **SECRET_ADMIN_TOKEN**.

- **Pre-requisites**

  None.

- **Test Inputs**

  1. First Attempt:
     - Token = **99999999** (invalid token).
  2. Second Attempt:
     - Token = **10203040** (valid token, matching **SECRET_ADMIN_TOKEN**).

- **Expected Results**

  - The function should return **TRUE**.
  - Reason: The input matches the predefined **SECRET_ADMIN_TOKEN** on the second attempt.

- **Actual Results**

  The function returned **TRUE**.

- **Reason**

  The function **Verify_Admin** correctly handles multiple attempts by rejecting the invalid token (**99999999**) on the first attempt and accepting the valid token (**10203040**) on the second attempt. The behavior aligns with the expected functionality.

- **Bug**

  None.

Test ID	Test Case	Technique	Status
38	Valid Token on Third Attempt	Equivalence Partitioning	Passed

- ## Description of Test Case
  Tests if the function **Verify_Admin** accepts a valid token (**10203040**) on the third attempt after two invalid tokens (**99999999** and **88888888**) are entered. The function should return **TRUE** when the input matches the predefined **SECRET_ADMIN_TOKEN**.

- ## Pre-requisites
  None.

- ## Test Inputs
  1. First Attempt:
     - Token = **99999999** (invalid token).
  2. Second Attempt:
     - Token = **88888888** (invalid token).
  3. Third Attempt:
     - Token = **10203040** (valid token, matching **SECRET_ADMIN_TOKEN**).

- ## Expected Results
  - The function should return **TRUE**.
  - Reason: The input matches the predefined **SECRET_ADMIN_TOKEN** on the third attempt.

- ## Actual Results
  The function returned **TRUE**.

- ## Reason
  The function **Verify_Admin** correctly handles multiple attempts by rejecting the invalid tokens (**99999999** and **88888888**) on the first and second attempts and

accepting the valid token (**10203040**) on the third attempt. The behavior aligns with the expected functionality.

- ## Bug

  None.

Test ID	Test Case	Technique	Status
39	Invalid Token on All Attempts	Equivalence Partitioning	Passed

- ## Description of Test Case

  Tests if the function **Verify_Admin** rejects invalid tokens on all three attempts. The function should return **FALSE** when the input does not match the predefined **SECRET_ADMIN_TOKEN** on any of the three attempts.

- ## Pre-requisites

  None.

- ## Test Inputs

  1. First Attempt:
     - ➢ Token = **99999999** (invalid token).
  2. Second Attempt:
     - ➢ Token = **88888888** (invalid token).
  3. Third Attempt:
     - ➢ Token = **77777777** (invalid token).

- ## Expected Results

  - ➢ The function should return **FALSE**.
  - ➢ Reason: The input does not match the predefined **SECRET_ADMIN_TOKEN** on any attempt.

- ## Actual Results

  The function returned **FALSE**.

- ### Reason
  The function **Verify_Admin** correctly handles multiple invalid token attempts by rejecting all three invalid tokens (**99999999**, **88888888**, and **77777777**) and returning **FALSE**. The behavior aligns with the expected functionality.

- ### Bug
  None.

Test Function (Verify User):

Test ID	Test Case	Technique	Status
40	Valid Username and Password	Decision Table Testing	Passed

- ### Description of Test Case
  Tests if the function **Verify_User** accepts a valid username and password. The function should return **Login_Successful** and correctly populate the **id_ptr** with the user's index in the database array.

- ### Pre-requisites
  A valid user exists in the database.

- ### Test Inputs
  1. Username: **"AdminUser1"**
  2. Password: **"Edges123"**

- ### Expected Results
  - The function should return **Login_Successful**.
  - The **id_ptr** should be updated to the user's index in the database array (**0** for the first user).
  - Reason: The username and password match an entry in the database.

- **Actual Results**
  - The function returned **Login_Successful**.
  - The **id_ptr** was correctly updated to **0**.
- **Reason**
  The function **Verify_User** correctly identifies the input username (**"AdminUser1"**) and password (**"Edges123"**) as matching an entry in the database. It returns **Login_Successful** and updates the **id_ptr** to the correct index (**0**). The behavior aligns with the expected functionality.
- **Bug**
  None.

Test ID	Test Case	Technique	Status
41	Valid Username, Incorrect Password	Decision Table Testing	Passed

- **Description of Test Case**
  Tests if the function **Verify_User** rejects an incorrect password for a valid username. The function should return **Password_incorrect** and set **id_ptr** to **-1**.
- **Pre-requisites**
  A valid user exists in the database.
- **Test Inputs**
  1. Username: **"AdminUser1"** (valid username).
  2. Password: **"WrongPassword"** (incorrect password).
- **Expected Results**
  - The function should return **Password_incorrect**.
  - The **id_ptr** should be updated to **-1**.
  - Reason: The password does not match the entry in the database.

- **Actual Results**
  - The function returned **Password_incorrect**.
  - The **id_ptr** was correctly updated to **-1**.
- **Reason**
  The function **Verify_User** correctly identifies the input username (**"AdminUser1"**) as valid but rejects the incorrect password (**"WrongPassword"**). It returns **Password_incorrect** and updates the **id_ptr** to **-1**. The behavior aligns with the expected functionality.
- **Bug**
  None.

Test ID	Test Case	Technique	Status
42	Non-Existent Username	Decision Table Testing	Passed

- **Description of Test Case**
  Tests if the function **Verify_User** rejects a non-existent username. The function should return **UserName_NotFound** and set **id_ptr** to **-1**.
- **Pre-requisites**
  No matching user exists in the database.
- **Test Inputs**
  1. Username: **"NonExistentUser"** (non-existent username).
  2. Password: **"Edges123"**.
- **Expected Results**
  - The function should return **UserName_NotFound**.
  - The **id_ptr** should be updated to **-1**.
  - Reason: The username does not exist in the database.

- ## Actual Results
  - The function returned **UserName_NotFound**.
  - The **id_ptr** was correctly updated to **-1**.
- ## Reason

The function **Verify_User** correctly identifies that the input username (**"NonExistentUser"**) does not exist in the database. It returns **UserName_NotFound** and updates the **id_ptr** to **-1**. The behavior aligns with the expected functionality.

- ## Bug
  None.