Computer science menu assignment

Analysis

Description of Project:

My project will be software design and development and integrates with database design and development. I intend to use Unreal Engine 5 and C++ to develop a menu for a hoverbike racing game. When users open the game, they will have the option to go to a login screen or a register screen. On the login screen, they will have the ability to enter their username and password into editable text boxes. When they press the login button, the details from the editable text boxes are hashed by a hashing plugin and put inside a SELECT query in prepared statement form. This statement is then executed, and the result is checked. If a record is found with the matching username and password hashes, then the user will be logged in and taken to the bike selection screen, and the raw username will be stored in an instance of the MainMenuManager class. If the query comes back unsuccessfully, an “Incorrect username or password” popup appears on the screen.

If the user clicks on the register screen from the main menu, they are taken to the register screen, where they are presented with 3 editable text boxes. One for the username, password, and confirmed password. These details are then validated to make sure they meet the requirements:

Username must not be null. Password must be at least 8 characters long. And the confirmed password must match the password already entered.

If any of these are not met, then before the rest query is executed, the user is displayed an error that corresponds to the requirement that they did not meet. The username and password are then hashed by the hashing plugin and placed inside the INSERT INTO SQL query in prepared statement form. The query is executed, and the result is checked to make sure the query is executed correctly. If the query is successful, then the raw username is stored in an instance of the MainMenuManager class, and the user is taken to the bike selection screen. If the query comes back unsuccessfully then the error is read. The error should be that the username is taken because that should be the only limiting factor at this point. If this is an error, then a “Username is taken” popup appears on the screen. If it’s a different error (this shouldn’t happen), then “Fail: reason unknown” popup appears on the screen.

When the user gets to the bike selection screen, they are presented with buttons with images of bikes on, and the name of each bike below. The user has the option to sort the bikes by each attribute by clicking either of 3 buttons: Top speed, Acceleration, Handling. These buttons call a function that instantiates the hoverbike objects and bubble sorts them by the appropriate property, and then changes the order of the image buttons and names, to reflect the result of the bubble sort. When the user clicks on one of the buttons, an instance of the button is stored in an instance of the MainMenuManager class, and the user is taken to the track selection screen. On the track selection screen, the user has a simple choice between 3 tracks, represented by buttons with images of the track layout on them. When the user clicks on one of these buttons, the image of the appropriate track is then stored in an instance of the MainMenuManager class, the user is then taken to the “Chosen” screen, which shows them their username and selected bike and track, which are fetched from the instance of the MainMenuManager class.

On every screen apart from the Main menu screen, a “Back” button will be in the top left corner. Clicking it will take the user back to the main menu screen.

The end users of my menu will be players, that are about to play a game, this menu should bring them through the process of signing in, and choosing a bike and track to play on.

Scope:

In this project I will create the following:

1. Requirements specification

2. A full design including pseudocode, UI wireframe, ER diagram, EO diagram, and data dictionary.

3. A working menu that integrates with a database containing hashes of usernames and passwords.

5. A full test plan including the type of tests, expected output, and testing to see if it is fit for purpose and meets end-user requirements.

6. The full results of all tests. Including test personas.

7. An evaluation of the project

Constraints:

The various constraints that will affect the project are:

1. Unreal Engine 5.3 will be used to create the menu as it is completely free, up-to-date and it has plenty of support.

2. My menu will be able to run on windows computers.

3. The budget for the project is £0 as all software and assets are fully licensed or free.

4. Visual studio code with C++ compiler for implementation.

5. I will have an existing database that my Database Manager class will connect to.

6. I will ensure that my project is completed by the deadline of 18th March as it will need to be delivered to SQA for marking.

Boundaries:

The result will have the following key features.

1. 6 screens that take you through the process of registering/login all the way to a confirmation of username chosen bike and chosen track.

2. A process for existing users to login using their username and password, which gets hashed and authenticated against the hash of their existing stored details to make sure they match.

3. All inputs will be validated

4. An array of objects, where the objects are bikes with attributes: Name, Top Speed, Acceleration and Handling.

5. A bubble sort that can sort the bikes by either Top Speed, Acceleration and Handling

6. A recurring colour theme, so as not to confuse the user.

Requirements Specification:

End user requirements:

The user would like to

* Register for an account using a username and password
* Be told what is wrong with their username and password when an error occurs on account creation
* Login to their account using their username and password
* Be given an error when the username and password entered in login cannot be validated
* Have to enter their password twice so as to make sure they don’t make a typo during registration
* Have their details hashed so malicious people can’t steal their data from the database
* Have different bikes to choose from
* Be able to sort the bikes to show which are best in each aspect
* Choose from different tracks
* Have their options confirmed at the end of the menu
* Navigate the menu through the use of buttons
* Have a button to go back to the main menu
* See a consistent colour scheme throughout so as not to be confusing
* Have minimum requirements of password creation so passwords are secure enough

Functional requirements SDD:

* Handle on click events of buttons that lead to the execution of SQL queries
* Retrieve values from editable text boxes
* Validate that passwords meet a criteria during account creation
* Validate that username is unique on account creation
* Validate that record exists with matching username and password hashes during login
* Set the text of an error message to the appropriate error during the failure to meet the validation on either login or registration screens.
* Hash username and password using a hashing plugin when creating an account and logging in.
* Have 3 hoverbike classes with unique values for properties.
* Have a bubble sort using the values of each of the 3 hoverbike classes.
* Display an ordered list of hoverbikes on the screen corresponding to the bubble sort results
* Have 3 bikes and tracks for the user to choose from.
* Store the user’s username, chosen bike and chosen track and show them their selections at the final screen.
* Have on-click functionality for buttons that change the screen to the next screen.
* Have a button on all screens that takes the user back to the main menu.

Functional requirements DDD:

* Have an existing database to connect to
* Have UserID as a primary key
* Have INSERT INTO query with 2 hashed values so that new users can be created.
* Have a SELECT query to check if there is a hashed username and password that matches the hashed values of the username and password entered by the user.

UML Use Case Diagram:

A diagram of a company

Description automatically generated

Gantt Chart Project Plan:

A graph with a black and white line

Description automatically generated

**Resources Required:**

1. Analysis

* Chrome (Websites for creating UML use case diagram)
* MS Excel (Creating Gantt chart)
* MS Word (writing analysis)

2. Design

* Chrome (Websites for creating diagrams.)
* MS Word (Writing design)

3. Implementation

* Chrome (Programming forums for help, accessing fully licensed free assets.)
* MS Word (Writing final implementation information into)
* Visual Studio Code (For writing C++)
* Unreal Engine 5.3 (For creating UI)
* SQLite3 plugin for UE5 (For accessing the database from the code)
* DB Browser (For creating the database and on-going testing)
* Hashing plugin for UE5

4. Testing

* MS Word (Writing testing section)
* Visual Studio Code (For adding logs)
* Unreal Engine 5.3
* SQLite3 plugin for UE5 (For accessing the database from the code)
* DB Browser (To check before and after INSERT INTO query)
* Hashing plugin for UE5

5. Evaluation

* MS Word (For writing evaluation into)

Design

**UML Class diagram:  
A screenshot of a computer screen

Description automatically generated**

This is the full class diagram, I will break it down for readability below

A white background with black text

Description automatically generated

A diagram of a computer

Description automatically generated with medium confidence

A diagram of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

**A grid with text and numbers

Description automatically generated with medium confidenceA white grid with black squares and a rectangle with text

Description automatically generated with medium confidence**

These all inherit from MenuInteraction class ^

**Pseudocode:**

**MenuInteraction:**

Each of the numbers is a function in this class. I will break down the functions that contain advanced higher concepts:

1. Constructer (NativeConstruct) function
2. ChangeScreenAndSetUsername: Optionally takes username as parameter and sets username in MainMenuManager instance if necessary and then calls change screen function.
3. ChangeScreenDefault: Calls ChangeScreenAndSetUsername with no username parameter.
4. SelectTrack1: Sets the relevant track as the chosen track in the MainMenuManager instance. Then calls ChangeScreenAndSetUsername function with no parameters.
5. SelectTrack2: Sets the relevant track as the chosen track in the MainMenuManager instance. Then calls ChangeScreenAndSetUsername function with no parameters.
6. SelectTrack3: Sets the relevant track as the chosen track in the MainMenuManager instance. Then calls ChangeScreenAndSetUsername function with no parameters.
7. SelectBike1: Sets the relevant bike as the chosen bike in the MainMenuManager instance. Then calls ChangeScreenAndSetUsername function with no parameters.
8. SelectBike2: Sets the relevant bike as the chosen bike in the MainMenuManager instance. Then calls ChangeScreenAndSetUsername function with no parameters.
9. SelectBike3: Sets the relevant bike as the chosen bike in the MainMenuManager instance. Then calls ChangeScreenAndSetUsername function with no parameters.
10. SortSpeed: Calls the SortBikeAndSetLooks function with sorting criteria 1
11. SortAcceleration: Calls the SortBikeAndSetLooks function with sorting criteria 2
12. SortHandling: Calls the SortBikeAndSetLooks function with sorting criteria 3
13. SortBikeAndSetLooks:
    1. IF Image container or Text container don’t exist THEN
    2. LOG “Container could not be found”
    3. RETURN
    4. END IF
    5. DECLARE Button Array
    6. DECLARE HoverBike Array
    7. INSTANTIATE button widget and ADD TO Button Array
    8. INSTANTIATE Hoverbike object and ADD TO HoverBike Array
    9. SET Integer ArrayLength as the length of the HoverBike Array
    10. SET Boolean swapped to TRUE
    11. SET Integer StopIndex as ArrayLength
    12. WHILE Swapped = TRUE and StopIndex >=0 DO
    13. LOOP IN RANGE StopIndex DO
    14. DECLARE float CurrentValue
    15. DECLARE float NextValue
    16. SWITCH SortingCriteria
    17. CASE = 1 DO
    18. SET CurrentValue = HoverBike Array [index] -> Get value of top speed
    19. SET NextValue = HoverBike Array [index+1] -> Get value of top speed
    20. BREAK
    21. CASE = 2 DO
    22. SET CurrentValue = HoverBike Array [index] -> Get value of acceleration
    23. SET NextValue = HoverBike Array [index+1] -> Get value of acceleration
    24. BREAK
    25. CASE = 3 DO
    26. SET CurrentValue = HoverBike Array [index] -> Get value of handling
    27. SET NextValue = HoverBike Array [index+1] -> Get value of handling
    28. BREAK
    29. DEFAULT DO
    30. LOG “Invalid sorting criteria
    31. RETURN
    32. END SWITCH
    33. IF CurrentValue LESS THAN NextValue DO
    34. SWAP HoverBike Array [index], HoverBike Array [index +1]
    35. SWAP Button Array [index], Button Array [index +1]
    36. SET Swapped = TRUE
    37. END IF
    38. END FOR LOOP
    39. SET StopIndex to StopIndex – 1
    40. END WHILE LOOP
    41. Clear children from image and text containers
    42. LOOP FOR ArrayLength DO
    43. SET ResultText = HoverBikeArray[index] -> get name
    44. DESTROY Hoverbike Array [index]
    45. SET TextBlock style and text
    46. ADD TextBlock as child of text container
    47. END FOR LOOP
    48. LOOP FOR Each Button DO
    49. ADD Button as child of image container
    50. SET Image Slot style and format
    51. END FOR LOOP
14. QueryButtonHandler: Gets the values from username, password and if existing, the confirm password editable text boxes. From this it determines whether login or register button has been pressed. Then calls either the login or create user function from the database manager class. If the function returns an error, it displays the error on the screen.
15. SpawnActor: Takes an actor class as a parameter. Gets a reference to the instance of the world, makes default Spawn parameters, instantiates the actor within the world and returns it.
16. CreateButtonWithImage: Takes path to image file as parameter and integer corresponding to which bike is being created as a parameter. Function creates a button widget, sets the style of it including adding the image to it. Then adds the onclick dynamic of the button according to the bike integer parameter.

MainMenuManager:

1. AMainMenuManager: Sets can tick to false.
2. BeginPlay: Super BeginPlay, and calls ChangeScreen function.
3. ChangeScreen: Takes the blueprint widget class as a parameter. Gets the instance of the world and creates an instance of the widget parameter and adds it to the viewport.

DatabaseManager:

1. ADatabaseManager: Sets can tick to false.
2. CreateUser:
3. ADatabaseManager: Sets can tick to false.
4. BeginPlay: Super BeginPlay.
5. Tick: Super Tick.
6. CreateUser: (username, password and confirmed password)
   1. SET Result = “Null”
   2. IF password length >= 8 and confirmed password == password and username length > 0 THEN
   3. INSTANTIATE HashGenerator
   4. Hash the username
   5. SET HashedUsername = hashed username
   6. Hash the password
   7. SET HashedPassword = hashed password
   8. DECLARE DBFilePath = path to the database file
   9. DECLARE CreateUserSQL = INSERT INTO QUERY with hashed username and password passed in
   10. IF INSTANTIATE Link to database using DBFilePath successful THEN
   11. IF INSTANTIATE Prepared SQL statement using CreateUserSQL THEN
   12. Run the Query and store the success as a Boolean.
   13. IF Success = TRUE THEN
   14. LOG “User Created Successfully”
   15. SET Result = “Success”
   16. END IF
   17. ELSE THEN
   18. LOG “Failed to create user:” pass in the error message
   19. IF last error is username taken THEN
   20. SET Result = “Username Taken”
   21. END IF
   22. ELSE THEN
   23. SET Result = “Fail: Reason Unknown”
   24. END ELSE
   25. END ELSE
   26. DESTROY prepared statement
   27. END IF
   28. DESTROY Database link
   29. END IF
   30. ELSE IF password length LESS THAN 8 THEN
   31. LOG “Failed to create user: Password is too short”
   32. SET Result = “Password is too short”
   33. END ELSE IF
   34. ELSE IF password NOT EQUAL TO confirmed password THEN
   35. LOG “Failed to create user: Passwords don’t match”
   36. SET Result = “Passwords don’t match”
   37. END ELSE IF
   38. ELSE IF username length EQUAL TO 0 THEN
   39. LOG “Failed to create user: Username is null”
   40. SET Result = “Username can’t be blank”
   41. END ELSE IF
   42. RETURN Result
7. Login: (username and password)
   1. INSTANTIATE HashGenerator
   2. Hash the username
   3. SET HashedUsername = hashed username
   4. Hash the password
   5. SET HashedPassword = hashed password
   6. DECLARE DBFilePath = path to the database file
   7. DECLARE LoginSQL = SELECT QUERY with hashed username and password passed in
   8. IF INSTANTIATE Link to database using DBFilePath successful THEN
   9. IF INSTANTIATE Prepared SQL statement using CreateUserSQL THEN
   10. Run the Query and store the success as a stepResult.
   11. IF the stepResult has a row THEN
   12. LOG “\_\_\_ Created Successfully” with username passed in
   13. SET Result = “Success”
   14. END IF
   15. ELSE THEN
   16. LOG “Failed to login \_\_\_: \_\_\_” pass in the username and error message
   17. SET Result = “Incorrect Username or password”
   18. END ELSE
   19. DESTROY prepared statement
   20. END IF
   21. DESTROY Database link
   22. END IF
   23. RETURN Result

HoverBike:

1. AHoverBike: Creates default values, name, top speed, acceleration and handling
2. Getters and for all properties

AeroSwift:

1. AAeroswift: Constructor with new values

NimbusRider:

1. ANimbusRider: Constructor with new values

SkyRunner:

1. SkyRunner: Constructor with new values

**User interface:**

A diagram of a computer program

Description automatically generated

**A screenshot of a login page

Description automatically generated**

**A screenshot of a registration page

Description automatically generated**

**A diagram of a mail

Description automatically generated with medium confidence**

**A diagram of a track selection page

Description automatically generated**

**A diagram of a diagram of a diagram

Description automatically generated with medium confidence**

**Integration queries:**

**Login query**

|  |  |
| --- | --- |
| **Field(s)/calculation(s)** | UserID |
| **Table(s) and query(-ies)** | Users |
| **Search Criteria** | Username = (Hash of entered username) AND Password = (Hash of the entered password) |
| **Grouping** |  |
| **Having** |  |
| **Sort Order** |  |

**Create User query**

|  |  |
| --- | --- |
| **Type of query** | INSERT INTO |
| **Field(s)/calculation(s)** | UserID |
| **Table(s) and query(-ies)** | Users |
| **New Values** | Username = (Hash of entered username) AND Password = (Hash of the entered password) |

**Data Dictionary:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute name** | **Key** | **Type** | **Size** | **Required** | **Validation** |
| UserID | PK | integer |  | yes |  |
| Username |  | varchar | 255 | yes | Unique |
| Password |  | varchar | 255 | yes |  |

**Data flow:**

|  |  |
| --- | --- |
| **Native Construct (MenuInteraction class)** |  |
| **Input** |  |
| **Processes** | Casting all components on screen |
| **Outputs** | References to all components on the relevant screen |

|  |  |
| --- | --- |
| **ChangeScreenDefault** |  |
| **Input** |  |
| **Processes** |  |
| **Outputs** | Calls the ChangeScreenAndSetUsername function with no parameters |

|  |  |
| --- | --- |
| **SelectTrack1** |  |
| **Input** |  |
| **Processes** |  |
| **Outputs** | Stores the relevant track in the MainMenuManager instance  Calls the ChangeScreenAndSetUsername function with no parameters |

|  |  |
| --- | --- |
| **SelectTrack2** |  |
| **Input** |  |
| **Processes** |  |
| **Outputs** | Stores the relevant track in the MainMenuManager instance  Calls the ChangeScreenAndSetUsername function with no parameters |

|  |  |
| --- | --- |
| **SelectTrack3** |  |
| **Input** |  |
| **Processes** |  |
| **Outputs** | Stores the relevant track in the MainMenuManager instance  Calls the ChangeScreenAndSetUsername function with no parameters |

|  |  |
| --- | --- |
| **SelectBike1** |  |
| **Input** |  |
| **Processes** |  |
| **Outputs** | Stores the relevant bike in the MainMenuManager instance  Calls the ChangeScreenAndSetUsername function with no parameters |

|  |  |
| --- | --- |
| **SelectBike2** |  |
| **Input** |  |
| **Processes** |  |
| **Outputs** | Stores the relevant bike in the MainMenuManager instance  Calls the ChangeScreenAndSetUsername function with no parameters |

|  |  |
| --- | --- |
| **SelectBike3** |  |
| **Input** |  |
| **Processes** |  |
| **Outputs** | Stores the relevant bike in the MainMenuManager instance  Calls the ChangeScreenAndSetUsername function with no parameters |

|  |  |
| --- | --- |
| **SortSpeed** |  |
| **Input** |  |
| **Processes** |  |
| **Outputs** | Calls SortBikeAndSetLooks function with sorting criteria 1 |

|  |  |
| --- | --- |
| **SortAcceleration** |  |
| **Input** |  |
| **Processes** |  |
| **Outputs** | Calls SortBikeAndSetLooks function with sorting criteria 2 |

|  |  |
| --- | --- |
| **SortHandling** |  |
| **Input** |  |
| **Processes** |  |
| **Outputs** | Calls SortBikeAndSetLooks function with sorting criteria 3 |

|  |  |
| --- | --- |
| **SortBikeAndSetLooks** |  |
| **Input** | Sorting criteria integer |
| **Processes** | Checks the containers are referenced correctly.  Instantiates buttons and hoverbike objects and adds them to respective arrays.  Bubble sorts the hoverbike objects by the sorting criteria specified.  Clears the existing components from the button and text box panels.  Creates and modifies the style of the text boxes and buttons. |
| **Outputs** | Adds the text boxes and buttons to the panels, in the new sorted order. |

|  |  |
| --- | --- |
| **QueryButtonHandler** |  |
| **Input** |  |
| **Processes** | Check the username and password text boxes are referenced.  Gets values from editable text boxes.  Instantiates database manager. |
| **Outputs** | Calls login or create user function depending on if the confirm password editable text box exists.  Calls ChangeScreen function if function called successfully.  Puts relevant error pop up on screen if function is unsuccessful. |

|  |  |
| --- | --- |
| **CreateUser** |  |
| **Input** | Username, password, confirmed password |
| **Processes** | Validates details against minimum criteria.  Hashes username and password.  Creates database link and prepared statement. |
| **Outputs** | Executes INSERT INTO query.  Returns a relevant error/success string depending on output. |

|  |  |
| --- | --- |
| **Login** |  |
| **Input** | Username, password |
| **Processes** | Hashes username and password.  Creates database link and prepared statement. |
| **Outputs** | Executes SELECT query.  Returns a relevant error/success string depending on output. |

|  |  |
| --- | --- |
| **ChangeScreenAndSetUsername** |  |
| **Input** | Mode, Username |
| **Processes** | Stores the username if the mode is 1. |
| **Outputs** | Calls change screen function |

|  |  |
| --- | --- |
| **ChangeScreen** |  |
| **Input** | UUserWidget |
| **Processes** | Get reference to player controller  Shows mouse cursor  Creates instance of user widget |
| **Outputs** | Adds the widget instance to the viewport |

Implementation

**UI Screens:**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

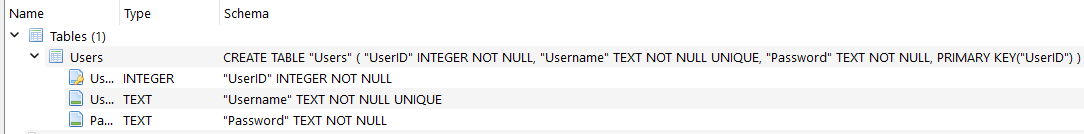
Description automatically generated

Database structure:

|  |  |  |
| --- | --- | --- |
| Name | Type | Schema |
| Users |  | CREATE TABLE “Users” ( “UserID” INTEGER NOT NULL, “Username” TEXT NOT NULL UNIQUE, “Password” TEXT NOT NULL, PRIMARY KEY(“UserID”) ) |
| UserID | INTEGER | “UserID” INTEGER NOT NULL |
| Username | TEXT | “Username” TEXT NOT NULL UNIQUE |
| Password | TEXT | “Password” TEXT NOT NULL |

No initial values necessary.

Actual database:



This shows that it meets the structure outlines.

**DDD:**

**Result of SELECT query (from log):**

Result: 1 rows returned in 3ms

At line 1:

SELECT Username, Password FROM Users WHERE Username = '348596b9c04e8892838181b4a33a8b5a0c89b0bb08507b4dcf5a2371da526f2f' AND Password = 'e24df920078c3dd4e7e8d2442f00e5c9ab2a231bb3918d65cc50906e49ecaef4';

Execution finished without errors.

**Tables updated by INSERT INTO query:**

Database INSERT query:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Implementation of user interface:**

MainMenu:

A screenshot of a computer

Description automatically generated

Login Screen:

A screenshot of a computer login

Description automatically generated

Incorrect Login:

A screenshot of a computer login screen

Description automatically generated

Registration Screen:

A screenshot of a computer

Description automatically generated

Register username taken:

A screenshot of a computer

Description automatically generated

Register Password short:

A screenshot of a computer

Description automatically generated

Register passwords don't match:

A screenshot of a computer

Description automatically generated

Bike Selection:

A screenshot of a computer

Description automatically generated

Track Selection:

A screenshot of a computer

Description automatically generated

Chosen Screen:

A screenshot of a computer

Description automatically generated

**Description of new skills and/or knowledge:**

Unreal Engine:

I have never used unreal engine before this. I chose unreal engine 5 because it is the most up-to-date version so it would have the most support. When I first downloaded it I had to uninstall visual studio code from my computer, and it automatically installed visual studio community 2019. I watched many videos on how to use unreal engine, for basic things like making a class. It took my about 4 tries to finally get a project that works. I initially made my whole project in blueprints with relative ease, changing it all to blueprints was the hard part. I had to learn how to get references to components in the blueprint UMG (visual UI maker). The main problem I experienced was the fact I could not get a reference to the UMG widgets I made inside C++, this meant on the main menu I had to make 2 pre-set buttons blueprints so that they could reference the other screens. It also took a while to figure out how the whole hierarchy works in unreal engine with classes and everything. Like I'm not sure how people are just supposed to remember all the functions and properties of every UE class.

https://youtu.be/k-zMkzmduqI?si=N7-KwBER3dMDWoyz

https://youtu.be/b4\_msnAbJtk?si=xy6k2BtyEpTz4sSx

https://youtu.be/LneO7jvrQhw?si=0ikvEE7cuRK9Ajn9

https://youtu.be/HNfpKFER2hI?si=vp9H-P80MV0Tr44F

https://youtu.be/Hn2aPOWsVzQ?si=NwlJ02AiamUgouMx

https://youtu.be/kumZj\_mov58?si=oDuAta97kRZjt29k

https://youtu.be/FBpnOuCgHu4?si=RDwXxB6r7kROyOPm

https://youtu.be/dS5AUaYFcdw?si=gx0HEpSxMNL8Ax5T

https://youtu.be/nATsj1ehYMU?si=vmr8nDx8sH6K81ss

https://youtu.be/qtUdbCpCGug?si=UOlD\_LuW4vKN8DEs

https://youtu.be/Yr8beK9FMe0?si=oTsmtIpc19eW5z12

https://docs.unrealengine.com/5.3/en-US/

C++:

I was initially hoping I could use python to program in Unreal engine but I discovered quite quickly that you can only use python to script, not actually code anything. I then started learning C++ in class using my good friend W3Schools. After I got comfortable, with C++, I started coding it in Unreal engine, I soon found out that C++ in unreal engine is extremely different to normal C++. Things like having to put UPROPERTY() before you declared properties in the header file. It took a while to get used to putting semicolons after everything, since my first programming language was python. This was aided by my Scottish Baccalaureate project where I was using JavaScript a fair amount, which helped that mindset of semicolon for execution. I was also very clueless when I created a class in unreal engine and I was presented with 2 files, a cpp and an h. What is a .h? It took a while to figure out how it all worked but I now have a good understanding of the uses of both file types. I also struggled with the use of functions that returned results. Like I had a function that created a button, but I was puzzled for a while on how to implement it in both the cpp and h. The solution I found is that it's similar to declaring a variable in that you put the type before the function name, instead of just "def myFunction():"

https://www.w3schools.com

https://docs.unrealengine.com/5.3/en-US/

**Log of ongoing testing:**

26/12/2023 - 07/01/2024

I kept encountering an error where my program would crash on running. This was extremely annoying as I would have to reopen unreal engine every time, meaning I had to recompile and set up the main menu manager instance. After a few weeks of fixing it by accident every time it happened I figured out the problem was when I had a function that returned something, but there was routes that the program could take which would cause the function to never return. To fix it you simply add a “break” or “return;”.

My code with the error:

A screenshot of a computer error

Description automatically generated

My fixed code:

A screen shot of a computer error

Description automatically generated

15/01/2024

I wasn't able to run any of my SQL queries through my program. I was looking at the logs and kept finding it saying Disk I/O error. I didn't have any idea what the error meant as it was the start of my SQLite usage in unreal engine. After some research I realized it was because SQLite3 databases can only be accessed by 1 program at a time. And I was running DB Browser in the background. As soon as I closed DB Browser and tried again, it ran perfectly 🤡.

Example of error in log:

LogSQLiteDatabase: Warning: Failed to open database 'S:/UE5/ComputingProject/PleasePleasePlease/Content/Database/UsersDB.db': disk I/O error

https://forums.unrealengine.com/t/error-opening-reading-sqlite-database-db-file-i-o-disk-error-ue-5-1/772924

21/01/2024

I initially used Blake3 to has my usernames and passwords, but while I was testing a feature, I stumbled across the fact that I could enter details that were very similar, and it would login. For example, if there was a user with username "test" and password "12345678", I could enter "test" and "12345679", and it would still login "successfully". To fix this, I changed my hashing plugin from Blake3 to SHA256, as I had heard it was quite well renowned as working. After researching how to switch plugins and implementing SHA256, I tested it with the false details I entered earlier, and it successfully hashed details correctly and didn't get confused with similar details. This plugin was also beneficial because it required less lines of code. This is my code once I had started using the plugin:



A screenshot of a computer code

Description automatically generated

https://www.unrealengine.com/marketplace/en-US/product/sha256-hash-functions

https://github.com/BlueMountainsIO/ue-plugin-sha256/tree/master

https://emn178.github.io/online-tools/sha256.html

10/01/2024

When I got a reference to a UMG buttons from my C++, I have to ability to "AddDynamic" of a onclick function. My main problem with this was that the function to be called on click, could not have parameters. I have no idea why unreal engine does this and I could not find a solution that let me use parameters. To counter this I made functions like SortSpeed, which was called on the click of the sort speed button. This function would simply call a function that did have a parameter, with the correct parameter. I personally see this style of coding very inelegant and I'm not sure why Unreal engine forces me to do it this way :/

Here is my code with the error:

A close-up of a computer screen

Description automatically generated

Here is an example of the fixed code:

A computer screen shot of a computer code

Description automatically generated

****

<https://forums.unrealengine.com/t/umg-what-kind-of-parameters-can-the-click-event-function-of-a-button-accept/12179>

**Code:**

**AeroSwift.cpp**

// Name: Geronimo Jones SCN: 120763237

#include "AeroSwift.h"

// Sets default values

AAeroSwift::AAeroSwift()

{

// Set custom values for properties

name = "AeroSwift";

topSpeed = 750.f;

acceleration = 1000.f;

handling = 1250.f;

}

**AeroSwift.h:**

// Name: Geronimo Jones SCN: 120763237

#pragma once

#include "CoreMinimal.h"

#include "HoverBike.h"

#include "AeroSwift.generated.h"

/\*\*

\*

\*/

UCLASS()

class AAeroSwift : public AHoverBike

{

GENERATED\_BODY()

public:

// Sets default values for this hoverbike

AAeroSwift();

};

**DatabaseManager.cpp:**

// Name: Geronimo Jones SCN: 120763237

#include "DatabaseManager.h"

#include "SQLiteDatabase.h"

#include "Hash/Blake3.h"

#include "SHA256Hash.h"

// Sets default values

ADatabaseManager::ADatabaseManager()

{

// Set this actor to call Tick() every frame

PrimaryActorTick.bCanEverTick = false;

}

// Function to create a new user from details passed in. Auto creates userID which auto-increments for every new user

FString ADatabaseManager::CreateUser(FString username, FString password, FString confirmedPass)

{

FString result = TEXT("null");

if ((password.Len() >= 8) && (confirmedPass == password) && (username.Len() > 0))

{

// Create an instance of FSHA256Hash

FSHA256Hash HashGenerator;

// Convert the string to hash

HashGenerator.FromString(password);

// Get the hashed string

FString HashedPass = HashGenerator.GetHash();

// Convert the string to hash

HashGenerator.FromString(username);

// Get the hashed string

FString HashedUsername = HashGenerator.GetHash();

// Defines the "DBFilePath" as the directory path to the database and defines "CreateUserSQL" as the SQL query to be run.

const FString DBFilePath = FPaths::ConvertRelativePathToFull(FPaths::ProjectContentDir() + "Database/UsersDB.db");

const FString CreateUserSQL = FString::Printf(TEXT("INSERT INTO Users (Username, Password) VALUES('%s', '%s');"), \*HashedUsername, \*HashedPass);

// If the Database opens correctly into "UserDB" so that it can be read and written to, then the following code is run.

if (FSQLiteDatabase\* UserDB = new FSQLiteDatabase();

UserDB->Open(\*DBFilePath, ESQLiteDatabaseOpenMode::ReadWrite))

{

// If the "SQLQueryStatement" is created correctly then it runs the following code

if (FSQLitePreparedStatement\* SQLQueryStatement = new FSQLitePreparedStatement();

SQLQueryStatement->Create(\*UserDB, \*CreateUserSQL, ESQLitePreparedStatementFlags::Persistent))

{

// Runs the SQL query and returns if it ran correctly or not

bool success = SQLQueryStatement->Execute();

// If the sql query ran correctly then sends the success message to the console. If it didn't run correctly then it would display the error message.

if (success)

{

UE\_LOG(LogTemp, Log, TEXT("User created successfully!"));

result = TEXT("Success");

}

else

{

UE\_LOG(LogTemp, Error, TEXT("Failed to create user: %s"), \*UserDB->GetLastError());

if (FString(UserDB->GetLastError()) == FString(TEXT("UNIQUE constraint failed: Users.Username")))

{

result = TEXT("Username Taken");

}

else

{

result = TEXT("Fail: Reason Unknown");

}

}

// Removes the SQL query statement from memory so as not to create a memory leak

SQLQueryStatement->Destroy();

delete SQLQueryStatement;

}

// Removes the database from memory so as not to create a memory leak.

UserDB->Close();

delete UserDB;

}

}

else if(password.Len() < 8)

{

UE\_LOG(LogTemp, Error, TEXT("Failed to create user: Password too short"));

result = TEXT("Password is too short");

}

else if (password != confirmedPass)

{

UE\_LOG(LogTemp, Error, TEXT("Failed to create user: Passwords don't match"));

result = TEXT("Passwords don't match");

}

else if (username.Len() == 0)

{

UE\_LOG(LogTemp, Error, TEXT("Failed to create user: Username is null"));

result = TEXT("Username can't be blank");

}

return result;

}

// Function to login users from details passed in. Checks to see if anyone in database matches entered details

FString ADatabaseManager::Login(FString username, FString password)

{

// Create an instance of FSHA256Hash

FSHA256Hash HashGenerator;

// Convert the string to hash

HashGenerator.FromString(password);

// Get the hashed string

FString HashedPass = HashGenerator.GetHash();

// Convert the string to hash

HashGenerator.FromString(username);

// Get the hashed string

FString HashedUsername = HashGenerator.GetHash();

// Defines the "DBFilePath" as the directory path to the database and defines "CreateUserSQL" as the SQL query to be run.

const FString DBFilePath = FPaths::ConvertRelativePathToFull(FPaths::ProjectContentDir() + "Database/UsersDB.db");

const FString LoginSQL = FString::Printf(TEXT("SELECT Username, Password FROM Users WHERE Username = '%s' AND Password = '%s';"), \*HashedUsername, \*HashedPass);

FString result = TEXT("null");

// If the Database opens correctly into "UserDB" so that it can be read and written to, then the following code is run.

if (FSQLiteDatabase\* UserDB = new FSQLiteDatabase();

UserDB->Open(\*DBFilePath, ESQLiteDatabaseOpenMode::ReadWrite))

{

// If the "SQLQueryStatement" is created correctly then it runs the following code

if (FSQLitePreparedStatement\* SQLQueryStatement = new FSQLitePreparedStatement();

SQLQueryStatement->Create(\*UserDB, \*LoginSQL, ESQLitePreparedStatementFlags::Persistent))

{

// Runs the SQL query and returns if it ran correctly or not

ESQLitePreparedStatementStepResult stepResult = SQLQueryStatement->Step();

// If the sql query ran correctly then sends the success message to the console. If it didn't run correctly then it would display the error message.

if (stepResult == ESQLitePreparedStatementStepResult::Row)

{

UE\_LOG(LogTemp, Log, TEXT("%s logged in successfully!"), \*username);

result = TEXT("Success");

}

else

{

UE\_LOG(LogTemp, Error, TEXT("Failed to login %s : %s"), \*username, \*UserDB->GetLastError());

result = TEXT("Incorrect Username or Password");

}

// Removes the SQL query statement from memory so as not to create a memory leak

SQLQueryStatement->Destroy();

delete SQLQueryStatement;

}

// Removes the database from memory so as not to create a memory leak.

UserDB->Close();

delete UserDB;

}

return result;

}

**DatabaseManager.h:**

// Name: Geronimo Jones SCN: 120763237

#pragma once

#include "CoreMinimal.h"

#include "GameFramework/Actor.h"

#include "DatabaseManager.generated.h"

/\*\*

\*

\*/

UCLASS()

class ADatabaseManager : public AActor

{

GENERATED\_BODY()

public:

// Sets default values for this actor's properties

ADatabaseManager();

UFUNCTION(BlueprintCallable, Category = "SQL Queries")

FString CreateUser(FString username, FString password, FString confirmedPass);

UFUNCTION(BlueprintCallable, Category = "SQL Queries")

FString Login(FString username, FString password);

};

// Name: Geronimo Jones SCN: 120763237

#include "HoverBike.h"

// Sets default values

AHoverBike::AHoverBike()

{

// Set this actor to call Tick() every frame. You can turn this off to improve performance if you don't need it.

PrimaryActorTick.bCanEverTick = false;

}

FString AHoverBike::GetName() const

{

return name; // Default top speed

}

float AHoverBike::GetTopSpeed() const

{

return topSpeed; // Default top speed

}

float AHoverBike::GetAcceleration() const

{

return acceleration; // Default top speed

}

float AHoverBike::GetHandling() const

{

return handling; // Default top speed

}

// Name: Geronimo Jones SCN: 120763237

#pragma once

#include "CoreMinimal.h"

#include "GameFramework/Actor.h"

#include "HoverBike.generated.h"

UCLASS()

class AHoverBike : public AActor

{

GENERATED\_BODY()

public:

// Sets default values for this actor's properties

AHoverBike();

UFUNCTION()

virtual FString GetName() const;

UFUNCTION()

virtual float GetTopSpeed() const;

UFUNCTION()

virtual float GetAcceleration() const;

UFUNCTION()

virtual float GetHandling() const;

protected:

UPROPERTY()

FString name;

UPROPERTY()

float topSpeed;

UPROPERTY()

float acceleration;

UPROPERTY()

float handling;

};

**MainMenuManager.cpp:**

// Name: Geronimo Jones SCN: 120763237

#include "MainMenuManager.h"

#include "Blueprint/UserWidget.h"

#include "Blueprint/WidgetBlueprintLibrary.h"

#include "MenuInteraction.h"

#include "Kismet/GameplayStatics.h"

// Sets default values

AMainMenuManager::AMainMenuManager()

{

// Set this actor to call Tick() every frame. You can turn this off to improve performance if you don't need it.

PrimaryActorTick.bCanEverTick = false;

}

// Called when the game starts or when spawned

void AMainMenuManager::BeginPlay()

{

Super::BeginPlay();

ChangeScreen(NextScreen);

}

void AMainMenuManager::ChangeScreen(TSubclassOf<UUserWidget> WidgetClass)

{

APlayerController\* PlayerControllerRef = UGameplayStatics::GetPlayerController(GetWorld(), 0);

PlayerControllerRef->SetShowMouseCursor(true);

// Make sure WidgetClass is set

if (WidgetClass)

{

// Get the first player controller from the world

APlayerController\* PlayerController = GetWorld()->GetFirstPlayerController();

// Create an instance of the widget

WidgetInstance = UWidgetBlueprintLibrary::Create(this, WidgetClass, PlayerController);

// Check if the instance is valid

if (WidgetInstance)

{

// Add the widget to the viewport

WidgetInstance->AddToViewport();

}

}

else{UE\_LOG(LogTemp, Error, TEXT("Widget class not true"))}

}

FString AMainMenuManager::GetUsername(){return Username;}

void AMainMenuManager::SetUsername(FString NewUsername){Username = NewUsername;}

UButton\* AMainMenuManager::GetChosenBike(){return ChosenBike;}

void AMainMenuManager::SetChosenBike(UButton\* NewChosenBike){ChosenBike = NewChosenBike;}

UTexture2D\* AMainMenuManager::GetChosenTrack(){return ChosenTrack;}

void AMainMenuManager::SetChosenTrack(UTexture2D\* NewChosenTrack){ChosenTrack = NewChosenTrack;}

**MainMenuManager.h:**

// Name: Geronimo Jones SCN: 120763237

#pragma once

#include "CoreMinimal.h"

#include "GameFramework/Actor.h"

#include "Components/Button.h"

#include "Blueprint/UserWidget.h"

#include "Blueprint/WidgetBlueprintLibrary.h"

#include "MainMenuManager.generated.h"

UCLASS()

class AMainMenuManager : public AActor

{

GENERATED\_BODY()

public:

// Sets default values for this actor's properties

AMainMenuManager();

UFUNCTION()

void ChangeScreen(TSubclassOf<UUserWidget> WidgetClass);

UFUNCTION()

FString GetUsername();

UFUNCTION()

void SetUsername(FString NewUsername);

UFUNCTION()

UButton\* GetChosenBike();

UFUNCTION()

void SetChosenBike(UButton\* NewChosenBike);

UFUNCTION()

UTexture2D\* GetChosenTrack();

UFUNCTION()

void SetChosenTrack(UTexture2D\* NewChosenTrack);

private:

// Called when the game starts or when spawned

virtual void BeginPlay() override;

UPROPERTY(EditDefaultsOnly, Category = "Widgets")

TSubclassOf<UUserWidget> NextScreen;

UUserWidget\* WidgetInstance;

UPROPERTY()

FString Username;

UPROPERTY()

UButton\* ChosenBike;

UPROPERTY()

UTexture2D\* ChosenTrack;

};

**MenuInteraction.cpp:**

// Name: Geronimo Jones SCN: 120763237

#include "MenuInteraction.h"

#include "MainMenuManager.h"

#include "DatabaseManager.h"

#include "Engine/World.h"

#include "Components/Button.h"

#include "Kismet/GameplayStatics.h"

#include "Components/EditableTextBox.h"

#include "Components/TextBlock.h"

#include "Components/Image.h"

#include "Components/PanelWidget.h"

#include "Components/HorizontalBox.h"

#include "Components/HorizontalBoxSlot.h"

#include "Blueprint/UserWidget.h"

#include "Blueprint/WidgetTree.h"

#include "AeroSwift.h"

#include "NimbusRider.h"

#include "SkyRunner.h"

void UMenuInteraction::NativeConstruct()

{

Super::NativeConstruct();

MainMenuManager = Cast<AMainMenuManager>(UGameplayStatics::GetActorOfClass(GetWorld(), AMainMenuManager::StaticClass()));

// Widget assignments

UsernameTextBox = Cast<UEditableTextBox>(GetWidgetFromName(TEXT("ETBUsername")));

PasswordTextBox = Cast<UEditableTextBox>(GetWidgetFromName(TEXT("ETBPassword")));

ConfirmPasswordTextBox = Cast<UEditableTextBox>(GetWidgetFromName(TEXT("ETBConfirmPassword")));

IncorrectPopUp = Cast<UTextBlock>(GetWidgetFromName(TEXT("TIncorrect")));

TArrayContainer = Cast<UHorizontalBox>(GetWidgetFromName(TEXT("BikeNameContainer")));

ImageContainer = Cast<UHorizontalBox>(GetWidgetFromName(TEXT("BikeImageContainer")));

// Sorting handling if ImageContainer exists

if (ImageContainer)

{

SortHandling();

}

// Set FinalUsername text

FinalUsername = Cast<UTextBlock>(GetWidgetFromName(TEXT("TFinalUsername")));

if (FinalUsername)

{

FinalUsername->SetText(FText::FromString(MainMenuManager->GetUsername()));

}

// Set FinalTrack image

FinalTrack = Cast<UImage>(GetWidgetFromName(TEXT("IFinalTrack")));

if (FinalTrack)

{

UTexture2D\* Texture = MainMenuManager->GetChosenTrack();

if (Texture)

{

FSlateBrush Brush;

Brush.SetResourceObject(Texture);

Brush.ImageSize.X = 500.0f;

Brush.ImageSize.Y = 500.0f;

FinalTrack->SetBrush(Brush);

FinalTrack->SetOpacity(1.0f);

FinalTrack->SetVisibility(ESlateVisibility::Visible);

}

else

{

UE\_LOG(LogTemp, Error, TEXT("Failed to load texture in Final track loading"));

}

}

// Set chosen bike button

ChosenContainer = Cast<UHorizontalBox>(GetWidgetFromName(TEXT("ChosenContainer")));

if (ChosenContainer)

{

UButton\* BFinalBike = MainMenuManager->GetChosenBike();

ChosenContainer->AddChild(BFinalBike);

UHorizontalBoxSlot\* ButtonSlot = Cast<UHorizontalBoxSlot>(BFinalBike->Slot);

if (ButtonSlot)

{

ButtonSlot->SetVerticalAlignment(EVerticalAlignment::VAlign\_Fill);

ButtonSlot->SetHorizontalAlignment(EHorizontalAlignment::HAlign\_Fill);

ButtonSlot->SetPadding(FMargin(0));

ButtonSlot->SetSize(ESlateSizeRule::Fill);

}

}

// Button event bindings

if (UButton\* BDB = Cast<UButton>(GetWidgetFromName(TEXT("BDB"))))

{

BDB->OnClicked.AddDynamic(this, &UMenuInteraction::QueryButtonHandler);

}

if (UButton\* BChangeScreen = Cast<UButton>(GetWidgetFromName(TEXT("BChangeScreen"))))

{

BChangeScreen->OnClicked.AddDynamic(this, &UMenuInteraction::ChangeScreenDefault);

}

if (UButton\* BSortSpeed = Cast<UButton>(GetWidgetFromName(TEXT("BSortSpeed"))))

{

BSortSpeed->OnClicked.AddDynamic(this, &UMenuInteraction::SortSpeed);

}

if (UButton\* BSortAcceleration = Cast<UButton>(GetWidgetFromName(TEXT("BSortAcceleration"))))

{

BSortAcceleration->OnClicked.AddDynamic(this, &UMenuInteraction::SortAcceleration);

}

if (UButton\* BSortHandling = Cast<UButton>(GetWidgetFromName(TEXT("BSortHandling"))))

{

BSortHandling->OnClicked.AddDynamic(this, &UMenuInteraction::SortHandling);

}

if (UButton\* BTrack1 = Cast<UButton>(GetWidgetFromName(TEXT("BTrack1"))))

{

BTrack1->OnClicked.AddDynamic(this, &UMenuInteraction::SelectTrack1);

}

if (UButton\* BTrack2 = Cast<UButton>(GetWidgetFromName(TEXT("BTrack2"))))

{

BTrack2->OnClicked.AddDynamic(this, &UMenuInteraction::SelectTrack2);

}

if (UButton\* BTrack3 = Cast<UButton>(GetWidgetFromName(TEXT("BTrack3"))))

{

BTrack3->OnClicked.AddDynamic(this, &UMenuInteraction::SelectTrack3);

}

}

void UMenuInteraction::ChangeScreenAndSetUsername(int mode, FString Username)

{

// Check if MainMenuManager is valid

if (!MainMenuManager)

{

// Log a warning if MainMenuManager is not found

UE\_LOG(LogTemp, Warning, TEXT("AMainMenuManager not found in the world!"));

return; // Return early to prevent potential crashes

}

// Check the mode to determine if we need to set the username

if (mode == 1)

{

// Set the provided username to MainMenuManager->Username

MainMenuManager->SetUsername(Username);

}

// Change the screen using MainMenuManager

MainMenuManager->ChangeScreen(NextScreen);

}

void UMenuInteraction::ChangeScreenDefault(){ ChangeScreenAndSetUsername(); }

void UMenuInteraction::SelectTrack1()

{

MainMenuManager->SetChosenTrack(Cast<UTexture2D>(StaticLoadObject(UTexture2D::StaticClass(), nullptr, TEXT("/Script/Engine.Texture2D'/Game/Images/track1.track1'"))));

ChangeScreenAndSetUsername();

}

void UMenuInteraction::SelectTrack2()

{

MainMenuManager->SetChosenTrack(Cast<UTexture2D>(StaticLoadObject(UTexture2D::StaticClass(), nullptr, TEXT("/Script/Engine.Texture2D'/Game/Images/track2.track2'"))));

ChangeScreenAndSetUsername();

}

void UMenuInteraction::SelectTrack3()

{

MainMenuManager->SetChosenTrack(Cast<UTexture2D>(StaticLoadObject(UTexture2D::StaticClass(), nullptr, TEXT("/Script/Engine.Texture2D'/Game/Images/track3.track3'"))));

ChangeScreenAndSetUsername();

}

void UMenuInteraction::SelectBike1()

{

MainMenuManager->SetChosenBike(CreateButtonWithImage("/Script/Engine.Texture2D'/Game/Images/IAeroSwift.IAeroSwift'"));

ChangeScreenAndSetUsername();

}

void UMenuInteraction::SelectBike2()

{

MainMenuManager->SetChosenBike(CreateButtonWithImage("/Script/Engine.Texture2D'/Game/Images/INimbusRider.INimbusRider'"));

ChangeScreenAndSetUsername();

}

void UMenuInteraction::SelectBike3()

{

MainMenuManager->SetChosenBike(CreateButtonWithImage("/Script/Engine.Texture2D'/Game/Images/ISkyRunner.ISkyRunner'"));

ChangeScreenAndSetUsername();

}

void UMenuInteraction::SortSpeed() { SortBikeAndSetLooks(1); }

void UMenuInteraction::SortAcceleration() { SortBikeAndSetLooks(2); }

void UMenuInteraction::SortHandling() { SortBikeAndSetLooks(3); }

void UMenuInteraction::SortBikeAndSetLooks(int SortingCriteria)

{

if (!ImageContainer || !TArrayContainer)

{

UE\_LOG(LogTemp, Error, TEXT("ImageContainer or TArrayContainer not found"));

return;

}

TArray<UButton\*> ButtonArray;

TArray<AHoverBike\*> HoverBikeArray;

// Create buttons and spawn hover bikes

ButtonArray.Add(CreateButtonWithImage("/Game/Images/IAeroSwift.IAeroSwift", 1));

ButtonArray.Add(CreateButtonWithImage("/Game/Images/INimbusRider.INimbusRider", 2));

ButtonArray.Add(CreateButtonWithImage("/Game/Images/ISkyRunner.ISkyRunner", 3));

HoverBikeArray.Add(Cast<AAeroSwift>(SpawnActor(AAeroSwift::StaticClass())));

HoverBikeArray.Add(Cast<ANimbusRider>(SpawnActor(ANimbusRider::StaticClass())));

HoverBikeArray.Add(Cast<ASkyRunner>(SpawnActor(ASkyRunner::StaticClass())));

// Sort based on the sorting criterion

int ArrayLen = HoverBikeArray.Num();

bool Swapped = true;

int StopIndex = ArrayLen;

while (Swapped && StopIndex >= 0)

{

Swapped = false;

for (int i = 0; i < (StopIndex - 1); i++)

{

float CurrentValue = 0.0f;

float NextValue = 0.0f;

switch (SortingCriteria)

{

case 1: // Sort by top speed

CurrentValue = HoverBikeArray[i]->GetTopSpeed();

NextValue = HoverBikeArray[i + 1]->GetTopSpeed();

break;

case 2: // Sort by acceleration

CurrentValue = HoverBikeArray[i]->GetAcceleration();

NextValue = HoverBikeArray[i + 1]->GetAcceleration();

break;

case 3: // Sort by handling

CurrentValue = HoverBikeArray[i]->GetHandling();

NextValue = HoverBikeArray[i + 1]->GetHandling();

break;

default: // Invalid criterion

UE\_LOG(LogTemp, Warning, TEXT("Invalid sorting criteria!"));

return;

}

if (CurrentValue < NextValue)

{

Swap(HoverBikeArray[i], HoverBikeArray[i + 1]);

Swap(ButtonArray[i], ButtonArray[i + 1]);

Swapped = true;

}

}

StopIndex -= 1;

}

TArrayContainer->ClearChildren();

// Create text blocks for each element in the HoverBikeArray and set their properties

for (int i = 0; i < ArrayLen; ++i)

{

FString ResultString = HoverBikeArray[i]->GetName(); // Get the name of the current HoverBike

HoverBikeArray[i]->Destroy();

UTextBlock\* TextBlock = WidgetTree->ConstructWidget<UTextBlock>(UTextBlock::StaticClass()); // Construct a new TextBlock widget

if (TextBlock)

{

// Set the text of the TextBlock to the name of the HoverBike

TextBlock->SetText(FText::FromString(ResultString));

// Set the color of the text

TextBlock->SetColorAndOpacity(FLinearColor::FromSRGBColor(FColor(0x2F, 0x3E, 0x46, 0xFF)));

// Set text justification to center

TextBlock->SetJustification(ETextJustify::Center);

// Add the TextBlock as a child to the TArrayContainer

TArrayContainer->AddChild(TextBlock);

// Set properties of the TextBlock's slot (positioning and sizing)

UHorizontalBoxSlot\* TextSlot = Cast<UHorizontalBoxSlot>(TextBlock->Slot);

if (TextSlot)

{

TextSlot->SetVerticalAlignment(EVerticalAlignment::VAlign\_Fill);

TextSlot->SetHorizontalAlignment(EHorizontalAlignment::HAlign\_Fill);

TextSlot->SetPadding(FMargin(0));

TextSlot->SetSize(ESlateSizeRule::Fill);

}

}

}

// Clear and repopulate the image container with sorted buttons

ImageContainer->ClearChildren(); // Clear any existing children of the ImageContainer

for (UButton\* Button : ButtonArray)

{

ImageContainer->AddChild(Button); // Add each button from ButtonArray to the ImageContainer

UHorizontalBoxSlot\* ButtonSlot = Cast<UHorizontalBoxSlot>(Button->Slot);

if (ButtonSlot)

{

// Set properties of the Button's slot (positioning and sizing)

ButtonSlot->SetVerticalAlignment(EVerticalAlignment::VAlign\_Fill);

ButtonSlot->SetHorizontalAlignment(EHorizontalAlignment::HAlign\_Fill);

ButtonSlot->SetPadding(FMargin(0));

ButtonSlot->SetSize(ESlateSizeRule::Fill);

}

}

}

void UMenuInteraction::QueryButtonHandler()

{

// Check if the required text boxes are valid

if (!UsernameTextBox || !PasswordTextBox)

{

// Log an error if text boxes are not found and return early

UE\_LOG(LogTemp, Error, TEXT("Text boxes not found"));

return;

}

// Retrieve text input from the text boxes

FString Username = UsernameTextBox->GetText().ToString();

FString Password = PasswordTextBox->GetText().ToString();

FString ConfirmPassword = ConfirmPasswordTextBox ? ConfirmPasswordTextBox->GetText().ToString() : FString();

// Attempt to spawn an instance of ADatabaseManager

ADatabaseManager\* DatabaseManager = Cast<ADatabaseManager>(SpawnActor(ADatabaseManager::StaticClass()));

if (!DatabaseManager)

{

// Log an error if ADatabaseManager fails to spawn and return early

UE\_LOG(LogTemp, Error, TEXT("ADatabaseManager not spawned successfully"));

return;

}

FString QueryResult;

// Perform login or user creation query based on the presence of ConfirmPassword

if (ConfirmPassword.IsEmpty())

{

QueryResult = DatabaseManager->Login(\*Username, \*Password);

}

else

{

QueryResult = DatabaseManager->CreateUser(\*Username, \*Password, \*ConfirmPassword);

}

// Check the query result

if (QueryResult == "Success")

{

// If the query was successful, change the screen and set the username

ChangeScreenAndSetUsername(1, \*Username);

}

else

{

// If the query failed, display an error message

IncorrectPopUp->SetText(FText::FromString(QueryResult));

IncorrectPopUp->SetVisibility(ESlateVisibility::Visible);

// Hide and then show the error message again after a short delay

if (IncorrectPopUp->GetVisibility() == ESlateVisibility::Visible)

{

IncorrectPopUp->SetVisibility(ESlateVisibility::Hidden);

FTimerHandle TimerHandle;

GetWorld()->GetTimerManager().SetTimer(TimerHandle, [&]() {IncorrectPopUp->SetVisibility(ESlateVisibility::Visible);}, 0.1f, false);

}

}

// Clean up by destroying the DatabaseManager instance

DatabaseManager->Destroy();

}

AActor\* UMenuInteraction::SpawnActor(TSubclassOf<AActor> ActorClass)

{

// Get the current world

UWorld\* World = GetWorld();

if (!World)

{

// Log an error if the world could not be found and return nullptr

UE\_LOG(LogTemp, Error, TEXT("World could not be found when spawning actor"));

return nullptr;

}

// Define the spawn parameters

FActorSpawnParameters SpawnParams;

SpawnParams.SpawnCollisionHandlingOverride = ESpawnActorCollisionHandlingMethod::AlwaysSpawn;

// Spawn and return the actor

return World->SpawnActor<AActor>(ActorClass, FVector::ZeroVector, FRotator::ZeroRotator, SpawnParams);

}

UButton\* UMenuInteraction::CreateButtonWithImage(FString Path, int bike)

{

UButton\* NewButton = WidgetTree->ConstructWidget<UButton>(UButton::StaticClass());

if (NewButton)

{

// Set button background color

NewButton->SetBackgroundColor(FLinearColor::Black);

UImage\* ButtonImage = WidgetTree->ConstructWidget<UImage>(UImage::StaticClass());

if (ButtonImage)

{

UTexture2D\* Texture = Cast<UTexture2D>(StaticLoadObject(UTexture2D::StaticClass(), NULL, \*Path));

if (Texture)

{

// Set brush image

FSlateBrush Brush;

Brush.SetResourceObject(Texture);

Brush.ImageSize.X = 500.0f;

Brush.ImageSize.Y = 500.0f;

ButtonImage->SetBrush(Brush);

ButtonImage->SetOpacity(1.0f);

// Set button image visibility

ButtonImage->SetVisibility(ESlateVisibility::Visible);

}

// Add image to button

NewButton->AddChild(ButtonImage);

// Set button slot properties

UHorizontalBoxSlot\* ButtonSlot = Cast<UHorizontalBoxSlot>(NewButton->Slot);

if (ButtonSlot)

{

ButtonSlot->SetVerticalAlignment(EVerticalAlignment::VAlign\_Fill);

ButtonSlot->SetHorizontalAlignment(EHorizontalAlignment::HAlign\_Fill);

ButtonSlot->SetPadding(FMargin(0));

}

}

if (bike == 1)

{

NewButton->OnClicked.AddDynamic(this, &UMenuInteraction::SelectBike1);

}

else if (bike == 2)

{

NewButton->OnClicked.AddDynamic(this, &UMenuInteraction::SelectBike2);

}

else if (bike == 3)

{

NewButton->OnClicked.AddDynamic(this, &UMenuInteraction::SelectBike3);

}

return NewButton;

}

else

{

return nullptr;

}

}

**MenuInteraction.h:**

// Name: Geronimo Jones SCN: 120763237

#pragma once

#include "CoreMinimal.h"

#include "MainMenuManager.h"

#include "Kismet/GameplayStatics.h"

#include "Blueprint/UserWidget.h"

#include "Components/EditableTextBox.h"

#include "Components/TextBlock.h"

#include "Components/PanelWidget.h"

#include "Components/Image.h"

#include "Components/HorizontalBox.h"

#include "MenuInteraction.generated.h"

/\*\*

\*

\*/

UCLASS()

class PLEASEPLEASEPLEASE\_API UMenuInteraction : public UUserWidget

{

GENERATED\_BODY()

public:

UFUNCTION()

AActor\* SpawnActor(TSubclassOf<AActor> ActorClass);

private:

virtual void NativeConstruct() override;

UFUNCTION()

void ChangeScreenAndSetUsername(int mode = 0, FString Username = "");

UFUNCTION()

void ChangeScreenDefault();

UFUNCTION()

void SelectTrack1();

UFUNCTION()

void SelectTrack2();

UFUNCTION()

void SelectTrack3();

UFUNCTION()

void SelectBike1();

UFUNCTION()

void SelectBike2();

UFUNCTION()

void SelectBike3();

UFUNCTION()

void SortSpeed();

UFUNCTION()

void SortAcceleration();

UFUNCTION()

void SortHandling();

UFUNCTION()

void SortBikeAndSetLooks(int SortingCriteria);

UFUNCTION()

void QueryButtonHandler();

UFUNCTION()

UButton\* CreateButtonWithImage(FString Path, int bike = 0);

UPROPERTY()

AMainMenuManager\* MainMenuManager;

UPROPERTY(EditDefaultsOnly, Category = "Widgets")

TSubclassOf<UUserWidget> NextScreen;

UUserWidget\* WidgetInstance;

UPROPERTY()

UEditableTextBox\* UsernameTextBox;

UPROPERTY()

UEditableTextBox\* PasswordTextBox;

UPROPERTY()

UEditableTextBox\* ConfirmPasswordTextBox;

UPROPERTY()

UTextBlock\* IncorrectPopUp;

UPROPERTY()

UTextBlock\* FinalUsername;

UPROPERTY()

UHorizontalBox\* TArrayContainer;

UPROPERTY()

UHorizontalBox\* ImageContainer;

UPROPERTY()

UHorizontalBox\* ChosenContainer;

UPROPERTY()

UImage\* FinalTrack;

};

**NimbusRider.cpp:**

// Name: Geronimo Jones SCN: 120763237

#include "NimbusRider.h"

// Sets default values

ANimbusRider::ANimbusRider()

{

// Set custom values for properties

name = "NimbusRider";

topSpeed = 1100.f;

acceleration = 1100.f;

handling = 800.f;

}

**NimbusRider.h:**

// Name: Geronimo Jones SCN: 120763237

#pragma once

#include "CoreMinimal.h"

#include "HoverBike.h"

#include "NimbusRider.generated.h"

/\*\*

\*

\*/

UCLASS()

class ANimbusRider : public AHoverBike

{

GENERATED\_BODY()

public:

// Sets default values for this hoverbike

ANimbusRider();

};

**SkyRunner.cpp:**

// Name: Geronimo Jones SCN: 120763237

#include "SkyRunner.h"

// Sets default values

ASkyRunner::ASkyRunner()

{

// Set custom values for properties

name = "SkyRunner";

topSpeed = 1500.f;

acceleration = 750.f;

handling = 750.f;

}

**SkyRunner.h:**

// Name: Geronimo Jones SCN: 120763237

#pragma once

#include "CoreMinimal.h"

#include "HoverBike.h"

#include "SkyRunner.generated.h"

/\*\*

\*

\*/

UCLASS()

class ASkyRunner : public AHoverBike

{

GENERATED\_BODY()

public:

// Sets default values for this hoverbike

ASkyRunner();

};

Testing

**End-User requirements:**

**Register for an account using a username and password & Have to enter their password twice so as to make sure they don’t make a typo:**

Registration screen:

A screenshot of a computer

Description automatically generated

User is successfully able to input a unique username and matching passwords of length at least 8. Meaning this requirement is met.

**Be told what is wrong with their username and password when an error occurs on account creation:**

I ran normal, extreme and exceptional test data to show what errors occur on each data

NOTE: Existing data is a user with Username = "Username123" and Password = "12345678" (Details are hashed in database)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type of Data | Data | Expected | Actual | Comment |
| Normal | NewUser, 123456789, 123456789 | Create User succesfully | Create User succesfully | Normal data so working as intended |
| Extreme | Username123 , 123456789, 123456789 | Create User successfully | Create User successfully | Space in username which counts as different username so working as intended |
| Extreme | NewUser, 12345678, 12345678 | Create User successfully | Create User successfully | Password length is on boundary but is 8 characters long so working as intended |
| Exceptional | Username123, 123456789, 123456789 | Username taken | Username taken | Username already exists in database so working as intended |
| Exceptional | NewUser, 1234567, 1234567 | Password is too short | Password is too short | Password is not 8 characters and shouldn't be allowed so working as intended |
| Exceptional | NewUser, FirstPassword, DifferentPassword | Passwords don't match | Passwords don't match | Passwords don't match, which isn't allowed so working as intended |

All data outputs the expected so this requirement is met.

**Login to their account using their username and password:**

Login Screen:

A screenshot of a computer login

Description automatically generated

User is successfully able to login to their account when they enter their correct details for their account. Meaning this requirement is met

**Be given an error when the username and password entered in login cannot be validated & Have minimum requirements of password creation so passwords are secure enough:**

I ran normal, extreme and exceptional test data to show what errors occur on each data

NOTE: Existing data is a user with Username = "Username123" and Password = "12345678" (Details are hashed in database)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type of Data | Data | Expected | Actual | Comment |
| Normal | Username123, 12345678 | Login Successful | Login Successful | Normal data works as intended |
| Extreme | Username123 , 12345678 | Incorrect login | Incorrect login | Note the space after the username. Spaces should alter what the input is so this is working as intended. |
| Extreme | Random, password | Incorrect login | Incorrect login | Username doesn't match so as intended |
| Exceptional | Username122, 12345678 | Incorrect login | Incorrect login | Username is very similar with same password. Result says details are incorrect so it's working as intended |
| Exceptional | Username123, 12345677 | Incorrect login | Incorrect login | Username is correct and password is 1 integer off, result says details are incorrect so it's working as intended |

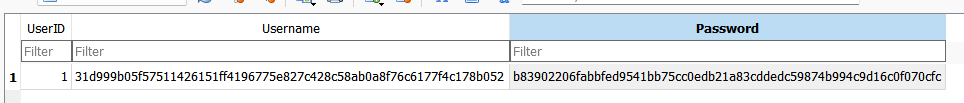
All data outputs the expected so this requirement is met.

**Have their details hashed so malicious people can’t steal their data from the database:**A screenshot of a computer

Description automatically generated

Password is: @4X7bvbgLmQgF3n8

Result:



Result shows that the details are hashed before they are stored in the database, meaning this requirement is met.

**Have different bikes to choose from:**

A screenshot of a computer

Description automatically generated

The user is presented with 3 bikes that they can click to choose. This requirement is met because the user can choose from the bikes.

**Be able to sort the bikes to show which are best in each aspect:**

A screenshot of a computer

Description automatically generated

The above screen shot shows the screen after pressing the "Top Speed" button, which sorts the objects by their top speed attribute, and displays the order in the image slots. This requirement is met because the bikes have been sorted by an attribute.

**Choose from different tracks:**

A screenshot of a computer

Description automatically generated

The user is presented with 3 tracks, that they can choose by clicking on them. This requirement is met because they can choose from different tracks.

**Have their options confirmed at the end of the menu:**

A screenshot of a computer

Description automatically generated

On the final screen the user is presented with their raw username, and the images of the track and bike they chose. This requirement is met because they can now be sure of what options they chose throughout the menu

**Navigate the menu through the use of buttons:**

On every menu screen, there is at least 1 button to progress further into the menu. This means the requirement is met because all navigation of the menu is done through the use of buttons. Evidence of this can be found in the implementation section.

**Have a button to go back to the main menu:**

As you can see from the screenshots in the implementation section, each screen apart from the main menu screen has a “Back” button that takes the user back to the main menu as requested. This means the requirement is met.

**See a consistent colour scheme throughout so as not to be confusing:**

Once again, the implementation screenshots show that all of the UMG screens have been styled in a mild green colour scheme. With recurring colours for button presses and text throughout the menu, and none of the persona raised the issue of a confusing colour scheme. Therefore, this requirement is met.

**Functional requirements SDD:**

**Handle on click events of buttons that lead to the execution of SQL queries & Retrieve values from editable text boxes & Hash username and password using a hashing plugin when creating an account and logging in:**

Code: Handles buttons that run the QueryButtonHandler:

**A screenshot of a computer code

Description automatically generated**

**Test**: Enter normal test data into the editable text boxes and click the register button. Then go to the login screen, enter the same details, and click the login button. I will be using the username “Test” and the password “Test12345”

**Expected results:**

DB:

New record in database

Log:

User created successfully!

Test logged in successfully!

**Results:**

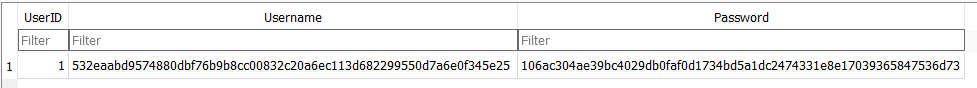
DB:

Database before clicking register button:

A screenshot of a computer

Description automatically generated

Database after clicking register button:



Log:

A screenshot of a computer

Description automatically generated

These results show that the on-click functions of the buttons work, and that the text is successfully being retrieved from the editable text boxes, hashed, and then put in the SQL query and executed.

**Validate that passwords meet a criteria during account creation:**

Refer to the registration full test data. This data shows that the passwords must meet all the criteria otherwise an error will be shown.

**Validate that username is unique on account creation:**

**Test:** Use a blank username with password “Test12345”.

**Expected results:**

Username can’t be blank popup appears

**Results:**

Before registration button clicked:

A screenshot of a login form

Description automatically generated

After registration button clicked:

A screenshot of a login form

Description automatically generated

Since program has an error pop up that says username can’t be blank, this requirement is met.

**Validate that record exists with matching username and password hashes during login:**

Refer to the login full test data. This data shows that if the username and password are anything except the exact same as the records, the details will not be validated. Therefore, this requirement is met.

**Set the text of an error message to the appropriate error during the failure to meet the validation on either login or registration screens:**

**Test:** Attempt to sign in using incorrect details on login screen using username “Test1” and password “Random”. Then Attempt to register for an account using a blank username and password/confirmed password “Test12345”. Then attempt to register with username “Test” (already taken) and password/confirmed password “Test12345”. Then attempt to register with username “Test2” and Password/confirmed password “1234”. Then attempt to register with username “Test2” and Password “Test12345” and confirmed password “Test54321”.

**Expected Results:** To be given the “Username or password incorrect” error, then “Username can’t be blank”, then “Username taken”, then “Password is too short” and finally “Passwords don’t match”.  
**Results:**

Incorrect username or password

A screenshot of a login screen

Description automatically generated

Blank username

A screenshot of a login form

Description automatically generated

Username taken

A screenshot of a login form

Description automatically generated

Password too short

A screenshot of a login form

Description automatically generated

Passwords don’t match

A screenshot of a login screen

Description automatically generated

All results matched the expected results. This means that this requirement is met.

**Have 3 hoverbike classes with unique values for properties:**

Code:

A screenshot of a computer code

Description automatically generated

A screenshot of a computer code

Description automatically generated

A screenshot of a computer code

Description automatically generated

This code shows the unique values of all 3 hoverbike objects. Given they have unique values, this requirement is met.

**Have a bubble sort using the values of each of the 3 hoverbike classes & Display an ordered list of hoverbikes on the screen corresponding to the bubble sort results:**

Code:

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

The first screenshot is bubble sort code that changed the sort values depending on the sort criteria. The second screenshot is displaying the button images in an order corresponding to the results of the bubble sort.

Test: Click all three buttons in turn.

Expected results: For top speed: SkyRunner, NimbusRider, AeroSwift. For Acceleration: NimbusRider, AeroSwift, SkyRunner. For Handling: AeroSwift, NimbusRider, SkyRunner.

Results:

Top Speed:

A screenshot of a video game

Description automatically generated

Acceleration:

A screenshot of a screenshot of a motorcycle

Description automatically generated

Handling:

A screenshot of a screenshot of a motorcycle

Description automatically generated

These results match the expected results. This shows that the bubble sort has been executed, that all 3 buttons work, and that the button images and text are being displayed in the correct order. This shows that these requirements are met.

**Have 3 bikes and tracks for the user to choose from & Store the user’s username,** **chosen bike and chosen track and show them their selections at the final screen:**

**Test:** Sign in using username “Test”, click on SkyRunner and track 2 to see if they are stored and displayed correctly.

**Expected results:** Chosen screen to display the username “Test”, image of SkyRunner (image with person in) and Image of track 2.

**Results:**

A screenshot of a computer

Description automatically generated

The Expected results match the actual results. This shows that the bikes and tracks can be chosen successfully and that the username is stored correctly, and that the chosen screen is displaying the right info. This means that both of these requirements are met.

**Have on-click functionality for buttons that change the screen to the next screen:**

Aside from the sort buttons, this code shows all the bindings for the buttons:

A screenshot of a computer program

Description automatically generated

As you have seen so far, the buttons that cause the screen to change work. This shows that this requirement is met.

**Have a button on all screens that takes the user back to the main menu:**

**Test:** Go to every screen and click the “Back” button

**Expected result:** The button will bring me back to the main menu every time it’s pressed.  
**Results:** The “Back” button on every screen brings me back to the main menu. Therefore, this requirement is met.

**Functional Requirements DDD:**

**Have UserID as a primary key:**

A white background with black text

Description automatically generated

Works as intended because UserID is the primary key.

**Have INSERT INTO query with 2 hashed values so that new users can be created:**

Results of validation and test data are in the registration section of comprehensive testing at the top of the testing section.

Code: 

Refer to the “Have their details hashed so malicious people can’t steal their data from the database” end-user requirement for testing and results of this query. The results show that this requirement is met.

**Have a SELECT query to check if there is a username and password that matches the details entered:**

Results of validation and test data are in the login section of comprehensive testing at the top of the testing section.

Code: 

Refer to the “Handle on click events of buttons that lead to the execution” end-user requirement for testing and results of this query. The results show that this requirement is met.

**Usability Testing:**

**Persona:** 17 year old computer science student:

**Task:** Sign Up for a new account, and sort the bikes and then select the bike and track

**Test result:** Persona Clicked register, clicked inside the editable text boxes and entered and acceptable username and passwords that were long enough and identical. Persona then sorted the bikes by acceleration, and then clicked the nimbus rider button. Persona then selected track 3 by clicking the picture.

**Persona comments:** "It's good yea, but I'd quite like to have a show password checkbox"

**Persona:** Elderly person, has eyesight issues.

**Task:** Sign into their account with Username "OldMan" and Password "Windows7IsTheBest". Then sort and select the bike with the best acceleration. Then go back to login screen

**Test result:** Persona navigates to the login screen with ease, this could be due to the large buttons with large text. Persona gets slightly confused when entering details, as the colour of the hint text is too similar to the colour of the editable text box. But the persona finds it after a few seconds. Persona manages to login correctly and is taken to the bike selection screen. Persona is slightly confused as it screen does not indicate which side is the best and which is the worst. But persona assumes its left-to-right, like reading order. Persona clicks nimbus rider which taken them to the track selection screen. Persona missed back button as it is slightly too small for their eyes. before clicking it and being taken back to the main menu. Persona then clicked login button to go back to login screen.

**Persona comments:** "I thought that the bike selection sorting could've had an arrow saying 'best to worst', so that I could tell more easily. I also expected the back button to take me back 1 screen, not back to the very start".

**Persona:** Millennial, who is neutrally proficient in computers, and regularly makes typos.

**Task:** Create and account and then go back to the main menu and sign in to that account

**Test result:** Persona clicks on the big Register button. They then enter details into the username, password and confirmed password fields. They click register to try and create their account but a pop up saying "Passwords don't match" persona then deletes their passwords and enters them again. This time when they click register, it brings them to the bike selection screen. They then click the back button, and get taken back to the main menu. From here they click login and enter their details in the username and password fields. Then clicked the login button to be presented with a "Username or password is incorrect" pop up. Persona then realised they had typed their username wrong, and retyped it. Persona then clicks login button again and is taken to the bike selection screen again.

**Persona comments:** “I type faster than I'm able to so I regularly make typos. It would've been nice to have a show password feature so that I can see where I made a mistake in the register screen instead of having to type the full thing all over again.”

**Persona:** 10 year old

**Task:** Create an account and select a bike and track

**Test result:** Persona clicked register to be taken to the registration screen. Persona then entered their username "EpicGamer999" and a password that was too short. They then click the register button to be given the (Password is too short) pop up. Persona then changed their first password to be longer without changing the second and clicks register again. They are given the "Passwords don't match" error now. They then change their second password and press register again, and was taken to the bike selection screen. They clicked on a bike and were taken to the track selection screen. They clicked on one of the tracks to be taken to the chosen screen, where they were presented with their username "EpicGamer999" and their chosen bike and track

**Persona comments:** “I kept getting confused on the register screen. But I eventually got through. I chose the bike and racetrack that were the coolest.”

**Testers comments:** I think that I should add a checkbox with an option to show the passwords as that's where the majority of my users problems were. I should also make the sorting process more intuitive and fool-proof.

**Component testing:**

**AAeroSwift::AAeroSwift(), ANimbusRider::ANimbusRider, ASkyRunner::ASkyRunner:**

**Test:** Click a sort button on the bike selection screen. This should call the constructors of all 3 classes. Including a temporary function that displays logs the name, topSpeed, acceleration and handling properties of each class.

**Expected result:**

AeroSwift created, parameters: 750.000000, 1000.000000, 1250.000000

NimbusRider created, parameters: 1100.000000, 1100.000000, 800.000000

SkyRunner created, parameters: 1500.000000, 750.000000, 750.000000

**Results:**

This is logged after the button is pressed

A screenshot of a computer

Description automatically generated

**AHoverBike::GetName():**

**Test:** Click the top speed sort button, I could click any button as the code that I am running does not depend on which button is pressed. In the SortBikeAndSetLooks function, the GetName function is called for every all 3 bike subclasses, so that these names can be displayed on the screen. I have added a temporary log UE\_LOG(LogTemp, Log, TEXT(“%s”), \*name); which logs the name of self (the instantiated child hoverbike class).

**Expected result:**

name is: SkyRunner

name is: NimbusRider

name is: AeroSwift

**Result:**

This is logged after the button is pressed

A screenshot of a computer

Description automatically generated

**AHoverBike::GetTopSpeed():**

**Test:** Click the top speed sort button. When the button is pressed, it executes a modified bubble sort, this bubble sort calls the GetTopSpeed function so that the values can be compared. I have added a temporary log UE\_LOG(LogTemp, Log, TEXT(“%f”), topSpeed); which logs the top speed of self (the instantiated child hoverbike class), every time the loop iterates.

**Expected result:**

NOTE: There is repeats because it calls the function for every sort.

topSpeed is: 750.000000

topSpeed is: 1100.000000

topSpeed is: 750.000000

topSpeed is: 1500.000000

topSpeed is: 1100.000000

topSpeed is: 1500.000000

**Result:**

This is logged after the button is pressed

A screenshot of a computer

Description automatically generated

**AHoverBike::GetAcceleration():**

**Test:** Click the acceleration sort button. When the button is pressed, it executes a modified bubble sort, this bubble sort calls the GetAcceleration function so that the values can be compared. I have added a temporary log UE\_LOG(LogTemp, Log, TEXT(“%f”), acceleration); which logs the acceleration of self (the instantiated child hoverbike class), every time the loop iterates.

**Expected result:**

NOTE: There is repeats because it calls the function for every sort.

acceleration is: 1000.000000

acceleration is: 1100.000000

acceleration is: 1000.000000

acceleration is: 750.000000

acceleration is: 1100.000000

acceleration is: 1000.000000

**Result:**

This is logged after the button is pressed

**A screenshot of a computer

Description automatically generated**

**AHoverBike::GetHandling():**

**Test:** Click the handling sort button. When the button is pressed, it executes a modified bubble sort, this bubble sort calls the GetHandling function so that the values can be compared. I have added a temporary log UE\_LOG(LogTemp, Log, TEXT(“%f”), handling); which logs the handling of self (the instantiated child hoverbike class), every time the loop iterates.

**Expected result:**

NOTE: There is repeats because it calls the function for every sort.

handling is: 1250.000000

handling is: 800.000000

handling is: 800.000000

handling is: 750.000000

**Result:**

This is logged after the button is pressed

A screenshot of a computer

Description automatically generated

**UMenuInteraction::ChangeScreenAndSetUsername:**

**Test:** I will enter the username “Test” and the password associated into the fields on the login screen. I will then click Login button, as it is bound to the QueryButtonHandler function, which calls the Login user query from the DatabaseManager class. If the query is executed correctly, then this function is called with parameters 1, and the username that was in the username field. In this situation, the Username in MainMenuManager is set. I have added the log UE\_LOG(Log, LogTemp, TEXT(“The username is now set to %s”), \*MainMenuManager->GetUsername()); after the username is set to show that the username is set correctly. I have also added the log UE\_LOG(LogTemp, Log, TEXT(“Called correctly”)); in the ChangeScreen function from the MainMenuManager class, to show that the function in called correctly.

**Expected result:**

The username is now set to Test

Called correctly

**Result:**

A screenshot of a computer

Description automatically generated

This shows that the username is set correctly, and that the ChangeScreen function is called correctly.

**UMenuInteraction::ChangeScreenDefault:**

**Test:** Click the login button from the main menu screen. I will do this because the login button is bound to this function. I have added the temporary log to the ChangeScreenAndSetUsername function: UE\_LOG(LogTemp, Log, TEXT(“Called correctly”)); So that when the button is pressed, this function calls the ChangeScreenAndSetUsername function, which outputs the log.

**Expected result:**

Called correctly

**Result:**

A screenshot of a phone

Description automatically generated

Since the log is outputted, the ChangeScreenAndSetUsername function is being called correctly meaning that his function is working.

**UMenuInteraction::SelectTrack1:**

**Test:** Click on track 1 on the Track selection screen. I have added the log UE\_LOG(LogTemp, Log, TEXT(“SetChosenTrack correctly”)); to the SetChosenTrack function, and I’ve added the log UE\_LOG(LogTemp, Log, TEXT(“ChangeScreenAndSetUsername Called correctly”)); to the ChangeScreenAndSetUsername function. This means that when the track1 button is clicked, this function is called, which then calls the SetChosenBike function (outputs a log) and the ChangeScreenAndSetUsername function (outputs a log).

**Expected result:**

GetChosenTrack Called correctly

ChangeScreenAndSetUsername Called correctly

**Results:**

A screenshot of a computer

Description automatically generated

**UMenuInteraction::SelectTrack2:**

**Test:** Click on track 2 on the Track selection screen. I have added the log UE\_LOG(LogTemp, Log, TEXT(“SetChosenTrack correctly”)); to the SetChosenTrack function, and I’ve added the log UE\_LOG(LogTemp, Log, TEXT(“ChangeScreenAndSetUsername Called correctly”)); to the ChangeScreenAndSetUsername function. This means that when the track2 button is clicked, this function is called, which then calls the SetChosenBike function (outputs a log) and the ChangeScreenAndSetUsername function (outputs a log).

**Expected result:**

GetChosenTrack Called correctly

ChangeScreenAndSetUsername Called correctly

**Results:**

A screen shot of a computer

Description automatically generated

**UMenuInteraction::SelectTrack3:**

**Test:** Click on track 3 on the Track selection screen. I have added the log UE\_LOG(LogTemp, Log, TEXT(“SetChosenTrack correctly”)); to the SetChosenTrack function, and I’ve added the log UE\_LOG(LogTemp, Log, TEXT(“ChangeScreenAndSetUsername Called correctly”)); to the ChangeScreenAndSetUsername function. This means that when the track3 button is clicked, this function is called, which then calls the SetChosenBike function (outputs a log) and the ChangeScreenAndSetUsername function (outputs a log).

**Expected result:**

GetChosenTrack Called correctly

ChangeScreenAndSetUsername Called correctly

**Results:**

A screenshot of a computer

Description automatically generated

**UMenuInteraction::SelectBike1:**

**Test:** Click on the AeroSwift bike on the bike selection screen. I have added the log UE\_LOG(LogTemp, Log, TEXT(“SetChosenBike correctly”)); to the SetChosenBike function, and I’ve added the log UE\_LOG(LogTemp, Log, TEXT(“ChangeScreenAndSetUsername Called correctly”)); to the ChangeScreenAndSetUsername function. This means that when the AeroSwift button is clicked, this function is called, which then calls the SetChosenBike function (outputs a log) and the ChangeScreenAndSetUsername function (outputs a log).

**Expected result:**

GetChosenBike Called correctly

ChangeScreenAndSetUsername Called correctly

**Results:**

A screenshot of a computer

Description automatically generated

**UMenuInteraction::SelectBike2:**

**Test:** Click on the NimbusRider bike on the bike selection screen. I have added the log UE\_LOG(LogTemp, Log, TEXT(“SetChosenBike correctly”)); to the SetChosenBike function, and I’ve added the log UE\_LOG(LogTemp, Log, TEXT(“ChangeScreenAndSetUsername Called correctly”)); to the ChangeScreenAndSetUsername function. This means that when this the NimbusRider button is clicked, this function is called, which then calls the SetChosenBike function (outputs a log) and the ChangeScreenAndSetUsername function (outputs a log).

**Expected result:**

GetChosenBike Called correctly

ChangeScreenAndSetUsername Called correctly

**Results:**

A screenshot of a computer

Description automatically generated

**UMenuInteraction::SelectBike3:**

**Test:** Click on the SkyRunner bike on the bike selection screen. I have added the log UE\_LOG(LogTemp, Log, TEXT(“SetChosenBike correctly”)); to the SetChosenBike function, and I’ve added the log UE\_LOG(LogTemp, Log, TEXT(“ChangeScreenAndSetUsername Called correctly”)); to the ChangeScreenAndSetUsername function. This means that when this the SkyRunner button is clicked, this function is called, which then calls the SetChosenBike function (outputs a log) and the ChangeScreenAndSetUsername function (outputs a log).

**Expected result:**

GetChosenBike Called correctly

ChangeScreenAndSetUsername Called correctly

**Results:**

A screenshot of a computer

Description automatically generated

**UMenuInteraction::SortSpeed:**

**Test:** Click the SortSpeed button on the bike selection menu. I put a log in the SortBikeAndSetLooks function that outputs “Called correctly”

**Expected results:**

Called correctly

**Results:**

A screenshot of a computer

Description automatically generated

**UMenuInteraction::SortAcceleration:**

**Test:** Click the SortAcceleration button on the bike selection menu. I put a log in the SortBikeAndSetLooks function that outputs “Called correctly”

**Expected results:**

Called correctly

**Results:**

A screenshot of a computer

Description automatically generated

**UMenuInteraction::SortHandling:**

**Test:** Click the SortHandling button on the bike selection menu. I put a log in the SortBikeAndSetLooks function that outputs “Called correctly”

**Expected results:**

Called correctly

**Results:**

A close-up of a computer screen

Description automatically generated

**UMenuInteraction::SortBikeAndSetLooks:**

**Test:** Click the sort by Top Speed button, then the sort by Acceleration button, then the sort by Handling button. All 3 of these button should call functions that call this function. I have added temporary logs to the code, one after populating the button array: if (ButtonArray.Num() == 3){UE\_LOG(LogTemp, Log, TEXT(“There are 3 in the button array”));} and one after populating the hoverbike array. : if (HoverBikeArray.Num() == 3){UE\_LOG(LogTemp, Log, TEXT(“There are 3 in the hoverbike array”));} The other outputs of this function are adding the buttons and text blocks to the screen, and setting their style.

**Expected results (After clicking first button):**

There are 3 in the button array

There are 3 in the hoverbike array

A screenshot of a computer screen

Description automatically generated

After clicking second button:

There are 3 in the button array

There are 3 in the hoverbike array

A screenshot of a computer screen

Description automatically generated

After clicking third button:

There are 3 in the button array

There are 3 in the hoverbike array

A screen shot of a computer screen

Description automatically generated

**Results (After clicking first button):**

A screenshot of a computer

Description automatically generated

A collage of a motorcycle

Description automatically generated

After clicking second button:

A screenshot of a computer

Description automatically generated

A collage of a motorcycle

Description automatically generated

After clicking third button:

A screenshot of a computer

Description automatically generated

A collage of a motorcycle

Description automatically generated

These results show that both of the arrays have been created and populated correctly, and that the order of them has been sorted by the bubble sort. These also show that my code affecting the style of the buttons is working, because the correct image is filling up the majority of each button and the horizontal panel container is taking the buttons at equal distances and giving them equal amounts of space. It also shows that the text is being sorted, coloured, centred and set to other styles correctly. This shows that this function is working as intended.

**UMenuInteraction::QueryButtonHandler:**

**Test 1:** I will enter “TestingUser” (a new unique username) in to the username box and “12345678” into the password and confirm password boxes, then I will click the Register button on the registration screen. That button should call this function which retrieves the values from the editable text boxes and convert them to strings. I have added a log after this that displays the string value of the text boxes to confirm that they have been retrieved correctly. There is already a log that shows if the database manager object is not instantiated correctly, so the absence of this log will show that the object has been instantiated correctly. The “User created successfully!” log is already in the DatabaseManager class, so that will confirm that the CreateUser function has been called correctly. For any of the error handling/ input validation testing, see the “Set the text of an error message to the appropriate error during the failure to meet the validation on either login or registration screens” functional requirement testing, as this shows the correct text being set for all errors. For calling the ChangeScreenAndSetUsername function, there is a temporary log “ChangeScreenAndSetUsername Called correctly”.

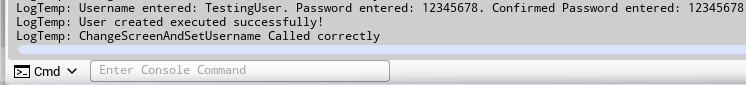
**Expected results:**

Username entered: TestingUser. Password entered: 12345678. Confirmed Password entered: 12345678

User created executed successfully!

ChangeScreenAndSetUsername Called correctly

**Actual results:**



This shows that the data has been extracted from the editable text boxes successfully, that the CreateUser function is called successfully, and that the ChangeScreenAndSetUsername function is called correctly.

**Test 2:** I will enter “TestingUser” (the same username before as the account is already created) in to the username box and “12345678” into the password box, then I will click the Login button on the login screen. That button should call this function which retrieves the values from the editable text boxes and convert them to strings. I have added a log after this that displays the string value of the text boxes to confirm that they have been retrieved correctly. There is already a log that shows if the database manager object is not instantiated correctly, so the absence of this log will show that the object has been instantiated correctly. The “%s logged in successfully!” (%s takes in the username as a parameter) log is already in the DatabaseManager class, so that will confirm that the Login function has been called correctly. For any of the error handling/ input validation testing, see the “Set the text of an error message to the appropriate error during the failure to meet the validation on either login or registration screens” functional requirement testing, as this shows the correct text being set if an error occurs. For calling the ChangeScreenAndSetUsername function, there is a temporary log “ChangeScreenAndSetUsername Called correctly”.

**Expected results:**

Username entered: TestingUser. Password entered: 12345678.

TestingUser logged in successfully!

ChangeScreenAndSetUsername Called correctly

**Actual results:**

A screenshot of a computer

Description automatically generated

This shows that the data has been extracted from the editable text boxes successfully, that the Login function is called successfully, and that the ChangeScreenAndSetUsername function is called correctly.

**UMenuInteraction::SpawnActor:**

**Test:** Click the login button with the same values as the above test in the text boxes. This will call the above function, which calls this function with the DatabaseManager class as the parameter. This function will then call the temporary log “SpawnActor called successfully”, to show it’s been called correctly. The function will then return the actor’s instance, if the QueryButtonHandler function then checks that the actor is spawned correctly. If not, the log “ADatabaseManager not spawned successfully” will occur.

**Expected results:**

SpawnActor called successfully

**Results:**

A screenshot of a computer

Description automatically generated

This shows that the function is called and no errors occur, meaning the actor is spawned and the function works as intended.

**UMenuInteraction::CreateButtonWithImage:**

**Test:** Open the bike selection screen and click AeroSwift, then reopen the screen and do the same for NimbusRider and SkyRunner. When the bike selection screen is opened. The SortBikeAndSetLooks function is called, which in turn calls this function three times to create the 3 different buttons. These button are displayed to the screen with the assigned styles. Clicking on each of the bikes will test the button OnClick dynamics, to make sure they work. I have added the temporary logs “(name) selected” where (name) is the respective name of each bike, to the select bike functions, to show they are being called.

**Expected results:**

Bike selection screen:

A screen shot of a computer screen

Description automatically generated

Logs after opening screen:

CreateButtonWithImage called successfully.

CreateButtonWithImage called successfully.

CreateButtonWithImage called successfully.

Log after clicking (NOTE: CreateButtonWithImage is called again to create the button instance that is stored in the MainMenuManager class):

AeroSwift:

AeroSwift selected

CreateButtonWithImage called successfully.

NimbusRider:

NimbusRider selected

CreateButtonWithImage called successfully.

SkyRunner:

SkyRunner selected

CreateButtonWithImage called successfully.

**Results:**  
Bike selection screen:

A screenshot of a video game

Description automatically generated

Logs after opening screen:

A screenshot of a computer

Description automatically generated

Log after clicking (NOTE: CreateButtonWithImage is called again to create the button instance that is stored in the MainMenuManager class):

AeroSwift:

A screenshot of a computer

Description automatically generated

NimbusRider:

A screenshot of a computer

Description automatically generated

SkyRunner:

A screen shot of a computer

Description automatically generated

These results show that the CreateButtonWithImage function gets called, and correctly adds the styles to each button. These results also show that the OnClick dynamics work for each button made.

**AMainMenuManager::ChangeScreen:**

**Test:** Click the login button on the main menu screen (any would do apart from the sort buttons, but it’s the easiest to get to). That should call ChangeScreenDefault which calls ChangeScreenAndSetUsername, which calls this function. I’ve added a “ChangeScreen called correctly” log to this function so that we can ensure the functions have been called correctly.

**Expected result:**

Log:

ChangeScreen called correctly

UI:

Screen changes from main menu to Login screen.

**Results:**

Log:

A screen shot of a computer

Description automatically generated

Before button click:

A screenshot of a menu

Description automatically generated

After button click:

A screenshot of a login screen

Description automatically generated

These results show that the function is called correctly, and then the screen changes to the relevant screen, showing that this function works.

**Getters and setters in the MainMenuManager class:**

**Test:** Enter the username and password “Test” and “12345678” at the login screen (valid details), then click the NimbusRider bike at the bike selection, and finally click track 1 at the track selection screen. The chosen screen should show the results from the GetUser, GetChosenBike and GetChosenTrack functions. I’ve also added logs to each of these functions that show they’ve been called correctly.

**Expected results:**

Log:

SetUsername called

SetChosenBike called

SetChosenTrack called

GetUsername called

GetChosenBike called

GetChosenTrack called

UI:

The username “Test” in the first box, with a picture of track 3 in the second box and a picture of NimbusRider in the third box.

**Results:**

Log:

A screenshot of a computer

Description automatically generated

UI:

A screenshot of a video game

Description automatically generated

This shows that all functions were called and that they set and returned the correct values, otherwise the boxes would not contain the values that were clicked/entered by the username.

**Integrative testing:**

**ADatabaseManager::CreateUser (INSERT INTO query integration):**

**Test:** Fill the editable text boxes on the register screen with values (Full test of input validation / error handling at top of testing section), then click the Register button. This will call the function that calls this function. There is already a log that says “User created successfully!” that comes up if the query is executed correctly. I will compare the database before and after the test.

**Expected results:**

Log:

User created successfully!

Database:

A new record in the database with hashed values.

**Results:**

Log:

A screenshot of a computer

Description automatically generated

Database before:

A screenshot of a computer

Description automatically generated

Database after:

A screenshot of a computer

Description automatically generated

This shows that the function and query are executed successfully, and that the hashed data is entered into the database.

**ADatabaseManager::Login (SELECT query integration):**

**Test:** Fill the editable text boxes on the login screen with username “Test” and password “Test12345” (valid details) (Full test of input validation / error handling at top of testing section), then click the Login button. This will call the function that calls this function. There is already a log that says “(username) logged in successfully” where (username) is the username the user entered, that comes up if the query is executed correctly, and that a record is found with details the same as the hash of what the user enters.

**Expected results:**

Test logged in successfully!

**Results:**

A screen shot of a computer

Description automatically generated

This shows that the function and query are executed successfully, and that the query found details that matched the hash of the ones entered.

Evaluation

**Fitness for purpose:**

My testing has shown that all the functional requirements are met. Usability testing has shown that the program would benefit from adding a checkbox to show the password fields. My program could also benefit from arrows showing which side the higher values are for bike sorting. Other than these details, my program meets all the specified requirements as it allows a user to create their own unique username and password which are hashed and stored in a database called Users.db with UserID as a primary key. Or login to the database by hashing the details and passing them into a SELECT query to check if a user with those details exists. The user is then able to select from and sort an array of hoverbike objects, represented on screen by their pictures. The user can also choose from a variety of tracks, finishing in a confirmation of their username, chosen bike, and chosen track. While being able to go back to the main menu at any time.

* End-User Requirements
  + Register for an account using a username and password
  + Be told what is wrong with their username and password when an error occurs on account creation
  + Login to their account using their username and password
  + Be given an error when the username and password entered in login cannot be validated
  + Have to enter their password twice so as to make sure they don’t make a typo during registration
  + Have their details hashed so malicious people can’t steal their data from the database
  + Have different bikes to choose from
  + Be able to sort the bikes to show which are best in each aspect
  + Choose from different tracks
  + Have their options confirmed at the end of the menu
  + Navigate the menu through the use of buttons
  + Have a button to go back to the main menu
  + See a consistent colour scheme throughout so as not to be confusing
  + Have minimum requirements of password creation so passwords are secure enough
* Functional requirements SDD:
  + Handle on click events of buttons that lead to the execution of SQL queries
  + Retrieve values from editable text boxes
  + Validate that passwords meet a criteria during account creation
  + Validate that username is unique on account creation
  + Validate that record exists with matching username and password hashes during login
  + Set the text of an error message to the appropriate error during the failure to meet the validation on either login or registration screens.
  + Hash username and password using a hashing plugin when creating an account and logging in.
  + Have 3 hoverbike classes with unique values for properties.
  + Have a bubble sort using the values of each of the 3 hoverbike classes.
  + Display an ordered list of hoverbikes on the screen corresponding to the bubble sort results
  + Have 3 bikes and tracks for the user to choose from.
  + Store the user’s username, chosen bike and chosen track and show them their selections at the final screen.
  + Have on-click functionality for buttons that change the screen to the next screen.
  + Have a button on all screens that takes the user back to the main menu.
* Functional requirements DDD:
  + Have UserID as a primary key
  + Have INSERT INTO query with 2 hashed values so that new users can be created.
  + Have a SELECT query to check if there is a hashed username and password that matches the hashed values of the username and password entered by the user.

These criteria have been checked according to the testing I have completed.

**Maintainability:**

Perfective: For improvement of solutions functionality, I can add the UI details mentioned (Arrows to indicate sorting order, and a checkbox to show the text in the password box for usability). Other than this, my program has no memory leaks that I’m aware of, and I’ve attempted to make my program as efficient as possible, though I believe there could be some improvements (more on that below).

Corrective: There are no errors in my code of which I am aware. If there are any that I have not found, they do not affect the requirements or effectiveness of the end result.

Adaptive: If a developer is to come to my code, they will find there is useful internal commentary and use of whitespace throughout the program to improve readability (as pictured below). If a new version of unreal engine releases it should be somewhat easy to migrate to the newer version, but this does not matter as it ins not mandatory to do so to keep the functionality.

A screenshot of a computer program

Description automatically generated

This example shows my se of internal commentary and whitespace helps with the readability for someone with less experience in unreal engine c++.

**Robustness:**

My program is extremely robust as it can handle all sorts of inputs. It validates all the inputs it takes against the database or minimum criteria. My program is even protected against simple SQL injection such as putting a '; after a username to finish the query and login using just a username. It is safe against this because the username is hashed before being placed inside the prepared statement. My program is also safe against malicious people who steal the database data as they will not be able to reverse the hash to access any of the usernames or passwords.

This is an example of validation I have used to make sure people are entering correct details for creating an account:

if ((password.Len() >= 8) && (confirmedPass == password) && (username.Len() > 0))

As you can see it checks that the password is of correct length, the passwords match, and that the username cannot be left blank. If any of these aren’t met, it returns the appropriate error.

Another method I’ve used to make my program robust is the interactivity for sorting. Having just 3 buttons for the user to choose from restricts their choice extremely narrowly so they cannot attempt to sort by a non-existent attribute. I believe that the only vulnerability of my code would by malicious users using high level SQL injection that block my hash plugin.

**Usability:**

My usability testing has shown that persona/end users think that the UI is very usable apart from the visual queues of the sorting functions, and the lack of checkbox to make the password visible to the user. I have also had a UX student look at my program and they believe it is quite usable.

**Accuracy of output:**

I will prove that my hash function is actually hashing:

This is a screenshot of my database after entering the password “ThisIsATestPasswordForAccuracy” with the username “TestingAccuracyName”.

A screenshot of a computer

Description automatically generated

This is the results from an online SHA256 hash website:

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

These show that the hashes are hashing correctly. For the other accuracy of output features, the INSERT INTO query works flawlessly, and the SELECT function also works perfectly, without ever giving a successful output with incorrect information or vice versa. And I have demonstrated that the texts from the editable text boxes are retrieved correctly. Therefore, my output is exactly accurate.