Development and Testing

Table of Contents

[Chapter Three: Development and Testing 2](file:///H:\Project\Report\Louis_Pattern_development_and_testing.docx#_Toc125545062)

[3.1 Introduction 3](#_Toc125545063)

[3.2.1 Stage 1: Building the Error Handling 4](#_Toc125545064)

[3.2.3.1 Code for Error handling section 4](#_Toc125545065)

[3.2.1.3 Feedback from Stakeholder 9](#_Toc125545066)

[3.2.1.4 Review 9](#_Toc125545067)

[3.2.1 Stage 1: Building the Database 9](#_Toc125545068)

[3.2.1.1 SQLITE + Python Code 9](#_Toc125545069)

[3.2.1.2 Testing Databases 10](#_Toc125545070)

[3.1.3 Feedback from Stakeholder (if appropriate) 10](#_Toc125545071)

[3.1.4 Review 10](#_Toc125545072)

[3.2.2 Stage 2: Building Log in 10](#_Toc125545073)

[3.2.2.1 The Code for Log in section (including validation) 10](#_Toc125545074)

[3.2.2.2 Testing Log in section (including validation) 11](#_Toc125545075)

[3.2.2.3 Feedback from Stakeholder (if appropriate) 11](#_Toc125545076)

[3.2.2.4 Review 11](#_Toc125545077)

[3.2.3 Stage 3: Building …….. PART THREE …. 11](#_Toc125545078)

[3.2.3.1 Code for …Part THREE … including validation 11](#_Toc125545079)

[3.2.3.2 Testing …. Part THREE ….including validation 11](#_Toc125545080)

[3.2.3.3 Feedback from Stakeholder (if appropriate) 11](#_Toc125545081)

[3.2.3.4 Review 11](#_Toc125545082)

[3.2.3 Stage 4: Building …….. PART FOUR …. 12](#_Toc125545083)

[3.2.3.1 Code For … part FOUR… (including validation) 12](#_Toc125545084)

[3.2.3.2 Testing …. Part FOUR ….including validation 12](#_Toc125545085)

[3.2.3.3 Feedback from Stakeholder (if appropriate) 12](#_Toc125545086)

[3.2.3.4 Review 12](#_Toc125545087)

[3.2.4 Stage 2: Building …….. PART FIVE …. 13](#_Toc125545088)

[3.2.4.1 Code for … Part FIVE… including validation 13](#_Toc125545089)

[3.2.4.2 Testing …. Part FIVE ….including validation 13](#_Toc125545090)

[3.2.4.3 Feedback from Stakeholder (if appropriate) 13](#_Toc125545091)

[3.2.4.4 Review 13](#_Toc125545092)

[3.3 Final Review, Improvements and Corrective Actions 13](#_Toc125545093)

# Chapter Three: Development and Testing

Development and Testing

## 3.1 Introduction

When programming my project, I will use a modular approach. This means I will use the fact that I have decomposed the project into many smaller sections and I will first test each of these sections or modules separately. In the code, each module can import other modules, when necessary, in order to be able to run a function from another module. This approach allows for reusability of code and improves the readability and ease of editing the code.

I will use sqlite3 in order to build by database. This will mean the database is stored locally on one machine rather than on an online server. This makes it more suitable for a low number of users or even a single user. Because my project is single-player and is not an online game, it is not important that the database can only be accessed from one device.

I have decided to use Python 3.1, because it is the programming language that I feel the most comfortable with. There is also a wide range of libraries available, some of which will be used frequently in my game.

I will use Tkinter for the login part of the project as well as displaying popup messages. This allows me to create a user-friendly, visually appealing interface that the user will use to log in as well as displaying error messages.

For creating my game, I have decided to use Pygame. This will allow me to display the GUI for the space game as well as displaying the game’s menus such as the highscore menu. Pygame is suited for my project because it allows for creation of sprites that are flexible and can be displayed on screen when needed. It also allows the player to input and control their character in the game using the keyboard.

I will use the bcrypt library which will allow me to hash and salt the passwords of users that are stored in the database. This will improve the security of the system.

When testing the project, I will use unittest in order to do automatic testing rather than manual testing. This will improve the efficiency when testing and allow modules to be tested quicker.

##### Modules

|  |  |  |  |
| --- | --- | --- | --- |
| Module name | Purpose | Relevant Design section | Development section |
| Admincontrol.py | Provides the UI for an admin to create or delete users |  |  |
| colour\_changer.py | Changing the colour hue of an image |  |  |
| dates.py | Handles getting the dates needed for storing a score |  |  |
| game.py | Main game module, runs the game GUI |  |  |
| HighscoresData.py | Connects to the database in order to read or edit scores |  |  |
| login.py | Provides the UI for a user to log in |  |  |
| LoginData.py | Connects to the database in order to read or edit user details |  |  |
| main\_app.py (Top-level module) | Starts the initial login window |  |  |
| messages.py | Displays Tkinter popup messages for error handling |  |  |
| settings.py | Manages editing games settings that are saved to a text file |  |  |
| sprites.py | Contains all sprite classes that are used in game.py |  |  |
| testing.py | Runs unit tests on each module |  |  |
| validation.py | Used anytime data must be validated | N/A | N/A |

## 3.2.1 Stage 1: Building the Error Handling

### 3.2.1.1 Code for Error handling section

I have developed the Error handling section according to my design in 2.5.4 – Error Handling.  
I created a module called messages.py. This contains all the code of this section and will only be used for displaying error messages.

##### Imported modules

1. **import** tkinter **as** tk
2. **from** tkinter **import** messagebox

I have used Tkinter Messagebox to display the windows in this section.

##### Prototype 1

Show\_message procedure:

1. **def** show\_message(title, message):
2. root = tk.Tk()
3. **messagebox.showinfo(title=title, message=message)**
4. root.destroy()

I first built this show\_message procedure, which displays a message window using Tkinter and creating a window called root. The Tkinter.messagebox module is then used to display an information box with the title as the value of the parameter ‘title’ and a message as the parameter ‘message’. When OK is clicked, the message box will be closed and the parent window will be destroyed with .destroy(). I will first test this prototype before continuing to develop this module.

###### Prototype 1 Testing:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Test data** | **Expected outcome** | **Comment** |
| 1.1 | Attempt to display a message | Run show\_message() procedure with title = “Test title” and message = “Test message” | Message box with title “Test title” and message “Test message” displayed | Failed |

###### Prototype 1 Results:

Test 1.1 failed because, while the message was displayed correctly, the Tkinter parent window was still visible alongside the message box. This should not be the case.



##### Prototype 2

Show\_message procedure (fixed):

1. **def** show\_message(title, message, opt):
2. root = tk.Tk()
3. root.withdraw() *# Hides tk window immediately*
4. messagebox.showinfo(title=title, message=message)
5. **root.destroy()**

In order to fix the bug found in test 1.1, I have added code that withdraws the blank Tkinter window before the message box is displayed. This should prevent the window from appearing and now only the message box should be displayed.

###### Prototype 2 Testing:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Test data** | **Expected outcome** | **Comment** |
| 1.1 | Attempt to display a message | Run show\_message() procedure with title = “Test title” and message = “Test message” | Message box with title “Test title” and message “Test message” displayed | Success |
| 1.2 | Ok button clicked in the message box | Clicking on ‘OK’ | Message box closes | Success |

###### Prototype 2 Results:

Test 1.1 was successful as shown in this evidence:



Test 1.2 was successful as when the ‘OK’ button was clicked the message box window closed and the parent Tkinter window was never visible.

##### Prototype 3

Show\_message procedure (improved)

1. *# Options: 1-show info, 2-show error, 3-show warning, 4-ask y/n*
2. **def** show\_message(title, message, opt):
3. root = tk.Tk()
4. root.withdraw() *# Hides tk window immediately*
5. **if opt == 1:**
6. messagebox.showinfo(title=title, message=message)
7. **elif** opt == 2:
8. messagebox.showerror(title=title, message=message)
9. **elif** opt == 3:
10. **messagebox.showwarning(title=title, message=message)**
11. **else**:
12. **return** messagebox.askyesno(title=title, message=message)
13. root.destroy()

With the base show\_message() procedure now working as expected, I decided to add the feature to display different kinds of messages depending on option selected using the value of the ‘opt’ parameter.

###### Prototype 3 Testing:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Test data** | **Expected outcome** | **Comment** |
| 1.3 | Attempt to display a show info box. | Run show\_message() with title = “Info title” and message = “Info message”. opt = 1 | Info box displayed with title “Info title and message “Info message” | Success |
| 1.4 | Attempt to display an error box | Run show\_message() with title = “Error title” and message = “Error message”. opt = 2 | Error message displayed with title “Error title” and message “Error message” | Success |
| 1.5 | Attempt to display warning message box | Run show\_message() with title = “Warning title” and message = “Warning message”. opt = 3 | Warning message displayed with title “Warning title” and message “Warning message” | Success |
| 1.6 | Attempt to display question box | Run show\_message() with title = “Question title” and message = “Question message”. opt = 4 | Question box displayed with title “Question title” and message “Question message” | Success |
| 1.7 | Attempt to click “Yes” on question box | Clicking on the “Yes” button | Message box closes and returns True | Success |
| 1.8 | Attempt to click “No” on question box | Clicking on the “No” button | Message box closes and returns False | Success |

###### Prototype 3 Results:

Test 1.3:



Test 1.4:



Test 1.5:



Test 1.6:



Test 1.7:



Test 1.8:



### 3.2.1.2 Feedback from Stakeholder

I have showed prototype 3 of the message boxes to Kurk Milo. I went through each type of window (info, error, warning, question) as in testing. He agreed that the messages worked well and particularly liked the inclusion of different types of message boxes. He was satisfied with overall how the message boxes function and didn’t mention the addition of another feature.

After testing Prototype 3 and finding no bugs as well as getting positive feedback from my stakeholder, I can mark the development of the error handling section as complete and move onto developing the next module of my project, the database.

## 3.2.3 Stage 2: Building the Database

### 3.2.2.1 SQLITE + Python Code

I have built the database according to my design in section 2.4 – Database Design. The name of the file that contains the code for this module is LoginData.py.

##### Imported modules

1. **import** sqlite3
2. **import** validation
3. **from** messages **import** \*

I have used sqlite3 to store my database. This means that the database will be stored locally on the machine. I will also use the messages module so an error message can be displayed in case there is an error that occurs with the database.

##### Prototype 1

Create\_table() function:

1. **def** create\_table():
2. **try**:
3. con = sqlite3.connect("LoginScores.db")
4. con.execute('''CREATE TABLE IF NOT EXISTS Users
5. **(Username VARCHAR PRIMARY KEY NOT NULL,**
6. Password TEXT NOT NULL);''')
8. con.execute('''CREATE TABLE IF NOT EXISTS Admins
9. (Username VARCHAR PRIMARY KEY NOT NULL,
10. **Password TEXT NOT NULL);''')**
11. con.commit()
12. con.close()
13. messages.show\_message("Success", "Database created successfully.", 1)
14. **except** Exception **as** ex:
15. **messages.show\_message("Error", ex, 2)**

This function connects to the database and creates the Users and Admins tables if they don’t already exist. The Username is the primary key field for both tables because each user must have a unique username.

###### Prototype 1 Testing:

I tested this code by running the function. As expected, the success message appeared on screen and the database file (LoginScores.db) was created.



##### Prototype 2

Enter\_user() function:

1. *# Function that inserts a user (username + password) into the database*
2. **def** enter\_user(u, p):
3. con = sqlite3.connect("LoginScores.db")
4. **try**:
5. **con.execute('''insert into Users (Username, Password) values (?, ?)''',**
6. (u, p))
7. con.commit()
8. con.close()
9. show\_message("Success", "Successfully added user: " + u, 1)
10. **return True**
11. except Exception as ex:
12. show\_message("Error creating user", ex, 2)
13. con.close()
14. **return** False

The enter\_user() function takes a username and password as parameters and inserts them into the Users table of the database. This function will be used by Admins to add new users.

Search() function:

1. **def** search(u, p, table):
2. con = sqlite3.connect("LoginScores.db")
3. cursor = con.cursor()
5. **if table == "Admins":**
6. cursor.execute("SELECT \* FROM Admins")
7. **else**:
8. cursor.execute("SELECT \* FROM Users")
9. records = cursor.fetchall()
10. **found = False**
11. **for** row **in** records:
12. **if** row[0] == u **and** row[1] == p:
13. found = True
14. cursor.close()
15. **con.close()**
16. **return** found

In the login section, I will need to check if the details a user enters matches those in the database. This will be done using this search() function. This takes the username, password, and the desired table to conduct the search and returns True if it finds a matching record in that table.

###### Prototype 2 Testing:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Test data** | **Expected outcome** | **Comment** |
| 2.1 | Attempt to create a user with valid details using enter\_user() function. | U = “test\_user1”  P = “testpassword01” | User is entered successfully to the Users table. | Success |
| 2.2 | Attempt to create a user with borderline details | U = “test2”  P = “Password2” | User is entered successfully to the Users table. | Success |
| 2.3 | Attempt to create a user with invalid details. | U = “t3”  P = “inval” | User should not be added to the database. An error message should appear. | Failed |
| 2.3.2 |  | U = 10  P = 5 | Wrong data type, an error message should appear. | Failed |
| 2.4 | Attempt to search for an existent user with search() function. | U = “test\_user1”  P = “testpassword01” | True returned | Success |
| 2.5 | Attempt to search for correct username but incorrect password. | U = “test\_user1”  P = “incorrectpass” | False returned | Success |
| 2.6 | Attempt to search for non-existent user but existent password. | U = “incorrectuser”  P = “testpassword01” | False returned | Success |
| 2.7 | Attempt to search for non-existent user | U = “incorrectuser” P = “incorrectpass” | False returned | Success |

###### Prototype 2 Results:

Test 2.1:



Test 2.2:



Adding a new user using the enter\_user() function was successful, but using invalid details also added them to the table when they should be rejected. I will improve this function by adding validation before entering the data.

In order to test the search function, I used the unittest module because it allows for more efficient, automatic testing which allows for a range of data to be tested quickly.

##### Prototype 3

**Functions from validation module:**

Is\_inrange() function:

1. *# range validation*
2. **def** is\_inrange(data, lo, hi):
3. **try**:
4. **if** (len(data) >= lo) **and** (len(data) <= hi):
5. **return True**
6. **return** False
7. **except** ValueError:
8. **return** "Error"

This function will be used to check if the length of the data is within range specified by the parameters ‘lo’ and ‘hi’. This will be used mainly to validate username and password length.

Is\_valid\_user() function:

1. *# username and password validation*
2. *# u - data to be validated, opt - validation option: username or password*
3. **def** is\_valid\_user(u, opt):
4. **if** opt == "username":
5. **if isinstance(u, str):**
6. **if** is\_inrange(u, 3, 20):
7. u = u.upper()
8. characters = []
9. **for** i **in** range(65, 91):
10. **characters.append(chr(i))**
11. **for** i **in** range(48, 58):
12. characters.append(chr(i))
13. characters.append("\_")
14. valid = True
15. **for n in range(len(u)):**
16. **if** **not** u[n] **in** characters:
17. valid = False
18. **else**:
19. valid = False
20. **else:**
21. valid = False
22. **else**:
23. valid = False
24. **if** isinstance(u, str) **and** is\_inrange(u, 8, 255):
25. **valid = True**
26. **return** valid

**Functions from LoginData module:**

Enter\_user() function (improved):

1. **def** enter\_user(u, p):
2. *# Validation performed on username and password before they are entered*
3. val\_u = validation.is\_valid\_user(u, "username")
4. val\_p = validation.is\_valid\_user(p, "password")
5. **if val\_u and val\_p:**
6. **try**:
7. con = sqlite3.connect("LoginScores.db")
8. con.execute('''insert into Users (Username, Password) values (?, ?)''',
9. (u, p))
10. **con.commit()**
11. con.close()
12. show\_message("Success", "Successfully added user: " + u, 1)
13. **return** True
14. **except** Exception **as** ex:
15. **show\_message("Error creating user", ex, 2)**
16. con.close()
17. **return** False
18. **else**:
19. show\_message("Error creating user", "Invalid", 2)
20. **return** False

This improved enter user function now validates the user details before inserting them into the database. The username must be between 3 and 20 characters and the password must be greater than or equal to 8 characters. The username must also only be composed of valid characters (upper and lowercase letters as well as underscores and numbers). Both username and password must be strings.

Delete\_user() procedure:

1. **def** delete\_user(u):
2. **try**:
3. con = sqlite3.connect("LoginScores.db")
4. cursor = con.cursor()
5. ***# Deleting single record***
6. sql = "DELETE FROM Users WHERE Username=?"
7. cursor.execute(sql, (u,))
8. con.commit()
9. cursor.close()
10. **con.close()**
11. show\_message("Success", "User deleted ", 1)
13. **except** sqlite3.Error **as** error:
14. *# Displays popup message*
15. **show\_message("Error", "Failed to delete record from sqlite table: " + str(error), 2)**

I have added a procedure to delete a user record from the table. This will be used later whenever an admin needs to remove a user from the database.

###### Prototype 3 Testing:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Test data** | **Expected outcome** | **Comment** |
| 2.8 | Attempt to create a user with invalid details with enter\_user() function | U = “t8”  P = “inval” | User should not be added to the database, error message. | Success |
| 2.9 | Attempt to create a user with invalid details. | U = 5  P = 10 | Error message | Success |
|  |  |  |  |  |

###### Prototype 3 Results:

2.8, 2.9:



### 3.2.3 Feedback from Stakeholder (if appropriate)

### 3.2.4 Review

With the main functions for the data developed, I will review the database.



Using [sqlite viewer](https://inloop.github.io/sqlite-viewer/) I can see the usernames and passwords stored in the db. Here, the password visible in plain text and has not been hashed or salted. This is a problem and compromises the security of the login system as well as not fitting the specifications that the system must hash user’s passwords.  
This means I must fix my functions to incorporate the hashing of passwords in the database.

Enter\_user function (fixed)

1. **def** enter\_user(u, p):
2. *# Validation performed on username and password before they are entered*
3. val\_u = validation.is\_valid\_user(u, "username")
4. **val\_p = validation.is\_valid\_user(p, "password")**
5. **if** val\_u **and** val\_p:
6. con = sqlite3.connect("LoginScores.db")
7. *# Convert password to bytes, hash and salt it*
8. p = bytes(p, encoding='utf-8')
9. **hash\_p = bcrypt.hashpw(p, bcrypt.gensalt())**
10. **try**:
11. con.execute('''insert into Users (Username, Password) values (?, ?)''',
12. (u, hash\_p))
13. con.commit()
14. **con.close()**
15. **return** True
16. **except** Exception **as** ex:
17. show\_message("Error creating user", ex, 2)
18. con.close()
19. **return False**
20. **else**:
21. show\_message("Error creating user", "Invalid", 2)
22. **return** False

Search function (fixed):

1. **def** search(u, p, table):
2. con = sqlite3.connect("LoginScores.db")
3. cursor = con.cursor()
4. p = bytes(p, encoding='utf-8')
6. **if** table == "Admins":
7. cursor.execute("SELECT \* FROM Admins")
8. **else**:
9. cursor.execute("SELECT \* FROM Users")
10. **records = cursor.fetchall()**
11. found = False
12. **for** row **in** records:
13. **if** row[0] == u **and** bcrypt.checkpw(p, row[1]):
14. found = True
15. **cursor.close()**
16. con.close()
17. **return** found

## 3.2.3 Stage 3: Building Login

### 3.2.3.1 Code for Login (including validation)

### 3.2.3.2 Testing …. Part THREE ….including validation

### 3.2.3.3 Feedback from Stakeholder (if appropriate)

### 3.2.3.4 Review

## 3.2.3 Stage 4: Building …….. PART FOUR ….

### 3.2.3.1 Code For … part FOUR… (including validation)

### 3.2.3.2 Testing …. Part FOUR ….including validation

### 3.2.3.3 Feedback from Stakeholder (if appropriate)

### 3.2.3.4 Review

## 3.2.4 Stage 2: Building …….. PART FIVE ….

### 3.2.4.1 Code for … Part FIVE… including validation

### 3.2.4.2 Testing …. Part FIVE ….including validation

### 3.2.4.3 Feedback from Stakeholder (if appropriate)

### 3.2.4.4 Review

## 3.3 Final Review, Improvements and Corrective Actions