CHAPTER 5
TESTING AND IMPLEMENTATION

5.1 Hardware Specification

The author’s proposed solution will be implemented in forms of APIs, game features, database and web application. During the development process, the author was using devices in the following list:

- Computer
  - OS: Windows 10
  - Memory: 8GB DDR4 RAM
  - Processor: Intel i5 9400 2.4GHz 64bit
  - Storage: 250GB SSD
- Oculus Rift
  - Video Card: NVIDIA GTX 1060 / AMD Radeon RX 480
  - CPU: Intel i5-4590 / AMD Ryzen 5 1500X
  - Memory: 8GB RAM
  - Video Output: Compatible HDMI 1.3
  - USB Ports: 3x USB 3.0 ports plus 1x USB 2.0 port

5.2 Software Specification

In addition to the hardware specification, the software technologies that are used during the development consist of:

- Visual Studio Code (v1.58) to build the user interface.
- Postman (v7. 33.0) to test out the APIs.
- Google Chrome as the browser to test the application.
- phpMyAdmin (v8.0) to build the database.
- Unity (v2020.1) to build some of the game features.
- Microsoft Visual Studio (Dev16 v16.0) to build the game features system.
- Google Charts to show the database in charts.

5.3 Website Testing (API testing)

This testing includes the features that is present on the web application like login, logout, register, edit/delete profile of the user and the admin. The charts there are present on several pages are also tested to make sure that the data
viewed in the chart matches the data in the database. The web application also follows the 8 golden rules for better user experience and interface.

Figure 5.1. Shows the home screen.

This is the default page for the potential admin of the web. By clicking the login button, the user will be redirected to the login page.

Figure 5.2. Shows the logging in screen.

By clicking the login button, the user will be redirected to this page where the user will have to input the registered email and password (in which only the registered admin has registered).
Figure 5.3. Shows if the inputted data do not match.

If the user inputted the wrong email or password, a pop up will appear telling the user that the inputted email or password is invalid. The user will stuck in this page until they have put in the right input. This follows the 3<sup>rd</sup> and 5<sup>th</sup> golden rules which is the to offer informative feedback and simple error handling in which users just need to retype their email and password when the invalid message pops up. This testing also includes the API testing where the inputted email and password are compared to the database, users are only allowed to access the web application if the inputted emails and password exist in the database.

Figure 5.4. Shows the default page after logging in.
After the admin has successfully logged in, the user will be redirected into this page where the data of the game users are shown in the form of chart. The admin can also manage the amount of admin by clicking the admin bar in the right side of the page (navbar). The navbar follows the 2\textsuperscript{nd} golden rules which is to enable users to have frequent shortcuts to access the page they want. In this page, the API are also tested to display the charts, the charts are the overall information of the game players stored in the database.

![Image of admin panel](image)

**Figure 5.5.** Shows the button to log out or manage admin.

![Image of confirm logout](image)

**Figure 5.6.** Shows the pop up to confirm log out.

By clicking the button present in the admin profile (upper top navbar), the admin can either manage the amount of user or simply log out. The 3\textsuperscript{rd} and 5\textsuperscript{th} golden rules are implemented in this to prevent users from accidental actions and unwanted confusions from those accidental actions (mis-clicks).
By clicking manage profile (top navbar) or admin (left navbar), the admin will be redirected to the admin profile page where the admin could add another admin, edit the admin profile or delete the admin. Here, the 1st and 4th golden rules are implemented as the page for admin management page and players management page are the same and the pagination is included to make users know what page they are in.
If the user clicked on the add admin profile button the user will receive a pop up in which if the information like username, email, password and confirm password are saved, a new admin profile will be added alongside a “Admin Profile Added” text will appear. This is where the 1st, 3rd, 4th, 6th and 7th golden rules are implemented. The API in this page is also tested to make sure that the new admin are also made in the database.

![EDIT Admin Profile](image)

**Figure 5.10.** Shows the add admin page.

![Your Data is Updated](image)

**Figure 5.11.** Shows your data is updated text.

If the user clicked on the edit button, the user will be redirected to a page in which if the information like username, email and password are updated, the admin profile will be updated alongside a “Your Data is Updated” text will appear. This is where the 1st, 3rd, 4th, 6th and 7th golden rules are implemented. The API in this page is also tested to make sure that the updated data in the web application are also updated in the database.
Figure 5.12. Shows when the admin has been deleted.

If the user clicked on the delete button, the admin profile will be deleted alongside a “Your Data is Deleted” text will appear. This is where the 1\textsuperscript{st}, 3\textsuperscript{rd} and 5\textsuperscript{th} golden rules are implemented. The API in this page is also tested to make sure that the deleted data in the web application are also deleted in the database.

5.3.1 Charts & Tables

![Line Chart](image)

Figure 5.13. Shows the line chart.

The line chart shows the average accuracy over time (date time) of all the player that has played Brushed Away. APIs are used in order to display this chart as the data are taken from database.
Figure 5.14. Shows the scatter chart.

The scatter chart shows the average accuracy over time taken of all the player that has played Brushed Away. APIs are used in order to display this chart as the data are taken from database.

Figure 5.15. Shows the pie chart.

The pie chart shows the average gender of all the player that has played Brushed Away. If the player played as guest, their gender will be set to others. APIs are used in order to display this chart as the data are taken from database.
Figure 5.16. Shows the data tables of the game user.

The tables that are used by the administrator to manage, edit and delete all of the players that has played Brushed Away. While if the admin clicked on delete, the player’s information will be deleted permanently. Here, the 1st and 4th golden rules are implemented as the page for admin management page and players management page are the same and the pagination is included to make users know what page they are in.

Figure 5.17. Shows the edit player profile page.
Just like the admin profile page, when the admin clicked on the edit button the admin will be redirected to edit player information page and if its saved, the player’s information will be updated. This is where the 1st, 3rd, 4th, 6th and 7th golden rules are implemented.

A message will appear in the player’s information table page depends on what action the admin does (edit/delete). This is where the 1st, 3rd and 5th golden rules are implemented. The API in this page is also tested to make sure that the updated/deleted data in the web application are also updated/deleted in the database.

5.4 Game Feature Testing (API & Feature Testing)

For this testing, the author is only responsible in the testing of the API. The testing is to ensure that the data of the players are stored in the database. The GET method of the data is also ensured so that the data can be fetch to Unity when the player is logging into the game.

5.4.1 Player As Guest

Figure 5.19. Shows the player choosing to play as guest.
When choosing to play as guest, the player may choose whether they want to play as Guest 1, Guest 2 or Guest 3. Choosing a different type of guest does not affect the game in any sort of way. The type of the guest chosen by the player will also be stored into the database alongside their playing data. The API in this feature is already tested as the guest information is stored in the database.

5.4.2 Player Registering

![Player Registering](image)

Figure 5.21. Shows the player registering.

<table>
<thead>
<tr>
<th>First Name</th>
<th>Last Name</th>
<th>Email</th>
<th>Password</th>
<th>Gender</th>
</tr>
</thead>
<tbody>
<tr>
<td>pop</td>
<td>lol</td>
<td><a href="mailto:poplol@gmail.com">poplol@gmail.com</a></td>
<td>poplol</td>
<td>Male</td>
</tr>
</tbody>
</table>

Table 5.22. Shows the player registered in the database.

![Offline Text File](image)

Figure 5.23. Shows offline text file.
If the player does not want to play as guest and is willing to register, the player will be able to register by clicking register at the home page. The player will have to input data such as their first name, last name, email/username, password and gender. The player’s information will be stored into the database so that the player could log in the next time they want to play. The API in this feature is tested as the inputted information are successfully stored in the database.

5.4.3 Player Logging In

![Login Screen]

Figure 5.24. Shows the player choosing to log in.

![Database Text File]

Figure 5.25. Shows database text file.

If the player has already registered before, they can log into their existing account. If the logging in player inputted the wrong data or has not registered yet the player will not be able to log into the game.
5.4.4 Player Logging In

Figure 5.26. Shows the player choosing to log in.

Figure 5.27. Shows game text file.

Once the player has already finished the level, the player’s playing game record like time taken, accuracy and level will be stored in the database.

5.4.5 Offline Storing

This feature is made to ensure that there will not be any data loss of the player’s information when the device suddenly gets disconnected with the internet. The data will be stored in the form of text file which will be present in Unity. In this case, 2 text files will be created, each with different purpose.

5.4.5.1 Offline.txt

From figure 5.23, this text file is created to store every data of the player, the purpose of this text file is to prevent the data loss if the device suddenly gets disconnected with the internet. When the device gets connected to the internet, Unity will read the data present in the text file and POST these data to the database.
5.4.5.2 Database.txt

From figure 5.25, this text file is created for the GET method, Unity will be reading the data present in the database and write these data into this text file. The purpose of this text file is for the log in function inside the game. When logging in the inputted username and password will be compared to the data in this text file, if the said username and password match the data from the database the player will proceed to playing the game.

5.4.5.3 Game.txt

From figure 5.27, this text file is created to store every data of the player’s game playing record. Just like offline.txt, the purpose of this text file is to prevent the data loss if the device suddenly gets disconnected with the internet. When the device gets connected to the internet, Unity will read the data present in the text file and POST these data to the database.

5.4.6 API Testing

These are the list of APIs that are used in connecting Unity to the database, several of the APIs are used in the features of the game and the API used for system of the administrator.

5.4.6.1 Connecting To Database / Register

```
string url = "http://localhost/AdminPhp/admin/apicrudphp/Unity/api/categories/create.php";
Form form = new Form();
form.AddField("f_name", firstname);
form.AddField("l_name", lastname);
form.AddField("username", signuppuser);
form.AddField("password", signuppword);
form.AddField("gender", gender);
using(UnityWebRequest www = UnityWebRequest.Post(url, form))
{

```

Figure 5.28. Shows the code of getting the register user API.

The code shown above is the code used in sending the player’s information from Unity directly to the database by using the POST method during register.
Figure 5.29. Shows the testing of getting the register user API.

The picture shown above is the testing result of the API that is used in sending player’s information from Unity to database. The test is done by using Postman.

Figure 5.30. Shows the function of sending data of game from Unity to the database.

The code shown above is the code used in sending the game information from Unity directly to the database by using the POST method when they are finished playing each level of the game.
5.4.6.2 Log In (User)

Figure 5.31. Shows the function of log in (in Unity).

The code shown above is the code used in comparing the player’s information from Unity with the text file (also in Unity) by using the POST method during log in. If the inputted username and password do not match the data that are registered the user will not be able to proceed.

Figure 5.32. Shows the testing of getting the log in user API.

The picture shown above is the testing result of the API that is used in comparing player’s information from Unity to database. The test is done by using Postman.

5.4.6.3 Log In (Administrator)

These are the API used in the log in, add, delete and edit of the administrator which can also edit and delete the data of the players from the database.
Figure 5.33. Shows the PHP code of the login API (admin).

The figure above shows the code of the API that is used when the admin is trying to log in from the home page into the admin dashboard. By comparing the data stored in the database with the input, if the input does not match then the user will not be able to go into the admin dashboard.

```php
if(isset($_POST['login_btn']))
{
    $email_login = $_POST['email'];
    $password_login = $_POST['password'];
    $query = "SELECT * FROM register WHERE email='$email_login' AND password='$password_login' LIMIT 1";
    $query_run = mysqli_query($connection, $query);
    if(mysqli_fetch_array($query_run))
    {
        $_SESSION['username'] = $email_login;
        header('location: index.php');
    }
    else
    {
        $_SESSION['status'] = "Email / Password is Invalid";
        header('location: login.php');
    }
}
```

Figure 5.34. Shows the PHP code of the register API (admin).

The figure above shows the code of the API that is used when the admin is making a new admin profile in the admin profile page. After the new admin is made, the new admin will now be able to log in.

```php
if(isset($_POST['registerbtn']))
{
    $username = $_POST['username'];
    $email = $_POST['email'];
    $password = $_POST['password'];
    $confirm_password = $_POST['confirm_password'];
    if($password === $confirm_password)
    {
        $query = "INSERT INTO register (username, email, password) VALUES ('$username', '$email', '$password');";
        $query_run = mysqli_query($connection, $query);
        if($query_run)
        {
            $_SESSION['success'] = "Admin Profile Added";
            header('location: register.php');
        }
        else
        {
            $_SESSION['status'] = "Admin Profile NOT Added";
            header('location: register.php');
        }
    }
    else
    {
        $_SESSION['status'] = "Password and Confirm Password Does Not Match";
        header('location: register.php');
    }
```
Figure 5.35. Shows the PHP code of the edit profile API (admin).

The figure above shows the code of the API that is used when the admin is editing the existing admin profile. This also directly affects the data from the database.

Figure 5.36. Shows the PHP code of the delete profile API (admin).

The figure above shows the code of the API that is used when the admin is deleting the existing admin profile. The deleted admin profile will now not be able to log in nor has access to the admin dashboard.
The figure above shows the code of the API that is used when the admin is wants to log out. When the admin has logged out the user will be redirected to the home page and needs to log in again if they want to access the admin dashboard. This also ensures the admin profile that is currently being used to access the admin dashboard.

The figure above shows the code of the API that is used when the admin is editing the existing player profile. This also directly affects the data from the database.
Figure 5.39. Shows the PHP code of the delete profile API (player).

The figure above shows the code of the API that is used when the admin is deleting the existing player profile. The deleted player profile will be permanently deleted from the database, affecting the tables and charts.

5.5 API Documentation

APIs that are available for both user and administrator roles are shown in Tables below.

<table>
<thead>
<tr>
<th>HTTP METHOD</th>
<th>API PATH</th>
<th>DESCRIPTION</th>
<th>RETURN</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>/Unity/api/categories/create</td>
<td>Allow player to create an account for Brushed Away</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>{ “user”: {</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;player_id&quot;:&lt;int&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;l_name&quot;:&lt;string&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;f_name&quot;:&lt;string&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;username&quot;:&lt;string&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;password&quot;:&lt;string&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;birth&quot;:&lt;string&gt;</td>
<td></td>
</tr>
<tr>
<td>GET</td>
<td>/Unity/api/categories/list</td>
<td>Retrieves player’s information from the database to be stored in a text file in Unity</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>{ “user”: {</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;player_id&quot;:&lt;int&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;l_name&quot;:&lt;string&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;f_name&quot;:&lt;string&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;username&quot;:&lt;string&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;password&quot;:&lt;string&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;birth&quot;:&lt;string&gt;</td>
<td></td>
</tr>
<tr>
<td>POST</td>
<td>/aplcrudphp/api/categories/login</td>
<td>Post the data to the database comparing the inputted username and password in the database</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>{ “user”: {</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;username&quot;:&lt;string&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;password&quot;:&lt;string&gt;</td>
<td></td>
</tr>
<tr>
<td>GET</td>
<td>/aplcrudphp/api/categories/loginread</td>
<td>Retrieving the database to be compared with the input inside the Unity</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>{ “user”: {</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;username&quot;:&lt;string&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;password&quot;:&lt;string&gt;</td>
<td></td>
</tr>
<tr>
<td>Method</td>
<td>Route</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>POST</td>
<td>/Game/api/categories/create</td>
<td>To store the data of the game after the player has finished each level into the database</td>
<td></td>
</tr>
<tr>
<td>GET</td>
<td>/Game/api/categories/list</td>
<td>To store the data from the database into the txt file in Unity</td>
<td></td>
</tr>
</tbody>
</table>

Tables 5.1. User API design.
### Tables 5.2. Administrator API design.

<table>
<thead>
<tr>
<th>Method</th>
<th>Endpoint</th>
<th>Description</th>
<th>Request Structure</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>/AdminPHP/admin/code</td>
<td>To register new admin in the website</td>
<td>`{ &quot;register&quot;: { &quot;username&quot;: &quot;&lt;string&gt;&quot;, &quot;email&quot;: &quot;&lt;string&gt;&quot;, &quot;password&quot;: &quot;&lt;string&gt;&quot; } }</td>
<td></td>
</tr>
<tr>
<td>POST</td>
<td>/AdminPHP/admin/code</td>
<td>To edit the administrator profile</td>
<td>`{ &quot;register&quot;: { &quot;username&quot;: &quot;&lt;string&gt;&quot;, &quot;email&quot;: &quot;&lt;string&gt;&quot;, &quot;password&quot;: &quot;&lt;string&gt;&quot; } }</td>
<td></td>
</tr>
<tr>
<td>POST</td>
<td>/AdminPHP/admin/code</td>
<td>To delete administrator</td>
<td>`{ &quot;register&quot;: { &quot;id&quot;: &quot;&lt;string&gt;&quot; } }</td>
<td></td>
</tr>
<tr>
<td>POST</td>
<td>/AdminPHP/admin/login</td>
<td>Used for administrator login</td>
<td>`{ &quot;register&quot;: { &quot;email&quot;: &quot;&lt;string&gt;&quot;, &quot;password&quot;: &quot;&lt;string&gt;&quot; } }</td>
<td></td>
</tr>
<tr>
<td>POST</td>
<td>/AdminPHP/admin/logout</td>
<td>Used for administrator login</td>
<td>`{ &quot;register&quot;: { &quot;password&quot;: &quot;&lt;string&gt;&quot; } }</td>
<td></td>
</tr>
</tbody>
</table>

### Tables 5.3. User administrator API design.

<table>
<thead>
<tr>
<th>Method</th>
<th>Endpoint</th>
<th>Description</th>
<th>Request Structure</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>/AdminPHP/admin/code</td>
<td>To edit the player profile</td>
<td>`{ &quot;user&quot;: { &quot;player_id&quot;: &quot;&lt;int&gt;&quot;, &quot;l_name&quot;: &quot;&lt;string&gt;&quot;, &quot;m_name&quot;: &quot;&lt;string&gt;&quot;, &quot;username&quot;: &quot;&lt;string&gt;&quot;, &quot;password&quot;: &quot;&lt;string&gt;&quot;, &quot;birth&quot;: &quot;&lt;string&gt;&quot; } }</td>
<td></td>
</tr>
<tr>
<td>POST</td>
<td>/AdminPHP/admin/code</td>
<td>To delete user</td>
<td>`{ &quot;user&quot;: { &quot;id&quot;: &quot;&lt;string&gt;&quot; } }</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/AdminPHP/admin/chart</td>
<td>To view user's data in charts</td>
<td>`{ &quot;data&quot;: { &quot;game&quot;: { &quot;game_id&quot;: &quot;&lt;int&gt;&quot;, &quot;player_id&quot;: &quot;&lt;int&gt;&quot;, &quot;level&quot;: &quot;&lt;string&gt;&quot;, &quot;url&quot;: &quot;&lt;string&gt;&quot;, &quot;url&quot;: &quot;&lt;string&gt;&quot;, &quot;user&quot;: &quot;&lt;string&gt;&quot;, &quot;title&quot;: &quot;&lt;string&gt;&quot;, &quot;film&quot;: &quot;&lt;string&gt;&quot;, &quot;film&quot;: &quot;&lt;string&gt;&quot;, &quot;accuracy&quot;: &quot;&lt;string&gt;&quot; } }</td>
<td></td>
</tr>
</tbody>
</table>