Tampere University of Applied Sciences

Integration of EOSSDK for game achievements with Epic Games Store

Dat Pham

BACHELOR’S THESIS
November 2023

Bachelor's Degree Program in Software Engineering

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Bachelor’s Degree Program in Software Engineering

Dat Pham:
Integration of EOSSDK for game achievements with Epic Games Store

Bachelor's thesis 42 pages, appendices 0 pages
November 2023

The thesis report aims to provide an insight into Epic Game’s ecosystem along with its services that are utilized for game development, more specifically the Epic Game’s Achievement System. The thesis focuses on implementing the EOSSDK provided by Epic Game to establish an achievement system and connect to the EOS server.

The goal of this project is to integrate the EOSSDK’s APIs for use in interacting with the EOS server and access the features provided by Epic Online Services. Implementing these features in game applications is a common practice in enhancing user experience and encouraging users to be more engaged in the application.

Key words: EOSSDK, EOS, Epic Games, achievement

CONTENTS

1 INTRODUCTION 5
2 THEORY 7
2.1 Achievement in gaming 7
2.2 How Achievement System works 8
2.3 EOSSDK overview 9
2.4 Tools used 10
3 IMPLEMENTATION 12
3.1 EOSSDK set up 12
3.2 EOSSDK integration 14
3.3 Connecting players to the server 19
3.4 Sending and receiving player's stats 26
4 TESTING 36
4.1 Testing criteria 36
4.2 Error handling 38
5 CONCLUSIONS 40
REFERENCES 41

ABBREVIATIONS AND TERMS

EOS	Epic Online Services
SDK	Software Development Kit
API	Application Programming Interface
PUID	Product User ID

1 INTRODUCTION

The game industry has been growing rapidly along with the development of information technology. The amount of games released and active players have been increasing significantly throughout recent years, and the demand for newer and more advanced game applications has been rising accordingly.

According to an article by Josh Howarth (2023), the gaming market in 2023 has a value of about 385 billion dollars and growing. It is approximated that by 2027, the gaming industry will worth over 522 billion dollars. Josh Howarth also stated that there are roughly 3.09 billion active video gamers worldwide, which has risen by over 1 billion in just seven years, or a 32% increase in quantity.

The term ‘Achievement’ has become quite popular among the gaming community, and is found on the majority of games introduced to the market in recent years. Achievement systems are implemented in games with the aim of providing players with goals to complete while experiencing their products, which encourages players to be more engaged into the game’s world.

Implementing Achievement systems has become a common practice among the game industry, meeting players’ demand on competing challenges during gaming sessions. The implementation of achievements generally extends the product’s longevity as players are encouraged to spend more hours exploring the game’s system and features, and in cases can also develop a more active community for the specific product and for the developers.

Epic Games announced their own store front for delivering game applications in 2018, and provided developers with an SDK for interacting with their online services. The SDK have been developed and updated rapidly to encourages more developers to publish their products on the Epic Games Store and explore their online features.

This thesis aims to implement the EOS SDK to access Epic Game’s Achievement feature to enhance user experience for games published on the Epic Games

Store. The goal of this project is to have the SDK's APIs integrated correctly within the game's code base, which allows the application to interact with Epic Online Services features. The report consists of 5 sections, with Section 2 discussing the basic ideas of achievement and the Epic Online Services, Section 3 describes an in-depth view on the integration of the SDK, Section 4 presents common testing criteria and errors, and Section 5 concludes the importance of the implementation.

2 THEORY

2.1 Achievement in gaming

Achievement in gaming are goals or challenges that are set out by developers to encourage players to complete tasks and compete with each other. These challenges are laid out in order to enhance user experience and encourage user interaction with the game environment and community. Implementing this feature has become a common practice in the game industry, and can be found on most titles releasing in recent years.

Achievements are implemented in games to motivate players to further explore the game environment and features, discover game secrets and completing challenges, thus improve player interaction and extend the game's longevity. These achievements can include beating levels, discovering secret areas, achieving certain statistics, completing different difficulties, or in some cases player can also obtain an achievement for completing every other achievement in a game. This system is so popular that it encourages a group of players to pursue the unlocking of achievements in various games, who are referred to as 'achievement hunters' according to an article published by Tyler Wilde in 2016.

Corresponding to Adrian Hon's article (2022), while gamification predates video games, achievements in video games systematized the mechanics and aesthetics of gamification. The earliest form of achievement can be dated back to 1982, with Activision's patches which players can receive for accomplishing high scores (Kyle Hilliard, 2013). Ten years later saw the introduction of E-Motion, one of the earliest games where achievements are programmed directly into the game itself. While it provided more instant feedback to players, the system lacked the possibility for players to showcase their achievements comparing to Activision's patches (Adrian Hon, 2022). In 1996, the launch of MSN Gaming Zone fulfilled players' need of sharing gaming accomplishments by introducing the MSN instant messaging network.

While many games developed their internal achievement systems, it was until 2005 that a truly platform-wide multigame achievement system first introduced by

Microsoft, with the Microsoft's Xbox 360 Gamerscore system introduce at the E3 event (Mikael Jakobsson, 2011). This system increased player engagement and competition on Microsoft's platform, which increased playtime in general, and with it the total profit. Other large publishers followed in Microsoft's footsteps, with Valve's Steam platform introducing a platform-based, multi-game achievement system in 2007, Sony's PlayStation 3 in 2008, and Apple's Game Center in 2011. In 2020, Epic Games started adding achievements into available titles on their storefront.

2.2 How Achievement System works

While most game features such as quests and levels are implemented as in-game systems, the achievement system tends to be managed separately from other game mechanics. Each achievement contains a completion condition, which is triggered through specific in-game events. These events can be player actions, a change in game state, or an update in player's statistics. Luca Galli (2014) defined the condition as a set of pre-requirements on the present state or on past actions that must hold in order for the completion of the achievement to be attributable. The Achievement system communicates with other game systems through game variables that specify the achievement's unlocking conditions, usually as an integer or a boolean, and sends notification after the conditions are met and an achievement is unlocked.

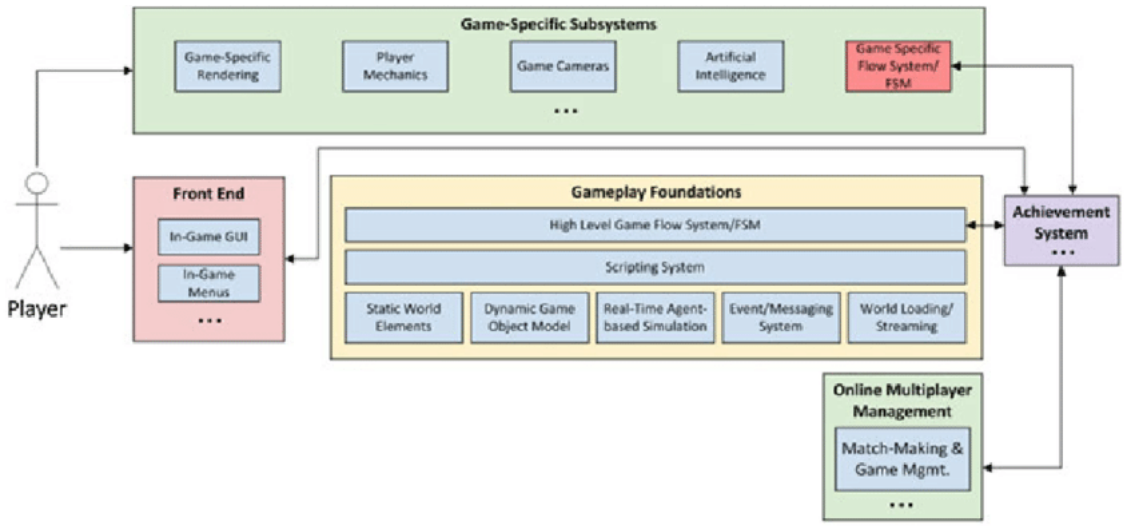

FIGURE 1. Game architecture with achievement system (Luca Galli, Achievement System Explained, 2014).

The Epic Online Services provide a Stats system that keeps track of player's statistics, which can be linked to specific achievements to represent unlocking conditions. Any linked achievements can then define a threshold which defines the conditions where the achievement can be completed. The Epic Online Services server handles achievement completion automatically by comparing the player's stats to the predefined threshold upon receiving update calls from the application. If the conditions are met, the server sends a notification to the application indicating that the achievement has been successfully unlocked, and a notification should also appear on the player's screen to announce their completion.

2.3 EOSSDK overview

Epic Games is a video game and software developer and publisher based in America. Their most notable product is the Unreal Engine, a powerful game engine made to deliver high-quality games. The Epic Game Store was announced in 2018 as their own storefront to deliver game titles.

Epic Online Services (EOS) is a set of developer resources created by Epic Games to add network-based functionalities and storefront distributions to games. EOS provides games-platform agnostic services to launch and operate games regardless of platforms or engines used in development. Epic Online Services can be separated into two smaller subsets: EOS Game Services and EOS Epic Account Services. The EOS Game Services allow developers to implement online game features such as multiplayer, recording player progression or managing game operations, while the EOS Account Services handle player accounts within the Epic Game's ecosystem, dealing with social features and authorization.

The EOS SDK was developed to allow developers to implement cross-platform play features into their game, namely match-making, leaderboards, and achievements. The SDK enables developers to access the Epic Online Services from within their application. The EOS Services are integrated within applications through the use of different interfaces, known as the SDK APIs. Each interface

provided by the SDK allows implementation of a specific service, for instance implementing the Achievement service would require the use of the Achievement Interface.

The SDK provides versions in both C and C# that are available on various platforms: Windows, macOS, Linux, IOS, Android and different consoles. Developers who wish to implement the SDK on consoles must apply for approval from Epic Games and from the platform holders: Microsoft (Xbox), Sony (PlayStation) and Nintendo (Switch).

The C# version of the SDK, while provides the same functionalities, follows different design patterns comparing to the C version according to the Epic Developer Resources. Most notably, the C# version adheres to C# best practices and follows an object-oriented approach, and the naming conventions are modified to match the typical C# patterns. It is recommended to install a .NET Framework 3.5 or higher to work with the SDK in C#.

2.4 Tools used

This project implements the EOSSDK using the Unity tool. Unity was developed by Unity Technologies to provide a cross-platform game development environment, for both 2D and 3D games. Unity has become one of the most popular game engines among the game development community due to ease of use and a wide range of accessible documentation. As development through Unity mainly use the C# programming language, the EOSSDK version used for integration is the C# version.

To set up the application for use on the Epic Games Store, developers must have access to the Epic Game's Developer Portal, a browser-based tool that allows developers to set up and configure their products. The Portal provides a user interface for managing product-related information and database for services like the Achievement system. Through the Portal, developers can keep track of their player's statistics and usage reports, and manage product deployments and binaries. To begin interacting with the Epic Online Services' features such as the Achievement feature, the related databases containing information on the game's

achievements and stats must be configured through the Developer Portal prior to implementing the SDK's APIs within the game's code base.

Epic Games also provide a Developer Authentication Tool that allows developers to remain logged in to the server between testing sessions, which improves iteration time. The Authentication Tool provides an easy method of accessing the EOS online features and test the implemented services. According to the Epic Developer Resources, the Tool is a combination of a web browser and a server, with the web browser portion provides the user interface, while the server portion responds to SDK requests. Using the Authentication Tool during the testing phase of development is therefore recommended for developers.

3 IMPLEMENTATION

3.1 EOSSDK set up

To obtain the EOS SDK and integrate it onto a product, an Epic Game's Developer Portal account must be registered, where the tools needed for integration can be found. The Developer Portal enables product managing on the storefront, updating game builds, and acquiring product statistics.

EOS products offer different sandboxes that aid development, allowing ease of management. These sandboxes are high-level distribution environments that contains information, configurations, and binaries for specific deployments of the product. Defining different deployments allows easy management of different builds for the product. Each sandbox provides a different set of sandbox ID and deployment ID, which are required for initializing the SDK in EOS products and accessing data for that deployment. Products will be in the Dev sandbox at earlier stages of development. Once features are implemented, developers can push their products to the Stage sandbox, where Epic Games will carry out tests on the deployments. After all criteria have been met, the products will go to Live sandbox and ready to be delivered to public users.

- Organization: `Epic Games`
  - Product: `Jazz Jackrabbit`
    - **Sandbox**: `Development`
      - Deployment: `DevGame01`
    - **Sandbox**: `Staging`
      - Deployment: `QATestGame01`
      - Deployment: `AlphaGame`
      - Deployment: `BetaGame`
    - **Sandbox**: `Live`
      - Deployment: `LiveGame`

FIGURE 2. Epic Game's product management system
(Epic Developer Resources)

A set of Client with an attached Client Policy must also be created for products to deliver features to targeted clients. The Client Policy defines what permissions users have while interacting with game features, for example matchmaking, updating and checking statistics, and accessing player data. Other than the actions defined in the Client Policy, unpermitted request called from the user's system to the Epic Game's server will be denied.

After setting up the product in the Developer Portal, the SDK can be downloaded through the Developer Portal homepage. There are different versions for different platforms, with the exception of console platforms which you have to apply for console developer access. The downloaded file contains samples and an SDK file with libraries and header files for SDK integration. This SDK file must be added to the product's Unity project following the structure in FIGURE 3.

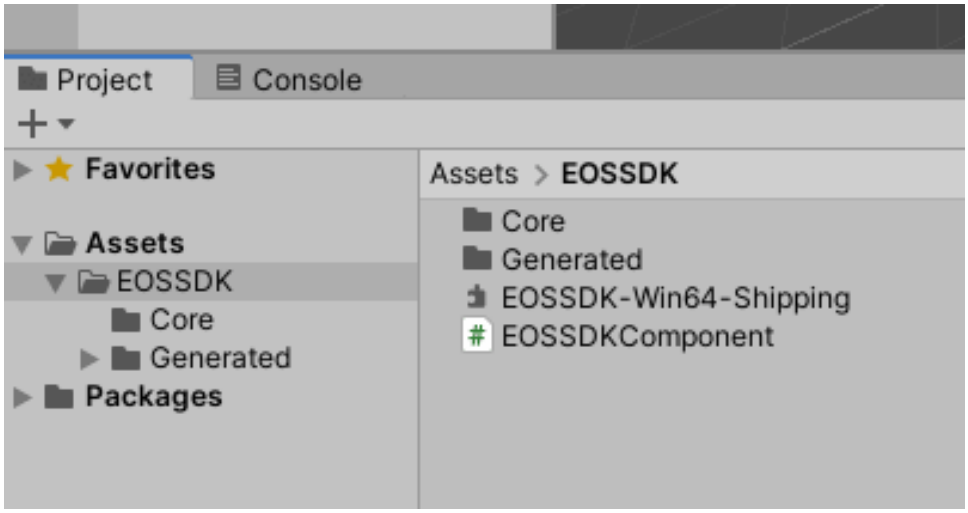

FIGURE 3. Project structure with EOSSDK in Unity (Epic Developer Resources)

The 'EOSSDKComponent' script must be manually created to handle the SDK's functions. This script should be added to a game object that maintains an instance through every scene in the game, which is known as the singleton pattern. According to Robert Nystrom (2011), the singleton pattern 'ensures that a class has one instance, and provide a global point of access to it'. This will keep the object associated with the 'EOSSDKComponent' script from being destroyed and avoid creating new objects when loading new scenes, thus avoiding errors when trying to access this object's functionalities while different versions of the same object exist in the scene. The singleton pattern also allows other objects in the game to easily access this object through a static variable instance.

```
Unity Message | 0 references
private void Awake()
{
    if (Instance == null || Instance == this)
    {
        Instance = this;
        DontDestroyOnLoad(this.gameObject);
    }
    else
    {
        Destroy(this.gameObject);
        return;
    }
}
```

FIGURE 4. Defining a singleton pattern in Unity.

The ’EOSSDKComponent’ script will store all necessary information needed to initialize and handle interaction with the EOSSDK system. This includes the product’s name, version, ID, the sandbox IDs and deployment IDs, the Client ID, and the client ID Token retrieved during initialization for handling the SDK. This component will also store references to different interfaces of the SDK for other objects in the game to interact with.

The object containing the ‘EOSSDKComponent’ script should be created as soon as possible to handle player authentication and connection before allowing players to interact with other game-related features.

3.2 EOSSDK Integration

When an application is launched from the Epic Games Launcher, the launcher will send a command line with a set of parameters which contains useful startup information. The command line will be in the following format:

app.exe -AUTH_LOGIN=unused -AUTH_PASSWORD=<password> -AUTH_TYPE=exchangecode -epicapp=<appid> -epicenv=Prod -EpicPortal -epicusername=<username> -epicuserid=<userid> -epiclocale=en-US -epicsandboxid=<sandboxid>

The program can get this command line by calling 'Environment.GetCommandLineArgs' function on C#. An important parameter that is needed for initializing the SDK is the 'AUTH_PASSWORD', which provides an 'Exchange Code' that is needed for users to log in. The application must parse this information and save the Exchange Code. This Exchange Code will later be used during the login process.

```
string[] args = Environment.GetCommandLineArgs();
string input = "";

if (args != null)
{
    for (int i = 0; i < args.Length; i++)
    {
        if (args[i].Contains("-AUTH_PASSWORD"))
        {
            string command = "-AUTH_PASSWORD=";
            input = args[i].Remove(0, command.Length);
            _exchangeCodeToken = input;
        }
```

FIGURE 5. Acquiring the Exchange Code token.

There are different interfaces that must be used for the product to interact with the Epic Game's server. The first and most important interface is called the Platform Interface, which provides access to every other interfaces provided by the SDK and handle their functions. To access this interface, you must call the interface's 'Initialize' and 'Create' functions.

The 'Initialize' function should be called as soon as the game start, which in Unity's case is the 'Awake' function, and should be called before calling any other functions provided by the SDK. The 'Initialize' function requires an object of type 'InitializeOption' as parameter, which contains information on the product's name

and version, and returns an object of type ‘Result’ to indicate the result of the function call.

```
var initializeOptions = new InitializeOptions()
{
    ProductName = m_ProductName,
    ProductVersion = m_ProductVersion
};

var initializeResult = PlatformInterface.Initialize(ref initializeOptions);
if (initializeResult != Result.Success)
{
    //throw new Exception("Failed to initialize platform: " + initializeResult);
    Debug.LogWarning("Failed to initialize Epic platform: " + initializeResult);
}
```

FIGURE 6. ‘Initialize’ function call.

To aid in debugging, the SDK allows registering callbacks that will output useful information. To access the debugging contents, a function of type ‘LogMessageFunc’ must be implemented, which handle the debugging results. These functions should be called as soon as possible, preferably following initialization.

```
// The SDK outputs lots of information that is useful for debugging.
// Make sure to set up the logging interface as early as possible: after initializing.
LoggingInterface.SetLogLevel(LogCategory.AllCategories, LogLevel.VeryVerbose);
LoggingInterface.SetCallback((ref LogMessage logMessage) => Debug.Log(logMessage.Message));
```

FIGURE 7. Register logging callbacks to access debugging information.

The Platform Interface’s ‘Create’ function is used to create a running instance of the interface, which will allow access to the Epic Online Services functionalities. The function requires a platform-specific parameter called ‘PlatformOptions’, which in this case is of type ‘WindowOptions’. This object contains the product-related information, such as the Sandbox and Deployment ID (which must be set to the corresponding ID for your targeted sandbox), and Client Credentials ID. This is also where options to enable the EOS’s Overlay can be set. The Overlay is an UI layer where Epic Games displays useful information to the user, such as user info or user’s achievements. The Overlay supports Direct3D9 (D3D9), Direct3D10 (D3D10) and OpenGL on Windows.

```
var options = new Epic.OnlineServices.Platform.WindowsOptions()
{

    ProductId = m_ProductId,
    SandboxId = _currentSandboxID,
    DeploymentId = _currentDeployID,
    ClientCredentials = new ClientCredentials()
    {
        ClientId = m_ClientId,
        ClientSecret = m_ClientSecret
    },
    IsServer = false,
    Flags = PlatformFlags.WindowsEnableOverlayD3D10
            | PlatformFlags.WindowsEnableOverlayD3D9
            | PlatformFlags.WindowsEnableOverlayOpengl,
};

s_PlatformInterface = PlatformInterface.Create(ref options);
```

FIGURE 8. Create a Platform Interface instance.

After obtaining a Platform Interface instance, the instance can be used to access other interfaces through getter functions. These functions return instances of the designated interfaces, and should be saved in variables for easy access.

```
s_UserInterface = s_PlatformInterface.GetUserInfoInterface();
s_StatsInterface = s_PlatformInterface.GetStatsInterface();
s_AchievementInterface = s_PlatformInterface.GetAchievementsInterface();
s_AuthInterface = s_PlatformInterface.GetAuthInterface();
s_ConnectInterface = s_PlatformInterface.GetConnectInterface();
```

FIGURE 9. Accessing different interfaces.

In addition to allowing access to other interfaces, the Platform Interface is also vital for their functionalities. As many of the interfaces provide functionalities that are asynchronous, the Platform Interface must be kept running throughout the program's life cycle to keep the SDK running correctly. The Platform Interface contains a 'Tick' function that must be call in the game's main loop to update the asynchronous functionalities.

At the end of the program's lifecycle, the Platform Interface must also be terminated to release the memory hold by the SDK. This includes calling 'Release'

function on the Platform Interface instance, then call the ‘PlatformInterface.Shutdown’ static function. After the ‘Shutdown’ function has been called, the SDK will not be accessible and unable to be reinitialized.

```
Unity Message | 0 references
private void OnApplicationQuit()
{
    if (s_PlatformInterface != null)
    {
        s_PlatformInterface.Release();
        s_PlatformInterface = null;
        PlatformInterface.Shutdown();
    }
```

FIGURE 10. Shut down the SDK.

The program can also save network and connection statuses to check successful connections and attempt reconnections when needed. This can be used in delaying or cancelling function executions relating to the SDK’s connections. Monitoring connection status can also aid in debugging during implementation.

```
2 references
public static bool IsOnline
{
    get
    {
        return NetworkStatus == NetworkStatus.Online;
    }
}

36 references
public static bool IsConnected
{
    get
    {
        if (String.IsNullOrEmpty(s_ProductUserID.ToString()) || s_ConnectInterface == null)
            return false;

        return LoginStatus == LoginStatus.LoggedIn;
    }
}
```

FIGURE 11. Variables for connection status.

3.3 Connecting players to the server

To enable interactions between the players and Epic Game's server, the program must authenticate users' identification and connect the players. This process involves the use of the Authenticate Interface and Connect Interface. The Authenticate Interface allows players to interact with Epic Games Account related information, while the Connect Interface is used to connect players to the EOS backend server and access their game data. After connecting players to the EOS backend server, a Product User Id (PUID) will become available to be used with user-specific calls to the server.

As many of the functions provided by the SDK are asynchronous, the program should have a way to handle delays when sending requests to Epic Game's server. One method to achieve this in Unity is using coroutines, which allows suspension of program execution. As Chistopher D. Marlin stated in 1979: 'the execution of a coroutine is suspended as control leaves it, only to carry on where it left off when control re-enters the coroutine at some later stage'. Christopher D. Marlin's view on coroutine's possibilities include the ability to 'transfer control explicitly from one coroutine to another, causing the currently executing coroutine to become suspended and a target coroutine to resume execution'. Specifically in this case, coroutines can be used to implement the connection process where the Connect Interface must wait for Authenticate Interface's process to complete before taking over the control, or used to inform other processes such as sending and receiving data to delay or cancel while no valid connection to the server is available.

This process of execution can be referred to as the communicating sequential process, which describe a pattern of interaction in concurrent systems (Roscoe, A. W., 1997). The connection process can be divided into two smaller subcoroutines: Authenticate coroutine and Connect coroutine. As the use of the Connect Interface requires an Exchange Code retrieve from authentication, the Connect coroutine's execution should be delayed until the Authenticate coroutine completes its tasks. The order in which these two coroutines execute is described in FIGURE 12. The Connect coroutine must wait for authentication to complete before executing, as it requires information received from authentication.

```
_authenticateCoroutine = StartCoroutine(AuthenticateUser());
yield return _authenticateCoroutine;

_connectCoroutine = StartCoroutine(ConnectUser());
yield return _connectCoroutine;
```

FIGURE 12. Connecting process using coroutine.

FIGURE 13 below demonstrates the Authentication flow of the program. The Exchange Code received from the launcher's command line parameter is used to authenticate the user with the Epic's authorization server. After authentication, the game will receive an access token from the server which is used in the later Connect process. In this project's specific case, the token will only be used in the Connect Interface to retrieve a Product User ID, thus continuously refreshing the token is not required.

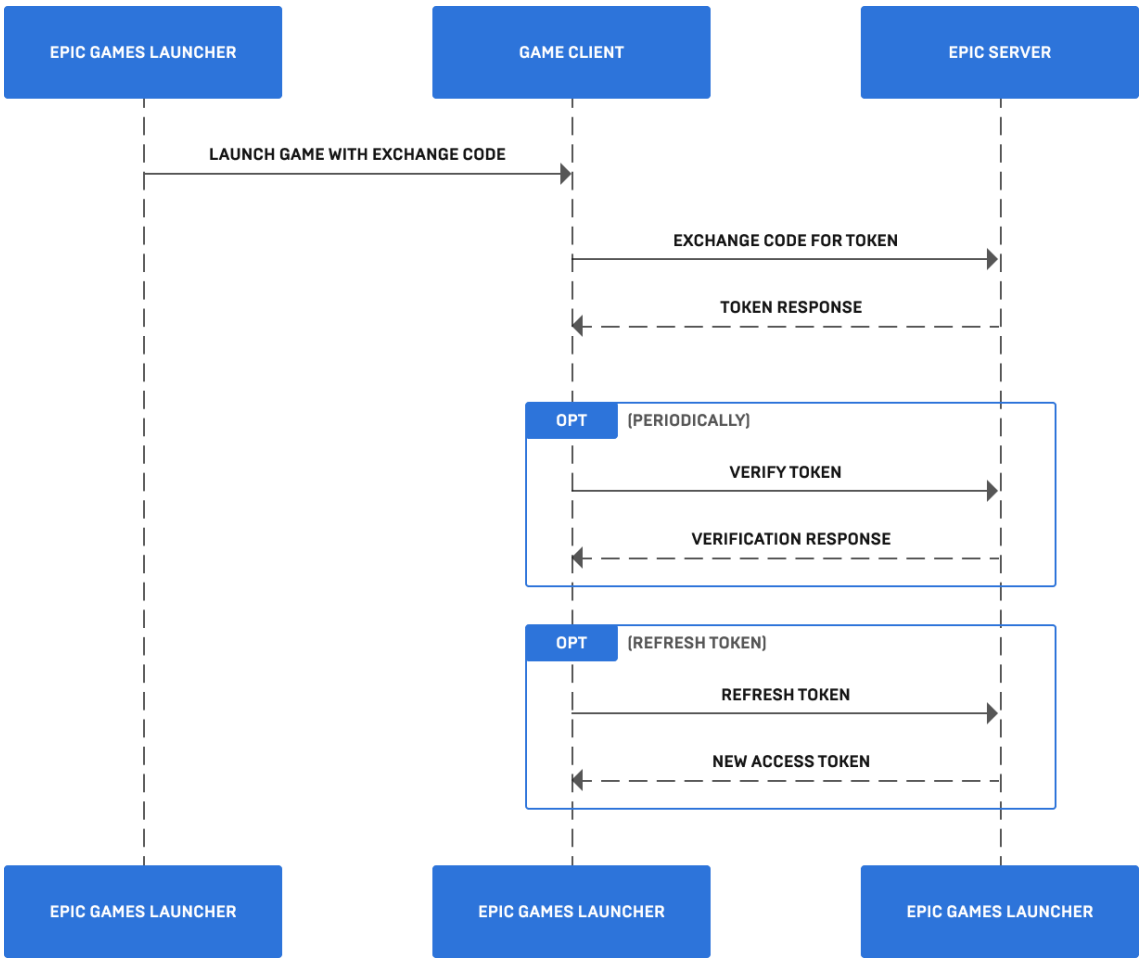

FIGURE 13. Authentication flow (Epic Developer Resources Documentation)

The access token retrieved from the above Authentication flow is then used to connect the user with the EOSSDK. The most important information received from the process is the PUID (Product User ID) which is used in interaction with the backend server. This ID is associated with a unique user for a specific product, and is required for accessing user game data associated with the current product. As demonstrated in FIGURE 14, after sending a login request to the server, if there are no user credentials valid for this product, the user will be given a choice between logging in using a different identity provider or creating a new user for this product. As the project in this thesis only aims for user from the Epic Games system, the program will only attempt to create a new user. After the user is connected to the server, through logging in or creating new user, a user token (which is the PUID) is available to be used in interaction with the backend server.

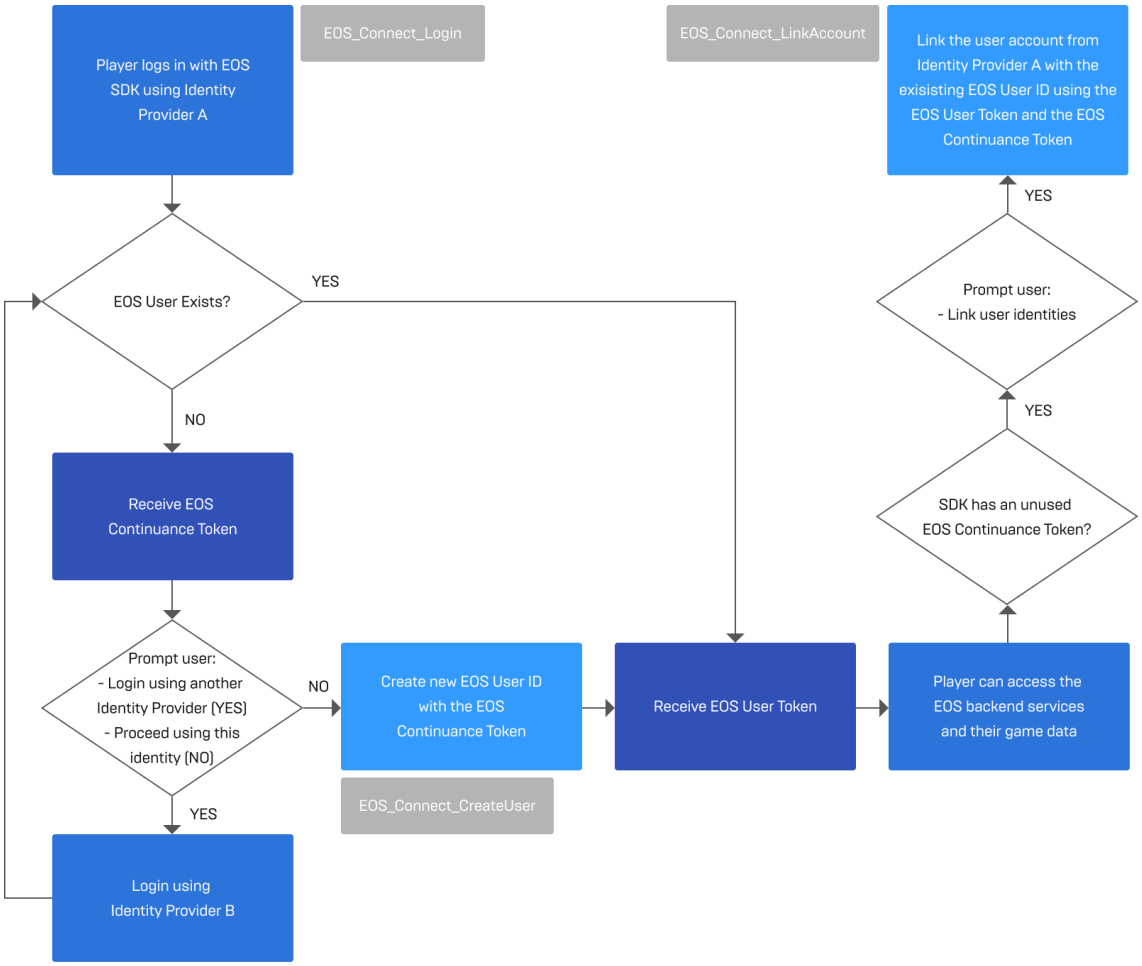

FIGURE 14. The EOSSDK' s Connect flow.
(Epic Developer Resources Documentation)

The Authenticate Interface provides the ‘Login’ function that must be called before interacting with Epic Account Service’s features. The ‘Login' function requires a parameter of type ‘IOSLoginOptions’ structure, which contains the required login credentials, and a callback function that will be executed when the login process is completed. The Credentials have three variables: ID, Token and Type. As logging in using the Epic Game’s launcher prefers the use of Exchange Code, ID information is not required and thus can be left blank. The Token field should be set to the Exchange Code acquired from the command line parameter sent from the launcher at program’s start up, as defined in the previous part and in the above FIGURE 5.

```
Epic.OnlineServices.Auth.IOSLoginOptions AuthLoginOptions = new IOSLoginOptions()
{
    Credentials = new IOSCredentials()
    {
        Token = _exchangeCodeToken,
        Type = LoginCredentialType.ExchangeCode
    },
};
```

FIGURE 15. Login options.

There are situations where logins cannot be performed due to failure in Exchange Code acquisition, which can be caused by errors in implementation of acquiring and parsing the command line, or caused by users not launching the product from the launcher (by double-clicking on the product’s icon), which provides no valid command line arguments. The program should implement checks to ensure that the login process is successful, for example by changing the Login Type to a different valid type. The recommended type for handling these situations is the ‘AccountPortal’ login type, which redirects users to the Epic Game’s Login Portal during the login process, where users can provide their accounts’ credentials to be used in connecting users with the server.

```
LoginCredentialType loginType = LoginCredentialType.ExchangeCode;

if (string.IsNullOrEmpty(_exchangeCodeToken))
{
    Debug.LogError("Exchange Code Not Found");
    loginType = Epic.OnlineServices.Auth.LoginCredentialType.AccountPortal;
}
```

FIGURE 16. Changing login types.

The ‘Login’ function provides a callback to handle information received from the login process under the ‘LoginCallbackInfo’ structure. This structure contains a Result Code that determines the success of function execution, that should be used for error handling in the process, and possibly reattempts to perform login actions. A Local User ID can also be obtained, which refers to the user’s account ID. This ID should be saved to be used in later processes of the login procedure. The program should also define a variable of type Boolean to indicate the function’s end of execution. This Boolean value can be used to delay further execution of the login process until the user’s Epic Account ID, defined as Local User ID, has been retrieved, which is essential to perform later tasks. Another way of achieving the same purpose is to save the login info in a variable that can be accessed by other functions in the login process.

```
s_AuthInterface.Login(ref AuthLoginOptions, null,
    (ref LoginCallbackInfo loginInfo) =>
    {
        if (loginInfo.ResultCode == Result.Success)
        {
            s_EpicAccountID = loginInfo.LocalUserId;
            userLoggedIn = true;
        }
        else if (Epic.OnlineServices.Common.IsOperationComplete(loginInfo.ResultCode))
        {
            Debug.LogWarning("Login failed: " + loginInfo.ResultCode);
        }
    });
```

FIGURE 17. Handling Authentication callback.

Using the Local User ID, an ID Token can be retrieved. According to the Epic Developer Resources Documentation, this ID Token is used to securely verify user identities with online services. A common use case for the ID Token is to authenticate user by their account ID that should be used to access game-scoped data for the current application. To retrieve the ID Token, the program should call a ’CopyIDToken’ function, which requires a parameter of type ‘idTokenOptions’ that contains the Local User ID.

```
Epic.OnlineServices.Auth.CopyIdTokenOptions idTokenOptions = new CopyIdTokenOptions
{
    AccountId = s_EpicAccountID
};

s_AuthInterface.CopyIdToken(ref idTokenOptions, out idToken);
```

FIGURE 18. Retrieving ID Token.

After the ID Token is acquired, it can be used for logging the user into the EOS server. To begin the process, the game should call the Connect Interface's 'Login' function, which requires 'loginOptions' that contains the Credentials with the login information. The Credentials contains two important fields: a 'Token' field and a 'Type' field. The 'Token' field takes in the ID Token retrieved from Authentication process under the 'Utf8String' structure, which requires using the ID Token's 'JsonWebToken' value that converts the ID Token to a 'JsonWebToken' string value. The 'Type' field is the login type used for the current process, which can be set according to the current identity provider used for accessing the game. In this case, the 'Type' field should be set to 'EpicIDToken'.

```
Epic.OnlineServices.Connect.LoginOptions loginOptions = new Epic.OnlineServices.Connect.LoginOptions
{
    Credentials = new Epic.OnlineServices.Connect.Credentials()
    {
        Token = idToken.Value.JsonWebToken,
        Type = ExternalCredentialType.EpicIdToken
    },
};
```

FIGURE 19. Login option credentials.

After specifying the login options, calling the Connect Interface's 'Login' function will attempt to connect the user to the EOS backend server. The function requires the login options, and a callback function that is executed upon completing the request.

```
s_ConnectInterface.Login(ref loginOptions, null, (ref Epic.OnlineServices.Connect.LoginCallbackInfo loginCallbackInfo) =>
{
    //If login success
    if (loginCallbackInfo.ResultCode == Result.Success)
        {
            Debug.Log("Platform Login succeeded");
            s_ProductUserID = loginCallbackInfo.LocalUserId;
        }
    //If not create new user
    else if (Common.IsOperationComplete(loginCallbackInfo.ResultCode) && loginCallbackInfo.ResultCode == Result.InvalidUser)
        {
            Debug.Log("Platform Login failed: " + loginCallbackInfo.ResultCode);

            Epic.OnlineServices.Connect.CreateUserOptions createUserOptions = new Epic.OnlineServices.Connect.CreateUserOptions
            {
                ContinuanceToken = loginCallbackInfo.ContinuanceToken
            };
            s_ConnectInterface.CreateUser(ref createUserOptions, null,
                (ref Epic.OnlineServices.Connect.CreateUserCallbackInfo createCallbackInfo) =>
                {
                    s_ProductUserID = createCallbackInfo.LocalUserId;
                });
        }
});
```

FIGURE 20. The Login process.

The callback function contains a ‘LoginCallbackInfo’ structure with information on the login attempt, including the ‘ResultCode’ that indicates the result of the function call. If the user associating with the given credentials is not found on the server, the API returns a ‘InvalidUser’ result along with a continuance token that contains details about the login attempt to continue the login flow. After receiving ‘InvalidUser’ result, the program should call ‘CreateUser’ to create a new user credentials for the current user with the current product. This function uses the continuance token received from the failed login attempt that contains user details to create a new user with the given information. Successful calls to the ’Login’ or ‘CreateUser’ API will result in a callback structure containing a Local User ID, which is the PUID. This Local User ID is the user ID associated with the current product, which is not identical to the Local User ID received from the Authentication process that refers to the account ID associated with the user’s Epic Games Account.

The access token received from the Login process have a limited lifetime, and must be refresh after expiration. The Connect Interface provides a ‘AddNotifyAuthExpiration’ function that will notify the program upon token expiration. This function registers a callback to be called approximately 10 minutes before the token expires, which gives the program enough time to call ‘Login’ function with the valid credentials to refresh access. The callback function should start a new Connect process to refresh connection to the server. If the program no longer requires refreshing access, the ‘RemoveNotifyAuthExpiration’ function can be called to stop listening to the expiration notifications.

```
//Notify token expiration and refresh
Epic.OnlineServices.Connect.AddNotifyAuthExpirationOptions addNotifyAuthExpirationOptions =
                                                            new Epic.OnlineServices.Connect.AddNotifyAuthExpirationOptions { };
s_ConnectInterface.AddNotifyAuthExpiration(ref addNotifyAuthExpirationOptions, null,
    (ref Epic.OnlineServices.Connect.AuthExpirationCallbackInfo callbackInfo) =>
    {
        if (_connectCoroutine != null)
        {
            StopCoroutine(_connectCoroutine);
            _connectCoroutine = null;
        }
        _connectCoroutine = StartCoroutine(ConnectUser());
    });
```

FIGURE 21. Add token expiration notification.

The program can also keep track of the user’s authentication status. This is useful to check for valid login status before registering any request to the server, or to

notify when authentication fails. The Connect Interface provides a 'AddNotifyUserLoginStatusChanged' function to register a callback that is called every time the user's login status is changed. According to the Epic Developer Resources Documentation, this only occurs when the expiration notification is ignored, the external provider credentials are unable to refresh the access token, or the access was revoked explicitly by the backend service for some administrative reason. The Documentation also states that the callback only fires when other calls in the EOS ecosystem detect that the current auth token is invalid, otherwise it does not automatically fires.

```
Epic.OnlineServices.Connect.AddNotifyLoginStatusChangedOptions addNotifyLoggedInStatusChangedOptions =
                                                            new Epic.OnlineServices.Connect.AddNotifyLoginStatusChangedOptions { };
s_ConnectInterface.AddNotifyLoginStatusChanged(ref addNotifyLoggedInStatusChangedOptions, null,
    (ref Epic.OnlineServices.Connect.LoginStatusChangedCallbackInfo callbackInfo) =>
    {
        LoginStatus = callbackInfo.CurrentStatus;
    });
```

FIGURE 22. Register login status change callback.

3.4 Sending and receiving player's stats

The Epic Games Store provides configuration-only achievement feature for the Epic ecosystem (Rajen Kishna), with achievement progress linked to the corresponding stat progress. The EOS backend server will check the stats received from the game and unlock achievements automatically when the stats associated with the achievements meet a required predefined amount. Thus, the game only needs to update the player's stats on the server in order to unlock player achievements. Alternatively, if the product does not utilize the use of stats, achievement logic can also be created through in-game code and unlock corresponding achievements using the EOS Achievements Interface API.

The Epic Game's stats system tracks player's statistics, which can be used to automatically unlock player achievements. Before interacting with the player's stats, the necessary stats must be defined on the Epic Game's Developer Portal. The Developer Portal provides a menu for creating and managing all stats defined for a product. Creating a stat requires a stat name which will be used to update the specific stat in the program's code, and a stat's aggregation type which defines how a stat will be updated upon sending new data to the server. There are

four aggregation types that can be used: SUM, LATEST, MIN and MAX. The SUM aggregation type defines a stat to be updated according to the total amount of every data sent through all requests the game sent to update stats on the server. The LATEST aggregation type updates the stats stored on the server base on the latest data sent from the game. According to Epic Games Documentation, the LATEST type is the only aggregation type that does not support automatic unlock of achievements, which makes this type only suitable for data storage. The MIN and MAX types store data according to the lowest or highest amount sent through the game's update calls to the server.

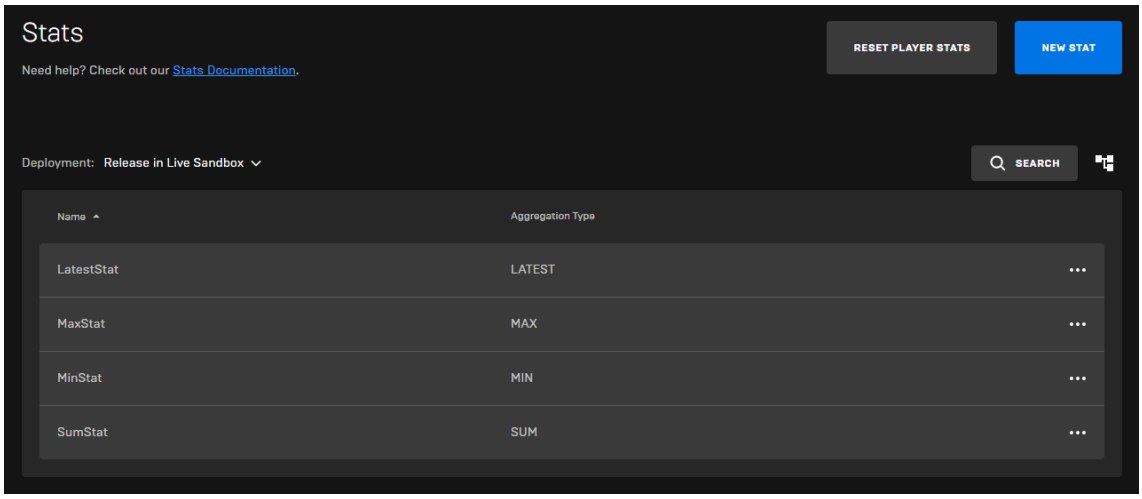

FIGURE 23. Developer Portal's menu for creating and managing stats (Rajen Kishna, Introduction to Epic Online Services (EOS))

The EOS server handles sent and received data as 32-bit integers. Depending on how the game handles data, the suitable data types should be chosen accordingly. For example, if the game increments data within the game loop, the use of MAX aggregation type is applicable, as it saves the highest value from every updates. If the game handles both incrementing and decrementing data values, using the SUM aggregation type is a more suitable choice, which saves the total value of data sent to the server across all sessions. Using the SUM aggregation type also exclude the program from having to retrieve data from the server to update data correctly. The use of aggregation type also depends on the conditions defined with different achievements. Handling data that requires saving the least amount, such as fastest completion time or shortest distance, should utilize the use of MIN aggregation type. The LATEST aggregation type, while unable to

associate with achievements, can still be used for saving and handling the most recent data sent to the server.

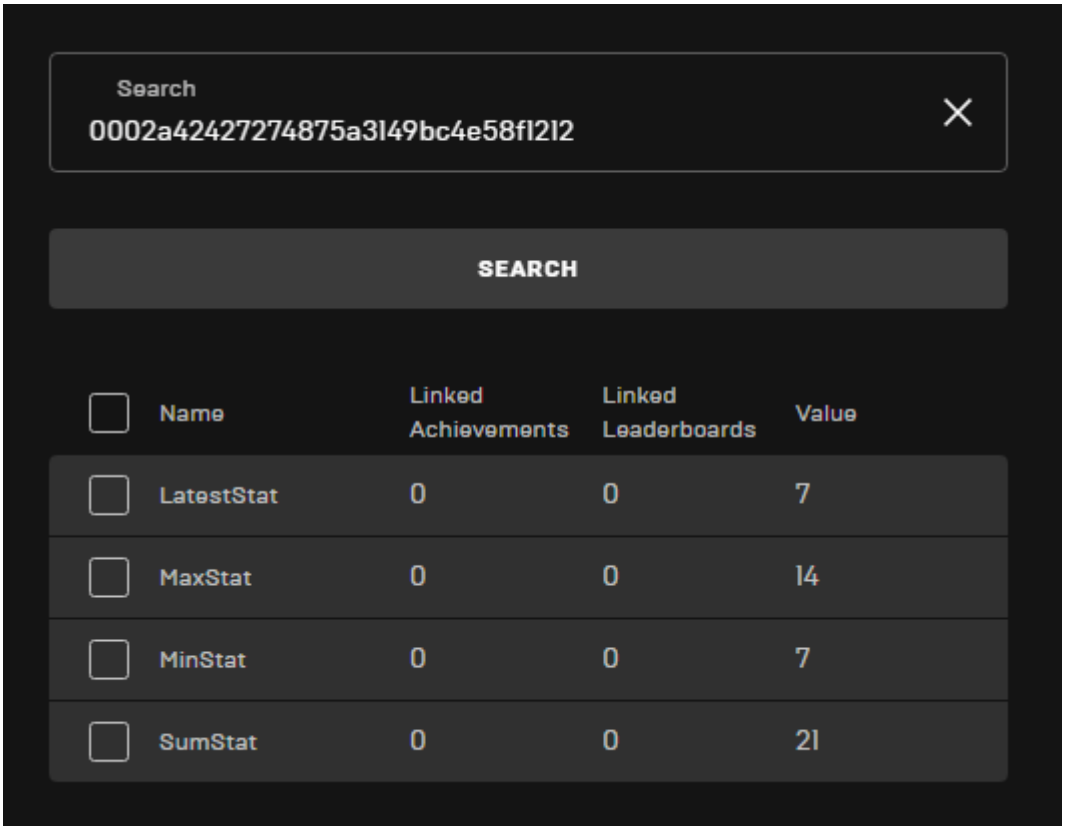

FIGURE 24. How different aggregation type is handled
(Rajen Kishna, Introduction to Epic Online Services (EOS))

After defining the appropriate stats for the game, corresponding achievements should also be created on the Developer Portal. Creating achievements requires different stats related to the achievement to be set up, and a threshold to be defined which indicates the value a stat must reach in order to unlock the achievement. An achievement can be associated with many different stats, in which case the achievement's progression is calculated based on the combined progression of all related stats. The achievement also contains achievement ID for used in accessing and handling achievement data within the game's update loop.

Epic Games provides two methods of defining achievements for a product: by creating multiple achievements at once using bulk import and uploading to the Developer Portal as a CSV file, or by defining different achievements once at a time using the Developer Portal's interface.

Bulk importing allows developers to upload multiple achievement definitions at once to the server. According to Epic Developer Resources, this method is recommended for products that already have defined achievements, as it allows downloading the CSV file containing the existing achievements and modify multiple definitions with ease. If the product's achievement database requires modification after being uploaded, this method allows developers to easily edit and update their product's list of achievements. Alternatively, an Epic Games Store achievements template can be downloaded for creating new achievements database. The downloaded template provides a 'achievementDefinitions.csv' file where the product's achievements can be listed. This file contains four columns for defining achievements: 'name', 'hidden', 'statThresholds' and 'user_epic_achievements_xp'. The Epic Developer Resources specifies that in some cases where the 'user_epic_achievements_xp' is missing from the file, developers should add this column manually. While this method provides easy modification of multiple definitions at once, possible errors can occur during the process where developers might provide incorrect or misspelled stats names.

	A	B	C	D
1	name	hidden	statThresholds	user_epic_achievements_xp
2	achvTutorial	FALSE		10
3	achvLumberjack	FALSE	numTreesChopped:3	50
4	achvWoodsman	TRUE	numTreesChopped:100;numAxesUsed:5	100

FIGURE 25. Achievement definition csv file (Epic Developer Resources)

The other way of defining achievements for a product is to manually create the definition once at a time using the Developer Portal. To create an Epic Games Store achievement, a corresponding Epic Online Services must also be created. The difference between these two types of achievements is that the Epic Games Store achievements contains information that are displayed to players and linked to an EOS achievement, while an EOS achievement handles the achievement's unlocking logic. The EOS achievements can be created through the Developer Portal and linked to specific stats to define logic for unlocking conditions, or defined through corresponding code logic in the product and unlock through the EOS Achievements Interface API. After creating the appropriate EOS achievements, Epic Games Store achievements can be created and linked for players to view and unlock.

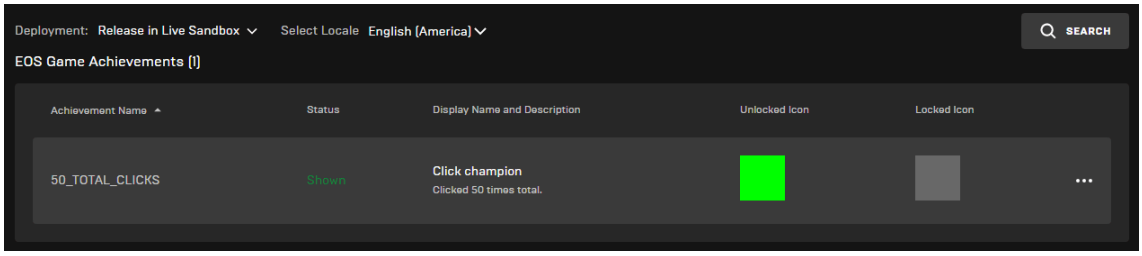

FIGURE 26. Achievement definition on the Developer Portal
(Rajen Kishna, Introduction to Epic Online Services (EOS))

After setting up the required data through the Epic Game's Developer Portal, the product can implement code logic for the achievements. There are two ways for the EOS achievements to be unlocked: using EOS Stats Interface API that sends stats data from which the linked achievements can be automatically unlocked, or using EOS Achievements Interface API that allows the product to unlock the achievements through implemented code logic. These two processes include partially identical steps of querying and ingesting data to and from the server, and based on the product's usage and developers' preferences the appropriate process can be utilized.

The EOS Stats Interface API provides many useful functions for developers to manage user's stats for the application. After defining stats data for the product through the Developer Portal, the product can interact with the server for handling user's stats. To access the Stats Interface, an interface handle can be acquired through the Platform Interface used in initialization (see FIGURE 9). After acquiring the Stats Interface handle, the program can call 'IngestStat' function to ingest stats inside the application and submit them to the EOS backend server. According to the Epic Developer Resources, the update on the backend server after stats have been ingested may noy take effect immediately, as submitting and processing stats can take time. For that reason, stats may not get updated immediately on the server and the linked achievements unlocking may have a delay.

The 'IngestStat' function requires a parameter of type 'IngestStatOptions' which contains user ID information, along with an array of type 'IngestData' which contains information on the stats being submitted. The 'IngestStatOptions' provides two ID fields: a Local User ID which specifies the user requesting the ingest, and

a Target User ID which specifies the user whose stats is being ingested. In most cases these two fields should use the same value, with the exception of dedicated servers where the Local User ID can be set to null (Epic Developer Resources). These ID fields refers to the Product User ID which is acquired from the Connect process (see FIGURE 20).

```
Epic.OnlineServices.Stats.IngestStatOptions ingestOptions = new Epic.OnlineServices.Stats.IngestStatOptions
{
    LocalUserId = EOSSDKComponent.s_ProductUserID,
    TargetUserId = EOSSDKComponent.s_ProductUserID,
    Stats = new Epic.OnlineServices.Stats.IngestData[]
    {
        new Epic.OnlineServices.Stats.IngestData( ) { StatName = "myFirstStat", IngestAmount = 0 },
        new Epic.OnlineServices.Stats.IngestData( ) { StatName = "mySecondStat", IngestAmount = 1 },
    }
};
```

FIGURE 27. Sending stats to the server.

After specifying the information to be ingested, calling the ‘IngestStat’ function with the ingest options will submit the data to the server. Similar to many of the SDK’s functions, the ‘IngestStat’ function also contains a parameter of type ‘CompletionDelegate’, which is a callback function that will be executed when the ingest stat operation is completed. The callback function provides a result code that indicates whether the request to the server is successful.

```
EOSSDKComponent.s_StatsInterface.IngestStat(ref ingestOptions, null,
    (ref Epic.OnlineServices.Stats.IngestStatCompleteCallbackInfo ingestInfo) =>
    {
        if (ingestInfo.ResultCode == Epic.OnlineServices.Result.Success)
        {
            Debug.Log("Uploaded Stats to Epic Online Services.");
        }
        else
        {
            Debug.Log("Failed to upload stats to Epic Online Service! Returns: " + ingestInfo.ResultCode);
        }
    });
```

FIGURE 28. The ‘IngestStat’ function.

The SDK also provides a ‘QueryStat’ function for developers to request user’s stats from the server. This function’s structure is mostly identical to the ‘IngestStat’ function, containing the options for querying and a callback function. The options required for querying contains the PUID of the user whose stats are being retrieved through the function, along with optional start time and end time for function execution, and optional list of stat names that will be queried. The callback function will return a result code indicating the function’s success.

```
EOSSDKComponent.s_StatsInterface.QueryStats(ref queryOptions, null,
    (ref Epic.OnlineServices.Stats.OnQueryStatsCompleteCallbackInfo callbackInfo) =>
    {
        Debug.Log("Get stat return code: " + callbackInfo.ResultCode);
    });
```

FIGURE 29. The ‘QueryStat’ function.

If the query operation is successful, the requested stats will be cached locally. To access this information, the cached stats should be copied into variables to be used. The Stats API provide different functions to copy the stats into variables using stat name or stat index. If the game contains predefined variables specified for different stats, copying stats by name and assigned stats to the corresponding variables manually is recommended.

The ‘CopyStatByName’ function requires the user’s PUID and the name of the stat being copied, and returns a structure of type ‘Stat’. This structure contains the information related to the query operation, including the data value being queried.

```
Epic.OnlineServices.Stats.CopyStatByNameOptions copyOptions = new Epic.OnlineServices.Stats.CopyStatByNameOptions
{
    TargetUserId = EOSSDKComponent.s_ProductUserID,
    Name = "myStat"
};
Epic.OnlineServices.Stats.Stat? outStat = new Epic.OnlineServices.Stats.Stat();

Epic.OnlineServices.Result result = EOSSDKComponent.s_StatsInterface.CopyStatByName(ref copyOptions, out outStat);

if (result == Epic.OnlineServices.Result.Success)
{
    value = outStat.Value.Value;
}
```

FIGURE 30. Copying the locally cached stat.

As the data are cached on local memory storage, it is generally a good practice to release unused memory to reduce the program’s memory usage. While the C version of the SDK provides functions that handles the releasing of unused memory and requires developers to implement these functions manually, the C# version automatically handles this process through its wrapper’s marshalling code (Epic Developer Resources), thus no additional actions are required.

During development, developers should keep in mind that the Stats Interface has usage limitation to ensure reliability and availability for all users. According to the

Epic Developer Resources, the Stats system is limited to 500 stats per deployment, with each stat limited to 256 characters in name length, and a 3000 threshold for max amount of stats ingested per call. The Developer Resources also state that stat ingestion is the endpoint for submitting stat changes to the service, and has a per-user rate limit. This limit includes a 60 requests per minute and 500 stats per request for ingest calls, and a 100 requests per minute for querying calls for each user.

Alternative to using the Stats Interface for unlocking achievements, the game can also implement the Achievement Interface for manually unlocking the achievements using built in logic. This method allows developers to have more control over when the achievement unlocks, and provide the ability of implementing more complex achievement unlock conditions.

To begin interacting with the server through the Achievement Interface API, the list of achievements created through the Developer Portal should be queried to be used within the game's logic. The Achievement Interface API provides two query functions: a 'QueryDefinitions' function to retrieve all achievement-related information including name, localized text, icon, … ; and a 'QueryPlayerAchievements' that retrieve the list of achievements of a specific player, which includes achievement progress towards completion. Should the game require the achievement definitions to be displayed to users through in-game UI, the 'QueryDefinitions' should be used. To manage user's achievement progress and manually unlock achievements, the game should use the 'QueryPlayerAchievements' request.

The structure of the 'QueryPlayerAchievements' function is similar to the stat-querying function, containing a query option and a callback function to be executed after the query operation is completed. The query option only requires the current user's PUID to retrieve the corresponding achievement information.

```
Epic.OnlineServices.Achievements.QueryPlayerAchievementsOptions queryAchievementOptions =
    new Epic.OnlineServices.Achievements.QueryPlayerAchievementsOptions
    {
        LocalUserId = EOSSDKComponent.s_ProductUserID,
        TargetUserId = EOSSDKComponent.s_ProductUserID
    };

EOSSDKComponent.s_AchievementInterface.QueryPlayerAchievements(ref queryAchievementOptions, null,
    (ref Epic.OnlineServices.Achievements.OnQueryPlayerAchievementsCompleteCallbackInfo callbackInfo) =>
    {
        Debug.Log("Get achievement return code: " + callbackInfo.ResultCode);
    });
```

FIGURE 31. Querying player's achievement.

After successfully querying the player's achievements, the data will be cached locally and should be copied into different variables to be accessed through the game's code. The copied achievement data will be saved as a 'PlayerAchievement' structure, which contains various information on player's achievement progress, achievement's unlock time if it has been unlocked, and an array containing different stats linked to the achievement.

```
Epic.OnlineServices.Achievements.CopyPlayerAchievementByAchievementIdOptions queryAchievementIdOptions =
    new Epic.OnlineServices.Achievements.CopyPlayerAchievementByAchievementIdOptions
    {
        LocalUserId = EOSSDKComponent.s_ProductUserID,
        AchievementId = "MyAchievementID"
    };

Epic.OnlineServices.Achievements.PlayerAchievement? playerAchievement = new Epic.OnlineServices.Achievements.PlayerAchievement();

EOSSDKComponent.s_AchievementInterface.CopyPlayerAchievementByAchievementId(ref queryAchievementIdOptions, out playerAchievement);
```

FIGURE 32. Acquiring player's achievement data.

While querying the player's achievements will result in a list of achievements along with achievement progress and relevant information, the operation has an exception: achievements that are marked as hidden will not appear among the queried list. The hidden achievements will only appear if it has been unlocked prior to requesting the query call to the server.

When an achievement's unlock criteria have been met, the program can call the 'UnlockAchievements' function to manually unlock the achievement for the targeted player. The option for calling this function contains the current player's ID and a list of achievements ID to be unlocked.

```
Epic.OnlineServices.Achievements.UnlockAchievementsOptions unloackAchievementOptions =
    new Epic.OnlineServices.Achievements.UnlockAchievementsOptions
    {
        UserId = EOSSDKComponent.s_ProductUserID,
        AchievementIds = new Epic.OnlineServices.Utf8String[]
                        {
                            "MyFirstAchievementID",
                            "MySecondAchievementID"
                        },
    };
```

FIGURE 33. Option for unlocking achievements

Similar to many other asynchronous functions provided by the SDK, the ‘Unlock-Achievements’ function also provides a callback function that will be executed after the unlock achievement operation is completed. This callback function will provide the result code operation, along with all related information, that can be used to determine successful request to the server and handle invalid requests.

```
EOSSDKComponent.s_AchievementInterface.UnlockAchievements(ref unloackAchievementOptions, null,
    (ref Epic.OnlineServices.Achievements.OnUnlockAchievementsCompleteCallbackInfo callbackInfo) =>
    {
        Debug.Log("Unlock achievement return code: " + callbackInfo.ResultCode);
    });
```

FIGURE 34. Manually unlocking achievements.

The EOS Achievement Interface API also has usage limitations to ensure the service’s reliability and availability for all users. A product’s achievement system is limited to 1000 total number of achievements, with each achievement limited to a maximum of 3 linked stats. Apart from the system’s usage limitations, there are also per-user limitations which includes 10 query definitions requests per minute, 100 requests per minute for querying a specific achievement’s definition, 100 requests per minute for querying player’s achievements and 100 requests per minute for unlocking achievements.

4 TESTING

4.1 Testing criteria

Epic Games provides developers with a Developer Authentication Tool, which acts as a stand-in for the Epic Games Store Launcher during game development (Epic Developer Resources). The tool enables developer to remain logged in to the SDK's server, even after closing and relaunching the application, which saves time during testing the SDK's online features. For that reason, using this tool during testing phase is recommended to minimize unnecessary delays from relogging into the server. The Authentication Tool can also store multiple credentials at once, which allows developers to run multiple instances of the game for testing multiplayer features.

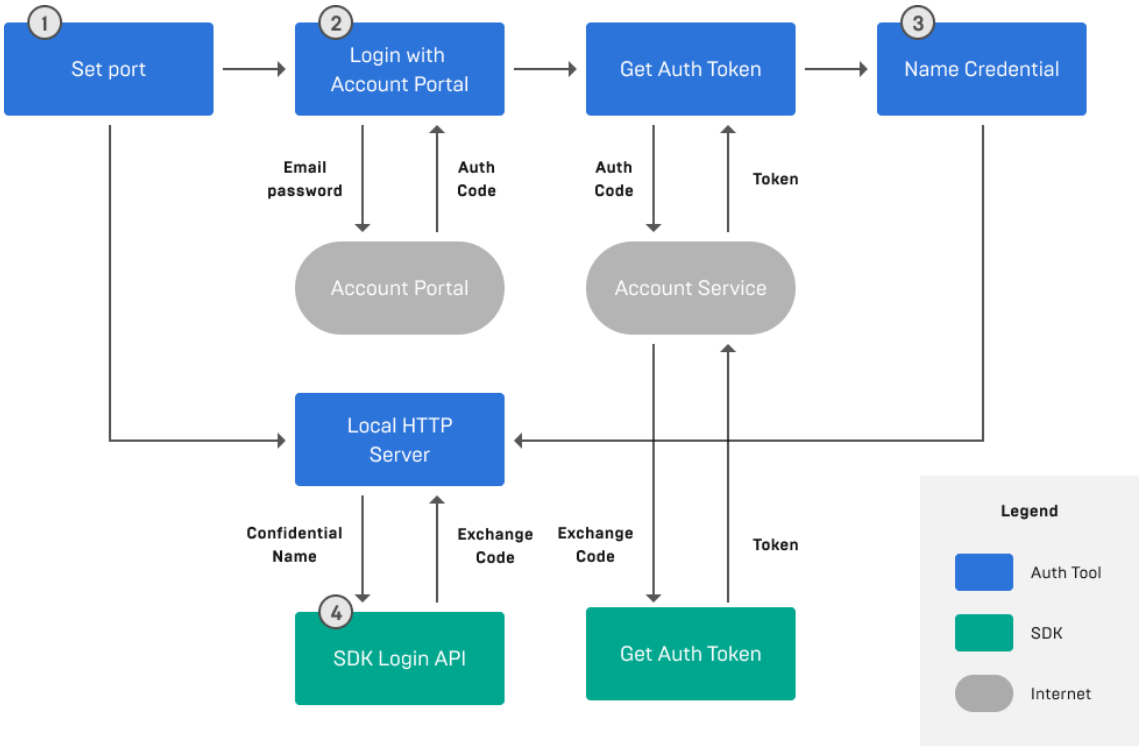

FIGURE 35. Developer Authentication Tool's workflow.
(Epic Developer Resources)

After setting up the listening port and the developer's credentials, the CredentialType used during Authentication should be changed to 'Developer' (see FIGURE 15), with an ID field set to the host and port the Authentication Tool is running on

(for example: ‘locahost:8080’) and the token field set to the name of the credential provided to the Authentication Tool.

The first testing scenario that should be carried out is the Epic Account Services Player Consent. Games that implement the Epic Account Services must acquire player consent to share data. If the game uses the correct CredentialType during login, players should be directed to grant consent the first time they launch the application. In some cases regarding games delivered through the Epic Games Store, the consent may be granted at the time of purchase. To test the correct behaviour, developers must use a separated account that is not associated with the Dev Portal organization and proceed to launch the game from the launcher. The user consent should only be prompted at initial game launch, and after completed, any subsequent game launches should allow players to enter the game without having to complete the user consent again.

Another important functionality that should be tested is the Connect Interface’s Access Token (or the PUID). This token is used throughout the game’s lifecycle to access the SDK’s online functionalities, thus maintaining the token’s validity is essential to keep the SDK running correctly. The token should be renewed on an hourly basis (see FIGURE 21), and failure to renew the token will result in invalid service calls. After the game is launched and run for 60 minutes, should the token renewal functionality is implemented correctly, any attempts to use EOS functionalities should result in successful calls, and a token expiration notification should appear within the game’s local log file.

Developers should also test the game’s achievement feature by proceeding the gameplay to the point of unlocking an achievement. Should the feature be implemented correctly, an achievement unlock notification should appear on the game’s screen, and the game’s social overlay should display the list of achievements along with the newly unlocked ones. The game data associated with the specific testing account can also be viewed through the Developer Portal to ensure that data are being sent to the server correctly.

After being thoroughly tested, the application can then be sent to Epic Games for evaluation. To be sent for evaluation, the application must be push to the Stage

sandbox (see Figure 2). After being staged, the application will be tested further to ensure that the functionalities are working correctly and communicating with the EOS server. Passing this stage, the application will receive the green light to be pushed to the Live sandbox, where the application can be published to the users.

4.2 Error handling

Most of the SDK's functions provide a result code upon completing execution which describes the request's success and any possible errors encountered. The result code is an enum type that includes common error values and interface-specific errors (Epic Developer Resources). The common error values will take the form of 'ErrorCode', while interface-specific errors will be displayed as 'Interface.ErrorCode'. Any errors encountered throughout the game's lifecycle will be displayed in the game's local log file, where developers can keep track on the game's state and execution. Setting the log level can help significantly as the log file will display more detailed messages (see FIGURE 7).

A successful function execution should return a result code of 'Success', or numeric code 0. Any other result codes indicate errors during function execution. A common possible error encountered during development is the 'InvalidParameters', which describes invalid parameters passed into the function. Similar errors include 'InvalidCredentials' and 'InvalidUser' can also be encountered if the application's function calls are handled incorrectly. These errors require developers to check function implementations and make corrections where applicable.

The application can also encounter errors due to incorrectly calling requests to the server. A 'TooManyRequest' errors indicates that the application is making more calls than the limited amount the server permits, or a 'AlreadyPending' result code specify a request was made while the previous request has not completed its execution. These problems can be handled by setting a delay time on function calls, or implementing checks to ensure that only one request is made at once.

In some cases, the application can also encounter errors due to connection or server errors. Common errors include ‘UnrecognizeResponse’, which describe a failure in parsing or recognizing the server’s response, or a ‘ServieFailure’ that indicates a failure with the backend service. A ‘OperationWillRetry’ specifies that connectivity to the backend server is impaired, and the SDK will retry the request.

5 CONCLUSIONS

The implementation of an achievement system in game applications has become a common practice among the game industry. By adding this feature, the application tends to receive more interest and engagement from users, which extends the application's longevity and maintain an active game community. Overall, the achievement system has become a significant part of a game application, contributing to the game's success.

The distribution of the EOSSDK provides developers with a mean of implementing Epic Game's online services within their product, expanding the game's functionalities and connect players to the Epic Game's ecosystem. Through the SDK's API services, an achievement system can be developed and managed that utilized the EOS functionalities and features. Games published through the Epic Games Store can make use of the fully developed Epic Games ecosystem and the EOSSDK to deliver engaging features to their products that increase the product's popularity and user experience.

This thesis explored the implementation of EOSSDK within game application, discovered the SDK functionalities and usage, and delivered a more in-depth sight into the Epic Online Services system. A detailed integration of the SDK was described to provide insight into the SDK's workflow and features, and the SDK testing criteria was also explored to gain understanding into the development process.

In conclusion, the integration of the EOSSDK provided developers with a mean of interacting with the EOS server, allowing more advanced and engaging features to be implemented and delivered to the end-users, elevating the gameplay elements and user experiences.

REFERENCES

Epic Developer Resources Documentation.
https://dev.epicgames.com/docs

Kishna, Rajen. 2021. Introduction to Epic Online Services (EOS)
https://dev.epicgames.com/en-US/news/introduction-to-epic-online-services-eos

Unity Documentation
https://docs.unity.com/

Howarth, Josh. 2023. How many gamers are there? (New 2023 statistics)
https://explodingtopics.com/blog/number-of-gamers

Hamari, Juho & Eranti, Veikko. 2011. Framework for Designing and Evaluating Game Achievements.
http://www.digra.org/digital-library/publications/framework-for-designing-and-evaluating-game-achievements/

Wilde, Tyler. 2016. The life of a top Steam achievement hunter.
https://www.pcgamer.com/steam-achievement-hunters/

Hilliard, Kyle. 2013. Activision Badges – The Original Gaming Achievement.
https://www.gameinformer.com/b/features/archive/2013/10/26/activision-badges-the-original-gaming-achievement.aspx

Hon, Adrian. 2022. How achievements took over the video game industry.
https://www.fastcompany.com/90786023/how-achievements-took-over-the-video-game-industry

Galli, Luca & Fraternali, Piero. 2014. Achievement System Explained.
https://www.researchgate.net/publication/273455882_Achievement_Systems_Explained

Jakobsson, Mikael. 2011. The Achievement Machine: Understanding Xbox 360 Achievements in Gaming Practices.
https://gamestudies.org/1101/articles/jakobsson

Nystrom, Robert. 2011. Game Programming Patterns.
https://gameprogrammingpatterns.com/

Marlin, Chistopher D., 1979. Coroutines: A Programming Methodology, a Language Design and an Implementation.
https://hdl.handle.net/2440/21027

Roscoe, A. W. 1997. The Theory and Practice of Concurrency.
https://www.cs.ox.ac.uk/publications/publication953-abstract.html