



# 게임콘텐츠2

## Section 2 : PlayFab



# What is PlayFab ?

- PlayFab is
  - a complete backend platform for live games with managed game services, real-time analytics, and LiveOps.
  - These features can help you boost your revenue and increase player engagement while cutting costs.
  - PlayFab's backend services reduce the barriers to launch for game developers, offering both large and small studios cost-effective development solutions that scale with their games and help them engage, retain, and monetize players.
  - PlayFab enables developers to use the intelligent cloud to build and operate games, analyze gaming data and improve overall gaming experiences..

- Using PlayFab you can:
  - Remove the challenges of building, managing, and running low latency multiplayer servers at scale with a complete back-end solution.
  - Use multiple forms of built-in authentication to track players across devices.
  - Provide players the ability to communicate via in-game chat with full transcription and translation services.
  - Deepen player engagement with sophisticated LiveOps tools by creating rich player segments and running A/B experiments.
  - Quickly create leaderboards and run content experimentation to deepen player engagement.
  - Accelerate growth with economy services that let you create and track virtual currencies, manage stores of items, and process payments.

# PlayFab ?

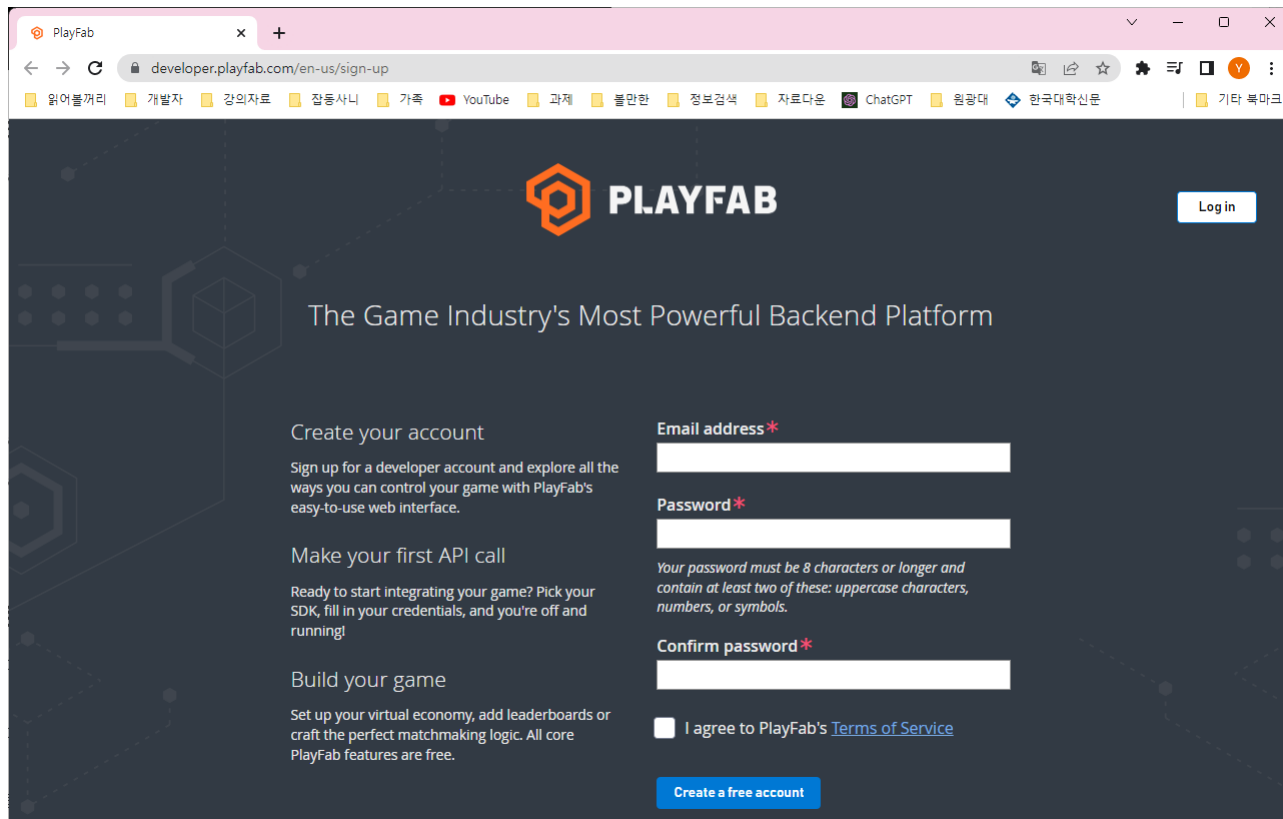
- PlayFab 은 ?
  - 관리되는 게임 서비스, 실시간 분석 및 LiveOps를 사용하는 라이브 게임을 위한 완벽한 백 엔드 플랫폼
  - 수익을 높이고 플레이어 참여를 높이는 동시에 비용을 절감하는 데 도움
  - PlayFab의 백 엔드 서비스는 게임과 함께 확장되고 플레이어의 참여, 유지, 수익 창출에 도움이 되는 비용 효율적인 개발 솔루션인 대규모 스튜디오와 소규모 스튜디오를 제공하여 게임 개발자의 진입 장벽을 낮춘다.
  - PlayFab을 활용하면, 개발자가 인텔리전트 클라우드를 사용하여 게임을 빌드 및 운영하고, 게임 데이터를 분석하고, 전반적인 게임 환경을 개선
  -

- PlayFab 기능

- 완벽한 백 엔드 솔루션을 사용하여 대기 시간이 짧은 멀티 플레이어 서버를 대규모로 빌드, 관리, 실행하는 문제를 제거합니다.
- 여러 형태의 기본 제공 인증을 사용하여 여러 디바이스에서 플레이어를 추적합니다.
- 플레이어에게 전체 전사 및 번역 서비스로 인게임 채팅을 통해 의사 소통할 수 있는 기능을 제공합니다.
- 풍부한 플레이어 세그먼트를 만들고 A/B 실험을 실행하여 정교한 LiveOps 도구로 플레이어 참여를 강화하세요.
- 신속하게 순위표를 만들고 콘텐츠 실험을 실행하여 플레이어 참여를 강화합니다.
- 가상 통화를 만들어서 추적하고, 아이템 저장소를 관리하고, 결제를 처리할 수 있는 경제 서비스로 성장을 가속화하세요.

- PlayFab
  - Requirements
    - A PlayFab Developer Account
    - An installed copy of the Unity Editor
      - The PlayFab Unity3D SDK supports Unity Editor version 5.3 (released December 2015) and higher.
    - A Unity Project (any on the following)
      - A brand new project
      - A guided tutorial project
      - An existing project
    - The PlayFab Unity3D SDK

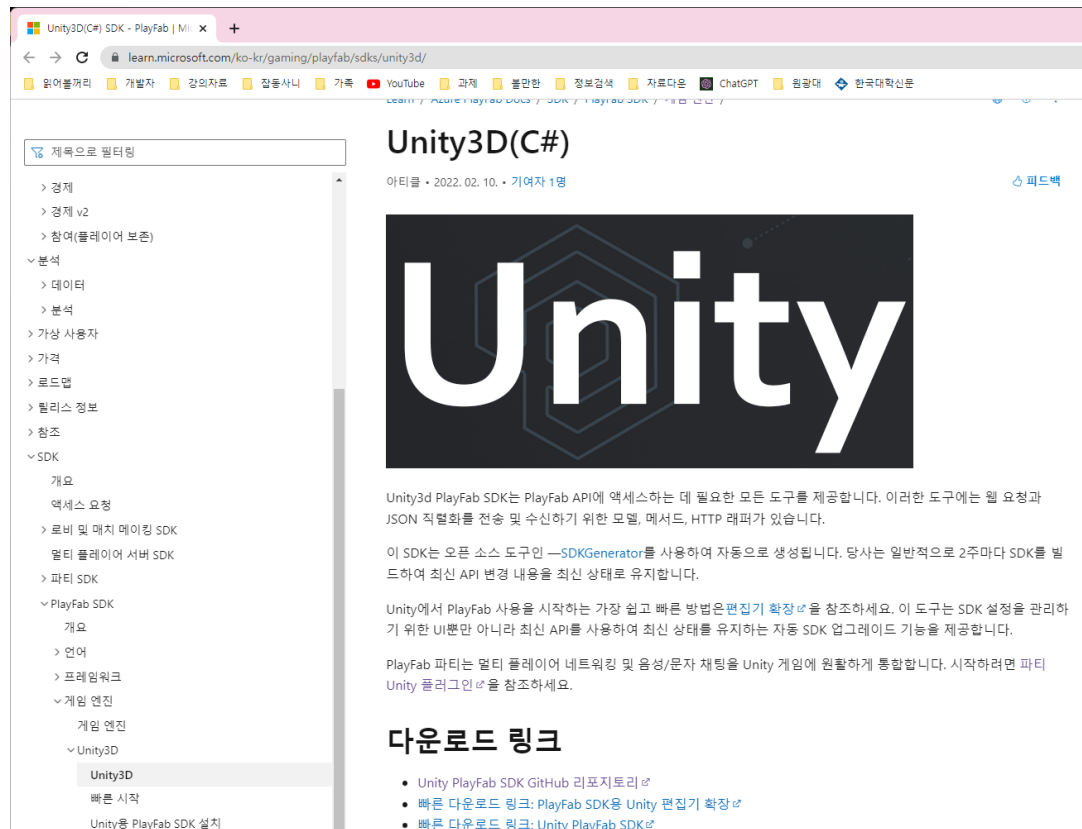
- PlayFab 계정 등록
  - <https://developer.playfab.com/en-us/sign-up>



The screenshot shows a web browser window with the URL `developer.playfab.com/en-us/sign-up`. The page features the PlayFab logo and the tagline "The Game Industry's Most Powerful Backend Platform". On the right, there is a "Login" button. The main content area is divided into two columns. The left column contains three sections: "Create your account" (with a description of signing up for a developer account), "Make your first API call" (with instructions on integrating a game), and "Build your game" (with details on setting up a virtual economy). The right column contains a sign-up form with fields for "Email address\*", "Password\*", and "Confirm password\*", each followed by a text input box. Below the password fields, there is a checkbox for "I agree to PlayFab's [Terms of Service](#)". At the bottom right of the form is a blue button labeled "Create a free account".

- Download and install PlayFab SDK

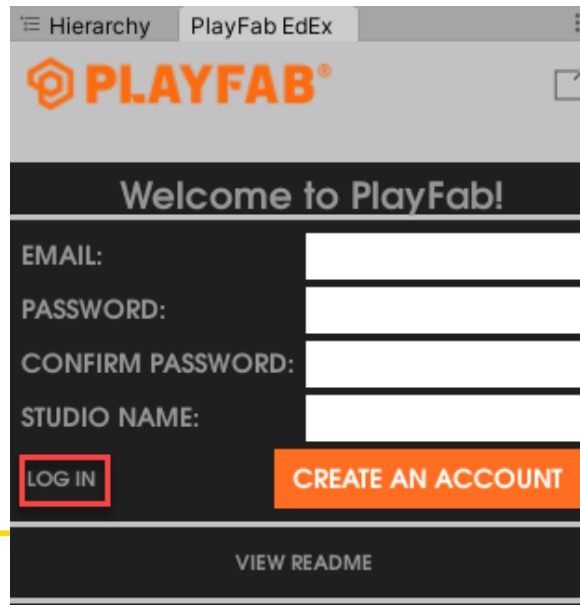
- <https://learn.microsoft.com/ko-kr/gaming/playfab/sdks/unity3d/>



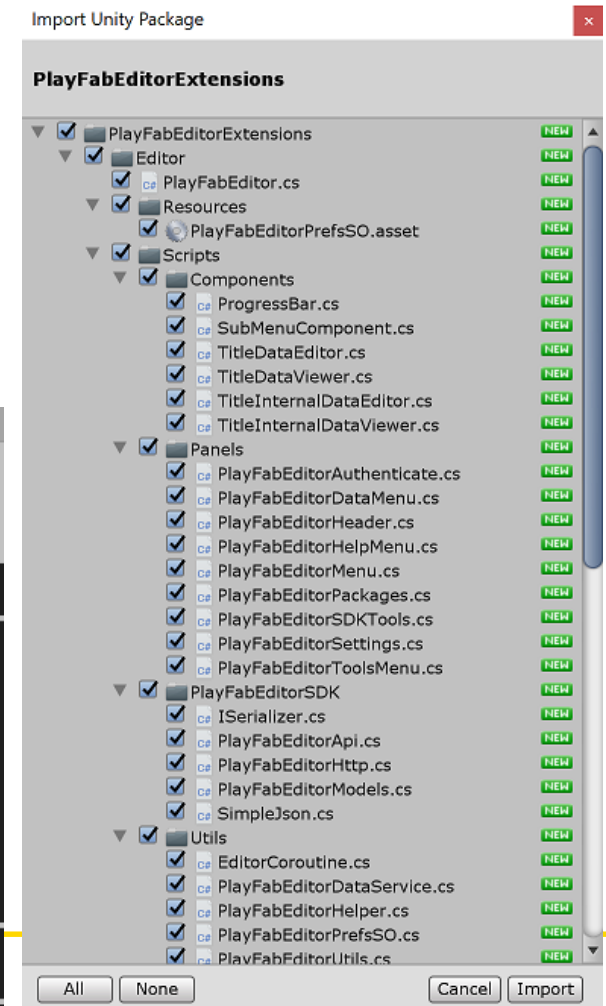


## – Installation

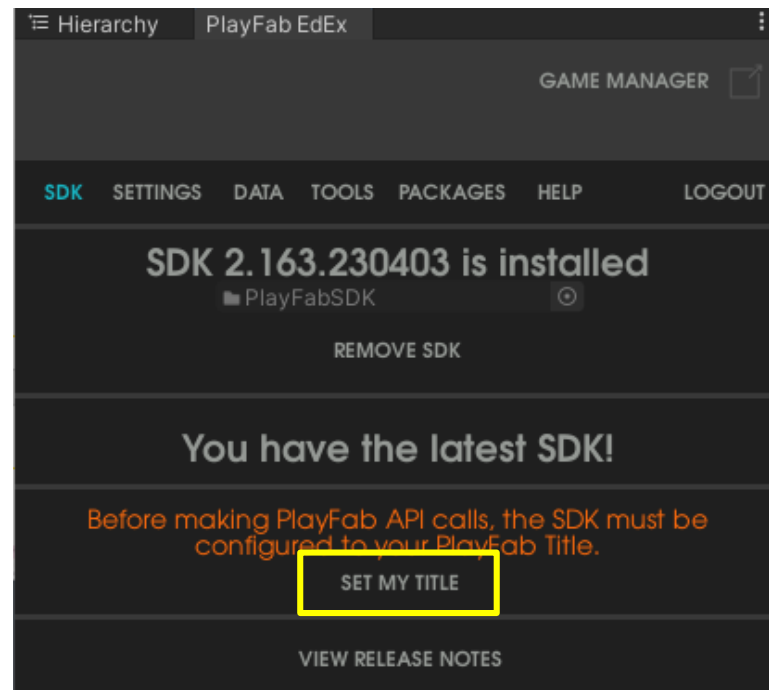
- Open your Unity project
- Import unity package in the Unity Editor
- When the import has completed, the PlayFab Unity Editor Extensions panel should open automatically  
If not open, click [Windows – PlayFab – Editor Extensions]
- Log in with your PlayFab username and password



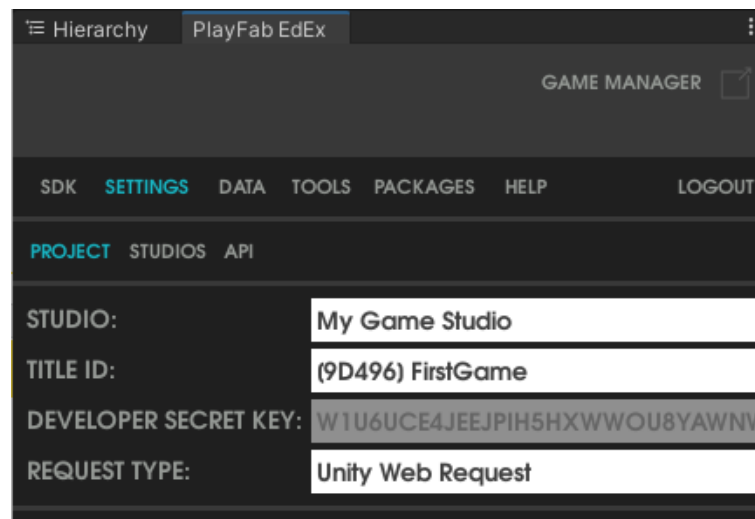
The image shows the PlayFab login interface. At the top, there's a 'Hierarchy' tab and a 'PlayFab EdEx' tab. Below the PlayFab logo, it says 'Welcome to PlayFab!'. There are four input fields: 'EMAIL:', 'PASSWORD:', 'CONFIRM PASSWORD:', and 'STUDIO NAME:'. Below these fields are two buttons: 'LOG IN' (highlighted with a red box) and 'CREATE AN ACCOUNT'. At the bottom, there is a 'VIEW README' link.



- After logging in, extension displays the SDK installation dialog
- Select `Install PlayFab SDK` to automatically import the SDK into your project or upgrade the version that is currently installed
- Before you make an API call, you must specify the Title to receive the call in the `PlayFab Title Settings`.



- Select `TitleID` entry to enter Titles associated with the selected studio
- The Developer Secret Key is automatically set to the default secret key for the Title.



- Create C# Script, named `PlayFabFirstCall`, in the Scripts folder
- Create Empty Object in the Hierarchy panel, then Drag and drop the C# script to the object.

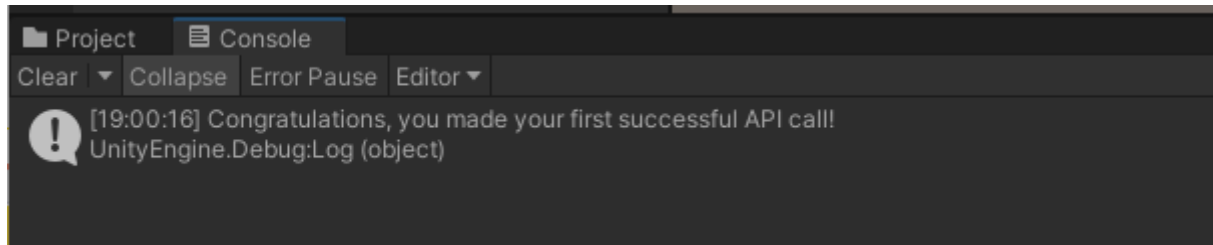
```
using UnityEngine;
using PlayFab;
using PlayFab.ClientModels;

public class PlayFabFirstCall : MonoBehaviour
{
    public void Start() {
        if (string.IsNullOrEmpty(PlayFabSettings.staticSettings.TitleId)) {
            /* Please change the titleId below to your own titleId from PlayFab Game Manager.
            If you have already set the value in the Editor Extensions, this can be skipped. */
            PlayFabSettings.staticSettings.TitleId = "42";
        }
        var request = new LoginWithCustomIDRequest { CustomId = "GettingStartedGuide", CreateAccount = true };
        PlayFabClientAPI.LoginWithCustomID(request, OnLoginSuccess, OnLoginFailure);
    }

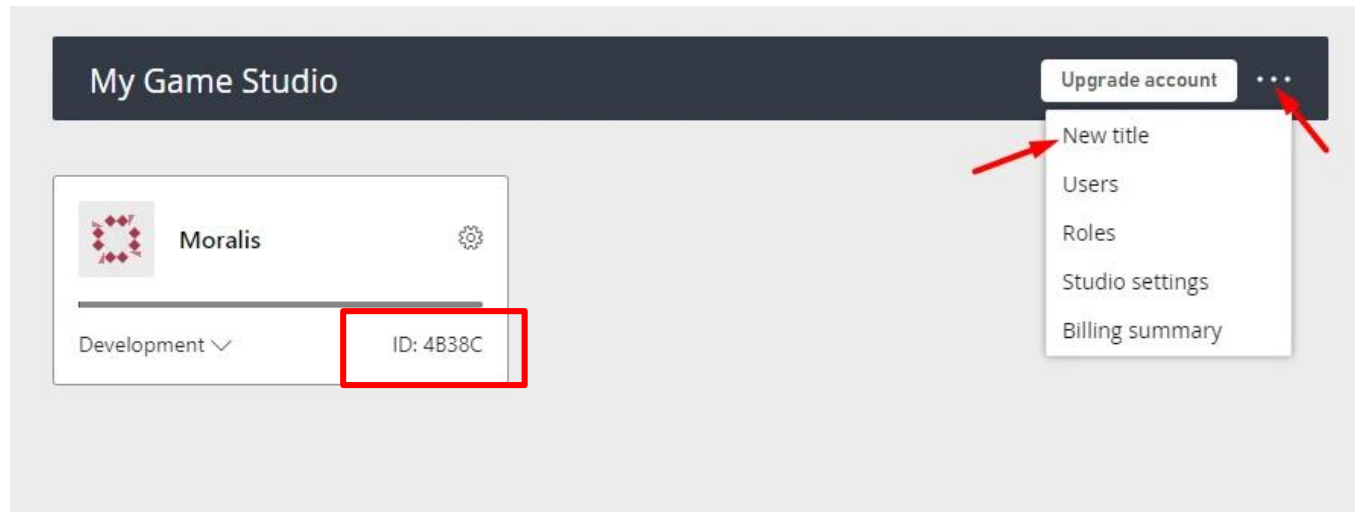
    private void OnLoginSuccess(LoginResult result) {
        Debug.Log("Congratulations, you made your first successful API call!");
    }

    private void OnLoginFailure(PlayFabError error) {
        Debug.LogWarning("Something went wrong with your first API call. :(");
        Debug.LogError("Here's some debug information:");
        Debug.LogError(error.GenerateErrorReport());
    }
}
```

- Be sure to save all files and return to the Unity Editor
- Press the Play button at the top of the editor.

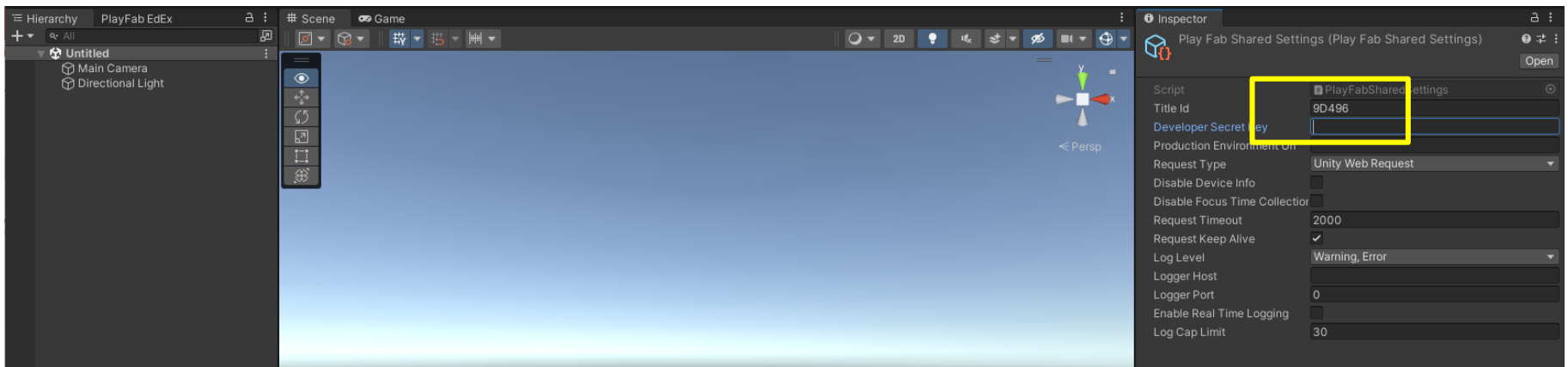


- Signup and Login to PlayFab Server
  - Create a new title on PlayFab



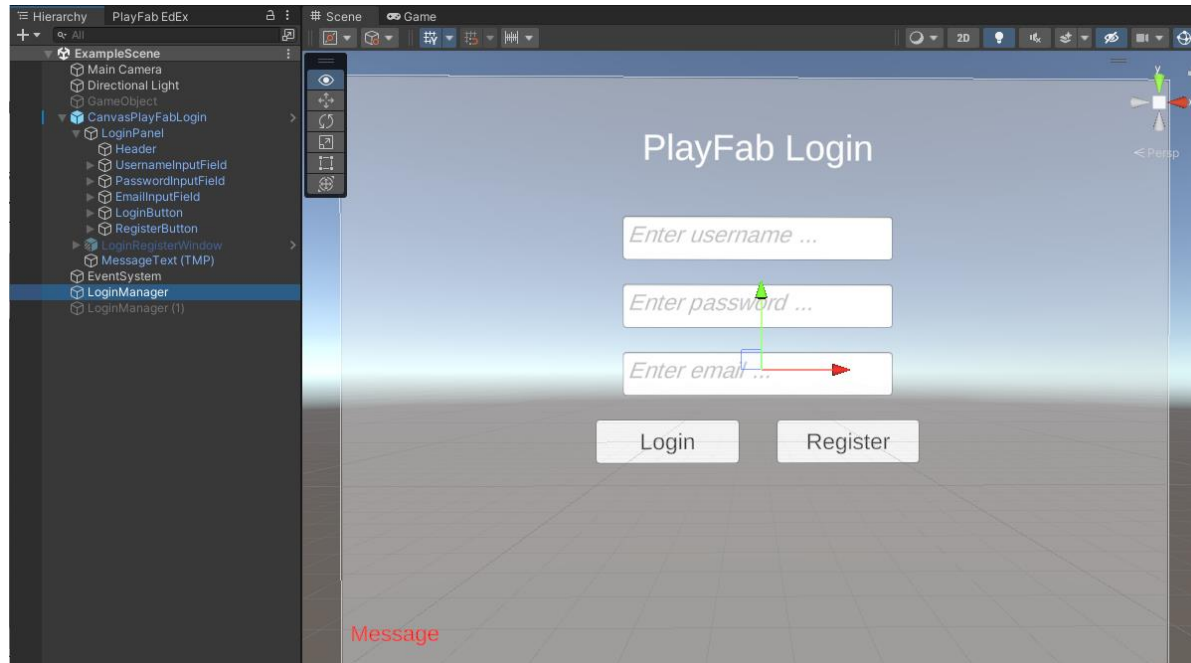
## – PlayFab Setting

- [PlayFab – MakePlayFabSharedSettings]



Assets/PlayFabSDK/Shared/Public/Resources/PlayFabSharedSettings.asset

## – Login (or Signup) UI 생성



- TPro를 사용할 경우, "TextMeshPro" Package Import, using TPro;
- UI – Legacy를 사용할 경우, using UnityEngine.UI;



- Create C# Script, named `PlayFabLogin`, in the Scripts folder
- Create Empty Object in the Hierarchy panel (named `LoginManager`), then Drag and drop the C# script to the object.

```
using UnityEngine;
using PlayFab;
using PlayFab.ClientModels;
using TMPPro;

public class PlayFabLogin : MonoBehaviour
{
    public TMP_InputField inputUserID;
    public TMP_InputField inputPassword;
    public TMP_InputField inputEmail;
    public TMP_Text displayMessage;

    private string username;
    private string password;
    private string email;

    // Start is called before the first frame update
    void Start() { PlayFabSettings.TitleId = "9D496"; }

    // Update is called once per frame
    void Update() { }

    public void UsernameValueChanged() { username = inputUserID.text.ToString(); }
    public void PasswordValueChanged() { password = inputPassword.text.ToString(); }
    public void EmailValueChanged() { email = inputEmail.text.ToString(); }
}
```

```

public class PlayFabLogin : MonoBehaviour {

    ...

    public void Login() {
        var request = new LoginWithPlayFabRequest { Username = username, Password = password };
        PlayFabClientAPI.LoginWithPlayFab(request, OnLoginSuccess, OnLoginFailure);
    }

    private void OnLoginSuccess(LoginResult result) {
        displayMessage.text = "Login successfully";
        StartGame();
    }

    private void OnLoginFailure(PlayFabError error) {
        Debug.LogWarning(error.GenerateErrorReport());
        displayMessage.text = error.GenerateErrorReport();
    }

    public void Register() {
        var request = new RegisterPlayFabUserRequest { Username = username, Password = password, Email = email };
        PlayFabClientAPI.RegisterPlayFabUser(request, RegisterSuccess, RegisterFailure);
    }

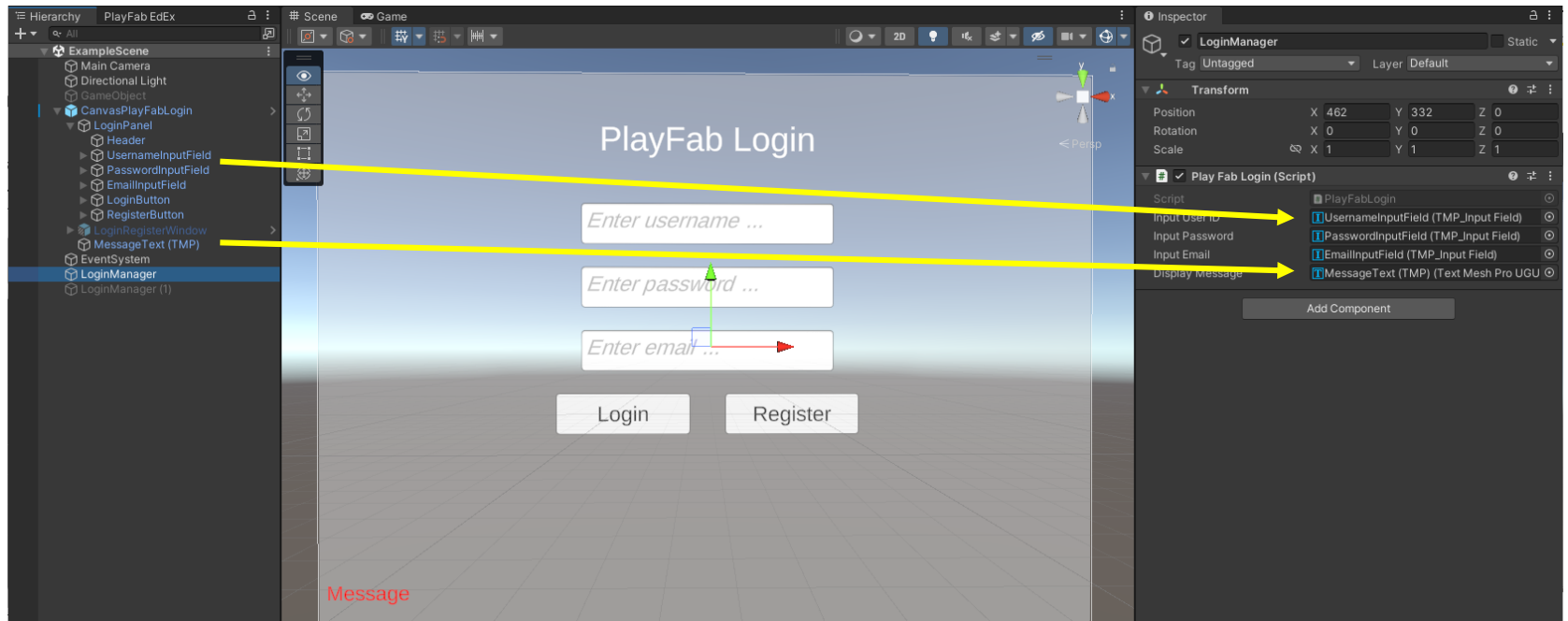
    private void RegisterSuccess(RegisterPlayFabUserResult result) {
        displayMessage.text = "Signup successfully";
    }

    private void RegisterFailure(PlayFabError error) {
        Debug.LogWarning(error.GenerateErrorReport());
        displayMessage.text = error.GenerateErrorReport();
    }

    void StartGame() { Debug.Log("Now, start the game, enjoy it"); }

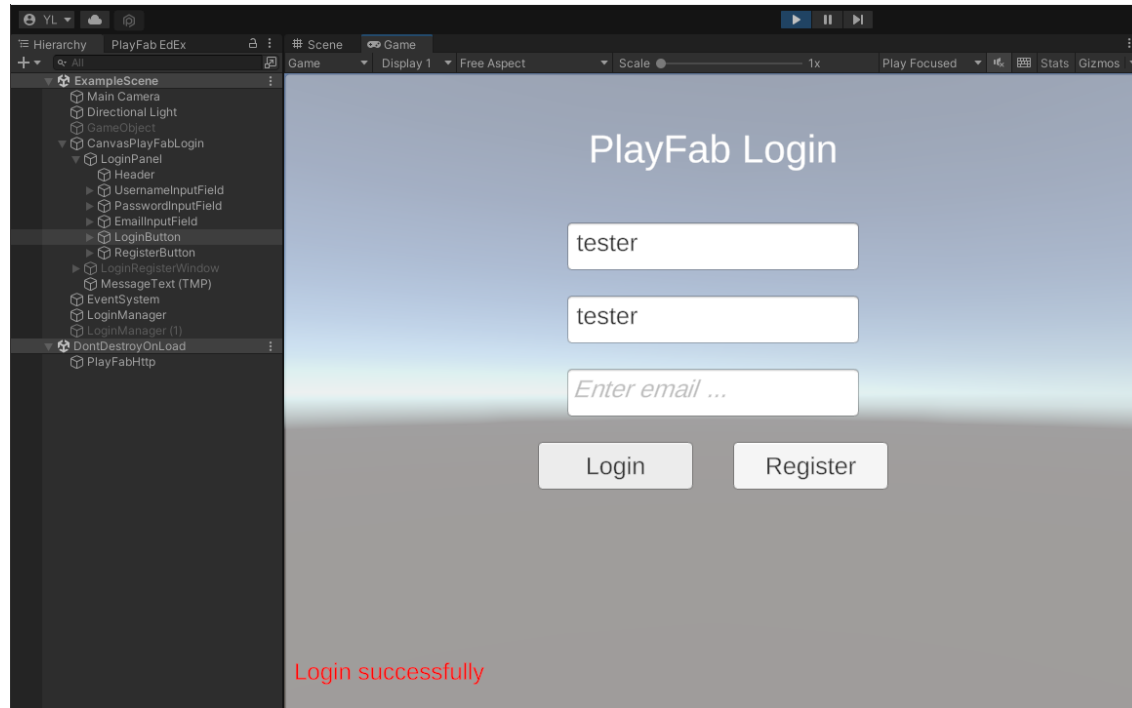
}

```



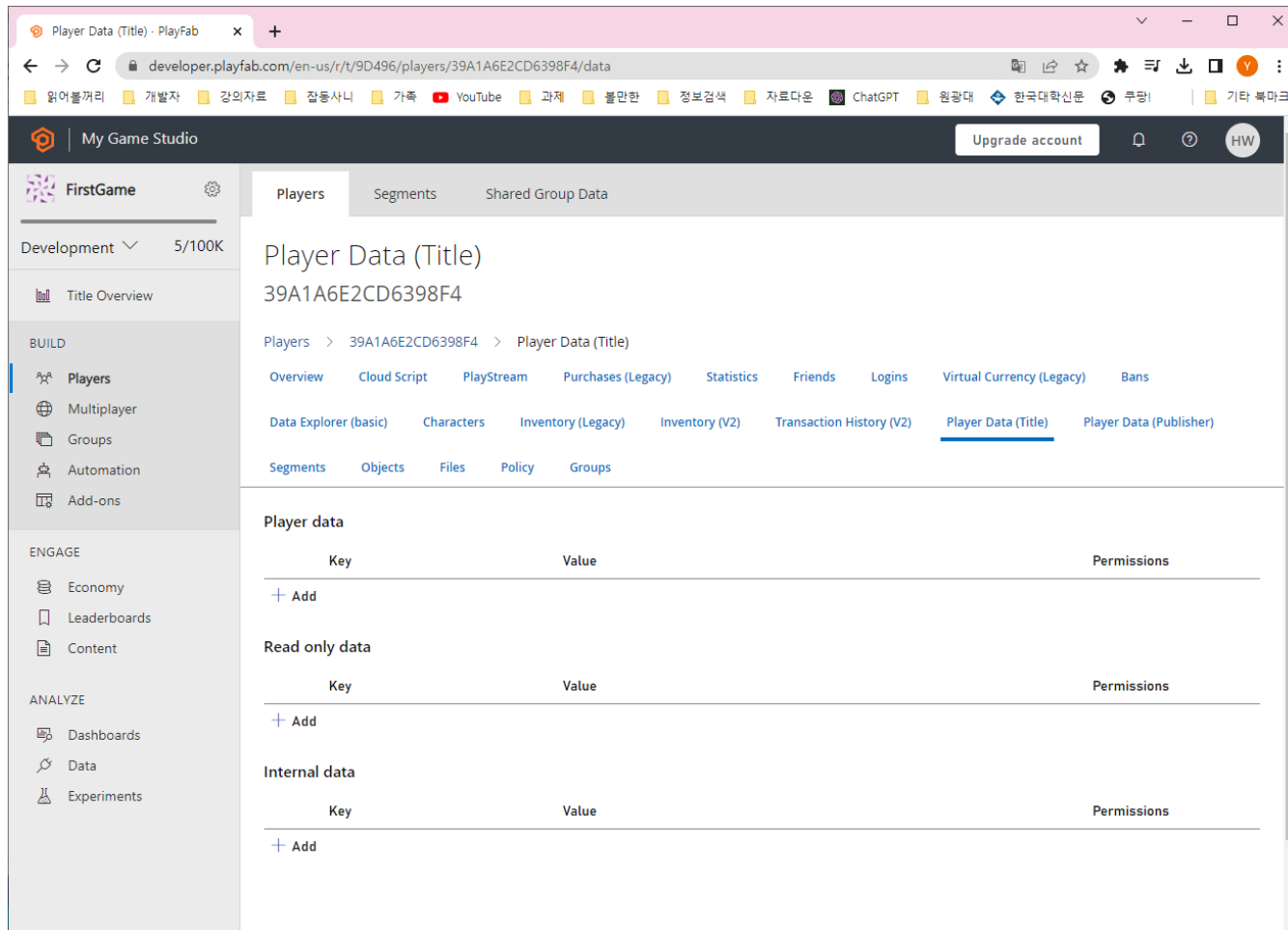
- Button의 `OnClick()`, InputField의 `OnValueChanged()` 함수 설정

- Press the Play button



- 오류 메시지들을 확인

- 데이터 처리



# What is PlayFab Data ?

- PlayFab Data is
  - a set of tools for data analytics, storage, processing, and exports.
  - The features are Title data, **Players data**, Characters data, Groups data, along with data management and provision features like Entities, Content delivery network and Webhooks.
  - Players data, Characters data, Groups data are saved which manage the store settings, game save state, or other data.
  - Title data manages the remote configurations of the game as key-value pairs.

- PlayFab Data is

- PlayFab 데이터는 데이터 분석, 스토리지, 처리 및 내보내기를 위한 도구 세트
- 타이틀 데이터, **플레이어 데이터**, 캐릭터 데이터 및 그룹 데이터 기능과 엔터티, 콘텐츠 배달 네트워크, 웹훅 같은 데이터 관리 및 프로비전 기능을 제공
- Store 설정, 게임 저장 상태 또는 기타 데이터를 관리하는 플레이어 데이터, 캐릭터 데이터, 그룹 데이터를 저장
- 타이틀 데이터는 게임의 원격 구성을 키-값 쌍으로 관리

- Player Data is
  - information about a player that is stored to the PlayFab service that you can share across multiple devices and multiple games.
  - PlayFab provides two ways to store player data:
    - **Entities**: Allows you to store data in objects and files across Players, Characters, and Groups.
    - **Player Data/UserData**: Allows you to store Key/Value pair data for players.
  - To provide the most flexibility and best performance, we recommended that all new titles use Entity objects.
  - In the PlayFab APIs, the function names use the term UserData. In the Game Manager, this concept is described as Player Data. They are identical, and interchangeable.



- There are three modes of access to player data:
  - **Client**: This is player data that is available to your title client to read and update. You use the client APIs `UpdateUserData` to create, update, or delete and `GetUserData` to read data for the player.
  - **Read Only**: This is player data that is created or updated by your server. Your title client can read, but not update, this data. You use the server API `UpdateUserReadOnlyData` to create, update, or delete and the client API `GetUserReadOnlyData` to read title-specific data for the player. This data is visible to the player, but can only be modified by the server.
  - **Internal**: This is player data that is only available to your server. You use the server APIs `UpdateUserInternalData` to create, update, or delete and `GetUserInternalData` to read title-specific data for the player. This data is server-only, and cannot be seen by the client.

- 플레이어 데이터는
  - 여러 디바이스와 여러 게임에서 공유할 수 있는 PlayFab 서비스에 저장된 플레이어에 대한 정보
  - 2가지 플레이어 데이터 저장 방법을 제공
    - 엔터티: 개체의 데이터와 플레이어, 캐릭터, 그룹 전반의 파일을 저장
    - 플레이어 데이터/UserData: 플레이어의 키/값 쌍 데이터를 저장
  - 최고의 유연성과 성능을 제공하려면, ‘모든’ 새 타이틀에서 엔터티 개체를 사용하는 것이 좋음
  - PlayFab API에서 함수 이름은 UserData 라는 용어를 사용
    - 게임 관리자에서 이 개념은 플레이어 데이터로 설명
    - 두 개념은 동일하며, 서로 바꿔서 사용이 가능

– 3가지 플레이어 데이터의 액세스 모드

- **클라이언트(Client)**: 타이틀 클라이언트가 읽고 업데이트할 수 있는 플레이어 데이터. 클라이언트 API UpdateUserData를 사용하여 플레이어 데이터를 생성, 업데이트 또는 삭제하고, GetUserData를 사용하여 플레이어 데이터를 읽음
- **읽기 전용(Read Only)**: 서버에서 만들거나 업데이트하는 플레이어 데이터. 타이틀 클라이언트는 이 데이터를 읽을 수 있지만, 업데이트는 할 수 없음. 서버 API UpdateUserReadOnlyData를 사용하여 플레이어의 타이틀 관련 데이터를 생성, 업데이트 또는 삭제하고, 클라이언트 API GetUserReadOnlyData를 사용하여 읽음. 이 데이터는 플레이어가 볼 수 있지만, 서버에서만 수정할 수 있음
- **내부(Internal)**: 서버에서만 사용할 수 있는 플레이어 데이터. 서버 API UpdateUserInternalData를 사용하여 플레이어의 타이틀 관련 데이터를 생성, 업데이트 또는 삭제하고, 클라이언트 API GetUserInternalData를 사용하여 읽음. 이 데이터는 서버 전용이며, 클라이언트에서는 볼 수 없음.

- Quickstart : Set and get player data

- Player data를 설정하려면, PlayFabLogin 클래스에 SetUserData() 메소드를 추가
- SetUserData는 UpdateUserData()를 사용하여 로그인한 플레이어의 Player data를 만들거나 업데이트

```
using System.Collections.Generic;

...

void SetUserData() {
    PlayFabClientAPI.UpdateUserData(new UpdateUserDataRequest() {
        Data = new Dictionary<string, string>() {
            {"Ancestor", "Arthur"},
            {"Successor", "Fred"}
        }
    },
    result => Debug.Log("Successfully updated user data"),
    error => {
        Debug.Log("Got error setting user data Ancestor to Arthur");
        Debug.Log(error.GenerateErrorReport());
    });
}
```

- Player data를 가져오려면, PlayFabLogin 클래스에 GetUserData() 메소드를 추가
- GetUserData는 GetUserData()를 사용하여 지정된 플레이어 데이터를 검색

```
void GetUserData(string myPlayFabId) {  
    PlayFabClientAPI.GetUserData(new GetUserDataRequest() {  
        PlayFabId = myPlayFabId,  
        Keys = null  
    }, result => {  
        Debug.Log("Got user data:");  
        if (result.Data == null || !result.Data.ContainsKey("Ancestor")) Debug.Log("No Ancestor");  
        else Debug.Log("Ancestor: "+result.Data["Ancestor"].Value);  
    }, (error) => {  
        Debug.Log("Got error retrieving user data:");  
        Debug.Log(error.GenerateErrorReport());  
    });  
}
```

- 어떤 데이터를 저장해야 하는가?

