LAMPIRAN

Lampiran 1: Source Code

AuthenticationManager.cs

Assets > Scripts > Entity > Singletons > Authentication > AuthenticationManager.cs > {} Authentication

```
using System.Collections;
using System.Collections.Generic;
using System.Text.RegularExpressions;
using System.Threading.Tasks;
using UnityEngine;
using UnityEngine.UI;
using Firebase.Auth;
using TMPro;
using Persistence;

namespace Authentication
{
    1 reference
    public class AuthenticationManager : Singleton<AuthenticationManager>
    {
        [Header("Login")]
        1 reference
        [SerializeField] private TMP_InputField loginEmailText;
        1 reference
        [SerializeField] private TMP_InputField loginPasswordText;
        1 reference
        [SerializeField] private Button loginButton;
        [Header("Register")]
        1 reference
        [SerializeField] private TMP_InputField registerEmailText;
        1 reference
        [SerializeField] private TMP_InputField registerPasswordText;
        1 reference
        [SerializeField] private TMP_InputField registerVerifyPasswordText;
        1 reference
        [SerializeField] private Button registerButton;

        3 references
        private FirebaseAuth auth;

    // Start is called before the first frame update
    0 references
    void Start()
    {
        auth = FirebaseAuth.DefaultInstance;
        loginButton.onClick.AddListener(Login);
        registerButton.onClick.AddListener(Register);
    }
```

```
1 reference
void Login()
{
    string email = loginEmailText.text;
    string password = loginPasswordText.text;

    // Validate text
    if (!IsValidEmail(email))
    {
        Debug.LogError("Email is invalid");
        return;
    }

    if (!IsValidPassword(password))
    {
        Debug.LogError("Password is invalid");
        return;
    }

    StartCoroutine(LoginCoroutine(email, password));
}
```

```
1 reference
void Register()
{
    string email = registerEmailText.text;
    string password = registerPasswordText.text;
    string verifyPassword = registerVerifyPasswordText.text;

    // Validate text
    if (!IsValidEmail(email))
    {
        Debug.LogError("Email is invalid");
        return;
    }

    if (!IsValidPassword(password))
    {
        Debug.LogError("Password is invalid");
        return;
    }

    if (verifyPassword != password)
    {
        Debug.LogError("Verify Password must match with Password");
        return;
    }


    StartCoroutine(RegisterCoroutine(email, password));
}
```

```
        1 reference
        IEnumerator RegisterCoroutine(string email, string password)
        {
            Task<FirebaseUser> task = auth.CreateUserWithEmailAndPasswordAsync(email, passwor

            yield return new WaitUntil(() => task.IsCompleted);

            if (task.Exception != null)
            {
                yield break;
            }

            Debug.Log("Successfully registered");
        }

        2 references
        bool IsValidEmail(string email)
        {
            return Regex.IsMatch(email, @"^[^@\s]+@[^@\s]+\.[^@\s]+$");
        }

        2 references
        bool IsValidPassword(string password)
        {
            return password.Length >= 8 && Regex.IsMatch(password, @"^[a-zA-Z0-9]+$");
        }
    }
}
```

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Newtonsoft.Json;

namespace Persistence
{
    [Serializable]
    5 references
    public class PlayerData
    {
        2 references
        public string userUID;
        1 reference
        public string username;
        2 references
        public int plasmids;
        3 references
        public Dictionary<string, bool> subjects;

        [JsonConstructor]
        1 reference
        public PlayerData(string userUID, string username, int plasmids, Dictionary<string, bool> subjects)
        {
            this.userUID = userUID;
            this.username = username;
            this.plasmids = plasmids;
            this.subjects = subjects;
        }
    }
}
```

```
using Newtonsoft.Json;
using System;
using System.Collections;
using System.Collections.Generic;
using System.Threading.Tasks;
using UnityEngine;
using Firebase.Database;

namespace Persistence
{
    2 references
    public class PersistenceManager : Singleton<PersistenceManager>
    {
        3 references
        private DatabaseReference databaseReference;

        5 references
        protected override void Awake()
        {
            base.Awake();
            DontDestroyOnLoad(gameObject);
        }

        // Start is called before the first frame update
        0 references
        void Start()
        {
            // Set up the database reference
            databaseReference = FirebaseDatabase.DefaultInstance.GetReference("users");
        }


    1 reference
    public IEnumerator SavePlayerData()
    {
        PlayerData playerData = Player.Instance.playerData;

        // Convert the PlayerData object to a JSON string
        string json = JsonConvert.SerializeObject(playerData);

        // Save the JSON string to the database
        Task task = databaseReference.Child(playerData.userUID).SetRawJsonValueAsync(json);

        // Wait for the database operation to complete
        yield return new WaitUntil(() => task.IsCompleted);

        // Check if the operation was successful
        if (task.Exception != null)
        {
            yield break;
        }
    }
```

```
1 reference
public IEnumerator LoadPlayerData(string userUID)
{
    // Get the player data in json string from the Realtime Database
    Task<DataSnapshot> task = databaseReference.Child(userUID).GetValueAsync();

    // Wait for the database operation to complete
    yield return new WaitUntil(() => task.IsCompleted);

    // Check if the operation was successful
    if (task.Exception != null)
    {
        yield break;
    }

    DataSnapshot snapshot = task.Result;

    // Check if the result of the operation is null (no data was found)
    if (snapshot.Value != null)
    {
        // Deserialize the snapshot's value as a PlayerData object using JsonConvert
        string json = snapshot.GetRawJsonValue();

        Player.Instance.playerData = JsonConvert.DeserializeObject<PlayerData>(json);
    }
    else
    {
        // Create new player data if there is none (user doesn't have player data in Realtime Database)
        string usernameTest = "New Player";
        int plasmidsTest = 100;
        Dictionary<string, bool> subjectsTest = new Dictionary<string, bool>
        {
            {"A001", true},
            {"D001", true},
            {"S001", true}
        };
        PlayerData playerData = new PlayerData(userUID, usernameTest, plasmidsTest, subjectsTest);
        Player.Instance.playerData = playerData;

        StartCoroutine(SavePlayerData());
    }

    // Load to Menu Scene
    LoadingManager.Instance.LoadScene((int)SceneIndices.Menu);
    }
}
```