Nakama Unity

> Unity client for Nakama server.

Nakama is an open-source server designed to power modern games and apps. Features include user accounts, chat, social, matchmaker, realtime multiplayer, and much more.

This client is built on the .NET client with extensions for Unity Engine. It requires the .NET4.6 scripting runtime version to be set in the editor.

NOTE: In Unity versions earlier than 2018 you must navigate to Edit -> Project Settings -> Player -> Configuration (subheading) to apply it.

Full documentation is online - https://heroiclabs.com/docs/unity-client-guide

Getting Started

You'll need to setup the server and database before you can connect with the client. The simplest way is to use Docker but have a look at the server documentation for other options.

1. Install and run the servers. Follow these instructions.
2. Download the client from the releases page and import it into your project. You can also download prebuilt binaries from the Asset Store.
3. Use the connection credentials to build a client object.

```
// using Nakama;
const string scheme = "http";
const string host = "127.0.0.1";
const int port = 7350;
const string serverKey = "defaultkey";
var client = new Client(scheme, host, port, serverKey);
```

Usage

The client object has many methods to execute various features in the server or open realtime socket connections with the server.

Authenticate

There's a variety of ways to authenticate with the server. Authentication can create a user if they don't already exist with those credentials. It's also easy to authenticate with a social profile from Google Play Games, Facebook, Game Center, etc.

```
var deviceId = SystemInfo.deviceUniqueIdentifier;
var session = await client.AuthenticateDeviceAsync(deviceId);
Debug.Log(session);
```

Sessions

When authenticated the server responds with an auth token (JWT) which contains useful properties and

gets deserialized into a `Session` object.

```
Debug.Log(session.AuthToken); // raw JWT token
Debug.LogFormat("Session user id: '{0}'", session.UserId);
Debug.LogFormat("Session user username: '{0}'", session.Username);
Debug.LogFormat("Session has expired: {0}", session.IsExpired);
Debug.LogFormat("Session expires at: {0}", session.ExpireTime); // in seconds.
```

It is recommended to store the auth token from the session and check at startup if it has expired. If the token has expired you must reauthenticate. The expiry time of the token can be changed as a setting in the server.

```
const string prefKeyName = "nakama.session";
ISession session;
var authToken = PlayerPrefs.GetString(prefKeyName);
if (string.IsNullOrEmpty(authToken) || (session = Session.Restore(authToken)).IsExpired)
{
    Debug.Log("Session has expired. Must reauthenticate!");
};
Debug.Log(session);
```

Requests

The client includes lots of builtin APIs for various features of the game server. These can be accessed with the async methods. It can also call custom logic as RPC functions on the server. These can also be executed with a socket object.

All requests are sent with a session object which authorizes the client.

```
var account = await client.GetAccountAsync(session);
Debug.LogFormat("User id: '{0}'", account.User.Id);
Debug.LogFormat("User username: '{0}'", account.User.Username);
Debug.LogFormat("Account virtual wallet: '{0}'", account.Wallet);
```

Socket

The client can create one or more sockets with the server. Each socket can have it's own event listeners registered for responses received from the server.

```
var socket = client.NewSocket();
socket.Connected += () => Debug.Log("Socket connected.");
socket.Closed += () => Debug.Log("Socket closed.");
await socket.ConnectAsync(session);
```

Unity WebGL

For WebGL builds you should switch the `IHttpAdapter` to use the `UnityWebRequestAdapter` and use the `NewSocket()` extension method to create the socket OR manually set the right `ISocketAdapter` per platform.

```
var client = new Client("defaultkey", UnityWebRequestAdapter.Instance);
var socket = client.NewSocket();

// or
#if UNITY_WEBGL && !UNITY_EDITOR
    ISocketAdapter adapter = new JsWebSocketAdapter();
#else
    ISocketAdapter adapter = new WebSocketAdapter();
#endif
var socket = Socket.From(client, adapter);
```

Contribute

The development roadmap is managed as GitHub issues and pull requests are welcome. If you're interested to enhance the code please open an issue to discuss the changes or drop in and discuss it in the community chat.

This project can be opened in Unity to create a ".unitypackage".

License