

Installation

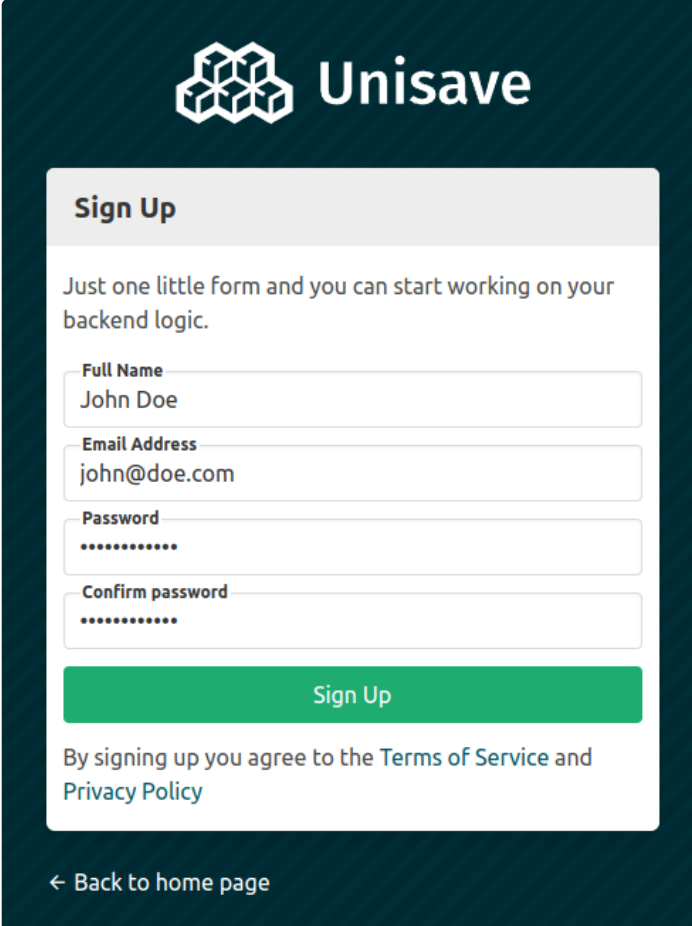
Getting started with Unisave is fast:

1. Create an account at unisave.cloud.
2. Import the [Unisave Asset](#) into your Unity project and connect it to the cloud.
3. Explore provided examples or start building.

And when you get stuck, feel free to ask on our [discord server](#), or send me an email to jirka@unisave.cloud.

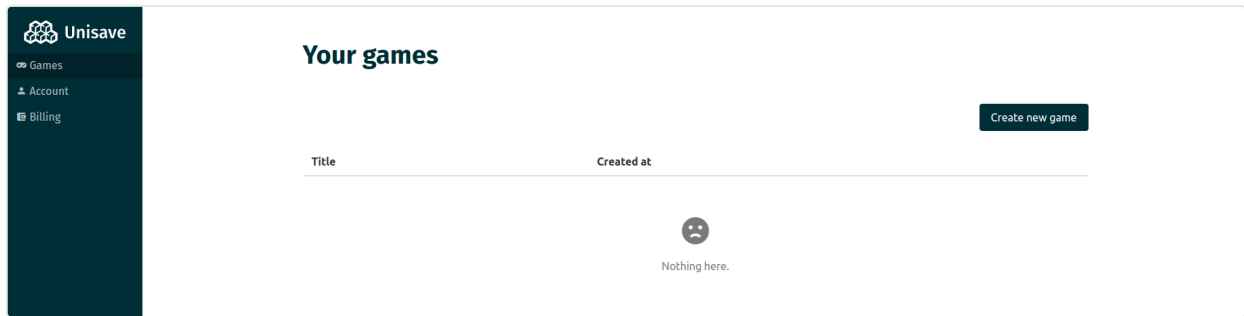
Create a cloud account

1. Go to <https://unisave.cloud/>.
2. Click `Create Account` and fill out the form:



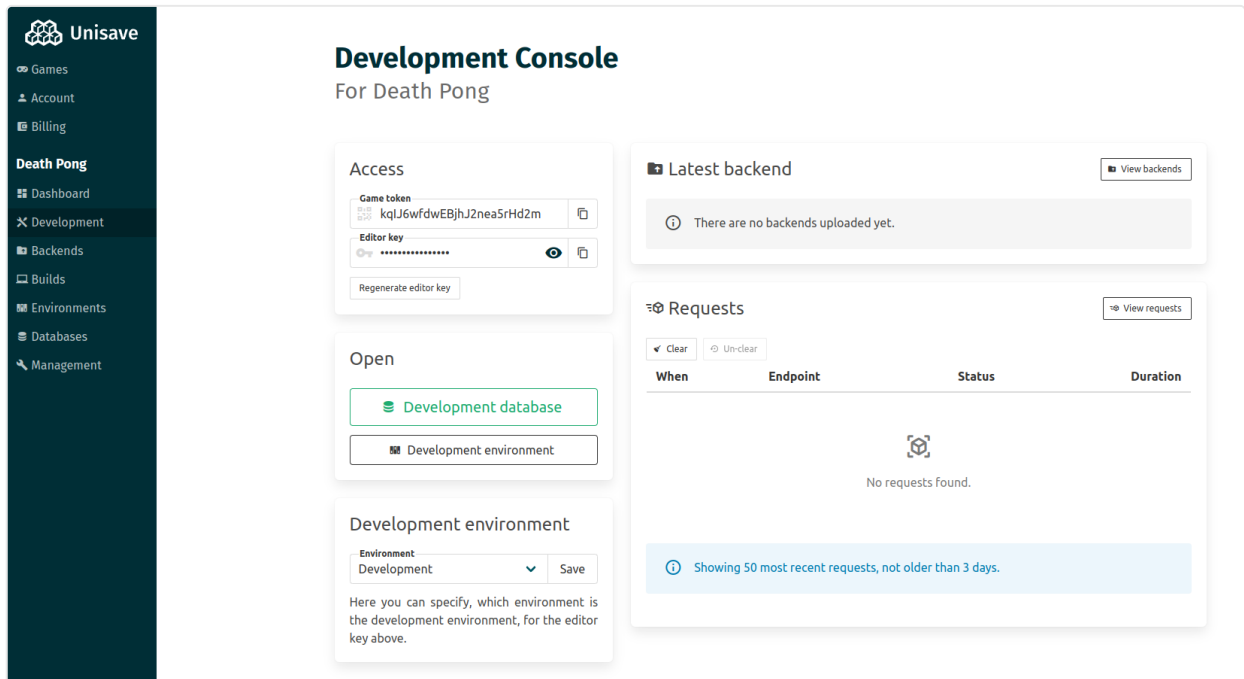
The screenshot shows a dark-themed web interface for the Unisave Sign Up page. At the top, the Unisave logo is displayed. Below it, a light gray box contains the title 'Sign Up' and a brief instruction: 'Just one little form and you can start working on your backend logic.' The form consists of four input fields: 'Full Name' (with the example 'John Doe'), 'Email Address' (with the example 'john@doe.com'), 'Password' (masked with dots), and 'Confirm password' (also masked with dots). A green 'Sign Up' button is positioned below the fields. At the bottom of the form, a line of text states: 'By signing up you agree to the [Terms of Service](#) and [Privacy Policy](#)'. Below the form box, a link with a left-pointing arrow reads '← Back to home page'.

3. You will be redirected to the app at <https://unisave.cloud/app>



4. Click **Create new game** and type in a name.

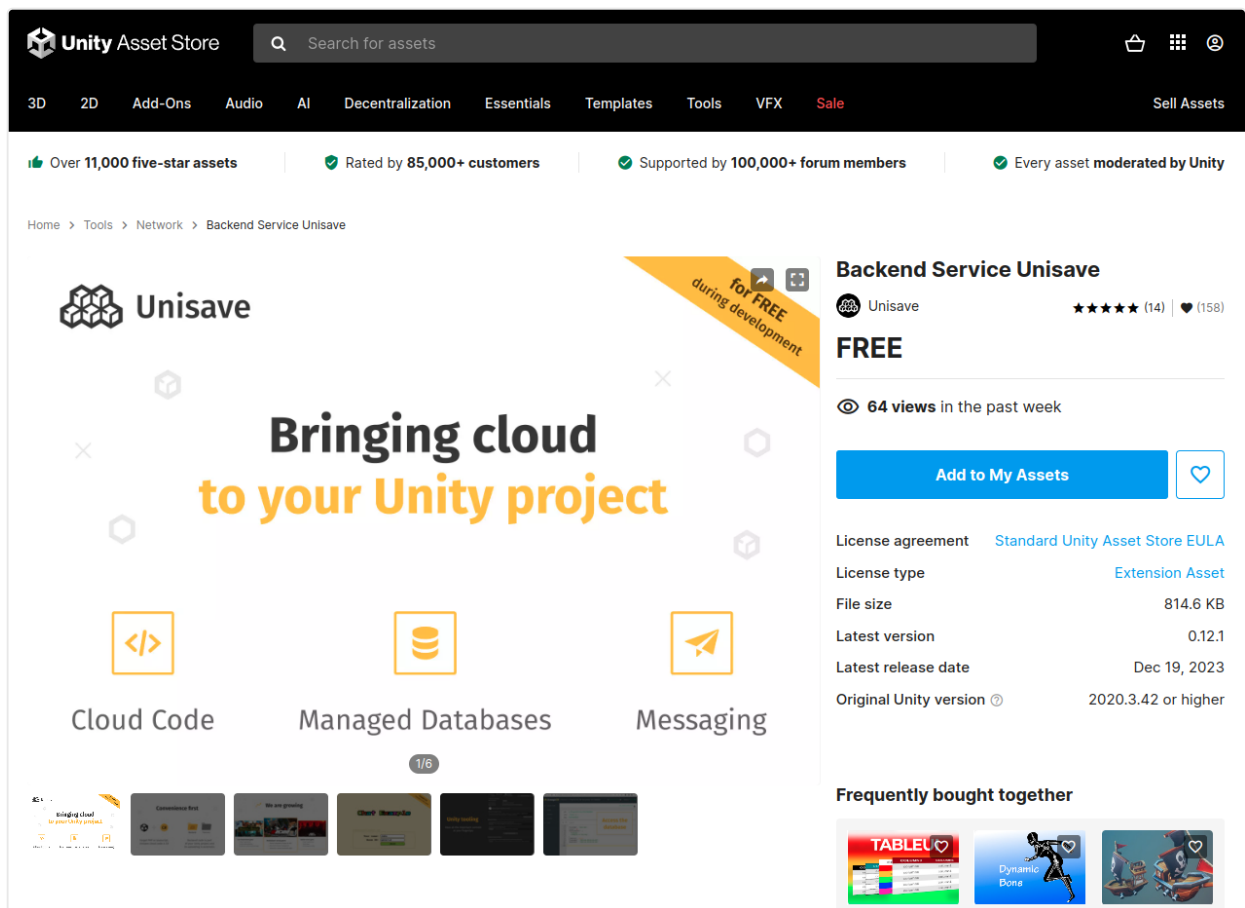
5. You will be redirected to the **Development Console** of your newly created game:



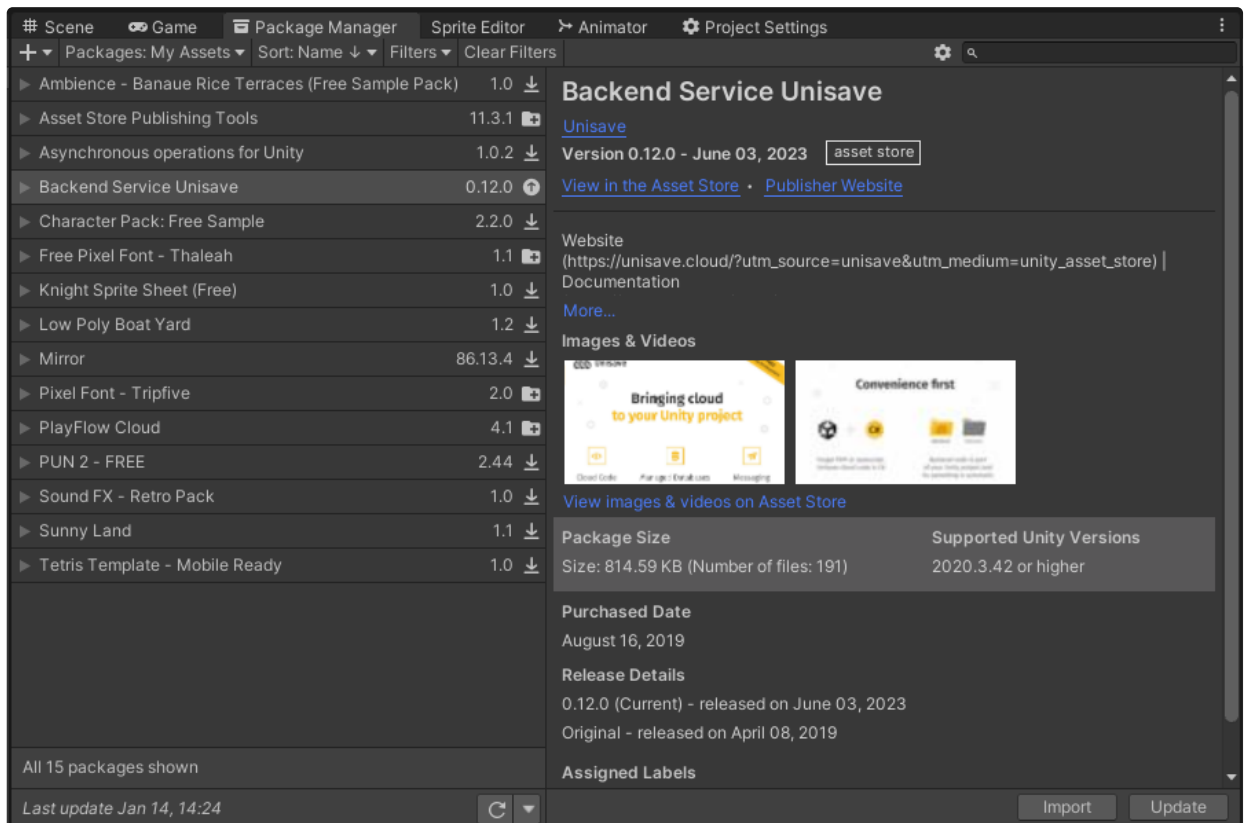
Keep this page open in your browser, you will need to copy the **Game Token** and **Editor Key** to your Unity project later - see [Connect project with the cloud](#).

Import the Unity asset

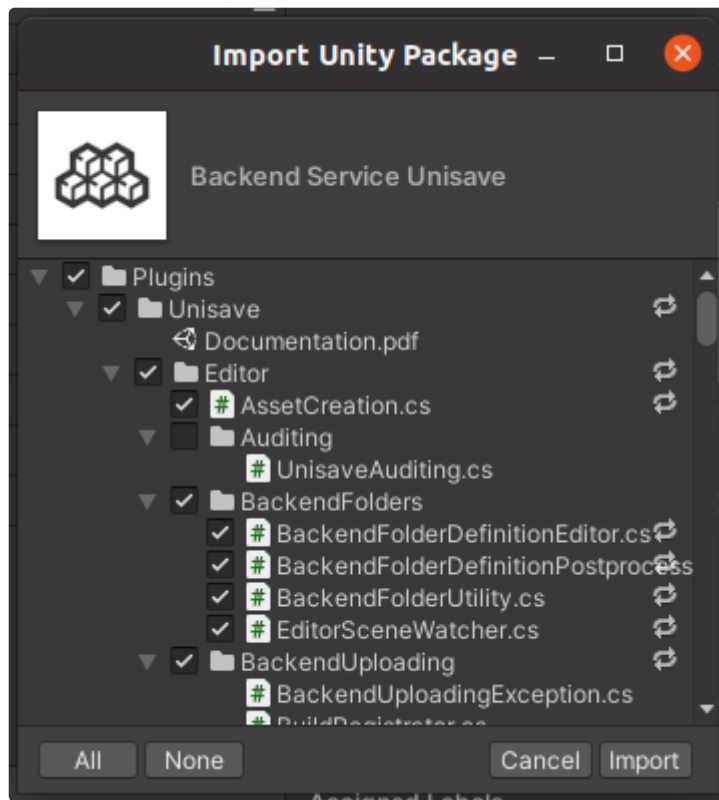
1. Open the [Unisave Asset](#) page and click the **Add to My Assets** button:



2. Open your game project in the Unity editor.
3. Open the **Package Manager** window by going to menu **Window > Package Manager**.
4. Open **Backend Service Unisave** asset, click on the **Download** button and then the **Import** button in the lower-left corner:

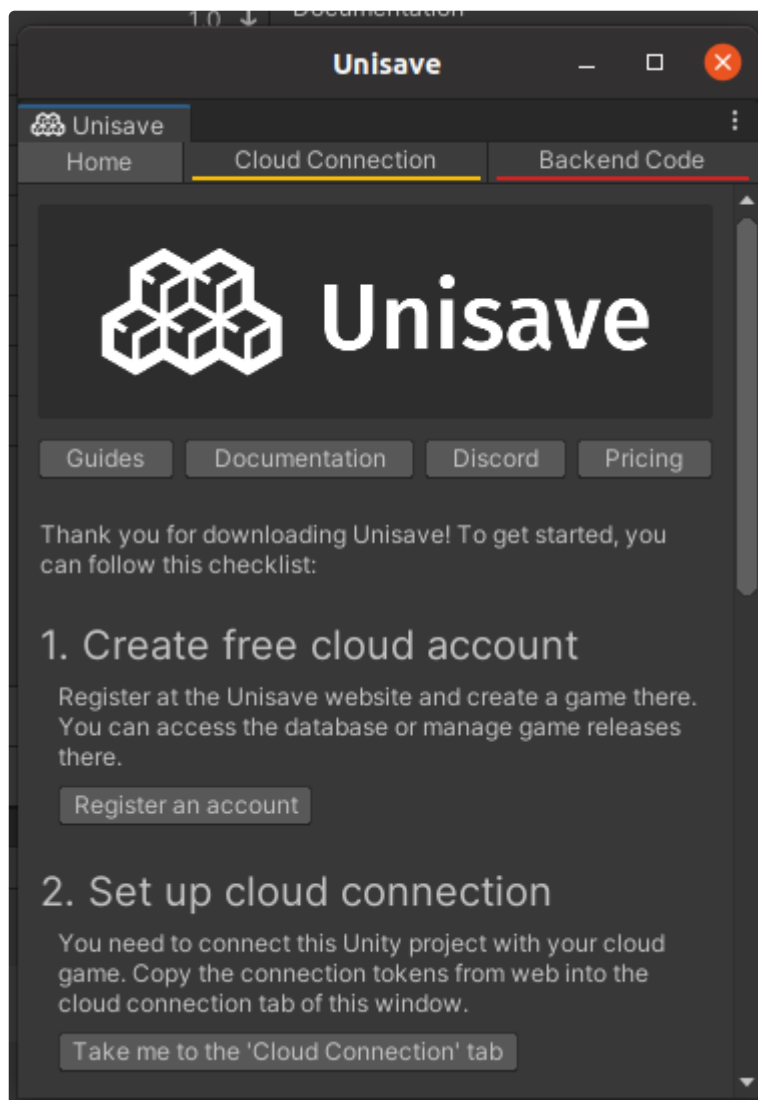


5. Import all the files:

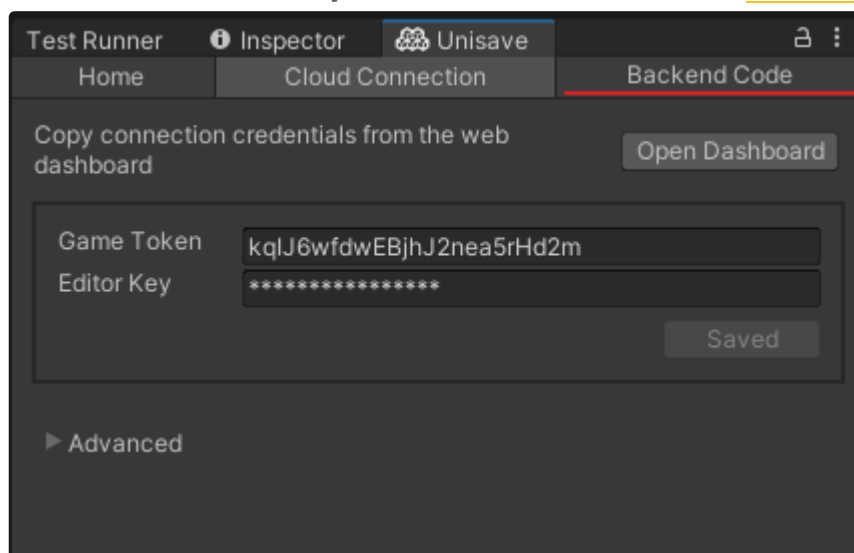


Connect project with the cloud

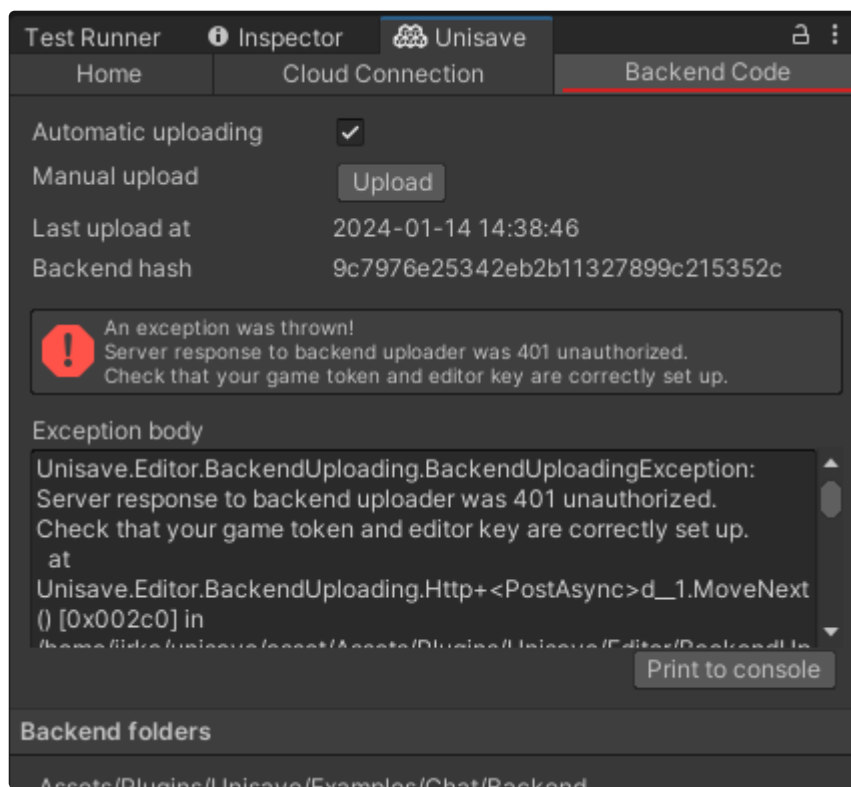
1. When the import finishes, the **Unisave Window** should open. If not, open it from the menu `Tools > Unisave > Unisave Window`:



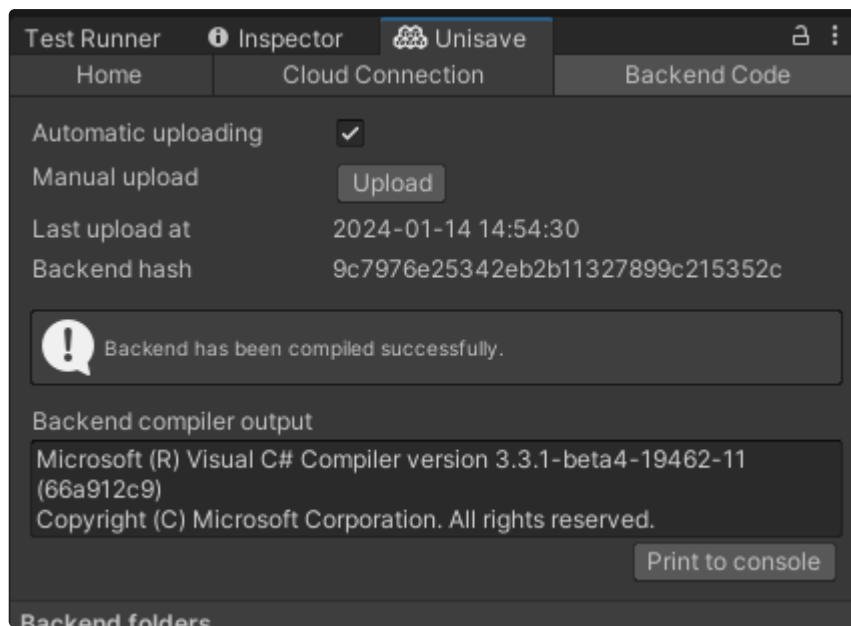
2. Click on the **Cloud Connection** tab and fill out the Game Token and Editor Key from the cloud **Development Console** website (see [Create a cloud account](#)):



3. The backend code (the code of your game's backend server) should be uploaded automatically, but sometimes this system gets confused (especially during configuration changes). In these cases you can go to the **Backend Code** tab, and click the manual **Upload** button:



4. In this case the manual upload worked and the backend code was successfully compiled. Now the automatic upload system should work, and if not, you will be notified in the Unity console. If the error persists, read the error message - you can see that (in this case) the error was due to the missing token and key.



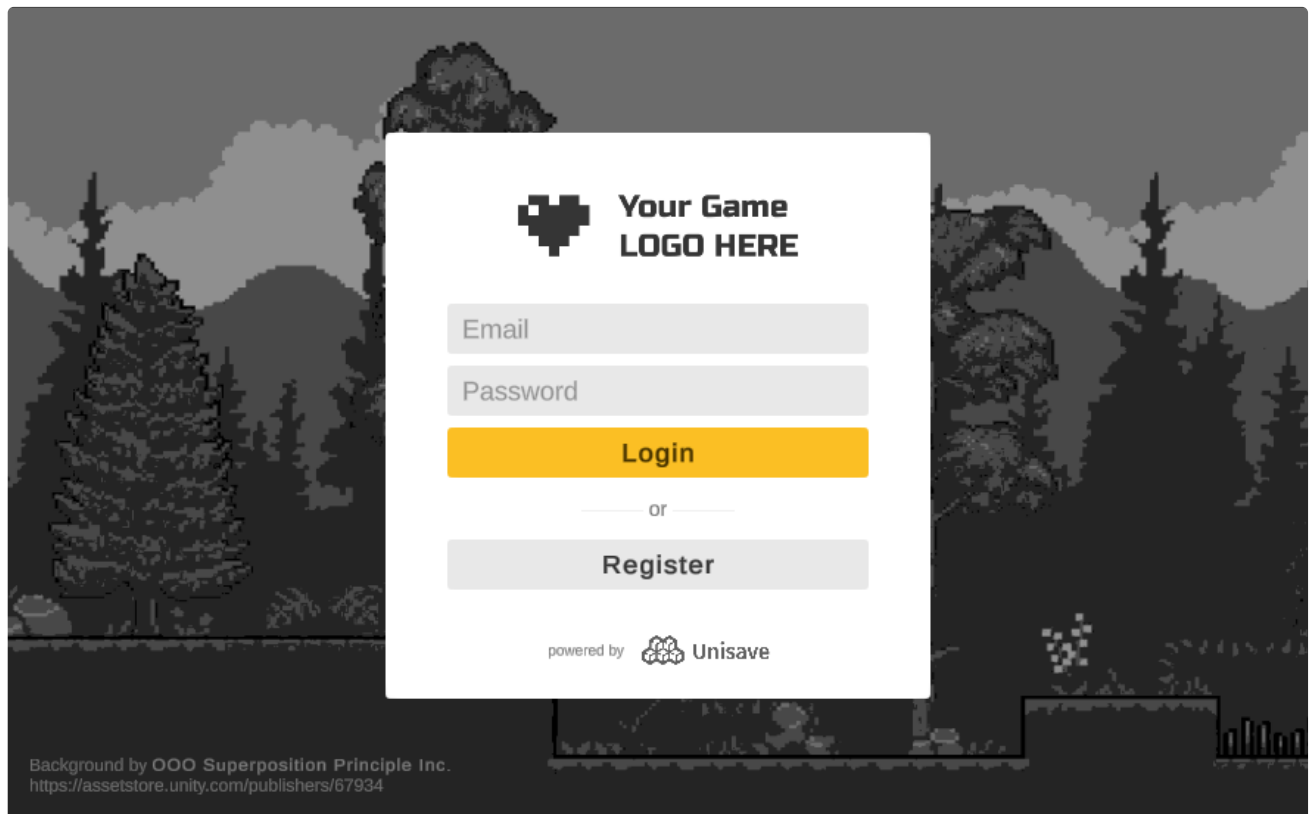
Explore examples

Now you probably have a specific problem in mind, that you wish Unisave can solve for you. Your actions now differ based on that problem. You can look at the provided examples, but it's likely your problem is very specific and there will not be an example scene for it.

Assets > Plugins > Unisave > Examples

- Chat
- EmailAuthentication

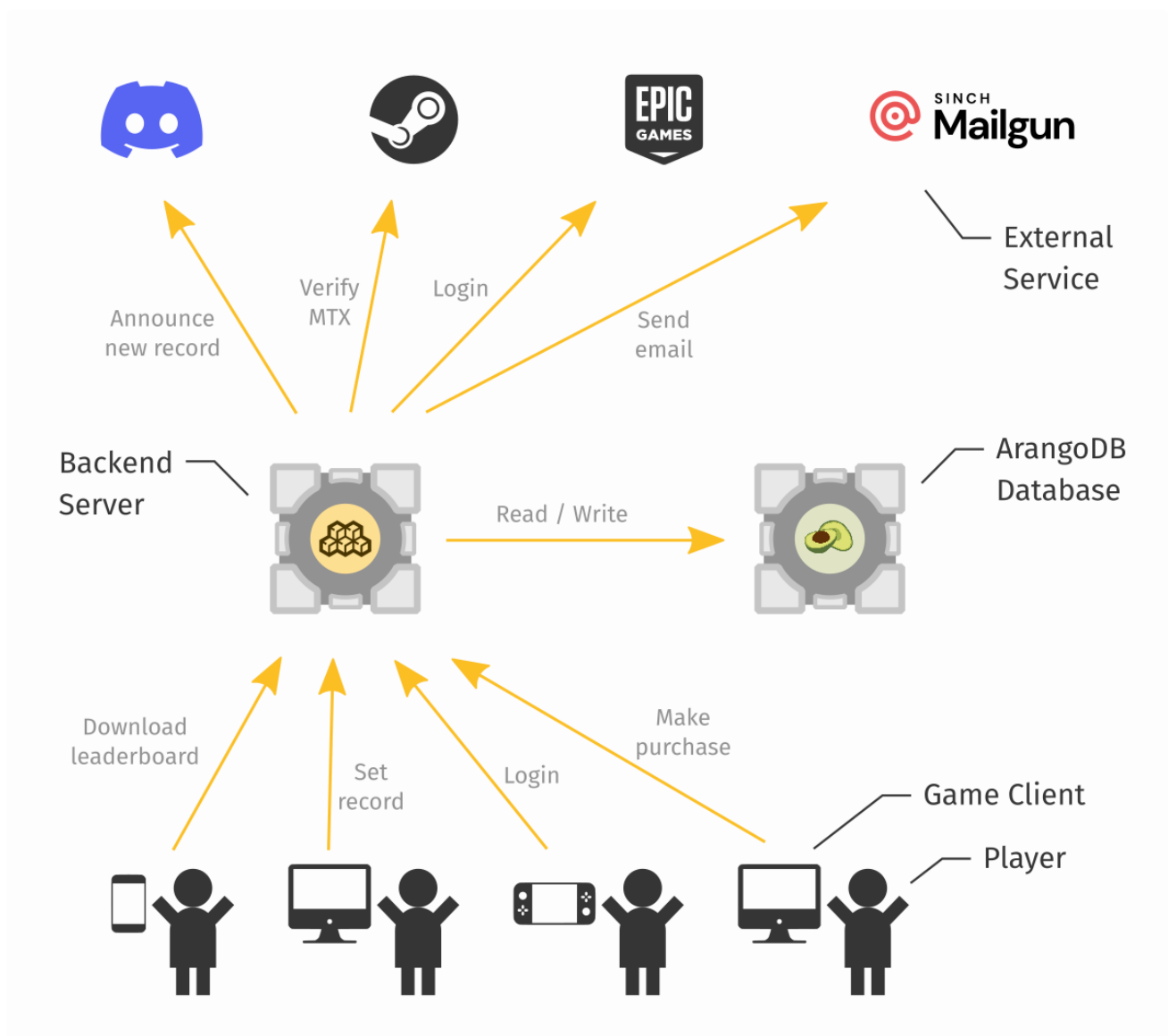
Either way, exploring these examples will give you an idea, how to build custom systems on top of Unisave. Read the next [Start building](#) section to learn about backend folders and then the [Introduction](#) documentation page, to learn about the structure of the Unisave platform, and which systems to use to achieve what goals. Maybe you don't have to build your feature from scratch, maybe there already is a module, ready to be configured and used.



Start building

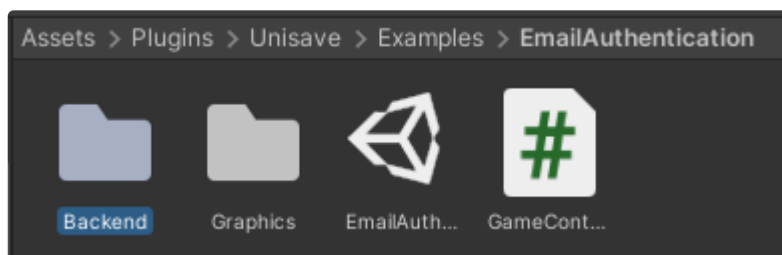
The Unisave platform, at the lowest-level, is a platform that runs your backend servers, and gives you access to a database. Your role here is to build the backend server.

The backend server is a piece of software, that runs in the cloud, accepts requests from your game clients (the game you build in Unity Editor), manages the data stored in the database, and communicates with external services.



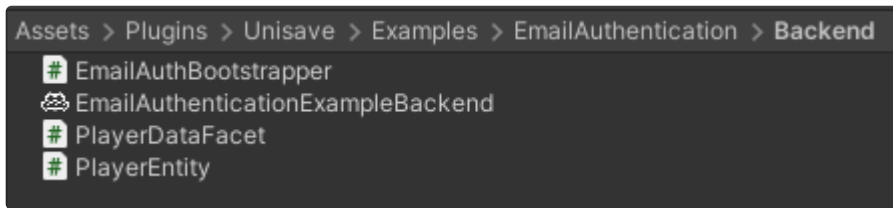
Note: You can view Unisave as a [FaaS](#) service, together with a [DBaaS](#) service, with a few other auxiliary functionalities, such as code deployment.

The backend server is written in C# and is part of your Unity project. The code is placed in so-called **Backend Folders**. It's the same as having your textures in a `Textures` folder, your code in a `Scripts` folder, you will have your backend scripts in a `Backend` folder. Here is the backend folder for the `Email Authentication` example:



A backend folder can contain C# code used to accept game client requests (see [Facets](#)), code to interact with the database (see [Entities](#)), code to configure other Unisave modules (see [Bootstrappers](#)), or any other custom C# code. The only restriction is that

the code cannot access client-side logic (e.g. `GameObjects` or `Components`), because there's no such thing on the server.

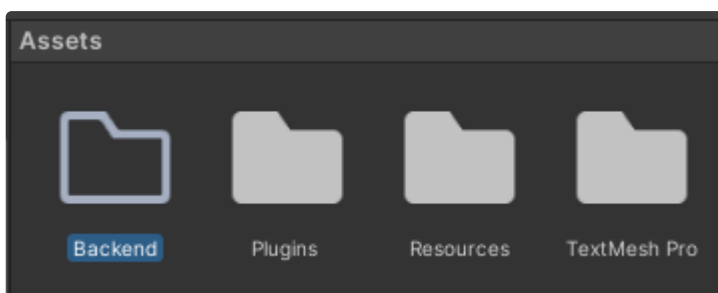


Because these folders are automatically uploaded to the cloud and compiled, they need to be identified somehow. This is done by creating a **Backend Folder Definition File** inside of the folder. The file can be called anything, you can see it here being named `EmailAuthenticationExampleBackend`.

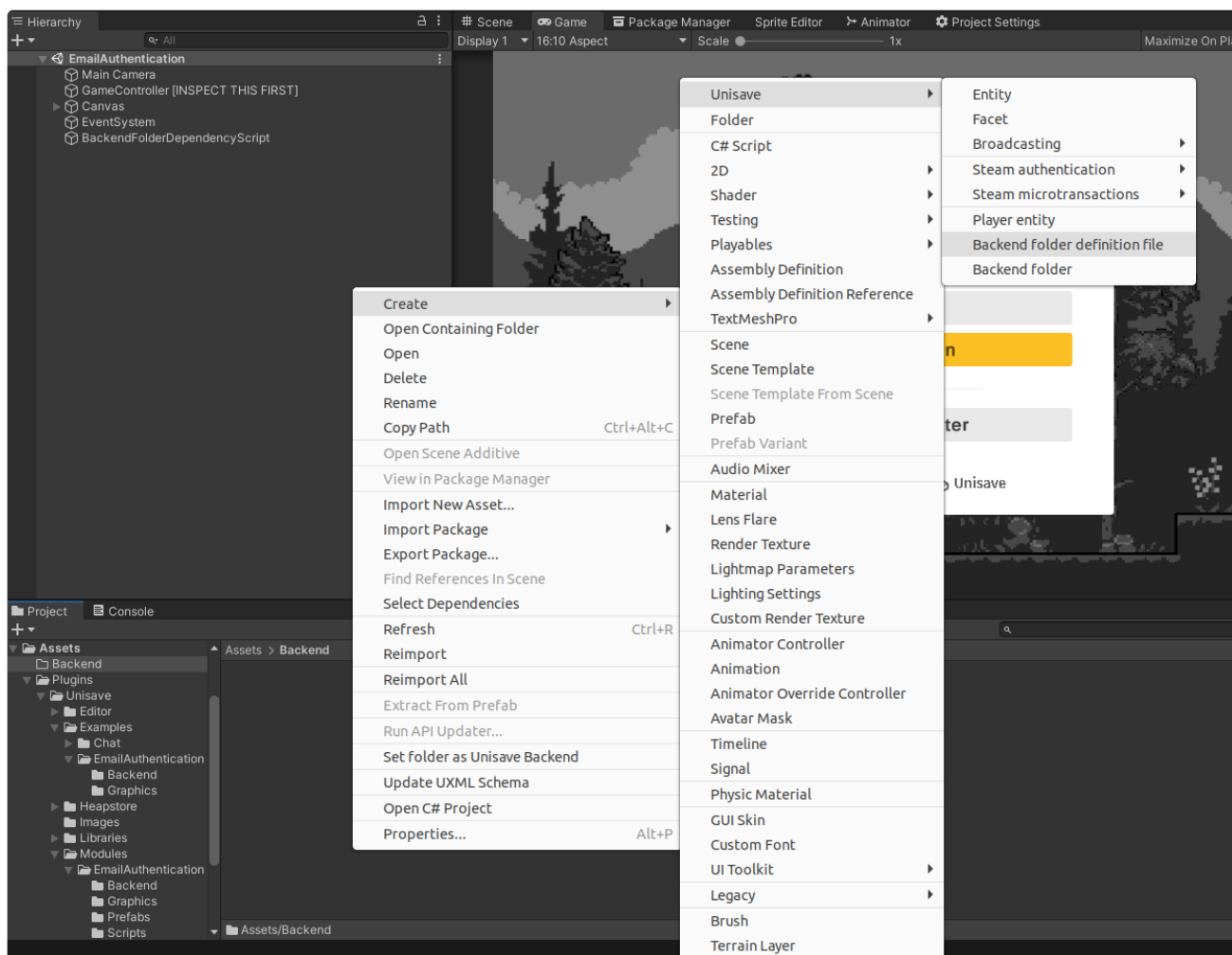
Create a backend folder

You will need at least one backend folder that contains your own backend code. Even if you only plan to use existing [Unisave modules](#), you still need this folder to configure the modules.

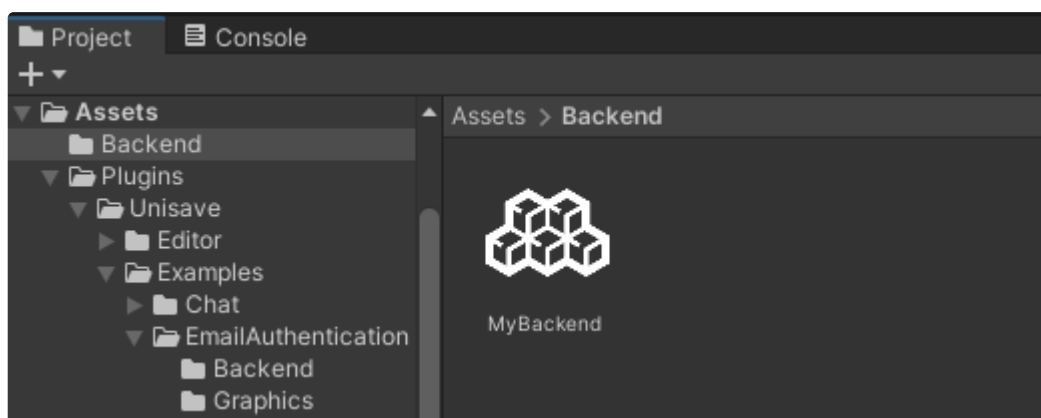
You can place it directly into your `Assets` folder. Simply create an empty folder named `Backend`:



Then enter that folder, and create a **Backend Folder Definition File** by right-clicking, and choosing `Create > Unisave > Backend Folder Definition File`. You can call the file `MyBackend`.



You can now create your own [Facets](#), [Entities](#), and other backend code in this folder.



You can now proceed onto the [Introduction](#) page.

Updating the asset

Unity has a bit tricky asset updating. The problem is that it only adds and modifies files, it does not remove old files. This is a problem since Unsave asset is full of C# code and you want to remove deprecated files, otherwise, the update might break the Unsave asset.

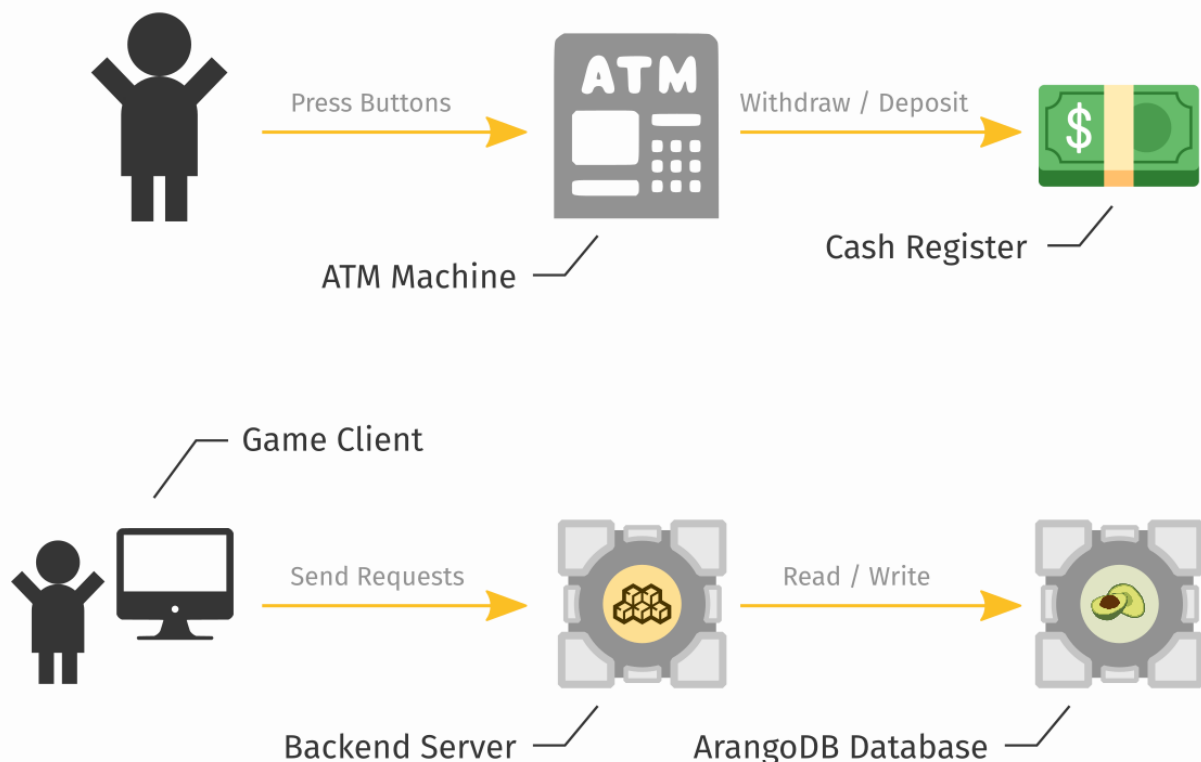
The best way to install an update is to delete the entire `Assets/Plugins/Unisave` folder and then to import the Unisave asset again.

This asset updating behavior is rather weird but it has its reasons so we just have to deal with it.

Introduction

Using Unisave is all about building and operating your own backend server. There are many tools to build it, starting from ready-made drop-in modules, all the way down to low-level HTTP and database requests. The modules get you going quickly, and can be configured, but when you need something special and custom, you have the low-level tools to build it from scratch.

But really, the most important thing about a backend system is the data in the database. The backend server is there only to manipulate the data. The database is the only component, that knows what players are registered, what they own, what they achieved. The database is like the cash register in an ATM, the backend server is like the screen and the buttons that dictate who and how can access that cash.

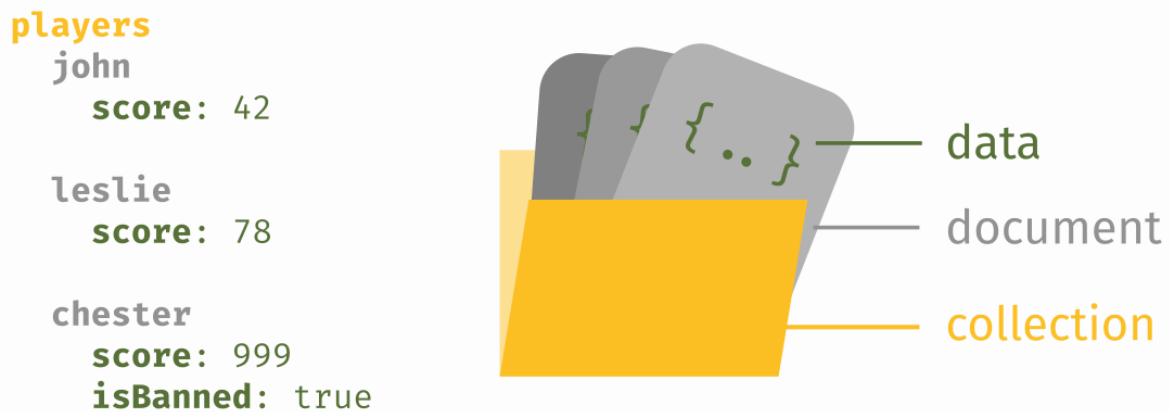


Therefore whenever you're designing some backend logic, always start with the data. What is going to be stored, how exactly, what values will be allowed, and what they mean.

The database

Unisave uses the [ArangoDB](#) database to store your game data. The reason for this choice is briefly described in [an article I wrote](#).

ArangoDB database organizes data into JSON documents, grouped into collections. You typically have one collection for each type of entity, say `players`, `bonus_codes`, `error_logs`, `leaderboard_records`.



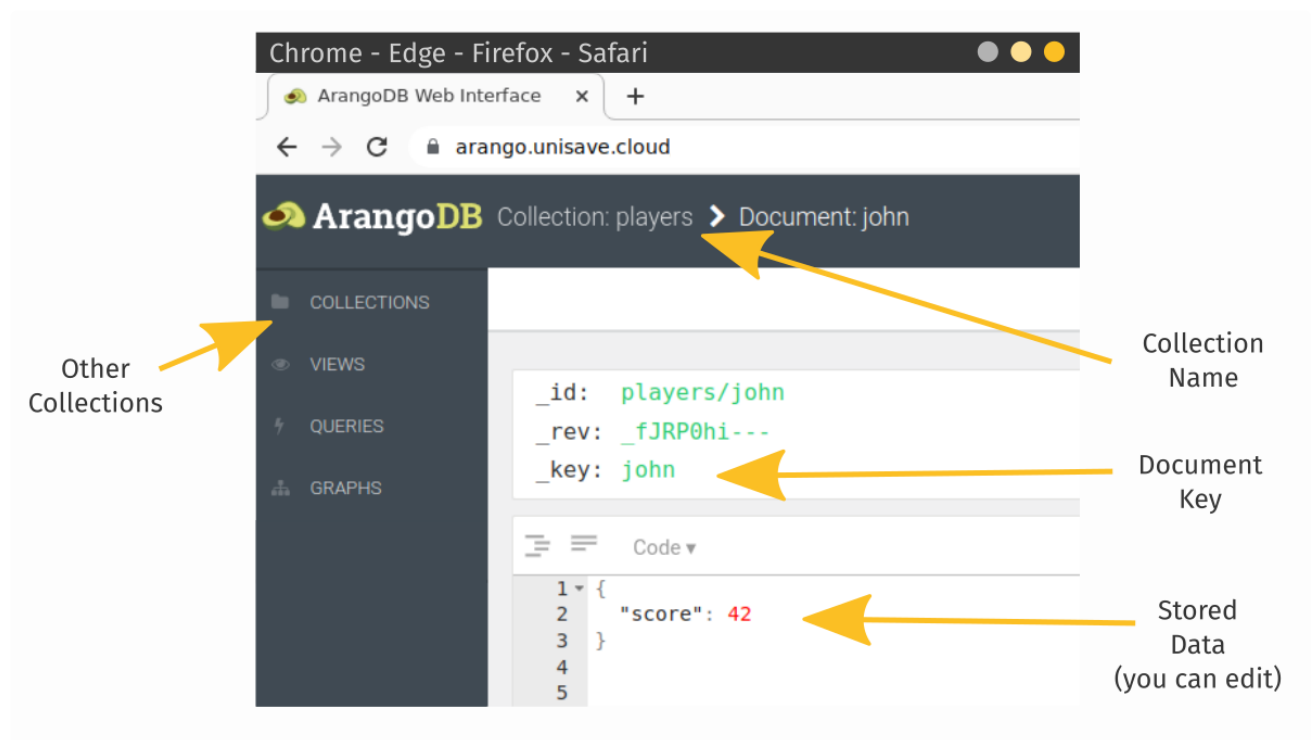
Each document is a JSON object containing fields. The document is identified by a `_key` within the collection, or an `_id` within the database (`_id` is just the `_key` combined with the collection name). The `_key` is either set by you, or automatically generated by the database. Documents inside a collection usually have the same fields (the same structure), but it's not necessary and may vary within a collection (say, for a `configuration` collection).

Here is an example document representing a player:

```
{
  "_key": "chester",
  "_id": "players/chester",
  "score": 999,
  "isBanned": true
}
```

Aardvark

When you log into the cloud dashboard, you can view and edit your game database through an ArangoDB interface called Aardvark:



The backend server

Your backend server stands between your game clients and your database. Its responsibility is to handle client requests (say, setting a new leaderboard record) and use and modify the database accordingly (say, add new record to the `leaderboard_records` collection and send back the top 10 records).

The code responsible for receiving and processing client requests is called [Facets](#). Public methods on a facet class can be called from the client over the Internet. Here is an example facet class, that should be placed in your [Backend Folder](#):

```
using System;
using System.Collections.Generic;
using Unisave;
using Unisave.Facades;
using Unisave.Facets;

public class LeaderboardFacet : Facet
{
    /// <summary>
    /// Returns all the leaderboard records.
    /// </summary>
    public List<LeaderboardRecordEntity> DownloadLeaderboard()
    {
        // fetches all documents of the `leaderboard_records`
        // collection using the Entities system (see below)
        return DB.TakeAll<LeaderboardRecordEntity>().Get();
    }
}
```

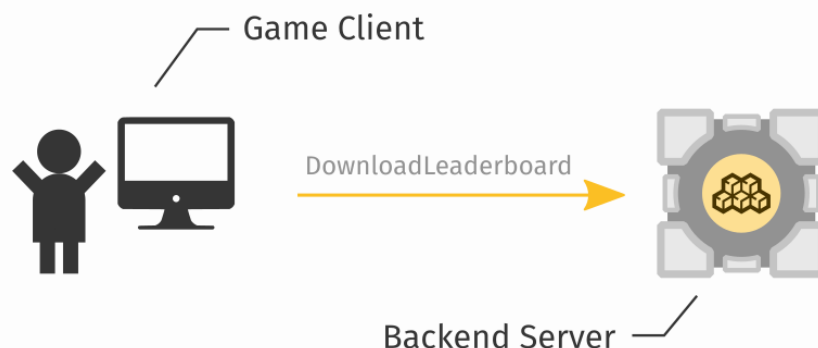
You can call this facet method from a `MonoBehaviour` script like this:

```
using System;
using Unisave;
using Unisave.Facets;
using UnityEngine;

public class LeaderboardController : MonoBehaviour
{
    async void Start()
    {
        var records = await this.CallFacet(
            (LeaderboardFacet f) => f.DownloadLeaderboard()
        );

        foreach (LeaderboardRecordEntity r in records)
            Debug.Log(r.nickname + ": " + r.score);
    }
}
```

This is what is happening on the diagram from the top of this page. A game client code (a `MonoBehaviour`) is making a request (a [Facet call](#)) to your backend server, to the `LeaderboardFacet` class:



The `LeaderboardFacet` uses the [Entities](#) system to communicate with the ArangoDB database. To work with the data in C#, we need to create a new class, called an entity, that will be used to facilitate this communication:

```
using System;
using Unisave;
using Unisave.Entities;

[EntityCollectionName("leaderboard_records")]
public class LeaderboardRecordEntity : Entity
{
    /// <summary>
    /// Nickname of the record holder
    /// </summary>
```

```

public string nickname;

/// <summary>
/// The achieved score
/// </summary>
public double score;
}

```

Entities let us easily work with the database from backend code:

```

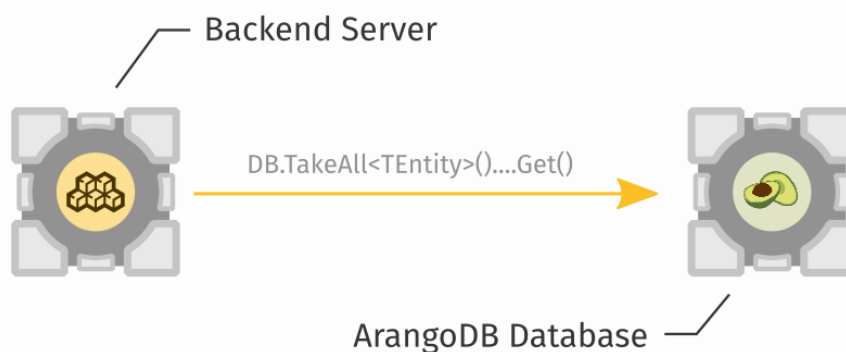
// get all documents
var records = DB.TakeAll<LeaderboardRecordEntity>().Get();

// get only some documents
var records = DB.TakeAll<LeaderboardRecordEntity>()
    .Where(e => e.score >= 10_000)
    .Get();

// create a new document
var r = new LeaderboardRecordEntity {
    nickname = "John",
    score = 42
};
r.Save();

```

Looking back at our diagram, this is how our `LeaderboardFacet` uses entities to download the documents from the database, before returning them back to the client:



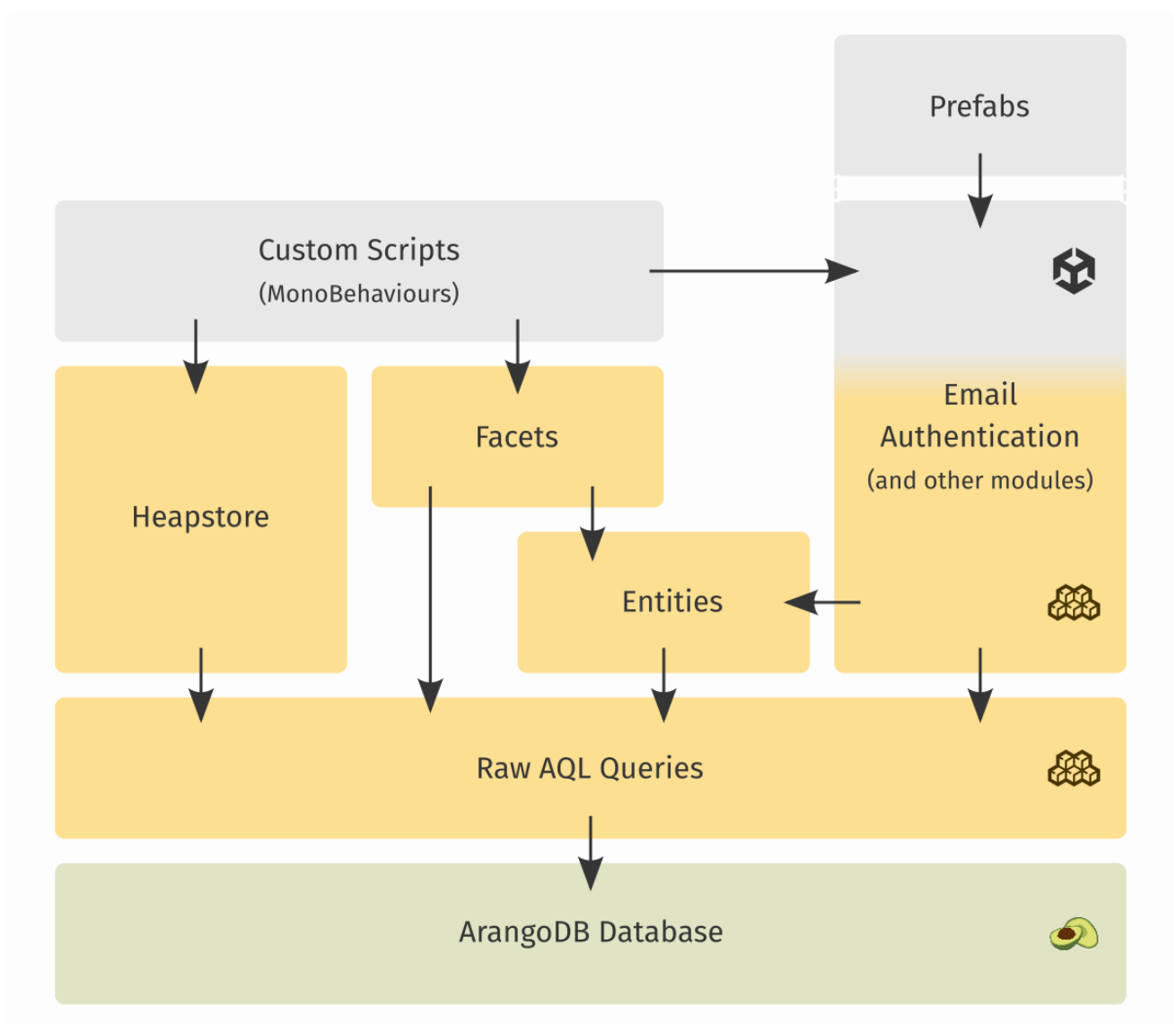
Systems and modules

[Entities](#) and [facets](#) are not the only way, how to build a backend server. In-fact, they are one of the more low-level systems that you use to build a feature from scratch. In practise, there are [modules](#), like [Email Authentication](#), that you use to add larger pieces of functionality to your game quickly. Then you use entities and facets only to tie these modules together and to add your custom specific features. Try going through the list of [modules](#) in to documentation, to see what's available.

Note: If the module you want does not exist, chances are there are other people who would like such a module as well. You can build such a module yourself and distribute or sell it via the Unity Asset Store.

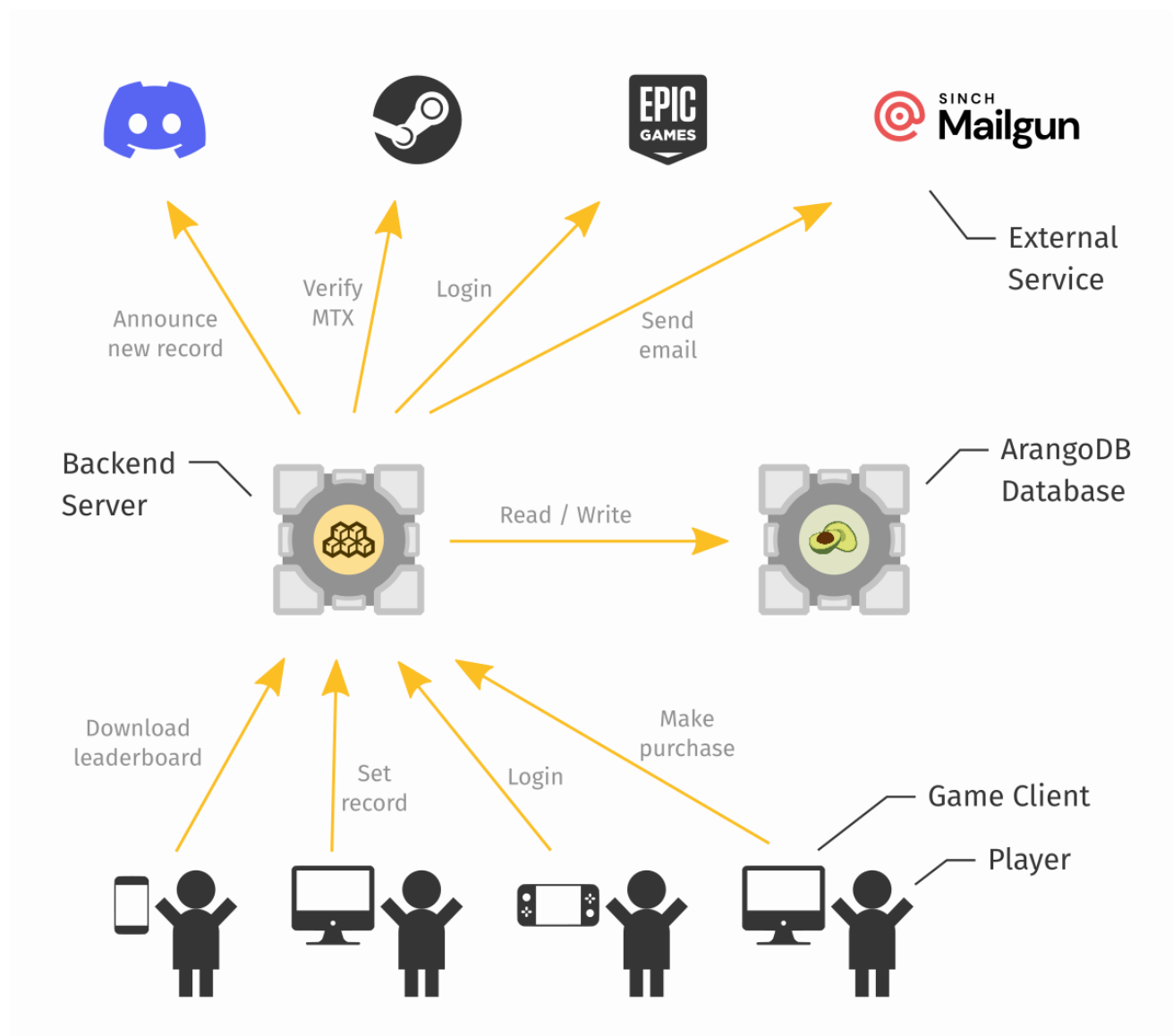
Modules are designed to provide a specific feature, or to solve a specific problem. Unisave also has systems/modules that are more general-purpose and may be used to build your backend server in a different way to the entities and facets systems. The two notable are:

- **Raw AQL queries:** These let you access the ArangoDB database at the lowest-level possible. If you hit a limitation of the entities systems, you can always find a solution by writing a custom AQL query. Using custom queries may also be necessary in performance critical cases. The entity system is actually built on top of this layer.
- **Heapstore:** Heapstore lets you access the database directly from your client code, skipping the need for writing custom facets or entities. This makes it very convenient to write simple logic and to draft out ideas. The data-access is protected by writing security rules. You can use Heapstore alongside other systems, since at the end, all the systems just edit the database contents. While easy to use, it's also limited in its features and you might find it necessary to resort back to facets and entities/AQL-queries for more complex logic. Also, it's heavily inspired by Firebase.



As you can see on the diagram, high-level feature-based modules, such as [Email Authentication](#) can be used by dragging-in prefabs and they handle everything all the way to the database. But they can't be used to build anything else, other than what they provide. Whereas more general-purpose, lower-level systems, like facets and entities, only focus on a little part of the backend server. But they can be used to build any custom system.

Also, don't forget that Unisave is not only about the database but also about connecting external services.



Where to go next

Now you can continue to a multitude of places, depending on your needs:

- Browse [modules](#), to see if there already is a solution for your problem.
- Read [guides](#) to learn how to build custom features.
- Read more on [facets](#) and [entities](#) to learn in what ways they can be used.
- Read about the [authentication system](#), to see how all the authentication modules yield control back to you, once a player becomes logged-in.
- If you want to communicate with external services, use the [HTTP client](#).
- Try to see if [Heapstore](#) does not fit your problem best, instead of facets and entities.

You can start by reading the [How to store player data online with Unity](#) guide.

LEGAL

- [Terms of Service](#)
- [Data Processing](#)
- [Privacy Policy](#)
- [Pricing](#)

SOCIAL

- [Discord](#)
- [LinkedIn](#)
- [Facebook](#)
- [Youtube](#)
- [Github](#)

OTHER

- [Guides](#)
- [Documentation](#)
- [Asset](#)
- [App](#)

CONTACT

- Vítězství 217
Slatiňany 538 21
Czech Republic
- info@unisave.cloud
- +420 604 112 523



Copyright © 2020 - 2024 Jiří Mayer, IN 06861814 (CZ)