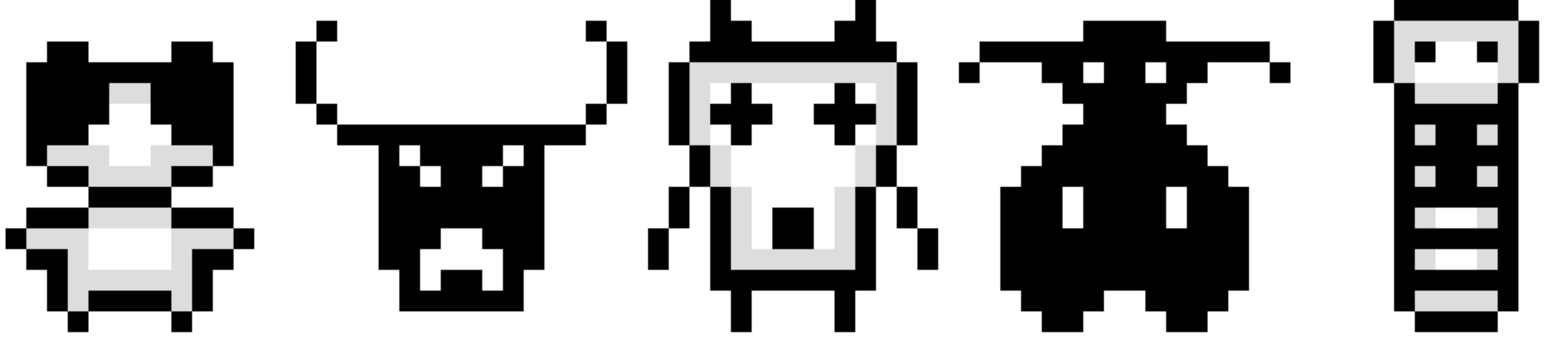
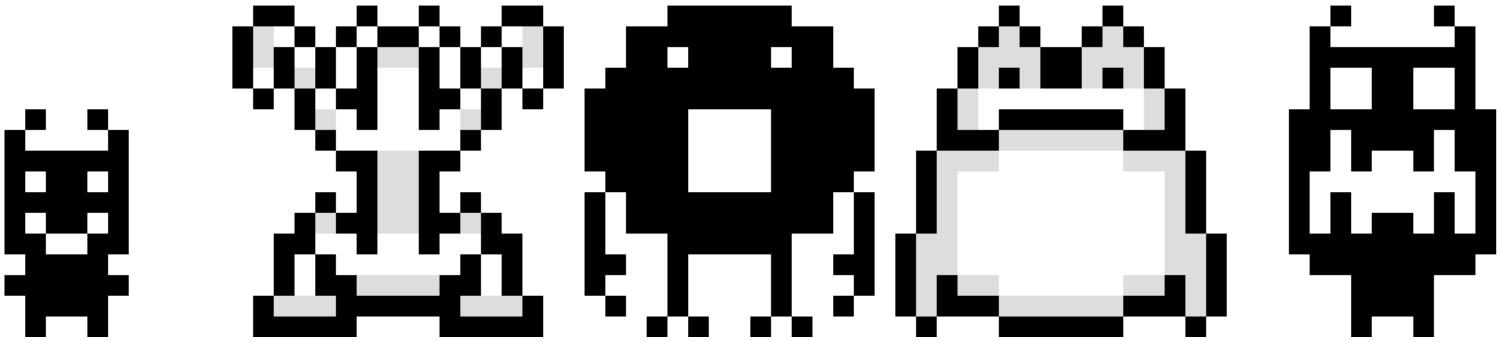


SPRITES SHARE



Sprites Share

User Manual

AZAR Majed
majed.azar7@gmail.com
Lome (Togo), November 27, 2022

Table of Content

Purpose

Sprites Share is a platform to share sprites online and download sprites manually or import them in real-time inside your project thanks to an API. This project is pretty simple and there are already lots of ways to get sprites online so, why did I create this? Let me elaborate on that…

What's Next? – The Video Game Generator

It all started with "What's Next?", the video game generator I'm working or "emergin" from its old name. To make it short, the initial idea to make a program where whenever you press a "generate new game" button, a brand-new game would be created for you from scratch. In this game, the sprites, sound effects, music, levels, worlds, scenarios, cutscenes, game genre and anything you could think of would be procedurally generated at runtime by the game generator. On the paper, it sounded like an amazing idea and making this project come to life would have definitely been an achievement from a software engineer perspective as it would have required advanced algorithms.
The problem with this draft concept though was that from a Game Designer perspective, this game would have been way to generic to be appreciated and remembered. For example, music is an important factor to connect memories to events. If someone plays the Super Mario Bros. theme, you can probably already see Mario jumping inside your head because a deep connection was established between this music and Mario. This was possible thanks to repetition: each time you play Super Mario Bros. you hear this music. Having fixed music makes the experience more memorable. Added to that, it also sets expectations. If you get lose health and you hear a certain sound, your brain makes a connection between the 2 events: "if I hear this sound, I just lost some health". But what if in your next playthrough, you now hear the same sound when you gain health? Your perception of the world gets broken and your ability to make predictions in the game is weaken, making it a less enjoyable experience.
So, here is the problem: too many randomness kills the experience? So, how did I solve it? Well, I decided to go for "controlled randomness". Some elements are fixed, like the music for example, and some are procedurally generated, like level layouts. Now, things are more organized this way. About sprites used in the game though, I decided to go for another road: what if sprites were fetched from an online database? This way, I can add more sprites over time to avoid the user always shuffling on the same set of sprites. Added to that, when I upload/delete sprites in my database, the users would instantly get the result without any updates. Sounds like a great idea, right? Well, this is how the project Sprites Share was starting to get in my mind.

Sharing outside of the project

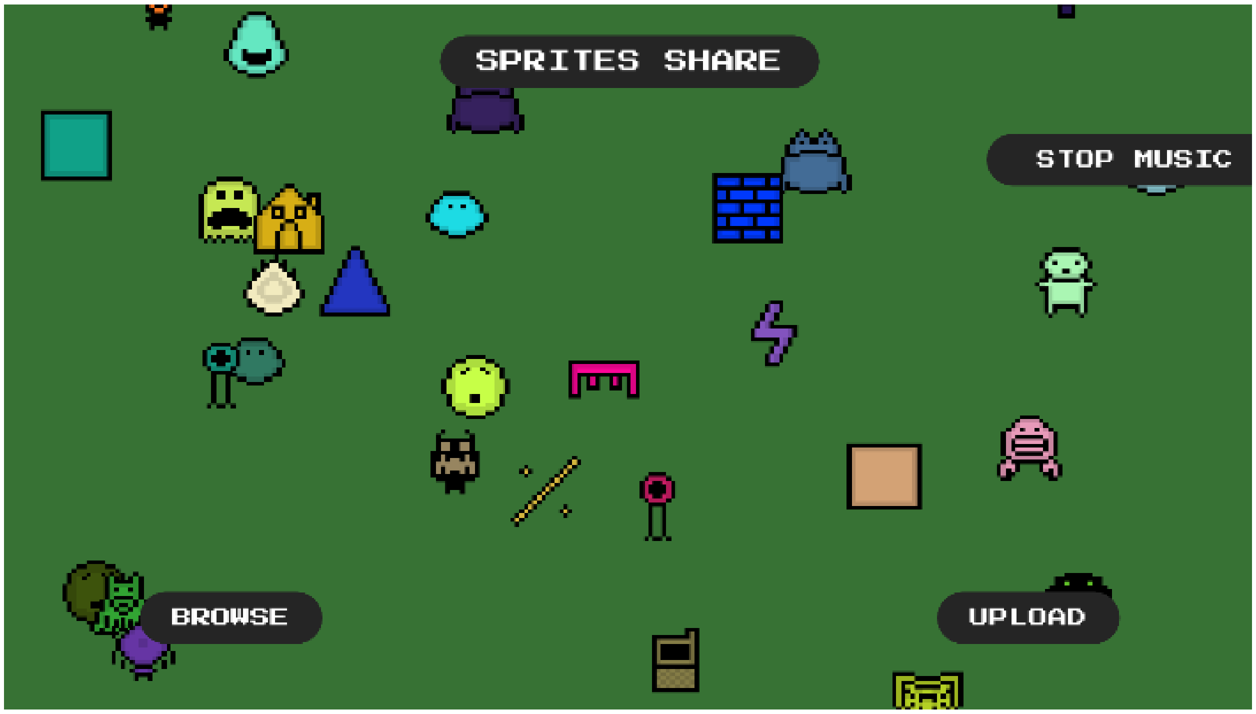
At this point, I wanted to keep my sprites inside an online database to be able to update it in real-time without having to upload a new build for my video game generator. That's was great idea, but then I thought: "if the sprites are already outside of my game generator, what not make them available inside my other projects?". Indeed, at the start, I wanted to have a section in What's Next dedicated to sprite sharing. This would have been similar to the current platform Sprites Share, but just embedded inside What's Next. But in the future, I might want

to use Sprites from this online database in other projects so why keep it in What’s Next and why not make a separate project for this idea?
The idea stuck with me so, I indeed decided to pull it out of the game generator to make it a standalone project. And then, a last thought came to mind: “if I pulled it outside to make it available to my other projects, why not share it with everyone at this point?”. I feel like that was the finishing touch for this project. If I kept this project to myself, I would be the only one upload sprites to the platform so there would have been less variety and it would have been very time consuming for me to always upload lots of sprites. If this project is public though, this is solved. Everyone can upload sprites on the platform so there is more variety and in exchange, everyone can use them inside his/her project. It’s a win-win situation. Having settled on this idea, I started to work on it and today, here is Sprites Share!

Manual usage

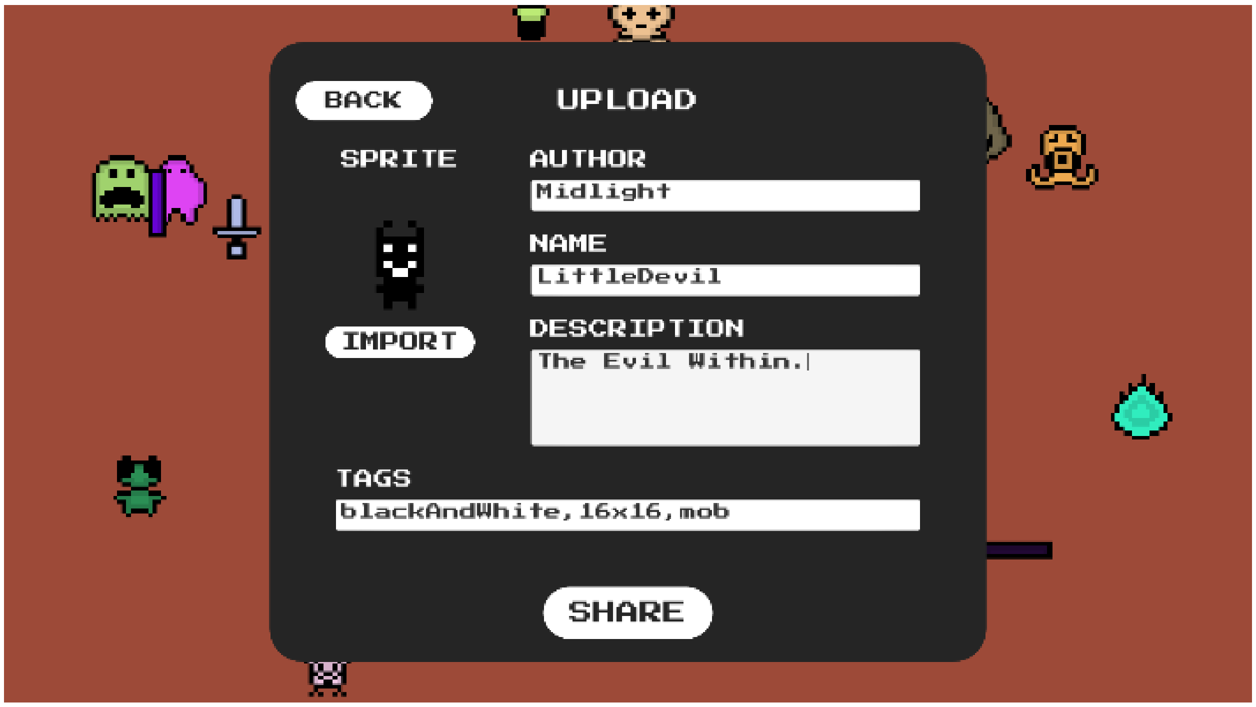
Now that you understand how the project was born and to what purpose, let's see how to actually use it. First, let's start with the simplest use case: manual usage. Once we understand everything about the platform, I'll go deeper inside the next section and explain how to make API calls to it.

Welcome Page



On the Welcome Page you can see the root of the project. To animate things, you can see some sprites from the database raining down the screen and you have the following controls: "Stop Music", "Browse", "Upload". The "Stop Music" button is, as its name implies, a toggle button to turn the music on/off. The "Upload" button takes you to the Upload page and the "Browse" button takes you to the Browse page. Let's start with upload first.

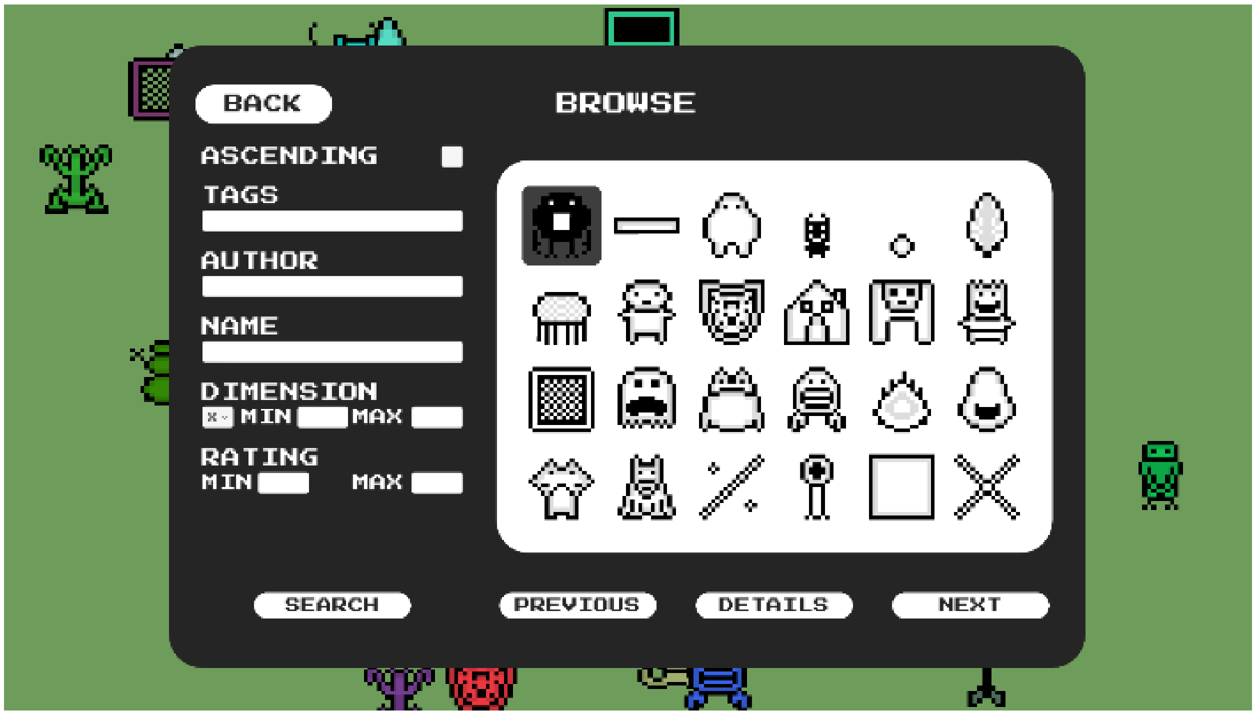
Upload Page



AUTHOR: Midlight
NAME: LittleDevil
DESCRIPTION: The Evil Within.
TAGS: blackAndWhite,16x16,mob

To upload a sprite from the Upload page, you have to complete the following fields: “sprite”, “author”, “name”, “description” and “tags” with tags being the only one not required. For the actual “sprite”, you can import one from your pc by clicking the “Import” button. The other fields are quite self-explanatory with the only tricky one being the “tags” field. If you want to add “tags” to your sprite for easier research in the future, you can enter all your tags inside the tags field by separating each one with a ‘,’ character. Once you’re happy with your sprite, you can share it with all of us by clicking the “Share” button, and you’re done! Now, let’s how we can browse through sprites and eventually download them.

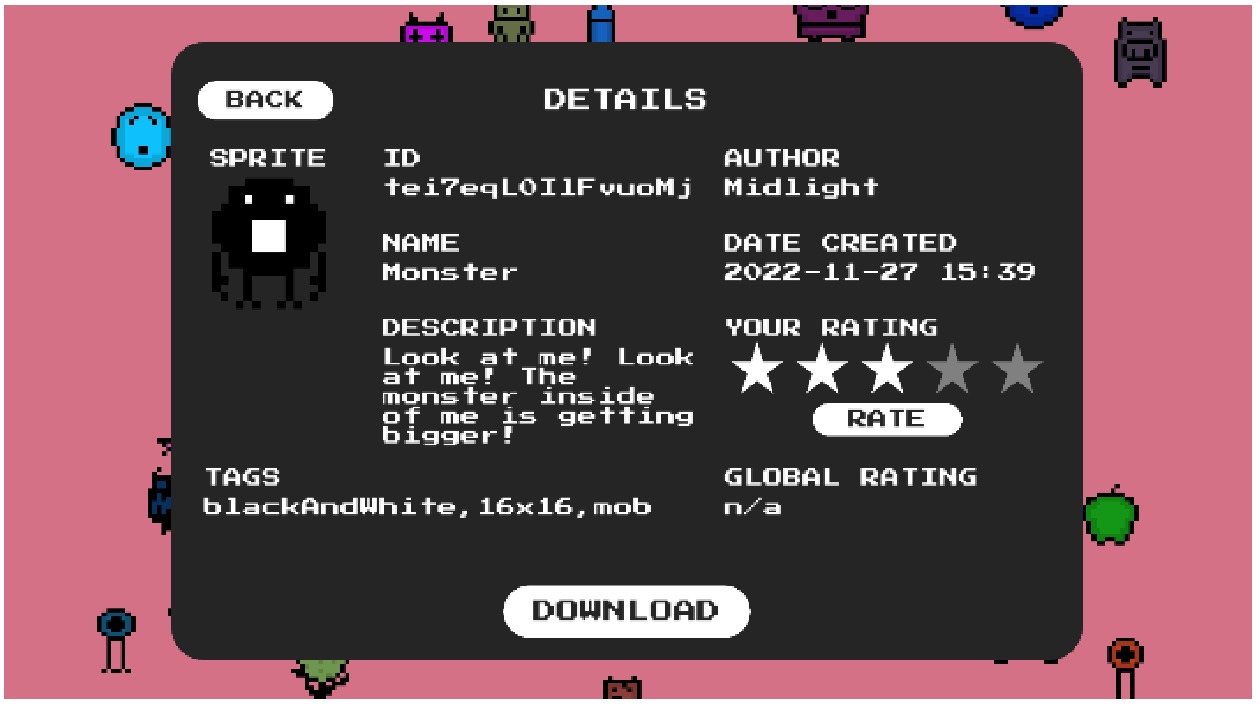
Browse Page



On the Browse page, you can look search for sprites according to a lot of fields. The “ascending” field determines if the sprites are going to be shown in ascending or descending order, descending being the default. “tags”, “author” and “name” fields are self-explanatory, you can use them to search for sprites based on the value of those fields. “dimension” and “rating” are a little different because they both have sub-fields “min” and “max”. “min” is the minimum for the corresponding field and “max” is the maximum. Added to that, the “dimension” field also has a dropdown containing “x” and “y” to determine if you make queries based on the width or the height of a sprite.

Once you’re done specifying all the fields of interest for you, you can press the “Search” button to start the research. Once the results are shown, you can switch page using the “Next” and “Previous” buttons. If you’re interested in one sprite in particular, you can click on it and press “Details” to get more info about this specific sprite.

Details Page



ID
tei7eqL0IlFvuoMj

AUTHOR
Midlight

NAME
Monster

DATE CREATED
2022-11-27 15:39

DESCRIPTION
Look at me! Look at me! The monster inside of me is getting bigger!

YOUR RATING
★★★☆☆

TAGS
blackAndWhite,16x16,mob

GLOBAL RATING
n/a

On the Details page, you can see all information regarding a sprite: id, author, name, date created, description, tags and global rating. Added to that you have 2 options on this page, “Rate” and “Download”. With the “Download” button you can download the sprite and save it on your computer and with the “Rate” button, you can rate the sprite. Rating is an important feature for such a Sprite Sharing software because you can search for sprites based on the ratings to avoid getting “bad sprites”.

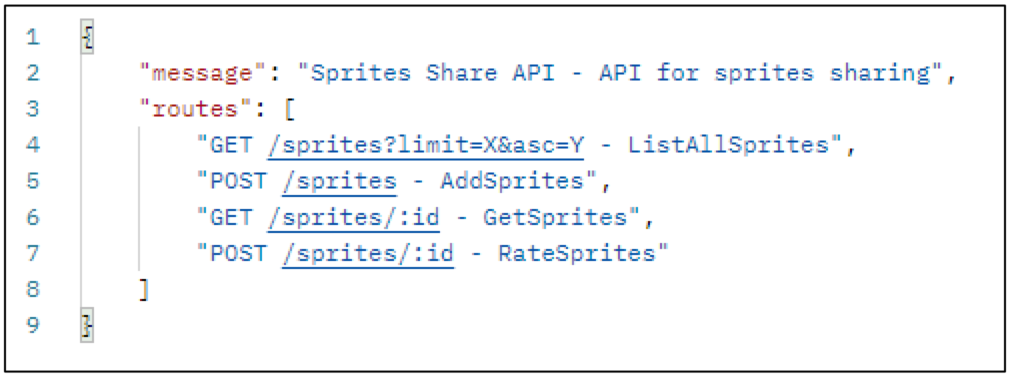
API Calls

Now that we’ve seen how the Sprites Share platform works inside the client app, we have a better understanding about the functionalities so, we’re in a better position to understand the API. Let’s how it works then.

Initial Setup

To communicate with the API, all requests are made on the server:

https://sprites-share-api-rwvkn4ky5a-lm.a.run.app
If you make a GET request at this address, you should see the following result:

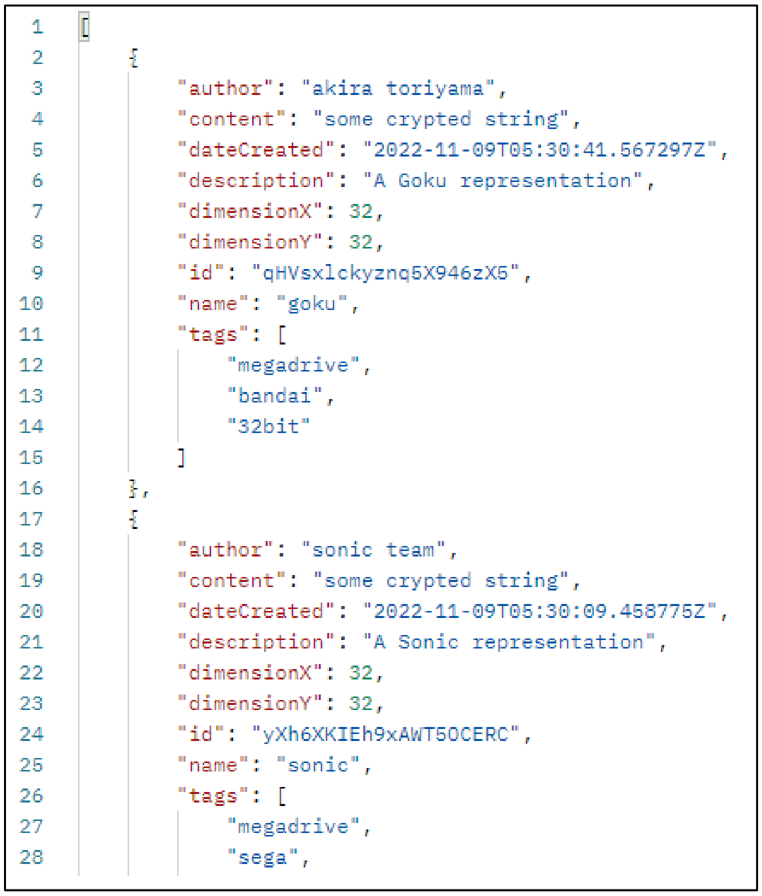


```
{
    "message": "Sprites Share API - API for sprites sharing",
    "routes": [
        "GET /sprites?limit=X&asc=Y - ListAllSprites",
        "POST /sprites - AddSprites",
        "GET /sprites/:id - GetSprites",
        "POST /sprites/:id - RateSprites"
    ]
}
```

To access different routes through the API, just add the path to this route to the base server address. For example, GET https://sprites-share-api-rwvkn4ky5a-lm.a.run.app/sprites access the /sprites route. Let’s start with this one.

GetAllSprites

You can see all sprites on the database with the following request: GET /sprites



```
[
    {
        "author": "akira toriyama",
        "content": "some crypted string",
        "dateCreated": "2022-11-09T05:30:41.567297Z",
        "description": "A Goku representation",
        "dimensionX": 32,
        "dimensionY": 32,
        "id": "qHVsxlckyznq5X946zX5",
        "name": "goku",
        "tags": [
            "megadrive",
            "bandai",
            "32bit"
        ]
    },
    {
        "author": "sonic team",
        "content": "some crypted string",
        "dateCreated": "2022-11-09T05:30:09.458775Z",
        "description": "A Sonic representation",
        "dimensionX": 32,
        "dimensionY": 32,
        "id": "yXh6XKIEh9xAWT5OCERC",
        "name": "sonic",
        "tags": [
            "megadrive",
            "sega",
```

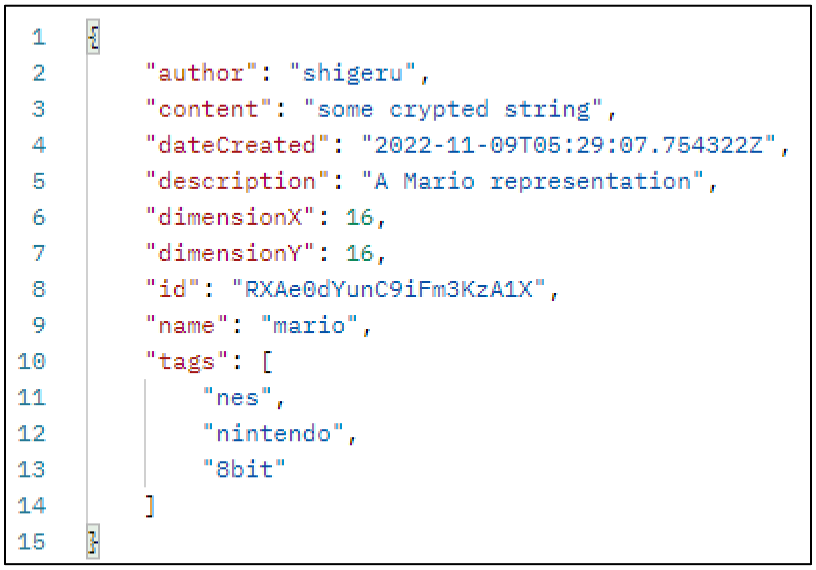
Added to that, you can pass in query params including:

- limit: how many results per page;
- lastItem: id of the starting point of the query (not included)
- asc: show results in ascending order? (true = yes)
- tags: tags separated by ‘,’ contained in sprites;
- author: author of sprites;
- name: name of sprites;
- dimensionXMin: minimum width of sprites;
- dimensionXMax: maximum width of sprites;
- dimensionYMin: minimum height of sprites;
- dimensionYMax: maximum height of sprites;
- ratingMin: minimum rating of sprites;
- ratingMax: maximum rating of sprites;

With those parameters, you can make queries like /sprites?limit=5&asc=true for example. About lastItem though, it’s actually used for pagination. Let’s say that you with the query /sprites?limit=10 you got the sprites but you want to see more sprites, the “next page”. If the last sprites returned by this query had an id of “yXh6XKIEh9xAWT50CERC” then, to see the sprites after this, you’ll need the query /sprites?limit=10&lastItem= yXh6XKIEh9xAWT50CERC. The result will be a list of sprites if you’re query matches data on the database or a null object if nothing was found.

GetSprite

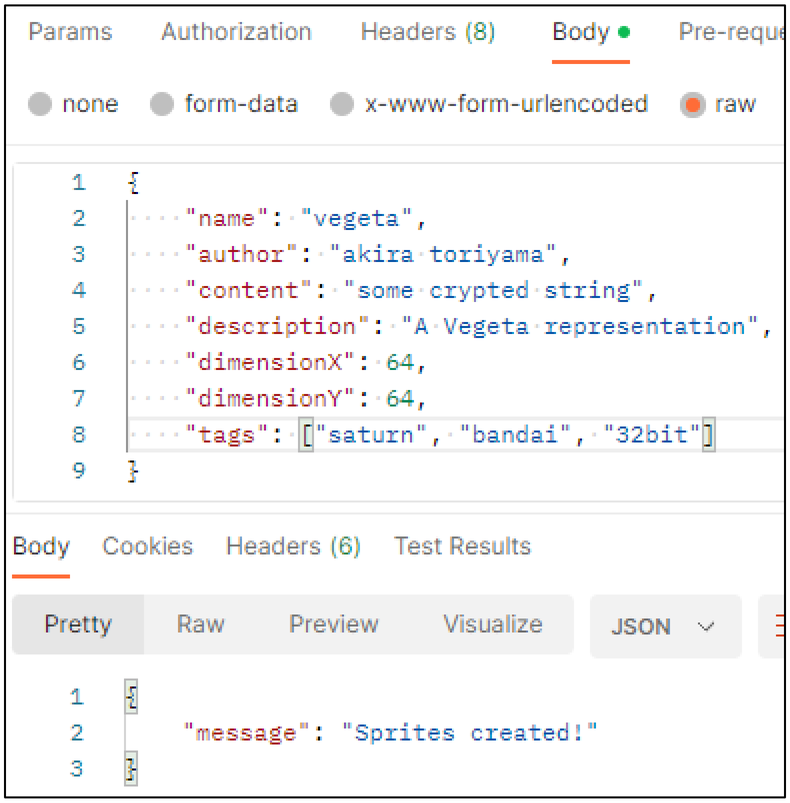
Instead of getting all the sprites, you can only get info about a specific one with the query:
GET /sprites/:id (with id = id of the sprite)



```
{
    "author": "shigeru",
    "content": "some crypted string",
    "dateCreated": "2022-11-09T05:29:07.754322Z",
    "description": "A Mario representation",
    "dimensionX": 16,
    "dimensionY": 16,
    "id": "RXAe0dYunC9iFm3KzA1X",
    "name": "mario",
    "tags": [
        "nes",
        "nintendo",
        "8bit"
    ]
}
```

AddSprite

To add sprites inside the database, you can use the query POST /sprites with the following JSON body:



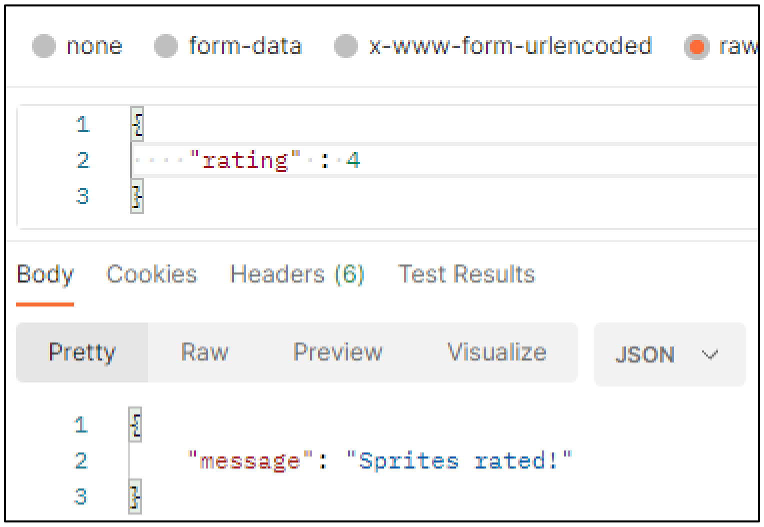
```
{
    "name": "vegeta",
    "author": "akira toriyama",
    "content": "some crypted string",
    "description": "A Vegeta representation",
    "dimensionX": 64,
    "dimensionY": 64,
    "tags": ["saturn", "bandai", "32bit"]
}
```

```
{
    "message": "Sprites created!"
}
```

About this functionality though, I wouldn't recommend using it just with raw API Calls because the content parameter is a Texture 2D in RGBA32 format converted to string in Unity. If you add a sprite that doesn't respect this standard, it will be unusable by end users.

RateSprite

Last but not least, to rate a sprite, you can use the query POST /sprites/:id (with id = id of the sprite) with the following JSON body:



```
{
    "rating" : 4
}
```

```
{
    "message": "Sprites rated!"
}
```

Pay in mind that a rating can only be between 1 and 5.

Unity Integration

Even if I explained how to communicate with the API through API Calls, I highly doubt anyone would do it outside of Unity or another game engine like Unreal or Godot. Unfortunately, I don't have tutorials of how to use it in those game engines, I'm not even sure if it's possible actually due to the content format but at least, I can share a hands-on example using Unity.

Initial Setup

To be able to communicate with the API, I prepared the following Models class:

```
using System;
using System.Collections.Generic;


[Serializable]
public class SpriteData
{
    public string author;
    public string content;
    public string dateCreated;
    public string description;
    public uint dimensionX;
    public uint dimensionY;
    public string id;
    public string name;
    public uint rating;
    public Dictionary<string, uint> ratings;
    public string[] tags;
}
```

```
using System;

[Serializable]
public class GetAllSpritesRequest
{
    public uint limit = 0;
    public bool asc = false;
    public string tags = string.Empty;
    public string author = string.Empty;
    public string name = string.Empty;
    public uint dimensionXMin = 0;
    public uint dimensionXMax = 0;
    public uint dimensionYMin = 0;
    public uint dimensionYMax = 0;
    public uint ratingMin = 0;
    public uint ratingMax = 0;
    public string lastItem = string.Empty;
}
```

```
using System;

[Serializable]
public class AddSpriteRequest
{
    public string author;
    public string content;
    public string description;
    public uint dimensionX;
    public uint dimensionY;
    public string name;
    public string[] tags;
}
```

```
using System;

[Serializable]
public class RateSpriteRequest
{
    public uint rating;
}
```

```
using System;

[Serializable]
public class ApiResponse
{
    public string message;
}
```

Added to that, I made the following Controller class:

```
using System.Net;
using System.IO;
using System.Threading.Tasks;
using UnityEngine;
using Newtonsoft.Json;

public static class SpritesController
{
    private const bool m_IsDebug = true;

    public static async Task<SpriteData[]> GetAllSprites( GetAllSpritesRequest
spritesRequest = null )
    {
        string query = "?";
        if( spritesRequest != null )
        {
            if( spritesRequest.limit > 0 )
            {
                query += "&limit=" + spritesRequest.limit;
            }
            if( spritesRequest.asc )
            {
                query += "&asc=true";
            }
            if( !string.IsNullOrWhiteSpace( spritesRequest.tags ) )
            {
                query += "&tags=" + spritesRequest.tags;
            }
            if( !string.IsNullOrWhiteSpace( spritesRequest.author ) )
            {
                query += "&author=" + spritesRequest.author;
            }
            if( !string.IsNullOrWhiteSpace( spritesRequest.name ) )
            {
                query += "&name=" + spritesRequest.name;
            }
            if( spritesRequest.dimensionXMax > 0 )
            {
                query += "&dimensionXMax=" + spritesRequest.dimensionXMax;
            }
            if( spritesRequest.dimensionXMin > 0 )
            {
                query += "&dimensionXMin=" + spritesRequest.dimensionXMin;
            }
            if( spritesRequest.dimensionYMax > 0 )
            {
                query += "&dimensionYMax=" + spritesRequest.dimensionYMax;
            }
            if( spritesRequest.dimensionYMin > 0 )
            {
                query += "&dimensionYMin=" + spritesRequest.dimensionYMin;
            }
            if( spritesRequest.ratingMax > 0 )
            {
                query += "&ratingMax=" + spritesRequest.ratingMax;
            }
            if( spritesRequest.ratingMin > 0 )
            {
                query += "&ratingMin=" + spritesRequest.ratingMin;
            }
            if( !string.IsNullOrWhiteSpace( spritesRequest.lastItem ) )
            {
                query += "&lastItem=" + spritesRequest.lastItem;
            }
        }
        query = query.Replace( "?&", "?" );

        HttpWebRequest request = (HttpWebRequest)WebRequest.Create(
"https://sprites-share-api-rwvkn4ky5a-lm.a.run.app/sprites" + query );
        Debug.Log( "https://sprites-share-api-rwvkn4ky5a-lm.a.run.app/sprites" +
query );

        HttpWebResponse response = (HttpWebResponse)( await
request.GetResponseAsync() );
        StreamReader reader = new StreamReader( response.GetResponseStream() );
        string jsonResponse = reader.ReadToEnd();
        if( m_IsDebug )
        {
            Debug.Log( jsonResponse );
        }
        SpriteData[] result = JsonConvert.DeserializeObject<SpriteData[]>(
jsonResponse );
        return result;
    }

    public static async Task<SpriteData> GetSprites( string spritesId )
    {
        HttpWebRequest request = (HttpWebRequest)WebRequest.Create(
"https://sprites-share-api-rwvkn4ky5a-lm.a.run.app/sprites/" + spritesId );

        HttpWebResponse response = (HttpWebResponse)( await
request.GetResponseAsync() );
        StreamReader reader = new StreamReader( response.GetResponseStream() );
        string jsonResponse = reader.ReadToEnd();
        if( m_IsDebug )
        {
            Debug.Log( jsonResponse );
        }
        SpriteData result = JsonConvert.DeserializeObject<SpriteData>(
jsonResponse );
```

```
        return result;
    }

    public static async Task<ApiResponse> AddSprites( AddSpriteRequest apiRequest
)
    {
        HttpWebRequest request = (HttpWebRequest)WebRequest.Create(
"https://sprites-share-api-rwvkn4ky5a-lm.a.run.app/sprites" );
        request.ContentType = "application/json";
        request.Method = "POST";
        using ( var streamWriter = new StreamWriter( request.GetRequestStream() )
)
        {
            streamWriter.Write( JsonConvert.SerializeObject( apiRequest ) );
        }

        HttpWebResponse response = (HttpWebResponse)( await
request.GetResponseAsync() );
        StreamReader reader = new StreamReader( response.GetResponseStream() );
        string jsonResponse = reader.ReadToEnd();
        if( m_IsDebug )
        {
            Debug.Log( jsonResponse );
        }
        ApiResponse apiResponse = JsonConvert.DeserializeObject<ApiResponse>(
jsonResponse );
        return apiResponse;
    }

    public static async Task<ApiResponse> RateSprites( string spritesId,
RateSpriteRequest apiRequest )
    {
        HttpWebRequest request = (HttpWebRequest)WebRequest.Create(
"https://sprites-share-api-rwvkn4ky5a-lm.a.run.app/sprites/" + spritesId );
        request.ContentType = "application/json";
        request.Method = "POST";
        using ( var streamWriter = new StreamWriter( request.GetRequestStream() )
)
        {
            streamWriter.Write( JsonConvert.SerializeObject( apiRequest ) );
        }

        HttpWebResponse response = (HttpWebResponse)( await
request.GetResponseAsync() );
        StreamReader reader = new StreamReader( response.GetResponseStream() );
        string jsonResponse = reader.ReadToEnd();
        if( m_IsDebug )
        {
            Debug.Log( jsonResponse );
        }
        ApiResponse apiResponse = JsonConvert.DeserializeObject<ApiResponse>(
jsonResponse );
        return apiResponse;
    }
}
```

And to convert the sprite content back and forth, we also need this class:

```
using UnityEngine;
using System;

public static class ConvertManager
{
    public const TextureFormat AcceptedTextureFormat = TextureFormat.RGBA32;
    public const uint SpritesPixelPerUnit = 16;

    public static string GetStringFromTexture( Texture2D texture2D )
    {
        if( texture2D.format != AcceptedTextureFormat )
        {
            throw new System.Exception( "Unsupported texture format." );
        }

        byte[] byteArray = texture2D.GetRawTextureData();
        string base64String = Convert.ToBase64String(byteArray);
        return base64String;
    }

    public static Texture2D GetTextureFromString( string encodedString, uint
dimensionX, uint dimensionY )
    {
        byte[] newByteArray = Convert.FromBase64String(encodedString);
        Texture2D newTexture = new Texture2D( (int) dimensionX, (int) dimensionY,
AcceptedTextureFormat, false );
        newTexture.filterMode = FilterMode.Point;
        newTexture.LoadRawTextureData( newByteArray );
        newTexture.Apply();
        return newTexture;
    }
    public static Sprite GetSpriteFromString( string encodedString, uint
dimensionX, uint dimensionY )
    {
        Texture2D newTexture = GetTextureFromString( encodedString, dimensionX,
dimensionY );
        Sprite newSprite = Sprite.Create( newTexture, new Rect( 0, 0,
newTexture.width, newTexture.height ), new Vector2( 0.5f, 0.5f ),
SpritesPixelPerUnit );
        return newSprite;
    }
}
```

To be able to use this inside your project, you create all those files and copy the code inside it or just download the SDK available on the project page. With this setup done, we can start working.

GetAllSprites

You can use the GetAllSprites functionality like this:

```
SpriteData[] spritesData = await SpritesController.GetAllSprites();
```

And you can convert the content field into an usable sprite like this:

```
Sprite convertedSprite = ConvertManager.GetSpriteFromString(
spritesData[i].content, spritesData[i].dimensionX, spritesData[i].dimensionY );
```

If you want to fetch sprites based on some criteria, you can use:

```
SpriteData[] spritesData = await SpritesController.GetAllSprites( new
GetAllSpritesRequest(){
   // params
} );
```

GetSprite

You can use the GetSprite functionality like this:

```
SpriteData spritesData = await SpritesController.GetSprites( "spriteId" );
```

AddSprite

You can use the AddSprite functionality like this:

```
await SpritesController.AddSprites( new AddSpriteRequest(){
   author = "author",
   content = ConvertManager.GetStringFromTexture( texture2D ),
   description = "description",
   dimensionX = (uint) texture2D.width,
   dimensionY = (uint) texture2D.height,
   name = "name",
   tags = new string[]{ "tag1", "tag2" },
} );
```

with texture2D as a Texture2D object.

RateSprite

You can use the RateSprite functionality like this:

```
            await SpritesController.RateSprites( "spriteId", new 
RateSpriteRequest()
            {
                rating = myRating
            } );
```

with myRating as a uint.

Last Word

This is only the first version of Sprites Share so, there might be lots of important features that are still missing. I thought about adding a built-in Sprite Editor inside the platform but I gave up on the idea because I realized that building a proper one according to today's standards would be way to time consuming compared to just using an existing solution, like paint. If I get requests about this feature, I might actually add it inside the next version of Sprites Share but it definitely won't be as advanced as Photoshop.

Other than that, if you have any suggestion for this project, you can email at majed.azar7@gmail.com and I'd gladly improve this project. You can also write to me if you have questions.

I hope all of this was helpful. On this last note, have a nice day!