Technical Documentation

GPU Clicker

https://github.com/willhuff0/GPUClicker

Made in Unity.

Summary of the concept: Tap to progress, accumulate different types of points with each click and unlock upgrades to reach higher levels. Clicking the GPU in the middle of the screen generates hashes. Once enough hashes are obtained, a block will be awarded, hashes reset, and the number of hashes required to get the next block increases. To overcome this, the player can prestige to a new cryptocurrency, after which the block difficulty resets and new upgrades are unlocked. Upgrades either generate hashes automatically or increase the effectiveness of tapping.

The app is available for iOS on TestFlight: https://testflight.apple.com/join/lw6eOvD1, and for Android through Google Play internal testing: https://play.google.com/apps/internaltest/4701185352615742940

I used some interesting code to simulate the game running in the background when the app is closed.

I use the singleton pattern often so static logic can be connected to the Unity scene.

```
private void _load()
{
    string json = PlayerPrefs.GetString("save");

    SaveData data;
    if (string.IsNullOrEmpty(json))
    {
        data = new SaveData();
    }
    else
    {
        data = JsonConvert.DeserializeObject<SaveData>(json);
    }

    PrestigeController.Singleton.Load(ref data);
    HashController.Singleton.Load(ref data);
    BlockController.Singleton.Load(ref data);
    MiningPoolClaim.Singleton.Load(ref data);
    CryptoController.Singleton.Load(ref data);
    UpgradeController.Singleton.Load(ref data);
    RigSpecialMenu.Singleton.OnLoad(ref data);


UpgradeController.Singleton.ApplyBackgroundTicks(_getSecondsSinceLastSave());

    _firstLoadComplete = true;
}
```

Apply is called on each upgrade type's scriptable object during UpgradeController's load function.

```
public override void Apply(ulong count, ref ApplyResult result)
{
    result.HashesPerClick *= Utils.IntPow(hashesPerClickMultiplier, count);
    result.HashesPerClick += hashesPerClickAddition * count;
}
```

Utils.IntPow is a very simple optimization for calculating a power.

```
public static double IntPow(double x, ulong pow)
{
    if (pow == 0) return 1;
    if (pow == 1) return x;

    double v = x;
    for (ulong i = 1; i < pow; i++) v *= x;

    return v;
}
```

Windows Automation and Kiosk App

Created for my job at the Goodwill Computer Store.

Open source projects used

- Flutter (https://github.com/flutter/flutter): UI toolkit
- Chocolatey (https://github.com/chocolatey/choco): package manager for installing some default programs we include on the computers we sell

This datastore class is used to store information such as device specs, benchmark scores, and intermediate information to allow resuming after restarts while installing driver updates.

```
abstract class DataStore {
  static String getStorePath(String store) =>
p.join(p.dirname(Platform.resolvedExecutable), 'data_store', '$store.json');

  final String store;

  DataStore(this.store);

  Map<String, dynamic> toMap();

  void fromMap(Map<String, dynamic> map);

  Future<void> save() async {
    final file = File(getStorePath(store));
    await file.create(recursive: true);

    final content = const JsonEncoder.withIndent('    ').convert(toMap());
    await file.writeAsString(content);
  }

  static Future<TDataStore?> read<TDataStore extends DataStore>(TDataStore
Function() creator) async {
    final dataStore = creator();

    final file = File(getStorePath(dataStore.store));
    if (!await file.exists()) return null;

    final content = json.decode(await file.readAsString());
    dataStore.fromMap(content);

    return dataStore;
  }
}
```

Some of the UI is shown in my video.

Nebula

A bare bones game and rendering engine. For rendering, this project uses Google ANGLE to translate OpenGLES to DirectX, Vulkan, or Metal calls. This improves performance compared to native OpenGL drivers. I also wrote a node tree system with parenting to make nodes move together. Nodes can be extended to do things like rendering a mesh. The engine manages serialization of assets and has some built in asset types.

Open source projects used

- ANGLE (https://github.com/google/angle): OpenGLES translation layer
- assimp (https://github.com/assimp/assimp): adds support for importing many types of 3D models
- ImageSharp (https://github.com/SixLabors/ImageSharp): decode and encode textures in compressed formats

ANGLE

Example of a native ANGLE call in Nebula.Graphics, GL.cs

```
using GLenum = UInt32;
using GLsizeiptr = Int64;

public const GLenum ARRAY_BUFFER = 0x8892;
public const GLenum STATIC_DRAW = 0x88E4;
```

```
[LibraryImport(glesDll, EntryPoint = "glBufferData" StringMarshalling =
StringMarshalling.Utf8)]
public static partial void BufferData(GLenum target, GLsizeiptr size, byte[]
data, GLenum usage);
```

```
GL.BufferData(GL.ARRAY_BUFFER, vertexData.LongLength, vertexData,
GL.STATIC_DRAW);
```

bufferData in OpenGLES / ANGLE uploads data to the GPU which contains a list of packed vertex data. Usually, position, texture coordinate, and normal direction information is placed

side-by-side in the array. A call to vertexAttribPointer specifies the exact layout of this information so it can be used correctly in shaders.

RenderShader

This asset loads a vertex and fragment shader for use in material assets, which usually hold a reference to a common shader paired with different textures or parameters.

```
using System.Numerics;
using System.Text.Json.Nodes;
using Nebula.Graphics;

namespace Nebula.Engine.Assets;

[AssetDefinition("renderShaders")]
public class RenderShader : FileAsset
{
    private string[] _imports;

    public RenderShader(Project project, JsonNode json) : base(project, json)
    {
        _imports = json["imports"]?.GetValue<string[]>() ??
Array.Empty<string>();
    }

    public override JsonObject Serialize()
    {
        var json = base.Serialize();
        json["imports"] = JsonValue.Create(_imports);
        return json;
    }

    public override Task<RuntimeAsset?> Load()
    {
        var content = FileAssetGetText();
        var lines = content.Split('\n');

        string preamble = "", vertexSource = "", fragmentSource = "";
        int stage = 0;
        foreach (var line in lines)
        {
            if (line.StartsWith("#VERTEX"))
            {
                stage = 1;
                continue;
            }
            if (line.StartsWith("#FRAGMENT"))
```

```
            {
                stage = 2;
                continue;
            }

            switch (stage)
            {
                case 0:
                    preamble += line + '\n';
                    break;
                case 1:
                    vertexSource += line + '\n';
                    break;
                case 2:
                    fragmentSource += line + '\n';
                    break;
            }
        }

        for (int i = _imports.Length - 1; i >= 0; i--)
        {
            var library = LibraryShader.FindById(_imports[i]);
            if (library != null)
            {
                vertexSource = vertexSource.Insert(0, library.Content + '\n');
                fragmentSource = fragmentSource.Insert(0, library.Content +
'\n');
            }
        }

        vertexSource = vertexSource.Insert(0, preamble + '\n');
        fragmentSource = fragmentSource.Insert(0, preamble + '\n');

        var vertexShader = GL.CreateShader(GL.VERTEX_SHADER);
        GL.ShaderSource(vertexShader, 1, new string[] { vertexSource }, null);
        GL.CompileShader(vertexShader);
        GL.GetShader(vertexShader, GL.COMPILE_STATUS, out int status);
        if (status == GL.FALSE)
        {
```

```
            GL.GetShaderInfoLog(vertexShader, 1024, out int length, out
string infoLog);
            GL.DeleteShader(vertexShader);
            Console.WriteLine($"Error compiling vertex shader ({Name}):
{infoLog}");
            return Task.FromResult<RuntimeAsset?>(null);
        }

        var fragmentShader = GL.CreateShader(GL.FRAGMENT_SHADER);
        GL.ShaderSource(fragmentShader, 1, new string[] { fragmentSource },
null);
        GL.CompileShader(fragmentShader);
        GL.GetShader(fragmentShader, GL.COMPILE_STATUS, out status);
        if (status == GL.FALSE)
        {
            GL.GetShaderInfoLog(fragmentShader, 1024, out int length, out
string infoLog);
            GL.DeleteShader(fragmentShader);
            Console.WriteLine($"Error compiling fragment shader ({Name}):
{infoLog}");
            return Task.FromResult<RuntimeAsset?>(null);
        }

        var program = GL.CreateProgram();
        GL.AttachShader(program, vertexShader);
        GL.AttachShader(program, fragmentShader);
        GL.LinkProgram(program);
        GL.DetachShader(program, vertexShader);
        GL.DetachShader(program, fragmentShader);
        GL.DeleteShader(vertexShader);
        GL.DeleteShader(fragmentShader);
        GL.GetProgram(program, GL.LINK_STATUS, out status);
        if (status == GL.FALSE)
        {
            GL.GetProgramInfoLog(program, 1024, out int length, out string
infoLog);
            GL.DeleteProgram(program);
            Console.WriteLine($"Error linking program ({Name}): {infoLog}");
            return Task.FromResult<RuntimeAsset?>(null);
```

```
        }

        GL.GetProgram(program, GL.ACTIVE_UNIFORMS, out int uniformCount);
        Dictionary<string, int> uniformLocations = new Dictionary<string,
int>(uniformCount);
        for (uint i = 0; i < uniformCount; i++)
        {
            GL.GetActiveUniform(program, i, 1024, out int length, out int
size, out uint type, out string uniformName);
            uniformLocations[uniformName] = GL.GetUniformLocation(program,
uniformName);
        }

        return Task.FromResult<RuntimeAsset?>(new
RuntimeRenderShader(Project, this, program, uniformLocations));
    }
}

public class RuntimeRenderShader : RuntimeAsset
{
    private readonly uint _program;
    private readonly Dictionary<string, int> _uniformLocations;

    public RuntimeRenderShader(Project project, Asset from, uint program,
Dictionary<string, int> uniformLocations) : base(project, from)
    {
        _program = program;
        _uniformLocations = uniformLocations;
    }

    public override Task Unload()
    {
        GL.DeleteProgram(_program);

        return Task.CompletedTask;
    }

    public void Bind() => GL.UseProgram(_program);
```

```
    public void SetUniform(string name, float value) =>
GL.ProgramUniform(_program, _uniformLocations[name], value);
    public void SetUniform(string name, Vector2 value) =>
GL.ProgramUniform(_program, _uniformLocations[name], value.X, value.Y);
    public void SetUniform(string name, Vector3 value) =>
GL.ProgramUniform(_program, _uniformLocations[name], value.X, value.Y,
value.Z);
    public void SetUniform(string name, Vector4 value) =>
GL.ProgramUniform(_program, _uniformLocations[name], value.X, value.Y,
value.Z, value.W);
    public void SetUniform(string name, int value) =>
GL.ProgramUniform(_program, _uniformLocations[name], value);
    public void SetUniform(string name, int v0, int v1) =>
GL.ProgramUniform(_program, _uniformLocations[name], v0, v1);
    public void SetUniform(string name, int v0, int v1, int v2) =>
GL.ProgramUniform(_program, _uniformLocations[name], v0, v1, v2);
    public void SetUniform(string name, int v0, int v1, int v2, int v3) =>
GL.ProgramUniform(_program, _uniformLocations[name], v0, v1, v2, v3);
    public void SetUniform(string name, bool value) =>
GL.ProgramUniform(_program, _uniformLocations[name], value ? 1 : 0);
    public void SetUniform(string name, Matrix4x4 value) =>
GL.ProgramUniformMatrix4(_program, _uniformLocations[name], 1, false, new []{
value.M11, value.M21, value.M31, value.M41, value.M12, value.M22, value.M32,
value.M42, value.M13, value.M23, value.M33, value.M43, value.M14, value.M24,
value.M34, value.M44 });

    public void SetUniform(string name, object value)
    {
        switch (value)
        {
            case int val:
                SetUniform(name, val);
                break;
            case bool val:
                SetUniform(name, val);
                break;
            case float val:
                SetUniform(name, val);
                break;
```

```
            case Vector2 val:
                SetUniform(name, val);
                break;
            case Vector3 val:
                SetUniform(name, val);
                break;
            case Vector4 val:
                SetUniform(name, val);
                break;
            case int[] val:
                switch (val.Length)
                {
                    case 1:
                        SetUniform(name, val[0]);
                        break;
                    case 2:
                        SetUniform(name, val[0], val[1]);
                        break;
                    case 3:
                        SetUniform(name, val[0], val[1], val[2]);
                        break;
                    case 4:
                        SetUniform(name, val[0], val[1], val[2], val[3]);
                        break;
                }
                break;
            case float[] val:
                switch (val.Length)
                {
                    case 1:
                        SetUniform(name, val[0]);
                        break;
                    case 2:
                        SetUniform(name, new Vector2(val[0], val[1]));
                        break;
                    case 3:
                        SetUniform(name, new Vector3(val[0], val[1], val[2]));
                        break;
                    case 4:
```

```
                        SetUniform(name, new Vector4(val[0], val[1], val[2],
val[3]));
                        break;
                    case 16:
                        SetUniform(name, new Matrix4x4(val[0], val[1],
val[2], val[3], val[4], val[5], val[6], val[7], val[8], val[9], val[10],
val[11], val[12], val[13], val[14], val[15]));
                        break;
                }
                break;
        }
    }
}
```

Physically Based Rendering

A fragment shader which uses PBR with the rather common Smith GGX model for specular, Cook-Torrance for fresnel, and Lambert for diffuse.

```
#version 310 es
precision highp float;

out vec4 FragColor;

layout(location = 0) in vec2 v_texCoord;
layout(location = 1) in vec3 v_worldPos;
layout(location = 2) in vec3 v_normal;

layout(binding = 0) uniform sampler2D nebula_texture_albedo;
layout(binding = 1) uniform sampler2D
nebula_texture_metallicRoughnessOcclusion;
layout(binding = 2) uniform sampler2D nebula_texture_normal;

layout(location = 2) uniform vec3 nebula_worldViewPos;

layout(location = 3) uniform vec3 nebula_directionalLight_direction;
layout(location = 4) uniform vec3 nebula_directionalLight_color;
layout(location = 5) uniform float nebula_directionalLight_illuminance;

const float PI = 3.14159265359;

//------------------
//  Specular

// GGX NDF
float specular_distribution(float NoH, float roughness) {
    float a = NoH * roughness;
    float k = roughness / (1.0 - NoH * NoH + a * a);
    return k * k * (1.0 / PI);
}

// Smith GGX geometric shadowing
float specular_visibility(float NoV, float NoL, float roughness) {
    float a2 = roughness * roughness;
    float GGXV = NoL * sqrt(NoV * NoV * (1.0 - a2) + a2);
    float GGXL = NoV * sqrt(NoL * NoL * (1.0 - a2) + a2);
    return 0.5 / (GGXV + GGXL);
}
```

```
// Schlick Cook-Torrance approximation fresnel
vec3 specular_fresnel(float u, vec3 f0) {
    float f = pow(1.0 - u, 5.0);
    return f + f0 * (1.0 - f);
}

//------------------


//------------------
//  Diffuse

// Lambert diffuse
float diffuse_lambert() {
    return 1.0 / PI;
}

//------------------

void main() {
    vec3 baseColor = texture(nebula_texture_albedo, v_texCoord).rgb;

    vec3 inMetallicRoughnessOcclusion =
texture(nebula_texture_metallicRoughnessOcclusion, v_texCoord).rgb;
    float metallic = inMetallicRoughnessOcclusion.r;
    float roughness = inMetallicRoughnessOcclusion.g *
inMetallicRoughnessOcclusion.g;
    float occlusion = inMetallicRoughnessOcclusion.b;

    vec3 diffuseColor = (1.0 - metallic) * baseColor;
    vec3 f0 = 0.16 * 1.0 * 1.0 * (1.0 - metallic) + baseColor * metallic;

    vec3 n = v_normal;
    vec3 v = normalize(v_worldPos - nebula_worldViewPos);
    vec3 l = normalize(-nebula_directionalLight_direction);

    vec3 h = normalize(v + l);
```

```
    float NoV = abs(dot(n, v)) + 1e-5;
    float NoL = clamp(dot(n, l), 0.0, 1.0);
    float NoH = clamp(dot(n, h), 0.0, 1.0);
    float LoH = clamp(dot(l, h), 0.0, 1.0);

    float D = specular_distribution(NoH, roughness);
    vec3 F = specular_fresnel(LoH, f0);
    float V = specular_visibility(NoV, NoL, roughness);

    vec3 specular = (D * V) * F;
    vec3 diffuse = diffuseColor * diffuse_lambert();
    vec3 brdf = diffuse + specular;

    float illuminance = nebula_directionalLight_illuminance * NoL;
    vec3 luminance = brdf * illuminance;

    FragColor = vec4(luminance, 1.0);
}
```

Calculator Snake

My video opens with me playing a snake game on my TI-84 Plus CE. I created it after completing my AP Statistics midterm while I was waiting to leave the testing room.

The program lays out a grid of points using Pt-On and Pt-Off, one of which starts as a snake head, represented as a green point, and one as an apple, represented as a red point.

The apple's position is any random point that doesn't contain a part of the snake.

The snake head moves one space each tick. When colliding with the apple, the snake eats it and its body grows in size by one. A new apple is spawned randomly again.

Since my calculator's smallest unit of time is integer seconds, I am unable to speed up the game over time, as in traditional snake games. The tick rate is fixed at 1 tick per second.

If the snake manages to fill up every space without colliding with its body or the walls, the player wins.

The program uses lists to store the snake's body positions: L1 for X and L2 for Y.

This snippet shows moving each snake body part forward by one if the snake has at least 2 body parts. By doing this each part of the snake will follow the exact path taken by the head as is traditional in snake.

```
If L>=2
Then
For(I,L,2,-1)
L1(I-1)->L1(I)
L2(I-1)->L2(I)
End
End
```

The program is based on a version of snake I wrote a bit earlier called 'file_explorer_snake' (https://github.com/willhuff0/file_explorer_snake). In this version a 10x10 grid of grey images are saved to a directory, the images are then swapped out for different colors creating the screen of the game. This version had a similar problem of having a limited refresh rate. Windows File Explorer refreshes at a fixed rate, so each tick has to happen about every 2 seconds.

I uploaded images of both versions of the game being played.

Music Service

https://github.com/willhuff0/music_fullstack

A full stack music streaming service built from scratch.

The project consists of a server and two client apps. Music is uploaded to the server where it is processed and indexed. Clients can then search and play music which is streamed over the network. Initially, I intended to deploy the server on the WAN, but over time I shifted the project's focus to also work well when hosted locally.

I created this project mostly for fun. Later, I adapted it to solve a specific problem. I wanted a way to synchronize multiple speakers in my home. Other solutions I found were either too expensive or didn't work anymore, so I added the functionally to my already existing music

service. The sync feature works best when hosted locally.

Open source projects used

Server

- Dart (https://github.com/dart-lang)
- Isar (https://github.com/isar/isar): Document database
- dart_phonetics (https://github.com/raycardillo/dart_phonetics): Provides the double metaphone algorithm
- Shelf (https://github.com/dart-lang/shelf): Web Server Middleware for Dart
- Cryptography (https://github.com/dint-dev/cryptography): Provides argon2id, Hmac and sha256 implementations
- dart-uuid (https://github.com/Daegalus/dart-uuid): Uuid generation

Client

- Flutter (https://github.com/flutter/flutter): UI toolkit
- just_audio (https://github.com/ryanheise/just_audio): Audio player for Flutter
- flutter_secure_storage (https://github.com/mogol/flutter_secure_storage): Access to iOS Keychain, etc.
- dyn_mouse_scroll (https://github.com/alesimula/dyn_mouse_scroll): Fixes an issue with scrolling in Flutter
- infinite_scroll_pagination (https://github.com/EdsonBueno/infinite_scroll_pagination): Drop in pagination for scrolling views
- desktop_drop (https://github.com/MixinNetwork/flutter-plugins/tree/main/packages/desktop_drop): Drag and drop files into Flutter app
- synchronized (https://github.com/tekartik/synchronized.dart): Provides locking for async dart code
- flutter_file_picker (https://github.com/miguelpruivo/flutter_file_picker): File picker on desktop

Stateless Server

Worker

The class mainly deals with Dart's threading system. To avoid common multithreading related

bugs, Dart doesn't allow shared memory between threads (yet) which is why, I assume, they call threads Isolates. To exchange information, messages have to be sent over ports which can be a pain and is why I keep all of that logic out of mind in these classes.

```dart
class WorkerManager {
  final Isolate _isolate;
  final Stream<dynamic> fromIsolateStream;
  final SendPort toIsolatePort;

  late final StreamSubscription _fromIsolateSubscription;

  WorkerManager._(this._isolate, this.fromIsolateStream, this.toIsolatePort) {
    _fromIsolateSubscription = fromIsolateStream.listen((message) {});
  }

  static Future<WorkerManager> start(WorkerLaunchArgs args, {String?
debugName}) async {
    final fromIsolatePort = ReceivePort();
    final fromIsolatePortBroadcast = fromIsolatePort.asBroadcastStream();
    final isolate = await Isolate.spawn<(SendPort, WorkerLaunchArgs,
String?)>(
      (message) => WorkerIsolate.spawn(message.$1, message.$2, debugName:
message.$3),
      (fromIsolatePort.sendPort, args, debugName),
      debugName: debugName,
    );
    final toIsolatePort = await fromIsolatePortBroadcast.first;
    return WorkerManager._(isolate, fromIsolatePortBroadcast, toIsolatePort);
  }

  Future<void> shutdown() async {
    toIsolatePort.send('shutdown');
  }
}

class WorkerIsolate {
  final Worker _worker;
  final Stream<dynamic> _fromManagerStream;
  final SendPort _toManagerPort;

  late final StreamSubscription _fromManagerSubscription;

  WorkerIsolate._(this._worker, this._fromManagerStream, this._toManagerPort)
```

```
{
    _fromManagerSubscription = _fromManagerStream.listen((message) {
      switch (message) {
        case 'shutdown':
          _shutdown();
          break;
      }
    });
  }

  static Future<WorkerIsolate> spawn(SendPort toManagerPort, WorkerLaunchArgs
args, {String? debugName}) async {
    final fromManagerPort = ReceivePort();
    toManagerPort.send(fromManagerPort.sendPort);
    final fromManagerStream = fromManagerPort.asBroadcastStream();

    final worker = await args.start(args, fromManagerStream, toManagerPort,
debugName: debugName);

    return WorkerIsolate._(worker, fromManagerStream, toManagerPort);
  }

  Future<void> _shutdown() async {
    _worker.shutdown();
    _fromManagerSubscription.cancel();
  }
}

abstract interface class Worker {
  Future<void> shutdown();
}

class WorkerLaunchArgs {
  Future<Worker> Function(WorkerLaunchArgs args, Stream<dynamic>
fromManagerStream, SendPort toManagerPort, {String? debugName}) start;
  final ServerConfig config;

  WorkerLaunchArgs({required this.start, required this.config});
}
```

```
class WorkerLaunchArgsWithAuthentication extends WorkerLaunchArgs {
  final Uint8List privateKey;

  WorkerLaunchArgsWithAuthentication({required super.start, required
super.config, required this.privateKey});
}
```

APIWorker

This class implements the Worker interface and allows me to provide a list of CustomHandler objects which take a route and a closure to be executed when an http request is received. I extend CustomHandlerBase to CustomHandlerAuthRequired which inserts some extra code before calling the provided closure to automatically handle session token verification. Since the closure may be executed on some arbitrary thread, CustomThreadData is passed to allow access to some thread bound state, such as the Isar database reference.

```
class APIWorker implements Worker {
  final APIWorkerLaunchArgs _args;
  final HttpServer _server;
  final Router _router;
  final CustomThreadData _threadData;

  APIWorker._(this._server, this._router, this._threadData, this._args) {
    for (final customHandler in _args.customHandlers) {
      addCustomHandler(customHandler);
    }
  }

  static Future<Worker> start(WorkerLaunchArgs args, Stream<dynamic>
fromManagerStream, SendPort toManagerPort, {String? debugName}) async {
    if (args is! APIWorkerLaunchArgs) throw Exception('APIWorker must be
started with APIWorkerLaunchArgs');

    final threadData = await args.createThreadData();

    final router = Router();
    final handler = Pipeline().addMiddleware(logRequests(logger: debugName !=
null ? (message, isError) => print('[$debugName] $message') :
null)).addHandler(router.call);
    final server = await serve(handler, args.config.address,
args.config.port, shared: true);

    return APIWorker._(server, router, threadData, args);
  }

  void addCustomHandler(CustomHandlerBase customHandler) =>
_router.all(customHandler.path, customHandler.createHandler(_threadData));

  @override
  Future shutdown() async {
    await _server.close();
    _args.onClose(_threadData);
  }
}
```

```
abstract class CustomHandlerBase<TThreadData extends CustomThreadData> {
  final String path;

  CustomHandlerBase({required this.path});

  FutureOr<Response> Function(Request request) createHandler(TThreadData
threadData);
}

class CustomHandler<TThreadData extends CustomThreadData> extends
CustomHandlerBase<TThreadData> {
  final FutureOr<Response> Function(Request request, TThreadData threadData)
handle;

  CustomHandler({required super.path, required this.handle});

  @override
  FutureOr<Response> Function(Request request) createHandler(TThreadData
threadData) => (request) => handle(request, threadData);
}

class CustomHandlerAuthRequired<TClaims extends IdentityTokenClaims,
TThreadData extends CustomThreadDataWithAuth<TClaims>> extends
CustomHandlerBase<TThreadData> {
  final FutureOr<Response> Function(Request request, TThreadData threadData,
IdentityToken<TClaims> identityToken) handle;

  CustomHandlerAuthRequired({required super.path, required this.handle});

  @override
  FutureOr<Response> Function(Request request) createHandler(TThreadData
threadData) => (request) {
        final encodedToken = request.headers['token'];
        if (encodedToken == null) return Response.forbidden('');
        final identityToken =
threadData.identityTokenAuthority.verifyAndDecodeToken(encodedToken);
        if (identityToken == null) return Response.forbidden('');

        return handle(request, threadData, identityToken);
```

```
      };
}

abstract class CustomThreadData {}

class CustomThreadDataWithAuth<TClaims extends IdentityTokenClaims> extends
CustomThreadData {
  final IdentityTokenAuthority<TClaims> identityTokenAuthority;

  CustomThreadDataWithAuth({required this.identityTokenAuthority});
}

class APIWorkerLaunchArgs extends WorkerLaunchArgs {
  final FutureOr<CustomThreadData> Function() createThreadData;
  final List<CustomHandlerBase> customHandlers;
  final FutureOr<void> Function(CustomThreadData threadData) onClose;

  APIWorkerLaunchArgs({
    required super.config,
    required this.createThreadData,
    required this.onClose,
    this.customHandlers = const [],
  }) : super(start: APIWorker.start);
}
```

The list of handlers used in music_server (note some functional programming aspects: the handle variable takes a Function object)

```
final musicServerCustomHandlers = [
  CustomHandler(path: '/status', handle: statusHandler),
  CustomHandler(path: '/speedTest/<size>', handle: speedTestHandler),

  // Auth
  CustomHandler(path: '/auth/createUser', handle: authCreateUserHandler),
  CustomHandler(path: '/auth/startSession', handle: authStartSessionHandler),
  CustomHandlerAuthRequired(path: '/auth/getName', handle:
authGetNameHandler),
  CustomHandlerAuthRequired(path: '/auth/searchUser', handle:
authSearchUserHandler),

  // Song
  CustomHandlerAuthRequired(path: '/song/create', handle: songCreateHandler),
  CustomHandlerAuthRequired(path: '/song/uploadData', handle:
songUploadDataHandler),
  CustomHandlerAuthRequired(path: '/song/uploadImage', handle:
songUploadImageHandler),
  CustomHandlerAuthRequired(path: '/song/uploadDone', handle:
songUploadDoneHandler),
  CustomHandlerAuthRequired(path: '/song/getData', handle:
songGetDataHandler),
  CustomHandler(path: '/song/getImage/<songId>/<size>', handle:
songGetImageHandler),
  CustomHandlerAuthRequired(path: '/song/search', handle: songSearchHandler),
  CustomHandlerAuthRequired(path: '/song/filter', handle: songFilterHandler),
  CustomHandlerAuthRequired(path: '/song/popular', handle:
songPopularHandler),

  // Sync
  CustomHandlerAuthRequired(path: '/sync/startOrJoinSession', handle:
syncStartOrJoinSessionHandler),
];
```

Sessions

Each time the server starts, it generates a new random key to use with Hmac. If this server was deployed for real, this key should be rotated at runtime and destroyed properly so it doesn't sit in memory for too long waiting to be garbage collected. With that said, I'm sure

there are plenty of other security vulnerabilities dotted around.

```
final privateKey = generateSecureRandomKey(config.tokenKeyLength);
```

The IdentityToken is analogous to a JWT (JSON Web Token), and is generated and signed by the server then stored on the client. On each API request, inside CustomHandlerAuthRequired, the tokens signature is verified to be authentic by comparing it to a freshly generated one based on the token's data. Tokens cannot be modified or forged as the signature would no longer match the token's data. In order to generate a valid signature, one would need to know the privateKey ^above.

Example of an IdentityToken and its signature:

```
{
    "uid": "018cd0d2-a69b-7bbe-a056-650ece3dcfb7",
    "name": "Will",
    "time": "2023-12-20T00:29:18.997310Z",
    "ip": "127.0.0.1",
    "agent": "PostmanRuntime/7.36.0",
    "key": <random bytes>,
    "claims": {
        "tier": 1
    }
}
```

```
0BClJqZtsVacCRHuf3+3MZIzoerHRJz75+5zdfxCRkY=
```

Passwords are hashed with Argon2 which is future proof against quantum computers.

```
final _passwordHashingAlgorithm = Argon2id(
  parallelism: 1,
  memory: 19456,
  iterations: 2,
  hashLength: 32,
);
```

```
@embedded
class HashedUserPassword {
  final List<byte> nonce;
  final List<byte> hash;

  HashedUserPassword({
    this.nonce = const [],
    this.hash = const [],
  });

  static Future<HashedUserPassword> createNew(String password) async {
    final passwordNonce = generateSecureRandomKey(4);
    final passwordHash = await
_passwordHashingAlgorithm.deriveKeyFromPassword(password: password, nonce:
passwordNonce).then((value) => value.extractBytes());
    return HashedUserPassword(hash: passwordHash, nonce: passwordNonce);
  }

  Future<bool> checkPasswordMatch(String password) async {
    final passwordHash = await
_passwordHashingAlgorithm.deriveKeyFromPassword(password: password, nonce:
nonce).then((value) => value.extractBytes());
    return ListEquality().equals(passwordHash, hash);
  }
}
```

Transcoding

A TranscodeWorker will acquire a new TranscodeOperation, which is inserted into the database when a song finishes uploading, and begin processing the audio.

My audio transcoding process is very procedural, the function mainly consists of verification and error checking. I use ffmpeg's loudnorm filter to normalize the audio so that one song isn't louder than any others. Then I transcode to three different qualities with both libopus and libfdk_aac (aac wouldn't be necessary if Apple used common standards, which would save more than half of the storage space since aac files are larger for the same quality).

```
Future<String?> processAudio({required MusicServerPaths paths, required
String inputFile, required String outputDir, required List<AudioPreset>
presets}) async {
  try {
    // get channel count
    final ffprobeResult = await Process.run(paths.ffprobePath, ['-v',
'quiet', '-print_format', 'json', '-show_streams', '-select_streams', 'a:0',
inputFile]);
    if (ffprobeResult.exitCode != 0) return 'ffprobe exited with an error
(${ffprobeResult.exitCode}): ${ffprobeResult.stderr}';

    final inputAudioFileInfoString = ffprobeResult.stdout as String?;
    if (inputAudioFileInfoString == null || inputAudioFileInfoString.isEmpty)
return 'ffprobe did not provide a result';

    final inputAudioFileInfo = jsonDecode(inputAudioFileInfoString) as
Map<String, dynamic>?;
    if (inputAudioFileInfo == null) return 'ffprobe did not return json';

    final inputAudioStreams = inputAudioFileInfo['streams'] as List<dynamic>?;
    if (inputAudioStreams == null) return 'ffprobe found no audio streams';

    final inputAudioFirstStream = inputAudioStreams.firstOrNull as
Map<String, dynamic>?;
    if (inputAudioFirstStream == null) return 'ffprobe found no audio
streams';

    final inputAudioChannelCount = inputAudioFirstStream['channels'] as int?;
    if (inputAudioChannelCount == null) return 'ffprobe output was
malformated: first audio stream does not contain \'channels\'';

    if (inputAudioChannelCount > 2) return 'first audio stream has more than 2
channels';

    // normalize audio
    final loudnormResult = await Process.run(paths.ffmpegPath, ['-i',
inputFile, '-y', '-hide_banner', '-nostats', '-filter:a',
'loudnorm=print_format=json', '-f', 'null', 'NULL']);
    if (loudnormResult.exitCode != 0) return 'ffmpeg loudnorm exited with an
```

```
error (${loudnormResult.exitCode}): ${loudnormResult.stderr}';

    final loudnormResultString = loudnormResult.stderr as String?;
    if (loudnormResultString == null || loudnormResultString.isEmpty) return
'ffmpeg loudnorm did not provide a result';

    final loudnormStartIndex = loudnormResultString.indexOf('{',
loudnormResultString.indexOf('Parsed_loudnorm_0'));
    final loudnormEndIndex = loudnormResultString.indexOf('}',
loudnormStartIndex);
    final loudnormParsedResultString =
loudnormResultString.substring(loudnormStartIndex, loudnormEndIndex + 1);
    if (loudnormParsedResultString.isEmpty) return 'ffmpeg loudnorm did not
provide a result or the result was malformed';

    final loudnormParsedResult = jsonDecode(loudnormParsedResultString) as
Map<String, dynamic>?;
    if (loudnormParsedResult == null) return 'ffmpeg loudnorm did not provide
a result or the result was malformed';

    final measured_i = loudnormParsedResult['input_i'] as String?;
    if (measured_i == null || measured_i.isEmpty) return 'ffmpeg loudnorm
result was malformed, input_i was null or empty';

    final measured_tp = loudnormParsedResult['input_tp'] as String?;
    if (measured_tp == null || measured_tp.isEmpty) return 'ffmpeg loudnorm
result was malformed, input_tp was null or empty';

    final measured_lra = loudnormParsedResult['input_lra'] as String?;
    if (measured_lra == null || measured_lra.isEmpty) return 'ffmpeg loudnorm
result was malformed, input_lra was null or empty';

    final measured_thresh = loudnormParsedResult['input_thresh'] as String?;
    if (measured_thresh == null || measured_thresh.isEmpty) return 'ffmpeg
loudnorm result was malformed, input_thresh was null or empty';

    const target_i = '-14.0';
    const target_tp = '-2.0';
    const target_lra = '7.0';
```

```
    const target_offset = '0.0';

    final intermediateInputFile =
'${p.withoutExtension(inputFile)}_normalized${p.extension(inputFile)}';
    try {
      final loudnormStep2Result = await Process.run(paths.ffmpegPath, ['-i',
inputFile, '-y', '-map_metadata', '-1', '-map', '0:a', '-filter:a',
'loudnorm=linear=true:i=$target_i:lra=$target_lra:tp=$target_tp:offset=$target_offset:meas
intermediateInputFile]);
      if (loudnormStep2Result.exitCode != 0) return 'ffmpeg loudnorm step 2
exited with an error (${loudnormStep2Result.exitCode}):
${loudnormStep2Result.stderr}';

      // transcode with each preset
      Future<String?> processPreset(AudioPreset preset) async {
        final outputFile = getAudioOutputFilePath(outputDir, preset);

        final encodeParameters = switch (preset.format) {
          CompressedAudioFormat.opus => ['-i', intermediateInputFile, '-y',
'-v', 'warning', '-progress', '-', '-codec:a', 'libopus', '-mapping_family',
'0', '-b:a', (preset.quality.opuskbpsPerChannel * 1000 *
inputAudioChannelCount).toString(), outputFile],
          CompressedAudioFormat.aac => ['-i', intermediateInputFile, '-y',
'-v', 'warning', '-progress', '-', '-codec:a', 'libfdk_aac', '-vbr',
preset.quality.aacVBRMode.toString(), outputFile],
        };
        final transcodeResult = await Process.run(paths.ffmpegPath,
encodeParameters);
        if (transcodeResult.exitCode != 0) return 'ffmpeg transcode on preset
$preset exited with an error (${transcodeResult.exitCode}):
${transcodeResult.stderr}';

        return null;
      }

      Future<void> revert() async {
        for (final preset in presets) {
          final outputFileObject = File(getAudioOutputFilePath(outputDir,
preset));
```

```
          if (await outputFileObject.exists()) await
outputFileObject.delete();
        }
      }

      await Directory(p.join(outputDir, 'audio')).create(recursive: true);

      try {
        if (transcodePresetsSimultaneously) {
          final processResultsFutures = presets.map(processPreset);
          final processResults = await Future.wait(processResultsFutures);
          for (final processResult in processResults) {
            if (processResult != null) {
              await revert();
              return processResult;
            }
          }
        } else {
          for (final preset in presets) {
            final processResult = await processPreset(preset);
            if (processResult != null) {
              await revert();
              return processResult;
            }
          }
        }
      } catch (e) {
        await revert();
        return e.toString();
      }
    } finally {
      final intermediateInputFileObject = File(intermediateInputFile);
      if (await intermediateInputFileObject.exists()) await
intermediateInputFileObject.delete();
    }
  } catch (e) {
    return e.toString();
  }
```

```
  return null;
}
```

Image processing is similar and uses ImageMagick.

```
final encodeParameters = [inputFile, '-resize', '${size.resolution}^',
'-gravity', 'center', '-extent', size.resolution, outputFile];

final transcodeResult = await Process.run(paths.magickPath, encodeParameters);
```

Search

This snippet from songSearchHandler preforms a query on the stored and indexed phonetic codes of each song's name. Isar generates functions with names based on the variables in the data model. For example, in "sortByPopularityDesc" popularity is a double stored in each song document. Searching on indexes is much faster than filtering each document, and using phonetic codes allows that search to still be fuzzy (such as allowing a spelling error in a song title).

```
final queryPhonetics = getPhoneticCodesOfQuery(queryString);
final searchResults = threadData.isar.songs.where().anyOf(queryPhonetics, (q,
element) =>
q.namePhoneticsElementEqualTo(element)).sortByPopularityDesc().offset(start).limit(limit)

return Response.ok(jsonEncode(searchResults.map((song) =>
song.toJson()).toList()));
```

Sync Session

Here's a tongue twister: SyncSessionWorkers manage web socket servers that mediate sync sessions.

Sync sessions allow multiple clients signed in as the same user to play music at the same time. Over the LAN, this can be quite precise.

The server serves mainly as a relay to allow clients to send messages to each other, but some important logic is preformed in this case statement inside _onRequest. The 'effective' time is

in the future - it is the exact time the clients should begin playback.

```
case 'callTimeSensitive':
  final call = json['call'];
  final effective = DateTime.timestamp().add(Duration(milliseconds:
1000)).microsecondsSinceEpoch;
  final response = jsonEncode({
    'method': 'callTimeSensitive',
    'call': call,
    'effective': effective,
  });
  for (final client in clients) {
    client.sink.add(response);
  }
  break;
```

Client side, the difference between the server and client devices' clocks is calculated.

```
void _timeResponse(dynamic json) {
    final clientReceiveTimestamp = DateTime.timestamp();

    if (_clientSendTimestamp == null) return;

    final serverSentTimestamp =
DateTime.fromMicrosecondsSinceEpoch(json['sent']);

    final roundTripDuration =
clientReceiveTimestamp.difference(_clientSendTimestamp!);
    final fromServerDuration =
clientReceiveTimestamp.difference(serverSentTimestamp);

    latency = roundTripDuration.inMicroseconds ~/ 2;
    difference = fromServerDuration.inMicroseconds - latency;

    syncSessionChangedController.add(syncSession);
  }
```

The difference value is then used determine how long the client should wait before beginning playback.

```
void _playResponse(dynamic json) async {
    _clientSendTimestamp = null;
    final effective = DateTime.fromMicrosecondsSinceEpoch(json['effective'] +
difference);
    final microsecondsUntilCall =
effective.difference(DateTime.timestamp()).inMicroseconds;
    sleep(Duration(microseconds: microsecondsUntilCall));
    await appPlayer.play();
  }
```

Mobile and Web Clients

Visuals

This is the main logic of a gradient shader I use as a background nearly everywhere in the music service apps. Each of the arrays, uPointPositions, uPointSizes, and uPointColors, contain 4 elements. Each fragment simply measures its distance to each of the points and adds that point's color multiplied by a distance based falloff.

```
float falloff(float dist, float size) {
    return 1 - clamp(dist / size, 0.0, 1.0);
}

void main() {
    vec2 fragPosition = FlutterFragCoord().xy;

    vec4 totalColor = vec4(0.0);
    for(int i = 0; i < numPoints; i++) {
        vec2 pointPosition = uPointPositions[i];
        float pointSize = uPointSizes[i];
        vec3 pointColor = uPointColors[i];

        float dist = distance(pointPosition, fragPosition);
        float intensity = falloff(dist, pointSize);

        totalColor += vec4(pointColor, 1.0) * intensity;
    }

    fragColor = totalColor * outputIntensity;
}
```

On the Dart side, I generate random positions with a simple Poisson Disk sample.

```
/// Simple Poisson Disk Sampling. Generates random points until count valid
points are found. Fails after 100 unsuccessful samples.
List<(double x, double y)> poissonDiskSample({required int count, required
double radius, required double sizeX, required double sizeY}) {
  final points = [(_random.nextDouble() * sizeX, _random.nextDouble() *
sizeY)];

  var i = 1;
  for (var k = 0; k < 100; k++) {
    final sample = (_random.nextDouble() * sizeX, _random.nextDouble() *
sizeY);

    var valid = true;
    for (final point in points) {
      final difference = (sample.$1 - point.$1, sample.$2 - point.$2);
      final distance = sqrt(difference.$1 * difference.$1 + difference.$2 *
difference.$2);

      if (distance < radius) {
        valid = false;
        break;
      }
    }
    if (valid) {
      points.add(sample);
      i++;
      if (i >= count) break;
    }
  }

  // If no more points can be generated, force random sample
  if (points.length < count) points.addAll(List.generate(count -
points.length, (index) => (_random.nextDouble() * sizeX, _random.nextDouble()
* sizeY)));

  return points;
}
```

Then I take two samples and interpolate between them.

```
final pointPositions = lerpPositions(pointPositionsA!, pointPositionsB!,
animationController.value);
```

I uploaded an image of the gradient with some red and blue colors which is labeled 'Music Service Gradient Background'. More demonstrations are in my video.