第2回 Unreal Engine KYUSHU LT会

Unreal Engine で OAuth 2.0 JWT Bearer Token Flow

(Google Cloud APIにアクセスしてみよう)

ぽちお (FutureSoftware 和田敏幸)

2024/7/6

自己紹介

妻の作品「わたしのお母さん」
単行本発売中です！！
UE5で一部背景描いてます。

FutureSoftware　和田 敏幸　(ぽちお)

X(Twitter) ：@toshiyuki_wada

静岡でフリーランスのシステム屋をやってます。

家で焼きラーメンを作る時は、必ず「マルタイの棒ラーメン」を使用するくらいに九州が好きです。

一児のパパでぷちコン勢　(第17回:ささき賞 第21回:グッド演出で賞受賞させて頂きました)

WEBアプリ(フロント・バックエンド)・スマホアプリ・インフラ構築・クラウド環境構築などなどシステムの事ならなんでもやってます。(UEを使いこなすシステム屋としてお仕事お待ちしております)

月１回のペースでUnreal Engineの特定の技術について語り合う「UE5 Night」の運営を始めました。興味のある方はDiscordサーバでお待ちしています。(7/14に第１回「UE5 Rig Night」を開催予定)

また、UE5ぷちコンの(非公式)打ち上げイベント「ぷちアゲ」の企画・幹事も担当しています。誰でも参加可能なイベントとなっていますので、参加お待ちしております。(詳しくは私のXまで！)

OAuth 2.0 JWT Bearer Token Flowって何？

一言で言えば

「APIサービスを利用する為の認可の流れ」

(PCとサーバ間でのやり取り)

です。

Google Cloud API/Microsoft各種API/Salesforce REST API/Net Suiteなど様々なAPIサービスを利用する為の標準的な認可の流れで利用されています。

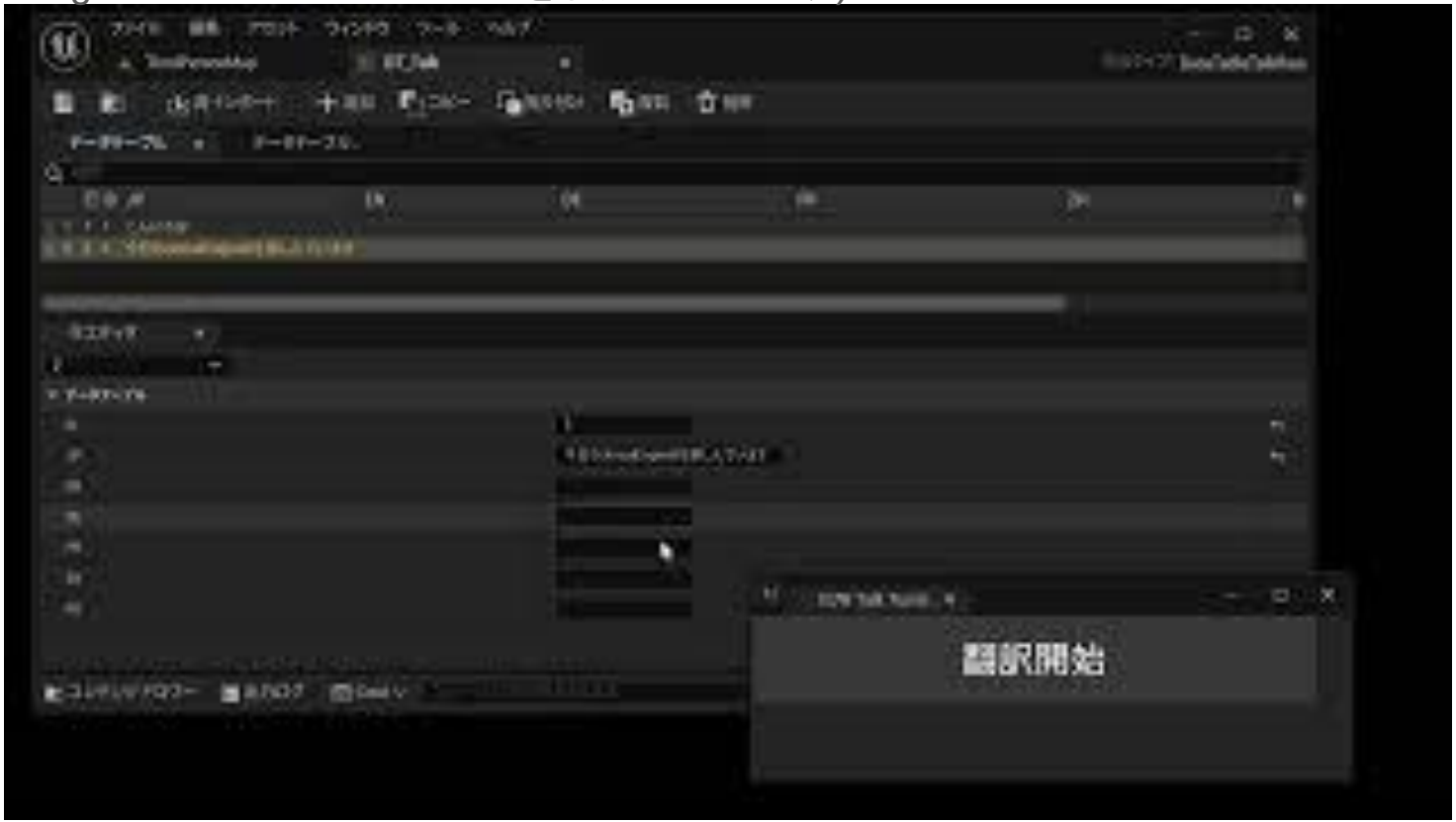
つまり、これを実装することができれば、UnrealEngineから様々なAPIサービスを使いこなすことができるんです！

事例１：UE5 で AI OCR & 翻訳

AI-OCRと翻訳APIを組み合わせることで、プレイヤーの視界に入った文字を認識し英語に翻訳することも可能となります。(Google Cloud Vision(Vision AI)& Translation APIを利用しています)

事例２：UE5 の Editor Utility Widgetで一括翻訳

ゲームなどのメッセージファイルの多言語化対応をAPI経由で終わらせてしまうという方法もあります。(Google Cloud Translation APIを利用しています)

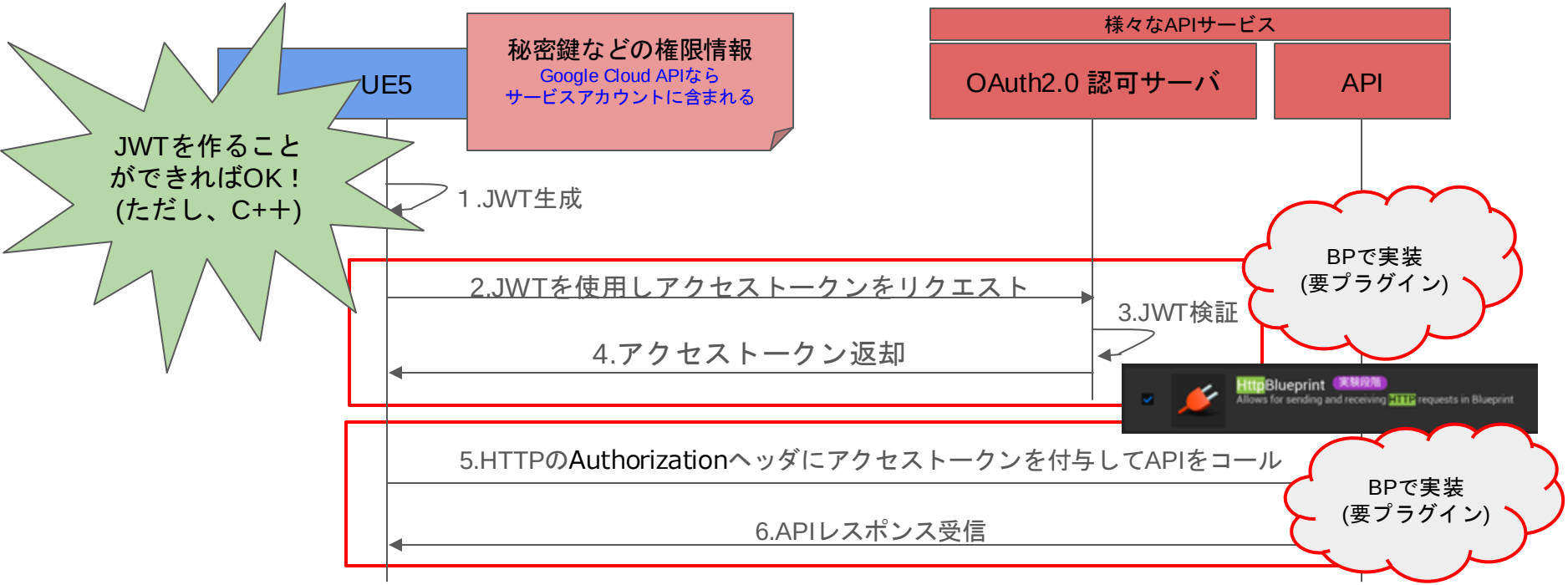


処理の流れ

OAuth 2.0 JWT Bearer Token Flowを、Google Cloud APIへのアクセスを例に解説していきます。(他のAPIサービスでも基本的には同じフローです)

本解説では以下の流れを実装していきますが、Google Cloud API に限っていえばC++用のライブラリがある為、そのライブラリを使った方が実装が簡単な場合もあります。

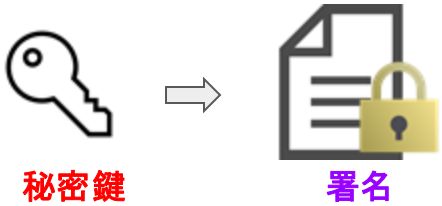
また、Google Cloud APIへのアクセスには「サービスアカウント」という仕組みを利用します。事前にコンソールから作成しておいてください。



JWTとは？

JSON Web Tokenの略で、「ヘッダ」「ペイロード」「署名」の３つの要素から構成された文字列です。

具体的には、以下の内容のJSONをBase64化(データを文字列化)したものになります。

ヘッダ	ペイロード	署名 (ヘッダとペイロード(のハッシュ)を秘密鍵で署名)
{ 'alg': 'RS256', 'typ':'JWT', 'kid': **Key ID** }	{ 'iss': **エンティティを識別情報**, 'scope': **トークンが許可する操作やリソース**, 'aud': **トークンの対象となる受信者**, 'iat': **トークンが発行された時刻**, 'exp': **トークンの有効期限** }	

秘密鍵　　署名

赤文字の部分：サービスアカウントから取得可能

字の部分：API利用方法に記載

の部分：プログラムで生成可能

JSON作れればOK
(なんとなくできそう)

秘密鍵で署名？
そんなんどーすんの？

安心してください！入ってますよ！

UnrealEngineには最初からこの処理に必要なOpenSSLが入ってますので、

Build.csに”OpenSSL”を定義するだけで秘密鍵で署名できるようになります！！

ということで、C++のBuild.csはこんな感じになります。

```
// Fill out your copyright notice in the Description page of Project Settings.

using UnrealBuildTool;
using System.IO;

public class GoogleCloudAPI : ModuleRules
{
    public GoogleCloudAPI(ReadOnlyTargetRules Target) : base(Target)
    {
        PCHUsage = PCHUsageMode.UseExplicitOrSharedPCHs;

        PublicDependencyModuleNames.AddRange(new string[] { "Core", "CoreUObject", "Engine", "InputCore", "OpenSSL", "Json", "JsonUtilities", "HTTP" });

        PrivateDependencyModuleNames.AddRange(new string[] {  });
    }
}
```

OpenSSL：署名に使用
Json&JsonUtilities：JWTの作成＆APIへのアクセスに使用
HTTP：トークンの取得・APIへのアクセス(今回は不使用)

JWT生成処理の実装を開始する１

１．ヘッダ・ペイロードの作成

```
FString GoogleCloudOAuth::CreateJWTHeader(FString KeyID)
{
  TSharedPtr<FJsonObject> Header = MakeShareable(new FJsonObject);
  Header->SetStringField("alg", "RS256");
  Header->SetStringField("typ", "JWT");
  Header->SetStringField("kid", KeyID);

  FString HeaderString;
  TSharedRef<TJsonWriter<>> Writer = TJsonWriterFactory<>::Create(&HeaderString);
  FJsonSerializer::Serialize(Header.ToSharedRef(), Writer);
  return FBase64::Encode(HeaderString);
}
```

ヘッダのJSON

```
{
  'alg': 'RS256',
  'typ':'JWT',
  'kid': Key ID
}
```

```
FString GoogleCloudOAuth::CreateJWTPayload(FString Issuer, FString Scope, FString Audience)
{
  FDateTime Now = FDateTime::UtcNow();
  TSharedPtr<FJsonObject> Payload = MakeShareable(new FJsonObject);
  Payload->SetStringField("iss", Issuer);
  Payload->SetStringField("scope", Scope);
  Payload->SetStringField("aud", Audience);
  Payload->SetNumberField("iat", Now.ToUnixTimestamp());
  Payload->SetNumberField("exp", Now.ToUnixTimestamp() + 3600);

  FString PayloadString;
  TSharedRef<TJsonWriter<>> Writer = TJsonWriterFactory<>::Create(&PayloadString);
  FJsonSerializer::Serialize(Payload.ToSharedRef(), Writer);
  return FBase64::Encode(PayloadString);
}
```

現在の時間とアクセストークンの有効期限はUNIXエポック秒で指定

ペイロードのJSON

```
{
  'iss': エンティティを識別情報,
  'scope':トークンが許可する操作やリソース,
  'aud': トークンの対象となる受信者,
  'iat': トークンが発行された時刻,
  'exp': トークンの有効期限
}
```

JWT生成処理の実装を開始する２

２．秘密鍵(PEM形式)をRSA鍵データに変換する

```
RSA* GoogleCloudOAuth::LoadPrivateKeyFromString(const FString& PrivateKeyString)
{
  // 変数定義
  BIO *bio;
  RSA *rsa_key = NULL;

  // 秘密鍵(PEM形式)の改行コード文字列を実際の改行に置換後Char配列に変換する
  TArray<char> CharArray;
  ConvertFStringToCharArray(PrivateKeyString.Replace(TEXT("¥¥n"), TEXT("¥n")), CharArray);

  // バッファから BIO オブジェクトを作成
  bio = BIO_new_mem_buf((void *)CharArray.GetData(), -1);
  if (bio == NULL) {
    return NULL;
  }

  // 秘密鍵を読み込む
  rsa_key = PEM_read_bio_RSAPrivateKey(bio, NULL, NULL, NULL);
  if (rsa_key == NULL) {
    UE_LOG(LogTemp, Error, TEXT("Failed to read private key PEM_read_bio_RSAPrivateKey Failed"));
  }

  // BIO を解放
  BIO_free(bio);

  return rsa_key;
}
```

```
#include <openssl/rsa.h>
#include <openssl/pem.h>
```

秘密鍵

JWT生成処理の実装を開始する３

３．ヘッダ・ペイロードとRSA鍵データを渡して署名する処理の作成

```
FString GoogleCloudOAuth::CreateJWTSignature(const FString& Header, const FString& Payload, RSA* PrivateKey)
{
  // ヘッダ・ペイロードの連結文字列からメッセージ作成
  FString Message = Header + "." + Payload;
  TArray<uint8> MessageBytes;
  MessageBytes.Append((uint8*)TCHAR_TO_UTF8(*Message), Message.Len());

  // メッセージのSHA-256ハッシュを求める
  uint8 Digest[SHA256_DIGEST_LENGTH];
  SHA256(MessageBytes.GetData(), MessageBytes.Num(), Digest);

  // 署名生成
  uint8 Signature[256]; // RSA 2048 bit の署名サイズ
  unsigned int SignatureLen;
  if (!RSA_sign(NID_sha256, Digest, SHA256_DIGEST_LENGTH, Signature, &SignatureLen, PrivateKey))
  {
    UE_LOG(LogTemp, Error, TEXT("Failed to sign the message"));
    return "";
  }

  // 署名のエンコードを返却する
  return FBase64::Encode(Signature, SignatureLen);
}
```

JWT生成処理の実装を開始する4

４．JWT作成

```
FString GoogleCloudOAuth::CreateJWTToken()
{
  // ヘッダ・ペイロードの作成
  FString Header = CreateJWTHeader(this->KeyID);
  FString Payload = CreateJWTPayload(this->Issuer, this->Scope, this->Audience);

  // RSA鍵の作成
  RSA* PrivateKey = LoadPrivateKeyFromString(this->PrivateKeyString);

  // 鍵の生成チェック
  if (!PrivateKey)
  {
    UE_LOG(LogTemp, Error, TEXT("Failed to load private key"));
    return "";
  }

  // 署名の作成
  FString Signature = CreateJWTSignature(Header, Payload, PrivateKey);
  RSA_free(PrivateKey);

  // 全てを連結しJWT作成完了
  return Header + "." + Payload + "." + Signature;
}
```

ヘッダ・ペイロード作成

RSA鍵作成 

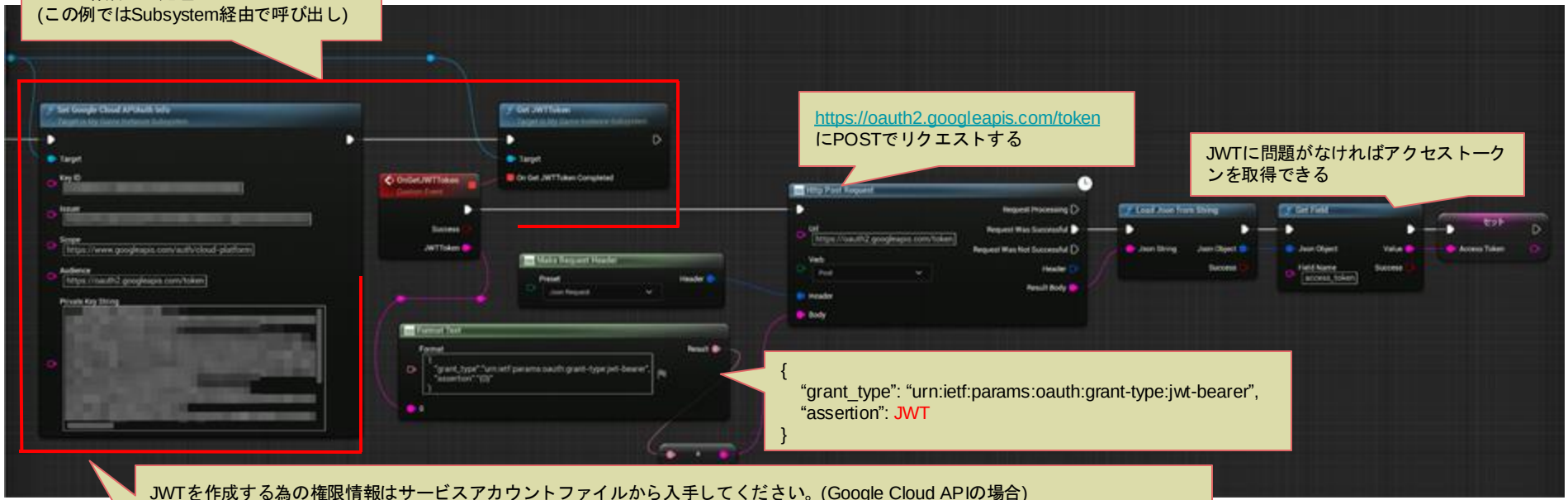
署名 

文字列連結でJWT完成！

アクセストークン取得

C++で作成したJWTを使用して、Google CloudのOAuth2.0 認可サーバからアクセストークンを取得する処理はBPで実装します。
(JWT作成処理はBlueprintFunctionLibraryやSubsystemに作っておくと、BP側から呼び出しやすいです)

C++で作成した処理
(この例ではSubsystem経由で呼び出し)



https://oauth2.googleapis.com/token
にPOSTでリクエストする

JWTに問題がなければアクセストークンを取得できる

```
{
  “grant_type”: “urn:ietf:params:oauth:grant-type:jwt-bearer”,
  “assertion”: JWT
}
```

JWTを作成する為の権限情報はサービスアカウントファイルから入手してください。(Google Cloud APIの場合)

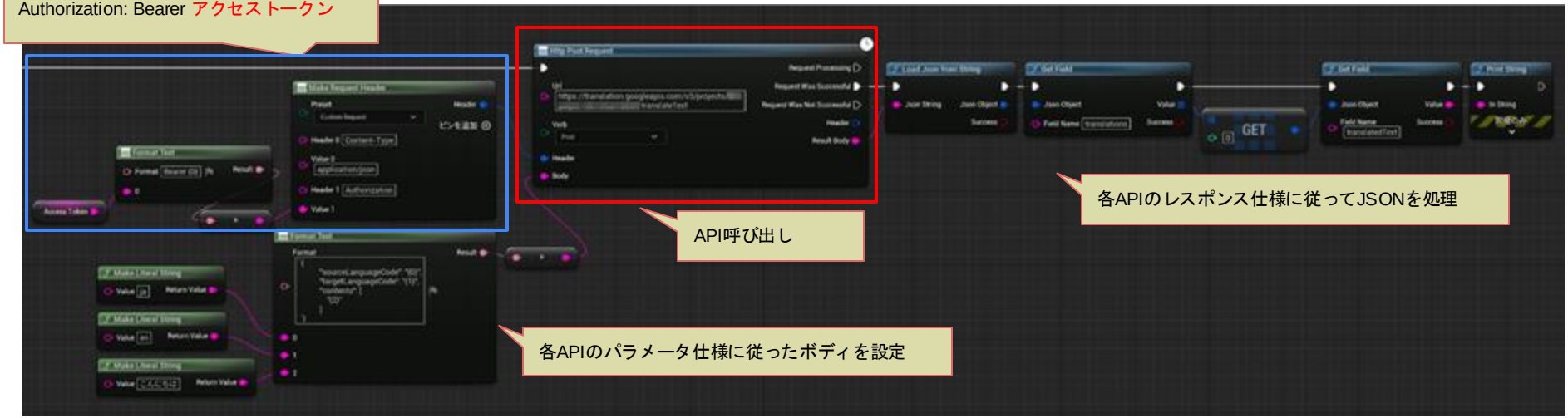
Key ID: サービスアカウントファイルの”private_key_id”
Issuer:サービスアカウントファイルの”client_email”
PrivateKeyString:サービスアカウントファイルの”private_key”

Scopeに関してはAPIの権限等を考慮して適切に設定してください。

APIにアクセス

Google Cloud Translation APIで日本語の「こんにちは」を英語にする処理を例にＡＰＩへのアクセス方法を解説します。
APIへのアクセスは、先ほど取得したアクセストークンをHTTPヘッダに組み込むだけです。
リクエスト・レスポンスの内容については、各ＡＰＩ仕様書の解説に従ってください。

Content-Type: application/json
Authorization: Bearer アクセストークン



APIにアクセスできました！

駆け足でしたが、OAuth 2.0 JWT Bearer Token Flowによる認可処理を紹介致しました。

一部C++を使用した部分はありますが、それほど難しくなくAPIサービスを呼び出すことができたかと思います。

UnrealEngineがゲームだけでなく、ゲーム以外の分野でも使われるにつれて、外部のＡＰＩとの連携は知っておいて損はない技術かと思います。

世の中には様々なＡＰＩサービスがありますので、アイディアと組み合わせ次第で面白いものが作れると思いますので、面白いものができましたら是非情報共有してください！

⚠️ 注意事項 ⚠️

本解説では、秘密鍵をアプリ側で保持した実装例を紹介しました。

本来、秘密鍵はその名の通り誰にも公開してはいけない内容になりますので、ビルドデータとは言え、配布用アプリに含める事はおススメ致しません。

また、一度リリースしてしまったアプリに含まれる秘密鍵を差し替えることは困難な為、鍵を一括管理できないという点も気にしなければなりません。

ここでは、技術的な事例の一つとして紹介させて頂きましたが、実際の運用で利用する際にはセキュリティ面についても十分に検討する必要がありますので、ご注意ください。

また、本資料でのロジックでは、アクセストークンに有効期限は考慮しておらず、有効期限内であればアクセストークを使いまわすような実装方法は紹介しておりません。こちらは、必要に応じて実装してみてください。

あと、APIは従量課金のものが多いので、予算アラート等を設定することをおススメします(この資料を作成する為の検証で2円かかっちゃいました！！いや、安いなっ！)

おまけ

事例のAI-OCRで使用していた TextureRenderTarget2DをAPIに渡す為にBase64に変換するロジックです

```
void ImageUtility::ConvertRenderTargetToBase64(UTextureRenderTarget2D* RenderTarget, FString& OutBase64String)
{
    if (!RenderTarget){ return; }

    FTextureRenderTargetResource* RenderTargetResource = RenderTarget->GameThread_GetRenderTargetResource();
    FReadSurfaceDataFlags ReadSurfaceDataFlags;
    ReadSurfaceDataFlags.SetLinearToGamma(false);
    ReadSurfaceDataFlags.SetOutputStencil(false);

    TArray<FColor> OutBMP;
    RenderTargetResource->ReadPixels(OutBMP, ReadSurfaceDataFlags);
    IImageWrapperModule& ImageWrapperModule = FModuleManager::LoadModuleChecked<IImageWrapperModule>(FName("ImageWrapper"));
    TSharedPtr<IImageWrapper> ImageWrapper = ImageWrapperModule.CreateImageWrapper(EImageFormat::PNG);

    int32 Width = RenderTarget->GetSurfaceWidth();
    int32 Height = RenderTarget->GetSurfaceHeight();

    if (ImageWrapper.IsValid() && ImageWrapper->SetRaw(OutBMP.GetData(), OutBMP.Num() * sizeof(FColor), Width, Height, ERGBFormat::BGRA, 8))
    {
        // PNGデータを取得する
        const TArray64<uint8>& PNGData = ImageWrapper->GetCompressed();

        // PNGデータをFDefaultAllocatorにコピーする
        TArray<uint8> PNGDataCopy;
        PNGDataCopy.Append(PNGData.GetData(), PNGData.Num());

        // Base64エンコード
        FString Base64String = FBase64::Encode(PNGDataCopy);
        OutBase64String = Base64String;
    }
}
```

ご清聴ありがとうございました！