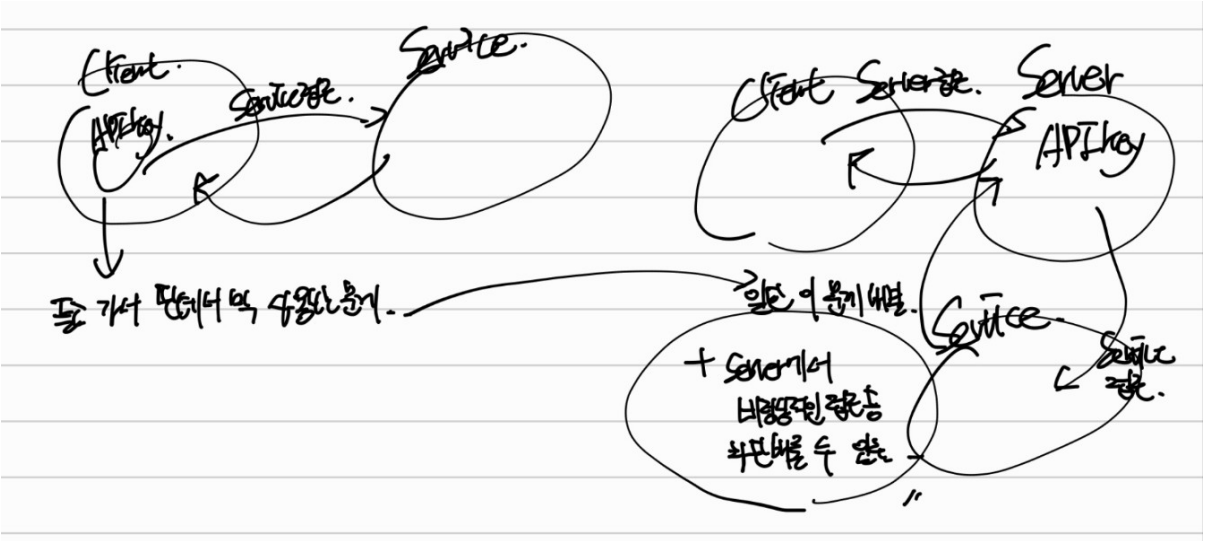
OpenAI. API Key 숨기기

API Key 를 Server 에 두어야 하는 이유



- API Key 가 Client 상에 들어있고, 이를 통해 직접적으로 Service 에 접근하는 경우 :
  - API Key 만 추출하거나, 빼내서 다른 코드에 넣어 무자비하게 호출하는 등의 문제 발생 가능. (상품 구매 API Key 를 막 500번 호출한다던가.)
- API Key 를 Server 상에 두는 경우 다음과 같은 이점을 얻을 수 있음.
  1. API Key 를 추출할 수 없으므로, 위와 같이 API Key 만 추출해서 다른 곳에서 사용하는 등의 문제를 해결하는 것이 가능.
  2. Server 에서 비정상적인 호출 등에 대한 처리를 마련함으로써, 일부 문제 또한 해결해 주는 것이 가능. (일부 비정상적인 함수 호출 등의 문제를 해결.)

약간 별개로, Level, 재화 등 주요 변수를 Local 에 두는 경우

- **따로 관리되어야 하는 문제. / 문제의 결이 약간 다름.**
- local 에서 해당 변수에 대응되는 메모리 주소를 찾아 이를 변조하였을 때 문제가 발생할 수 있음.
- Server 상에서 해당 값을 받아와 사용하는 경우, 이러한 문제를 막을 수 있음.

약간 별개로, 함수 자체를 변경하거나, 함수 자체를 여러번 호출할 수 있도록 코드를 수정하는 경우

- **따로 관리되어야 하는 문제. / 문제의 결이 약간 다름.**
- Code Tampering과 관련된 문제. 무결성을 검사하고, 이를 방지할 수 있는 기능은 Unity. Anti-Cheat Toolkit 등에 기록되어 있음.

OAuth 와 Access Token 학습

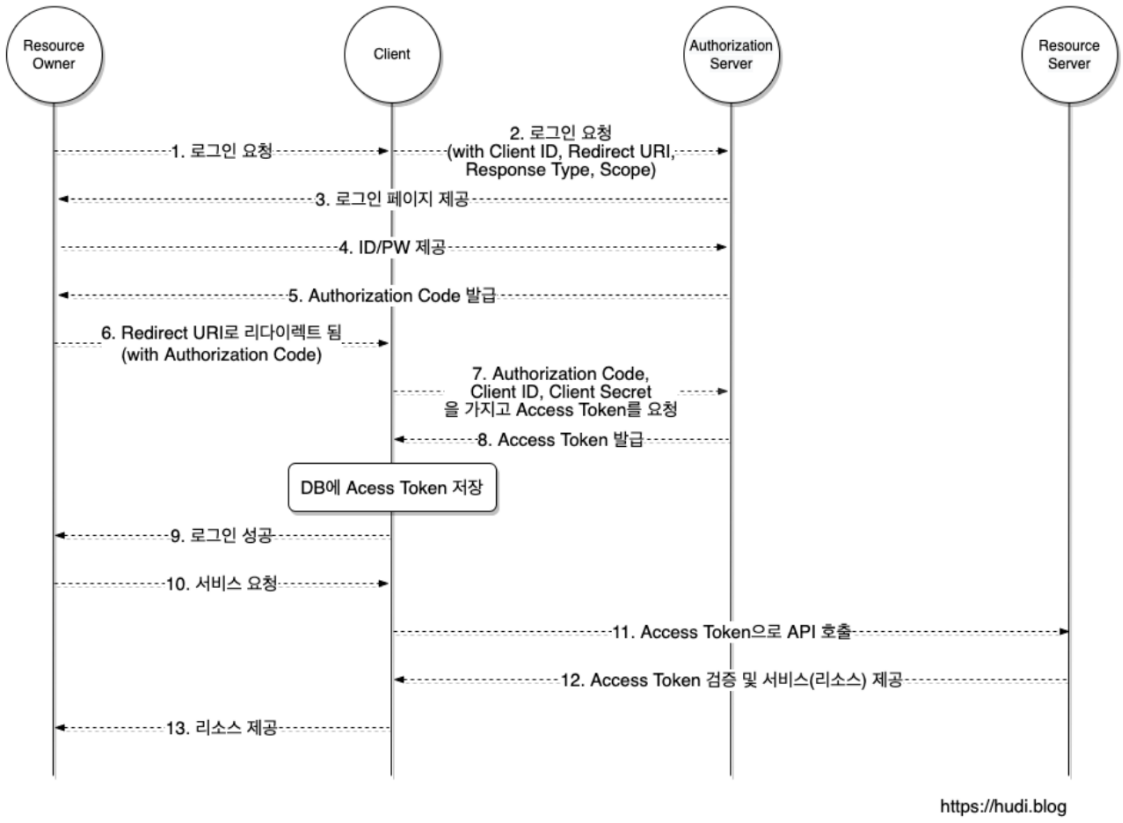
- 위 API Key 를 Server 에 두어야 하는 이유와 OAuth / Access Token 이 처음 관련되어 있는 줄 알았다. 근데, 지금 바로 탐색할 수 있는 지식 수준에선, 아예 다른 것이구나.
- OAuth 를 통해 얻은 Access Token 이 있어야, 해당 API 를 호출할 수 있게끔 한다면, 약간 관련이 있다고 생각할 수도 있긴 한데. 여기 OAuth 에서 말하는 Client 도, 결국 우리 입장에서는 Server 가 될 것 같아서...
  - Client 와 Server 는 상대적인 개념이다.
  - 내가 그냥 remote 에 존재하는 것을 Server 라 생각한데 반해, 그냥 이렇게 생각하면 된다. 요청하는 쪽이 Client. 요청 받아 그에 대한 처리 및 반응을 해주는 쪽이 Server.

OAuth / Access Token

- OAuth 2.0 개념과 동작원리 (hudi.blog) : 해당 내용 Reference 로 이용.
- **OAuth 란?** : Resource Owner (사용자) 의 다른 Platform (ex - 구글, Facebook 등) 정보 (Resource Server) 에 접근 할 수 있는 Authorization (권한) 을 Client (제 3의 클라이언트 = 우리 소프트웨어) 가 위임 받을 수 있도록 하는 Protocol 이다.
  - **24.01.11 추가 이해 기반 정의** : Client / Server 가 있다고 할 때. Server 가 유저 정보, 인증 (ID/PS 등 기반), 인증 완료 후 Server 내 해당 유저 정보 접근 허용 등을 관리한다 하자. 접근을 허용해준다 할 때, 어떻게 접근 허용을 관리할 것인가? Server 는 Client 의 인증 후, Client 가 Server 에 접근할 때 쉽게 할 수 있도록, Server 는 약식화된 내용인 Token 을 던져준다. 이러한 구조의 인증 / 접근 형식 (Protocol) 을 OAuth 라 한다.
    - 한 마디로 정리하면? : 인증 마치면 token 던져주는 거 다.
    - session / token 에 대한 직관적인 이해 : JWT 대충 쓰면 님들 코딩인생 끝남 (youtube.com)
- **용어**
  - Resource Owner : 리소스 소유자. 다른 Platform 에 계정 및, 정보 (ex - 카톡 친구 목록 등) 를 가지고 있는 사용자 이다.
  - Autorization Server : Resource Owner를 인증하고, Client에게 액세스 토큰을 발급해주는 서버.
  - Resource Server : 사용자의 Resource 를 가지고 있는 서버.
  - Client : Resource Server의 자원을 이용하고자 하는 서비스. 보통 우리가 개발하고자 하는 Service 가 여기에 해당된다.
  - Application 등록 관련 용어
    - Redirect URI : 인증이 성공한 사용자 (Autorization Server 가 접근자가 실제로 Resource Owner 인지를 확인하는 과정) 를 리디렉션 시킬 URI
      - 해당 URI 의 경우, Authorization Server 에 먼저 등록해두어야 함.
      - 등록을 완료하면, Client ID 와 Client Secret 을 얻을 수 있음.

        - Client ID, Client Secret : Client 가 Authorization Server 로 부터 Access Token 을 획득하는데 사용되는 식별자.
    - Access Token : OAuth 인증 과정을 통해 발급되는 코드로, Client 가 Resource Server 에 접근하여 정보를 가져올 수 있게끔, API를 호출할 수 있게끔 한다.
        - ex 1 - 또한 Unity Client 가 있고, UGS 처럼 Authentication 을 위한 다른 Server 가 존재하는 경우도 동일하다. UGS-Authentication Service (authorization server) 에서 Authentication을 하면, access token 을 받고, unity client 에서는 이를 관리하며, resource server (cloud save) 등에 접근할 수 있다.
            - 이런 일상적인 경우도 OAuth. (Client - Server 단일.)
        - ex 2 - 웹페이지가 있고, 여기서 KakaoTalk, Facebook 등을 이용한 로그인을 지원 및, 여기의 데이터를 활용한다고 하자. 이것도 OAuth 를 이용해서 인증 및 Resource 를 가져올 것이다.
            - 이런 일상적인 경우도 OAuth. (Client - Server(Platform) 단일.)
        - ex 3 - Unity Client 가 있고, 내가 Spring 이든, NodeJS 로든 만든 Server 가 있고, 여기에 추가적으로 아이디 / 비번 제 3 플랫폼과의 연결 등을 지원한다고 하자. 또한 해당 Spring Server 의 Service 는 제 3 플랫폼 (카카오톡) 등으로부터 OAuth 인증을 기반으로 받은 Acess Token 을 이용하여 Resource 에 접근한다.
            - 이런 A (Unity Client) - B (Spring Server) - C (Platform Server) 구조도 OAuth 를 이용하는 것이다. (Client (A) - Server (B) (이것도 OAuth 가능 - 인증, access token, 이를 기반으로한 Resource 접근을 지원하는 경우) / Client (B) - Server (C) (이건 무조건 OAuth))
        - 사실 OAuth 뿐만 아닌, 사용자 인증 후, Token 을 발급한 후, 이후 접속 시 해당 Token 을 확인하여 사용자를 인증하는 방식인 **토큰 기반 인증**에서, 발급되는 모든 토큰을 Access Token 이라 하면 될 듯!
            - 좀 더 복합적으로 가면, OAuth 가 Token 기반 인증 아래에 속한다. 클라이언트가 인증을 위해 사용자의 자격 증명(예: 로그인 정보)을 제공하면, 서버는 그 정보를 검증한 후에 클라이언트에게 토큰을 발급하는 방식 전체를 통칭하며, Token 기반인증의 공통점으로, 이 토큰은 클라이언트에 저장되고, 일반적으로 HTTP 요청의 헤더에 포함되어 서버로 전송된다는 점이 있다. (Header 중, Authorization 에 해당 Token 이 일반적으로 포함되어 전송됨.)
            - 더 자세한 내용은 [Bearer란? (tistory.com)]() 이 내용 참고.

- 위 용어를 기반으로 OAuth 2.0 의 동작 **메커니즘**을 확인해보자.



OpenAI API Key, Unity Client 에서 숨기기

> RageAgainstThePixel/com.openai.unity: A Non-Official OpenAI Rest Client for Unity (UPM) (github.com) : Unity를 사용 중이고, Unity 에서 OpenAI API 연결을 쉽게 하기 위해 해당 Library 를 사용할 것임.

- 해당 Library 에서 제공하는 API Key 를 Source 상에 직접적으로 넣지 않는 방법은 다음과 같이 두 가지가 존재하는 듯.
  - Azure OpenAI
  - OpenAI API Proxy
- Azure (애저) 는 확인했을 때, Microsoft Azure 의 내용을 직접적으로 이용해야 해서 선행되어야 할 지식이 너무 많다 판단.
- 아래, OpenAI API Proxy 를 좀 더 탐색하고자 하였음.

OpenAI API Proxy

- API Key 가 직접적으로 드러나는 문제를 막기위해, Front-end App (우리의 게임 콘텐츠) 를 대신해 OpenAI에 요청하는 중간(intermediate) API를 설정하는 것이 좋다.
- 이 Library 는 front-end, intermediary host 양 쪽 모두로 사용될 수 있다.

Front End Example

이 예제에서는 다음 **주요 과정**을 수행한다.

- OAuth provider 를 이용해서 사용자를 인증.
- 사용자를 인증한 후 (authenticated), custom auth token (access token) 과 backend 상에 있는 API key 를 교환한다.

이에 대한 **상세한 단계**는 다음과 같다.

1. OpenAI-DotNet or com.openai.unity packages 를 이용해 new project 를 setup.
2. 우리가 만든/사용하는 OAuth provider 를 통해 user 를 Authenticate 한다.
3. authentication 을 성공적으로 완료한 후, 새로운 `OpenAIAuthentication` object 를 만들고, `sess-` 접두사가 있는 custom token 을 전달한다.
4. 새로운 `OpenAISettings` object 를 생성하고, intermediate API 가 있는 domain 을 특정한다.
5. `auth` 와 `setting` object 를 새롭게 만든 `OpenAIClient` Constructor 에 포함하여 넘겨준다.

이에 대한 **코드 예시**는 다음과 같다.

```C#
var authToken = await LoginAsync();
var auth = new OpenAIAuthentication($"sess-{authToken}");
var settings = new OpenAISettings(domain: "api.your-custom-domain.com");
var api = new OpenAIClient(auth, settings);
```

이렇게 설정할 경우, `OpenAI-DotNet-Proxy` 가 사용된 Backend 와 안전하게 통신한 후, OpenAI API 에 요청을 전달하도록 할 수 있다.
OpenAI 의 API Key 및 민감한 정보가 Process 내 안전하게 유지된다.

Backend Example

이 예제에서는 다음 **주요 과정**을 수행한다.

- 새로운 ASP.NET Core web app 에서, `OpenAIProxyStartup` 를 Setup 하고, 사용하는 방법에 대한 증명 (demonstrate).
  - proxy server 는 authentication(인증) and forward requests to the OpenAI API (API 에 request 를 중간 전달) 함으로써, API Key 및 민감한 정보를 안전하게 유

지한다.

> Proxy server 란? : 클라이언트와 다른 네트워크 서비스 사이에서 중간 역할을 하는 서버. 클라이언트의 Reqeust 를 받아 해당 이를 대신해서 서비스 서버에 전달, 그에 대한 Response를 서비스 서버로부터 다시 클라이언트에 전달합니다.
>
> - 이점 :
> - 보안 및 익명성 : 사용자의 IP 주소를 숨김으로써 익명성을 보장.
> - 캐싱 기능 : 자주 요청되는 데이터를 로컬에 저장(캐시)함으로써, 같은 요청에 대해 더 빠르게 응답.
> - 성능 향상 : 프록시 서버를 사용하여 네트워크 요청을 관리함으로써, 전반적인 네트워크 성능을 최적화.
> - 우리의 경우에서는 API Key 등 민감한 정보를 Client 로부터 숨기기 위해 해당 Proxy Server 를 이용!

이에 대한 **상세한 단계**는 다음과 같다.

1. 새 ASP.NET Core minimal web API project 를 만든다.
2. OpenAI-DotNet nuget package 를 1번에서 새로 만든 project에 넣는다.
   - Powershell 이용 : `Install-Package OpenAI-DotNet-Proxy`
   - Manually editing .csproj : `<PackageReference Include="OpenAI-DotNet-Proxy" />` // 이거 약간 unity 에서 manipest 조정하는 것과 비슷한 듯.
3. `AbstractAuthenticationFilter` 를 상속받는 새 Class 만들기.
   1. 이후, `ValidateAuthentication` method 를 override 하기.
   - 이는 internal server 에 대응되는 "user session token 을 확인하는 용도로 사용되는 `IAuthenticationFilter`" 를 implement 하는 것임.
4. `Program.cs` 에서, `OpenAIProxyStartup.CreateDefaultHost` 를 호출하고, type argument `AuthenticationFilter` 를 전달하여, proxy web application 새로 만들기.
5. `OpenAIAuthentication` / `OpenAIClientSettings` 를 API keys, org id, Azure setting 과 함께, 만들기.

이에 대한 **코드 예시**는 다음과 같다.

C#

```
public partial class Program
{
    private class AuthenticationFilter :
AbstractAuthenticationFilter
    {
        public override void
ValidateAuthentication(IHeaderDictionary request)
        {
            // You will need to implement your own class to
properly test
            // custom issued tokens you've setup for your end
users.
            if
(!request.Authorization.ToString().Contains(userToken))
            {
                throw new AuthenticationException("User is not
authorized");
            }
        }
    }

    public static void Main(string[] args)
    {
        var auth = OpenAIAuthentication.LoadFromEnv();
        var settings = new OpenAIClientSettings(/* your custom
settings if using Azure OpenAI */);
        var openAIClient = new OpenAIClient(auth, settings);
        var proxy =
OpenAIProxyStartup.CreateDefaultHost<AuthenticationFilter>(args,
openAIClient);
        proxy.Run();
    }
}
```

프록시 서버를 설정하면, 사용자는 OpenAI API에 직접 요청하는 대신 프록시 서버에 요청할 수 있음. 프록시 서버는 인증을 처리하고 요청을 OpenAI API로 전달하여 API 키 및 기타

민감한 정보를 안전하게 보호.

> *Q. ASP.NET Core web app 에 대한 이해 부족 => A. ASP.NET Core web app 설명 참조.*

OpenAI API Proxy Server 제작 실습

> ASP.NET Core web app 이 뭔지는 알았다. 그럼, OpenAI. API Key 숨기기 의 Proxy Server 제작에서 나와 있는 추가 내용인, "새 ASP.NET Core minimal web API project 를 만들고(1)" / "OpenAI-DotNet nuget package 를 1번에서 새로 만든 project에 넣은 후(2)" / "활용하는(3)" 방법에 대해 알아보자.

1. 새 ASP.NET Core minimal web API project 만들기

- 위 링크에 나와있는 API Project 만들기 (Visual Studio) 버전을 이용.

1. Visual Studio 시작 > 새 프로젝트 > ASP.NET Core Web API 선택.
   - Razer Page 라고 예시있는데, 이거 체크하지 않을 수 있도록 할 것.
2. Additional Information >
   - Framework : .NET 8.0 (Long Term support)
   - Authentication type > None
   - Configure for HTTPS : Check
   - Enable Doker : UnCheck
   - Use Controller : UnCheck
   - Enable OpenAPI support : Check

> 이렇게 하면 새 프로젝트가 만들어 지긴 함.

2. Add NuGet Package

- OpenAI-DotNet nuget package 를 어떻게 추가할 수 있을까?

1. Powershell 이용 : `Install-Package OpenAI-DotNet-Proxy`
   - 설치 안 됨 :

```
Install-Package : 지정된 검색 조건 및 패키지 이름 'OpenAI-DotNet-Proxy'에 대해 일치하는 항목을 찾을 수 없습니다. 사용
가능한 모든 등록된 패키지 원본을 확인하려면 Get-PackageSource를 사용하세요.
위치 줄:1 문자:1
+ Install-Package OpenAI-DotNet-Proxy
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : ObjectNotFound: (Microsoft.Power....InstallPackage:InstallPackage) [Install-Package], Ex
   ception
    + FullyQualifiedErrorId : NoMatchFoundForCriteria,Microsoft.PowerShell.PackageManagement.Cmdlets.InstallPackage
```

   - 이유 추정 :

1. 아마 .NET 8.0 을 기준으로 하지 않아서 그럴 거다.
   => 실제로 7.6.1 혹은 그보다 하위 버전만 존재함.
   => dot net 6.0 을 다운로드 후 다시 시도.
2. 그냥 nuget 이 제대로 동작하지 않는다.
   => Powershell 에서 직접하는 것이 아니라, Tutorial: Create a minimal API with ASP.NET Core | Microsoft Learn 여기 나왔던 것을 따라줘야 했던 것일 수도?

> NuGet 이란? : 마이크로소프트의 .NET 프레임워크를 위한 패키지 관리자. 다른 프레임워크를 쉽게 설치할 수 있는 창고 역할 겸, 라이브러리에 대한 메니페스트(의존성 관리자) 역할 겸라고 보면 됨. (따로 약자는 아닌 듯.)

2. Tutorial: Create a minimal API with ASP.NET Core | Microsoft Learn 튜토리얼 내 NuGet packages 이용 방법 따라가기.
   - Tools > NuGet Package Manager > Package Manager Console.
   - Enter `Install-Package OpenAI-DotNet-Proxy`
- 바로 성공!

3. 활용하기

다음과 같은 **세부적인 과정**을 거쳐야 하나.
3. `AbstractAuthenticationFilter` 를 상속받는 새 Class 만들기.
1. 이후, `ValidateAuthentication` method 를 override 하기.
- 이는 internal server 에 대응되는 user session token 을 확인하는 용도로 사용되는 `IAuthenticationFilter` 를 implement 하는 것임.
4. `Program.cs` 에서, `OpenAIProxyStartup.CreateDefaultHost` 를 호출하고, type argument `AuthenticationFilter` 를 전달하여, proxy web application 새로 만들기.
5. `OpenAIAuthentication` / `OpenAIClientSettings` 를 API keys, org id, Azure setting 과 함께, 만들기.

이 코드를 Program.cs 에 바로 붙여넣기 시도!

C#

```
public partial class Program
{
    private class AuthenticationFilter :
AbstractAuthenticationFilter
    {
        public override void
ValidateAuthentication(IHeaderDictionary request)
        {
            // You will need to implement your own class to
properly test
            // custom issued tokens you've setup for your end
users.
            if
(!request.Authorization.ToString().Contains(userToken))
            {
                throw new AuthenticationException("User is not
authorized");
            }
        }
    }

    public static void Main(string[] args)
    {
        var auth = OpenAIAuthentication.LoadFromEnv();
        var settings = new OpenAIClientSettings(/* your custom
settings if using Azure OpenAI */);
        var openAIClient = new OpenAIClient(auth, settings);
        var proxy =
OpenAIProxyStartup.CreateDefaultHost<AuthenticationFilter>(args,
openAIClient);
        proxy.Run();
    }
}
```

- 위의 코드 바로 붙여 넣기 과정 중 문제 발생

- *P. You will need to implement your own class to properly test custom issued tokens you've setup for your end users. = custom token 이 유효한지 test 하는, 자체 logic 을 구현해야 함.*
    - 이게 위의 첫번째 class 인 AbstractAuthenticationFilter 를 상속 받는 AuthenticationFilter 의 `ValidateAuthentication(~)` 에서 구현되어야 함.
    - AuthenticationFilter class 는 `OpenAIProxyStartup.CreateDefaultHost<AuthenticationFilter>(args, openAIClient);` 에서와 같이, ProxyServer 처음 만들 때, Generic 으로 들어가며, 이후 해당 Proxy Server 가 인증 체크할 때 사용됨.

문제 해결을 위한 배경 지식

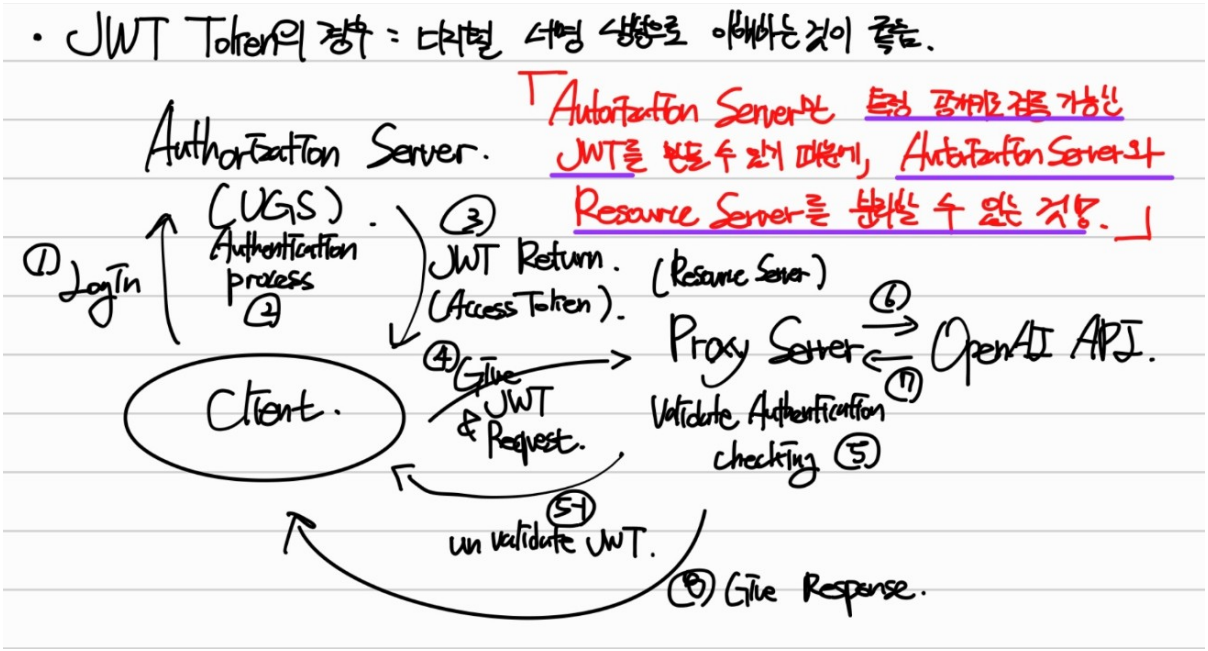
- 해당 ValidateAuthentication(~) 가 어떻게 token 의 유효성 유무를 체크할지는 OAuth Provider가 발급하는 토큰의 종류에 따라 다를 것.
    - 단적으로, 인증 발급 Server 와, 인증 체크 Server 가 동일하거나, 연결되어 있다면, 발급 Token 에 대한 DB 를 만들어 두고, 인증 체크 Server 에 들어온 Token 이 해당 DB 에 있는지 확인해보는 것도 방법임.
        - (조금 더 추가 생각 : 결국 인증 및 사용자에 관한 자세한 정보는 현재 사용자가 이용하는 응용 프로그램 그 바깥에 존재할 수밖에 없다는 생각이 기저에 깔려있음. 그 바깥은 Server임. 이렇게 되면 애초에, OpenAI. API Key 숨기기 > OAuth / Access Token 에서 배운 구조랑 일치함. (Resource Owner, Client, Server) 그럼 평소 우리가 사용하는 Application(Client) 가 있고, 외부에 Server 를 두고, 인증(계정/유저 확인, 유저에 따른 Resource 존재, 계정 확인에 따른 Resource 접근을 위한 Access Token 반환.)을 여기서 처리하는 순간, OAuth 와 동일하게 처리된다는 말. 그럼 Access Token 이 애초에 일상적으로 존재함.)
- Token 의 종류는 여러 가지가 존재함.
    - JWT : self-contained 방식으로 정보를 안전하게 전송할 수 있는 독립적 토큰 형식
        - JSON Web Token
    - SAML : 엔터프라이즈 환경에서 단일 로그인(Single Sign-On, SSO) 기능을 제공하기 위해 사용되는 Token
        - Security Assertion Markup Language
    - OAuth2 Access Token : 클라이언트 애플리케이션이 사용자를 대신하여 리소스 서버에 접근할 수 있도록 하는 목적.
        - 토큰 형식은 표준화되어 있지 않아, 구현에 따라 다름. Bearer 등.
- 가장 대표적인 JWT (JSON Web Token) 의 구조에 대해 알아보자.

    1. 헤더(Header): 토큰의 타입(일반적으로 JWT)과 사용된 서명 알고리즘(HMAC, RSA 등)을 표시. 헤더는 JSON 형식으로 작성되며 Base64Url로 인코딩됨.
    2. 페이로드(Payload): 토큰에 담길 클레임(claims)을 포함. 클레임은 사용자에 대한 속성이나 데이터(예: 사용자 ID, 권한 정보)를 지정하는 것을 의미. JSON 형식으로 작성되어 Base64Url로 인코딩됨.
    3. 서명(Signature): 헤더와 페이로드를 증명하는데 사용. 인코딩된 헤더와 인코딩된 페이로드를 결합, 비밀 키를 사용하여 알고리즘에 따라 해시(대체)하여 암호화 함. 이렇게 생성된 서명은, 검증시 복호화하여, 토큰의 무결성 (변경되지 않음) 을 탐색할 목적으로 사용됨.
- 가장 대표적인 예시인 JWT 의 유효성을 검사하는 방식은 다음과 같음.
    1. 토큰 형식 (헤더, 페이로드, 서명 구조 갖추고 있는지) / 만료일, 시간 검사 / 발급자 검사 등 기타 다양 검사.
    2. 서명 검증 : JWT 가 변조되지 않았음을 서명부를 확인함으로써 검증. 이 확인은 일반적으로 다음 과정을 거침.
        1. 인코딩된 헤더와 페이로드 추출: 서명을 검증하려는 JWT에서 인코딩된 헤더와 페이로드를 추출함.
        2. 서명 데이터 재생성: 받은 헤더와 페이로드로부터 서명 데이터를 재생성.
        3. 서명 비교: 서버에 저장된 공개 키(또는 비밀 키)를 사용하여 JWT의 서명을 해독. 이를 재생성된 서명 데이터와 비교.
        4. 검증 결과: 해독된 서명이 재생성된 서명 데이터와 일치하면, 토큰은 유효하며 변조되지 않은 것으로 간주. 일치하지 않으면, 토큰은 무효하거나 변조된 것으로 간주.
- 공개 키, 개인 키, 암호화와 복호화, 서명 생성 및 검증
- JWT 의 공개 키는 일반적으로 Json 형식인 JWK (JSON Web Key) 로 제공됨.
    - 정의 : JWK는 공개 키의 속성과 값을 JSON 형식으로 표현한 것.
    - key 의 유형 / key 의 용도 / 암호화 알고리즘 등에 대한 내용을 담고 있음.
    - JWK 자체는 JWT 와는 독립적인 개념이므로, 암호화 / 복호화 키 모두를 포괄.
    - 단, JWT의 디지털 서명을 검증하기 위해 제공되는 공개 키의 목록일 경우, 공개 키는 디지털 서명 검증을 위한 목록이므로, 복호화 키만 존재할 것.
- Bearer Token : Token 기반 인증에서 사용되는 Token 의 종류(type) 중 하나.
    - OAuth Framework 에서 사용되는 Token, JWT 를 통칭하는 종류라고 생각하면 된다. OAuth Framework 에서 사용하는 Token 을 통칭하기도 하기 때문에, Bearer 자체가 인증방식으로 말해지는 경우도 있긴 한데, Token 의 종류 중 하나로 국한해서 생각하는 편이 더 낫다고 생각한다.
    - RFC 6750 - The OAuth 2.0 Authorization Framework: Bearer Token Usage (ietf.org) 또한 그렇게 서술하고 있다.

문제해결 방법

- 위 배경 지식을 기반으로, custom token 이 유효한지 test하는 코드를 작성하기 위해선 두 가지 과정을 거쳐야 함.
    1. 내가 사용하는 OAuth Provider 가 어떤 형식의 Token 을 발급하는지.
    2. 해당 Token 형식을 기반으로 어떻게 유효성을 체크할지 방법을 정한 후, 검증 절차를 작성.

1. 내가 사용하는 OAuth Provider 가 어떤 형식의 Token 을 발급하는지.
    - OAuth Provider는 만약 사용한다면 UGS - Authentication 기능을 사용할 예정.
    - Authentication (unity.com) : UGS - Account 가 아닌, Cloud Code 에 나와있긴 하지만, 어쨌든, 이렇게 명시되어 있다. "Unity authentication uses Bearer authentication with JSON Web Tokens (JWT)"
    - *Q. 그럼 이 JWT Access Token 을 어디서 얻을 수 있는가? => A.* `AuthenticationService.Instance.AccessToken` *이라고 따로 있네!*
        - 해당 Access Token 이 JWT 의 형식을 따르는지 확인.
        - Base64 로 Encoding 되어 알아보긴 힘들었지만, **JWT 임을 확인!**
2. JWT 라면, 그리고 내가 짠 구조라면, 다음과 같이 동작함.



- 이거 정리 좀 기가 막히게 한 듯!
- 이에 따라, 유효성을 체크하는 방법은 다음과 같음.
    1. Server 에 들어온 JWT (Access Token) 의 형식을 확인.
    2. JWT 에 부합하는 경우, Header 와 Payload를 합친 것 / Header 에 나온 암호화 방식과, UGS 에서 제시된 공개키를 기반으로 복호화 (검증) 한 Signature 를 비교.
    3. 이후 같다면 return 으로 함수를 종료. 다르다면, Throw Exception.

2번 과정 자세

- 2번 과정의 경우, Proxy Server 의 코드로 직접 구현해야 하므로 좀 더 상세하게 작성.
- 해당 이론적 이해를 기반으로 GPT 에게 물어본 결과 다음과 같은 방법을 제시.
    1. Tokens 를 쉽게 분석하도록 해주는 NuGet Package 설치
        - `Install-Package Microsoft.IdentityModel.Tokens`
        - `Install-Package System.IdentityModel.Tokens.Jwt`
        - `Install-Package Newtonsoft.Json`
    2. 제시한 코드를 붙여넣기.
        - 위 패키지의 내용을 기반으로 JWT 를 검증. 이상 없으면 넘어감.
        - 자세한 내용까지 보기엔 너무 많이 봐야 할 듯.

> 해당 과정을 거쳐 Proxy Server 에 대응되는 친구들을 다 만들어 두긴 했다!
> 이제 이걸 어떻게 "배포할지"를 알아봐야 한다!

Access Token 을 검증하는 이유

- 생각해보니까, Access Token 을 검증하는 이유도 단순함! 만약 이를 검증하지 않고, 인증되지 않는 사용자 또한, 계속 받아들일 경우, server 정보만 따와서 다른 곳에서 계속 호출하는 것이 가능하잖나! 그래서 Authentication 을 사용하는 것이라고 생각하면 되겠네!

4. 배포하기

- 위 과정 및 Library 를 사용하여 Visual Studio 에 ASP.NET Core 를 이용한 web app 제작을 끝냈음. *Q. 빌드하여 이를 사용하거나, web app 을 server 상에 올리기 위해선, 무엇을 사용할 수 있나? => A. 웹 앱 및 웹 앱 배포에 대한 이해가 부족할 경우 ASP.NET Core web app 설명를 이용하기. 배포를 위한 클라우드 Service 의 경우, Azure App Service 를 이용하는 게 가장 간편할 것으로 보임. 이외에도 AWS 나 GCP 를 이용할 수 있음.*
- Azure App Service 가 대학생 대상으로 무료 Credit 을 제공하므로, 이를 최우선적으로 알아보고자 함.
    - 또한, 해당 코드가 Microsoft 의 ASP.NET Core 를 이용하고 있으므로, 연동성 측면에서 용이하리라는 생각도 깔려있음.
- 기본 스터디의 경우 ASP.NET Core web app 설명 > Microsoft Azure 를 활용한 배포 부분에 작성할 것!