ASP.NET Core web app 설명

기본 개념

1. .NET Framework : 마이크로소프트가 개발한 소프트웨어 개발 프레임워크, 윈도우 운영 체제에서 실행되는 애플리케이션을 개발하기 위해 사용되었음.
    - 공용 언어 런타임 (CLR) : .NET Framework 애플리케이션의 실행 환경으로, 다양한 프로그래밍 언어(C#, VB.NET, F# 등)로 작성된 코드를 실행할 수 있도록 함.
2. ASP.NET : "Active Server Pages .NET"의 약자. 웹 애플리케이션 및 웹 서비스를 구축하기 위한 기능을 제공하는 .NET Framework 의 구성 요소 중 하나임. ASP.NET은 동적 웹 페이지, 웹 API, 웹 폼 등을 생성할 수 있도록 지원.
    - 언어 독립적.
    - 웹 폼을 이용한 쉬운 웹 UI 제작 가능.
    - MVC 패턴을 기본 제공하여 웹 애플리케이션을 구축.
    - RESTful 애플리케이션 (Web API) 과 서비스를 쉽게 만들 수 있도록 지원.
    - 이외에도 기타 등등 지원!
3. ASP.NET Core : ASP.NET의 새로운 버전으로, 전체적인 재설계와 개선이 이루어진 프레임워크.
4. ASP.NET Core web app : ASP.NET Core 프레임워크를 사용하여 구축된 웹 애플리케이션.
5. 추가 용어 :
    - 웹 어플리케이션? (웹 앱(web app) 이란?) : 인터넷을 통해 접근 가능한 어플리케이션. Server 에서 Hosting 되며, Client 는 Web Browser 를 통해 상호작용할 수 있음.
        - Web Site 와의 차이점? : 웹사이트는 주로 정보를 제공하는 정적 콘텐츠 위주의 온라인 존재. 웹 애플리케이션은 사용자와의 상호작용 및 데이터 처리 기능을 갖춘 동적 서비스.
        - Web Service 와의 차이점? (Web Service & REST & Json) : 웹 서비스는 네트워크를 통해 다른 시스템과 상호작용하기 위한 방법을 제공. 최종 사용자가 웹 브라우저를 통해 상호작용하는 동적 웹사이트.
    - 웹 앱 (웹 어플리케이션) 을 배포한다는 것은? : 웹 앱은 인터넷을 통해 접근 가능한 어플리케이션을 뜻함. 인터넷을 통해 접근 가능하려면 해당 어플리케이션이 Server (Client 를 지원하기 위한 Computer System) 에 설치 및, 구성되어 있어야 함. 배포한다는 것은, 이 웹 어플리케이션을 Server 에 설치 및 구성하는 것을 의미함. 일반적으로 웹 앱을 배포하는 것은 다음과 같은 단계를 거침.
        1. 코드 준비 : 코드 최적화, 버그 수정, 보안 설정
        2. 호스팅 선택 : 호스팅할 서버, 클라우드 서비스를 선택

3. 서버 설정 : 데이터 베이스, 파일 저장소, 네트워크 등 설정
4. 코드 업로드 : 코드를 서버에 업로드
5. DB 마이그레이션 : 필요한 경우, 스키마 (데이터베이스 구조 - 쌍) 를 업데이트 하거나, 데이터를 마이그레이션
6. 종속성 설치 : 필요 라이브러리 및 프레임워크 설치.
7. 테스트 및 검증.
    - 일반적으로 4, 5, 6 번에 해당하는 항목을 "배포"라 함.

- 프레임워크? : 소프트웨어 개발을 위한 기반 구조를 제공하는 라이브러리나 도구 모음. 위의 ASP.NET Core web app의 경우, 일반적인 Library (기반 코드 뭉치) 뿐 아닌, CLR 등의 Tool 도 함께 제공하므로, 말 그대로 프레임워크라 부를 만함.

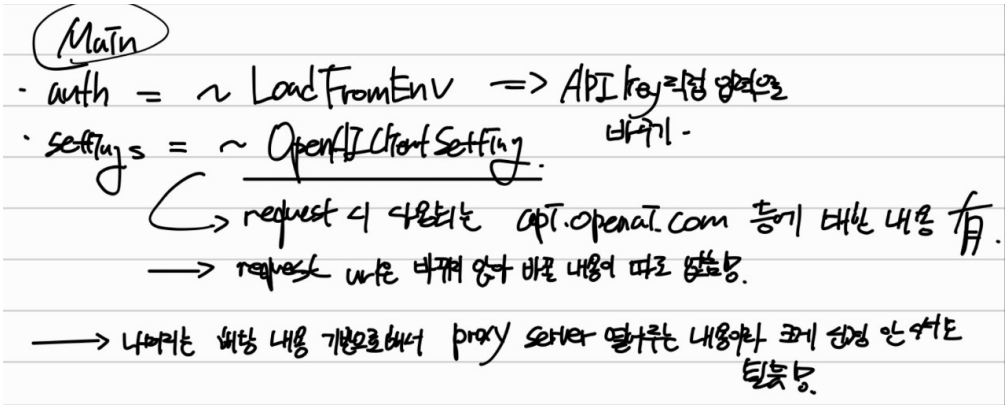
Microsoft Azure 를 활용한 web app 배포

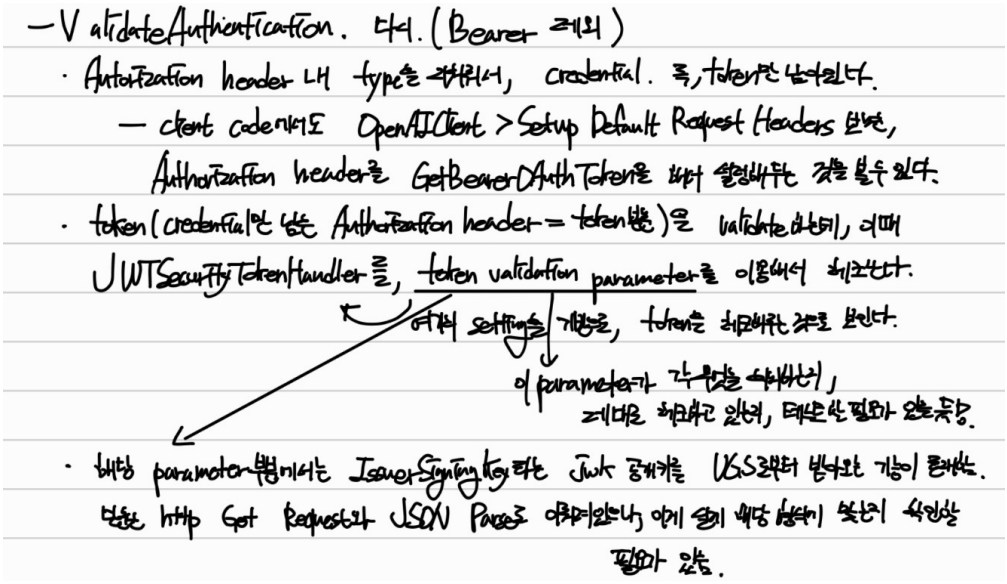
- Azure for Students - 나무위키 (namu.wiki) : 해당 혜택이 너무 좋아서, 해당 혜택 얻기.
    - 완료! 학교 계정 등 인증 시, 바로 사용 가능함!

Proxy Server 구성에서, 애매한 부분 체크 및 수정

> 한 번 쭉 보면서 **이해**한 후, 거기서 수정해야 하는 것을 확인하는 쪽이 낫겠다!

기존 코드 이해





- Main

- OpenAIAuthentication("~") : API Key, Organization ID 등에 대한 입력을 기반으로 OpenAI 에 보내는 인증 정보를 관리해주는 Class.
  - API Key 를 직접 입력하도록 바꾸었다.
- OpenAIClientSettings("~")
  - Request 시 사용되는 URL (api.openai.com / version = v1 등) 에 관한 정보.
  - 여기에 나와있는 URL 과 API Reference - OpenAI API 에 존재하는 URL 의 차이가 존재하지 않기 때문에, 그대로 이용하여도 될 듯!
- 나머지는 Proxy Server 열어주는 내용이라 신경 안 써도 됨!

- ValidateAuthentication ~

- `authToken` 관련 : Bearer란? (tistory.com) 을 보면 알 수 있음. Token 기반 인증을 사용할 때, Client 가 Server 로 Request 를 전송하는 경우, `Authorization: <type> <credentials>` 이와 같은 Header 형식을 사용함. type은 인증 종류, credential 은 token 의 본문에 해당.
  - 따라서 `requestHeaders["Authorization"].ToString().Replace("Bearer ", "");` 는, 해당 Header에서 token = credential 만을 추출하기 위해 Authorization Header 가 포함하는 type 에 관한 내용을 공백으로 치환하는 것.
  - Client Code (com.openai.unity package) 내 코드를 확인하였을 때도, OpenAIClient Class > SetupDefaultRequestHeaders() method 에, Authorization Header 로 `Rest.GetBearerOAuthToken(Authentication.Info.ApiKey)` 를 반환받아 처리하는 것을 확인하였음. access token 을 활용할 때, `Authentication.Info.ApiKey` 는 access token 이 들어감.
- token validate 시, `JWTSecurityTokenHandler` 를, `TokenValidationParameters` 를 이용해서 체크함.

    - 여기서 여러 셋팅을 해두면, 여기서 해둔 여러 셋팅을 기반으로 token 을 체크해주는 것으로 보임.
    - 해당 셋팅 한번 체크해서, *제대로 validate check가 이뤄질 지 확인해 줄 필요* 있음.
- `TokenValidationParameters` 부분에서 `GetIssuerSigningKey()` method 를 활용하여 IssuerSigningKey 에 할당함. 공개키 관련 부분인 것 같음.
    - 단순한 http get request 와 JSON Parse 로 이뤄져 있긴 하나 몇 가지 더 체크할 필요 있음.
        1. http get request url 이 맞는지.
        2. 가져오는 json string 의 형식과, 여기서 Parse 하고 있는 형식이 매칭되는지.

애매한 부분 확인 및 수정

- `TokenValidationParameters` : 각 parameter 부분의 뜻과, 잘 할당되어 있는지
    - TokenValidationParameters Class (Microsoft.IdentityModel.Tokens) - Microsoft Authentication Library for .NET | Microsoft Learn : 해당 내용 참고.
    1. ValidateIssuerSigningKey : token의 signature validation 할지 말지 bool.
    2. IssuerSigningKey : token의 signature validation 에 사용할 key. (string)
    3. ValidateIssuer : token validation 중, issuer를 검증 할지 말지 bool.
    4. ValidateAudience : token validation 중, audience를 검증 할지 말지 bool.
    5. ValidateLifetime : token validation 중, lifetime을 검증 할지 말지 bool.
    6. ClockSkew : 시간을 검증할 때, 사용할 clockskew 에 대한 setting.
        - 토큰 만료 연장 시간 의미 : ClockSkew :: ZepehWAVE (tistory.com)
    - issuer, audience, lifetime 은 각각 토큰의 발급자, 수신자, 만료시간 등을 의미함. 이는 JWT 의 Payload 부분에 claim 으로서 들어있음.
        - 클레임들의 검증은 토큰 발급에 사용된 규칙 및 정책에 따라 이루어지며, 서버 측에서 별도의 로직을 통해 검증하는 것이 일반적임.
        - ex - issuer : 서버에 신뢰할 수 있는 발급자 목록을 만들어 두고, 토큰의 issuer 부분과 비교함.
- 해당 내용을 기반으로 확인했을 때, `TokenValidationParameters` 는 합리적으로 설정되어 있는 듯!
- `GetIssuerSigningKey()`
    1. http get request url 이 맞는지 :
        - Unity Services Web API docs > GetJSONWebKeySet 에서 확인.
        - 'https://player-auth.services.api.unity.com/.well-known/jwks.json' 를 써야 함. 기존과 달라서 수정하였음.
    2. json string 의 형식과, 여기서 Parse 하고 있는 형식이 매칭되는 것을 확인.

3. 추가 고려 : `keys.FirstOrDefault()` 를 사용해서 하나의 key 만 가져오는데, 이 key 가 서명 검증 시 사용할 수 있는 공개키인지 모르겠음.
    - 이거 직접 request 해서 몇 개의 key 가 들어있는 지 확인하는 과정이 필요함.
    - => JWT의 서명을 검증하기 위해서는, JWT의 `kid` (Key ID) 클레임과 JWKS 응답 내 키들의 `kid` 값을 비교하여 일치하는 키를 찾아야 함. JWT Token 을 Parsing 해서 Header 에서 key id 를 찾은 후, JWK 에서 key id 를 기반으로 찾는 부분 제작.

```
// JWT 토큰 파싱
var jwtToken = tokenHandler.ReadJwtToken(token);

// JWT 헤더에서 'kid' 값을 추출
var kid = jwtToken.Header.Kid;

var validationParameters = new TokenValidationParameters
{
    ValidateIssuerSigningKey = true,
    IssuerSigningKey = await GetIssuerSigningKey(kid), //
```

해당 ASP.NET Core Web app 을 Local 에서 돌렸을 때, 잘 돌아가나.

- ASP.NET Core 웹 앱을 로컬에서 테스트하는 가장 간편한 방법은 Visual Studio의 내장된 개발 서버(Kestrel)를 사용하는 것.

사용법

굉장히 간편함!

1. Visual Studio 실행
2. ISS Express 버튼을 누르거나, F5 (디버그 시작) 을 누름!
3. 이후 기본 Web Browser 가 자동으로 열리고 앱의 홈페이지가 로드 됨.
4. 해당 Browser 에 표시된 Local Browser 를 사용하여 앱에 접근할 수 있음.

- 만약 해당 프로젝트에 할당된 포트 설정을 변경하고 싶은 경우, `appsettings.json` 파일 또는 `Properties/launchSettings.json` 파일에서 URL 및 포트를 변경할 수 있음.
- 만약 `NET::ERR_CERT_INVALID` 가 나타날 경우, Browser 의 아무곳이나 클릭한 후, `thisisunsafe` 를 치면 된다!

실제 테스트

- swagger 에 표시된 api 가 weather 관련 (기본적으로 제공되는 예시 api) 밖에 없어서 약간 우려.
- 신기한 점 : gitignore 하려고 했었는데, 더 상위의, 그 하위에 있는 directory 는 반영이 안 되더라. + 상위로 옮기니까 library 추적 제외 해둔 것 추적되어서, gitignore 를 두개

로 만들었더니 되더라.
- Unity Client 에 작성해둔 구조 및 코드가 별로여서 손 좀 보고 있는 중. (이거 2. 만들기 - 개발하기 절차 적용할 수 있도록 하기.)
    1.

GPT 에게 물어본 Azure 내 ASP.NET Core Web app 배포

Microsoft Azure에서 ASP.NET Core 웹 애플리케이션을 배포하는 과정은 여러 단계를 포함합니다. 아래는 Azure에서 웹 앱을 배포하기 위한 기본적인 단계들입니다:

1. Azure 구독 및 리소스 준비

- **Azure 구독**: Azure를 사용하기 위해서는 Azure 구독이 필요합니다. 이미 구독이 없다면, Azure 공식 웹사이트에서 가입할 수 있습니다.
- **리소스 그룹 생성**: Azure 포털에서 리소스 그룹을 생성합니다. 리소스 그룹은 관련 Azure 자원을 조직화하는 데 사용됩니다.

2. Azure App Service 만들기

- **App Service 만들기**: Azure 포털에서 "App Services"로 이동하여 새 App Service를 만듭니다. 이 서비스는 웹 애플리케이션을 호스팅하는 데 사용됩니다.
- **앱 구성**: 웹 앱의 이름, 리소스 그룹, App Service 플랜(호스팅 계획)을 선택합니다. App Service 플랜은 애플리케이션의 요구 사항에 따라 다양한 가격 및 성능 옵션을 제공합니다.

3. 애플리케이션 배포

- **Visual Studio에서 배포**: Visual Studio를 사용하여 애플리케이션을 직접 Azure App Service로 배포할 수 있습니다.
    - 솔루션 탐색기에서 웹 프로젝트를 마우스 오른쪽 버튼으로 클릭하고 "Publish"를 선택합니다.
    - "App Service"를 선택하고, 이전에 만든 App Service를 선택하거나 새로 만듭니다.
    - 설정을 구성하고 "Publish" 버튼을 클릭하여 배포합니다.
- **Azure DevOps 또는 GitHub Actions 사용**: 지속적인 통합 및 배포(CI/CD) 파이프라인을 설정하여 자동으로 배포할 수도 있습니다.

4. 데이터베이스 및 기타 서비스 구성

- 필요한 경우, Azure SQL Database나 다른 저장소 서비스를 설정하고, 애플리케이션과 연결합니다.

- 애플리케이션 설정(환경 변수, 연결 문자열 등)을 Azure App Service에서 구성합니다.

5. 모니터링 및 관리

- Azure App Service에서 제공하는 모니터링 도구(예: Application Insights)를 사용하여 애플리케이션의 성능을 모니터링합니다.
- 필요에 따라 확장성, 백업, 보안 설정을 조정합니다.

6. 도메인 및 SSL 설정 (선택적)

- 사용자 정의 도메인을 설정하고 SSL 인증서를 구성할 수 있습니다.

이 과정은 Azure의 사용자 인터페이스나 사용 가능한 서비스가 변경될 수 있으므로, 최신 Azure 문서나 지침을 참조하는 것이 좋습니다. Azure 포털의 지침을 따르고, 필요에 따라 고급 설정을 조정할 수 있습니다.