# WEB API

[WEB API 1](#_Toc25250)

[References 1](#_Toc30801)

[Concepts 2](#_Toc10223)

[API Questions 2](#_Toc26437)

[Getting started 2](#_Toc7711)

[Launchsettings.json 2](#_Toc15532)

[Appsettings.json 2](#_Toc29351)

[Secrets.json 3](#_Toc19200)

[Program.cs 3](#_Toc1415)

[Controllers 3](#_Toc13290)

[Http Verbs 3](#_Toc21704)

[FromBody attribute 4](#_Toc23630)

[Testing 4](#_Toc31951)

[API security 4](#_Toc21952)

[Versoning 4](#_Toc15489)

[Monitoring 4](#_Toc4269)

[Protecting 4](#_Toc9036)

[Building 4](#_Toc1209)

[Consuming 4](#_Toc131)

[Open API 4](#_Toc10531)

[Minimal API 4](#_Toc9917)

[Best practices 4](#_Toc9515)

### References

Tim videos: Tim\Web API

Code: DotnetDev\WebAPI\WebAPI

## Concepts

## API Questions

## Getting started

Project creation “ASP net core web API”

### Launchsettings.json

Helps us to launch the app with various profiles.

Profile with app name runs in kestrel(used for internal testing) server

Development or Production settings. Production will be selected by default.

Authentication settings

### Appsettings.json

Configuration settings. Used to avoid hard coding in c#.

Example: Logging information - loglevel, etc

Separate files will be present for production and development mode mentioned in appsettings.json file

Need of this file?

Text file, so we can change the information during production for debug. Connection string based on the stage of application release we can select the appropriate server.

### Secrets.json

Update app settings that are specific to particular machine. This will not be in source code location. So this data will not be committed to repo.

### Program.cs

Dependency injection of service. Scoped(one instance per user of API), transient(new instance every time) or singleton(one instance for entire instance of API) service

WebApplication supports many features(middleware) like UseAuthorization, UseHttpsRedirection, UseSwagger, Run(Actually starts the API service).

### Controllers

Define various endpoints. Various decorator like [ApiController] , Route

### Http Verbs

**Http Verbs: Put, Post, Get, Patch and Delete**

### FromBody attribute

From the body of the request

Builder.Services.AddEndPointsApiExplorer();

Builder.Services.AddSwaggerGen();

## Testing

Use Postman, swagger to verify the endpoints without client.

### Swagger/Open API

Provides information about all endpoints and also helps to initiate requests to all endpoints.

Add nuget package Swashbuckle.AspNetCore

## API security

Authentication provider: Identity 4.0, Auth 0, Azure active directory, B2C.

### Secrets

Right click on project and select “**Manage user secrets**” option to generate **secrets.json** file. Created in “C:\Users\balaj\AppData\Roaming\Microsoft\UserSecrets\7b0f6a97-a343-4b6a-924d-1ff07ec07cfd” location. Settings in secrets.json will override appsettings.json. Useful for local development. Its not required that all fields in Secrets.json should be present in appsettings.json. But its recommended to add so that we can understand the structure that is present in secrets.json.

### Tokens

Token will be provided by server to user at the end of successful authentication. Lasts for few minutes to few hours.

### Authentication and authorization

Authentication - Valid user name and password.

#### OAuth 2.0 for accessing google drive API

To implement authentication in a C# application, you can use the OAuth 2.0 framework, which is a widely used protocol for secure authorization. Here's a step-by-step guide to implementing OAuth 2.0 authentication in a C# application:

1. Register your application with the authentication provider (e.g., Google, Facebook, Microsoft, etc.) to obtain the client ID and client secret. This step will vary depending on the provider you choose.

2. Install the appropriate **NuGet package** for OAuth 2.0 in your C# project. For example, if you are using OAuth with Google, you can use the **`Google.Apis.Auth`** NuGet package.

3. Add the necessary **using statements** at the top of your C# file:

```csharp

using Google.Apis.Auth.OAuth2;

using Google.Apis.Auth.OAuth2.Flows;

using Google.Apis.Auth.OAuth2.Responses;

using Google.Apis.Services;

using Google.Apis.Drive.v3; // Replace this with the appropriate API you want to access.

```

4. **Implement the OAuth 2.0** authentication flow **in your code**:

```csharp

public class OAuth2Authentication

{

// Set these values with the client ID and client secret obtained during registration

private const string ClientId = "YOUR\_CLIENT\_ID";

private const string ClientSecret = "YOUR\_CLIENT\_SECRET";

public static UserCredential GetCredentials()

{

// Define the scopes of the APIs you want to access

string[] scopes = { DriveService.Scope.DriveReadonly }; // Replace this with the appropriate scope for your use case.

// Generate the OAuth 2.0 flow

var flow = new GoogleAuthorizationCodeFlow(new GoogleAuthorizationCodeFlow.Initializer

{

ClientSecrets = new ClientSecrets

{

ClientId = ClientId,

ClientSecret = ClientSecret

},

Scopes = scopes

});

// Start the OAuth 2.0 authorization process

UserCredential credential = GoogleWebAuthorizationBroker.AuthorizeAsync(

flow,

new[] { DriveService.Scope.DriveReadonly }, // Replace this with the appropriate scope for your use case.

"user", // User identifier

CancellationToken.None).Result;

return credential;

}

}

```

5. Use the obtained credentials to make API calls to the service:

```csharp

public class Program

{

public static void Main()

{

UserCredential credential = OAuth2Authentication.GetCredentials();

// Use the credential to make API calls

// For example, if using Google Drive API:

var service = new DriveService(new BaseClientService.Initializer()

{

HttpClientInitializer = credential,

ApplicationName = "YourAppName"

});

// Now you can use the 'service' variable to interact with the Drive API or other APIs based on the scopes you requested.

// For example, to list files from Google Drive:

var files = service.Files.List().Execute();

foreach (var file in files.Files)

{

Console.WriteLine(file.Name);

}

}

}

```

Note: This example demonstrates how to use OAuth 2.0 with the Google Drive API. If you're implementing OAuth 2.0 for a different provider or API, you may need to modify the scopes and API-specific code accordingly.

Make sure to handle exceptions and errors appropriately and store the credentials securely if required for future use.

Keep in mind that OAuth 2.0 is a complex topic, and the implementation details can vary based on the specific provider and API you are using. Always refer to the official documentation for the specific API you want to access to ensure the correct implementation.

#### OAuth 2.0 for user defined APIs

Implementing OAuth 2.0 for your own API involves setting up an Authorization Server to issue access tokens and protecting your API resources using those access tokens. Below is a simplified example of how you can implement OAuth 2.0 for your own API using the `IdentityServer4` library, which is a popular choice for building an Authorization Server in C#.

**Step 1: Set up the Authorization Server**

First, create a new ASP.NET Core Web Application and select the "Empty" template. Then, install the `IdentityServer4` NuGet package.

```bash

Install-Package IdentityServer4

```

**Step 2: Configure the Authorization Server**

In the `Startup.cs` file, configure the `IdentityServer` services and add a test user for demonstration purposes:

```csharp

using IdentityServer4.Models;

using IdentityServer4.Test;

public class Startup

{

public void **ConfigureServices**(IServiceCollection services)

{

// Configure IdentityServer

services.AddIdentityServer()

.AddInMemoryClients(GetClients())

.AddInMemoryApiResources(GetApiResources())

.AddTestUsers(GetUsers())

.AddDeveloperSigningCredential();

// Other service configurations...

}

private IEnumerable<Client> GetClients()

{

return new List<Client>

{

new Client

{

ClientId = "your-client-id",

ClientSecrets = { new Secret("your-client-secret".Sha256()) },

AllowedGrantTypes = GrantTypes.ClientCredentials,

AllowedScopes = { "your-api-scope" }

}

};

}

private IEnumerable<ApiResource> GetApiResources()

{

return new List<ApiResource>

{

new ApiResource("your-api-scope", "Your API Name")

};

}

private List<TestUser> GetUsers()

{

return new List<TestUser>

{

new TestUser

{

SubjectId = "1",

Username = "testuser",

Password = "testpassword"

}

};

}

// Other methods...

}

```

**Step 3: Protect your API with OAuth 2.0**

In your API project, you need to protect the resources using OAuth 2.0. This can be done using the **`Authorize` attribute**.

```csharp

[ApiController]

[Route("api/[controller]")]

public class SampleController : ControllerBase

{

[HttpGet]

**[Authorize]**

public IActionResult Get()

{

// Your API logic here...

return Ok("Hello from your protected API!");

}

}

```

**Step 4: Requesting an Access Token**

To access the protected API, clients need to request an access token from the Authorization Server. In this example, we'll use the **`HttpClient`** to simulate a client application making the request:

```csharp

using System.Net.Http;

using System.Net.Http.Headers;

public class Program

{

private static async Task Main()

{

var token = await GetAccessTokenAsync();

if (token != null)

{

await CallProtectedApiAsync(token);

}

}

private static async Task<string> GetAccessTokenAsync()

{

var client = new HttpClient();

var disco = await client.GetDiscoveryDocumentAsync("https://your-authorization-server-url");

if (disco.IsError)

{

Console.WriteLine(disco.Error);

return null;

}

var tokenResponse = await client.RequestClientCredentialsTokenAsync(new ClientCredentialsTokenRequest

{

Address = disco.TokenEndpoint,

ClientId = "your-client-id",

ClientSecret = "your-client-secret",

Scope = "your-api-scope"

});

if (tokenResponse.IsError)

{

Console.WriteLine(tokenResponse.Error);

return null;

}

return tokenResponse.AccessToken;

}

private static async Task CallProtectedApiAsync(string token)

{

var client = new HttpClient();

client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", token);

var response = await client.GetAsync("https://your-protected-api-url");

if (response.IsSuccessStatusCode)

{

var content = await response.Content.ReadAsStringAsync();

Console.WriteLine("API Response: " + content);

}

else

{

Console.WriteLine("Failed to call API. Status code: " + response.StatusCode);

}

}

}

```

Replace the placeholders (e.g., "your-authorization-server-url", "your-client-id", "your-client-secret", "your-api-scope", "https://your-protected-api-url") with the appropriate values.

Remember that this is a simplified example for demonstration purposes. In a real-world scenario, you need to handle various aspects like token expiration, token storage, and token validation more securely and efficiently.

### Claims

### Advanced Authorization

## Versoning

## Monitoring

## Protecting

## Building

## Consuming

## Open API

## Minimal API

## Best practices

## Misc and Questions

### CORS??????

### Redis Output caching

Reference youtube video: <https://www.youtube.com/watch?v=_bg5dGnudPs>

### Sample Task