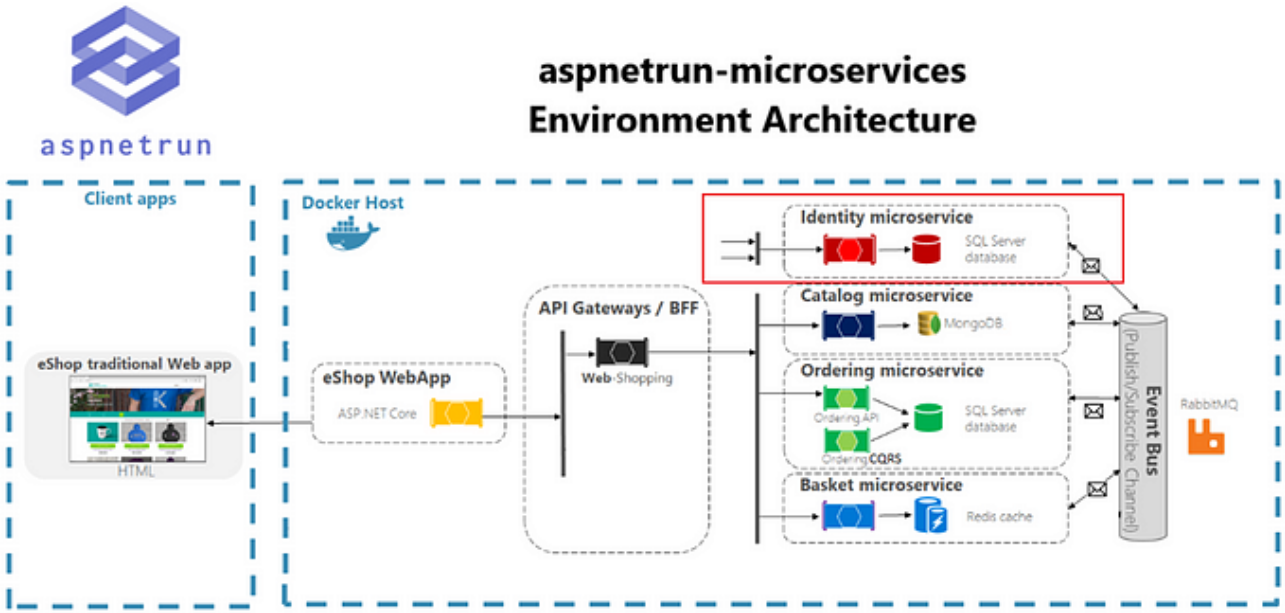
Securing Microservices with IdentityServer4, OAuth2 and OpenID Connect fronted by Ocelot API Gateway



In this article, we're going to learn how to secure microservices with using standalone **Identity Server 4** and backing with **Ocelot API Gateway**. We're going to protect our ASP.NET Web MVC and API applications with using **OAuth 2** and **OpenID Connect** in IdentityServer4. Securing your web application and API with tokens, working with claims, authentication and authorization middlewares and applying policies, and so on.

Step by Step Development w/ Course

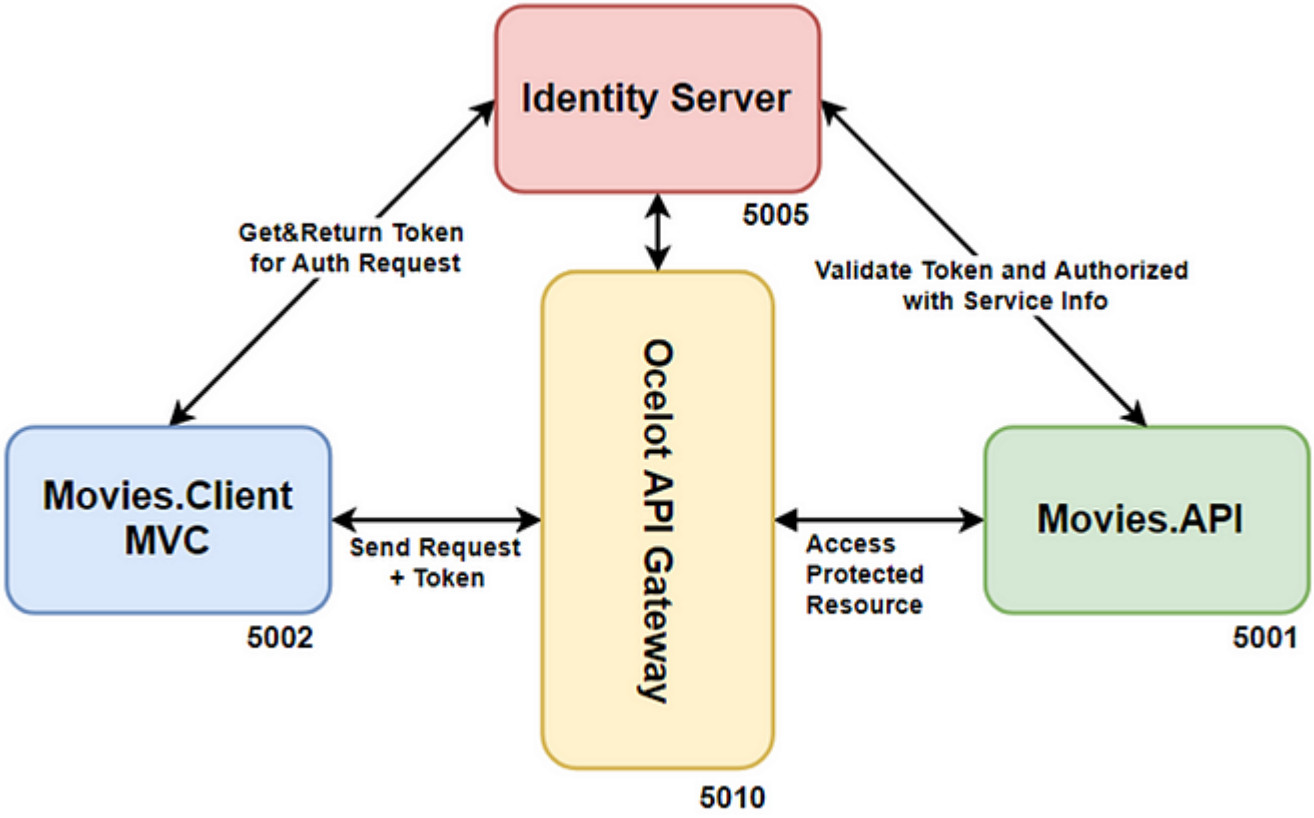
I have just published a new course — Securing .NET 5 Microservices with IdentityServer4 with OAuth2, OpenID Connect and Ocelot Api Gateway.
In the course, we are securing **.Net 5 microservices** with using standalone **Identity Server 4** and backing with **Ocelot API Gateway.** We're going to protect our ASP.NET Web MVC and API applications with using **OAuth 2** and **OpenID Connect** in **IdentityServer4.**

Source Code

Get the Source Code from AspnetRun Microservices Github — Clone or fork this repository, if you like don't forget the star. If you find or ask anything you can directly open issue on repository.

Overall Picture

See the overall picture. You can see that we will have 4 microservices which we are going to develop.



Movies.API

First of all, we are going to develop Movies.API project and protect this API resources with IdentityServer4 **OAuth 2.0** implementation. Generate JWT Token with client_credentials from IdentityServer4 and will use this token for securing **Movies.API protected resources**.

Movies.MVC

After that, we are going to develop Movies.MVC Asp.Net project for Interactive Client of our application. This Interactive Movies.MVC Client application will be secured with OpenID Connect in IdentityServer4. Our client application pass credentials with logging to an Identity Server and receive back a **JSON Web Token (JWT)**.

Identity Server

Also, we are going to develop centralized standalone Authentication Server and Identity Provider with implementing **IdentityServer4 package** and the name of microservice is Identity Server.
Identity Server4 is an open source framework which implements **OpenId Connect** and **OAuth2** protocols for .Net Core.
With Identity Server, we can provide authentication and access control for our web applications or Web APIs from a single point between applications or on a user basis.

Ocelot API Gateway

Lastly, we are going to develop Ocelot API Gateway and make secure protected API resources over the **Ocelot API Gateway** with transferring JWT web tokens.
Once the client has a bearer token it will call the API endpoint which is fronted by Ocelot. Ocelot is working as a reverse proxy.

After Ocelot **reroutes the request** to the internal API, it will present the token to Identity Server in the authorization pipeline. If the client is authorized the request will be processed and a list of movies will be sent back to the client.

Also over these picture, we have also apply the claim based authentications.

Background

You can follow the previous article which explains overall microservice architecture of this example.

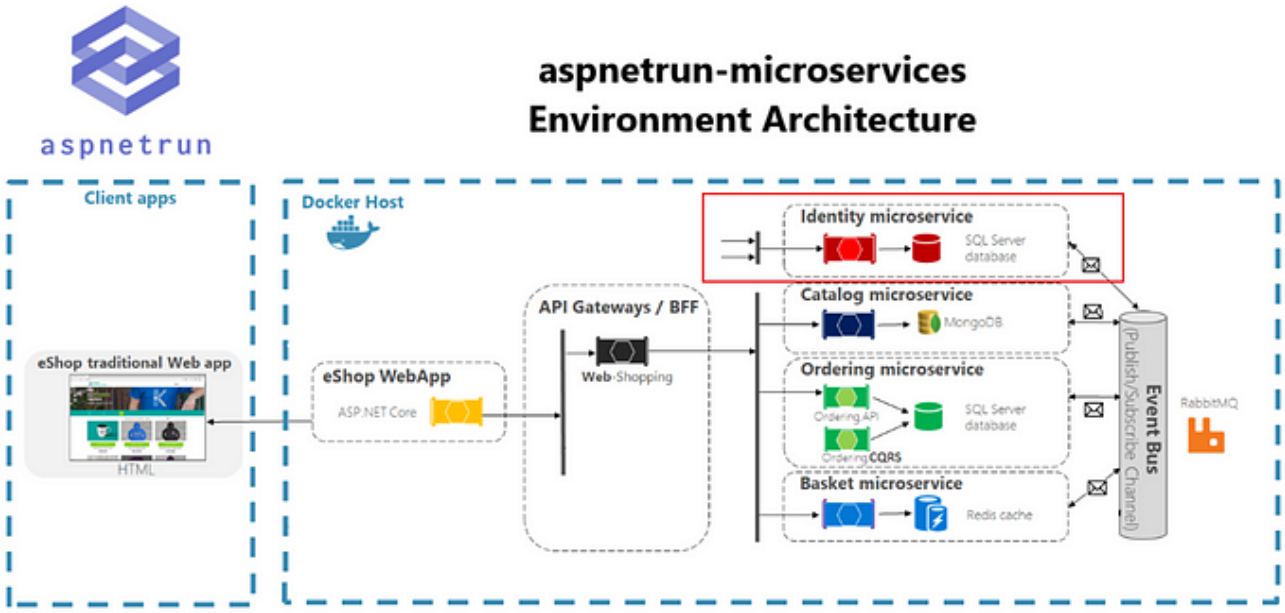
Check for the previous article which explained overall microservice architecture of this repository.

Prerequisites

- Install the .NET 5 or above SDK
- Install Visual Studio 2019 v16.x or above
- Install Postman

Introduction

We will develop security operations with **Identity Server integration** for an existing microservices reference application.
We had developed **run-aspnetcore-microservices reference application** before this article. You can see the overall picture of the reference microservices application.

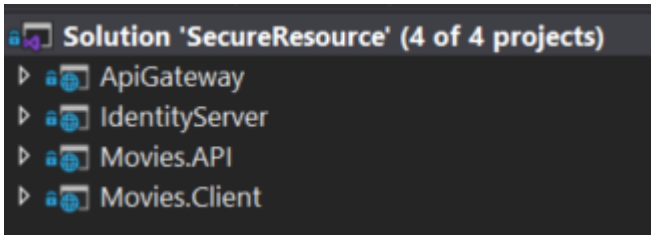


Now with the newly added red box in picture, we will extend this application with **IdentityServer OAuth 2.0 and OpenId Connect** features.

By the end of this article, you'll learn all about **how to secure your ASP.NET based microservices applications** with **IdentityServer4** using **OAuth 2 and OpenID Connect**. And Also you'll learn how to secure protected APIs backing with Ocelot API Gateway in a microservices architecture.

Code Structure

Let's check our project code structure on the visual studio solution explorer window. You can see the 4 asp.net core microservices project which we had see on the overall picture.



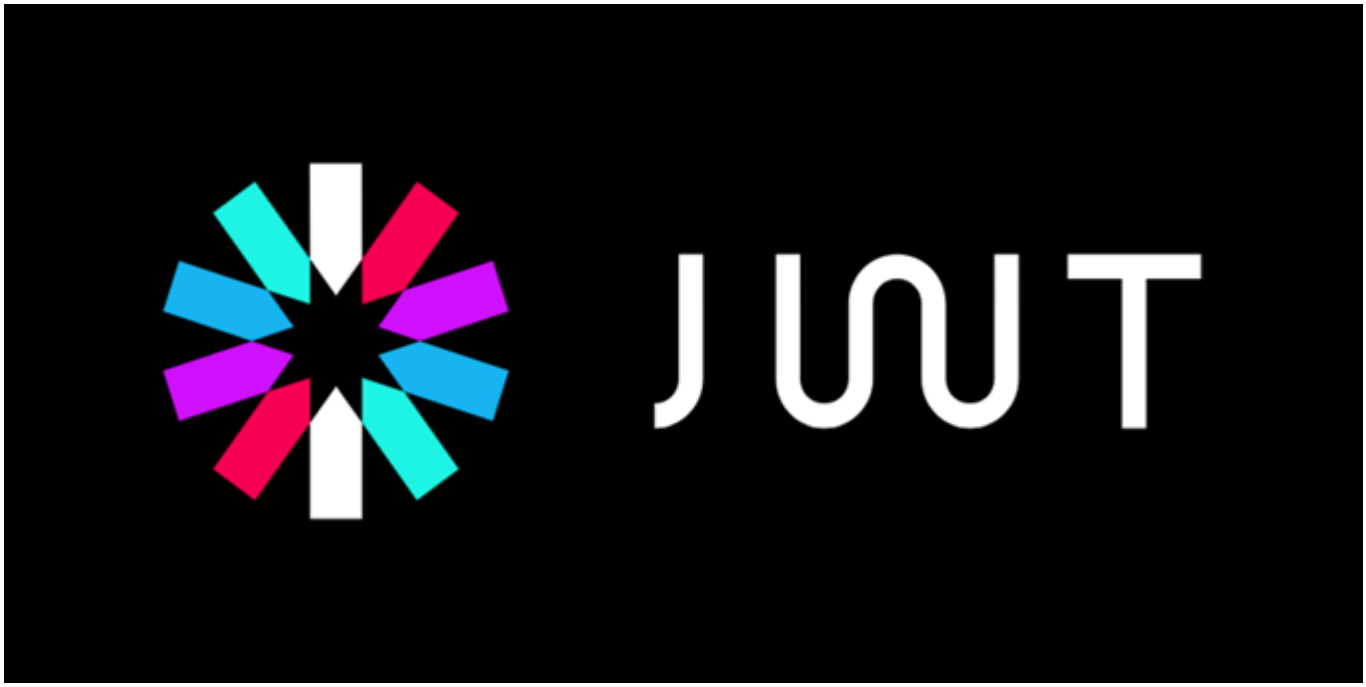
If we expand the projects, you will see that;
Movies.API is an asp.net core web api project which includes crud api operations.
Movies.MVC is an asp.net core MVC web application project which consumes **Movies.API** project and represent the data.
IdentityServer is a standalone Identity Provider for our architecture.
APIGateway is an api gateway between Movies.MVC and Movies.API.

Before we start we should learn the basics of terminology.

What is JWT (Token) ?

User authentication and authorization is very important while developing our web projects. We use various methods to protect our application from unauthorized persons and we provide access to it only for authorized users. One of these solutions is to use tokens.

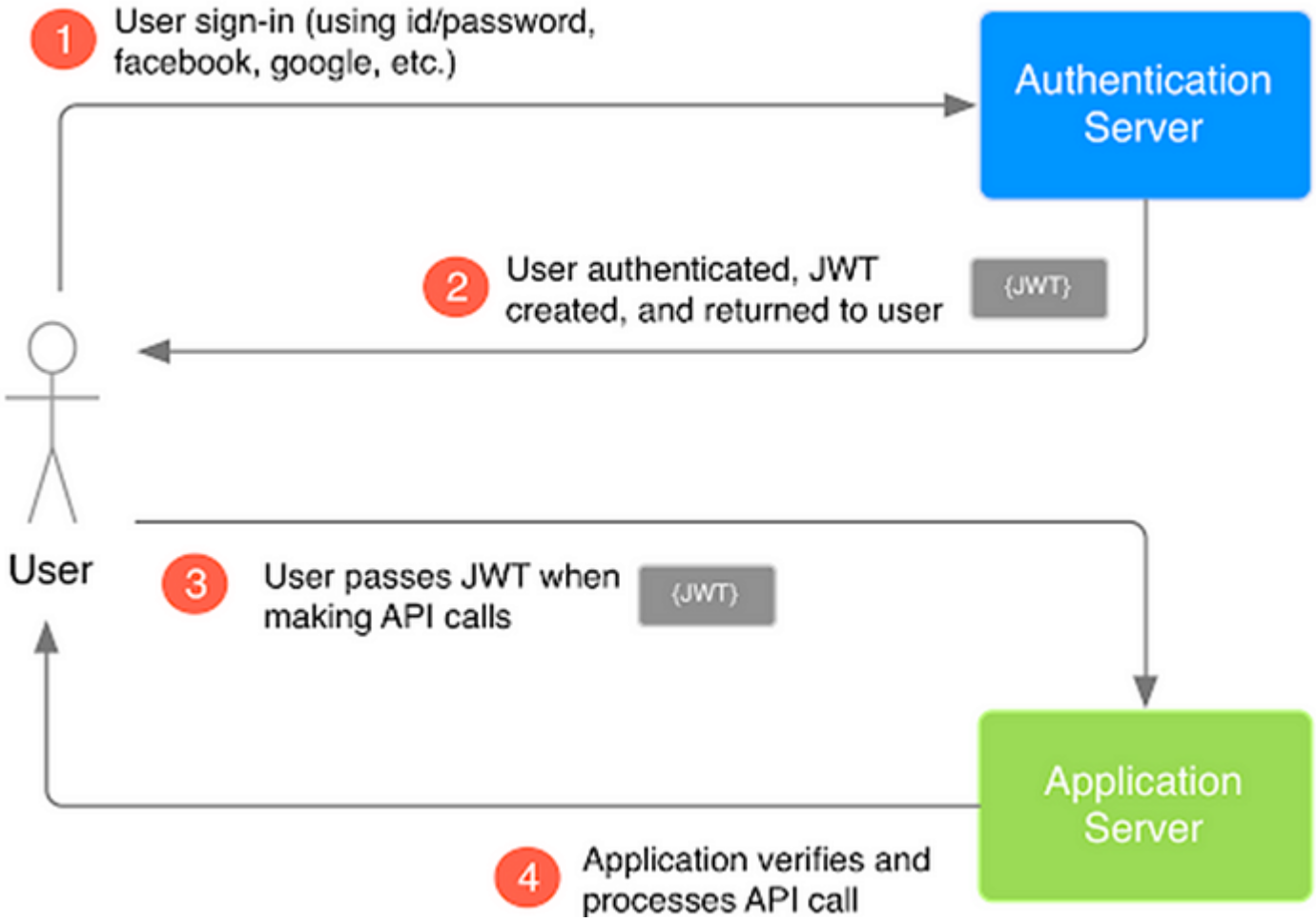
At this point, there are some **industry standards**. We are going to talk about **JSON Web Token (JWT)**.



JWT (JSON Web Tokens) is an **RFC7519** industry standard. It is used for user authentication and authorization on systems that communicate with each other. JWT can be used for many issues such as user authentication, web service security, and information security. JWT is a very popular and preferred method when protecting our resources. If we want to access a protected resource, the first thing we have to do is to **retrieve a token.**

JWT Example Scenario

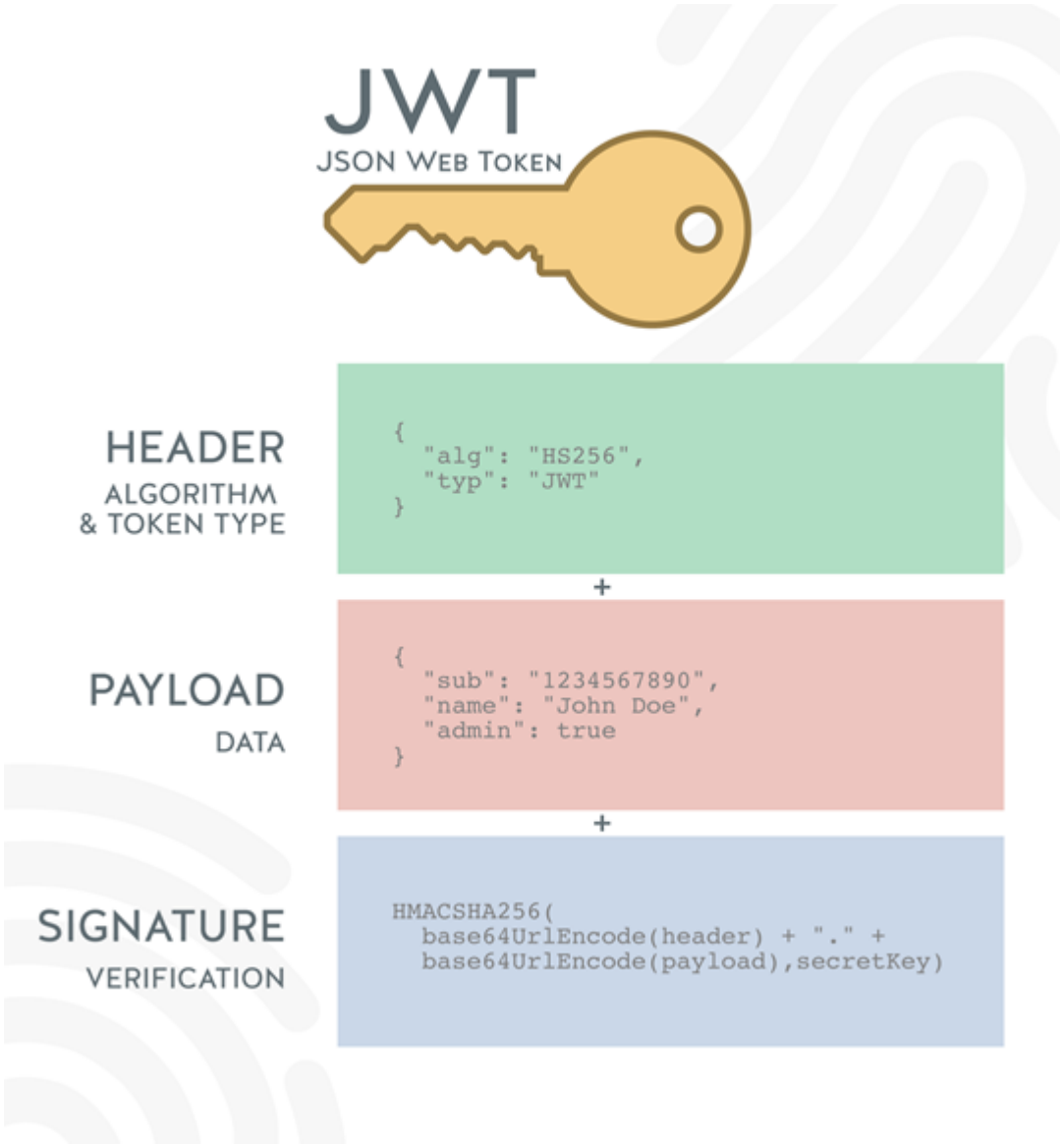
The general scenario is:



The user sends a login request to the server with username and password.
The Authentication Server takes this information and queries it from the database, combines the user name with its secret key and generate the token.
When user authenticated, Authentication server returns **token** and the server writes this information to the related user's Token column in the database.
The user passes **JWT token** when making API calls into header part of the request and the server side checks each time whether the token is valid or not from the database.
Application verifies and processes API call.

JWT(JSON Web Tokens) Structure

The token produced with JWT consists of 3 main parts encoded with Base64. These are **Header** (Header), **Payload** (Data), **Signature** parts. Let's take a closer look at these parts. If we pay attention to the token example on the side, there are 3 fields separated by dots in the token form.



Header

This part to be used in JWT is written in JSON format and consists of 2 fields. These are the token type and the name of the algorithm to be used for signing.

Payload (Data)

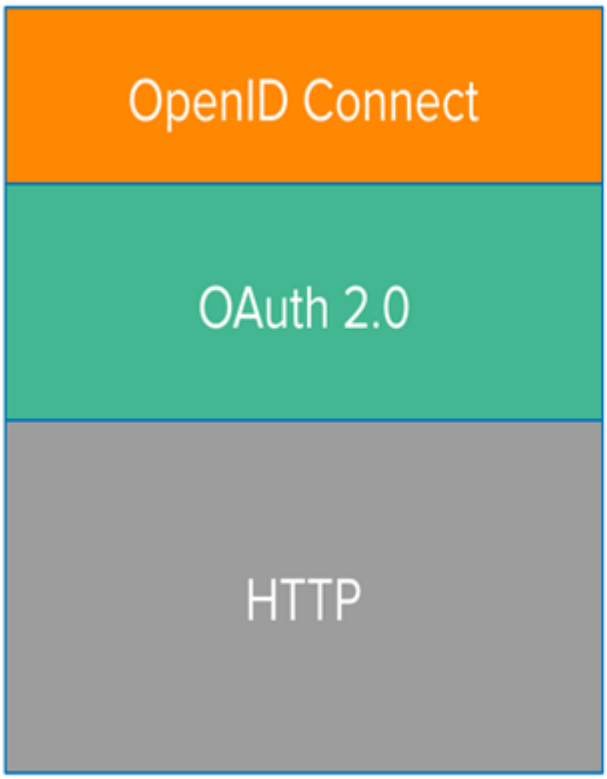
This section includes claims. The data kept in this section is unique between the token client and server. This captured claim information also provides this uniqueness.

Signature

This part is the last part of the token. Header, payload and secret key are required to create this part. With the signature part, data integrity is guaranteed.

What is OAuth2 ?

OAuth2 is an open authorization protocol used in data communication between applications. Lets think that we have developed an authorized application which includes **OAuth2** support. So once we authorized the user group then we can give access to the data source of another application with the **OAuth2 protocol.**



OpenID Connect is for **authentication**

OAuth 2.0 is for **authorization**

For example, when a user connects or logs in to the web application you are developing with facebook, google or github. OAuth2 is a protocol used for user authorization, not for authentication. Applications such as Google, Facebook, Twitter provide **OAuth2** support and verified users in these applications can be authorized to other developed web applications that use and rely on the Oauth2 protocol.

OAuth2 is the **industry-standard** protocol for authorization. It delegates user authentication to the service that hosts the user's account and authorizes third-party applications to access that account. In short, OAuth2 performs the authorization process between applications.

There is industry standard documentation regarding **OAuth 2.0 — RFC 6749** that helps us a lot to understand OAuth related topics.

OAuth2 Authorization Types and Flows

OAuth2 Grant Types or authorization flows determine the interaction between a client application and token service. We have said that the OAuth2 protocol is a protocol that does not provide user authentication but has the necessary support for it.

OAuth2 authorization types:
Authorization Code Grant
Implicit Grant
Resource Owner Password Credential Grant
Client Credential Grant

We can see roles included in OAuth2 flows with the image here;

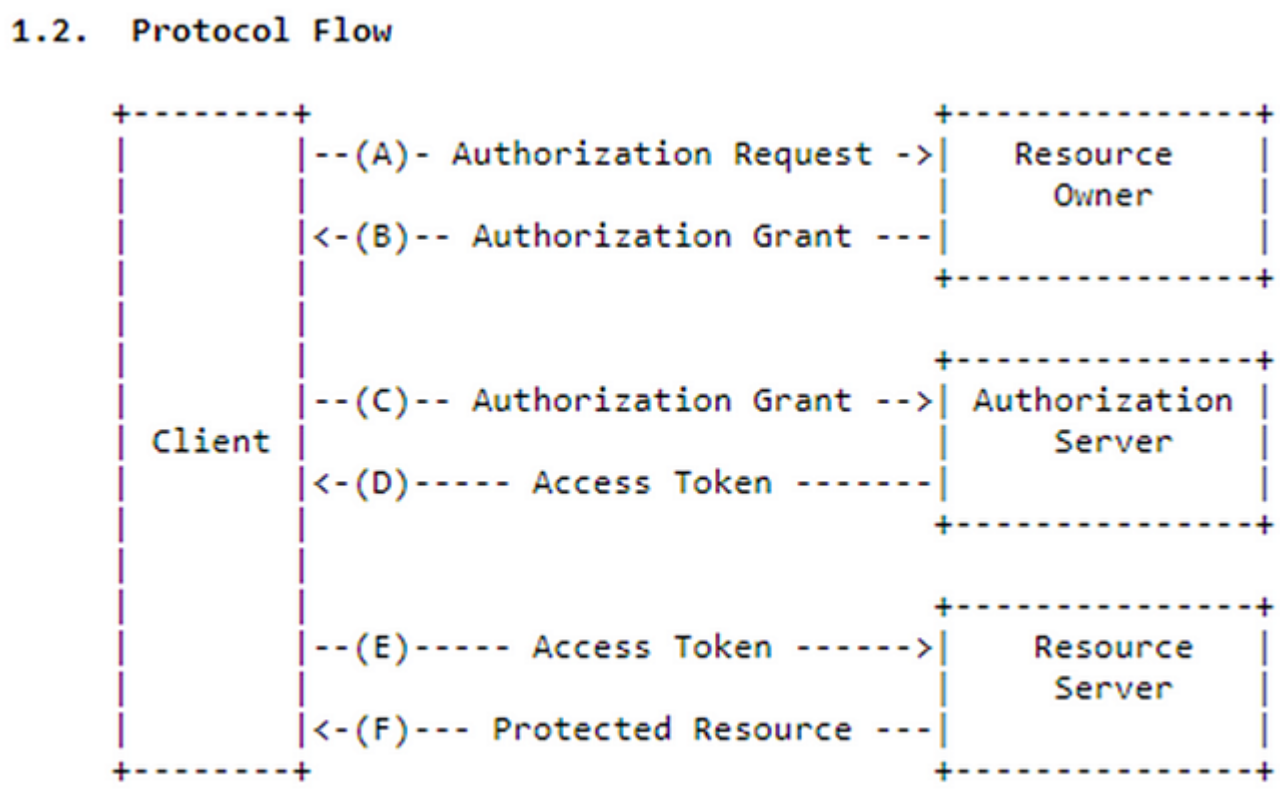


Figure 1: Abstract Protocol Flow

Client Application: A client application is an application that accesses protected data sources on behalf of the resource owner. (Web, Javascript, mobile…)
Client applications are classified as public and confidential. Public client applications; native applications (applications developed for desktop, tablet, mobile, etc.), user-agent based applications (Javascript based or SPA applications).

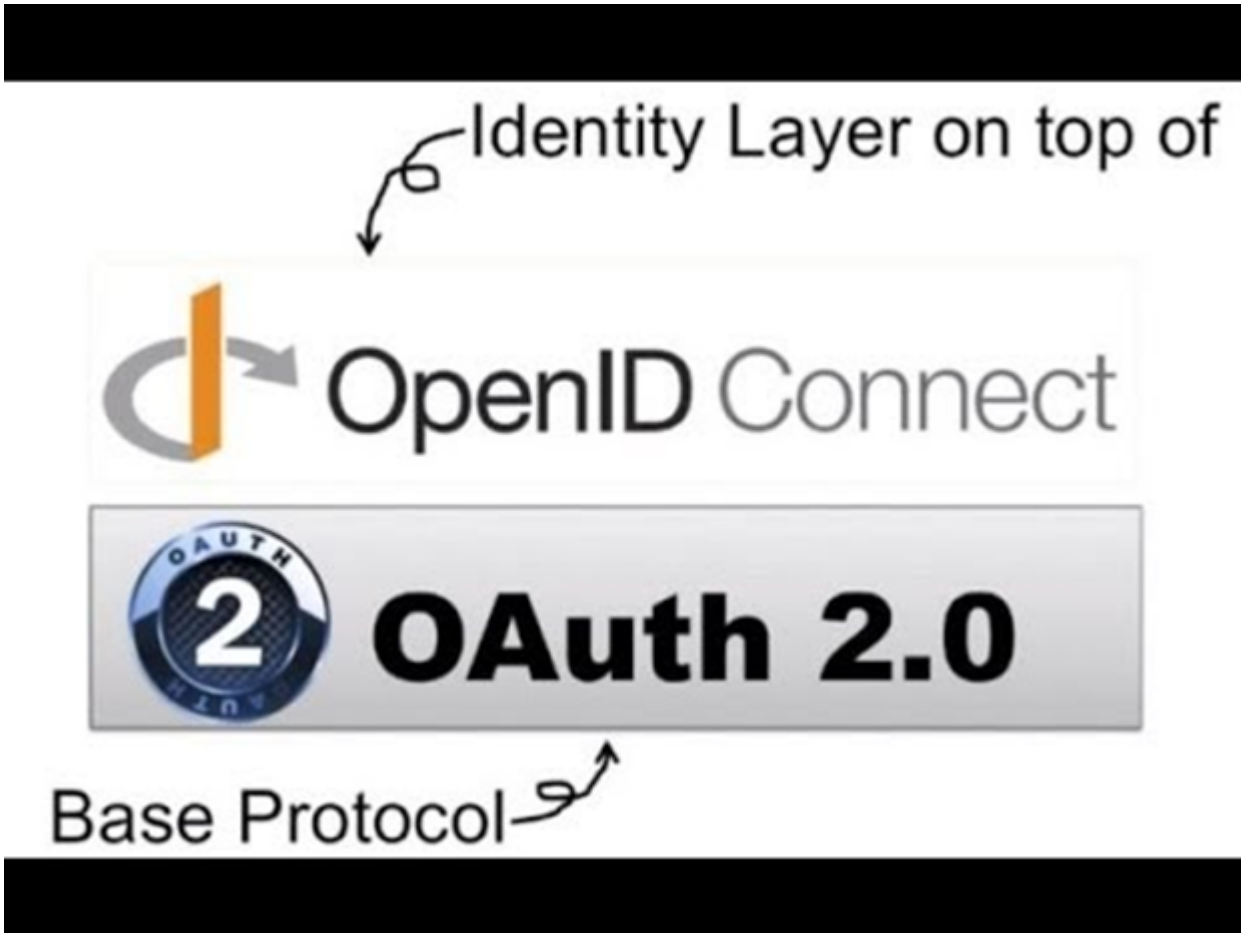
Resource Owner: The person (end user) or application that owns the data in the system.

Authorization Server: Gives access tokens within the resource owner authority for authenticated client applications. In short, it manages authorization and access.

Resource Server: Allows access according to the access token and authorization scope by managing the data source that needs to be protected. The protected data source can be eg profile information or personal information of the person.

What is OpenId Connect ?

OpenId 1.0 is a simple authentication layer built on the **OAuth2** protocol in fact OpenID Connect is an extension on top of **OAuth 2.0**. In addition to the end user authentication by an Authorization Server for client applications, it enables the end user to obtain simple profile information with a structure similar to REST. **OpenId Connect (OIDC)** provides an API friendly structure compared to previous versions. OIDC is an extension that extends the OAuth2 Authorization Code Grant structure.



OIDC keeps transactions simple, built on **OAuth2**, and simplifies token (JWT) usage. It can be integrated with OIDC web applications, mobile applications, Single page applications (SPA) and server side applications.

OIDC is the industry-standard protocol for authentication.
There is great documentation regarding OIDC — Core as below link that helps us a lot to understand OIDC related topics.

https://openid.net/specs/openid-connect-core-1_0.html

OpenID Connect Endoints

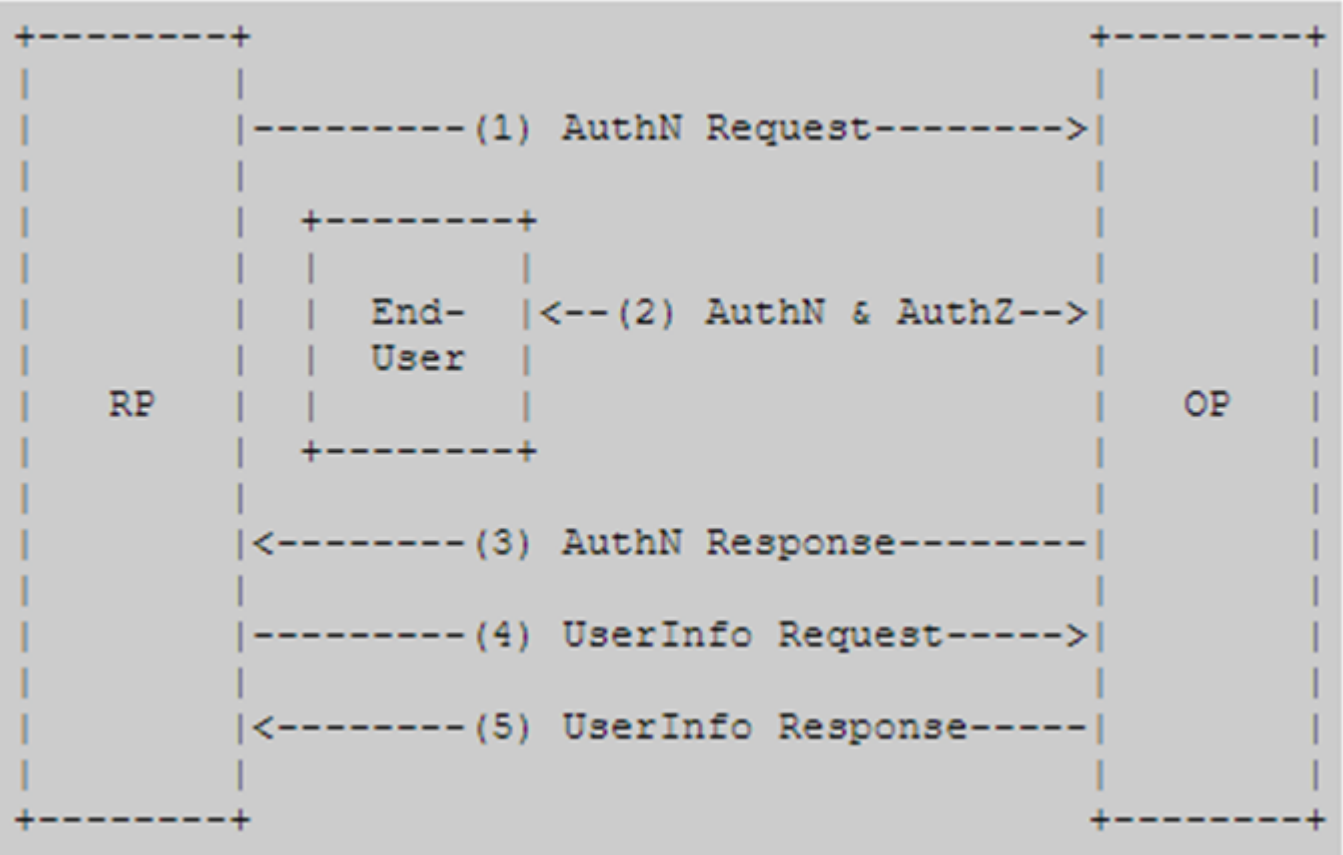
Authorization Endpoint: It is an endpoint defined in OAuth2. It is responsible for the authentication and consent process of the end user.
Token Endpoint: Allows the exchange of a client application with the authorization code or client Id and client secret and access token.
UserInfo Endpoint: It is an endpoint defined with OIDC. A client or resource server is the point where additional claim requests are provided.

You can check the details of **OpenId Connect Endpoints** and other details in below urls. I am sharing reference document links into video.
OpenID Connect Core 1.0 (spec)
OpenID Connect Discovery 1.0 (spec)

OpenID Connect Authentication Flows

As you know that **OAuth2** has defining **Authorization Grant** and **Extension Grant** (extension authorizations), so as the same way OIDC defines authentication flows.

There are three authentication flows. These flows differ according to the parameters passed to the **OpenId Provider (OP),** the content of the responses and how they are processed. The difference between the requests made to the Authorization Endpoint is determined by the **response_type** parameter.



Flows and parameter values;

Authorization Code Flow -> “code”
Implicit Flow -> “id_token” or“id_token token”
Hybrit Flow -> “ code id_token” or“code token” or “code id_token token”

When we see the code value in the **response_type** parameter, the authorization endpoint always returns the authorization code.

If an **id_token** is included in a **response_type**, the id_token is included in the response, or an access token will be included in the response from the auhorization endpoint if the token is included in the **response_type** parameter.

In short, this parameter determines the response structure and which operation steps the client application will use.

For these three flows, the **Authorization Code Flow** is an extension of the **OAuth2 Authorization Code Grant**.

OIDC Implicit Flow and **Hybrid Flow** are an extension of **Authorization Code** Flow.

What is Identity Server 4 ?

Identity **Server4** is an open source framework which implements OpenId Connect and OAuth2 protocols for .Net Core.
With **IdentityServer**, we can provide authentication and access control for our web applications or Web APIs from a single point between applications or on a user basis.

IdentityServer determines how is your web or native client applications that want to access Web Api or Api (Resource) in corporate applications or modern web applications can be accessed using authentication and authorization. So for this operations, there is no need to write identification specific to the client application on the Web API side.

You can call this centralized security system as a security token service, identity provider, authorization server, IP-STS and more.
As a summary **IdentityServer4** provides that issues security tokens to clients.

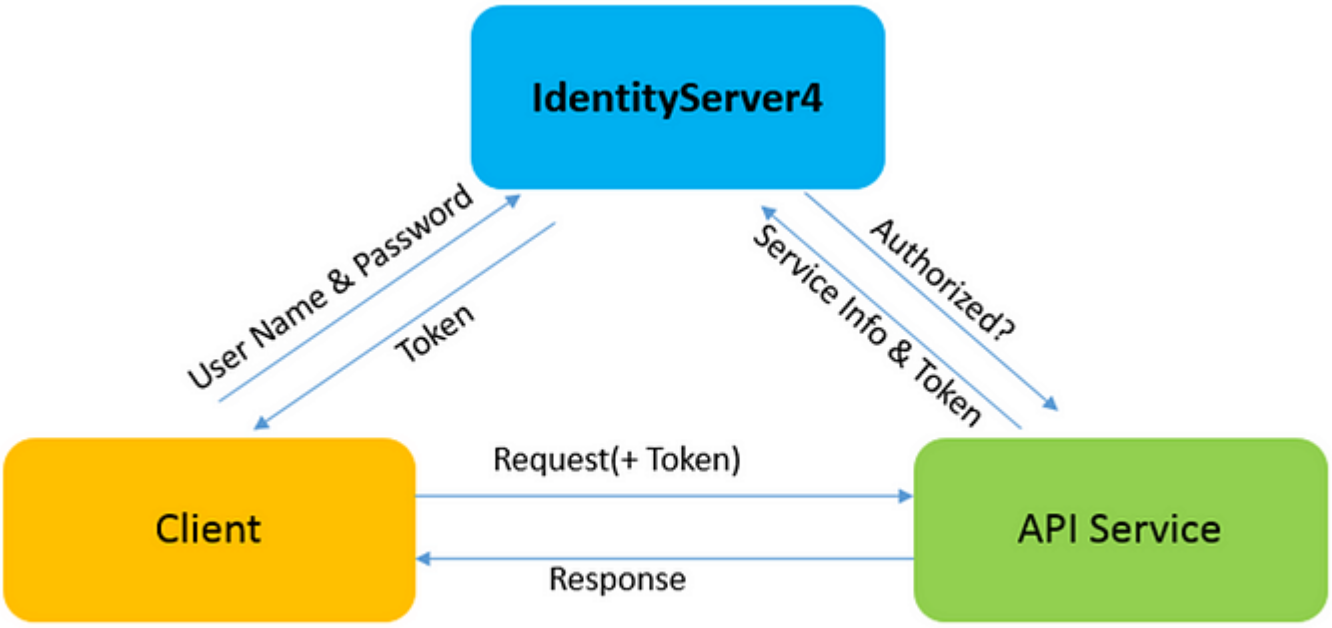
IdentityServer has a number of jobs and features — including:

- Protect your resources
- Authenticate users using a local account store or via an external identity provider

- Provide session management and single sign-on
- Manage and authenticate clients
- Issue identity and access tokens to clients
  validate tokens

Identity Server 4 Terminologies

Basically, we can think of our structure as client applications (web, native, mobile), data source applications (web api, service) and **IdentityServer** application. You can see these system into image.



Client Applications
Client Applications are applications that want to access secure data sources (Web Api) that users use. A client must be first registered with IdentityServer before it can request tokens.
Examples for clients are web applications, native mobile or desktop applications, SPAs, server processes etc.

Resources
Data Resources are the data we want to be protected by IdentityServer. Data sources must have a unique name and the client that will use this resource must access the resource with this name.

IdentityServer
IdentityServer that we can say **OpenId Connect** provider over the **OAuth2** and **OpenId** protocols. In short, it provides security tokens to client applications. IdentityServer protects data sources, ensures authentication of users, provides single sign-on and session management, verifies client applications, provides identity and access tokens to client applications and checks the authenticity of these tokens.

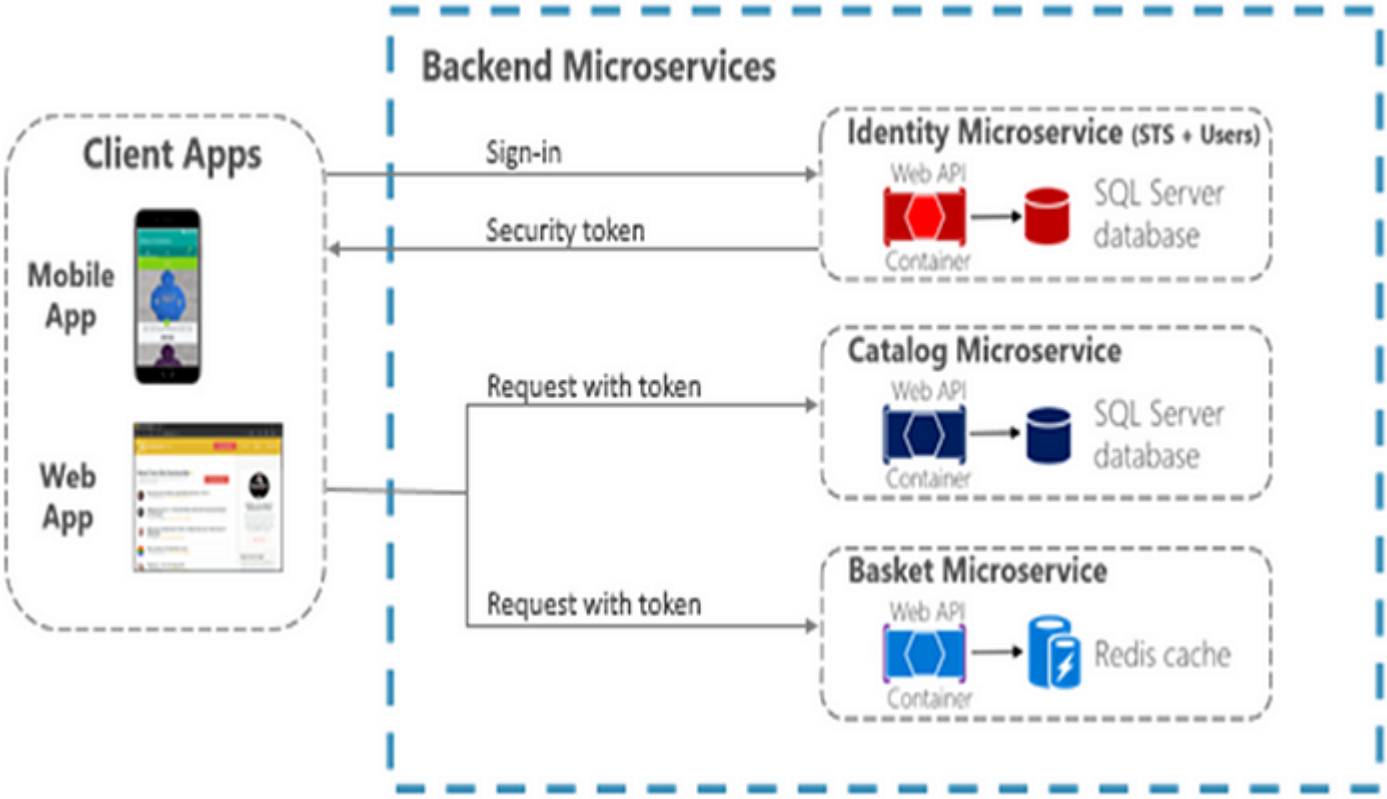
Identity Token
Identity Token represents to the result of the authentication process. It contains a sub-identifier for the user and information about how and when the user will be authenticated. It also contains identity information.

Access Token
Access Token provides access to the data source (API). The client application can access the data by sending a request to the data source with this token. Access token contains client and user information. The API uses this information to authorize and allow access to data.

Identity Server 4 in Microservices World

The security of the application we have developed that the protection and management of the data used by the application are very important. With the developing technology, the access of applications or devices in different technologies to our data may have some security and software problems or additional loads for us.

For example, let's assume that the users of a **SPA (Single Page App) Client application** that will use a Web API we have developed log into the system with their username and password. Then, when another client web application to our same **Web API** wants to verify with windows authentication and enter, a native mobile client application wants to access or another web API wants to access the Web API that we have developed, the security supported by each client application technology, we may need to develop or configure the security part of our application.



In such a case, you can see the image, we need a central (Authentication / Authorization Server) application on the basis of **client** and **resource** (api) dynamically for the access and security of client applications and data sources for the same user groups.
In microservice architectures, authentication is handled centrally by nature.