# JSON WEB TOKEN (JWT)

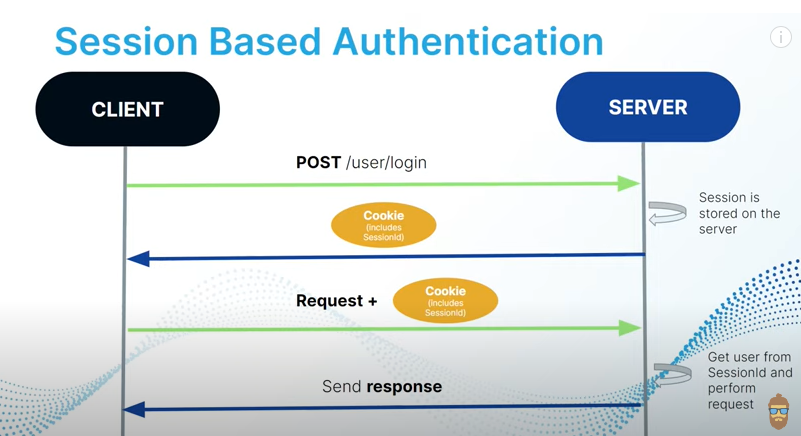
Es un estándar abierto que permite definir de una manera compacta y autocontenida transmitir información entre distintas aplicaciones o distintas entidades utilizando objetos json.

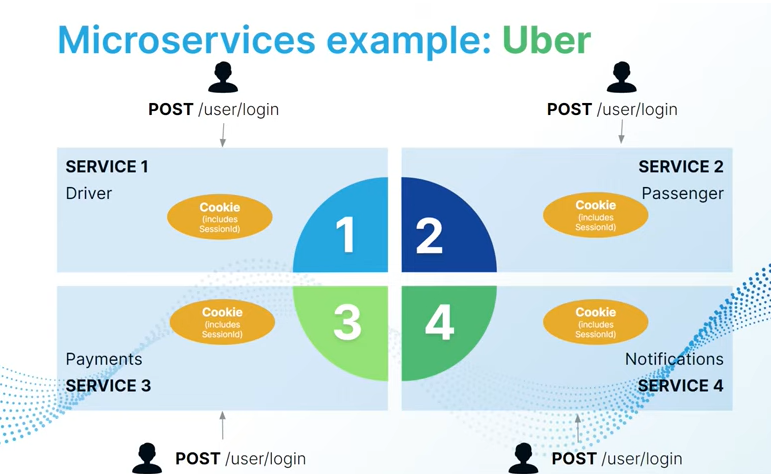
La información que está contenida en el token puede ser verificada y confiada porque está firmada digitalmente utilizando un secreto que solamente conocen el cliente y el servidor o las aplicaciones que se quieren conectar y que utilizan un algoritmo para encriptar ese secreto que puede ser una llave pública o privada.

El caso de uso más común para un JWT es el de autorización, en donde un usuario va a utilizar un JWT en cada uno de los request que haga (por ejemplo a una API) para poder definir quien es y qué accesos tiene en cada una de las solicitudes.

**SESSION BASED AUTHENTICATION**

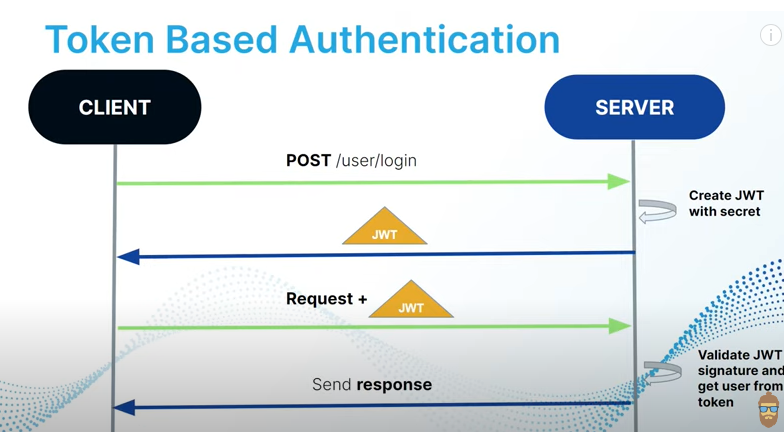
***Se crea y se almacena la sesión del usuario en el servidor***

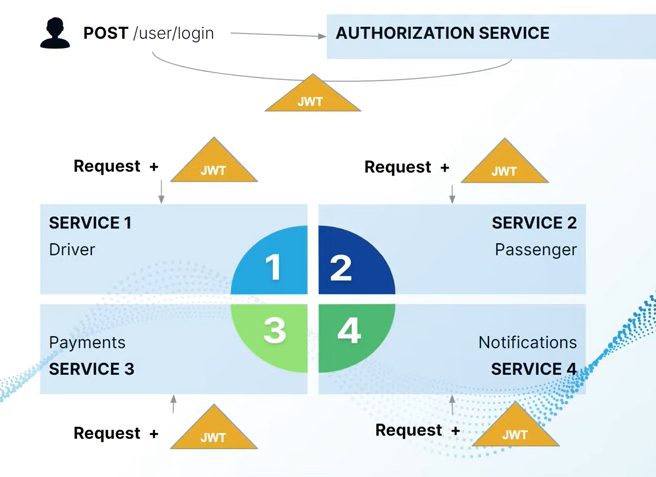




**TOKEN BASED AUTHENTICATION**

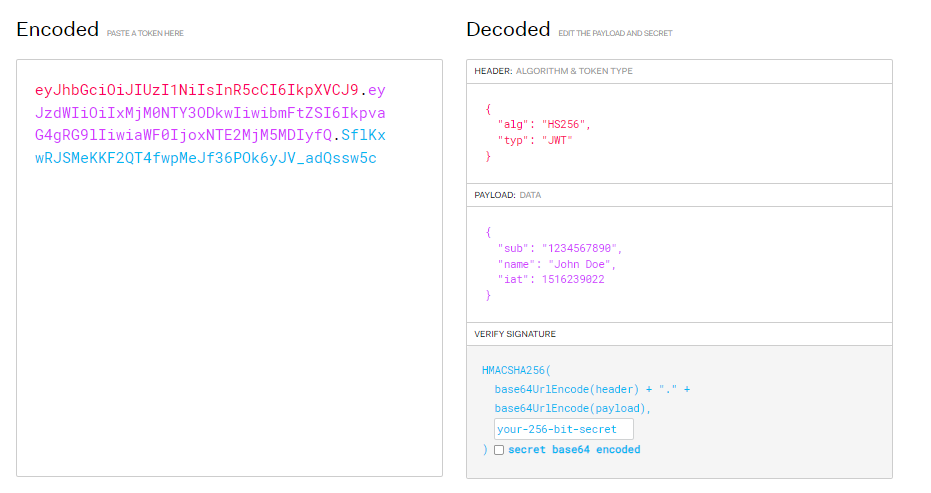
***Se crea un JWT con un secreto (No se almacena en el servidor)***





* El servidor no tiene que almacenar nada.
* El servidor crea el JWT utilizando un secreto, lo firma digitalmente y lo envía al cliente.
* Con una arquitectura como la de microservicios el cliente NO necesita autenticarse todas las veces y crear y recibir una cookie cada vez.

**La estructura de un JWT es así:**



Header:

* Define el tipo de toquen
* Define el algoritmo utilizado

Payload:

* Aquí se encuentran los datos
* Cada uno de los datos es lo que se conoce como claims
* Los claims son propiedades de la entidad. (Por ejemplo ‘username’, además se pueden agregar datos adicionales)
* Existen 3 tipos de claims
  + Registrados:
    - Son claims predefinidos.
    - No son obligatorios, pero si recomendados
    - “sub” puede ser el id del usuario, “exp” puede ser cuando expira el token.
  + Públicos:
    - Pueden ser definidos como queramos
  + Privados:
    - Claims personalizados que contienen información propia de la aplicación
    - Ejemplo: “Admin”: true

## PRACTICA

Endpoints

|  |  |  |  |
| --- | --- | --- | --- |
| https://localhost:7036/api/login | POST | token | PUBLICA |
| https://localhost:7036/api/country | GET | listCountry | PUBLICA |
| https://localhost:7036/api/product | GET | listProduct | LOGUEADOS (No importa el rol) |
| https://localhost:7036/api/employee | GET | listEmployee | ADMINISTRADOR |

1. Crear un proyecto “ASP.NET Core Web Api”
2. Para generar el token se va a utilizar el algoritmo HS256
3. En el archivo “appsettings.json” se crea la siguiente configuración:

"Jwt": {

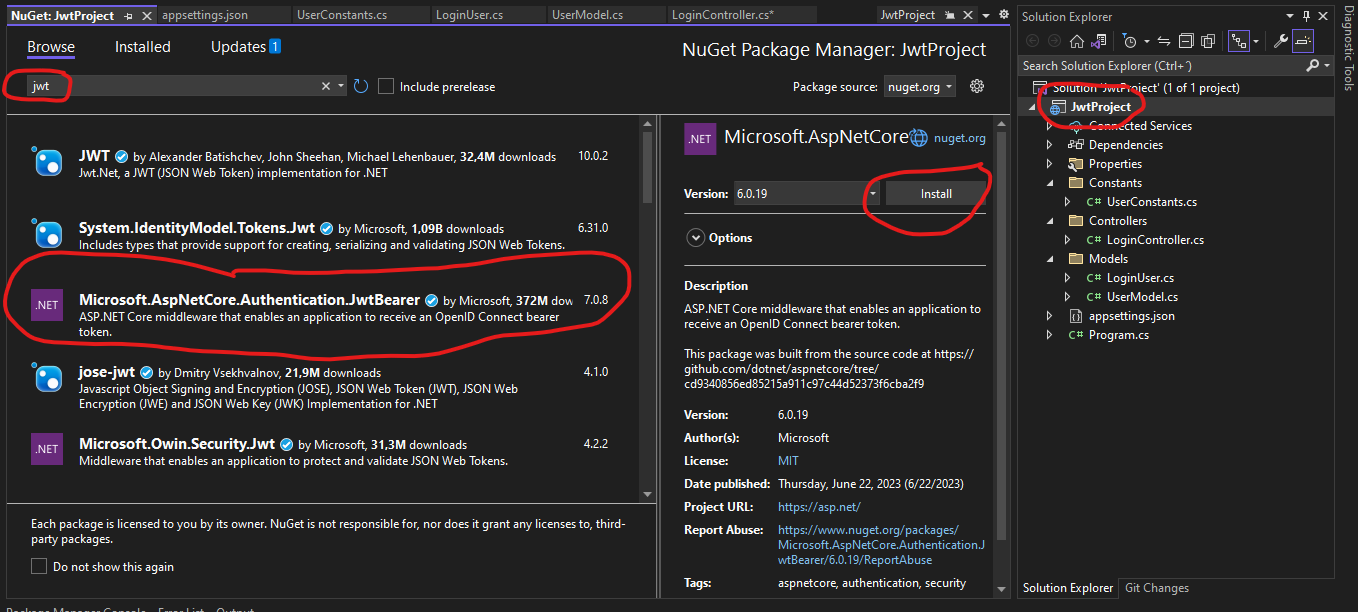
"key": "Axg%wO#Pi!c2q^34LR17",

"Issuer": "https://localhost:4200",

"Audience": "https://localhost:4200"

}

1. Instalar el nuget en la aplicación Web Api



1. En la clase ‘***Program.cs’*** se agrega el siguiente código después del middleware ‘***builder.Services.AddControllers();***’:

builder.Services.AddControllers();

//TODO: Se le agrega al contenedor del servicio la autentificación.

//TODO: Cada vez que el usuario se loguee (envía una petición al servidor) primero va a pasar por este middelware (verifica si tiene token, si expiró si está corrupto etc )

builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme).AddJwtBearer(options =>

{

options.RequireHttpsMetadata = false;

options.SaveToken = true;

options.TokenValidationParameters = new TokenValidationParameters()

{

//TODO: Valida el 'Issuer'. Lo que se creó en el archivo 'appsettings.json'

ValidateIssuer = true,

//TODO: Valida el 'Audience'. Lo que se creó en el archivo 'appsettings.json'

ValidateAudience = true,

//TODO: Valida si el token expiró

ValidateLifetime = true,

//TODO: Valida la clave

ValidateIssuerSigningKey = true,

ValidAudience = builder.Configuration["Jwt:Audience"],

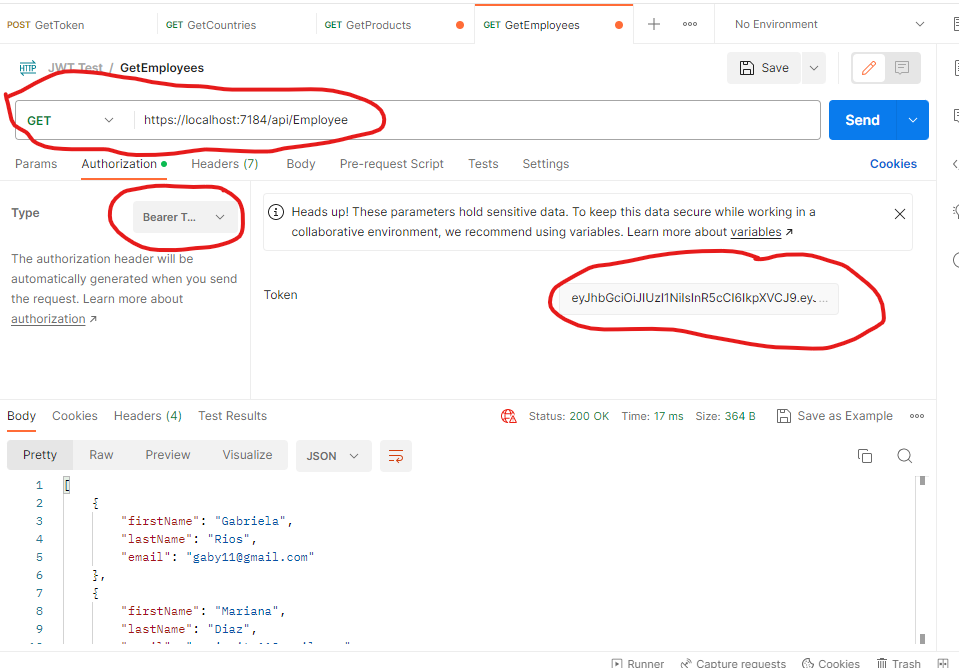
ValidIssuer = builder.Configuration["Jwt:Issuer"],

IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(builder.Configuration["Jwt:key"]))

};

});

1. La API ‘country’ es pública y no requiere autorización
2. La API ‘product’ No es pública y requiere que ***el usuario deba tener autorización por medio de un token***. Por esta razón se pone la etiqueta ***[Authorize]***. Esta etiqueta puede ser a nivel de método o a nivel de todo el controlador.
3. La API ‘employee’. No es pública y requiere que ***el usuario deba tener autorización por medio de un token y además pertenecer al rol ‘Administrador’*** Por esta razón se pone la etiqueta ***[Authorize(Roles = ("Administrador"))]***. Esta etiqueta puede ser a nivel de método o a nivel de todo el controlador.
4. Desde postman se debe realizar el request solicitando previamente el token y enviándolo en la cabecera para solicitar el recurso de la API ‘employee’



1. Se construye el método ‘Get’ en la API ‘Login’ para acceder a los claims utilizados del usuario que solicitó el token.

[HttpGet]

public IActionResult Get()

{

var currentUser = this.GetCurrentUser();

return Ok($"Hola {currentUser.FirstName}, tu eres {currentUser.Rol}");

}