Clase N° 16: Autenticación y Autorización

Crombie

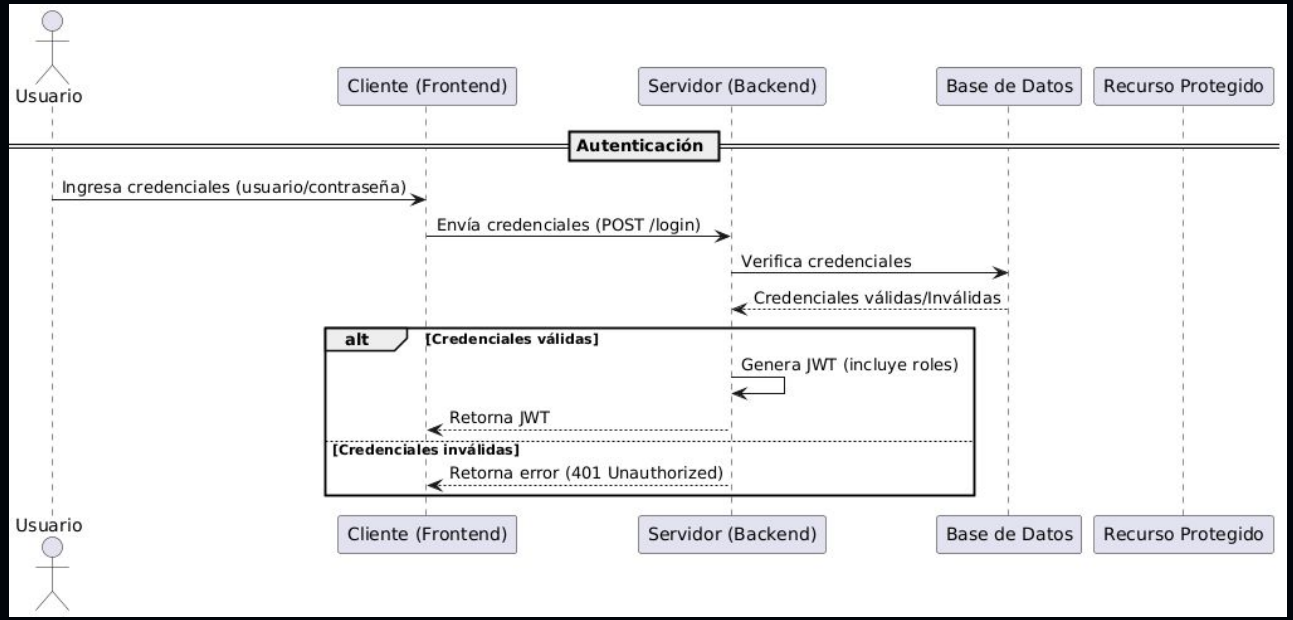
Clase N° 16: Agenda

- ¿Qué es la autenticación?
- Por qué es importante.
- Uso de tokens JWT.
- Como autenticar endpoints en en ASP.NET

Crombie

Qué es la autenticación?

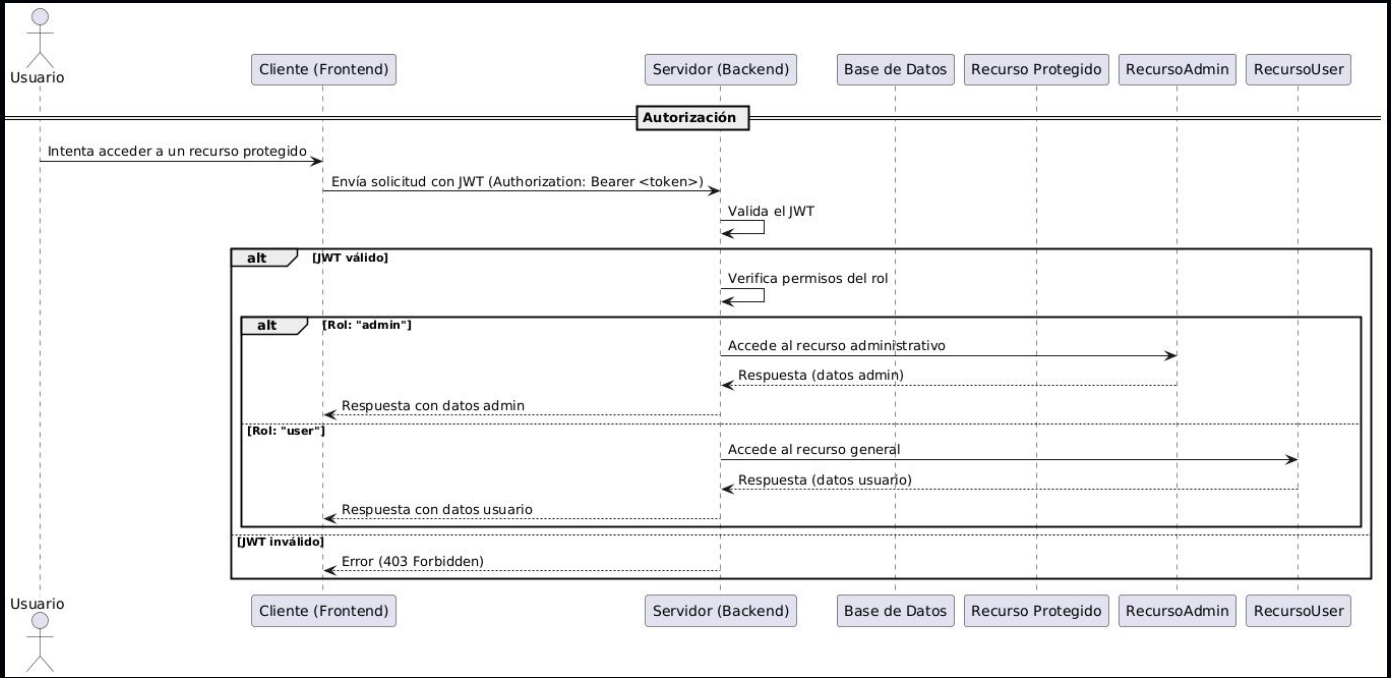
➔ Proceso de verificar la identidad de un usuario para otorgarle acceso a recursos específicos.



Crombie

Qué es la autorización?

➔ Proceso de verificar determinar a qué recursos tiene permiso un usuario (¿Qué puedes hacer?).



Crombie

Por qué es importante?

- ➔ Protege datos sensibles.
- ➔ Permite personalizar la experiencia del usuario.

Crombie

Algunas formas de autenticación

- Single Sign-On (SSO):
  - Permite a los usuarios acceder a múltiples aplicaciones con una sola credencial.
  - Ejemplo: Iniciar sesión con Google o Microsoft en varias aplicaciones.
  - Ventajas: Simplifica la experiencia del usuario y mejora la seguridad.

- Magic Link (enviado al email):
  - El usuario proporciona su correo electrónico y recibe un enlace único para iniciar sesión.
  - No requiere contraseña, minimizando riesgos asociados a contraseñas débiles o robadas.

Algunas formas de autenticación

- Con credenciales:
  - Método clásico con usuario y contraseña.
  - Requiere almacenamiento seguro de contraseñas (p. ej., usando hashing con bcrypt).

- OAuth:
  - Protocolo estándar para delegar autenticación y autorización.
  - Ejemplo: Iniciar sesión con cuentas de terceros como Google, GitHub o Facebook.
  - Permite obtener acceso controlado a recursos sin compartir contraseñas.

Cómo elegir uno?

- ➔ Facilidad de uso para el usuario.
- ➔ Requisitos de seguridad del proyecto.
- ➔ Compatibilidad con dispositivos y plataformas.
- ➔ Escalabilidad para futuros usuarios o servicios.

¿Qué son los JWT?

- Los JWT (JSON Web Tokens) son un estándar para representar información de forma compacta y segura entre dos partes, como un objeto JSON.

- Contienen tres partes separadas por puntos:
  - header
  - payload
  - signature

- Se utilizan para autenticación y autorización

¿Qué son los JWT?

➔ Header:

El encabezado contiene información sobre el tipo de token y el algoritmo de firma que se utiliza.

Ejemplo:

{ "alg": "HS256", "typ": "JWT" }

¿Qué son los JWT?

- Body:

  El payload contiene los claims, que son los datos que se quieren transmitir.

  Ejemplo:

```
{
    "sub": "1234567890",
    "name": "John Doe",
    "role": "admin",
    "iat": 1516239022
}
```

¿Qué son los JWT?

➔ Signature:

La firma asegura la integridad del token.

Si se usa un algoritmo simétrico como HS256, se requiere una clave secreta compartida.

Si se usa un algoritmo asimétrico como RS256, se utiliza una clave privada para firmar y una clave pública para verificar.

Implementandolo en ASP.NET

Primer paso:

Modificar actual UserService para no almacenar las credenciales como texto plano.

```
private readonly PasswordHasher<User> _passwordHasher;
0 references
public AuthService(UserService userService, IConfiguration configuration)
{

    this._userService = userService;
    this._configuration = configuration;
    this._passwordHasher = new PasswordHasher<User>();
```

AspNetCore.Identity ofrece PasswordHaser, una clase que puede usarse con esta finalidad.

Implementandolo en ASP.NET

```
User NewUser = new User();
NewUser.Id = Guid.NewGuid();
NewUser.Password = this._passwordHasher.HashPassword(NewUser, user.Password);
NewUser.Name = user.Name;
NewUser.Email = user.Email;
NewUser.CreatedDate = DateTime.Now;
NewUser.Role = user.Role?.ToLower();
await this._userService.CreateUser(NewUser);
return NewUser;
```

Implementandolo en ASP.NET

Nugget necesario:

- JWT Bearer

dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer

En nuestro Program.cs añadimos el uso de autenticacion y autorizacion en el builder y configuramos para el uso de JWT.

Además indicamos a la app que use dicha autenticacion y autorizacion.

Implementandolo en ASP.NET

```
builder.Services.AddAuthorization();
builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme).AddJwtBearer(o =>
{
    o.RequireHttpsMetadata = false;
    o.TokenValidationParameters = new TokenValidationParameters
    {
        IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(builder.Configuration["JWT:SECRET"])),
        ValidIssuer = builder.Configuration["JWT:ISSUER"],
        ValidAudience = builder.Configuration["JWT:AUDIENCE"],
        RoleClaimType = "Role",
        ClockSkew = TimeSpan.Zero,
    };
});
```

```
app.UseAuthentication();
app.UseAuthorization();
```

Implementandolo en ASP.NET

Con eso tendríamos lista nuestra aplicación para validar JWT, pero necesitamos en primera instancia crear los tokens de usuario.

El token JWT debería ser generado una vez las credenciales del usuario sean validadas.

```
PasswordVerificationResult IsCorrectPassword = this._passwordHasher.VerifyHashedPassword(logginUser, logginUser.Password, credentials.Password);

if (IsCorrectPassword == PasswordVerificationResult.Success)
{
    return this.CreateJWTAuthToken(logginUser);
}
else
{
    throw new Exception("Unable to authenticate");
}
```

Implementandolo en ASP.NET

Vamos a crear un servicio de generación de token, en el que se añadan al body del token los claims necesarios y se genere, con un secreto elegido por nosotros, la signature.

```
public string CreateJWTAuthToken(User user)
{
    string secretKey = this._configuration["JWT:SECRET"] ?? "";
    var securityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(secretKey));
    var credentials = new SigningCredentials(securityKey, SecurityAlgorithms.HmacSha256);

    if (secretKey == null)
    {
        throw new Exception("Unable to create token");
    }


    var tokenDescriptor = new SecurityTokenDescriptor
    {
        Subject = new ClaimsIdentity([
            new Claim(JwtRegisteredClaimNames.Sub, user.Id.ToString()),
            new Claim(JwtRegisteredClaimNames.Name, user.Name.ToString()),
            new Claim(JwtRegisteredClaimNames.Email, user.Email.ToString()),
            new Claim("Roles", user.Role)
             ]),
        Expires = DateTime.UtcNow.AddMinutes(15),
        SigningCredentials = credentials,
        Issuer = this._configuration["JWT:ISSUER"],
        Audience = this._configuration["JWT:AUDIENCE"]
    };

    var handler = new Microsoft.IdentityModel.JsonWebTokens.JsonWebTokenHandler();

    string token = handler.CreateToken(tokenDescriptor);

    return token;
}
```

Crombie

Implementandolo en ASP.NET

```
public string CreateJWTAuthToken(User user)
{
    string secretKey = this._configuration["JWT:SECRET"] ?? "";
    var securityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(secretKey));
    var credentials = new SigningCredentials(securityKey, SecurityAlgorithms.HmacSha256);

    if (secretKey == null)
    {
        throw new Exception("Unable to create token");
    }


    var tokenDescriptor = new SecurityTokenDescriptor
    {
        Subject = new ClaimsIdentity([
            new Claim(JwtRegisteredClaimNames.Sub, user.Id.ToString()),
            new Claim(JwtRegisteredClaimNames.Name, user.Name.ToString()),
            new Claim(JwtRegisteredClaimNames.Email, user.Email.ToString()),
            new Claim("Roles", user.Role)
             ]),
        Expires = DateTime.UtcNow.AddMinutes(15),
        SigningCredentials = credentials,
        Issuer = this._configuration["JWT:ISSUER"],
        Audience = this._configuration["JWT:AUDIENCE"]
    };

    var handler = new Microsoft.IdentityModel.JsonWebTokens.JsonWebTokenHandler();

    string token = handler.CreateToken(tokenDescriptor);

    return token;
}
```

Implementandolo en ASP.NET

Solo queda pendiente indicar en que endpoints queremos usar autenticación. Podemos hacerlo general al controlador (autenticación a todos los métodos de esa entidad):

```
[Authorize]
[Route("api/[controller]")]
[ApiController]

1 reference
public class ProductsController : ControllerBase
```

Específicas a un endpoint, o específicas a un rol:

```
// GET: api/Products
[Authorize]
[HttpGet]
```

```
// PUT: api/Products/5
[Authorize(Roles = "admin")]
[HttpPut("{id}")]
```