

Práctica. Backend API en .NET con autenticación.

Instrucciones. Elabore una aplicación web de .NET utilizando el patrón de diseño de software MVC con controladores y modelos que implemente el método de autenticación JWT.

1. Sigue las instrucciones de la práctica en la siguiente página.
2. Crea un reporte de práctica en un **archivo de Word** con una portada con tu nombre. Guárdalo con el nombre **NombreAlumno_APINetAuth.docx**.
3. Coloca **las siguientes capturas de pantalla**, cada una con una **descripción textual** arriba de la captura.
 - a. Coloca la captura de pantalla de tu código fuente de tus archivos donde se agrega la funcionalidad para la seguridad de la aplicación.
 - b. Coloca la captura de pantalla de Visual Studio Code con la extensión REST API ejecutando la consulta POST donde se vean la obtención de un JWT.
 - c. Coloca la captura de pantalla de Visual Studio Code con la extensión REST API ejecutando la consulta GET donde se vean tres usuarios agregados por ti.
 - d. Coloca la captura de pantalla de la consola de MySQL listando las tablas en tu base de datos de MySQL creadas por el ORM.
 - e. Publica tu código fuente en un proyecto público de GitHub. **Coloca la URL** del código fuente publicado en GitHub.
4. Sube tu reporte de práctica archivo Word a la actividad en Eminus.

Prerrequisitos

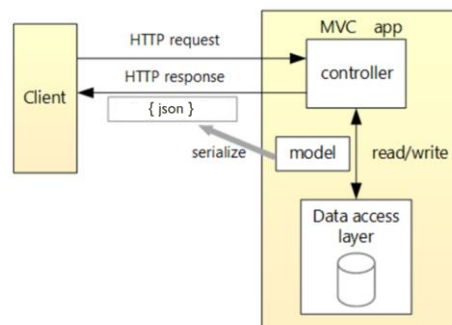
1. **Esta práctica tiene como prerrequisito haber realizado la práctica anterior de Backend .NET.**
2. Esta práctica tiene como prerrequisito tener instalado el software **Visual Studio Code** con la extensión para C# y la extensión REST Client.
Puede seguir las instrucciones de cómo instalar Visual Studio Code desde la práctica [Instalación de Visual Studio Code](#).
3. Esta práctica tiene como prerrequisito tener instalado tener instalado el SDK de .NET descárguelo desde <https://dotnet.microsoft.com/en-us/download/dotnet/8.0>.
4. Esta práctica tiene como prerrequisito tener instalado MySQL en su equipo.
Puede seguir las instrucciones de cómo instalar MySQL desde la práctica [Instalación de MySQL](#).
5. Puede seguir las instrucciones de cómo publicar un proyecto en Visual Studio Code a GitHub desde la práctica [Como publicar un proyecto a GitHub desde Visual Studio Code](#).

Instrucciones. Defina su API.

Lo primero que tenemos que realizar, es definir su API. no comience a escribir código y crear estructuras sin saber lo que se quiere realizar. Esta práctica crea la siguiente API:

API	Descripción	Cuerpo de la solicitud	Cuerpo de la respuesta	Rol requerido
<i>GET</i> /api/categorias	Obtener todas las categorías	Ninguno	Lista de categorías	Administrador
<i>GET</i> /api/categorias/{id}	Obtener una categoría por ID	Ninguno	Categoría	Administrador
<i>POST</i> /api/categorias	Agregar una nueva categoría	Categoría	Categoría agregada	Administrador
<i>PUT</i> /api/categorias/{id}	Actualizar una categoría existente	Categoría	Ninguno	Administrador
<i>DELETE</i> /api/categorias/{id}	Eliminar una categoría	Ninguno	Ninguno	Administrador
<i>GET</i> /api/peliculas	Obtener todas las películas	Ninguno	Lista de películas	Administrador, Usuario
<i>GET</i> /api/peliculas?s=titulo	Obtener todas las películas que contengan en el título la cadena s	Ninguno	Lista de películas	Administrador, Usuario
<i>GET</i> /api/peliculas/{id}	Obtener una película por ID	Ninguno	Película	Administrador, Usuario
<i>POST</i> /api/peliculas	Agregar una nueva película	Película	Película agregada	Administrador
<i>PUT</i> /api/peliculas/{id}	Actualizar una película existente	Película	Ninguno	Administrador
<i>DELETE</i> /api/peliculas/{id}	Eliminar una película	Ninguno	Ninguno	Administrador
<i>GET</i> /api/usuarios	Obtener todos los usuarios	Ninguno	Lista de usuarios	Administrador
<i>GET</i> /api/usuarios/{id}	Obtener un usuario por ID	Ninguno	Usuario	Administrador
<i>POST</i> /api/usuarios	Agregar un nuevo usuario	Usuario	Usuario agregado	Administrador
<i>PUT</i> /api/usuarios/{id}	Actualizar un usuario existente	Usuario	Ninguno	Administrador
<i>DELETE</i> /api/usuarios/{id}	Eliminar un usuario	Ninguno	Ninguno	Administrador
<i>GET</i> /api/auth	Inicio de sesión	Ninguno	JWT	Ninguno

El siguiente diagrama muestra el diseño de la aplicación.



¿Qué es un JSON Web Token?

Un JSON Web Token (JWT) es un estándar ([RFC 7519](#)) que define una forma segura y compacta de transmitir información entre dos entidades en forma de un objeto JSON.

Esta información puede ser verificada y es confiable ya que está firmada digitalmente. Los JWTs pueden ser firmados utilizando una llave privada (con un algoritmo [HMAC](#)) o con llaves públicas y privadas utilizando [RSA](#) o [ECDSA](#).

La autenticación basada en tokens carece de estado (es stateless).

¿Cuándo se debería utilizar Json Web Tokens?

Aquí veremos un par de escenarios donde es útil y recomendable utilizar los JWTs:

- **Autorización:** Este es el caso de uso más común de los JWTs. Una vez que un usuario ha iniciado sesión, cada llamada subsecuente al servicio incluirá el JWT, permitiendo al usuario acceder a rutas, servicios o recursos que solo están permitidos con su debido token. SSO (Single Sign On) es una funcionalidad que hoy en día usa los JWTs ampliamente, porque son de tamaño reducido y por su habilidad de ser usado entre diferentes dominios.
- **Intercambio de Información:** Los JWTs son útiles también para transmitir información entre dos entidades. Debido a que los JWTs pueden estar firmados — por ejemplo, utilizando una llave pública/privada — podemos estar seguros que quien manda la información es verdaderamente él quien lo manda. Adicionalmente, la firma es calculada utilizando el encabezado del JWT y el contenido (payload) por lo que también estamos seguros que el contenido del JWT no fue alterado.

¿Qué estructura tiene un JWT?

Un JWT está separado por puntos (.) en tres partes, las cuales son:

1. Encabezado (**header**)
2. Contenido (**payload**)
3. Firma (**signature**)

Un JWT comúnmente tiene la siguiente forma.

```
xxxxx.yyyyy.zzzzz
```

Veamos que significa cada una de estas partes.

1. Header

El encabezado *típicamente* consiste de dos partes: el tipo de token (que será JWT) y el algoritmo que se está usando en la firma, que puede ser **HMAC SHA256** o **RSA**.

Por ejemplo:

```
{  
  "alg": "http://www.w3.org/2001/04/xmldsig-more#hmac-sha256",  
  "typ": "JWT"  
}
```

Después, este JSON se codifica en **Base64URL** para formar parte del primer segmento del JWT.

2. Payload

La segunda parte del JWT es el contenido que se transmite o certifica (payload), el cual contiene la serie de claims. Claims son afirmaciones sobre una entidad (usualmente, el usuario) e información adicional. Hay tres tipos de claims: registrados, públicos y privados.

- **Claims registrados:** Son un conjunto de claims predefinidos que no son obligatorios pero sí recomendados, para proveer un conjunto de claims interoperables. Algunos de ellos son: **iss** (issuer), **exp** (tiempo de expiración), **sub** (subject), **aud** (audience), entre [otros](#).

Nótese que los nombres de los claims son de tres letras por la misma intención de mantener el JWT de tamaño reducido.

- **Claims públicos:** Estos pueden ser definidos como cada quien desee, pero para evitar colisiones de nombres y mantener un estándar (ya que puede usarse en distintos servicios), se utiliza la siguiente lista llamada [IANA JSON Web Token Registry](#).
- **Claims privados:** Estos claims son personalizados por cada quien que implemente los JWTs y al igual que los públicos, para evitar colisiones es recomendable utilizar un formato URL con algún namespace y así asegurar que son únicos
 - Por ejemplo, un claim que guarda los roles de ASP.NET Core tendría el siguiente nombre: `http://schemas.microsoft.com/ws/2008/06/identity/claims/role`.

Un ejemplo de un payload sería el siguiente:

```
{  
  "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/sid": "09e30209-acbf-4391-baea-c719dd271c5f",  
  "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name": "gvera@uv.mx",  
  "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress": "gvera@uv.mx",  
  "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname": "Guillermo Humberto Vera Amaro",  
  "http://schemas.microsoft.com/ws/2008/06/identity/claims/role": "Administrador",  
  "exp": 1701147212,  
  "iss": "ServidorFeiJWT",  
  "aud": "ClienteFeiJWT"  
}
```

Y al igual que el header, este segmento se codifica en **Base64Url**.

Nota: Aunque los JWT estén firmados, solo están protegidos para evitar falsificaciones (editar el payload) pero de igual forma, toda la información en el payload es visible para cualquiera. **NO se recomienda incluir información sensible en el payload** al menos que esté cifrada.

3. Signature

Para crear la firma debemos de tomar el header codificado, el payload codificado, una llave secreta, el algoritmo especificado en el header y firmar todo eso.

Por ejemplo, si vamos a utilizar el algoritmo de encriptación HMAC SHA256, la firma será creada de la siguiente forma:

```
HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secret)
```

La firma se usará para verificar que el mensaje no ha cambiado mientras viaja por la red, y en caso de ser tokens firmados por una llave privada de un certificado, también se puede verificar el emisor.

Juntando todo

Al final, tendremos tres cadenas de texto codificadas en Base64-URL separadas por puntos y se podrán incluir en solicitudes HTTP o contenido HTML sin ningún problema. Esto es una forma mucho más compacta comparado a otros estándares como [SAML](#) que utiliza XML.

Al final, tendríamos un JWT de la siguiente forma:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36P0k6yJV_adQssw5c
```

Para verificar JWT, se puede visitar el sitio jwt.io.

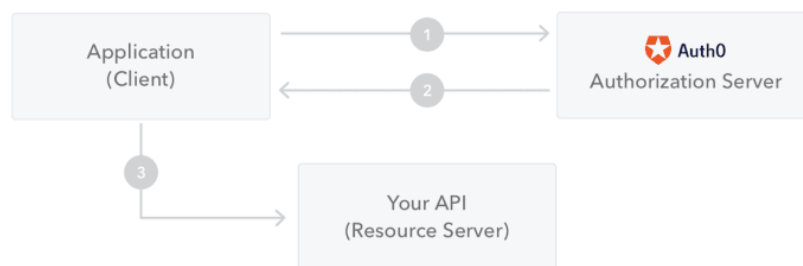
¿Cómo funcionan los JWT?

Cuando un usuario ha sido autenticado, el servicio deberá regresar un JSON Web Token para ser usado como sus credenciales. Dado que esto es usado para autorizar el usuario, debes de considerar cuidar muy bien donde guardas el token, y eliminarlo lo más pronto posible si ya no se requiere.

Cuando un usuario quiere acceder a contenido restringido en una ruta protegida, se debe de incluir el token en el HTTP Header **Authorization** y utilizando el esquema **Bearer**.

Ejemplo:

```
HeaderNames.Authorization, "Bearer " + accessToken
```



Generalmente en Web APIs (y como lo haremos más adelante) que son aplicaciones stateless, siempre requerirá que el token vaya incluido en el encabezado Authorization. El servicio verificará lo necesario para determinar si es un token válido o no, y si este es válido, leerá su información (los claims) y lo usará en la solicitud de ser necesario.

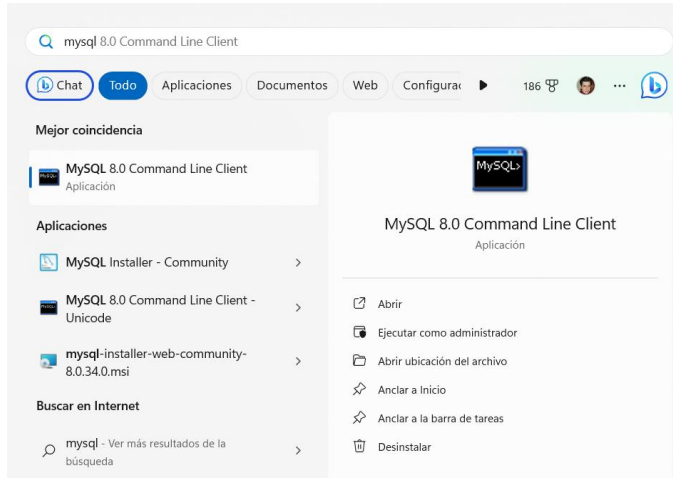
Esto también reduce las consultas a bases de datos para leer información del usuario, ya que el token puede contener información común para poder operar (como username, email, roles, etc).

Dado que el token va incluido en el header, no habrá problemas con el Cross-Origin Resource Sharing (CORS) ya que no se utilizan cookies (las cookies son por dominio).

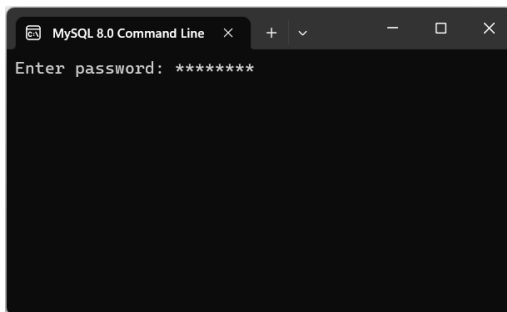
1. La aplicación cliente solicita autorización al Identity Server (como el que vamos a utilizar).
2. Cuando se autoriza el acceso, el servidor de autorización regresa el access token a la aplicación cliente.
3. La aplicación cliente usa el access token para acceder a recursos protegidos (como los endpoints de una API).

Instrucciones. Crear la base de datos

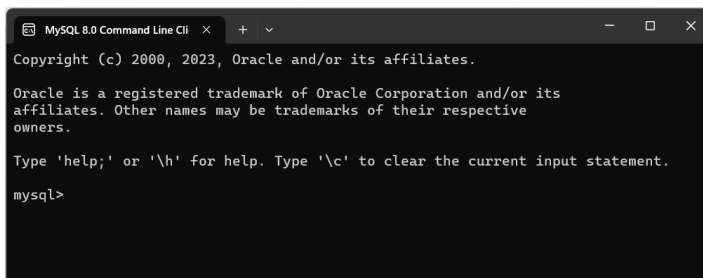
1. Usted ya debería contar con esta base de datos del proyecto anterior. Solo realice el paso de eliminarla y volverla a crear para tenerla completamente vacía.
2. En el menú de inicio de Windows, busque el programa MySQL escribiendo `mysql` y haga clic en ejecutar. También puede ejecutar la versión “**MySQL 8.0 Command Line Cliente Unicode**” que tiene soporte para acentos y caracteres especiales como la letra ñ.



3. Escriba la contraseña del super usuario `root` que creó en la instalación de MySQL.



4. Si se equivoca en la contraseña, la ventana se cerrará y tendrá que volver a abrir el programa cliente.
5. El cliente de MySQL debe verse de la siguiente forma. Esto quiere decir que se ha conectado correctamente a su servidor local MySQL con el usuario `root`.



6. Abra Visual Studio Code.
7. Asegúrese de contar con una carpeta de trabajo en `C:\codigo`. Si aún no ha creado la carpeta `C:\codigo` hágalo antes de continuar.

8. Seleccione **Archivo > Abrir carpeta** en el menú principal.
9. En el cuadro de diálogo **Abrir carpeta**, cree una carpeta en la ruta **C:\codigo\tw\backendnetauth** y selecciónela. Luego haga clic en **Seleccionar carpeta**.
10. Cree un nuevo archivo llamado **bd.sql** para crear la base de datos de la aplicación. Agregué las siguientes instrucciones al archivo.

```
DROP DATABASE IF EXISTS netflixnet;  
CREATE DATABASE netflixnet CHARACTER SET utf8mb4;  
USE netflixnet;
```

11. Dentro de MySQL, para ejecutar el script de creación de la base de datos, ejecute las siguiente instrucción.

```
mysql> source C:\codigo\tw\backendnetauth\bd.sql
```

12. Verifique que se la base de datos se haya creado.

```
mysql> show databases;  
+-----+  
| Database |  
+-----+  
| biblioteca |  
| clases |  
| information_schema |  
| mysql |  
| netflixnet |  
| netflixnode |  
| performance_schema |  
| sys |  
| tienda |  
+-----+  
9 rows in set (0.00 sec)
```

13. Ahora vamos a crear el usuario para conectarse desde su aplicación. En este ejemplo le colocaremos el nombre **netflix_user** con password **N3tf1x**, pero usted puede colocar el que prefiera.

```
mysql> CREATE USER netflix_user@localhost IDENTIFIED BY 'N3tf1x';  
Query OK, 0 rows affected (0.01 sec)
```

14. Y para darle permisos en la nueva base de datos, se utiliza:

```
mysql> GRANT ALL ON netflixnet.* TO netflix_user@localhost;  
Query OK, 0 rows affected (0.01 sec)
```

15. Si solo quisiéramos darle permisos de lectura y escritura, sin que pudiera crear objetos, se usaría:

```
mysql> GRANT SELECT, INSERT, UPDATE, DELETE ON netflixnet.* TO netflix_user@localhost;
```

16. No utilice este comando en este momento, pues queremos que el usuario si pueda crear, modificar y eliminar las tablas que serán parte de la base de datos.
17. Si quisiera revocar los permisos del usuario se podría usar:

```
mysql> REVOKE ALL ON netflixnet.* FROM netflix_user@localhost;
```


18. Para verificar que los permisos se hayan aplicado se ejecuta:

```
mysql> SHOW GRANTS FOR netflix_user@localhost;
+-----+
| Grants for netflix_user@localhost |
+-----+
| GRANT USAGE ON *.* TO `netflix_user`@`localhost` |
| GRANT ALL PRIVILEGES ON `netflixnet`.* TO `netflix_user`@`localhost` |
+-----+
2 rows in set (0.00 sec)
```

19. Cerrar la sesión con `exit;` y salir.

Instrucciones: Creación de la aplicación API.

1. Realice una copia de la carpeta del código de la práctica anterior llamada **Backend API en .NET** ubicada en **C:\codigo\tw\backendnet**.
2. Cambie el nombre de la carpeta a **C:\codigo\tw\backendnetauth**.
3. Abra esta nueva carpeta en Visual Studio Code.
4. El nombre de la carpeta se convierte en el nombre del proyecto y el nombre del espacio de nombres de forma predeterminada.
5. Abra el archivo llamado **backendnet.csproj**.
6. Verifique que tiene instalado el paquete **Microsoft.EntityFrameworkCore.Design**, el paquete **Pomelo.EntityFrameworkCore.MySql**, y el paquete **Swashbuckle.AspNetCore** como dependencias.
7. Ahora verifique que en el archivo llamado **appsettings.json** en la raíz de su aplicación se encuentren los datos correctos de su conexión a MySQL.

```
"AllowedHosts": "*",  
"ConnectionStrings": {  
  "DataContext": "Server=localhost;User ID=netflix_user;Password=N3tf1x;Database=netflixnet"  
}
```

8. En la consola escriba **dotnet build**. Y verifique que no marque errores.
9. Escriba **dotnet run** o **dotnet watch run** para ejecutar su aplicación. La diferencia con ambos, es que la segunda aplica los cambios al guardar el archivo de código fuente modificado sin necesidad de reiniciar todo el servidor web de desarrollo.

```
dotnet run
```

10. Observe que su aplicación está ejecutándose en la ruta <http://localhost:3000/>.

```
PS C:\codigo\tw\backendnetauth> dotnet run  
Compilando...  
info: Microsoft.Hosting.Lifetime[14]  
      Now listening on: http://localhost:3000  
info: Microsoft.Hosting.Lifetime[0]  
      Application started. Press Ctrl+C to shut down.  
info: Microsoft.Hosting.Lifetime[0]  
      Hosting environment: Development  
info: Microsoft.Hosting.Lifetime[0]  
      Content root path: C:\codigo\tw\backendnetauth  
~
```

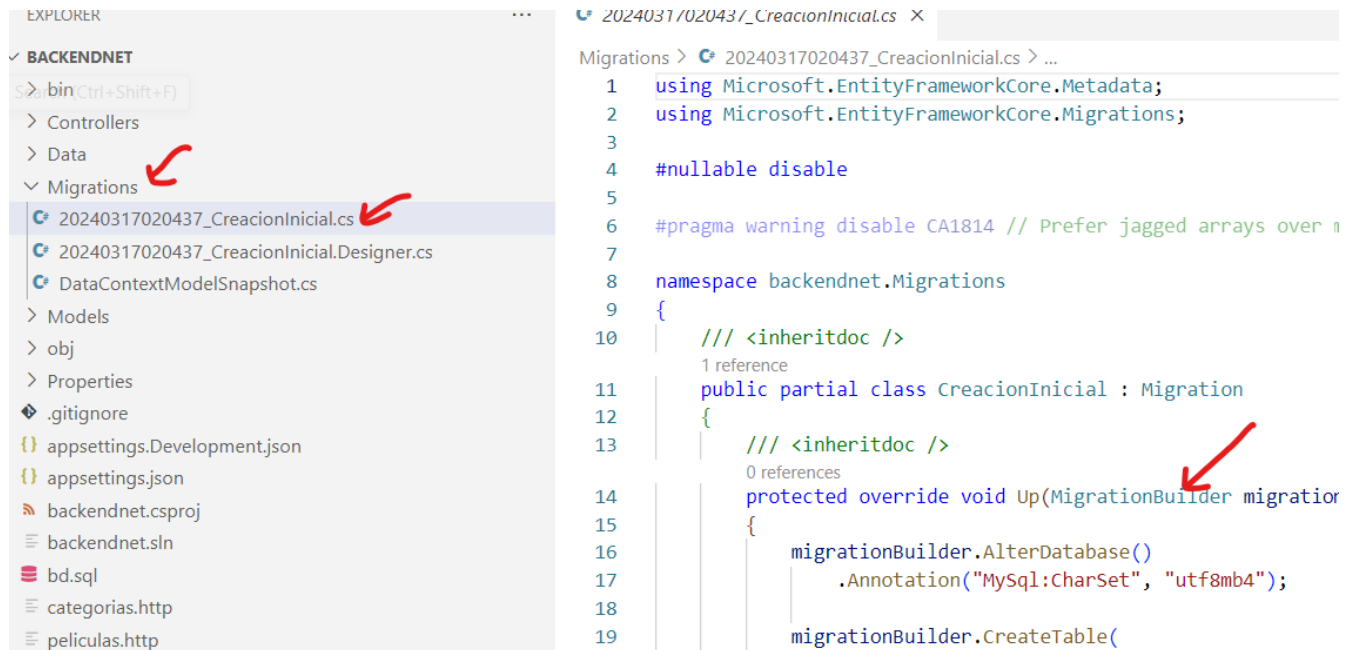
11. Para detener el servidor, solo presione **Ctrl + C** en la consola.
12. Recuerde que también puede utilizar **dotnet run --launch-profile https** para iniciar su aplicación con https utilizando un certificado SSL.

```
PS C:\codigo\tw\backendnetauth> dotnet run --launch-profile https  
Compilando...  
info: Microsoft.Hosting.Lifetime[14]  
      Now listening on: https://localhost:7198  
info: Microsoft.Hosting.Lifetime[14]  
      Now listening on: http://localhost:5101  
info: Microsoft.Hosting.Lifetime[0]  
      Application started. Press Ctrl+C to shut down.  
~
```

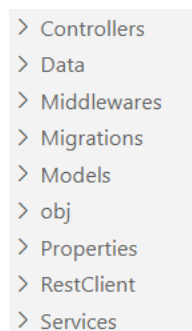
13. Para terminar la ejecución escriba en la consola **Ctrl + C** en Visual Studio Code.
14. Felicidades, ha creado su aplicación de manera correcta.

Instrucciones: Agregar la funcionalidad ORM

1. El proyecto ya cuenta con el paquete para darle el soporte de **ORM** a su aplicación. En .NET el ORM se llama **EntityFramework** y funciona para PostgreSQL, MySQL, MariaDB, Oracle, MongoDB, SQL Server entre otras. Puede encontrar más información desde [Entity Framework](#).
2. Se utiliza **dotnet ef** para generar migraciones y aplicar seeders.
3. Al ejecutar la inicialización en la práctica anterior, este comando creó la carpeta **Migrations**.



4. Recuerde que agregamos un archivo llamado **DataContext.cs** en la carpeta **Data** que es nuestro enlace con la base de datos y las clases de la aplicación.
5. Finalmente, creamos las siguientes carpetas en nuestro proyecto:
 - **Models**. Aquí almacenaremos los modelos.
 - **Controllers**. Aquí van los controladores.
 - **Data**. Aquí va la capa para el acceso a los datos.
 - **Services**. Aquí van los componentes reutilizables de código.
 - **Middlewares**. Aquí van las funciones intermedias entre peticiones y solicitudes.
6. El árbol de sus archivos debería ser similar a este:



7. Felicidades, ha configurado correctamente su aplicación.

Instrucciones. Agregar los Modelos.

1. Recuerde que ya contamos con modelos: `Categoria`, `Pelicula` y la relación muchos a muchos entre ellos llamada `CategoriaPelicula`.
2. Agregue un nuevo archivo en la carpeta `Models` llamado `AsignaCategoriaDTO.cs` con el siguiente código.

```
namespace backendnet.Models;
```

2 references

```
public class AsignaCategoriaDTO
{
    2 references
    public int? CategoriaId { get; set; }
}
```

3. Ahora vamos a crear un modelo nuevo llamado `Usuario`.
4. En .NET para el manejo de roles y usuarios con autenticación, se utiliza el paquete `Microsoft.AspNetCore.Identity.EntityFrameworkCore`. Instálelo utilizando el comando siguiente.

```
dotnet add package Microsoft.AspNetCore.Identity.EntityFrameworkCore
```

5. Agregue un nuevo archivo en la carpeta `Models` llamado `CustomIdentityUser.cs` con el siguiente código.

```
using Microsoft.AspNetCore.Identity;
```

```
namespace backendnet.Models;
```

15 references

```
public class CustomIdentityUser : IdentityUser
{
    8 references
    public required string Nombre { get; set; }
    2 references
    public bool Protegido { get; set; } = false;
}
```

6. Observe que esta clase hereda los atributos de `IdentityUser` que es la clase que tiene todos los atributos restantes como id, email y contraseña.
7. Ahora vamos a agregar los modelos para la transferencia de datos. Estos modelos se utilizan para el envío y recepción de los datos evitando utilizar las clases asociadas con la base de datos.
8. Agregue un nuevo archivo en la carpeta `Models` llamado `CustomIdentityUserDTO.cs` con el siguiente código. Este será usado para enviar y recibir datos del catálogo de usuarios.

```
namespace backendnet.Models;
```

8 references

```
public class CustomIdentityUserDTO
{
    3 references
    | public string? Id { get; set; }
    5 references
    | public required string Email { get; set; }
    4 references
    | public required string Nombre { get; set; }
    6 references
    | public required string Rol { get; set; }
}
```

9. Agregue un nuevo archivo en la carpeta **Models** llamado **CustomIdentityUserPwdDTO.cs** con el siguiente código. Este será usado para la creación de un nuevo usuario.

```
namespace backendnet.Models;
```

1 reference

```
public class CustomIdentityUserPwdDTO
{
    0 references
    | public string? Id { get; set; }
    6 references
    | public required string Email { get; set; }
    1 reference
    | public required string Password { get; set; }
    2 references
    | public required string Nombre { get; set; }
    2 references
    | public required string Rol { get; set; }
}
```

10. Agregue un nuevo archivo en la carpeta **Models** llamado **UserRolDTO.cs** con el siguiente código. Este será usado para el envío del catálogo de roles de usuario.

```
namespace backendnet.Models;
```

3 references

```
public class UserRolDTO
{
    1 reference
    | public required string Id { get; set; }
    1 reference
    | public required string Nombre { get; set; }
}
```

11. Recuerde que usted puede seguir agregando más modelos si los necesita en su proyecto. Solo repita estos mismos pasos para realizarlo.

Instrucciones. Agregar datos de inicio con el ORM.

1. Para agregar datos de inicio vamos a utilizar el ORM llamado **EntityFramework**, en este caso, para trabajar con MySQL. Normalmente se acostumbra inicializar la base de datos con datos de los catálogos.
2. Su proyecto ya cuenta con código para los datos de inicio de los modelos **Categoria**, **Pelicula** y su relación **CategoriaPeliculas**.
3. Agregue un nuevo archivo llamado **SeedIdentityUserData.cs** dentro de la carpeta **Seed** con el siguiente código.

```
using backendnet.Models;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;

namespace backendnet.Data.Seed;

0 references
public static class SeedIdentityUserData
{
    0 references
    public static void SeedUserIdentityData(this ModelBuilder modelBuilder)
}
```

4. Dentro de la función **SeedUserIdentityData()** agregamos el código para crear el **rol** de **Administrador** y el rol de **Usuario**.

```
// Agregar el rol "Administrador" a la tablaAspNetRoles
string AdministradorRoleId = Guid.NewGuid().ToString();
modelBuilder.Entity<IdentityRole>().HasData(new IdentityRole
{
    Id = AdministradorRoleId,
    Name = "Administrador",
    NormalizedName = "Administrador".ToUpper()
});

// Agregar el rol "Usuario" a la tablaAspNetRoles
string UsuarioRoleId = Guid.NewGuid().ToString();
modelBuilder.Entity<IdentityRole>().HasData(new IdentityRole
{
    Id = UsuarioRoleId,
    Name = "Usuario",
    NormalizedName = "Usuario".ToUpper()
});
```

5. Debajo agregue un usuario, por ejemplo **gvera@uv.mx** con una contraseña.

```
// Agregamos un usuario a la tablaAspNetUsers
var UsuarioId = Guid.NewGuid().ToString();
modelBuilder.Entity<CustomIdentityUser>().HasData(
    new CustomIdentityUser
    {
        Id = UsuarioId, // primary key
        UserName = "gvera@uv.mx",
        Email = "gvera@uv.mx",
        NormalizedEmail = "gvera@uv.mx".ToUpper(),
        Nombre = "Guillermo Humberto Vera Amaro",
        NormalizedUserName = "gvera@uv.mx".ToUpper(),
        PasswordHash = new PasswordHasher<CustomIdentityUser>().HashPassword(null!, "patito"),
        Protegido = true // Este no se puede eliminar
    }
);
```

6. Y asigne el rol de **Administrador** a este nuevo usuario.

```
// Aplicamos la relación entre el usuario y el rol en la tablaAspNetUserRoles
modelBuilder.Entity<IdentityUserRole<string>>().HasData(
    new IdentityUserRole<string>
    {
        RoleId = AdministradorRoleId,
        UserId = UsuarioId
    }
);
```

7. Debajo agregue otro usuario, por ejemplo `patito@uv.mx` con una contraseña.

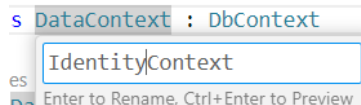
```
// Agregamos un usuario a la tablaAspNetUsers
UsuarioId = Guid.NewGuid().ToString();
modelBuilder.Entity<CustomIdentityUser>().HasData(
    new CustomIdentityUser
    {
        Id = UsuarioId, // primary key
        UserName = "patito@uv.mx",
        Email = "patito@uv.mx",
        NormalizedEmail = "patito@uv.mx".ToUpper(),
        Nombre = "Usuario patito",
        NormalizedUserName = "patito@uv.mx".ToUpper(),
        PasswordHash = new PasswordHasher<CustomIdentityUser>().HashPassword(null!, "patito")
    }
);
```

8. Y asigne el rol de `Usuario` a este nuevo usuario.

```
// Aplicamos la relación entre el usuario y el rol en la tablaAspNetUserRoles
modelBuilder.Entity<IdentityUserRole<string>>().HasData(
    new IdentityUserRole<string>
    {
        RoleId = UsuarioRoleId,
        UserId = UsuarioId
    }
);
```

9. Ahora cambie el nombre del archivo `Data/DataContext.cs` a `Data/IdentityContext.cs`.

10. Abra este archivo y modifique el nombre de la clase presionando la tecla `F2` sobre el nombre. Coloque el nombre `IdentityContext`.



11. Cambie el formato de la clase para que utilice un `Primary constructor`. Un constructor primario en C# es una forma concisa de declarar e inicializar propiedades directamente dentro de la declaración de clase. Simplifica el proceso de definición y asignación de valores a propiedades, ofreciendo una sintaxis más declarativa y legible. También cambie la clase de la que se hereda de `DbContext` a `IdentityDbContext`.

```
public class IdentityContext(DbContextOptions<IdentityContext> options) : IdentityDbContext<CustomIdentityUser>(options)
{
    7 references
    public DbSet<Pelicula> Pelicula { get; set; }
```

12. Y en la parte inferior, dentro de la función `OnModelCreating()`, agregue la llamada para crear usuarios.

13. También se debe agregar la llamada al método base `OnModelCreating()`.

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    // Inicializa la base de datos
    modelBuilder.ApplyConfiguration(new SeedCategoria());
    modelBuilder.ApplyConfiguration(new SeedPelicula());
    modelBuilder.SeedUserIdentityData();

    base.OnModelCreating(modelBuilder);
}
```

14. Finalmente verifique que en la parte superior estén los paquetes necesarios.

```
using backendnet.Data.Seed;
using backendnet.Models;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;
```

15. Un **ORM** ofrece varios beneficios, pues el mapeo de las tablas relacionales a objetos de programación, facilita la comunicación con el repositorio, simplifica las consultas, además que agrega seguridad a la aplicación.

16. Enhorabuena, la conexión con su base de datos utilizando un **ORM** esta correctamente realizada.

Instrucciones. Crear las estructuras de la base de datos.

1. Ya que tenemos listos los modelos, ahora estamos listos para generar las tablas en la base de datos ejecutando las migraciones.
2. En esta migración, se crearán los datos de prueba que agregamos de manera programática.
3. Elimine la carpeta llamada **Migrations** en la raíz de su proyecto.
4. Abra su archivo **Program.cs** y verifique que la conexión a su base de datos este habilitada.

```
// Agrega el soporte para MySQL
var connectionString = builder.Configuration.GetConnectionString("DataContext");
builder.Services.AddDbContext<IdentityContext>(options =>
{
    options.UseMySQL(connectionString, ServerVersion.AutoDetect(connectionString));
});
```

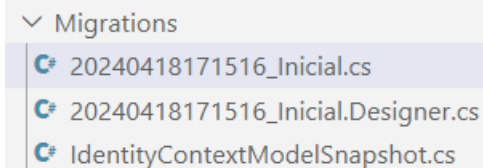
5. Ejecute el siguiente comando para actualizar la línea de comandos del ORM llamada **dotnet-ef**.

```
dotnet tool update --global dotnet-ef
```

6. Abra una terminal en Visual Studio Code y ejecute el siguiente comando.

```
dotnet ef migrations add Inicial
```

7. Observe que se crea una carpeta nueva llamada **Migrations** y dentro está el código para la creación de las tablas y datos en su base de datos.



```
▼ Migrations
  20240418171516_Inicial.cs
  20240418171516_Inicial.Designer.cs
  IdentityContextModelSnapshot.cs
```

8. Revise el código generado y reflexione sobre las instrucciones de creación de estructuras.
9. Observe que en este momento en la consola de **MySQL** no se han creado aún las tablas.

```
mysql> SHOW TABLES;
Empty set (0.00 sec)
```

10. Si deseara remover esta migración y los archivos generados, puede utilizar el comando.

```
dotnet ef migrations remove
```

11. Ahora ejecute el siguiente comando para crear sus tablas e insertar los datos de inicio.

```
dotnet ef database update
```

12. Regrese a la consola de **MySQL** y revise las tablas creadas.

```
mysql> SHOW TABLES;
+-----+
| Tables_in_netflixnet |
+-----+
| __efmigrationshistory |
| aspnetroleclaims      |
```

```
| aspNetroles |  
| aspNetuserclaims |  
| aspNetuserlogins |  
| aspNetuserroles |  
| aspNetusers |  
| aspNetusertokens |  
| categoria |  
| categoriapelicula |  
| pelicula |  
+-----+  
11 rows in set (0.00 sec)
```

13. Observe como el **ORM** genera automáticamente la tabla de enlace entre la entidad **Categoria** y la entidad **Pelicula**. Como es una relación de muchos a muchos, es necesario crear una tabla en medio en el modelo relacional. Este **ORM** llamado **EntityFramework** lo hace por usted.
14. Observe también que se han creado las tablas para el manejo de usuarios y roles de usuario.
15. También genera otra tabla de apoyo llamada **__efmigrationshistory** que le indica al **ORM** las migraciones realizadas.
16. Si quisiéramos revertir todas las migraciones aplicadas a la base de datos.

```
dotnet ef database update 0
```

17. Si se revisa la base de datos, las tablas han sido removidas.

```
mysql> SHOW TABLES;  
+-----+  
| Tables_in_netflixnet |  
+-----+  
| __efmigrationshistory |  
+-----+  
1 row in set (0.00 sec)
```

18. Y se puede ejecutar el comando para remover el código generado en la carpeta de migración.

```
dotnet ef migrations remove
```

19. En este momento no elimine las estructuras. Es solo información adicional de cómo puede revertir una migración.
20. Cada vez que realice un cambio en sus modelos, debe volver a efectuar una migración con diferente nombre para mantener sincronizado su código y su base de datos.

Instrucciones. Agregar los Controladores.

1. El proyecto ya cuenta con los controladores `CategoriasController.cs` y `PeliculasController.cs`.
2. Revise que su archivo llamado `CategoriasController.cs` en la carpeta `Controllers` cuente con el siguiente código. Realice los cambios necesarios.

```
using System.Data.Common;
using backendnet.Data;
using backendnet.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace backendnet.Controllers;

[Route("api/[controller]")]
[ApiController]
0 references
public class CategoriasController(DataContext context) : Controller
{
    // GET: api/categorias
    [HttpGet]
    0 references
    public async Task<ActionResult<IEnumerable<Categoria>>> GetCategorias() ...

    // GET: api/categorias/5
    [HttpGet("{id}")]
    1 reference
    public async Task<ActionResult<Categoria>> GetCategoria(int id) ...

    // POST: api/Categorias
    [HttpPost]
    0 references
    public async Task<ActionResult<Categoria>> PostCategoria(CategoriaDTO categoriaDTO) ...

    // PUT: api/Categorias/5
    [HttpPut("{id}")]
    0 references
    public async Task<IActionResult> PutCategoria(int id, CategoriaDTO categoriaDTO) ...

    // DELETE: api/Categorias/5
    [HttpDelete("{id}")]
    0 references
    public async Task<IActionResult> DeleteCategoria(int id) ...
}
```

3. Cada línea marcada con azul, es un método que iremos creando paso a paso.
4. Agregue el código del método `GetCategorias()`.

```
public async Task<ActionResult<IEnumerable<Categoria>>> GetCategorias()
{
    return await context.Categoria.AsNoTracking().ToListAsync();
}
```

5. Agregue el código del método `GetCategoria()`.

```
public async Task<ActionResult<Categoria>> GetCategoria(int id)
{
    var categoria = await context.Categoria.FindAsync(id);
    if (categoria == null) return NotFound();

    return categoria;
}
```

6. Agregue el código del método `PostCategoria()`.

```
public async Task<ActionResult<Categoria>> PostCategoria(CategoriaDTO categoriaDTO)
{
    Categoria categoria = new()
    {
        Nombre = categoriaDTO.Nombre
    };

    context.Categoria.Add(categoria);
    await context.SaveChangesAsync();

    return CreatedAtAction(nameof(GetCategoria), new { id = categoria.CategoriaId }, categoria);
}
```

7. Agregue el código del método `PutCategoria()`.

```
public async Task<IActionResult> PutCategoria(int id, CategoriaDTO categoriaDTO)
{
    if (id != categoriaDTO.CategoriaId) return BadRequest();

    var categoria = await context.Categoria.FindAsync(id);
    if (categoria == null) return NotFound();

    categoria.Nombre = categoriaDTO.Nombre;
    await context.SaveChangesAsync();
    return NoContent();
}
```

8. Agregue el código del método `DeleteCategoria()`.

```
public async Task<IActionResult> DeleteCategoria(int id)
{
    var categoria = await context.Categoria.FindAsync(id);
    if (categoria == null) return NotFound();

    if (categoria.Protegida) return BadRequest();

    context.Categoria.Remove(categoria);
    await context.SaveChangesAsync();

    return NoContent();
}
```

9. Revise que su archivo llamado `PeliculasController.cs` en la carpeta `Controllers` cuente con el siguiente código. Realice los cambios necesarios.

```
using backendnet.Data;
using backendnet.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace backendnet.Controllers;

[Route("api/[controller]")]
[ApiController]
0 references
public class PeliculasController(IdentityContext context) : Controller
{
    // GET: api/peliculas?s=titulo
    [HttpGet]
    0 references
    public async Task<ActionResult<IEnumerable<Pelicula>>> GetPeliculas(string? s) ...

    // GET: api/peliculas/5
    [HttpGet("{id}")]
    1 reference
    public async Task<ActionResult<Pelicula>> GetPelicula(int id) ...

    // POST: api/peliculas
    [HttpPost]
    0 references
    public async Task<ActionResult<Pelicula>> PostPelicula(PeliculaDTO peliculaDTO) ...

    // PUT: api/peliculas/5
    [HttpPut("{id}")]
    0 references
    public async Task<IActionResult> PutPelicula(int id, PeliculaDTO peliculaDTO) ...

    // DELETE: api/peliculas/5
    [HttpDelete("{id}")]
    0 references
    public async Task<IActionResult> DeletePelicula(int id) ...

    // POST: api/peliculas/5/categoria
    [HttpPost("{id}/categoria")]
    0 references
    public async Task<IActionResult> PostCategoriaPelicula(int id, AsignaCategoriaDTO itemToAdd) ...

    // DELETE: api/peliculas/5/categoria/1
    [HttpDelete("{id}/categoria/{categoriaid}")]
    0 references
    public async Task<IActionResult> DeleteCategoriaPelicula(int id, int categoriaid) ...
}
```

10. Cada línea marcada con azul, es un método que iremos creando paso a paso.
11. Agregue el código del método `GetPelículas()`.

```
public async Task<ActionResult<IEnumerable<Película>>> GetPelículas(string? s)
{
    if (string.IsNullOrEmpty(s))
        return await context.Película.Include(i => i.Categorías).AsNoTracking().ToListAsync();

    return await context.Película.Include(i => i.Categorías).Where(c => c.Título.Contains(s)).AsNoTracking().ToListAsync();
}
```

12. Agregue el código del método `GetPelícula()`.

```
public async Task<ActionResult<Película>> GetPelícula(int id)
{
    var película = await context.Película.Include(i => i.Categorías).AsNoTracking().FirstOrDefaultAsync(s => s.PelículaId == id);
    if (película == null) return NotFound();

    return película;
}
```

13. Agregue el código del método `PostPelícula()`.

```
public async Task<ActionResult<Película>> PostPelícula(PelículaDTO películaDTO)
{
    Película película = new()
    {
        Título = películaDTO.Título,
        Sinopsis = películaDTO.Sinopsis,
        Año = películaDTO.Año,
        Poster = películaDTO.Poster,
        Categorías = []
    };

    context.Película.Add(película);
    await context.SaveChangesAsync();

    return CreatedAtAction(nameof(GetPelícula), new { id = película.PelículaId }, película);
}
```

14. Agregue el código del método `PutPelícula()`.

```
public async Task<IActionResult> PutPelícula(int id, PelículaDTO películaDTO)
{
    if (id != películaDTO.PelículaId) return BadRequest();

    var película = await context.Película.FirstOrDefaultAsync(s => s.PelículaId == id);
    if (película == null) return NotFound();

    película.Título = películaDTO.Título;
    película.Sinopsis = películaDTO.Sinopsis;
    película.Año = películaDTO.Año;
    película.Poster = películaDTO.Poster;
    await context.SaveChangesAsync();

    return NoContent();
}
```

15. Agregue el código del método `DeletePelícula()`.

```
public async Task<IActionResult> DeletePelícula(int id)
{
    var pelicula = await context.Película.FindAsync(id);
    if (pelicula == null) return NotFound();

    context.Película.Remove(pelicula);
    await context.SaveChangesAsync();

    return NoContent();
}
```

16. Agregue el código del método `PostCategoriaPelícula()`.

```
public async Task<IActionResult> PostCategoriaPelícula(int id, AsignaCategoriaDTO itemToAdd)
{
    Categoria? categoria = await context.Categoria.FindAsync(itemToAdd.CategoriaId);
    if (categoria == null) return NotFound();

    var pelicula = await context.Película.Include(i => i.Categorias).FirstOrDefaultAsync(s => s.PelículaId == id);
    if (pelicula == null) return NotFound();

    if (pelicula?.Categorias?.FirstOrDefault(categoria) != null)
    {
        pelicula.Categorias.Add(categoria);
        await context.SaveChangesAsync();
    }

    return NoContent();
}
```

17. Agregue el código del método `DeleteCategoriaPelícula()`.

```
public async Task<IActionResult> DeleteCategoriaPelícula(int id, int categoriaid)
{
    Categoria? categoria = await context.Categoria.FindAsync(categoriaid);
    if (categoria == null) return NotFound();

    var pelicula = await context.Película.Include(i => i.Categorias).FirstOrDefaultAsync(s => s.PelículaId == id);
    if (pelicula == null) return NotFound();

    if (pelicula?.Categorias?.FirstOrDefault(categoria) != null)
    {
        pelicula.Categorias.Remove(categoria);
        await context.SaveChangesAsync();
    }

    return NoContent();
}
```

18. Enhorabuena, ha modificado correctamente este apartado. Vamos a agregar controladores nuevos.

19. Agregue un nuevo archivo llamado **UsuariosController.cs** en la carpeta **Controllers** con el siguiente código.

```
using backendnet.Data;
using backendnet.Models;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace backendnet.Controllers;

[Route("api/[controller]")]
[ApiController]
public class UsuariosController(IdentityContext context, UserManager<CustomIdentityUser> userManager) : Controller
{
    // GET: api/usuarios
    [HttpGet]
    public async Task<ActionResult<IEnumerable<CustomIdentityUserDTO>>> GetUsuarios() ...

    // GET: api/usuarios/email
    [HttpGet("{email}")]
    public async Task<ActionResult<CustomIdentityUserDTO>> GetUsuario(string email) ...

    // POST: api/usuarios
    [HttpPost]
    public async Task<ActionResult<CustomIdentityUserDTO>> PostUsuario(CustomIdentityUserPwdDTO usuarioDTO) ...

    // PUT: api/usuarios/email
    [HttpPut("{email}")]
    public async Task<ActionResult> PutUsuario(string email, CustomIdentityUserDTO usuarioDTO) ...

    // DELETE: api/usuarios/email
    [HttpDelete("{email}")]
    public async Task<ActionResult> DeleteUsuario(string email) ...

    private string GetUserRol(CustomIdentityUser usuario) ...
}
```

20. Cada línea marcada con azul, es un método que iremos creando paso a paso.

21. Agregue el código del método **GetUsuarios()**.

```
public async Task<ActionResult<IEnumerable<CustomIdentityUserDTO>>> GetUsuarios()
{
    var usuarios = new List<CustomIdentityUserDTO>();

    foreach (var usuario in await context.Users.AsNoTracking().ToListAsync())
    {
        usuarios.Add(new CustomIdentityUserDTO
        {
            Id = usuario.Id,
            Nombre = usuario.Nombre,
            Email = usuario.Email!,
            Rol = GetUserRol(usuario)
        });
    }
    return usuarios;
}
```

22. Agregue el código del método **GetUsuario()**.


```
public async Task<ActionResult<CustomIdentityUserDTO>> GetUsuario(string email)
{
    var usuario = await userManager.FindByEmailAsync(email);

    if (usuario == null) return NotFound();

    return new CustomIdentityUserDTO
    {
        Id = usuario.Id,
        Nombre = usuario.Nombre,
        Email = usuario.Email!,
        Rol = GetUserRol(usuario)
    };
}
```

23. Agregue el código del método `PostUsuario()`.

```
public async Task<ActionResult<CustomIdentityUserDTO>> PostUsuario(CustomIdentityUserPwdDTO usuarioDTO)
{
    var usuarioToCreate = new CustomIdentityUser
    {
        UserName = usuarioDTO.Email,
        Email = usuarioDTO.Email,
        NormalizedEmail = usuarioDTO.Email.ToUpper(),
        Nombre = usuarioDTO.Nombre,
        NormalizedUserName = usuarioDTO.Email.ToUpper()
    };

    // Agrega al usuario
    IdentityResult result = await userManager.CreateAsync(usuarioToCreate, usuarioDTO.Password);
    if (!result.Succeeded) return BadRequest(new { mensaje = "El usuario no se ha podido crear." });

    // Lo agrega al Rol deseado
    result = await userManager.AddToRoleAsync(usuarioToCreate, usuarioDTO.Rol);

    // Regresa el usuario creado
    var usuarioViewModel = new CustomIdentityUserDTO
    {
        Id = usuarioToCreate.Id,
        Nombre = usuarioDTO.Nombre,
        Email = usuarioDTO.Email,
        Rol = usuarioDTO.Rol
    };
    return CreatedAtAction(nameof(GetUsuario), new { email = usuarioDTO.Email }, usuarioViewModel);
}
```

24. Agregue el código del método `PutUsuario()`.

```
public async Task<IActionResult> PutUsuario(string email, CustomIdentityUserDTO usuarioDTO)
{
    if (email != usuarioDTO.Email) return BadRequest();

    var usuario = await userManager.FindByEmailAsync(email);
    if (usuario == null) return NotFound();

    // Verifica que exista el rol recibido
    if (await context.Roles.Where(r => r.Name == usuarioDTO.Rol).FirstOrDefaultAsync() == null) return NotFound();

    // Actualiza los datos
    usuario.Nombre = usuarioDTO.Nombre;
    usuario.NormalizedUserName = usuarioDTO.Email.ToUpper();
    IdentityResult result = await userManager.UpdateAsync(usuario);
    if (!result.Succeeded) return BadRequest();

    // Actualiza el rol seleccionado
    foreach (var rol in await context.Roles.ToListAsync())
    {
        await userManager.RemoveFromRoleAsync(usuario, rol.Name!);
    }
    await userManager.AddToRoleAsync(usuario, usuarioDTO.Rol);

    return NoContent();
}
```

25. Agregue el código del método `DeleteUsuario()`.

```
public async Task<IActionResult> DeleteUsuario(string email)
{
    var usuario = await userManager.FindByEmailAsync(email);
    if (usuario == null) return NotFound();

    if (usuario.Protegido) return StatusCode(StatusCodes.Status403Forbidden);

    IdentityResult result = await userManager.DeleteAsync(usuario);
    if (!result.Succeeded) return BadRequest();

    return NoContent();
}
```

26. Agregue el código del método `GetUserRol()`.

```
private string GetUserRol(CustomIdentityUser usuario)
{
    var roles = userManager.GetRolesAsync(usuario).Result;
    return roles.First();
}
```

27. Agregue un nuevo archivo llamado `RolesController.cs` en la carpeta `Controllers` con el siguiente código.

```
using backendnet.Data;
using backendnet.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace backendnet.Controllers;

[Route("api/[controller]")]
[ApiController]
0 references
public class RolesController(IdentityContext context) : Controller
{
    // GET: api/roles
    [HttpGet]
    0 references
    public async Task<ActionResult<IEnumerable<UserRolDTO>>> GetRoles() ...
}
```

28. Cada línea marcada con azul, es un método que iremos creando paso a paso.

29. Agregue el código del método `GetRoles()`.

```
public async Task<ActionResult<IEnumerable<UserRolDTO>>> GetRoles()
{
    var roles = new List<UserRolDTO>();

    foreach (var rol in await context.Roles.AsNoTracking().ToListAsync())
    {
        roles.Add(new UserRolDTO
        {
            Id = rol.Id,
            Nombre = rol.Name!
        });
    }
    return roles;
}
```

30. Enhorabuena, ha creado sus controladores de manera correcta.

Instrucciones. Agregar middlewares.

1. Agregue un controlador para el manejo de errores. Agregue un nuevo archivo llamado `ErrorController.cs` en la carpeta `Controllers` con el siguiente código.

```
using Microsoft.AspNetCore.Diagnostics;  
using Microsoft.AspNetCore.Mvc;  
  
namespace backendnet.Controllers;  
  
[ApiController]  
[ApiExplorerSettings(IgnoreApi = true)]  
public class ErrorController() : Controller  
{  
    [Route("/error")]  
    public IActionResult HandleErrorDevelopment([FromServices] IHostEnvironment hostEnvironment)  
    {  
        if (!hostEnvironment.IsDevelopment())  
        {  
            return BadRequest(new { mensaje = "No se ha podido procesar la petición. Inténtelo nuevamente más tarde." });  
        }  
  
        var exceptionHandlerFeature = HttpContext.Features.Get<ExceptionHandlerFeature>(!);  
  
        return Problem(  
            detail: exceptionHandlerFeature.Error.StackTrace,  
            title: exceptionHandlerFeature.Error.Message);  
    }  
}
```

2. Y en el `Program.cs` de la raíz, agregue la llamada a este controlador en forma de middleware justo arriba de `app.UseRouting()`.

```
// Agregamos un middleware para el manejo de errores  
app.UseExceptionHandler("/error");  
  
// Utiliza rutas para los endpoints de los controladores  
app.UseRouting();
```

3. Más adelante agregaremos más middlewares en lo referente a la autenticación y autorización de usuarios.

Instrucciones. Configurar el programa principal.

1. Abra su archivo `Program.cs` y agregue el siguiente código en la parte superior.

```
using Microsoft.EntityFrameworkCore;  
using backendnet.Data;  
using backendnet.Models;  
using Microsoft.AspNetCore.Identity;
```

2. Debajo del soporte para su conexión a base de datos y antes del soporte de `Cors`, agregue el soporte para el uso de `IdentityFramework` e ingrese las condiciones para la creación de contraseñas.

```
// Soporte para Identity  
builder.Services.AddIdentity<CustomIdentityUser, IdentityRole>(options =>  
{  
    options.User.RequireUniqueEmail = true;  
    // Cambie aqui como quiere se manejen sus contraseñas  
    options.Password.RequireDigit = false;  
    options.Password.RequireLowercase = false;  
    options.Password.RequireNonAlphanumeric = false;  
    options.Password.RequireUppercase = false;  
    options.Password.RequiredLength = 6;  
    options.Password.RequiredUniqueChars = 1;  
})  
.AddEntityFrameworkStores<IdentityContext>();
```

3. Observe que todo este bloque se encarga de registrar los servicios que la aplicación utilizará y construirla.
4. Lea los comentarios que se proveen, le explican el funcionamiento de cada instrucción.
5. En la parte inferior después de `var app = builder.Build();` se agregan las configuraciones y funcionalidades de los servicios.
6. Recuerde que la última instrucción, ejecuta su aplicación.
7. Enhorabuena, su aplicación esta lista para trabajar con usuarios.

Instrucciones. Probar el funcionamiento.

1. En la consola escriba `dotnet build`. Y verifique que no marque errores.
2. Verifique que no marque errores.
3. Si no aparece ningún error, al parecer su API funciona correctamente.
4. Verifique el puerto en el que se ejecutará su aplicación abriendo el archivo `/Properties/launchSettings.json`.

```
"http": {  
  "commandName": "Project",  
  "dotnetRunMessages": true,  
  "launchBrowser": true,  
  "applicationUrl": "http://localhost:3000",  
  "environmentVariables": {  
    "ASPNETCORE_ENVIRONMENT": "Development"  
  }  
}
```

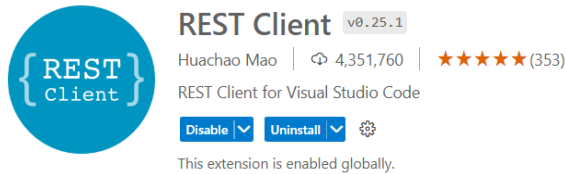
5. Ejecute su aplicación usando `dotnet run` o `dotnet watch run` y una ventana de su navegador se abrirá.

```
PS C:\codigo\tw\backendnet> dotnet run  
Compilando...  
info: Microsoft.Hosting.Lifetime[14]  
      Now listening on: http://localhost:3000  
info: Microsoft.Hosting.Lifetime[0]  
      Application started. Press Ctrl+C to shut down.  
info: Microsoft.Hosting.Lifetime[0]  
      Hosting environment: Development  
info: Microsoft.Hosting.Lifetime[0]  
      Content root path: C:\codigo\tw\backendnet
```

6. Observe que en este momento cualquier cliente web podría acceder a sus datos.
7. Generalmente el acceso a su API debe protegerse por algún mecanismo de seguridad. Eso lo realizaremos después de probar que el controlador de usuarios funcione correctamente.

Instrucciones. Probar la API

1. Utilizando su navegador web no es la forma más práctica de probar una API. Generalmente se utiliza otro tipo de clientes web para hacer las pruebas de manera más eficiente.
2. Asegúrese de tener instalada la extensión **REST Client** para Visual Studio Code.



3. Agregue una nueva carpeta llamada **RestClient** y dentro un nuevo archivo llamado **usuarios.http** en la raíz de su proyecto con el siguiente código.

Send Request

GET <http://localhost:3000/api/usuarios>

###

Send Request

GET <http://localhost:3000/api/usuarios/gvera@uv.mx>

###

Send Request

POST <http://localhost:3000/api/usuarios>
Content-Type: application/json

```
{
  "email": "prueba@uv.mx",
  "password": "patito",
  "nombre": "Un nuevo usuario pato",
  "rol": "Administrador"
}
```

###

Send Request

PUT <http://localhost:3000/api/usuarios/prueba@uv.mx>
Content-Type: application/json

```
{
  "email": "prueba@uv.mx",
  "nombre": "Usuario pato editado",
  "rol": "Usuario"
}
```

###

Send Request

DELETE <http://localhost:3000/api/usuarios/prueba@uv.mx>

4. Respete los saltos de línea. Observe como hemos creado varias peticiones a nuestra API.

- Observe como aparece el botón de **Send Request** encima de cada solicitud.
- Comience con la primer prueba, presione el botón de **Send Request** para obtener la lista de usuarios.

```
ent > usuarios.http > ...
Send Request
GET http://localhost:3000/api/usuarios

###

Send Request
GET http://localhost:3000/api/usuarios/gvera@uv.mx

###

Send Request
POST http://localhost:3000/api/usuarios
Content-Type: application/json

{
  "email": "prueba@uv.mx",
  "password": "patito",
  "nombre": "Un nuevo usuario pato",
  "rol": "Administrador"
}

###

1 HTTP/1.1 200 OK
2 Connection: close
3 Content-Type: application/json; charset=utf-8
4 Date: Fri, 19 Apr 2024 17:34:29 GMT
5 Server: Kestrel
6 Transfer-Encoding: chunked
7
8 √ [
9 √ {
10   "id": "4e7f2d1d-6690-4018-bfa4-c2d63c9f2e40",
11   "email": "patito@uv.mx",
12   "nombre": "Usuario patito",
13   "rol": "Administrador"
14 },
15 √ {
16   "id": "dbb904be-009c-4c71-aeff-de16fe065b4d",
17   "email": "gvera@uv.mx",
18   "nombre": "Guillermo Humberto Vera Amaro",
19   "rol": "Administrador"
20 }
21 ]
```

- Del lado derecho aparecen los resultados de la consulta. Verifique que se obtienen correctamente su lista de elementos.
- Pruebe que puede agregar un nuevo usuario utilizando el método **POST**.

```
ent > usuarios.http > ...
Send Request
POST http://localhost:3000/api/usuarios
Content-Type: application/json

{
  "email": "prueba@uv.mx",
  "password": "patito",
  "nombre": "Un nuevo usuario pato",
  "rol": "Administrador"
}

###

Send Request
PUT http://localhost:3000/api/usuarios/prueba@uv.mx

1 HTTP/1.1 201 Created
2 Connection: close
3 Content-Type: application/json; charset=utf-8
4 Date: Fri, 19 Apr 2024 17:34:51 GMT
5 Server: Kestrel
6 Location: http://localhost:3000/api/Usuarios/prueba@uv.mx
7 Transfer-Encoding: chunked
8
9 √ {
10   "id": "0617d6ff-dcb3-41ee-8007-5c068dca6db3",
11   "email": "prueba@uv.mx",
12   "nombre": "Un nuevo usuario pato",
13   "rol": "Administrador"
14 }
```

- Intente actualizar el usuario cambiando su rol de **Administrador** a **Usuario**.

```
ent > usuarios.http > ...
Send Request
PUT http://localhost:3000/api/usuarios/prueba@uv.mx
Content-Type: application/json

{
  "email": "prueba@uv.mx",
  "nombre": "Usuario pato editado",
  "rol": "Usuario"
}

1 HTTP/1.1 204 No Content
2 Connection: close
3 Date: Fri, 19 Apr 2024 17:35:09 GMT
4 Server: Kestrel
5
6
```


10. Finalmente, intente eliminar el usuario que se acaba de crear.

```
nt > usuarios.http > ...  
Send Request  
DELETE http://localhost:3000/api/usuarios/prueba@uv.mx  
1 HTTP/1.1 204 No Content  
2 Connection: close  
3 Date: Fri, 19 Apr 2024 17:35:27 GMT  
4 Server: Kestrel
```

11. Si intenta eliminar a un usuario protegido y debería obtener un mensaje de **Prohibido**.

```
it > usuarios.http > ...  
Send Request  
DELETE http://localhost:3000/api/usuarios/gvera@uv.mx  
1 HTTP/1.1 403 Forbidden  
2 Connection: close  
3 Content-Type: application/problem+json; charset=utf-8  
4 Date: Fri, 19 Apr 2024 17:36:20 GMT  
5 Server: Kestrel  
6 Transfer-Encoding: chunked  
7  
8 {  
9   "type": "https://tools.ietf.org/html/rfc9110#section-15.5.4",  
10  "title": "Forbidden",  
11  "status": 403,  
12  "traceId": "00-928625c8a5c77c0d8307f549b9f51051-80bfe69327bf8dc:  
13 }
```

12. Ahora que todo funciona correctamente, es el momento de agregar seguridad a su aplicación.

Instrucciones. Agregar autenticación y autorización a la API

JSON Web Token ayuda a proteger una ruta de un usuario no autenticado. El uso de JWT en la aplicación evitará que los usuarios no autenticados accedan a **endpoints** protegidos.

JWT crea un token, lo envía al cliente y, a continuación, el cliente utiliza el token para realizar solicitudes. También ayuda a verificar que sea un usuario válido que realiza esas solicitudes.

1. Se tiene que instalar el soporte para JWT antes de usarlo en la aplicación.

```
dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer
```

2. Ahora se tiene que generar una llave privada llamada **secret** para poder generar los JWT.
3. Desde la línea de comandos genere una cadena aleatoria de 35 bytes en hexadecimal. Entre a la línea de comandos de **node**.

```
node
Welcome to Node.js v20.11.0.
Type ".help" for more information.
>
```

4. Y luego escriba este comando para generar la cadena.

```
> require("crypto").randomBytes(35).toString("hex")
```

```
PS C:\codigo\tw\backendnetauth> node
Welcome to Node.js v20.11.0.
Type ".help" for more information.
> require("crypto").randomBytes(35).toString("hex")
'865ad24cf96fa78ab5c3cdb48f5fb114923d95c8a4e4f1bc9c33f6eeda32ed9d4b8c57'
```

5. Para salir de la línea de comandos de **node** presione dos veces **Ctrl + C**.
6. Abra su archivo **appsettings.json** y copie y pegue esta cadena creando el nuevo apartado de **Jwt**.

```
{
  "DataContext": "Server=localhost;User ID=netflix_user;Password=N3tf1x;Database=net",
  "Jwt": {
    "Issuer": "ServidorFeiJWT",
    "Audience": "ClientesFeiJWT",
    "Secret": "865ad24cf96fa78ab5c3cdb48f5fb114923d95c8a4e4f1bc9c33f6eeda32ed9d4b8c57"
  }
}
```

7. No comparta esta cadena secreta. Si se filtra esta cadena secreta, los usuarios no autenticados pueden crear tokens falsos para acceder a las rutas.
8. Continuemos con el inicio de sesión.

Instrucciones. Agregar servicios.

1. Sus clientes van a requerir tokens de acceso, así que tenemos que crear un servicio que pueda generarlos.
2. Agregue un nuevo archivo llamado `JwtTokenService.cs` en la carpeta `Services` con el siguiente código.

```
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;
using Microsoft.IdentityModel.Tokens;

namespace backendnet.Services;

public class JwtTokenService(IConfiguration configuration, IHttpContextAccessor httpContextAccessor)
{
    public string GeneraToken(List<Claim> claims)
    {
        var securityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(configuration["Jwt:Secret"]!));
        var credentials = new SigningCredentials(securityKey, SecurityAlgorithms.HmacSha256Signature);
        var tokenDescriptor = new JwtSecurityToken(
            issuer: configuration["Jwt:Issuer"],
            audience: configuration["Jwt:Audience"],
            claims: claims,
            expires: DateTime.Now.AddMinutes(20),
            signingCredentials: credentials);

        var jwt = new JwtSecurityTokenHandler().WriteToken(tokenDescriptor);

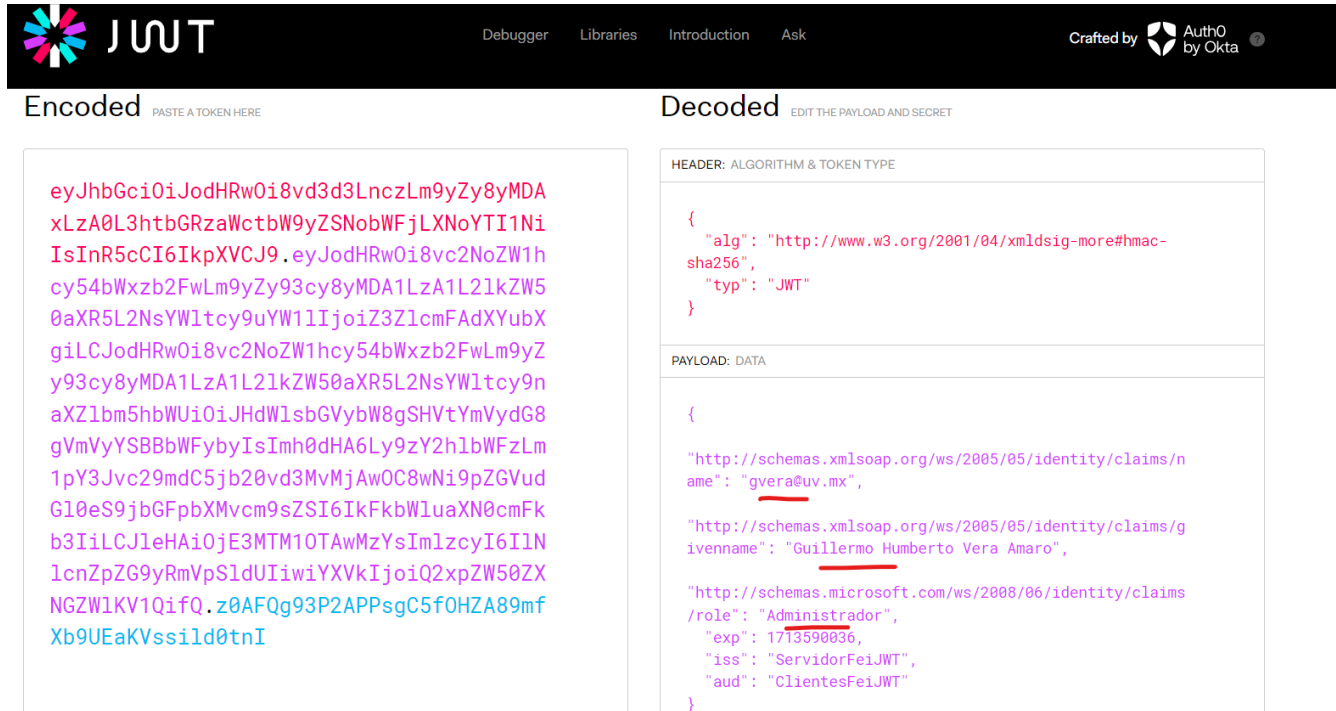
        return jwt;
    }
}
```

3. Observe que se crea el token mediante la función `JwtSecurityToken()`. Esta función toma tres parámetros:
 - El `payload` son los primeros tres parámetros que pasará a la función. Esta carga útil contiene datos relacionados con el usuario, y estos datos no deben contener información confidencial como contraseñas.
 - El `jwtSecret` como último parámetro.
 - Cuánto `tiempo` durará el token como penúltimo parámetro.
4. Después de pasar todos estos argumentos, JWT generará un token. Una vez generado el token, se envía al cliente.
5. Este token deberá agregarse en cada petición que se haga para obtener el acceso a los métodos protegidos de la API. Por ejemplo, si se genera el token:

eyJhbGciOiJodHRwOi8vd3d3LnczLm9yZy8yMDAxLzA0L3htbGRzZWVudW9yZSNOblFjLXNoYTI1NiIsInR5cCI6IkpvcXVCJ9.eyJodHRwOi8vc2NoZW1hcy54bWxzbnZlLnM9Zy93cy8yMDAxLzA1L2lkZW50aXR5L2NsYWltcy9uYW1lIjoizDZlcmFAdXYubXglCJodHRwOi8vc2NoZW1hcy54bWxzbnZlLnM9Zy93cy8yMDAxLzA1L2lkZW50aXR5L2NsYWltcy9naXZlbm5hbWUiOiJHdWlsbGVybW8gSHVtVmVydG8gVmVvYSBBbnFybyIsImh0dHA6Ly9yZ2hlbmFzLm1pY3Jvc29mdC5jb20vd3MvMjAwOC8wNi9pZGVudGl0eS9jbGF

```
pbXMvcm9sZSI6IkFkbWluaXN0cmFkb3IiLCJleHAiOjE3MTM1OTAwMzYsImZlcyI6IiNlcnZpZG9yRmVpSldUIiwiYXVkJoiQ2xpZW50ZXNGZWlKV1QifQ.z0AFQg93P2APPsgC5f0HZA89mfXb9UEaKVssild0tnI
```

6. Podemos validar que sea un **JWT** válido. Entre a la página <https://jwt.io/> y pegue este token para ver sus propiedades.



The screenshot shows the JWT.io website interface. The 'Encoded' section on the left contains the token: `eyJhbGciOiJIodHRwOi8vd3d3LnczLm9yZy8yMDAxLzA0L3htbGRzaWctbW9yZSNOYWJjLXNoYTI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm9yZy93cy8yMDA1LzA1L2lkZW50aXR5L2NsYW1tcy9uYW11IjoiZ3Z1cmFAdXYubXgilCJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm9yZy93cy8yMDA1LzA1L2lkZW50aXR5L2NsYW1tcy9naXZ1bm5hbWUiOiJHdWlsbGVyYyBw8gSHVtYmVydG8gVmVyYSBBbWFybyIsImh0dHA6Ly9yZ2h1bWZlM1pY3Jvc29mdC5jb20vd3d3LmVjAwOC8wNi9pZGVudG10eS9jbGFpbXMvcm9sZSI6IkFkbWluaXN0cmFkb3IiLCJleHAiOjE3MTM1OTAwMzYsImZlcyI6IiNlcnZpZG9yRmVpSldUIiwiYXVkJoiQ2xpZW50ZXNGZWlKV1QifQ.z0AFQg93P2APPsgC5f0HZA89mfXb9UEaKVssild0tnI`. The 'Decoded' section on the right shows the token's structure:

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "http://www.w3.org/2001/04/xmldsig-more#hmac-sha256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name": "gvera@uv.mx",
  "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname": "Guillermo Humberto Vera Amaro",
  "http://schemas.microsoft.com/ws/2008/06/identity/claims/role": "Administrador",
  "exp": 1713590036,
  "iss": "ServidorFeiJWT",
  "aud": "ClientesFeiJWT"
}
```

7. El token es válido y observe que puede ver el usuario que tiene el acceso. Solo con esa cadena token, usted puede tener acceso al sistema por 20 minutos. Es por ello que **JWT** no se considera un método de acceso tan estricto, pues si este token es robado, cualquiera podría acceder a los recursos.
8. Sus clientes deberán entrar a la ruta `/api/auth/` para iniciar sesión en la API y obtener un token válido.
9. Debajo de esta función, agregue el cálculo del tiempo restante del token generado.

```
public string? TiempoRestanteToken()
{
    string authorization = HttpContextAccessor.HttpContext?.Request.Headers.Authorization!;
    if (string.IsNullOrEmpty(authorization) || !authorization.StartsWith("Bearer"))
        return null;

    JwtSecurityToken token = new JwtSecurityTokenHandler().ReadJwtToken(authorization[7..]);

    return token?.ValidTo.Subtract(DateTime.UtcNow).ToString(@"hh\:mm\:ss");
}
```

10. Enhorabuena, ha realizado correctamente este apartado.

Instrucciones. Agregar middlewares.

Sus clientes van a requerir tokens de acceso no expiren mientras los están utilizando. Así que agregue la funcionalidad para refrescar la validez del token si el usuario haciendo peticiones al sistema.

1. Agregue un nuevo archivo llamado `SlidingExpirationJwt.cs` en la carpeta `Middlewares` con el siguiente código.

```
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using backendnet.Services;

namespace backendnet.Middlewares;

1 reference
public class SlidingExpirationJwt(RequestDelegate next)
{
    0 references
    public async Task InvokeAsync(HttpContext context, JwtTokenService jwtTokenService) ...
}

// Esta clase es para poder agregarlo en Program.cs
0 references
public static class SlidingExpirationJwtExtensions
{
    1 reference
    public static IApplicationBuilder UseSlidingExpirationJwt(this IApplicationBuilder builder)
    {
        return builder.UseMiddleware<SlidingExpirationJwt>();
    }
}
```

2. Agregue el código dentro del método `InvokeAsync()`.

```
public async Task InvokeAsync(HttpContext context, JwtTokenService jwtTokenService)
{
    try
    {
        string authorization = context.Request.Headers.Authorization!;
        JwtSecurityToken? token = null;
        if (!string.IsNullOrEmpty(authorization) && authorization.StartsWith("Bearer"))
            token = new JwtSecurityTokenHandler().ReadJwtToken(authorization[7..]);

        if (token != null && token.ValidTo > DateTime.UtcNow)
        {
            TimeSpan timeRemaining = token.ValidTo.Subtract(DateTime.UtcNow);
            //Si quedan 5 minutos, le mandamos un nuevo token
            if (timeRemaining.Minutes < 5)
            {
                var claims = new List<Claim>
                {
                    new(ClaimTypes.Name, context.User.FindFirstValue(ClaimTypes.Name)!),
                    new(ClaimTypes.GivenName, context.User.FindFirstValue(ClaimTypes.GivenName)!),
                    new(ClaimTypes.Role, context.User.FindFirstValue(ClaimTypes.Role)!);
                };
                context.Response.Headers.Append("Set-Authorization", jwtTokenService.GeneraToken(claims));
            }
        }
    }
    catch (Exception)
    {
        // Ocurrió un fallo al revisar el token
    }
    await next(context);
}
```

3. Enhorabuena, ha realizado correctamente el apartado.

Instrucciones. Agregar el inicio de sesión: Autenticación.

1. Agregue un nuevo archivo en la carpeta **Models** llamado **LoginDTO.cs** con el siguiente código.

```
namespace backendnet.Models;

0 references
public class LoginDTO
{
    0 references
    public required string Email { get; set; }
    0 references
    public required string Password { get; set; }
}
```

2. Utilizaremos este modelo para recibir los datos de inicio de sesión de los clientes.
3. En la carpeta **Controllers**, cree un nuevo archivo llamado **AuthController.cs** y agregue el método para realizar el **login** de los usuarios.

```
using System.Security.Claims;
using backendnet.Models;
using backendnet.Services;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;

namespace backendnet.Controllers;

[Route("api/[controller]")]
[ApiController]
0 references
public class AuthController(UserManager<CustomIdentityUser> userManager, JwtTokenService jwtTokenService) : Controller
{
    // POST api/auth
    [HttpPost]
    0 references
    public async Task<IActionResult> PostAsync([FromBody] LoginDTO loginDTO) ...
}
```

4. Dentro del método **PostAsync()** agregue la verificación de usuario válido en la base de datos.

```
// Verificamos credenciales con Identity
var usuario = await userManager.FindByEmailAsync(loginDTO.Email);

if (usuario is null || !await userManager.CheckPasswordAsync(usuario, loginDTO.Password))
{
    // Regresa 401 Acceso no autorizado
    return Unauthorized(new { mensaje = "Usuario o contraseña incorrectos." });
}
```

5. Debajo agregue la información del usuario que llevará el token.

```
// Estos valores nos indicarán el usuario autenticado en cada petición usando el token
var roles = await userManager.GetRolesAsync(usuario);
var claims = new List<Claim>
{
    new(ClaimTypes.Name, usuario.Email!),
    new(ClaimTypes.GivenName, usuario.Nombre),
    new(ClaimTypes.Role, roles.First()),
};
```

6. Observe que es obligatorio colocar el `ClaimTypes.Name` y `ClaimTypes.Role` para que .NET sepa en cada petición, el nombre del usuario y su rol asignado. Cada uno es basado en un esquema estándar.

ClaimTypes.Name	http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name
ClaimTypes.GivenName	http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname
ClaimTypes.Role	http://schemas.microsoft.com/ws/2008/06/identity/claims/role

7. Debajo agregue la creación y envío del JWT.

```
// Creamos el token de acceso
var jwt = jwtTokenService.GeneraToken(claims);

// Le regresa su token de acceso al usuario con validez de 20 minutos
return Ok(new
{
    usuario.Email,
    usuario.Nombre,
    rol = string.Join(",", roles),
    jwt
});
```

8. Ahora debajo, agregue una función para enviar al cliente el tiempo restante del token generado.

```
// GET: api/auth/tiempo
[Authorize]
[HttpGet("tiempo")]
0 references
public IActionResult GetTiempo()
{
    string? tiempo = jwtTokenService.TiempoRestanteToken();
    if (tiempo is null)
        return BadRequest();
    return Ok(tiempo);
}
```

9. Enhorabuena, es hora de continuar protegiendo los recursos; es decir, hacer obligatorio el envío de estos tokens para poder consultar los `endpoints` de la aplicación.

Instrucciones. Proteger las rutas privadas: Autorización.

1. Ahora se tienen que proteger todos los métodos de los controladores que van a requerir autenticación para ser accedidos. Además agregaremos la autorización por roles. Para esto se utilizará un `middleware` integrado de .NET en formato de `FilterAttribute` que verifique la validez del token enviado por el cliente y también se verificará si el rol enviado está autorizado para realizar la acción solicitada.
2. Hay que agregar este `Attribute` para cada ruta o controlador que la aplicación requiera proteger.
3. Por ejemplo, abra su archivo de `CategoriasController.cs`. Agregue en la parte superior la dependencia al middleware.

```
[Route("api/[controller]")]
[ApiController]
[Authorize(Roles = "Administrador")]
1 reference
public class CategoriasController : Controller
```

4. Haga lo mismo para los controladores `UsuariosController.cs` y `RolesController.cs`.
5. Si desea que dos o más roles pudieran acceder a ese `endpoint`, solo agregue el nombre del rol separado por comas en el método que implementa el `endpoint`.
6. Por ejemplo, en el controlador `PeliculasController.cs` no agregue el atributo de `[Authorize]` en la parte superior de la clase. Agréguelo directamente en los métodos con GET que pueden accederse utilizando el rol de `Administrador` y el rol de `Usuario`.

```
// GET: api/peliculas?s=titulo
[HttpGet]
[Authorize(Roles = "Usuario,Administrador")]
0 references
public async Task<ActionResult<IEnumerable<Pelicula>>> GetPeliculas(string? s)
```

```
// GET: api/peliculas/5
[HttpGet("{id}")]
[Authorize(Roles = "Usuario,Administrador")]
1 reference
public async Task<ActionResult<Pelicula>> GetPelicula(int id)
```

7. Y en los otros métodos de `PeliculasController.cs`, agregue el atributo solo para `Administrador`.

```
// POST: api/peliculas
[HttpPost]
[Authorize(Roles = "Administrador")]
0 references
public async Task<ActionResult<Pelicula>> PostPelicula(PeliculaDTO peliculaDTO)
```

8. Haga lo mismo para cualquier otra ruta que requiera protección en su proyecto.
9. Observe que en el caso del controlador `AuthController.cs` no colocaremos autorización pues deseamos permitir peticiones de usuarios anónimos que puedan obtener su token de autenticación.
10. Felicidades, ha asegurado su aplicación de manera correcta.

Instrucciones. Configurar el programa principal.

1. Abra su archivo `Program.cs` y verifique las dependencias a utilizar en el archivo.

```
using Microsoft.EntityFrameworkCore;
using backendnet.Data;
using backendnet.Models;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.IdentityModel.Tokens;
using System.Text;
using backendnet.Middleware;
using backendnet.Services;
```

2. Debajo de la creación el `builder` agregue el servicio que acabamos de crear.

```
// Soporte para generar JWT
builder.Services.AddScoped<JwtTokenService>();
```

3. Debajo del soporte de `Identity`, agregue el soporte para utilizar JWT como método de autenticación.

```
// Soporte para JWT
builder.Services
    .AddHttpContextAccessor() // Para poder acceder al HttpContext()
    .AddAuthorization() // Para autorizar en cada método el acceso
    .AddAuthentication(options =>
    {
        options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
        options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
    })
    .AddJwtBearer(options => // Para autenticar con JWT
    {
        options.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuer = true,
            ValidateAudience = true,
            ValidateLifetime = true,
            ValidateIssuerSigningKey = true,
            ValidIssuer = builder.Configuration["Jwt:Issuer"], // Leído desde appSettings
            ValidAudience = builder.Configuration["Jwt:Audience"], // Leído desde appSettings
            IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(builder.Configuration["Jwt:Secret"]!))
        };
    });
```

4. En este mismo archivo, en la parte inferior debajo de `app.UseRouting()` agregue la autorización, autenticación y el refresco de tokens a su proyecto. El orden es importante, no lo cambie.

```
// Utiliza rutas para los endpoints de los controladores
app.UseRouting();

// Utiliza Autenticacion
app.UseAuthentication();
// Utiliza Autorizacion
app.UseAuthorization();
// Agrega el middleware para refrescar el token
app.UseSlidingExpirationJwt();
```

5. Su programa principal debe quedar de la siguiente manera:

```
using Microsoft.EntityFrameworkCore;
using backendnet.Data;
using backendnet.Models;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.IdentityModel.Tokens;
using System.Text;
using backendnet.Middleware;
using backendnet.Services;

var builder = WebApplication.CreateBuilder(args);

// Soporte para generar JWT
builder.Services.AddScoped<JwtTokenService>();

// Agrega el soporte para MySQL
var connectionString = builder.Configuration.GetConnectionString("DataContext");
builder.Services.AddDbContext<IdentityContext>(options =>
{
    options.UseMySQL(connectionString, ServerVersion.AutoDetect(connectionString));
});

// Soporte para Identity
builder.Services.AddIdentity<CustomIdentityUser, IdentityRole>(options =>
{
    options.User.RequireUniqueEmail = true;
    // Cambie aqui como quiere se manejen sus contraseñas
    options.Password.RequireDigit = false;
    options.Password.RequireLowercase = false;
    options.Password.RequireNonAlphanumeric = false;
    options.Password.RequireUppercase = false;
    options.Password.RequiredLength = 6;
    options.Password.RequiredUniqueChars = 1;
})
.AddEntityFrameworkStores<IdentityContext>();

// Soporte para JWT
builder.Services
    .AddHttpContextAccessor() // Para poder acceder al HttpContext()
    .AddAuthorization() // Para autorizar en cada método el acceso
    .AddAuthentication(options =>
    {
        options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
        options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
    })
    .AddJwtBearer(options => // Para autenticar con JWT
    {
        options.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuer = true,
            ValidateAudience = true,
            ValidateLifetime = true,
            ValidateIssuerSigningKey = true,
            ValidIssuer = builder.Configuration["Jwt:Issuer"], // Leido desde appSettings
            ValidAudience = builder.Configuration["Jwt:Audience"], // Leido desde appSettings
            IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(builder.Configuration["Jwt:Secret"]!))
        };
    });

// Agrega el soporte para CORS
builder.Services.AddCors(options =>
{
    options.AddDefaultPolicy(
        policy =>
        {
            policy.WithOrigins("http://localhost:3001", "http://localhost:8080")
                .AllowAnyHeader()
                .WithMethods("GET", "POST", "PUT", "DELETE");
        });
});

// Agrega la funcionalidad de controladores
builder.Services.AddControllers();
// Agrega la documentación de la API
builder.Services.AddSwaggerGen();

// Construye la aplicación web
var app = builder.Build();
```

```
// Mostraremos la documentación solo en ambiente de desarrollo
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

// Agregamos un middleware para el manejo de errores
app.UseExceptionHandler("/error");

// Utiliza rutas para los endpoints de los controladores
app.UseRouting();

// Utiliza Autenticacion
app.UseAuthentication();
// Utiliza Autorizacion
app.UseAuthorization();
// Agrega el middleware para refrescar el token
app.UseSlidingExpirationJwt();

// Usa Cors con la policy definida anteriormente
app.UseCors();
// Establece el uso de rutas sin especificar una por default
app.MapControllers();

app.Run();
```

6. Enhorabuena, compile su proyecto y verifique que no marque errores.

```

1 HTTP/1.1 200 OK
2 Connection: close
3 Content-Type: application/json; charset=utf-8
4 Date: Fri, 19 Apr 2024 21:47:50 GMT
5 Server: Kestrel
6 Transfer-Encoding: chunked
7
8 {
9   "email": "gvera@uv.mx",
10  "nombre": "Guillermo Humberto Vera Amaro",
11  "rol": "Administrador",
12  "jwt": "eyJhbGciOiJIodHRwOi5vd3d3LnczLm9yZy8yMDAxLzA0L3htbGRzaWctbW9yZSNo
bWFlXjXN0eTI1NiIsInR5cCI6IkpXVCJ9.ejJodHRwOi5vc2NoZW1hcy45bWxzZ2FwLm9yZy93c
y8yMDAxLzA1L2lkZW50aXR5L2NsYWltcy9wYWI1Ijoiz3ZlcmFAdXVubXgiLjodHRwOi5vc2No
ZW1hcy45bWxzZ2FwLm9yZy93cy8yMDAxLzA1L2lkZW50aXR5L2NsYWltcy9wYWI1bm5hbWUo
i2hWdl5bcG9ybm9wSjVtVmYydyG8G9VmVvYSBBbW9yYyIsImh0dHA6Ly9yZ2h1bWZlLm1pY3Jvc29
mdC5jb2V0d3MwMjAwOC8wNi9pZGVudG80eS9jbGFBbXVmc9s2Si6IkFkbWluaXN0cmFkb3IiL
C3leHAiojE3MTM1NjQ0N2AsImZlcy6iLm1pY3Jvc29yZy9wYWI1dUIiwiYXVkiOjoiqo2xpZw50ZXN
hWdlKV1QifQ.qwba2PIZ86_h234UfppRkDq7pKnxtpkNetOQdKJcu"
13 }

```

- The screenshot displays a web interface for working with JSON Web Tokens (JWT). At the top, there's a navigation bar with logos for JWT, OJCTF, and tools like Debugger, Libraries, Introduction, and Ask. It also mentions it was "Crafted by AuthO by Okta".

The main area is split into two panels:

 - Left Panel (Token Generation):** A form where a JWT token is generated. The input field contains a long alphanumeric string. Below the input, the resulting token is displayed in green text: `KnxptknEtOQdDKjclUeyJhbGciOiJodHRwOi8vd3d3LnczLm9yZy8yMDAxLzA0L3htbGRzaWctbW9yZSNoWFjLXNoYTI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm9yZy93cy8yMDA1LzA1L2lkZW50aXR5L2NsYWltcy9uYW1lIjoiz3ZlcmlAdXYubXgiLCJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm9yZy93cy8yMDA1LzA1L2lkZW50aXR5L2NsYWltcy9nZXZlbm5hbWU1OiJHdWlsbGVybW8gSHVtYmVydG8gVmVyYSBBbWFFbyIsImh0dHA6Ly9zY2hlbWFFZLM1pY3Jvc29mdC5jb20vd3MvMjAwOC8wNi9pZGVudGl0eS9jbGFpbXMvcm9sZSI6IkFkbWluaXN0cmFKb3IiLCJleHAiOjE3MTM1NjQ0ZAsImklzcyciOiE1bnZpZG9yRmVpSlldUiwiYXVkIjoizxpZW50ZXNGZW1KV1QifQ.qwba2PIZ86_h234UFfprKrDq7p`.
 - Right Panel (Token Decoding):** Shows the decoded payload of the token. It includes a header section with `"alg": "http://www.w3.org/2001/04/xmldsigs-more#hmac-sha256"` and `"typ": "JWT"`. Below this, a section labeled "PAYLOAD: DATA" shows the decoded claims: `{ "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name": "gvera@uv.mx", "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname": "Guillermo Humberto Vera Amaro", "http://schemas.microsoft.com/ws/2008/06/identity/claims/role": "Administrador", "exp": 1713564470, "iss": "ServidorFeiJWT", "aud": "ClientesFeiJWT" }`. The values `gvera@uv.mx`, `Administrador`, and `ServidorFeiJWT` are highlighted with red boxes.

- ```
Send Request
GET http://localhost:3000/api/categorias
Content-Type: application/json
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwidXNlcm5hbnWUiOiJndmVyYUB1di5teCIsIm5
###
```

- ```
##  
Send Request  
GET http://localhost:3000/api/categorias  
Content-Type: application/json  
Authorization: Bearer eyJhbGciOiJIodHRwOi8vd3d3LnczLm9yZy8yMDU...  
  
###  
  
Send Request  
GET http://localhost:3000/api/peliculas  
Content-Type: application/json  
Authorization: Bearer eyJhbGciOiJIodHRwOi8vd3d3LnczLm9yZy8yMDU...  
  
###
```

10. Haga lo mismo con los otros **endpoints** como por ejemplo, verifique la lista de películas.

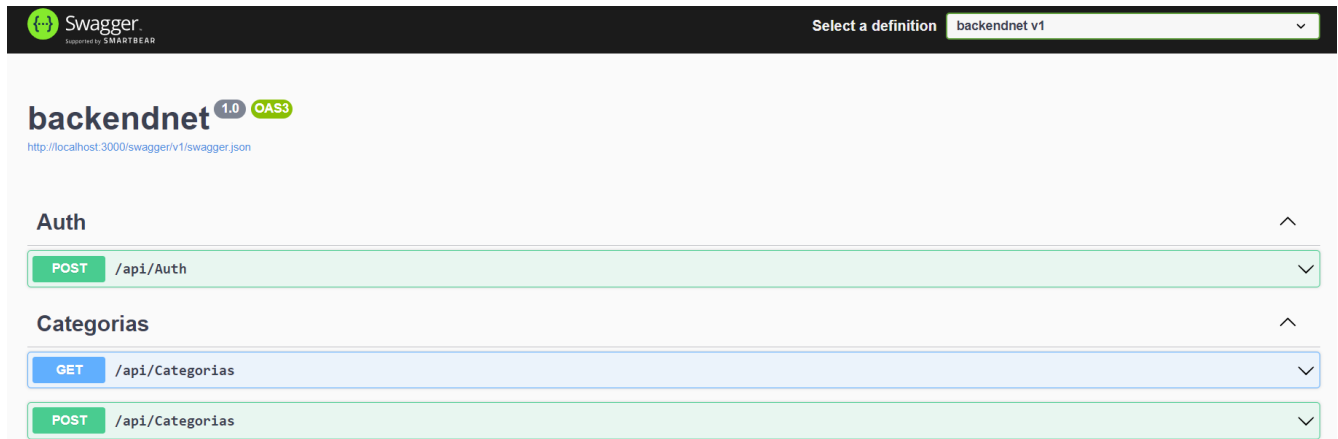
11. Pruebe con un usuario que tenga el perfil de **Usuario** y verifique que no pueda acceder a los recursos del rol **Administrador**.

12. Al faltar 5 minutos para que su token expire, verifique sus peticiones devuelvan un nuevo token en el encabezado **Set-Authorization** a sus clientes para refrescar la autenticación.

13. Enhorabuena, su aplicación funciona y está protegida con autenticación y autorización.

Instrucciones. Agregar documentación a su API

1. Su proyecto ya cuenta con un generador de documentación de API llamado **Swagger** agregado en la práctica anterior.
2. Se genera automáticamente al compilar su proyecto. La configuramos solo para verla en el ambiente de desarrollo.
3. Vuelva a ejecutar su aplicación y ahora cuando entre a la ruta `/swagger`, verá la interfaz de **Swagger** donde se documentan los **endpoints** expuestos por su API.

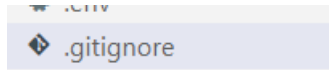


4. Puede utilizar esta interfaz también para probar el funcionamiento de su API. Navegue en ella y verifique que todo funciona correctamente.
5. Enhorabuena, ha configurado su documentación de API de manera correcta.

Instrucciones. Colocar su aplicación a un repositorio de código fuente.

1. Por último, recuerde que antes de subir su aplicación a GitHub, debe contar en la raíz con el archivo llamado `.gitignore` que previene el subir información innecesaria al repositorio público.
2. Abra su archivo desde la raíz de su proyecto y agregue la carpeta `restclient` y la extensión `*.sql` para no subir estos elementos a GitHub pues pueden contener información sensible y privada.

```
# REST Client
restclient
```



```
# SQL scripts
*.sql
```

3. Ahora ya puede publicar su archivo en GitHub para terminar la práctica.
4. Felicidades, ha creado su aplicación de manera correcta.

Tarea. Defina endpoints para generar una bitácora (log) de eventos.

Defina los siguientes **endpoints** para registrar y visualizar los eventos que ocurren en el sistema.

API	Descripción	Cuerpo de la solicitud	Cuerpo de la respuesta	Rol requerido
<i>GET /api/bitácora</i>	Obtener todas los eventos	Ninguno	Lista de eventos del sistema	Administrador
<i>GET /api/bitácora/{id}</i>	Obtener detalles de un evento por ID	Ninguno	Categoría	Administrador

1. Para contar con este registro, debe crear un nuevo **middleware** que guarde en la base de datos los eventos de inicio de sesión, creación, modificación o eliminación de películas, categorías y/o usuarios.
2. Posteriormente cree un nuevo controlador llamado **Bitacora** donde pueda exponer las rutas de la tabla a sus clientes autenticados.