Ejercicio propuesto:

Hasta el momento, todos los endpoints que expone nuestra API son públicos. Vamos a cambiar eso. Intente agregar autenticación en su proyecto y proteger algunos endpoints (Puede guiarse del apunte de la cátedra y del proyecto de referencia). Sugerimos realizar los siguientes pasos en orden para conseguirlo:

- Agregar una nueva entidad llamada User al proyecto con el que venimos trabajando (asegurarse de que contenga un userName o email y también una password para poder autenticarse).

- Hacer una nueva migración y aplicarla para agregar la nueva tabla de usuarios en la base de datos. (considere agregar algunos usuarios con una semilla de datos para pruebas rápidas, o cree parte del CRUD de usuario para agregar uno nuevo en la base de datos) - Crear un nuevo DTO para recibir las credenciales de autenticación.

- Crear UserRepository (y su interfaz) que contenga un método para buscar usuario por email o userName (lo que hayan escogido para autenticarse). (Necesario para el servicio de autenticación). (Recuerda registrar la nueva clase y su interfaz en el contenedor de servicios)

- Crear una nueva Excepción personalizada en Domain.Exceptions para gestionar el escenario donde las credenciales de autenticación fallen (Por ejemplo InvalidCredentialsException). La utilizaremos luego y la lanzaremos en caso de que el proceso de login no tenga éxito. Por supuesto recuerda también modificar tu middleware de manejo de errores global si es que tienes uno. Deberías agregar un case más al switch para que contemple este nuevo tipo de excepción y establecer el status code de la response en un 401 (unauthorized).

- Agrega una nueva sección al archivo appSettings.Development.json para alojar:
    * un string que funcionará como clave secreta que usarás para el signature del JWT (recomendado que dicho string contenga 32 caracteres de longitud).
    * un string que funcione de Issuer. (originante del token)
    * un string que funcione de Audience. (destinatario del token)

- Recuerde instalar el paquete Nuget “Microsoft.Extensions.Configuration.Abstractions” en la capa de infraestructura para poder acceder desde dicha capa a los valores alojados en los archivos de configuración .json que se encuentran en la capa de Presentación.

- Instalar los paquetes Nuget necesarios para poder implementar JWT en nuestra API (Recuerde en qué capa del proyecto se instalan los paquetes de terceros de acuerdo a la arquitectura Clean).

- Dentro de la carpeta de Interfaces de la capa de Application crear una nueva interfaz llamada ICustomAuthenticationService que contenga la firma de un método llamado Autenticar, que retorne un string (token) y reciba por parámetro el nuevo DTO que contiene las credenciales de autenticación.

- Crear un AuthenticationService para que realice la lógica de autenticación y si la autenticación tiene éxito, que genere y retorne el nuevo JWT al controlador. Dicho AuthenticationService deberá estar alojado dentro de una carpeta llamada Services en la capa de Infraestructura, y también deberá implementar la interfaz ICustomAuthenticationService que se encuentra en la capa de Application.

(NOTA: en el AuthenticationService deberá inyectar el UserRepository (a través de su interfaz) y también el IConfiguration para poder acceder a través de él a los valores alojados en el archivo appSettings.Development.json).
(No olvide registrar ICustomAuthenticactionService y AuthenticationService en el contenedor de inyección de dependencias de la clase Program.cs)

- Agrega en la clase Program.cs la configuración necesaria para que nuestra API sepa cómo verificar la autenticación. (esto le indica qué valores del token debe observar y contra qué valores debe compararlos).

- Crear un AuthenticationController que contenga el endpoint de autenticación que llame al servicio de autenticación (debería ser un POST y traer las credenciales en el body).

- Agregar al pipeline de la clase Program.cs los métodos app.UseAuthentication() y app.UseAuthorization() si es que no figuran allí. (Prestar atención de ubicarlos en este orden, y de agregarlos luego de app.UseHttpRedirction() pero antes de app.MapControllers()).

- Protege algunos de tus endpoints de producto o categoría con el decorador [Authorize] para restringirlos y poder probar el token que generaste. (NOTA: el decorador [Authorize] puede utilizarse a nivel de controlador o a nivel de endpoint)

- Si utilizas Swagger para testear tus endpoints deberás agregar en el Program.cs la configuración necesaria para agregar a swagger la funcionalidad de recibir el JWT como parámetro y poder probar endpoints protegidos con [Authorize]. Si utilizas Postman, este paso no es requerido.

Bien, con todo esto debería ser suficiente para poder implementar correctamente el proceso de autenticación en nuestra API. Intente probar la autenticación, conseguir un token, y luego de ingresarlo, intente acceder a algún endpoint protegido.