Linq:

- Nos proporciona comprobaciones de tipo consultas durante la compilación
- Sirve para consultar y guardar datos de diferentes orígenes de datos
- Ejemplo: var people = From p in Personas
  Where p.Edad >= 20 && p.Edad <= 30
  Select p
  Return people

POO:

- Herencia: Heredar los atributos de otra clase. Vamos a poder agregar o modificar su comportamiento
- Encapsulamiento: Modificadores:
  1. Public (acceso total)
  2. Internal (acceso del mismo proyecto)
  3. Private (acceso solo la clase)
  4. Protected (acceso a la clase y las que heredan)

- Polimorfismo: Sobreescritura de métodos:
  - Virtual (le decimos que va a ser sobreescrito)
  - Override (le decimos que lo sobreescriba)

- Abstracción: Proceso de simplificar un sistema complejo

Clean architecture: Aplicación que está compuesta por capas, cada una tiene diferentes responsabilidades y una capa es un conjunto de cosas que tienen cierta responsabilidad. Cuando nos referimos a capas es una abstracción de responsabilidades.

- Aplicación: Define la lógica de negocio (único sistema)
- Domain: Define las entidades y sus relaciones (múltiples sistemas)
- Infra: Comunicación con servicios y paquetes de terceros
- Web: Endpoint (request y response)

Interfaz:

- Contrato de la firma de los métodos, se compromete a usar todos los métodos.
- Puede implementar múltiples interfaces
- Cualquier clase que tenga implementada esta interfaz es válida como tipo de retorno

Inyección de dependencia:

- Evita la dependencia entre clases y simplifica la creación y gestión de objetos en una aplicación
- Es flexible, testable y mantenible

ORM:

- Vincula la BD con POO y en el contexto le damos las instrucciones que necesita
- Mapea estructuras de una BD relacional
- Su objetivo: Simplificar y acelerar el desarrollo de la aplicación

HTTP (comunicación de datos básica en internet) request y response: La comunicación de datos empieza con un request enviado del cliente y termina con la respuesta del servidor web. Las peticiones se hacen siempre siempre al endpoint (punto de acceso de un api que procesa la solicitud y responde de acuerdo a su implementación interna)

Librería: Entity framework core es una librería creada por Microsoft que nos permite acceder a las base de datos relacionales

BD relacionales: Se vincula con las entidades lógicas o con la bd virtual (orm) para que las distintas acciones del CRUD se realiza de manera indirecta a través del ORM. Es un tipo de bd que organiza los datos en tablas (columnas y filas)

Migración: Conjunto de instrucciones que con comandos se corren para crear la BD

Tablas:

- Estructura de datos en la unidad de almacenamiento e interacción de datos en cualquier aplicación que es BD relacional
- Está compuesta por columnas (campos) y filas (registros)
- A través de estas tablas se pueden gestionar y consultar datos

Patrón de diseño:

- Solución común para un problema común
- Patrón repository:
    - Permite crear una clase en donde estén definidos todos los métodos que interactúan con la BD, consume el contexto en la BD
    - Es una abstracción de la capa de datos (abstrae la implementación)
    - Un beneficio es que tu código sea más limpio y fácil de mantener
    - Problema que resuelve: Dispersión de la interacción con la BD
    - Cómo lo resuelve:
        1. Interacción de una entidad con la BD encapsulada
        2. Toda interacción con la aplicación con la BD en un solo sular

- En infra tenemos:
    - Application context: Configuración general de cómo se traduce tu POO a paradigma relacional
        - Hereda de DBContext: Es una clase de entity framework core que se utiliza para interactuar con una BD. Es una clase principal que se utiliza para realizar operaciones CRUD con la BD. Es una clase abstracta que se puede heredar para crear una clase concreta que represente el contexto de la BD de nuestra aplicación. Esta clase concreta se utiliza para la configurar la conexión con la BD, definir las entidades que se almacenarán en la BD y aplicar cambios en la BD
        - Contiene DBSet: Es una clase de entity framework core que representa un conjunto de entidades en la BD. Cada BDSet es una clase BDContext representa una tabla en la BD.

    - Repositories:
        - 1 repository por cada entidad
        - Inyectamos el contexto porque tiene que interactuar con la BD
        - Repositorio base: Todo lo que interactúa con la BD de manera común. (datos genéricos)

Tipo de datos genéricos (T):

- No existe hasta que lo usamos
- La herencia como tal no nos soluciona el poder tener los métodos unificados en una misma clase porque cada uno es distinto. Entonces usamos T que encapsulan

operaciones que no especifican un tipo de dato determinado. where T : class es una restricción de tipo genérico que especifica que T debe ser una clase. Esto impide que T sea un tipo de valor (como int, bool, etc.). : base(dbContext): Es una llamada al constructor de la clase base. Esto significa que la clase hereda de otra clase y está pasando el parámetro dbContext al constructor de esa clase base. Esto es común cuando se está utilizando herencia y la clase base requiere ciertos parámetros en su constructor.

- En el repositorio base inyectamos el BDContext que va a devolver T y ahora los métodos también devuelven T y se va adaptando a cada entidad
- Ahorramos código
- No existe DBSet<T> así que ponemos Set<T>
- Nos permite crear código reutilizable entre múltiples entidades

Autenticación: Proceso de verificar quien es y si es quien dice ser

Cuando autenticamos le estamos dando cierta autorización para que tenga los permisos para ejecutar la acción, una vez que validamos que somos quien decimos ser, debemos comparar si tenemos acceso a ciertas partes de la aplicación

Encriptación: El cifrado hace que los datos sean ilegibles y difíciles de decodificar para un atacante y evita que sean robados. Utiliza “claves criptográficas”. Con la clave, los datos se cifran en el extremo del remitente. Y, con la misma clave o una clave diferente, los datos se descifran en el extremo del receptor

Clave simétrica: Antes de enviar los datos al receptor, el remitente utiliza una clave privada para cifrar los datos. Esta clave privada es conocida solo por el remitente y el receptor. Entonces, una vez que el receptor obtiene los datos cifrados, él o ella usa la misma clave privada del remitente para descifrarlos

Clave asimétrica: Esta técnica implica dos claves diferentes. Se utiliza una clave para cifrar los datos, que se conoce como clave pública, y es conocida prácticamente por todos en Internet. La otra clave se utiliza para descifrar los datos, que se conoce como clave privada, y solo la conoce el receptor y debe mantenerse discreta. Por lo tanto, el uso de dos claves distintas hace que el sistema sea más seguro y se vuelve demasiado difícil para un atacante descifrarlo

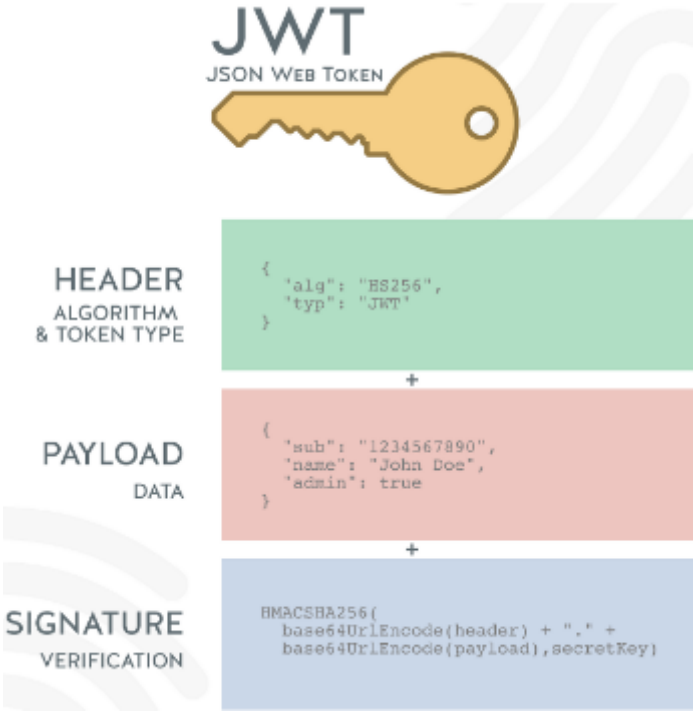
Hash: Es una cadena que se genera a partir de la cadena de entrada pasándola a través de un algoritmo Hash. Esta cadena hash es siempre de una longitud fija sin importar el tamaño de la cadena de entrada. El hash también se puede considerar como “cifrado unidireccional”, es decir, los datos una vez procesados no se pueden revertir a su forma original. Significa que se asegura de que incluso si se cambia una sola cosa, tu puedes saber que se ha cambiado. El hash protege tus datos contra posibles alteraciones para que tus datos no se modifiquen ni un poco.

Autenticación con JWT:

1. El cliente solicita mediante una request a un endpoint de autenticación, el token con el cual va a poder acceder a los servicios de la api. Para esto manda en dicha request las credenciales de autenticación que, en muchos casos, es el usuario y la contraseña.

2. La API verifica que el usuario exista en la base de datos y además verifica si realmente es quien dice ser a través de la confirmación de que la contraseña es la correspondiente para el usuario.
3. La API retorna el token de autenticación que el cliente va a usar a futuro si corresponde. En caso de no reconocer el usuario o de que la validación de las credenciales haya fallado entonces retorna una response con status code 401.
4. Desde ese momento, cada vez que el cliente quiera usar un endpoint de la api va a mandar en la request en un header el token que obtuvo en el paso 3 de la API. Generalmente anteponiendo “Bearer ” Ej: Bearer tokenquerecibiodelaapienelpaso3
5. La api verifica que el token lo haya generado ella y que nadie lo haya alterado.
6. La api procede a procesar la request del cliente normalmente y a responderle lo que corresponda

Estructura JWT:



Un JSON web token es un estándar de código abierto que se utiliza para transmitir información de manera segura entre dos partes en formato JSON. Está diseñado para ser un mecanismo de autenticación y autorización en aplicaciones web y servicios API. Los JWT son especialmente populares en el contexto de aplicaciones basadas en RESTful API y aplicaciones de una sola página

Un JWT consta de tres partes: la cabecera (header), el payload y la firma. La cabecera contiene información sobre el algoritmo de cifrado utilizado y el tipo de token. El payload es el contenido propiamente dicho del token y contiene la información que se quiere transmitir. La signature es la firma del token.

Header:
La cabecera del JWT dentro de la estructura de un JSON web token es la que indica qué formato criptográfico está utilizando el token.

Payload:
En el payload del JWT es donde se encuentra la información propiamente dicha. Esta parte está codificada en Base64, lo que permite que cualquiera pueda ver su contenido, no se puede modificar.

En término de estructura el payload es un diccionario en donde cada par clave valor lo denominamos una “claim”. Como es un diccionario no puede haber más de una clave con el mismo identificador. Existen claims obligatorias y también el servidor puede agregar las que quiera a la hora de generar el JWT. Algunas de las obligatorias más comunes son: exp (fecha de expiración) iat (fecha de creación)

Firma del token (signature): Es la parte de la estructura de un JSON web token que garantiza la autenticidad del JWT. Aquí es donde entra en juego la clave privada. La firma se genera usando el contenido de la cabecera y el payload, además de una clave secreta que solo conoce el servidor. De esta manera, al recibir el JWT, el servidor puede verificar si ha sido alterado en el camino.

Signature = Hash(Base64(Header)+Base64(Payload)+Secret)

Como el secreto es algo que el servidor solo conoce y a partir de la signature nadie puede deducir el Secret porque los hash son irreversibles entonces no hay forma de que alguien puede generar una Signature válida si se crean o modifican un payload y/o un header porque les falta un parte para poder generarla, el Secret.

- Issuer: es quien creó el token, el identificado de la api que generó el JWT. Se coloca como claim en el payload
- Audience: a quién va dirigido dicho token. Se coloca como claim en el payload
- Claims personalizadas: una lista de claims que se van a agregar al payload del JWT.
- Fecha de creación: en UTC que es GMT0 para poder hablar un lenguaje común
- Fecha de expiración: un periodo a partir de la fecha de creación en donde el JWT va a ser válido. Este valor es una claim del payload
- Signature: es la firma del JWT que pre generamos con anterioridad que ahora él constructora la va usar para armar la firma definitiva del JWT usando el payload entero del JWT con todos los valores que le pasamos

Para que el chequeo de autenticación se haga efectivo tenemos que explicitar en los lugares que así lo deseamos para que se aplique con el decorador [Authorize], este decorador puede ser aplicado a todo el controlador poniéndolo justo de la definición de la clase controller o bien a cada endpoint en particular.