1. Descripción General

Este proyecto es una API de autenticación de usuarios construida usando ASP.NET Core. Permite el registro, inicio de sesión, obtención de información de usuarios y edición de usuarios en un sistema. Además, está protegida con autenticación JWT (JSON Web Token) y roles de usuario.

2. Estructura del Proyecto

El proyecto está dividido en varias capas de la siguiente manera siguiendo una clean architecture:

1. Domain
2. Application
3. Infrastructure
4. Presentation

Los endpoints principales:

1. Register - POST /api/AuthenticationApi/Register

- **Descripción**: Registra un nuevo usuario en el sistema.
- **Requiere**: Un objeto UserDTO con los detalles del usuario (nombre, teléfono, dirección, correo electrónico, contraseña y rol).
- **Responde**:
    - **200 OK**: Si el registro fue exitoso.
    - **400 Bad Request**: Si hay algún error en el modelo o los datos enviados (como un correo ya registrado).

2. Login - POST /api/AuthenticationApi/Login

- **Descripción**: Inicia sesión en el sistema utilizando el correo electrónico y la contraseña del usuario.
- **Requiere**: Un objeto LoginDTO con el correo electrónico y la contraseña.
- **Responde**:
    - **200 OK**: Si el inicio de sesión es exitoso, devuelve un token JWT.
    - **400 Bad Request**: Si las credenciales son incorrectas o el modelo no es válido.

3. GetUser - GET /api/AuthenticationApi/{id:int}

- **Descripción**: Obtiene la información de un usuario por su ID.
- **Requiere**: Un parámetro id que representa el ID del usuario.
- **Responde**:
    - **200 OK**: Si el usuario existe, devuelve los datos del usuario.
    - **400 Bad Request**: Si el ID no es válido.
    - **404 Not Found**: Si el usuario no es encontrado.

4. GetAllUsers - GET /api/AuthenticationApi

- **Descripción**: Obtiene una lista de todos los usuarios registrados en el sistema.
- **Requiere**: **Autenticación** y **Permiso de Admin**.
- **Responde**:
    - **200 OK**: Si hay usuarios registrados, devuelve una lista de ellos.
    - **404 Not Found**: Si no se encuentran usuarios registrados.

5. EditUser - PUT /api/AuthenticationApi

- **Descripción**: Edita la información de un usuario existente en el sistema.
- **Requiere**: Un objeto UserDTO con los nuevos datos del usuario (ID obligatorio).
- **Responde**:
    - **200 OK**: Si el usuario fue actualizado correctamente.
    - **400 Bad Request**: Si el modelo de datos es inválido o el ID del usuario es incorrecto.
    - **404 Not Found**: Si el usuario no existe.

Funcionamiento del Inicio de Sesión con JWT y Seguridad en mi API

1. Proceso de Inicio de Sesión

El proceso de inicio de sesión con JWT (JSON Web Token) en **mi API** sigue estos pasos:

1. **Solicitud de Inicio de Sesión**:
    - El usuario envía una solicitud POST a la ruta /api/AuthenticationApi/Login con un objeto LoginDTO que contiene su correo electrónico y contraseña.
    - Ejemplo de los datos enviados:

json

Copiar

```
{
  "email": "user@example.com",
  "password": "password123"
}
```

2. **Verificación de Credenciales**:
    - **Mi API** recibe la solicitud y busca al usuario en la base de datos utilizando el correo electrónico proporcionado.
    - Si el usuario no existe, se devuelve un mensaje de error ("credenciales inválidas").
    - Si el usuario existe, **mi API** utiliza **BCrypt** para comparar la contraseña proporcionada con la que está almacenada en la base de datos. BCrypt es un algoritmo de hash de contraseñas que asegura que la contraseña nunca sea almacenada como texto claro en la base de datos.
3. **Generación del JWT**:
    - Si las credenciales son válidas, **mi API** genera un **JSON Web Token (JWT)** para autenticar futuras solicitudes del usuario.
    - El token es generado utilizando los datos del usuario (como su nombre, correo electrónico y rol) y una clave secreta definida en la configuración de la API.
    - El token se firma con **HMAC SHA-256**, un algoritmo de seguridad simétrica, lo que garantiza que el token no pueda ser modificado por un atacante.
4. **Envío del Token**:

- **Mi API** envía el token generado como respuesta al cliente en formato JSON.
- El token tiene un tiempo de expiración, configurado en la API (por ejemplo, 1 hora), lo que limita su validez.
- Ejemplo de la respuesta:

json

Copiar

{

"flag": true,

"message": "Login successful",

"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiIxMjM0NTYiLCJleHBpcmVzIjoxMjM0NTYsImVtYWlsIjoidXNlckBleGFtcGxlLmNvbSJ9.12345abcde..."

}

5. **Autenticación en Solicitudes Posteriores**:
   - Para realizar operaciones protegidas, como obtener o editar datos de usuarios, el cliente debe incluir el **JWT** en el encabezado de autorización de sus solicitudes HTTP.
   - Ejemplo:

h

Copiar

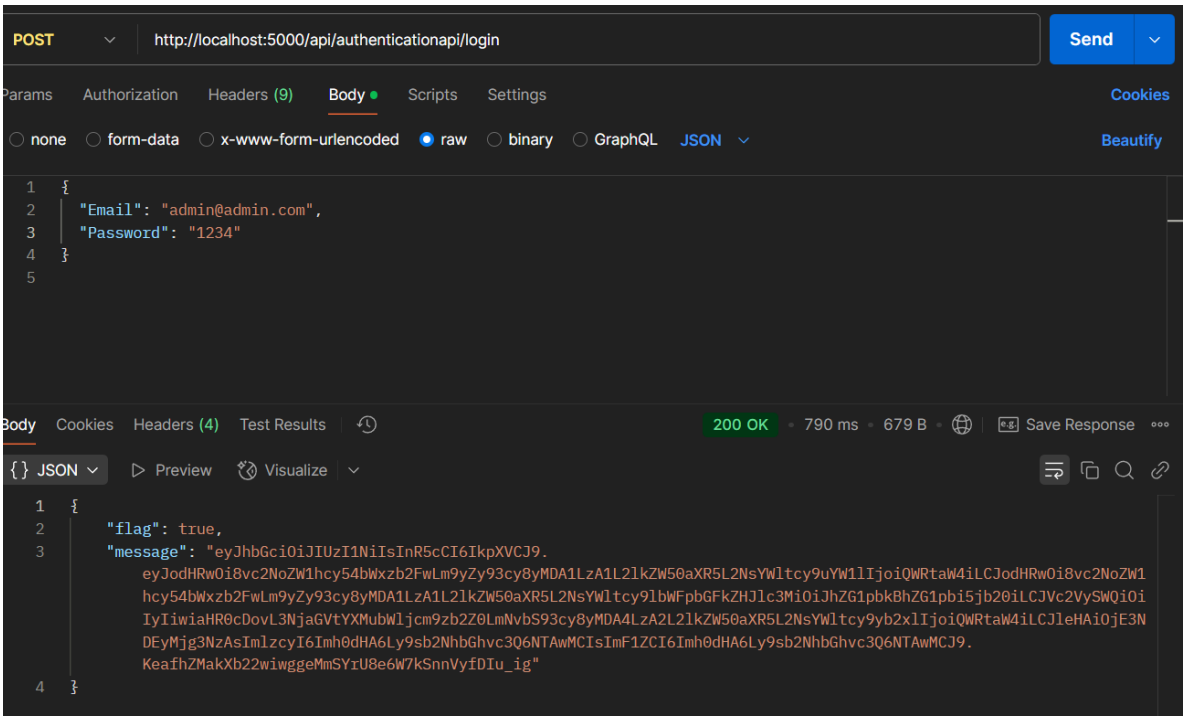
Authorization: Bearer <token_jwt_aqui>

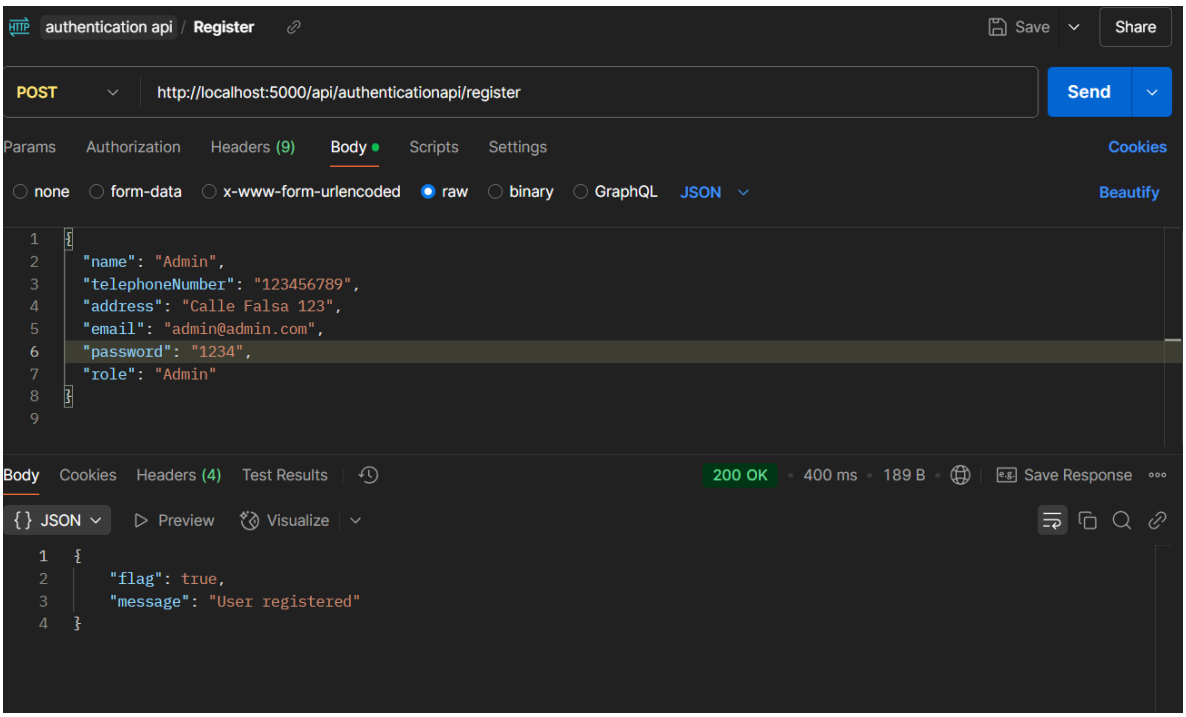
2. Seguridad del JWT en Mi API

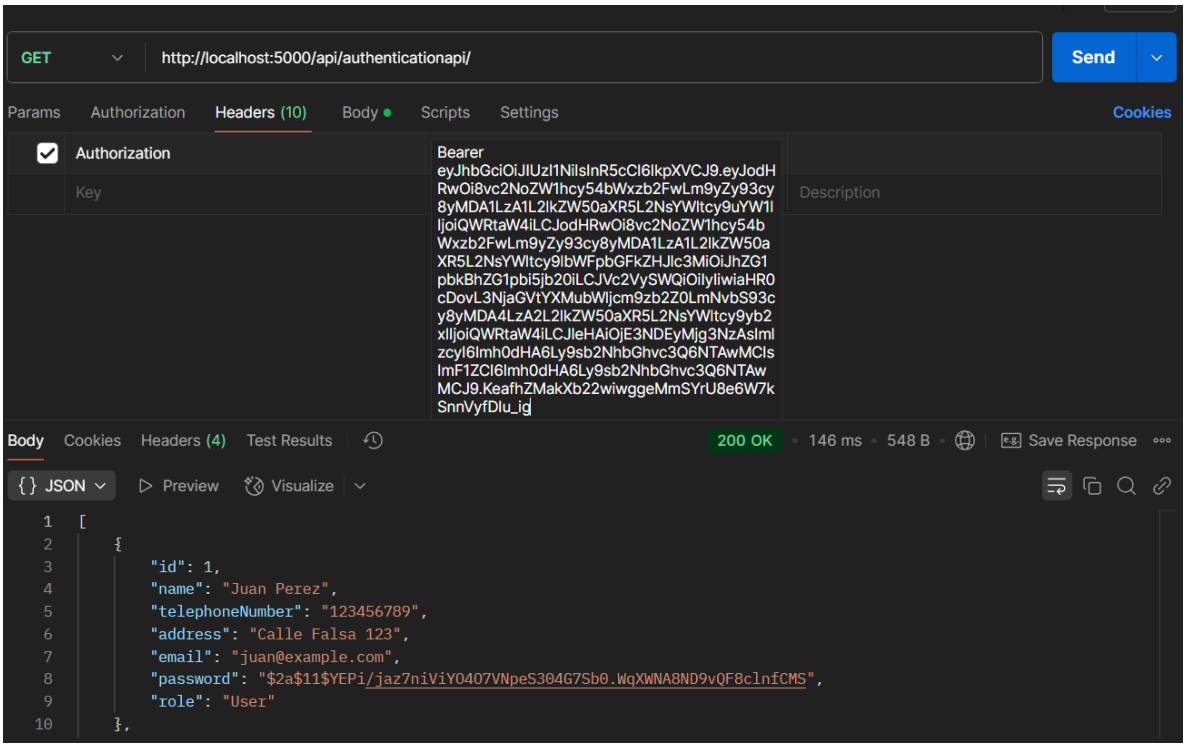
El uso de JWT mejora la seguridad de **mi API** en varios aspectos:

- **Firmado del Token**: El JWT está firmado con una **clave secreta** utilizando el algoritmo HMAC SHA-256. Esto asegura que el token no pueda ser alterado sin invalidarlo. Si un atacante intenta modificar el token, la firma no coincidirá con el contenido y será rechazado por **mi API**.

- **Datos Encapsulados (Claims)**: El JWT incluye un conjunto de **claims** (información del usuario) que está codificada pero no cifrada. Estos claims pueden incluir información como el ID del usuario, su rol, y el tiempo de expiración del token. Como el token está firmado, los datos no pueden ser manipulados, aunque son accesibles para el cliente.
- **Expiración del Token**: Cada JWT tiene un **tiempo de expiración** (por ejemplo, 1 hora). Esto limita el tiempo en que un atacante puede usar un token robado. Después de la expiración, el usuario debe iniciar sesión nuevamente para obtener un nuevo token.
- **Almacenamiento Seguro del Token**: Aunque el JWT es accesible en el cliente, es crucial almacenarlo de forma segura, típicamente en **almacenamiento local** (localStorage) o **cookies seguras**. Si se almacena en cookies, se debe configurar correctamente con las opciones **HttpOnly** y **Secure** para evitar ataques como el cross-site scripting (XSS).
- **Autorización Basada en Roles**: Al incluir información sobre el **rol** del usuario en el JWT, **mi API** puede verificar los permisos de acceso antes de permitir acciones específicas (como obtener todos los usuarios). Esto protege los endpoints sensibles y garantiza que solo los usuarios con permisos adecuados puedan acceder a ciertos recursos.


POST
http://localhost:5000/api/authenticationapi/login
Send
Params
Authorization
Headers (9)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"Email": "admin@admin.com",
"Password": "1234"
}
Body
Cookies
Headers (4)
Test Results
200 OK
790 ms
679 B
Save Response
JSON
Preview
Visualize
{
"flag": true,
"message": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm9yZy93cy8yMDA1LzA1L2lkZW50aXR5L2NsYWltcy9uYW1lIjoiQWRtaW4iLCJodHRwOi8vc2NoZW1
hcy54bWxzb2FwLm9yZy93cy8yMDA1LzA1L2lkZW50aXR5L2NsYWltcy9lbWFpbGFkZHJlc3MiOiJhZG1pbkBhZG1pbi5jb20iLCJVc2VySWQiOi
IyIiwiaHR0cDovL3NjaGVtYXMubWljcm9zb2Z0LmNvbS93cy8yMDA4LzA2L2lkZW50aXR5L2NsYWltcy9yb2xlIjoiQWRtaW4iLCJleHAiOjE3N
DEyMjg3NzAsImlzcyI6Imh0dHA6Ly9sb2NhbGhvc3Q6NTAwMCIsImF1ZCI6Imh0dHA6Ly9sb2NhbGhvc3Q6NTAwMCJ9.
KeafhZMakXb22wiwggeMmSYrU8e6W7kSnnVyfDIu_ig"
}


authentication api / Register
Save
Share
POST
http://localhost:5000/api/authenticationapi/register
Send
Params
Authorization
Headers (9)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"name": "Admin",
"telephoneNumber": "123456789",
"address": "Calle Falsa 123",
"email": "admin@admin.com",
"password": "1234",
"role": "Admin"
}
Body
Cookies
Headers (4)
Test Results
200 OK
400 ms
189 B
Save Response
JSON
Preview
Visualize
{
"flag": true,
"message": "User registered"
}


GET
http://localhost:5000/api/authenticationapi/
Send
Params
Authorization
Headers (10)
Body
Scripts
Settings
Cookies
Authorization
Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm9yZy93cy8yMDA1LzA1L2lkZW50aXR5L2NsYWltcy9uYW1lIjoiQWRtaW4iLCJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm9yZy93cy8yMDA1LzA1L2lkZW50aXR5L2NsYWltcy9lbWFpbGFkZHJlc3MiOiJhZG1pbkBhZG1pbi5jb20iLCJVc2VySWQiOiIyIiwiaHR0cDovL3NjaGVtYXMubWljcm9zb2Z0LmNvbS93cy8yMDA4LzA2L2lkZW50aXR5L2NsYWltcy9yb2xlIjoiQWRtaW4iLCJleHAiOjE3NDEyMjg3NzAsImlzcyI6Imh0dHA6Ly9sb2NhbGhvc3Q6NTAwMCIsImF1ZCI6Imh0dHA6Ly9sb2NhbGhvc3Q6NTAwMCJ9.KeafhZMakXb22wiwggeMmSYrU8e6W7kSnnVyfDlu_ig
Key
Description
Body
Cookies
Headers (4)
Test Results
200 OK
146 ms
548 B
Save Response
JSON
Preview
Visualize
[
{
"id": 1,
"name": "Juan Perez",
"telephoneNumber": "123456789",
"address": "Calle Falsa 123",
"email": "juan@example.com",
"password": "$2a$11$YEPi/jaz7niViYO4O7VNpeS304G7Sb0.WqXWNA8ND9vQF8clnfCMS",
"role": "User"
},