Elías Rojano Núñez

Configuración de Hardware y Software de Desarrollo:

Sistema Operativo: macOS 14.2.1 (23C71)
Hardware: Macbook Pro 16 pulgadas, 2021
Procesador: Apple M1 Pro
.NET SDK: 8.0

Requisitos Recomendados de Software:

Postman 10.17 o superior
.NET SDK 8.0 o superior
Windows, Mac o Linux
(NOTA: Si bien el sistema podría funcionar en versiones inferiores, las que se mencionan son las que fueron utiliadas y corroboradas en el desarrollo)

Instrucciones para ejecutar el sistema de forma local:

1. Dejar el directorio HaulmerTest dentro de su dispositivo de almacenamiento local.

2. Abrir la consola de comandos de su sistema operativo y posicionarse dentro del directorio HaulmerTest.

3. Ejecutar el comando
dotnet run

4. Tomar nota del puerto que utilizará .NET Core para su ejecución (será necesario para las consultas de API).

5. Importar la colección de consultas en Postman y en caso de ser necesario, modificar el puerto en la variable de entorno llamada url. Por defecto se utiliza http://localhost:5255. En caso de utilizar el proyecto montado en mi servidor se deberá modificar toda la url.

6. Comenzar a probar las distintas pruebas realizadas en la colección. La mayoría de pruebas contienen más sub-pruebas en a modo de pruebas de ejemplo.

Instrucciones para ejecutar el sistema en Docker:

1. Dejar el directorio HaulmerTest dentro de su dispositivo que contiene Docker y Docker Compose.

2. Abrir la consola de comandos de su sistema operativo y posicionarse dentro del directorio HaulmerTest.

3. Ejecutar el comando
docker compose up -d (en algunos sistemas este comando puede variar)

Se utilizará el puerto 8097, por lo que debería reemplazar en Postman la variable de url a http://localhost:8097

Desiciones de desarrollo

1. Uso de Modelos para los Usuarios
Con tal de realizar buenas prácticas de desarrollo se decide describir la estructura que deberán seguir los registros de los usuarios, donde todos los campos son requeridos a excepción del id, y las fechas, dado que estos se asignan por el sistema y no por los datos recibidos mediante POST.

2. **Lógica de Encriptación en Utilities**
Con tal de mantener una buena práctica de desarrollo, se separa el módulo de la lógica de encriptación de los controladores.

3. C**reación de id como un parámetro adicional a los pedidos en los Usuarios**
Se decide añadir el parámetro id el cual se trata de un int que va aumentado con cada nuevo usuario registrado, esto con tal de poder identificar a cada usuario aún así se repitieran sus datos, útil para el endpoint para buscar por id.

4. **Creación de un nuevo endpoint que permite buscar id**
Se creó un nuevo endpoint url/users/id/{id} el cual permite obtener los datos de un usuario en particular. Por ejemplo: url/users/id/1 nos devolverá toda la información referente únicamente al usuario con el id 1.

5. Si bien la encriptación podría servir al momento de registrar un usuario, esto rompería la instrucción original de almacenar la contraseña y luego mostrarla al momento de obtener todos los usuarios, por lo que se descarta su implementación.

6. **Desición de utilizar Docker Compose en vez de únicamente Docker**
Con tal de escalar el sistema a mi servidor personal y dejar disponible la API al público mediante una conexión segura a través de HTTPS, decidí utilizar Docker Compose, ya que para poder añadirlo a Traefik, esto es un paso necesario.

7. La versión pública de la API se puede encontrar en mi dominio con una url personalizada el cual se le hará entrega mediante correo electrónico con tal de mantenerlo privado. Además de contar con los certificados HTTPS necesarios, también está a través de Cloudflare Proxy lo cual nos brinda una capa de seguridad adicional.