Security_API

Álvaro Maleno Alférez

Tabla de contenidos

Security API

Manual simple de usuario

1. Puesta en marcha

1.1 Instalando dependencias

Para que este API pueda funcionar son necesarias las siguientes dependencias:

- SDK de .Net Core 3.1

- ASP de .Net Core 3.1

- .Net Core 3.1

- PostgreSQL 11.8

Pueden seguirse los siguientes tutoriales:

- https://docs.microsoft.com/es-es/dotnet/core/install/linux

- https://www.postgresql.org/docs/11/installation.html

1.2 Arrancando la Aplicación

1. El primer paso será la configuración de una base de datos sobre la cual realizar la persistencia y consulta de la información referente a los nuevos usuarios. Desde la página oficial de PostgreSQL puede consultarse una guía completa: https://www.postgresql.org/docs/11 .

2. Una vez generada una base de datos, será necesario configurar el programa para que acceda a la misma. Para ello se ha generado un archivo .json en el cual es posible introducir las credenciales y url de la base de datos. La API se encargará de gestionar esa información tras el primer arranque sin que el usuario tenga que realizar ninguna tarea adicional.

La ruta al archivo es **/APIs/Secutiry_API/DAOs/Connection/connectionProperties.json** . Se trata de sustituir la información de cada campo por la generada en el paso anterior:

{

"server": "introducir url",

"port": "introducir puerto",

"userId": "introducir usuario",

"pass": "introducir contraseña",

"dataBase": "introducir base de datos"

}

1.3 Levantando la aplicación

1. Navegar hasta la ruta **APIs/Security_API**.

2. Ejecutar el comando **dotnet build** desde la consola de comandos. Ref: https://docs.microsoft.com/es-es/dotnet/core/tools/dotnet-build

3. Ejecutar el comando **dotnet run** desde la consola de comandos. Ref: https://docs.microsoft.com/es-es/dotnet/core/tools/dotnet-run

4. Es posible sustituir los dos pasos anteriores por al arranque desde un **IDE**.

5. Esperar varios minutos. La aplicación generará todas las bases de datos necesarias para su empleo e insertará todos los datos necesarios. Si tras varios minutos no se observase la creación de ninguna tabla en base de datos se recomienda detener la aplicación y comprobar la conexión al servidor de base de datos.

2. Usando la aplicación. Los métodos.

La aplicación se encuentra configurada para atender peticiones GET y POST enviadas a la url https://localhost:5003/Security . Si desea cambiarse el puerto de acceso es posible modificando el archivo **APIs/Seurity_API/Properties/launchProperties.json** .

Nota: Todas las encriptaciones serán sometidas a una conversión posterior a **Base64**. Desde la misma se volverá a generar el código binario que será desencriptado posteriormente.

2.1 **Get**

Está pensado para eliminar la necesidad de exponer los datos sensibles del usuario. Antes de dar de alta un nuevo usuario se solicitará, mediante éste método una clave pública con la que encriptar los datos. Una respuesta de ejemplo será, simplemente, la que sigue:

<RSAKeyValue>
<Modulus>nlFoRNsY4J2Bfvg/5e8NHocLCLkLbrWte/JnBCnbT2hn1Zh3s/mOHv6SCh1UmaXZ9b5Ey0/hKi
<Exponent>AQAB</Exponent></RSAKeyValue>

2.2 **Nuevo Usuario**

Se efectúa una única vez y sirve para dar de alta a un nuevo usuario en el sistema. Éste usuario recibirá una pareja de clave pública y privada únicas e intrasferibles para él. Su contraseña será almacenada en base de datos encriptada con su propia clave privada y, a su vez, su clave privada se almacenará, junto a la pública, encriptada mediante una clave pública propia de la aplicación que se irá modificando cada 100 usos de la misma. Esto quiere decir que una buena parte de la base de datos será actualizada cada cierto tiempo para salvaguardar la información crítica del usuario.

Un ejemplo de petición sería la siguiente:

{

"email":
"PIwpNi5y3S3g37JXpBfWUF9mhKo2mQHWOmy79HFnLb6BvGSy8S1ayXQaKctaQsBXvDBYJzUizhBQKcH

"pass":
"iLqLoAOSx2oecCirygpc9UcCrJCdWTlnHc2+piYLxY/YnfmU5hTOrpai9A/8aoFR2WUbaBv+cUpJmLKfLy6/U

"new": true,

}

Con el elemento **“new”** se indica que se pretende dar de alta a un usuario nuevo.

Una respuesta con la clave pública única de usuario, que éste deberá de almacenar es generada:

<RSAKeyValue>
<Modulus>nlFoRNsY4J2Bfvg/5e8NHocLCLkLbrWte/JnBCnbT2hn1Zh3s/mOHv6SCh1UmaXZ9b5Ey0/hKil
<Exponent>AQAB</Exponent></RSAKeyValue>

2.3 **Autentificar**

Con éste método se comprueba si el usuario ha sido registrado previamente en base de datos y si los datos transmitidos son correctos. Como la petición anterior, se compone de los elementos “email” y “pass”, pero en esta ocasión deben de haber sido encriptados con la clave pública única para el usuario que será remitida dentro del elemento “public_key”.

Una petición de ejemplo es la que sigue:

{

"email":
"PIwpNi5y3S3g37JXpBfWUF9mhKo2mQHWOmy79HFnLb6BvGSy8S1ayXQaKctaQsBXvDBYJzUizhBQKcH

"pass":
"iLqLoAOSx2oecCirygpc9UcCrJCdWTlnHc2+piYLxY/YnfmU5hTOrpai9A/8aoFR2WUbaBv+cUpJmLKfLy6/U

"new": false,

"public_key" : "<RSAKeyValue>
<Modulus>nlFoRNsY4J2Bfvg/5e8NHocLCLkLbrWte/JnBCnbT2hn1Zh3s/mOHv6SCh1UmaXZ9b5Ey0/hKil
<Exponent>AQAB</Exponent></RSAKeyValue>"

}

2.4 Método Adicional. Encriptar texto.

Para facilitarle las cosas tanto al desarrollador como al posible usuario se ha habilitado un servicio extra que permite la recepción de un texto a encriptar y una clave pública a emplear y devuelve el texto encriptado y posteriormente trasladado a Base64. Está disponible desde la url https://localhost:5003/EncryptationService/ y permite la recepción de peticiones como la siguiente:

{

"key":

"<RSAKeyValue>
<Modulus>nlFoRNsY4J2Bfvg/5e8NHocLCLkLbrWte/JnBCnbT2hn1Zh3s/mOHv6SCh1UmaXZ9b5Ey0/hKil
<Exponent>AQAB</Exponent></RSAKeyValue>",

"text": "miuser"

}

La respuesta sería la siguiente:

xP9BAdai/dJDWZqAKaWjUWiSvK4U1I76xxkYcU2q+dFf2jtY8yN3MUXqGxCdmmyAPtw2lVnWQTPLlp8EbT

3. El modelo de datos