7-12-2022

Compresión y Seguridad

Práctica 3

Godofredo Folgado Pastor

Índice

Contenido del archivo comprimido

En el archivo comprimido podemos encontrar una memoria del proyecto realizado y tres carpetas principales:

- La carpeta "*cs*": esta carpeta actúa como servidor. Está desarrollado en typescript con la herramienta nodejs y que utiliza las librerías Express, Morgan y Mongojs para su correcto funcionamiento.
  Además hace uso de la base datos llamada *CS* de tipo mongodb, y por ende no sql. Lo más importante a destacar es el archivo index.js donde están impletadas todas las rutas utilizadas en el cliente y algunas extras para depuración.
- La carpeta "*archivos_prueba"*: en esta carpeta hay varios archivos de diferentes extensiones con las que nuestro equipo de desarrollo ha estado trabajando.
- La capeta "*cs-app"*: esta actúa como cliente. Está desarrollada en java y hace uso de un software externo llamado Maven.
  Dentro de la capeta "*icons"* podemos encontrar los iconos utilizados en el cliente.
  Las clases y el código fuente se encuentra en el siguiente directorio tomando como nodo origen esta carpeta: "*demo/src/main/java/com/example*".
  El fichero donde se encuentran las dependencias utilizadas de Maven se encuentra en la carpeta "*demo"* con el nombre de "*pom*".

Manual de usuario

Es este apartado se explicarán los pasos para que un usuario que nuca haya utilizado el programa pueda realizar la subida y bajada de un fichero.

Aclaración: Únicamente se va ha explicar el proceso donde todos los pasos son correctos, pero, por supuesto que se han tenido en cuenta errores de usuario y todos estos son informados a los usuarios mediante dialogs.

Aclaración: La siguiente imagen es un dialog:

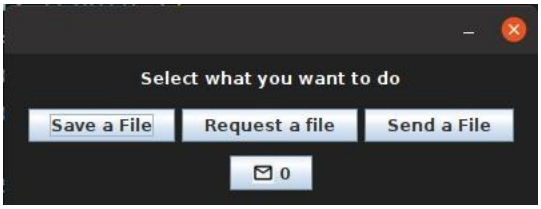
Al lanzar el cliente lo primero que verá el usuario será la siguiente ventana:

Como podemos ver nos pide que introduzcamos un nombre de usuario y contraseña. Seguidamente el usuario habrá de clicar uno de los dos botones inferiores.

En el caso de haber ciclado en registrarnos y que el proceso sea correcto se abrirán dos dialogs. El primero informará al usuario que se el va ha mostrar una imagen qr, que tenga cuidado para que no la muestre a otras personas y que deberá escanearlo con Google autenticator. El segundo diaglog es el código qr. Este último dialog se ve de la siguiente forma:

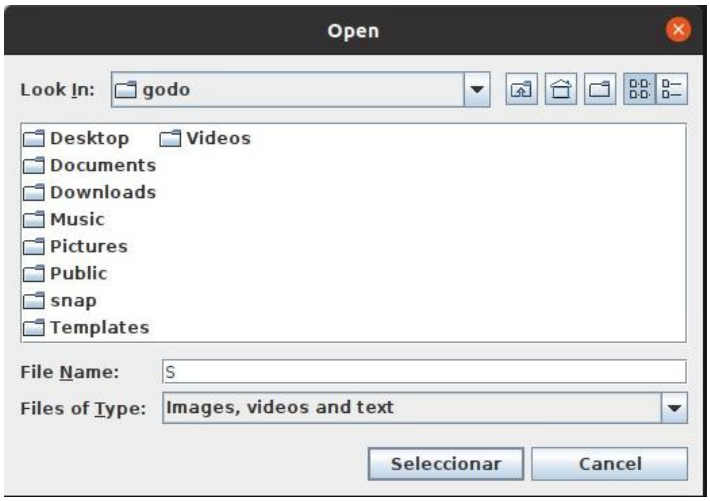
Tras este paso el usuario deberá de introducir el código de 6 digitos que le proporcione Google autenticator en la siguiente ventana:

Desde este punto el usuario tendrá cuatro opciones: Subir un archivo, Descargar un archivo, Enviar un archivo o Ver las notificaciones que le han llegado. Estas opciones aparecen en la siguiente ventana:



Subir un archivo

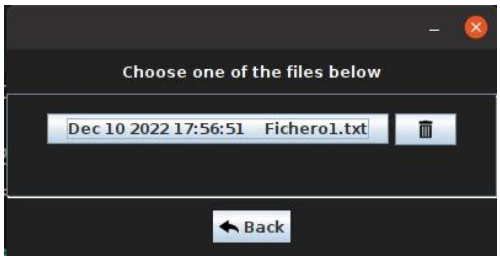
En el caso de que el usuario clique el botón “save a file” se abrirá un dialog pidiendo que escoja un archivo. Cuando el dialog sea cerrado, se abrirá un filechooser como el siguiente, donde el usuario tendrá que seleccionar un archivo.



Una vez se escoja el archivo aparecerán tres dialogs más, uno informando de la correcta selección del archivo, otro informando de la correcta subida del mismo al servidor y un último dando las gracias por la confianza en la aplicación.

Descargar un archivo

En el caso de que el usuario clique el botón “request a file” se abrirá la siguiente ventana:



Como podemos ver, hay un scrollPane donde aparecen todos los ficheros del usuario. En este punto el usuario tendrá la opción de clicar en uno de los botones que representan a sus archivos subidos o el botón de papelera que aparece al lado de cada archivo.

En el caso de que pinche en el botón del archivo y si todo ha salido correcto, a la hora de la selección del fichero, aparecerá un dialog informando de esto mismo.

Tras cerrar el dialog anterior se abrirá un filechooser donde el usuario tendrá que seleccionar una carpeta, en la cual se guardará el archivo.

En el caso de que el usuario clique en el botón de la papelera aparece un dialog pidiendo una confirmación. Una vez se confirme la eliminación del fichero, este será borrado de la base de datos.

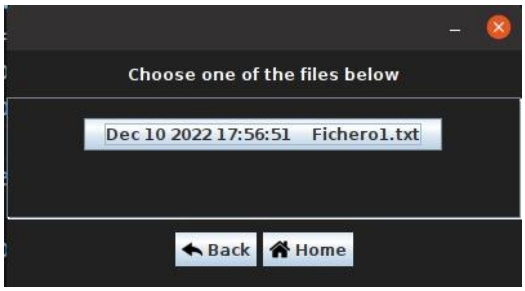
Por último, aparecerá un dialog de agradecimiento por la confianza en la aplicación.

Enviar un fichero

Si el usuario clica en el botón “send a file” se abrirá la siguiente ventana:

En esta ventana el usuario deberá, o bien, introducir el nombre del usuario, o bien, seleccionar uno de los usuarios de su lista de amigos.

Tras este proceso se abrirá una ventana donde aparecerán todos los ficheros del usuario. Cuando clique encima de uno de estos ficheros, este se enviará al usuario. La apariencia de la ventana donde aparecen los ficheros es la siguiente:



Una vez completado el envio del fichero aparecerá un dialog preguntando si desea añadir al usuario a su lista de amigos en el caso de que no se encuentre ya en la misma.

Ver notificaciones

Por último, si el usuario clica en el botón de notificaciones se abrirá, por cada notificación, un dialog informando al usuario quien ha enviado el fichero y el nombre del mismo. En este mismo dialog se le preguntará al usuario si quiere añadir el archivo a su lista de archivos.

En el caso de que el usuario no quiera añadirlo simplemente se abría la siguiente notificación en el caso de existir.

En el caso donde el usuario si ha deseado añadir el archivo a su lista se abríra otro dialog preguntando si desea descargarlo inmediatamente (sin tener que ir a la opción “request a file”).

Documentación sobre la implementación

Sofware

El software utilizado para esta segunda práctica ha sido el mismo que para la primera.

Como editor de texto se ha utilizado Visual Studio Code con plugins como:

- Debugger for JAVA v0.46.0 supported by Microsoft
- Extension Pack for JAVA v0.25.6 supported by Microsoft
- IntelliCode v1.2.29 supported by Microsoft
- IntelliCode API Usage Examples v0.2.6 supported by Microsoft
- Language Support for JAVA™ v1.12.0 supported by RedHAT
- Maven for JAVA v0.39.2 supported by Microsoft
- Proyect Manager for JAVA v0.21.1 supported by Microsoft
- Test runner for Java v0.37.1 supported by Microsoft

Para la parte servidora hemos utilizado Nodejs como entorno de ejecución y mongodb como base de datos. Para el desarrollo del servidor han hecho falta librerías como:

- Express para realizar un api rest
- Morgan como logger
- Mongojs para acceder a la base de datos

Clases utilizadas

En este apartado voy a nombrar las clases que hay en el cliente y que acciones realizan cada una. Así como las librerías más importantes.

Clases utilizadas como objetos auxiliares

Button.java: se utiliza para relacionar un JButton con el id de un fichero.

Fichero.java: se utiliza para simplificar la información de los ficheros obtenidos de la base de datos.

Clases utilizadas como librerías

CheckData.java: se utiliza para comprobar que el nombre y contraseña introducidos en el registro son válidos.

Conexión.java: se utiliza para realizar peticiones al servidor. Podemos identificar 6 métodos, todos importantes:

- sendGet(): realiza una petición get a la ruta especificada en el constructor. Llama a getResponse y recibe un JSONObject como respuesta.
- sendGet2(): realiza una petición get a la ruta especificada en el constructor. Llama a getResponse2 y recibe un JSONArray como respuesta.

Aclaración: es necesario dos métodos distintos para el correcto manejo de respuestas. Se podría realizar la misma función en un método, pero de esta forma el código es más legible.

- sendPost(JSONObject): realiza una petición post a la ruta especificada en el contructor e inserta en el body el json pasado por parámetro. Llama a getResponse y recibe un JSONObject como respuesta.
- sendPut(JSONObject): realiza una petición post a la ruta especificada en el contructor e inserta en el body el json pasado por parámetro. Llama a getResponse y recibe un JSONObject como respuesta.
- getResponse(CloseableHttpResponse): realiza la conversión de CloseableHttpResponse a JSONObject. Devuelve el JSONObject.
- getResponse2(CloseableHttpResponse): realiza la conversión de CloseableHttpResponse a JSONArray. Devuelve el JSONArray.

Hash.java: se utiliza tanto para generar un salt como para realizar el hash del password. Podemos identificar 3 métodos importantes:

- genSalt(byte[]): Si el valor de la variable pasada por parámetro es null genera un salt. Si no simplemente asigna el salt pasado por parámetro a la variable de clase.
- doHash(string): Llama a la función generateArgon2id() pasándole el string que le llega por parámetro. También realiza la conversión de byte[], que devuelve generateArgon2id(), a string para devolver esa string.
- generateArgon2id(): Establece los parámetros que va a tener el constructor de Argon2 y seguidamente llama a este constructor con los parámetros establecidos. Devuelve el hash en forma de array de bytes.

Clases de encriptación

EncriptadorAES.java: se utiliza para generar una clave AES y usar esa clave para encriptar y desencriptar. Tiene 3 funciones, todas importantes:

- genKey(byte[]): Utiliza el array de bytes pasados por parámetro para generar un SecretKeySpec. Devuelve este SecretKeySpec.
- encript(byte[], byte[]): Utiliza el segundo array de bytes para llamar a genKey y con el SecretKeySpec que devuelve encripta el primer array de bytes pasado por parametro. Devuelve el array de bytes encriptado.
- decript(string, byte[]): Utiliza el array de bytes para llamar a genKey y con el SecretKeySpec que devuelve desencripta el string pasado por parametro. Devuelve el array de bytes desencriptado.

EncriptadorRSA.java: se utiliza para crear un par de claves, pública y privada, y para encriptar y desencriptar. Tiene 3 funciones claves:

- EncriptadorRSA(byte[], bool): es el constructor de la clase y se utilizar para crear un par de claves si el array de bytes pasado por parámetro es null. En caso contrario se creara una clave publica o una privada con el array de bytes. Que la clave que se cree sea publica o privada depende del valor del booleano.
- encript(byte[], PublicKey): encripta el array de bytes con la clave publica que se pasa por parámetro.
- decript(byte[], privateKey): desencripta el array de bytes con la clave privada que se pasa por parámetro.

Clases de interfaz

Interfaz_Choose.java: Recibe un JFrame por parámetro y lo transforma para que aparezca un texto y cuatro botones. Dependiendo del botón en el que se pulse se redirigirá a Interfaz_En.java, Interfaz_Des.java, Interfaz_Send.java o se mostraran notificaciones (esta última acción se realiza dentro de la misma interfaz).

Interfaz_Des.java: Recibe un JFrame por parámetro y lo transforma para que aparezcan dos botones por cada archivo que el usuario haya subido. Dependiendo de si clicar en uno u otro de estos botones se redirigirá a LgNeg_Des.java o se eliminará el fichero.

Interfaz_En.java: Crea un FileChooser para que el usuario pueda elegir el fichero a encriptar.

Interfaz_Login.java: Crea un JFrame con dos labels, dos inputs y dos botones. Dependiendo del botón que se clique se redirigirá a LgNeg_Login.java o LgNeg_Register.java.

Interfaz_Send.java: Crea un JFrame con dos labels, un input y un botón pro cada amigo que el usuario tenga en su lista. Al clicar en un botón o enviar el name del amigo se redirigirá a Interfaz_PickFile.java.

Interfaz_PickFile.java: Recibe un JFrame por parámetro y lo transforma para que aparezca un botón por cada archivo que el usuario haya subido. Tras clicar en uno de estos botones se redirigirá a LgNeg_Send.java

Interfaz_Autenticator: Recibe un JFrame por parámetro y lo transforma para que aparezcan un label, un input y un botón. Al clicar en el botón se redirigirá a Interfaz_Choose.java.

Clases de lógica de negocio

Debido a la amplia lista de acciones que se realizan en cada una de las clases, simplemente se va a nombrar su propósito general. Si se desea saber con más detalle cómo se realiza cada proceso, se explica en los apartados Autenticación de usuarios, Encriptación de datos, Desencriptación de ficheros y trasferencia de ficheros.

LgNeg_Des.java: Se utiliza para desencriptar un fichero dado su id y el id del usuario.

LgNeg_En.java: Se utiliza para encriptar un fichero dado su path y el id del usuario.

LgNeg_Login.java: Se utiliza para la verificación de un usuario dado su nombre y contraseña.

LgNeg_Register.java: Se utiliza para el registro de un usuario dado su nombre y contraseña.

LgNeg_Send.java: Se utiliza para el envío de ficheros de un usuario a otro.

Librerías utilizadas

Las librerías que se van a nombrar son exclusivas del lenguaje de programación java pero posiblemente tendrán un equivalente en otros lenguajes. Para el uso de algunas de las librerías que se nombrarán a continuación es necesario la instalación de un software externo llamado Maven, aunque hay otras opciones.

Interfaz

- java.awt.BorderLayout;
- java.awt.event.ActionEvent;
- java.awt.event.ActionListener;
- java.awt.Color;
- javax.swing.ImageIcon;
- java.awt.EventQueue
- javax.swing.border.EmptyBorder
- javax.swing.JScrollPane
- javax.swing.JButton;
- javax.swing.JFrame;
- javax.swing.JLabel;
- javax.swing.JPanel;
- javax.swing.JTextField;
- javax.swing.BoxLayout;
- javax.swing.JOptionPane;
- import java.awt.Component;
- javax.swing.JFileChooser;
- javax.swing.UIManager;
- javax.swing.filechooser.FileNameExtensionFilter;

Seguridad

- javax.crypto.spec.SecretKeySpec;
- javax.crypto.Cipher;
- java.security.KeyFactory;
- java.security.KeyPairGenerator;
- java.security.spec.X509EncodedKeySpec;
- java.security.spec.PKCS8EncodedKeySpec;
- java.security.KeyPair;
- java.security.PrivateKey;
- java.security.PublicKey;
- java.security.SecureRandom;
- org.bouncycastle.crypto.params.Argon2Parameters;
- org.bouncycastle.crypto.generators.Argon2BytesGenerator;

Conexión

- java.net.URI;
- org.apache.http.HttpEntity;
- org.apache.http.client.methods.CloseableHttpResponse;
- org.apache.http.client.methods.HttpGet;
- org.apache.http.client.methods.HttpPost;
- org.apache.http.client.methods.HttpPut;
- org.apache.http.client.methods.HttpDelete;
- org.apache.http.entity.StringEntity;
- org.apache.http.impl.client.CloseableHttpClient;
- org.apache.http.impl.client.HttpClients;

Excepciones

- java.net.URISyntaxException;
- java.io.IOException;
- java.security.NoSuchAlgorithmException;
- java.security.InvalidKeyException;
- java.io.UnsupportedEncodingException;
- javax.crypto.BadPaddingException;
- javax.crypto.IllegalBlockSizeException;
- javax.crypto.NoSuchPaddingException;
- java.security.spec.InvalidKeySpecException;
- org.json.JSONException;

Ficheros

- java.io.InputStreamReader;
- java.io.Reader;
- java.io.File;
- java.io.ByteArrayInputStream;
- java.awt.image.BufferedImage;
- java.io.FileOutputStream;
- java.io.InputStream;
- javax.imageio.ImageIO;
- java.io.FileInputStream;
- java.nio.file.Files;
- java.nio.file.Paths;
- java.io.ByteArrayOutputStream;
- java.io.BufferedInputStream;
- com.google.zxing.MultiFormatWriter;
- com.google.zxing.common.BitMatrix
- com.google.zxingclient.j2se.MatrixToImageWriter

Auxiliares

- java.util.Arrays;
- org.json.JSONObject;
- org.json.JSONArray;
- java.util.Base64;
- java.nio.charset.StandardCharsets;
- org.apache.commons.io.FileUtils;
- com.google.zxing.BarcodeFormat
- org.apache.commons.codec.binary.Base32
- org.apache.commons.codec.binary.Hex
- de.taimos.totp.TOTP

Métodos criptográficos utilizados

Descripción general del proyecto

Es esta última práctica se hemos implementado un sistema que registra y autentica usuarios con un segundo factor de autenticación. Además, estos usuarios pueden subir archivos desde su ordenar personal, descargar los mismos desde un servidor y compartir esos archivos a otros usuarios.

Para explicar cómo hemos abordado este problema dividiremos la explicación en 3 pasos:

- Autenticación de usuarios
- Encriptación de datos
- Desencriptación de datos
- Trasferencia de archivos

En todo momento se irán poniendo ejemplos de datos reales de la base de datos para que sea más comprensible el uso del servidor.

Para el correcto entendimiento de los procesos de encriptado utilizaremos la siguiente nomenclatura:

- Contraseña original: contraseña introducida por el usuario (Pwd_ORI)
- Hash: array de bytes aleatorios de tamaño 16.
- Contraseña hasheada: contraseña calculada por la función hash(Pwd_HASH).
- Fichero original: fichero seleccionado por el usuario (FILE_ORI)
- Fichero encriptado: fichero cifrado simétricamente (FILE_ENCRIPT)
- Fichero desencriptado: fichero una vez descifrado (FILE_DECRIPT)
- AES: método criptográfico simétrico.
- Secreto: array de bytes aleatorios de tamaño 16.
- Clave AES: clave simétrica generada a partir del secreto para cifrar un archivo. (keyAES)
- RSA: método criptográfico asimétrico.
- Clave publica: clave publica utilizada para encriptar. (publickKeyRSA)
- Secreto encriptado: array de bytes encriptado. (SECRET_ENCRIPT)
- Clave AES: clave simétrica generada a partir del secreto para cifrar la clave privada RSA. (keyAES_RSA)
- Secreto: array de bytes de tamaño 16. (SECRET_RSA)
- Clave privada encriptada: clave privada cifrada con el algoritmo AES. (privateKeyRSA_ENCRIPT)
- Clave privada desencriptada: clave privada cifrada con el algoritmo AES. (privateKeyRSA_DECRIPT)
- Clave de autenticación: clave utilizada para Google Autenticator descifrada con el algoritmo AES (keyAuth_ENCRIPT)
- Clave de autenticación desencriptada: clave utilizada para Google Autenticator descifrada con el algoritmo AES (keyAuth_DECRIPT)

Autenticación de usuarios

Tanto en el proceso de registro como en el de login tienen algunos pasos en común.

- Primero se comprueba que los campos name y password no estén vacíos.
- Seguidamente se comprueba que cumplen la longitud mínima, que el caso del name es de 4 caracteres, mientras que en el del password es de 8.
- El siguiente paso se realiza por separado, y es que se comprueba que el name solo contenga letras del albabeto inglés y números, mientras, que por otra parte en el password se comprueba que contenga al menos una mayúscula, una minúscula, un carácter especial y un número. Y por supuesto que los caracteres sean del alfabeto inglés.

Proceso de registro

Una vez comprobado que la contraseña y el nombre son correctos, llevaremos a cabo los siguientes pasos.

Cálculo del hash de la contraseña

A la contraseña original (Pwd_ORI) le añadimos un Salt, y a la cadena resultante aplicamos una función hash, que en nuestro caso es Argón2 . Esta función admite 3 parámetros que permiten al programador especificar, sin entrar en muchos detalles, el tiempo que se va a tardar en hashear la contraseña. Los parámetros utilizados en nuestro sistema son los siguientes:

- Version: ARGON2_VERSION_13
- Iterations: 3
- MemoryAsKB: 262144
- Paralelism: 1

En el dispositivo que hemos estado probando el programa cada hash tarda aproximadamente 2 segundos en calcularse.

Generación de claves RSA

Comenzaremos creando un par de claves, una pública (publickKeyRSA) y una privada (privateKeyRSA), y encriptamos la clave privada con el algoritmo AES.

Ahora generaremos un secreto dividiendo la contraseña hasheada (pwd_HASH) en dos arrays de 128 bits cada uno. Uno de estos arrays será el secreto (SECRET_RSA) con el que obtendremos la clave AES (keyAES_RSA).

Aclaración: La función hash, ARGON2, siempre devuelve 256 bits independientemente de la longitud de la contraseña a hashear.

Aclaración: 16 Bytes = 128 bits.

Una vez tenemos el secreto, simplemente creamos una clave AES (keyAES_RSA) con el mismo. Ahora con esa clave AES podemos cifrar la clave privada (privateKeyRSA), obteniendo la clave privada cifrada (privateKeyRSA_ENCRIPT) para subir esta última al servidor.

Generación de la clave Google Autenticator

Ahora crearemos una contraseña totalmente aleatoria de 20 bytes la cual codificaremos en base 32. Esto se realiza de esta forma porque es el tipo de contraseña que necesita Google Autenticator. Esta contraseña se cifra de igual modo que la clave privada del proceso anterior.

Ahora ya tenemos todos los datos necesarios para realizar una petición al servidor y crear a un usuario.

Se envía la información con el siguiente formato JSON:

```
{
        “name” : name,
        “password” : password,
        “salt” : salt,
        “publicKeyRSA”: publiKeyRSA,
        “privateKeyRSA”: privateKeyRSA,
        “keyAuth”:keyAuth
}
```

Esta información se envía en el body de una petición POST a la siguiente ruta: *http://localhost:3000/user/register*

Cuando al servidor le llega esta petición hace una llamada a la base de datos mongodb, concretamente a la tabla users, buscando en la propiedad name un valor que sea igual al valor proporcionado en req.body.name.

En el caso de que la petición devuelva un valor esto significará que ya existe un usuario con ese nombre y por lo tanto devolverá un JSON con la siguiente información:

```
{
        “result” : “KO”,
        “err” : “Invalid UserName”
}
```

En el caso de que la petición a la base de datos no devuelva nada, se guardará la información del usuario, además de un id único que crea la base de datos y se deolverá un JSON con el campo “result” : “OK”.

Si todo se ha resuelto de forma exitosa el usuario pasará al proceso de Segundo Factor de Autentificación.

Proceso de Login

Al igual que el proceso de registro es necesario obtener el hash de la contraseña, pero no es tan sencillo como generar cualquier salt, porque recordemos que el salt es aleatorio es decir nunca va ha ser la misma contraseña hasheada (Pwd_HASH), aunque las contraseñas (Pwd_ORI) sí sean idénticas.

Es por ello que una vez realizadas las comprobaciones de la contraseña y el nombre de usuario, se envía una petición GET a la siguiente ruta: *http://localhost:3000/user/salt/:name* ,siendo name el nombre del usuario.

Cuando al servidor le llega esta petición ejecuta una sentencia en la tabla users de la base de datos buscando que el atributo name coincida con req.params.name. En el caso de encontrarlo devuelve el salt asociado a ese usuario. En el caso de no encontrarlo devolverá un JSON con el campo “result” : “KO”

Una vez recuperado el salt podemos hashear la contraseña de la misma forma que en el proceso de registro.

Tras esto se envía el siguiente JSON:

```
{
        “name” : name,
        “password” : password,
        “salt” : salt
}
```

Siendo name la variable donde hemos guardado el nombre usuario, password la variable donde hemos guardado la contraseña hasheada y salt el salt utilizado.

Esta información es enviada en el body de una petición POST a la siguiente ruta: *http://localhost:3000/user/login*

El servidor ahora realizará una búsqueda en la tabla users con el campo name del body. Cuando lo encuentre recogerá el hash asociado a ese usuario y lo comparará con el hash enviado en el body.

Si son idénticos significa que el proceso de autenticación ha sido un éxito y devolveremos un JSON con el campo “result” : “OK”. En el caso de no ser correcto devolveremos también un JSON pero con el campo “result” : “KO”.

En el caso de obtener el campo “result” y que sea “OK” pasaremos al proceso de Segundo Factor de Autentificación.

Segundo Factor de Autenticación

En este proceso lo primero que haremos será recoger la clave de Google (keyAuth_ENCRIPT) de la base de datos realizando una petición a *http://localhost:3000/user/:id* , siendo id el id del usuario. Esta ruta devolverá todo el usuario y de ahí recogeremos la clave que se guardó como keyAuth.

Tras recoger la clave de autenticación la desencriptaremos con la mitad de nuestras contraseña hasheada (Pwd_HASH).

El siguiente paso es comprobar si el usuario ha realizado un registro o un login. En el caso de que el usuario se haya registrado tendremos que mostrar un código qr para que lo pueda escanear y empezar a recibir los códigos en Google autenticator.

Tras este proceso el usuario verá una ventana donde tendrá que introducir el código de verificación de Google autenticator, independientemente de si ha realizado un login o un registro.

Para concluir este proceso, una vez llegados a esta parte, en la base de datos encontraríamos usuario tendría la siguiente información:


```
{
        "_id": "6394a86f70f349f7fcf74e20",
        "name": "godo1",
        "password": "/py95CIvmKST3SBYyaTF7r7PVCAUC2XEr6Gn5DXzEoc=",
        "salt": "4HEg/F9difblLzfOBwdAQw==",
        "publicKeyRSA":
"MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAqSE8BvfCJ+UHnCNb3H/sYEMgf5m+yBC6YCpCyKDFrHrOpY
Tk0bl+xZU0nEj/40NHkSaBKXlpr+9TTbg/Jl8nHdK18anvSinF0enW38eciQAuqcv0uNN5Q67uLrH3VkHGydL1bNIiM/UIdz
+BvVX/MxXzVL9Kv+CAIyOX4b603RCeLfZhpZJqvtPz/eEhYIJwVaffUvk6khW6u0e8ToOVfjX6S2GqHyy7fsZEQWxkOQmh
TwoCCU9bQmkWe4yZHNswkMe9Mq20CRDCmTj/EXbn1uPvzcOz7FJ/5o1hQ3Itwo3e+iepusj3yZzaKe6JAEPG6bUQFpf
wHd33dJ2HCmfBkQIDAQAB",
        "privateKeyRSA":
"YyWeuJ2OtmoOZ9zfUg4Tf3FVxim4hbeqGW5YP0Kozt4k/1h4R2Rsl4g2JY2NeOU0qYeMD/CsbGVh5lXFHQyzwZBrGTK
qqGx2g3FdC9EVuY0vKepVIGylBS1MfQrU7DEyuCYetgHT10946pJaw9tuOmPI9T8cLktAYK3/wNukyXfo51PlJnsnOHdW
ESK7ErsGEqXu67bsW3XBK1nwqMYsBO9CRlLJaPl46SSvgm1eQlHl5EpHT1Cmg/RUCxEZb7MVkVLpb3TCDfzgYNXL3MU
NKhi4Sikz5g1zLnpl36inuWEJa/5mbdDeKtgldD5Qva3zyY49NEynB1Bri8Gf6F/Yo1qzQJ93iiXFhe21kjPNyYQKGXTPs2qSc
WmSz/0B9HJ0MQbepAti7TV0uGhIjbYTCxGGd7ZyHoKFDXpeH8BRzdnb6OnJQYsOmtjl5oaJ5+wsaNV8YL+qSxK5dVQ7k
YN2dXjvp0vRnWvyWjLIZtAh7VdVd5H5l57ceFIDVx3g0B4IELBQyubXgUnq5QCFq0HHhWlAecRGwSsn72BYdutNqypyQ
7QsplQqAdOPixHOdrBHoNdFBIIl85dP/QwP5Ot972PtxzOy/goVZu/eo1TSsbuRvTYGomRlz08u5I1tE6yTP2m+b0EA94vp
70/k+Mp9R0lMcvHrCZOmiicMclgGEAIJHhU8zLf5MW64ii2yE5/h/DLM4Llh48GhQgc202TNZKBiWx2YjJuT/h/vUIHmAk
8u0xt/vLTza2lxWHL1Sz3hl/MRp9Ou4qg58/vF9250dT/uBOVXXmXpSiAWnjNpAYG6MazExVYkzE0qmeLac1wLFS1m+N
2ZJmbYrzn/BYtiWdsZWmdZBiKJIm1F1RVDK68o4bf0FDq0BDEFbe0yTdgvXbB0HyKdwmH5M+7wmbMm42tn8pfy8GP
UtSOeBCmelqah9tsrYZJGz3qm69Qle9UzP6XU8nXzX+id0vJHlWNBSB50c6O2cBeWRMHpqs+K9igM/0hVwmJDOJFGzK
W412PPFCJUo0gWcA+lcI0j2HD9MrmYHkHOR8W96C8bEXx7CLZIZaeMxfQlWQH9TL7Tsd6LO4Eyl4FJ/UIpCCfw2fHPQ
D8Gmy1a+cYyeWI5gYNQTsFkByZ7SCkpCYvNCXbeISHoMywGrvZX0RXwaCGJR4tGjBGFgipqsXErvui+0jWRj5N4bXwz0O
Qj13vqD55txhllorvG5YCXQa4MbDRJ0XIaA/lF3VbXa1TD3JcSd7WlsyAQaHbg7KLJ44VBwVcs8ePm9kh8ciIfmOYRpQacN
SlVL25rtKIx96+qh2wOy9yScDt4dNCKewbqDHasWudp2LR470uN+q9gBTrz8de9yJDN/yCGoQE9RMfakShzZXEvS6MqE
yEaxnOEjGCCN9SgKmjl3X1dEh4zJfTWPeB/jEV4jKruY9hj4lIswcCb1St8zarswt8mec/OyRTibGgqlVSI/PSW8PKZARRRy2X
ksFyIqnt3CAmFgnLotMDW9qYyl1c2l1YkN6slJxInUlhE1kg+KQsChkWiEajodnM0A1Sei0hk2Z+f1eXNvA5HEPNmmMHU
yKo0Nz/6zjrGlbCJtxmywZmWnLPhVPjRqq7HDALUVPJlS0X1zMi+KOW7jopDZ6g=",
        "keyAuth": "MLATOYANIHDHNWOQNFZTGX3KXXM7TUDUEYNJTYMADHY5TOVTXRCA====="
}
```


Este ejemplo es un usuario real creado en una de las pruebas realizadas.

Encriptación de datos

Para la encriptación de datos comenzaremos creando una serie de bytes aleatorios a los cuales denominaremos secreto. Para la creación de estos bytes utilizaremos el objeto Securandom, cuyo método *.nextBytes()* da esa aleatoriedad que necesitamos.

Para ser más exactos hemos utilizado un array de bytes de tamaño 16. Esto se hace para crear una clave AES de 128 bits (keyAES) con ese secreto, que recordemos es aleatorio para cada archivo, por lo tanto, la clave AES también es diferente para cada archivo.

Con la clave AES ya creada, podemos cifrar el archivo original (FILE_ORI) con la misma (keyAES), obteniendo entonces el archivo cifrado (FILE_ENCRIPT).

Ahora se plantea el problema de qué hacer con el secreto, ya que no se puede guardar en claro en el servidor porque cualquiera que accediera al fichero encriptado (FILE_ENCRIPT) podría obtener, también, el secreto y descifrar el fichero. Por lo tanto, tenemos que cifrar el secreto. Para cifrar el secreto vamos a utilizar el algoritmo RSA.

Lo que haremos será recoger la clave pública del usuario realizando una petición get a la ruta: *http://localhost:3000/user/publicKey/:id*, siendo :id el id del usuario

Aclaración: En el caso de no recogerlas lo que haríamos sería crear dos claves nuevas y remplazar las que haya en el servidor.

Una vez recogida la clave pública (publicKeyRSA) tendremos que utilizar el método getString del objeto json, para pasar del json que devuelve el servidor a string, y la sentencia BASE64. getDecoder().decode() para pasar de string a byte.

Teniendo la clave publica como un array de bytes podremos utilizar el objeto X509EncodedKeySpec para obtener la clave pública como tal.

Ahora simplemente tenemos que utilizar la clave pública (publicKeyRSA) ya en claro para cifrar el secreto AES.

Una vez tenemos todo encriptado podemos subir los datos al servidor.

Primero haremos una petición POST a la ruta: *http://localhost:3000/files* con el siguiente JSON pasado en el body:

```
{
        “name” : “nombre del archivo”,
        “extensión” : “tipo del archivo”,
        “data_en” : “archivo encriptado como array de bytes”,
        “keyAES” : “(SECRET_ENCRIPT)”,
        “userId” : “id del usuario que ha subido el archivo”
}
```

Esta petición, en el servidor, lo que hará será crear y guardar un archivo en la base de datos con la siguiente información:


```
{
        “_id” : “636947e65aa23761d88d78da”,
        “name” : “Fichero1.txt”,
        “date” : “Nov 07 2022 19:01:10”,
        “extensión” : “txt”,
        “data_en” :
        “Ml8YGX2Q1MAHZIMN1R6f6v+CHkvL2ned1USd5kkk6TBKnqq+97XOICHPZV5qy+Mo2qDLmOwP
        QQDyVWXdY49PnG7OZA19a4DYrNkK/n6cFJ5UB9pyNJuEbCfmdiKDL+SMschFBEGio2yeoUzVsAiY
        TffuoN335EPZztck3CQ/Bp4eYy0KsmbUzmO7AxfSsFC8BcqARdt9Pe7Cw64kfW6k4jhi9wM/nbKy40
        Zuy3W4uFJcdki9vUoHlI51uT9cCZMhVS6PNT7FnF09PYfEEkdnYMJTBgTrgLCrmTrQr2Nc8B28IM/6S
        woswSPXs/0/lUhms4qghD51f+OgHPS+fQPL9DJfGBl9kNTAB2SDDdUen+ot0uO7FFNtt9t4APj6uk0
        uFmFV7548AeOi6HEoy67NyEwdCHp7oU+JtVtGZK38jJGCpr6qWnkfCwhTsrXaz7pD/iAvdHnOROvt
        RME6/VK4sFJmzkoFnb/n72uqmxn96kT8ic5Bgnr/v+fu+20GQqTiMU9S0B9iJMrUrhyO+Nrb7shPq4I
        rNjbiaiXzoUbPUNa5I9+m34fbidLvQBKO/f7CLb9aMsO9TAPqPb/zL0VFQOds6cMXMqWLnOs5jkjw
        27gm5+jpGZTpxQBEZO2OcWLTIfwM4umghAEMS4HQN+Ve1A31o9uXOxevYalnV4csVX3ILIWc2M
        5vpYRMCiCO8Sjt559ES0KeV6S9LsEtbZXi7yOU+j0oMKjMvJYCCdWG5RMV4DhC0JXmNiZyxc/TpHY
        chSEmSGex+8tt0b+zqb26jg==”,
        “keyAES” :
        “W4woVkNO42SUcvBUeGT+3ZZuCENRV41D19oeeXoooBdCZMF50qbNjJuMzLFMqdZLM47exQvf
        bqjwhHU02U4vzgsr9uWcgIrLhQnwU8NYUZgMkGFWLWvmshoR8MMKDbMu3Sbqehm9DnNj6BJ
        N3AW5232UcFaW87hAQk8TGFu3ci+8bRj/Kl9s7hDwfFtFFcMIIhCP3Kyw4fEFUNQX2F8cOlSIILwC6
        KQWOUs6jPUZu6FhY2cEuj7SFX6R7Zh8W/BWFHrmIZPFlhcJXqr9/Hqcz6N5b48vlq2jvFEQ/ejbg0/s
        QylPo8VpKQwcIzkiqhyoEzs5/uhdvS9+7cG5mMwh4A==”,
        “userId” : “636947b15aa23761d88d78d8”
}
```


Este ejemplo es un archivo real creado en una de las pruebas realizadas.

Desencriptación de ficheros

El proceso de desencriptación cosiste, a grandes rasgos, en recoger los datos almacenados en la base de datos y obtener el archivo original desencriptado (FILE_DECRIPT).

Para ello lo primero que haremos, después de pasar por el proceso de login o de registro, será recoger el id del fichero que queremos obtener clicando en uno de los botones que aparecen en la interfaz.

Aclaración: Todo el proceso de la construcción de la interfaz no se va ha comentar dado que no tiene que ver con el proyecto o la asignatura.

Una vez obtenido el id del fichero a desencriptar lo que haremos será realizar una petición get a la ruta: *http://localhost:3000/files/:id*, siendo :id el id del fichero.

A su vez realizaremos otra petición get a la ruta: *http://localhost:3000/user/:id*, siendo :id el id del usuario, que lo obtenemos como respuesta al proceso de login o registro.

El siguiente paso será, con la mitad del hash del usuario, obtenido en el proceso de verificación de usuario, crear la clave AES (keyAES_RSA) que nos permita descifrar la clave privada obtenida del servidor.

Una vez tenemos la clave AES (keyAES_RSA), desencriptamos la clave privada encriptada (privateKeyRSA_ENCRIPT). Con la clave desencriptada como array de bytes, utilizamos el objeto PKCS8EncodedKeySpec para obtener la clave privada como tal.

Ahora, teniendo la clave privada (privateKeyRSA_DECRIPT) podemos descifrar el secreto encriptado (SECRET_ENCRIPT) para crear la clave AES que nos permita desencriptar el fichero encriptado.

Cuando hayamos creado la clave AES (keyAES) podemos descifrar el fichero cifrado (FILE_ENCRIPT) y así obtener el fichero desencriptado (FILE_DECRIPT).

Trasferencia de ficheros

Para el proceso de envío de ficheros se combinan las técnicas tanto de descifrado como de cifrado de archivos. Se explicarán por separado el proceso de envío de ficheros y el de recogida de los mismos.

Envío de ficheros

El primer paso es obtener el id del fichero a enviar y descifrarlo con el mismo proceso que el punto anterior Desencriptación de ficheros.

Una vez tenemos el archivo descifrado, tenemos que realizar una petición get a la ruta: *http://localhost:3000/user/:id*, siendo :id el id del usuario al que le vamos a enviar el fichero, que lo obtenemos como respuesta de la interfaz.

El paso anterior nos devolverá los datos del usuario al que le vamos a enviar el fichero. De todos los datos lo que nos interesa es su clave pública, que la usaremos para cifrar el archivo descifrado de la misma forma que se realiza en el proceso de Encriptación de datos.

Una vez se realice esta parte del proceso, el usuario original no tendrá acceso al fichero original si no realizamos una copia del mismo.

Aclaración: No se realiza una copia literal, sino que cuando se realice la petición al servidor se hará un post y no un put, ya que esto genera un nuevo archivo con otro id y no se sobreescribe el orginal.

Pasado el proceso de encriptación realizaremos una petición post a la ruta *http://localhost:3000/files* pasando en el body el fichero de la misma forma que se pasa en el proceso de Encriptación de datos

, pero con un único cambio, el campo userId no añade en el body.

Aclaración: No se añade el campo userId porque hasta que el usuario que recibe el archivo no acepte el archivo, este archivo no debe de aparecer en su lista de archivos.

En esta parte del proceso el archivo nuevo ya esta guardado en la base de datos, solo que sin dueño.

Ahora crearemos un objeto json con los siguientes datos:

```
{
        “userSender”: userSenderId,
        “userReciber”:userReciberId,
        “file”: fileId
}
```

Este objeto json se envía al servidor en el body de una petición post a la ruta: *http://localhost:3000/notifications*. Esto guarda en la tabla notificaciones un objeto como lo siguiente:

```
{
        "_id": "6394aa4a70f349f7fcf74e26",
        "userSender": "6394a9f370f349f7fcf74e22",
        "userReciber": "6394a1f6bb9913cd8c71c65a",
        "file": "6394aa4a70f349f7fcf74e25"
  }
```

Este proceso se realiza para que quede constancia del envío y sea fácil de referencial a ambos usuarios y al archivo.

Antes de guardar la notificación en la base datos el propio servidor comprueba que el mismo usuario que envió el archivo no pueda enviar más de 5 archivos al mismo usuario que los recibe. Si ese caso se da el archivo que ya estaba guardado en la base de datos, es eliminado.

Aquí termina el proceso de envío de un fichero. Ahora se explicará el proceso de recogida del mismo.

Recogida de ficheros

El proceso de recogida de ficheros se comienza en cuando termina todo el proceso de autenticación.

Lo primero que se realiza es una petición a la ruta *http://localhost:3000/notifications/number/:id*, siendo id el id del usuario que recibe las notificaciones. Esta ruta devuelve el número total de notificaciones que el usuario tiene.

Si ese numero es 0 simplemente termina el proceso. En el caso contrario se piden todos los datos de todas las notificaciones y se crean dos dialogs por cada notificación.

El primero dialog pregunta al usuario si quiere añadir el archivo a su lista de archivos.

En el caso de que no quiera añadirlo simplemente se pasa a la ejecución de los dialogs de otra notificación.

En caso de que sí desee añadirlo a su lista de ficheros, se realiza una petición put al servidor donde se pasa el id del fichero y el id del usuario que lo recibe. Esto hace que el fichero ahora tenga un campo userId y sea igual al usuario que ha recibido el fichero (tiene dueño).

El segundo dialog pregunta al usuario si quiere descargar el fichero directamente, sin tener que ir al apartado de “request a file”.

En el caso de que no quiera, se termina la ejecución.

En el caso de que sí quiera, se realiza el mismo proceso que en el apartado de Desencriptación de ficheros.

Explicación de algunos conceptos utilizados

Salt: comprende bits aleatorios que se usan como una de las entradas en una función derivadora de claves.

Argón2 es una función de derivación de clave que fue seleccionada como ganadora del concurso de descifrado de contraseñas en julio de 2015.12 Fue diseñada por Alex Biryukov, Daniel Dinu, y Dmitry Khovratovich de la Universidad de Luxemburgo.3 La implementación de referencia de Argon2 se publica bajo una licencia Creative Commons CC0 licencia (es decir, de dominio público) o la licencia apache 2.0, y proporciona tres versiones relacionadas:

- Argon2d maximiza la resistencia a los ataques de cracking de la GPU. Accede al arreglo en memoria en un orden dependiente de la contraseña, lo que reduce la posibilidad de situación de compromiso espacio-tiempo (TMTO), pero introduce posibles ataques de canal lateral.
- Argon2i está optimizado para resistir los ataques de canal lateral. Accede al arreglo en memoria en un orden dependiente de la contraseña.
- Argon2id es una versión híbrida. Sigue el enfoque de Argon2i para la primera mitad del paso sobre la memoria y el enfoque de Argon2d para los pasos siguientes. The internet draft4 recomienda el uso de Argon2id excepto cuando hay razones para preferir uno de los otros dos modos.

Los tres modos permiten la especificación por tres parámetros que controlan:

- Tiempo de ejecución
- Memoria requerida
- Grado de paralelismo

El algoritmo AES, también conocido como Rijndael, es una función matemática de encriptación creada en Bélgica y adoptada por el gobierno de los Estados Unidos como estándar en el año 2001. Desde entonces, se conoce como el método de cifrado por bloques más seguro que existe, ya que en la práctica no se puede romper y, además, es rápido y eficiente.

El nombre RSA proviene de las iniciales de sus tres creadores, Rivest, Shamir y Adleman, allá por 1997. Se trata de un algoritmo de cifrado asimétrico, o de clave pública, y es uno de los más utilizados en la actualidad. De hecho, la mayor parte de los sitios web hoy integran seguridad SSL/TLS, y permiten la autenticación mediante RSA.

RSA, al ser un cifrador asimétrico, trabaja con dos claves, una pública y una privada. Todo el contenido de texto plano, o contenido sin cifrar, que se haya encriptado con la clave pública, podrá ser descifrado mediante la clave privada, y viceversa, todo contenido cifrado con la clave privada podrá ser descifrado mediante la clave pública.

Enlaces de interés

Video explicativo del proyecto:
https://drive.google.com/file/d/1x3k9tghX6FfHGtjw1IlwOjFgI1ksAovp/view?usp=sharing

Creación de una clave publica desde una cadena de caracteres:
https://stackoverflow.com/questions/10900643/how-can-i-construct-a-java-security-publickey-object-from-a-base64-encoded-strin

Ejemplo de cifrado con AES y RSA:
https://stackoverflow.com/questions/10007147/getting-a-illegalblocksizeexception-data-must-not-be-longer-than-256-bytes-when

Explicación algoritmo RSA:
https://juncotic.com/rsa-como-funciona-este-algoritmo/

Explicación algoritmo AES:
https://keepcoding.io/blog/que-es-el-algoritmo-aes/

Ejemplo de encriptación y desencriptación con RSA:
https://gustavopeiretti.com/rsa-encrypt-decrypt-java/

Uso del objeto SecureRandom:
https://www.geeksforgeeks.org/securerandom-nextbytes-method-in-java-with-examples/