

**UNIVERSIDAD CARLOS III DE MADRID**

**ESCUELA POLITÉCNICA SUPERIOR**

**AÑO ACADÉMICO (2024 – 2025)**

**TERCER CURSO**

**ENTREGA 1**

**MEMORIA DE APLICACIÓN**

**Información Alumno 1**

**Nombre:** Gabriel José

**Apellidos:** Rivera Amor

**NIA:** 100477706

**Correo:** 100477706@alumnos.uc3m.es

**Grupo Prácticas:** 21

**Titulación:** Grado en Ingeniería Informática

**Información Alumno 3**

**Nombre:** Santiago José

**Apellidos:** Díaz Rodríguez

**NIA:** 100479095

**Correo:** 100479095@alumnos.uc3m.es

**Grupo Prácticas:** 21

**Titulación:** Grado en Ingeniería Informática

**Información Alumno 2**

**Nombre:** Joaquín

**Apellidos:** Pujol Carrillo

**NIA:** 100495826

**Correo:** 100495826@alumnos.uc3m.es

**Grupo Prácticas:** 21

**Titulación:** Grado en Ingeniería Informática

**Madrid 26 de octubre de 2024**

**DESCRIPCIÓN DE LA APLICACIÓN**

La aplicación que hemos desarrollado tiene como finalidad gestionar el alquiler de canchas y espacios deportivos de una determinada comunidad. Para ello, cada individuo interesado en alquilar los espacios deberá rellenar un formulario de registro, en el cual deberá proporcionar datos bancarios, su nombre de usuario, su contraseña y su nombre completo real.

Para el almacenamiento de datos, hemos utilizado archivos JSON sobre los cuales escribimos la información de cada uno de los usuarios, dado que nos parece una forma eficiente de almacenar la información según los conocimientos adquiridos en la asignatura de Desarrollo del Software.

Adicionalmente, una vez el usuario se registra, debe iniciar sesión y mediante diferentes algoritmos que explicaremos a continuación, dado un nombre de usuario se procederá a la validación de los datos y se les permitirá el acceso a los usuarios al área de la aplicación destinada al alquiler de las canchas. Es importante destacar que nuestra aplicación solo funciona como un alquiler de espacios, por lo que no existirá ningún tipo de comunicación entre los usuarios.

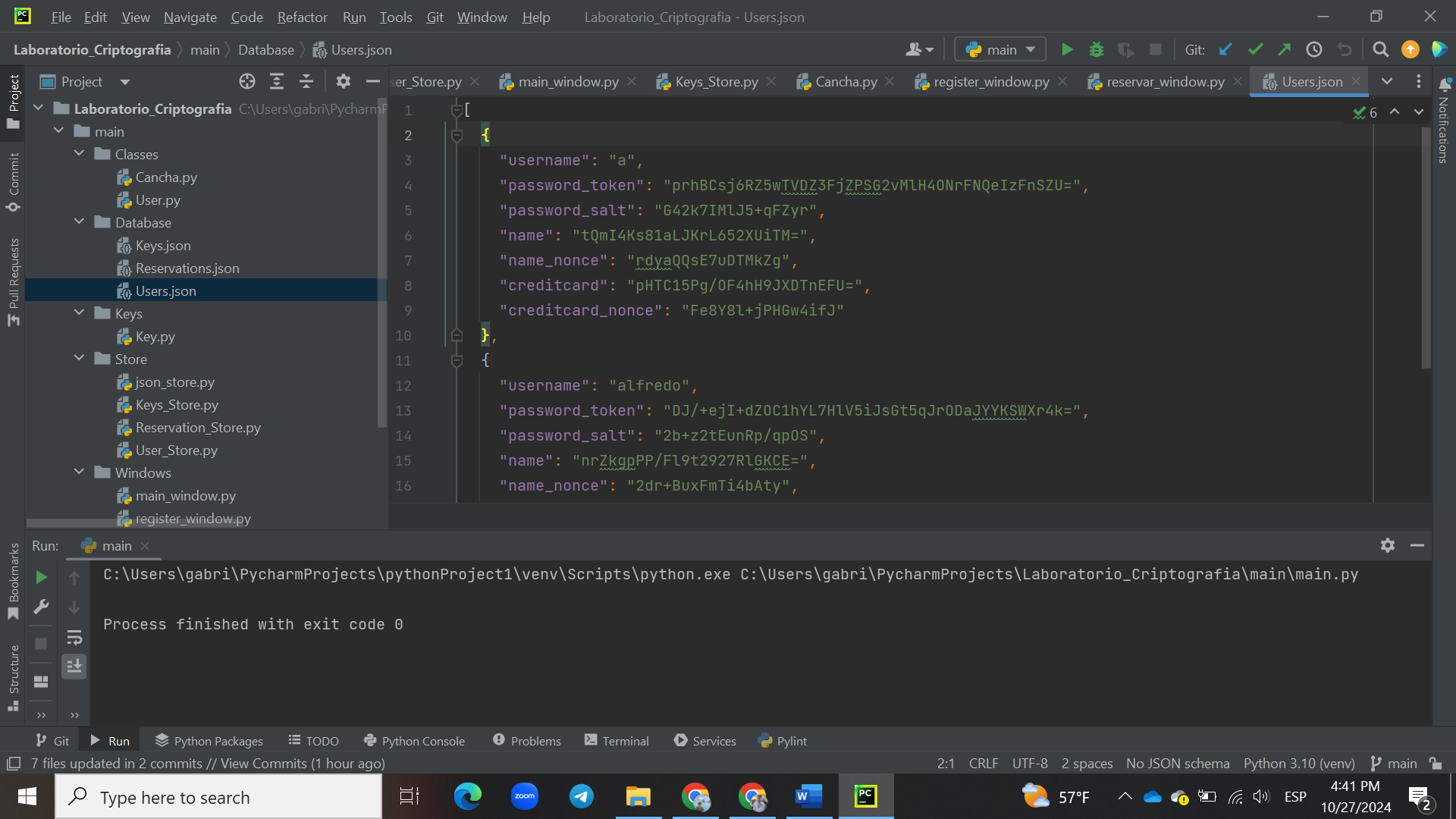
**AUTENTICACIÓN DE USUARIOS**

Primeramente, es necesario mencionar que, al momento de almacenar el nombre de cada uno de los usuarios, estos se dejan en claro dentro de los archivos de tipo JSON. Esto lo hacemos dado que el nombre de usuario no solo es un dato que no amerita ser encriptado, dado que sólo con el nombre de usuario no será suficiente para acceder al área de los datos y descifrarlos.

Teniendo en cuenta lo antes mencionado, para autenticar a los usuarios de la aplicación, nosotros mediante un algoritmo de *“Key Derivation Function”* (KDF), la contraseña introducida por el usuario se va a derivar en un token que se va a almacenar en cada uno de los JSON acompañados de su respectiva salt. Esta forma de descifrar los datos de la contraseña nos va a permitir que al momento de que al momento que el usuario desee ingresar a su área personal, la contraseña que indique deberá someterse a una comprobación con el token almacenada el JSON y que va a permitir verificar que el usuario que ingresa conoce su contraseña, y por ende queda autenticado.

Por ello, se pude decir que nosotros utilizamos una información que se encuentra almacenada en el sistema al hacer el registro y eso nos lleva a afirmar que estamos usando un algoritmo “Basado en algo que tenemos”, ya que disponemos del token y el salt generados a partir de la contraseña introdujo en el momento de registrarse. El funcionamiento específico y detalles acerca de como hacemos este proceso, lo explicaremos en el apartado de “ALGORITMO DE CIFRADO-AUTENTICADO”, por lo que ahora sólo nos ceñimos a mostrar el dato almacenado que usamos para hacer el proceso de autenticación.

Imagen 1: Almacenamiento de Token y Salt en el archivo JSON



**CIFRADO SIMÉTRICO DE DATOS DEL USUARIO**

En nuestra aplicación hemos implementado únicamente el cifrado simétrico sobre la información de cada uno de los usuarios al momento de registrarse. Como hemos mencionado anteriormente el usuario se deja en claro en los JSON, la contraseña se cifra mediante otro mecanismo que permite la autenticación al mismo tiempo que se cifra y que se explicará en el siguiente apartado. Por ende, solo quedan por cifrar el nombre completo del usuario y su tarjeta.

Teniendo en cuenta que estos datos no serán transferidos a ningún otro usuario y que a su vez no son datos compartidos, por lo que un solo usuario podrá editarlos, entonces no será necesario un cifrado asimétrico para los datos. Esto se debe a que, para descifrar los datos, es necesaria una clave, y aunque el atacante pudiera llegar a los datos, no podría descifrarlos dado que la clave no se está compartiendo, dado que no es necesario, y está almacenada con cada usuario.

Para ello, hemos decido utilizar un cifrado-autenticado simétrico del tipo ChaCha20Poly1305, el cual es un algoritmo que es rápido y seguro para infinidad de dispositivos, dado que se encuentra optimizado para arquitecturas de propósito general. También existía la posibilidad de implementar este cifrado con AES, pero dado que nuestra aplicación está pensada para también funcionar en dispositivos móviles, así como se quiere que el usuario pueda en un tiempo sumamente reducido escoger la cancha que quiera alquilar, ameritamos que el acceso a sus datos sea sumamente rápido y con el menor coste energético posible.

Captura de pantalla de computadora

Descripción generada automáticamenteTexto

Descripción generada automáticamenteComo se podrá observar a continuación, se podrá evidenciar como llamamos a la función que invoca al ChaCha20Poly1305, importada de la librería “cryptography” de Python. En ella, se podrá observar como se genera la clave por cada usuario y luego cómo se genera un *nonce* por cada uno de los datos que se cifran. Luego de invocar a la función ChaCha, hacemos el cifrado de datos y será necesario guardar la clave general del usuario y cada uno de los *nonce*, para poder descifrarlos posteriormente cuando sea necesario.

Imagen 3: Función de Cifrado con ChaCha20Poly1305 y Cifrado de Datos de Registro

Imagen 2: Implementación de ChaCha20Poly1305

Captura de pantalla de computadora

Descripción generada automáticamenteCaptura de pantalla de computadora

Descripción generada automáticamenteSin embargo, la clave del usuario que se genera por el mismo algoritmo ChaCha20Poly1305 no puede ser almacenada en claro, dado que estaríamos haciendo un esfuerzo por cifrar datos inútiles, debido a que el atacante pudiera ver la clave de descifrado en el almacenamiento. Por ello, hemos tomado la decisión de diseño, que nuestra aplicación tendrá una clave y un *nonce* propios del sistema y que servirán para que, al momento de almacenar la clave, la podamos cifrar mediante el mismo algoritmo de ChaCha20Poly1305.

Imagen 5: Cifrado de la Clave del Usuario para ChaCha20Poly1305

Imagen 4: Clave y Nonce del Sistema

Captura de pantalla de computadora

Descripción generada automáticamenteEsta operación lo que permite es guardar es la clave del usuario de una forma segura dentro de un JSON distinto a donde se guarda el resto de la información cifrada.

Imagen 6: JSON con el Username y la Clave del Usuario para el ChaCha20Poly1305

Luego la función de descifrado es similar a la de cifrado, con el pequeño detalle de que el proceso es al revés a como se hizo con anterioridad, es decir, primero se descifra la clave del usuario con la clave y *nonce* del sistema y posteriormente, se descifra la información del usuario utilizando la clave recién descifrada y el *nonce* almacenado.

Por último, es importante agregar que el algoritmo utiliza la información almacenada en bytes y luego los archivos JSON solo guardan información en bytes si están en base 64. Por ello, al momento de cifrar, convertimos la información del usuario en bytes. Luego para guardarla, pasamos la cadena de bytes que nos devolvió el algoritmo a bytes base 64 para almacenarlo. En el caso de descifrado, es el proceso inverso realizado.

**ALGORITMO DE CIFRADO – AUTENTICADO PARA CONTRASEÑA**

Teniendo en cuenta lo antes mencionado, entonces lo último que faltaría por cifrar sería la contraseña del usuario, la cual se tiene que hacer por un método distinto al de resto de datos. Es importante mencionar que todo lo que hemos hecho hasta el momento ha sido cifrado con autenticado, pero con la diferencia de que lo anterior era una información que no se utiliza para autenticar al usuario. Por ende, este proceso depende de un algoritmo más delicado que el anteriormente utilizado.

Captura de pantalla de computadora

Descripción generada automáticamentePor ende, dadas las circunstancias y lo delicado de la información que se está manejando, decidimos utilizar el algoritmo denominado Key Derivation Fuction con Scrypt, el cual genera un token y un salt para la contraseña registrada. Luego, lo que se guarda junto a la información será el salt y el token de cada uno de los usuarios, tal y como se muestra a continuación.

Imagen 7: Key Derivation Function y Almacenamiento en JSON

Este algoritmo lo que hace es cifrar la clave de tal forma que sean necesarios dos elementos para poder descifrarla correctamente y que se combinan de una determinada forma con la función *derive*. Ambos dos se almacenan en claro dentro del archivo JSON para luego determinar la contraseña al momento de descifrarla con la operación KDF *verify*.

Texto

Descripción generada automáticamenteLa operación antes mencionada, al introducirse la contraseña por parte del usuario y compararla con el token almacenado y usando el salt en el llamado a la función, entonces devolverá un *None* si se realiza correctamente o una excepción en caso de error. Por ende, al momento de autenticar debemos ser capaces de captar dicha excepción para avisarle al usuario que la contraseña es incorrecta. Todo lo antes dicho se puede evidenciar en la siguiente imagen.

Imagen 8: Key Derivation Function y Autenticación del Usuario

Una vez que se ha hecho la operación de verificación y no se ha lanzado ninguna excepción, implica que todo ha funcionado correctamente, por lo que el usuario ha sido autenticado y se comienza a aplicar el algoritmo ChaCha20Poly1305 explicado anteriormente para descifrar el resto de los datos almacenados. Por ello, podemos afirmar que este proceso que se lleva a cabo con las contraseñas en el método de autenticación de cada uno de los usuarios, ya que el hecho que la contraseña que introduce al iniciar sesión coincida con el token almacenado, implica que estamos autenticando que los datos que estamos verificando son verídicamente del usuario.

Por ello, podemos afirmar que el proceso de cifrado-autenticado es mucho más efectivo que solo cifrar, ya que no solamente estamos almacenando de forma segura los datos, sino, que también autenticamos que los datos que estamos cifrando y descifrando pertenecen al usuario que accede a la aplicación, lo que otorga mayor seguridad y confianza del usuario.

**ANEXOS**

Es importante mencionar que a lo que hemos realizado o llevado a cabo, hemos podido añadirles ciertas mejoras adicionales a los algoritmos. Una de ellas es la antes mencionada, en la cual se le especifica al usuario que la contraseña es incorrecta en el caso de que salte la excepción de la Key Derivation Function (KDF). Por otro lado, también generamos un mensaje por pantalla en el caso de que el usuario ya exista al momento de hacer el registro y otro mensaje que indique si el usuario existe o no al momento de iniciar sesión. Estos son aspectos agregados que no se especifican como parte de las funcionalidades al aplicar los algoritmos de cifrado y descifrado pero que nos parecieron importantes agregar.

***Repositorio del Proyecto:*** <https://github.com/100479095/Laboratorio_Criptografia>