



Índice

[**• ¿Cuál es el propósito de su aplicación?**](#_gjdgxs) **3**

[**• ¿Para qué utiliza el cifrado simétrico?**](#_30j0zll) **3**

[• ¿Qué algoritmos ha utilizado y por qué?](#_1fob9te) 3

[• ¿Cómo gestiona las claves?](#_3znysh7) 4

[CAPTURAS CÓDIGO](#_2et92p0) 4

[**• ¿Para qué utiliza las funciones hash o HMAC?**](#_tyjcwt) **5**

[• ¿Qué algoritmos ha utilizado y por qué?](#_3dy6vkm) 6

[• ¿Qué hash queremos utilizar?](#_ab9ap8bjm7zy) 6

[• ¿Quién crea la clave? ¿La introduce el usuario? ¿Cómo/dónde se almacena? ¿Cómo/ dónde se almacena el hash?](#_esdkduf4j5t2) 6

[• ¿De qué datos calculamos el hash?](#_ke85ahxiav8g) 6

[• ¿Qué hace que un hash sea bueno?](#_xdxrktrsm6eb) 6

[CAPTURAS CÓDIGO](#_vxcaz2ghjut) 7

[**•**](#_tyjcwt) [**M**](#_d3oqbphi8gqn)**ejoras 8**

Grupo: 12

ID de grupo de prácticas: 12

Nombre de todos los alumnos:

* José David Rico Días (100441800)
* Samuel Fernández Fernández (100432070)

------------------------------------------------------------------------------------------------------------------------------

**Credenciales para acceso a nuestra aplicación criptográfica:**

[Enlace a la aplicación](https://losladrones.josedaviddavi41.repl.co) (Si el enlace está roto enviar correo a 100441800@alumnos.uc3m.es)

**DNI / NIE:** 24769653X

**Clave:** 12345678

------------------------------------------------------------------------------------------------------------------------------

# • ¿Cuál es el propósito de su aplicación?

Nuestra aplicación es un sistema de gestión bancaria llamado "LOSLAdrones", donde tendremos diferentes funcionalidades como el registro de usuarios, el inicio de sesión, la transferencia de dinero, consulta de saldo, ver movimientos o borrar la cuenta. En esta primera entrega hemos implementado el inicio de sesión, la consulta de movimientos y el cierre de la sesión.Logotipo, nombre de la empresa

Descripción generada automáticamente

Mediante esta aplicación pretendemos crear un sistema seguro y encriptado (como expondremos a continuación) de gestión personal de cuentas.

# **•** ¿Para qué utiliza el cifrado simétrico?

Hemos utilizado el algoritmo de cifrado simétrico AES CFB para cifrar toda la BBDD usando una clave de 32 bytes.

Desciframos la BBDD únicamente cuando se hace una consulta (query, insert…) y la ciframos cuando salimos de ella. Con esto pretendemos que, si hubiese un *breach* de los datos, los datos de los clientes y movimientos no pudiesen ser accedidos. Además, si alguien intentase modificar la BBDD y colocar una “nueva”, en el proceso de descifrado toda la BBDD se corrompería por el efecto cascada del AES. Por último mencionar que la BBDD al descifrarse reside en memoria, ya que primero se crea un stream y luego se escribe a disco.

## • ¿Qué algoritmos ha utilizado y por qué?

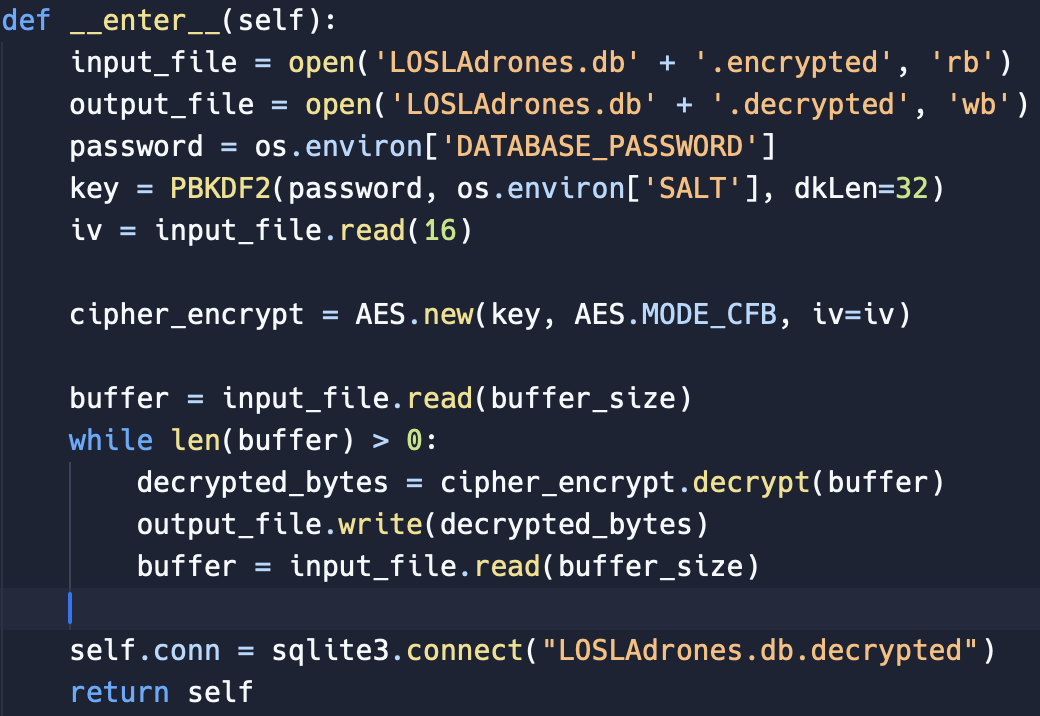
Hemos utilizado PyCryptodome con el cifrado AES tipo CFB (AES.MODE\_CFB) por el cual no es necesario utilizar *padding*. En nuestro caso nos permitía que las operaciones de cifrado/descifrado fuesen más rápidas respecto de AES CTR. AES permite que cifremos y descifremos la BBDD antes y después de cualquier operación, sin que se penalice el tiempo de respuesta de la app.

## **•** ¿Cómo gestiona las claves?

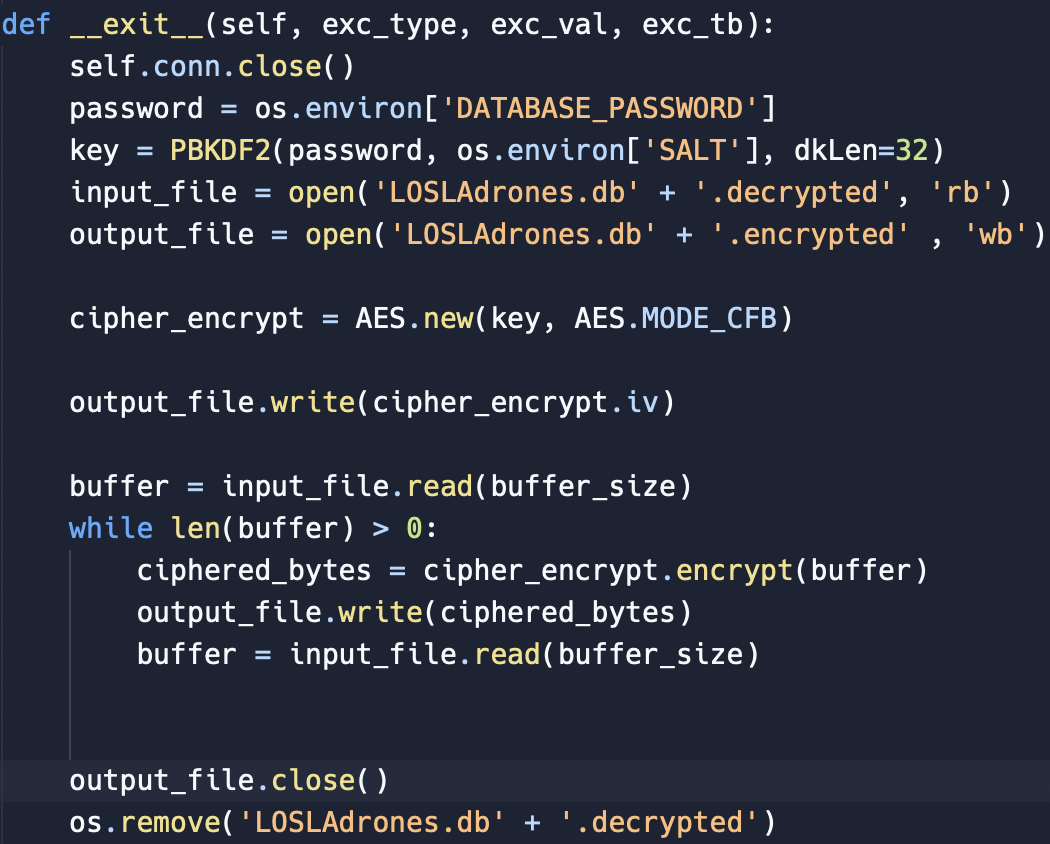
Gestionamos la key almacenándola en una variable de entorno secreta que solo el proceso que ejecuta la página web puede acceder. Concretamente usando la funcionalidad *secrets* de *repl.it*.

### CAPTURAS CÓDIGO

Desencriptado al entrar (nótese que **no** se cierran los streams para que no se almacenen en disco):



Encriptado al salir (aquí **sí** se cierra el stream cifrado y se borra el stream no cifrado):



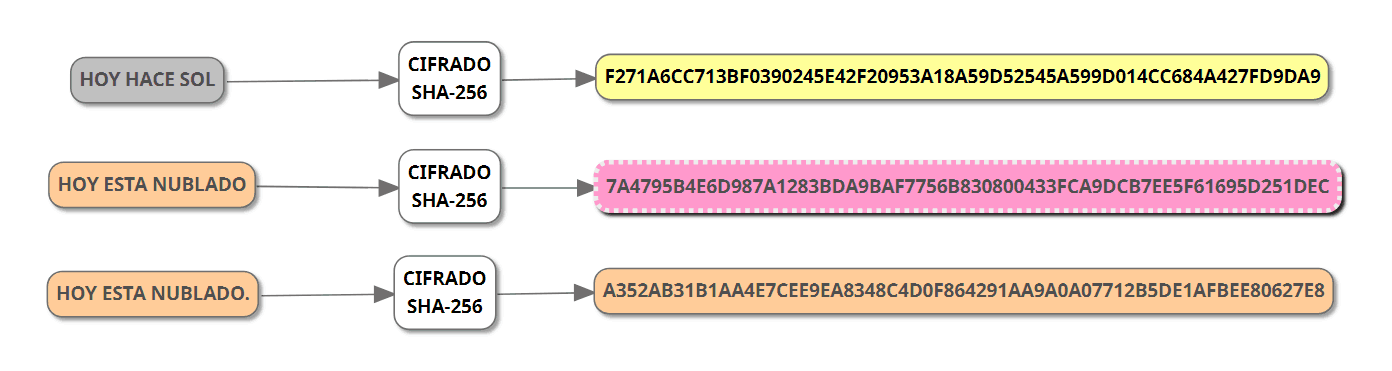
# • ¿Para qué utiliza las funciones hash o HMAC?

En nuestra aplicación hemos hasheado las contraseñas de los usuarios para una mayor seguridad. Con esta estrategia las contraseñas nunca son almacenadas en plano en la BBDD.

Adicionalmente, las hemos salteado, ya que esto nos proporciona una mayor seguridad a la hora de que alguna persona acceda a conocer la palabra que origina nuestros hashes, debido a que al saltearlas se añade un número de dígitos aleatorios ya sea al principio o al final y que el usuario no conocerá. Con esto protegemos contra ataques de *rainbow tables*.

## **•** ¿Qué algoritmos ha utilizado y por qué?

Hemos utilizado el algoritmo "SHA256" debido a que es uno de los algoritmos más utilizados por su equilibrio entre seguridad y coste computacional.



## **•** ¿Qué hash queremos utilizar?

Hemos utilizado el hash “Secure Hash Algorithm”, ya que este hash dada una cadena de bits del buffer de salida resulta prácticamente imposible hallar una cadena origen que devuelva el mismo contenido. Además, dispone de una cantidad de combinaciones muy elevada, esto puede evitar posibles duplicados de datos o colisiones las cuales podrían llegar a comprometer la seguridad de los archivos.

## **•** ¿Quién crea la clave? ¿La introduce el usuario? ¿Cómo/dónde se almacena? ¿Cómo/ dónde se almacena el hash?

En nuestra BBDD tenemos almacenados los diferentes usuarios con sus respectivos id, nombre, apellido, fecha nacimiento, género, nacionalidad, email, contraseña (hasehada y salteada) y a la hora de iniciar sesión el usuario entra en la página de login e introduce su id y su contraseña.

Las contraseñas son creadas desde la oficina presencialmente o por videollamada con un empleado (tercero de confianza) que introduce los datos a través de la aplicación de gestión de empleados. Son transferidas a nuestra BBDD periódicamente, donde las hasheamos (con la función expuesta mas abajo) y almacenamos. En nuestro caso queremos el hash únicamente para contrastar que la contraseña pertenece a ese usuario: para ello nos valemos de check\_password\_hash(), incluida en la misma biblioteca werkzeug.security.

## **•** ¿De qué datos calculamos el hash?

Con el generate\_password\_hash() incluido en el paquete werkzeug.security creamos el hash de las contraseña. Se hace a partir del texto en claro de la contraseña de *login*.

## **•** ¿Qué hace que un hash sea bueno?

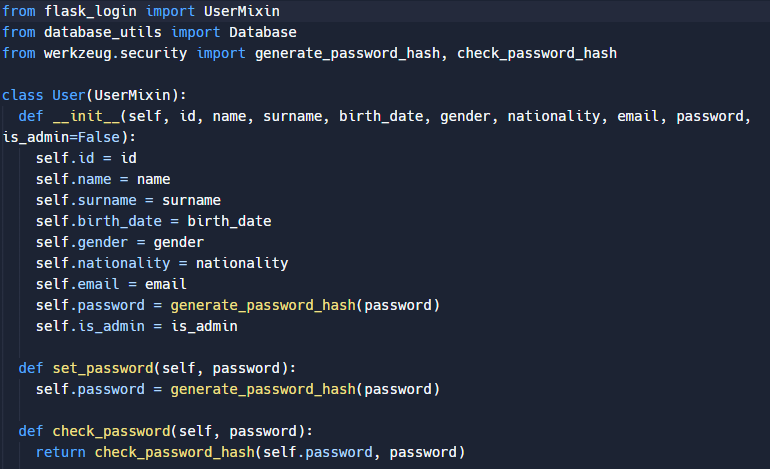
Que sea comprensible (una cadena de 64 caracteres), que sea fácil de calcular (no requieren grandes potencias de cálculo para calcularse), que funcione de forma de que si se realiza algún mínimo cambio se genera un hash diferente, que sea imposible de calcular mediante otro hash, y que sea irreversible

### CAPTURAS CÓDIGO

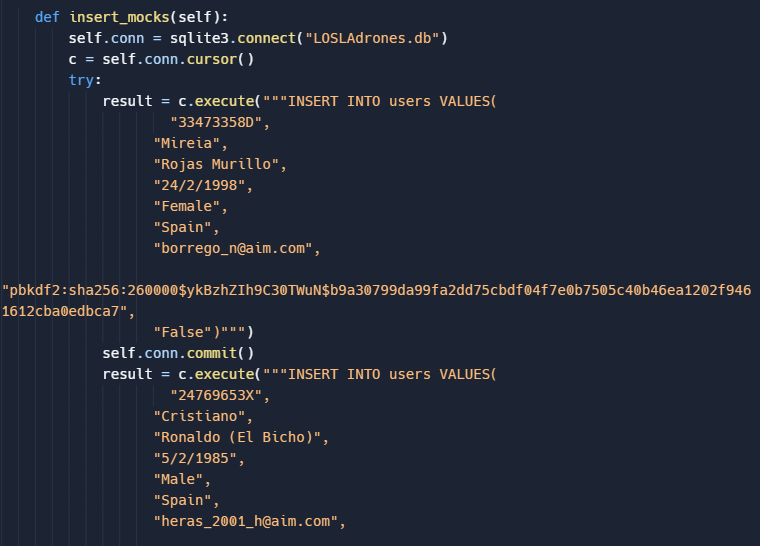
Ejemplo de contraseña hasheada y salteada:



Creación de hash:

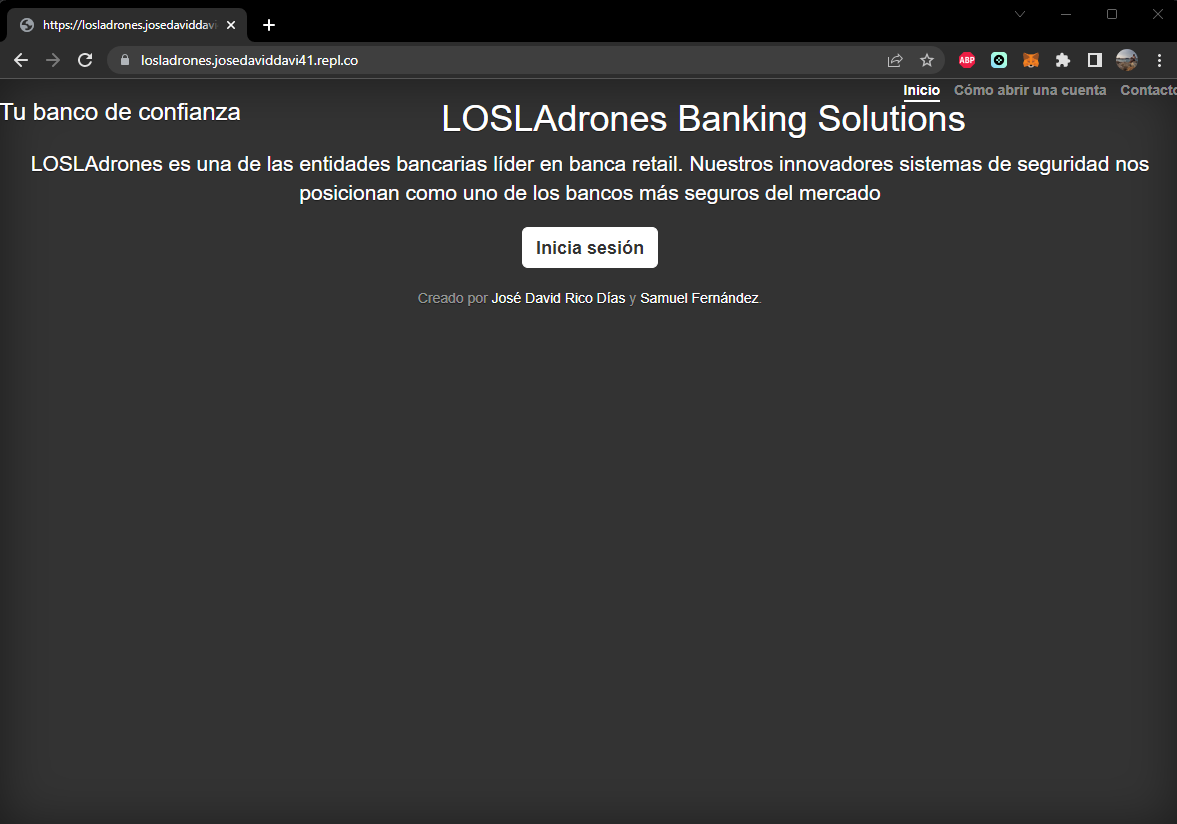
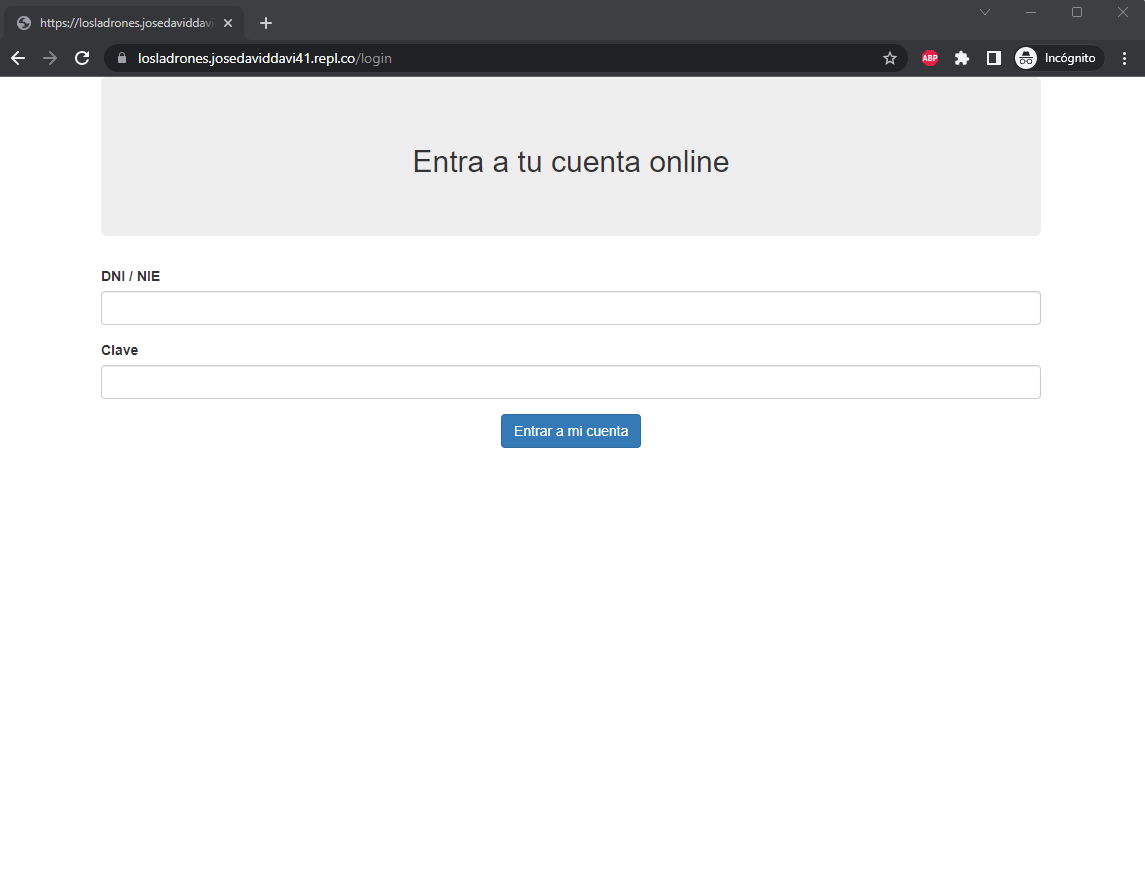


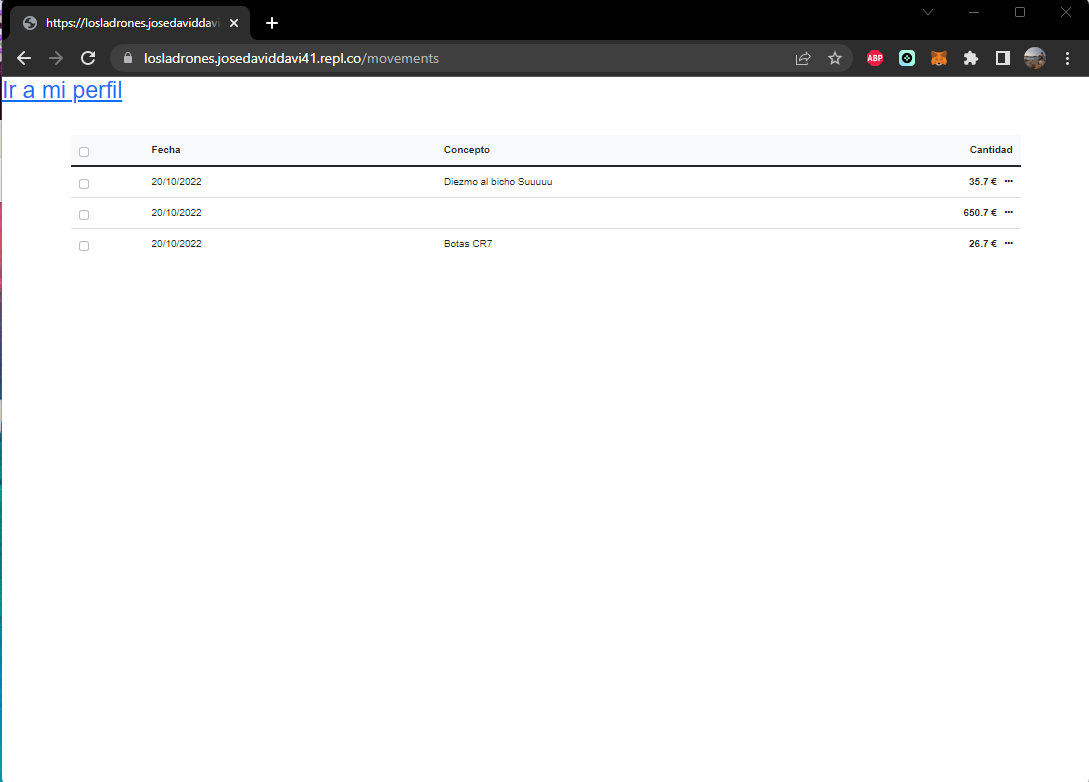
Datos de Usuarios:

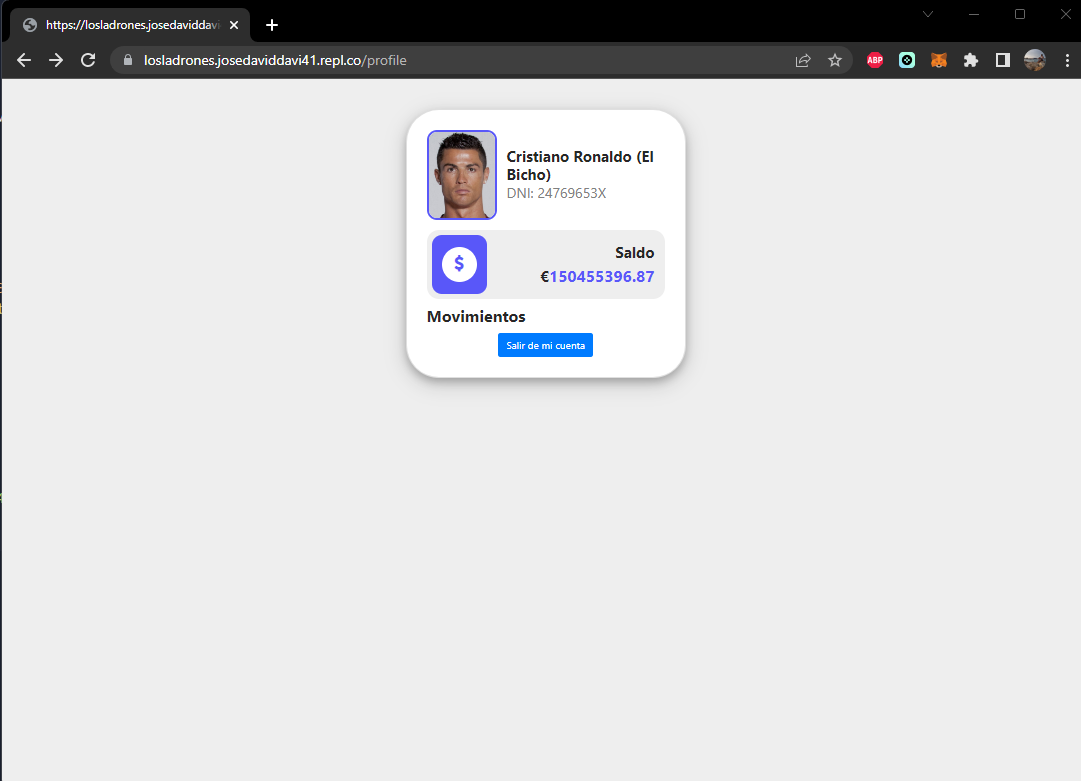


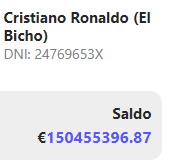
Mejoras

Hemos implementado una interfaz gráfica amigable y completamente funcional a nuestro código (Página de home, login, movimientos y perfil). Además está totalmente online (hemos incluido el enlace de acceso al inicio de esta memoria)

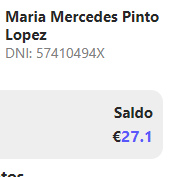






Hemos usado una BBDD donde tenemos a nuestros usuarios y se puede acceder a los diferentes perfiles.





Para las conexiones con la página web hemos usado cifrado SSL; para que ningún dato viaje en plano a través de la red.

