**Ouadi Chamit (X4788351G)**

**Manuel García Menárguez (48640683F)**

**5 de junio de 2017**

**Gestión de contraseñas**

seguridad en el diseño de software – 2016/2017

**Contenido**

[**1-** **Introducción** 2](#_Toc484206456)

[**2-** **Programas alternativos** 3](#_Toc484206457)

[**3-** **Tecnologías y herramientas utilizadas** 5](#_Toc484206458)

[**4-** **Estructura de la aplicación** 6](#_Toc484206459)

[**5-** **Implementación de funcionalidades básicas** 7](#_Toc484206460)

[**6-** **Implementación de funcionalidades adicionales** 21](#_Toc484206461)

[**7-** **Despliegue de la aplicación** 24](#_Toc484206462)

[**8-** **Comentarios y conclusiones** 26](#_Toc484206463)

[**9-** **Tabla de tareas** 27](#_Toc484206464)

1. **Introducción**

En esta memoria se especifican todos los detalles relacionados con la práctica final de SDS, la cual consta de realizar una aplicación de gestión de contraseñas, con el objetivo más importante, que sea segura.

En los siguientes apartados se explican todos y cada uno de los puntos desarrollados, desde las aplicaciones que ya existen en el mercado, así como la estructura de la aplicación desarrollada, tecnologías utilizadas, funcionalidades básicas y más avanzadas, hasta finalizar con un apartado de conclusiones y una tabla de tareas.

1. **Programas alternativos**

No solo existe una aplicación de gestores de cuentas o contraseñas seguras (la desarrollada en esta práctica), sino que existe una variedad bastante extensa de aplicaciones que sirven básicamente para lo mismo: gestionar las contraseñas.

En este apartado se citan algunas de ellas y se explican por encima ciertas características propias:

* **Lastpass**
  + Es un servicio multiplataforma y por tanto, está disponible para escritorio, web (navegadores) y móvil. Tiene versión premium y versión para empresas, la cual es de pago.
  + Usuario se registra con usuario y contraseña, la cual hace de “maestra” para poder acceder a las cuentas dadas de altas en el programa.
  + Utiliza cifrado AES 256 bits con PBKDF2 SHA-256 y hashes con sal.
  + Servidor conocimiento 0. Los datos se cifran y descifran en el dispositivo.
  + Autenticación doble factor. El usuario puede elegir, además de su usuario y contraseña, un segundo paso para autenticarse, como es: enviar un email a otra cuenta de correo o enviar un SMS de control.
* **1Password**
  + También está disponible para varias plataformas. Es de pago (3 $ / mes), aunque tiene una versión web gratuita si introduces tu email, con un período de evaluación.
  + Cifrado AES-256 bits y PBKDF2 derivación de clave.
  + Clave maestra y clave secreta de 128-bits.
  + Acceso a la información mediante transporte seguro con mecanismos TLS/SSL.
* **Keepass**
  + Licencia Open Source, por tanto, es gratuita su instalación.
  + Soporta dos algoritmos de cifrados: AES y Twofish.
  + SHA-256 bits para hash.
  + Disponible en varias plataformas además de tener varios “ports”. Esto es, versiones del programa desarrollados por terceros, pero utilizando las mismas técnicas de seguridad que Keepass oficial.
  + Tiene multitud de plugins o extensiones instalables: Backups de las bases de datos que tienen como cuenta almacenada, integrar Firefox o Chrome con el software para almacenar las contraseñas que se guardan en local mediante dichos navegadores, etc.

1. **Tecnologías y herramientas utilizadas**

Las tecnologías y/o herramientas utilizadas en esta práctica son las siguientes:

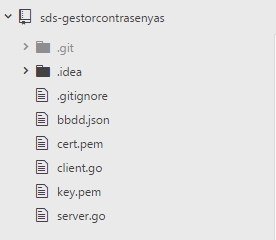
* **Windows 8.1:** Sistema operativo utilizado para desarrollar el código. Los dos componentes del grupodisponemos del mismo sistema operativo en nuestros equipos.
* **Go 1.8:** Lenguaje de programación utilizado para implementar todas las funcionalidades de esta práctica.
* **Git y Github:** Git como control de versiones y Github como repositorio remoto de código en privado, pasado posteriormente a público a la hora de la entrega de esta práctica:

Es el siguiente:

<https://github.com/manugm1/sds-gestorcontrasenyas>

* **Atom:** Editor de código, al cual se le instala el plugin [Go-Plus](https://atom.io/packages/go-plus) para disponer de autocompletado de código.

1. **Estructura de la aplicación**



* **bbdd.json:** Fichero con contenido en formato JSON. Contiene los datos de los usuarios registrados así como sus entradas asociadas. El motivo de almacenamiento en este formato es básicamente por facilidad de procesado y compatibilidad con Go, dado que las librerías por defecto incluyen mecanismos de conversión de mapas o listas del lenguaje Go a su posterior conversión a JSON, y viceversa.
* **cert.pem y key.pem:** Son los certificados necesarios para la compatibilidad del servidor con el sistema de transporte seguro TLS. Son certificados auto-firmados, que sirven para realizar pruebas sin necesidad de pagar por un certificado original y firmado por un organismo oficial.
* **client.go:** Fichero escrito en el lenguaje Go que incorpora la lógica del cliente del gestor de contraseñas. De él nacen el menú, la comunicación con el servidor, cifrado de contraseña del propio usuario, etc.
* **server.go:** Fichero escrito en el lenguaje Go que incorpora la lógica del servidor del gestor de contraseñas. De él nacen la comunicación de transporte seguro TLS, control de tiempo de expiración de la sesión, sistema de Tokens, generación más envío de emails por autenticación doble, etc.

1. **Implementación de funcionalidades básicas**

**Comunicación Cliente/Servidor**

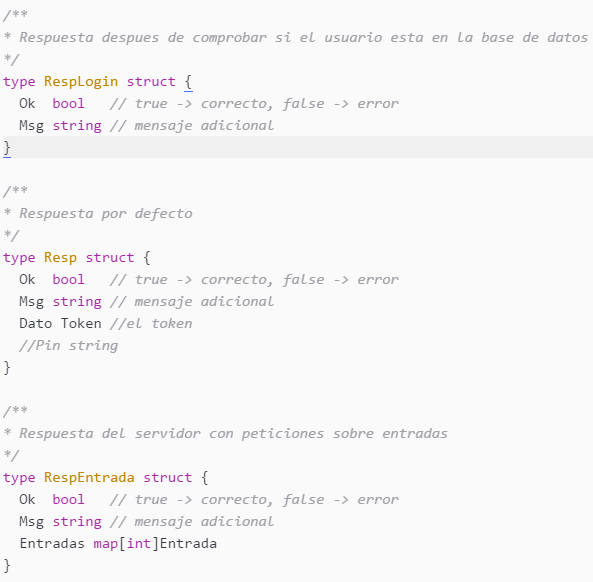
La comunicación del cliente con el servidor se realiza mediante transporte seguro TLS, además que todo lo que viaja va codificado en base64.

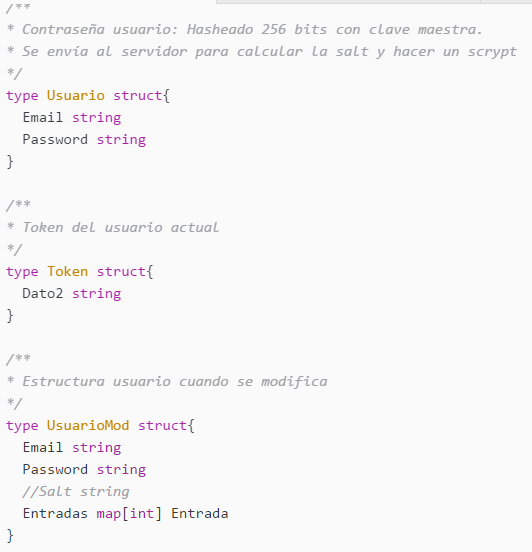
La configuración necesaria de TLS, se realiza en el servidor, utilizando los certificados auto-firmados (cert.pem y key.pem). El servidor arrancará en una dirección url (localhost) y un puerto, 10443. Quedará a la escucha de peticiones por parte del cliente.

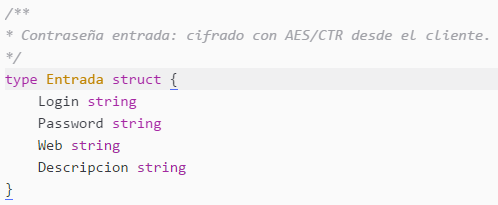
El sistema de peticiones será como un servidor REST, salvo que en vez de complicarse en la creación de rutas, se trabajará con un parámetro GET que indicará el tipo de opción/petición a realizar.

El envío de información es mediante estructuras Go. Las estructuras creadas para el traspaso de información son las siguientes:

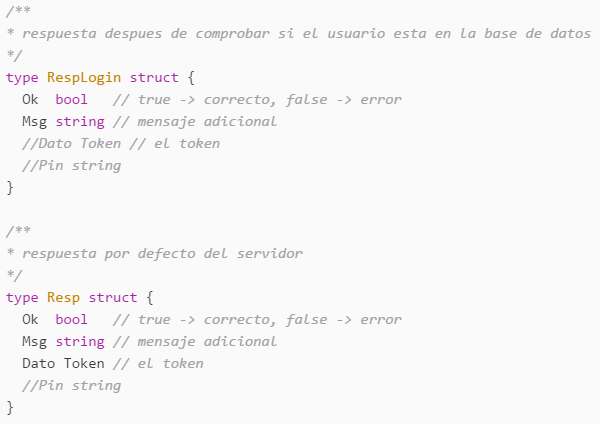
Estructuras cliente

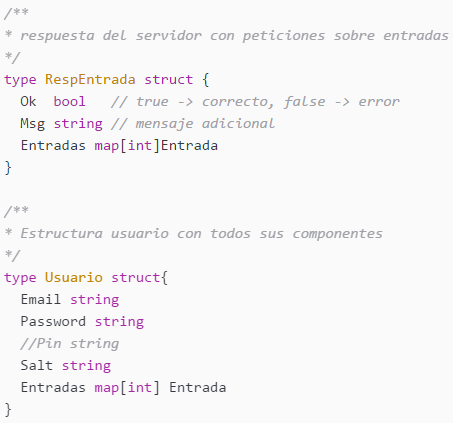


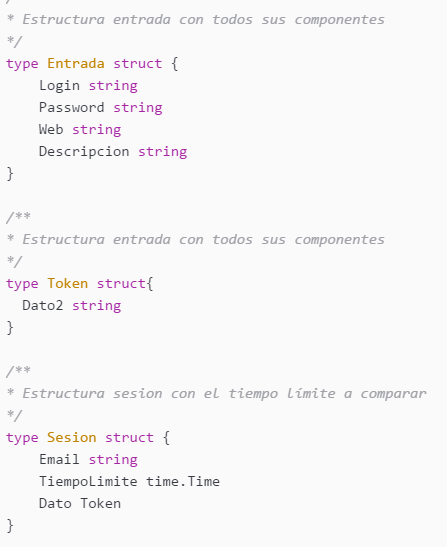


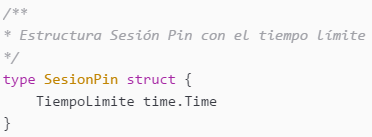


Estructuras Servidor

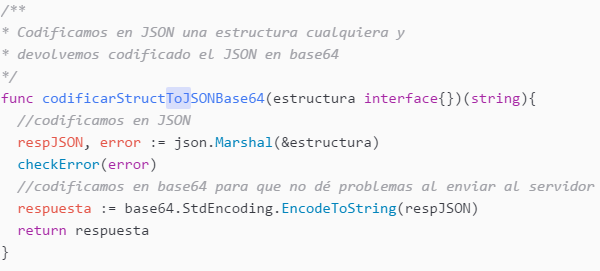


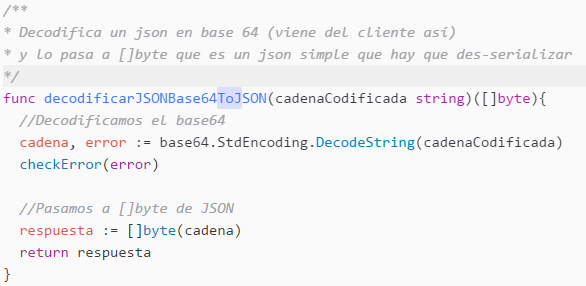






Por cada petición/consulta del cliente al servidor o del servidor al cliente, como se ha comentado, se utilizan los métodos para codificar a base64. Estos métodos utilizan el sistema por defecto de Go para codificar/decodificar en base64 (StdEncoding.EncodeToString y StdEncoding.DecodeString), más algún añadido, lo cual nos sirve para reutilizar un mismo método en distintos puntos del programa y que comprendan el mismo funcionamiento. En las siguientes capturas se pueden ver los métodos:



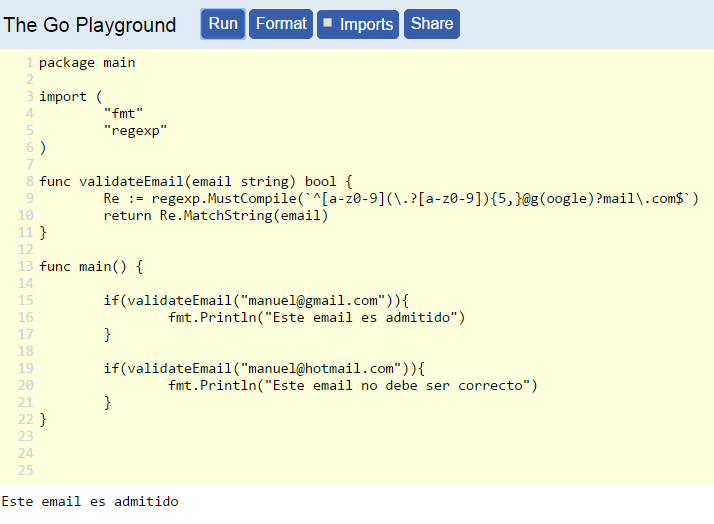


**Login y Registro Usuario**

Registro

Para poder registrarse un usuario en el sistema, se le pedirá que introduzca su email y su contraseña. El email será validado y si no cumple con las características de un email de GMAIL, obligará al usuario a introducir uno correcto.

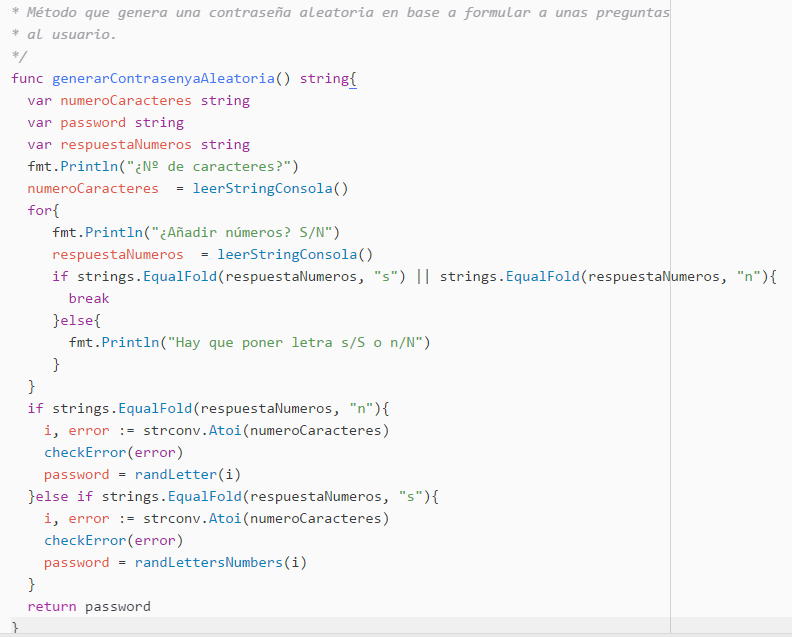
A continuación, una captura del método antes de incorporarlo al desarrollo, viendo como una cuenta de GMAIL es admitida mientras que una de HOTMAIL no.



Antes de introducir la contraseña, se preguntará si quiere una contraseña aleatoria generada por el sistema o si él mismo quiere insertarla manualmente.

*¿Generar contraseña aleatoria? (S/N)*

Si la respuesta es “S” (sí), se invocará a un método que preguntará al usuario por la longitud de la contraseña y si quiere números y letras o tan solo letras.



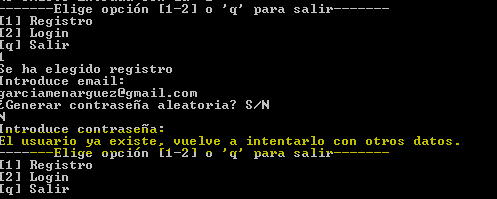
Tanto si se ha generado una contraseña aleatoria como si la ha escrito el usuario, el siguiente paso en el registro por parte del cliente, es generar el hash correspondiente SHA 256 y codificarla a base64 antes de enviar al servidor.

Por último, al enviar al servidor y rellenar los campos de la estructura “Usuario” con el email y la contraseña hasheada, se codifica a base64 toda la estructura.

El registro en el servidor recoge la estructura enviada “Usuario” por parte del cliente, la decodifica del base64 y la guarda en una estructura copia, llamada también “Usuario”.

Verifica si existe el usuario (un método aparte para comprobar si existe el email que ha introducido el usuario por teclado) y si existe, devuelve una estructura “Resp” que compone un “Ok=false” (booleano) y una variable “Msg=El usuario ya existe, vuelve a intentarlo con otros datos.”.





Si no existe el usuario, se procede a darlo de alta. Se encripta la contraseña y su salt (ver apartado “Seguridad en contraseñas” para más información) y guardamos la estructura “Usuario” en el mapa de usuarios (ver apartado “Sesiones y tiempos de expiración” para más información).

Se devuelve al cliente una estructura “Resp” que compone un “Ok=true” y una variable “Msg=Registrado con éxito. Inicia sesión para empezar”.



Login

A la hora de hacer login, el usuario como es normal, deberá estar registrado en el sistema. Si no lo está, una vez se envíen los datos al servidor, se devolverá error.

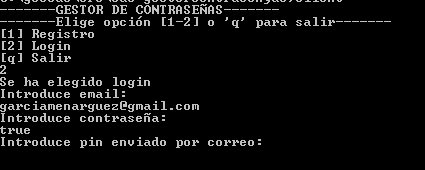
Al intentar hacer login, se le pide al usuario su email y la contraseña. El email, al igual que en el registro, volverá a ser comprobado que cumple con el formato correcto de una cuenta de GMAIL. La contraseña seguirá el mismo proceso de cifrado que en el registro, se generará su hash de 256 bytes.

Una vez se tenga, se enviará codificado en base64 en una estructura “Usuario” al servidor.

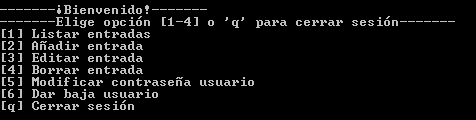
El servidor comprobará si el usuario existe (si existe el email) y, aparte de eso, si coincide el usuario y la contraseña registrado. Para comprobar la contraseña hay que encriptarla con el salt como si se fuera a guardar y entonces ver si coincide la almacenada con la introducida.

Si sí coinciden, el servidor generará un pin aleatorio (ver apartado “Autenticación con pin (autenticación múltiple)” para más información), por lo que habrá que realizar un paso más, que será introducir desde el cliente el pin generado aleatoriamente en el servidor.

Si el pin también coincide, se devolverá respuesta correcta al cliente y entrará al menú de opciones de la aplicación.



El menú principal al acceder es el siguiente:



**CRUD Entradas**

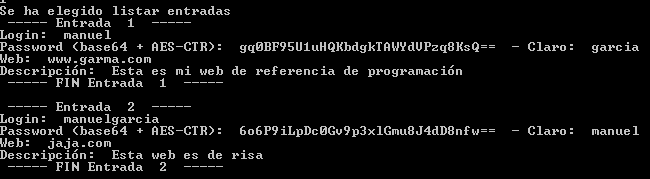
Se han implementado todas las operaciones básicas sobre la entidad “Entrada”. Es decir, se permite crear, leer (listar todas), modificar y borrar.

Una entrada consta de los siguientes datos:

* Login del servicio
* Contraseña
* Web
* Descripción

Todos son cadenas de datos. La mayor dificultad existe en cifrar la contraseña introducida, comentada en el punto “Seguridad en contraseñas”.

A continuación, cuando se le da a listar entradas, un ejemplo de cómo se visualizan las 2 entradas que tiene el usuario “garciamenarguez@gmail.com”:



Como se puede ver, la contraseña se muestra cifrada y “en claro”. Esto no es lo correcto, pero se muestra a modo de depuración para que se vea que la contraseña con AES-CTR se puede descifrar, es decir, volver atrás y dejar el valor original.

Lógicamente, esto estaría PROHIBIDO si es un entorno de producción, de hecho, la password debería mostrarse como “\*\*\*\*\*\*” o directamente, vacío (enmascarar).

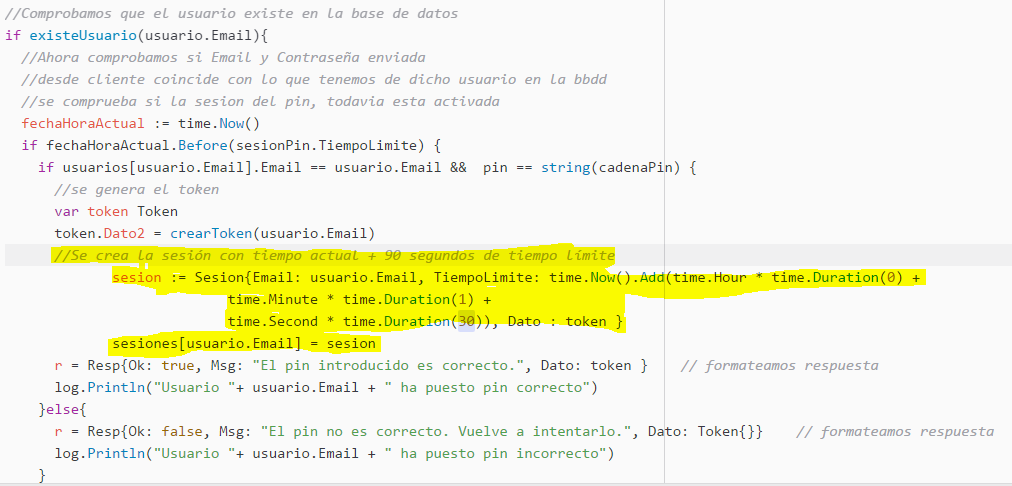
**Sesiones y tiempos de expiración**

Cuando un cliente es logueado en el sistema, el servidor guarda la fecha y hora de ese login y, también, por cada movimiento que realice el usuario. Una vez se realiza ese movimiento, el tiempo se compara con el del movimiento actual y el último registrado, esto es, la diferencia entre el tiempo del último movimiento con el que se acaba de hacer.

El tiempo configurado en el servidor es de 1 minuto y 30 segundos.

Pasada esa fecha entre movimientos, el servidor “romperá” la sesión del usuario, por lo que se cerrará su sesión y el cliente pasará a estar fuera del menú principal.

Un ejemplo del escenario para que se dé el caso sería: el usuario inicia sesión (movimiento 1), quiere ver el listado de cuentas creadas (movimiento 2), y se ausenta 2 minutos. Cuando intente realizar cualquier otra cosa, se cerrará el sistema, volviendo al menú principal de registro o inicio de sesión.



**Seguridad en contraseñas**

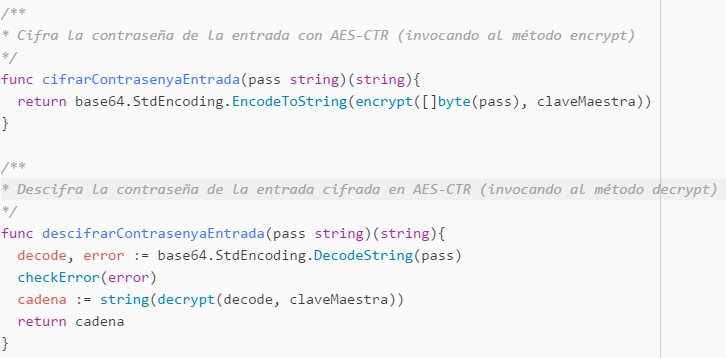
La seguridad en contraseñas es una de las partes esenciales de la aplicación.

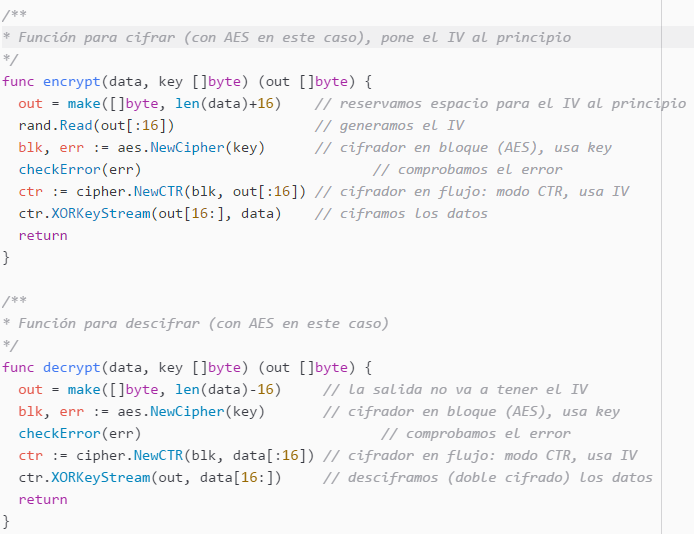
Consta de dos tipos de cifrado, uno utilizando el cifrado de contraseñas del usuario y otro utilizado en el cifrado de contraseñas de las entradas.

Cifrado de contraseñas entradas

El cifrado de contraseñas para las entradas se realiza utilizando AES- CTR, se realiza en el cliente por lo que el servidor tiene conocimiento cero. Al servidor le llegan ya las entradas cifradas (y codificadas en base64), así que solo debe preocuparse de almacenar los datos.

Previamente al cifrado en AES, se extrae de la contraseña del usuario el hash SHA-256, por lo que se utiliza posteriormente ese hash para cifrarlo con AES-CTR.





Cifrado de contraseñas usuario

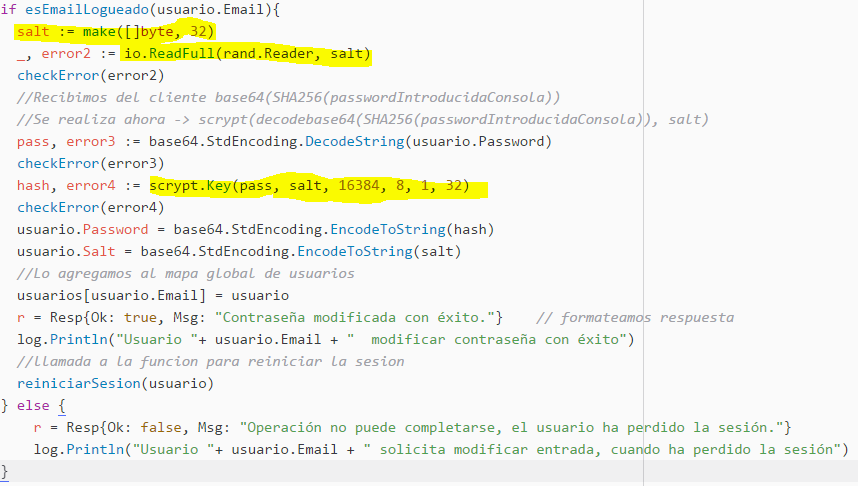
Este cifrado es algo más “complejo”. Es a nivel del registro del usuario. Utiliza SHA-256 como hash y se envía al servidor (codificado en base64).

El servidor, lo primero crea una salt aleatoria. Esta salt servirá para “mezclarla” con el hash recibido de parte del cliente y realizar un encriptado con “Scrypt”.

Una vez se realice el scrypt, se codifica a base64 para evitar problemas de guardado en la base de datos, y se guarda en la base de datos.

A la hora de realizar el login y poder comprobar la contraseña introducida si es la almacenada, como este sistema solo es de IDA, es decir, no se puede recuperar la contraseña original (en principio), lo que queda es realizar el mismo proceso de cifrado + encriptado y comparar con la contraseña almacenada.

Esto lleva a pensar que también hay que guardar la salt generada al registrarse el usuario. Esa salt quedará guardada para ese usuario y solo la sabrá el servidor, por lo que el cliente no tendrá acceso.



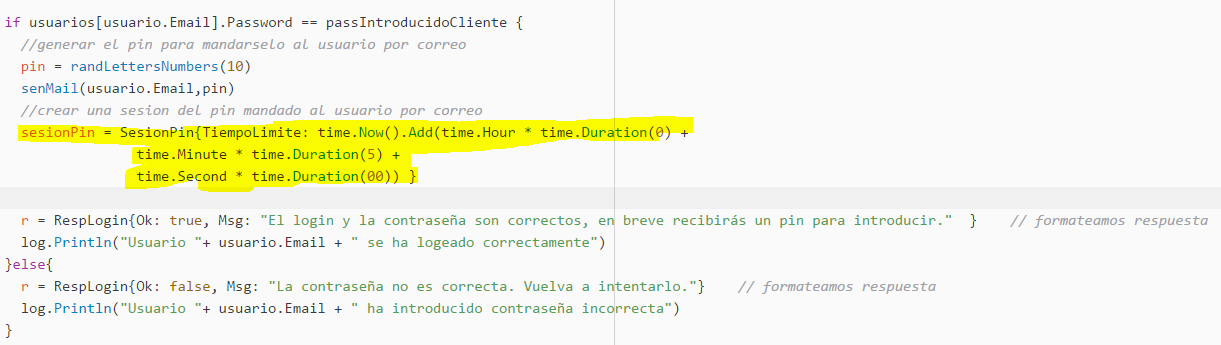
1. **Implementación de funcionalidades adicionales**

**Autenticación con pin (autenticación múltiple)**

Es una medida extra de seguridad implementada a nivel de login del usuario.

Cada vez que el usuario intenta iniciar sesión, se le pedirá su email y contraseña (como es normal) más aparte un pin de seguridad. El pin de seguridad se envía al email del usuario.

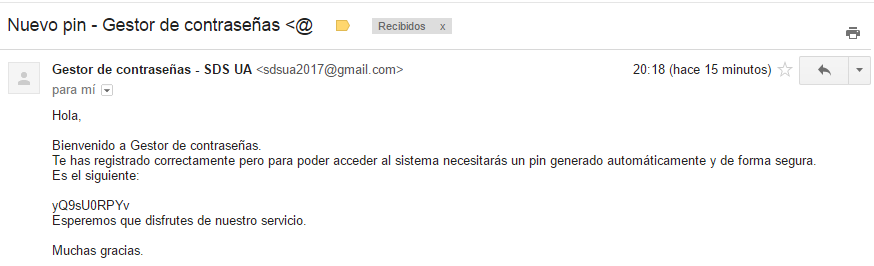
Dispone de 5 minutos antes de que se caduque el pin. Pasado ese tiempo, si se intenta introducir, dará error y tendrá que volver a intentar iniciar sesión (introduciendo de nuevo los datos y accediendo al correo de nuevo en busca de un nuevo pin).



El correo lo envía el servidor a través de una librería de Go "net/smtp" y "net/mail". Se ha creado una cuenta de GMAIL específica para poder enviar los correos de notificación con el pin incluído.

El correo en concreto es: [sdsua2017@gmail.com](mailto:sdsua2017@gmail.com) y su contraseña: “sds2017ua”.

El correo luciría así:



**Sesiones con Tokens**

Cuando el usuario, en tiempo de inicio de sesión, introduce su pin, el servidor creará un token de seguridad que será pasado al cliente y éste, deberá pasarlo por cada operación que realice. Si intenta listar entradas, si intenta modificar una entrada, etc., deberá pasar el token al servidor para que verifique si el token pertenece al usuario que acaba de realizar la petición, verificando que “es él”.

De esta forma, incrementamos la seguridad evitando que alguien suplante o se haga pasar por el usuario sin previo logueo.

**Modificación de usuarios y baja**

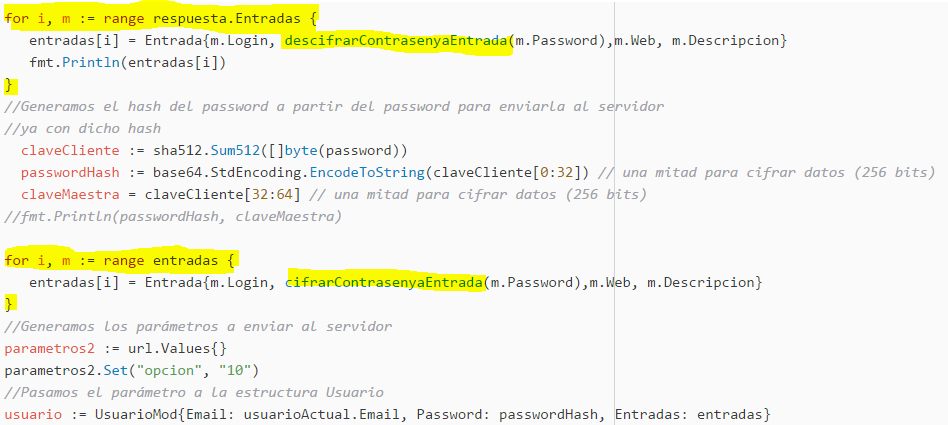
Esta funcionalidad consiste en algo necesario para el usuario y no es otra cosa que dar la posibilidad de modificar sus datos como su contraseña, y poder darse de baja del sistema, borrando toda su información.

Para poder modificar su contraseña, existe un pequeño problema y es que, tal y como se ha desarrollado la encriptación de contraseñas de las cuentas, al estar basada en la contraseña maestra del usuario, si ésta se cambia, no se pueden recuperar las contraseñas de las cuentas.

Tal y como se ha comentado en el apartado de “Seguridad en contraseñas”, al realizarse un SHA512, y dividirse 256 bits para la contraseña del usuario y 256 bits para cifrar, si queremos dar la posibilidad de cambiar en cualquier momento la contraseña del usuario, deberemos “recifrar” todas las contraseñas guardadas de las cuentas.

Esta es la mayor dificultad, solucionada con éxito.

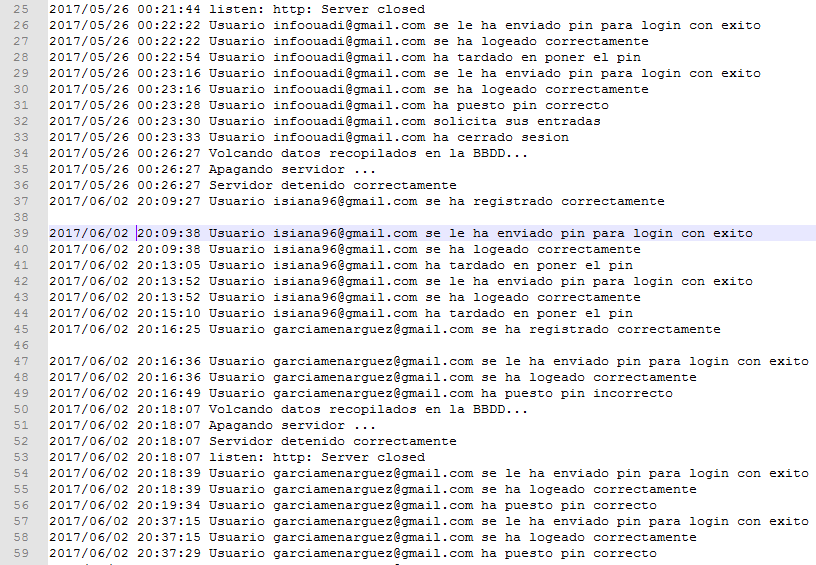
Si el usuario decide cambiar su contraseña, basta obtener todas las entradas creadas, recorrerlas, descifrarlas e implementarles el nuevo SHA, cifrando cada una y pasándolas al servidor para su guardado.



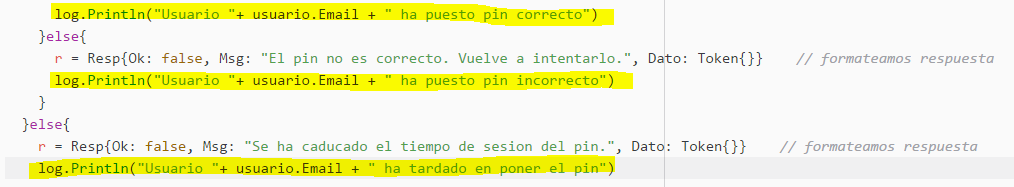
**Sistema de log**

El último punto adicional implementado es un sistema de log por cada petición realizada por el usuario.

Cuando un usuario se registra, se loguea, lista sus entradas, edita una entrada… en definitiva, realiza cualquier cosa, es registrado en un fichero, marcando la hora y fecha de la acción junto con la acción realizada.



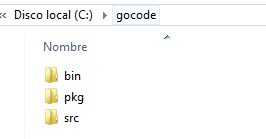
A continuación, una captura de cómo se escribe en el fichero, justo a la hora de crear una entrada:



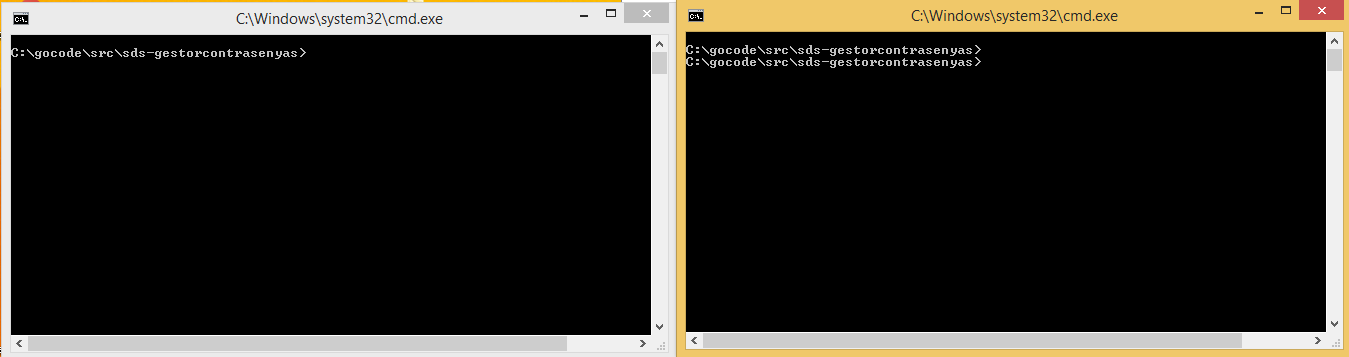
1. **Despliegue de la aplicación**

En este apartado se detallan los pasos necesarios para la puesta en marcha de la aplicación, desde la configuración necesaria hasta llegarla a ejecutar:

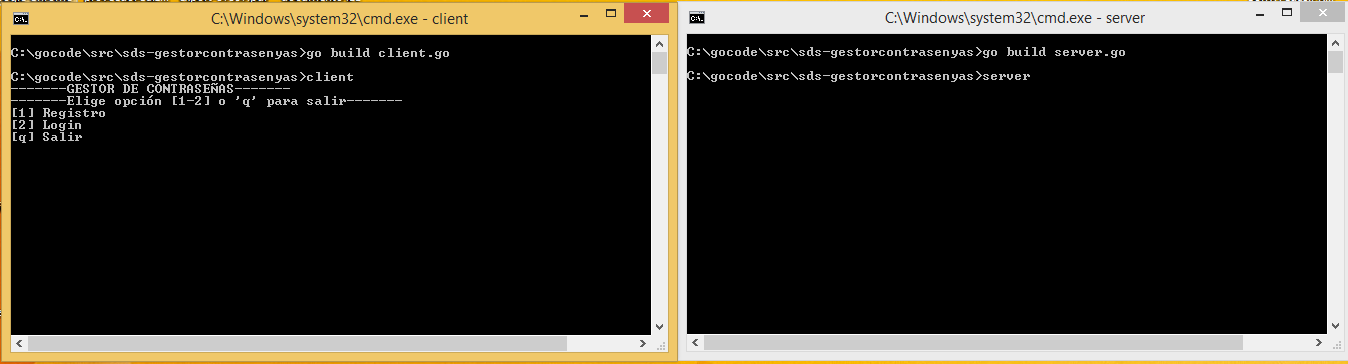
1. Suponiendo que el sistema operativo es Windows 8.1 o, en su defecto, cualquier Windows.
2. Instalación de Go 1.8.
3. Crear carpeta “C:\gocode” que será la carpeta donde se creen los proyectos desarrollados, en nuestro caso, la práctica en sí. Crear dentro de ella las tres subcarpetas: bin, pkg y src.



1. Crear/Modificar variables de entorno:
   * 1. Crear variable “GOROOT” con valor “C:\Go\”.
     2. Crear variable “GOPATH” con valor “C:\gocode”.
     3. Modificar variable “Path” y añadir al final: “;C:\Go\bin;”.
2. Copiar carpeta de la práctica “sds-gestorcontrasenyas” en la subcarpeta “src” antes creada, quedando: “C:\gocode\src\sds-gestorcontrasenyas”.
3. Abrir dos terminales (CMD Windows) mínimo: una para el cliente y otra para el servidor. Que conste que pueden haber N terminales para clientes. Escribir en cada una la ruta a la práctica: cd C:\gocode\src\sds-gestorcontrasenyas:



1. Compilar y ejecutar:



1. **Comentarios y conclusiones**

Como comentarios finales o conclusiones podemos comentar las mejoras que han quedado pendientes y su intento de solución (básicamente una) ya que todo lo que se planteó al principio se ha cumplido:

* ¿Qué pasa si el usuario olvida/pierde la contraseña? Actualmente si el usuario olvida o pierde su contraseña, deberá volver a registrarse con otro usuario (email) porque no hay forma de recuperarlo.

Si nos centramos en lo “técnico”, tal y como se almacenan con Scrypt, no hay vuelta atrás, es decir, las contraseñas de los usuarios no pueden descifrarse una vez guardadas.

Una posible solución, podría ser mediante el ya utilizado pin de seguridad, dándole una vuelta más. Esto sería: enviar el pin de seguridad al email del usuario. Cuando el usuario acceda a su correo, recoja el pin y lo introduzca en la aplicación, entonces el servidor tendrá “permiso” para resetear la clave del usuario y concederle “otra oportunidad” de acceso, con una nueva contraseña. El servidor la cifrará como siempre, con “scrypt” más la salt aleatoria, pero esta vez sin comprobar si coincide con lo introducido, ya que sería una especie de registro pero con el usuario ya existente.

Otra solución, algo más rebuscada, sería tener un fichero extra con los usuarios y sus contraseñas, y cifrar a nivel de fichero, no de password. Utilizar de nuevo el pin de seguridad. Esto es, enviar el pin de seguridad al email del usuario. Cuando el usuario acceda a su correo, recoja el pin y lo introduzca en la aplicación, entonces el servidor tendrá “permiso” para acceder al fichero cifrado donde están todas las claves y reenviarla por correo, para que se acuerde. Esto es un poco peligroso, ya que estaríamos enviando la contraseña en claro al correo del usuario. ¿Qué ocurriría si alguien intercepta el mensaje? Entraríamos ya en la comunicación segura vía correo electrónico… por tanto, nos quedaríamos con la primera opción como posible solución, más viable.

1. **Tabla de tareas**

|  |  |  |
| --- | --- | --- |
|  | **Ouadi** | **Manuel** |
| Definir Objetos Estructuras + Comunicación Cliente/Servidor (con TLS) |  | X |
| CRUD Entradas en claro + BBDD | X | X |
| Readaptación CRUD Entradas + transporte Base64 | X | X |
| Sesiones con tiempo de expiración |  | X |
| Seguridad en Contraseñas (Pass Usuario + Pass Entradas) |  | X |
| Autenticación con pin + envío email | X |  |
| Posibilidad modificar contraseña usuario + rehash contraseñas cuentas | X |  |
| Tokens en sesiones usuario | X |  |
| Sistema de log en el servidor | X |  |
| Memoria final |  | X |