



5 DE JUNIO DE 2017

GESTIÓN DE CONTRASEÑAS

SEGURIDAD EN EL DISEÑO DE SOFTWARE – 2016/2017

Ouadi Chamit (X4788351G)

Manuel García Menárguez (48640683F)

Contenido

| | | |
|----|---|----|
| 1- | Introducción..... | 2 |
| 2- | Programas alternativos | 3 |
| 3- | Tecnologías y herramientas utilizadas | 5 |
| 4- | Estructura de la aplicación | 6 |
| 5- | Implementación de funcionalidades básicas | 7 |
| 6- | Implementación de funcionalidades adicionales | 21 |
| 7- | Despliegue de la aplicación | 24 |
| 8- | Comentarios y conclusiones | 26 |
| 9- | Tabla de tareas..... | 27 |

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.

2- 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.

3- Tecnologías y herramientas utilizadas

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

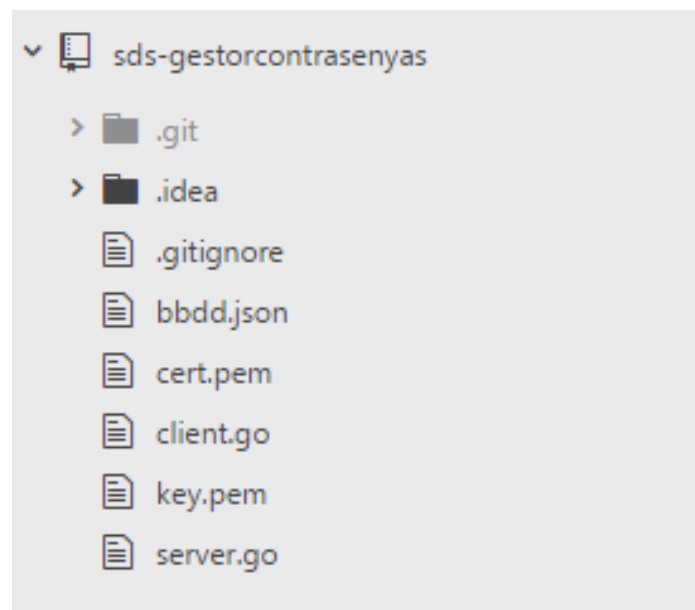
- **Windows 8.1:** Sistema operativo utilizado para desarrollar el código. Los dos componentes del grupo disponemos del mismo sistema operativo en nuestros equipos.
- **Go 1.8:** Lenguaje de programación utilizado para implementar todas las funcionalidades de esta práctica.
- **Git y Github:** Git como 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](#) para disponer de autocompletado de código.

4- 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.

5- 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


```
/**
 * Respuesta despues de comprobar si el usuario esta en la base de datos
 */
type RespLogin struct {
    Ok bool // true -> correcto, false -> error
    Msg string // mensaje adicional
}

/**
 * Respuesta por defecto
 */
type Resp struct {
    Ok bool // true -> correcto, false -> error
    Msg string // mensaje adicional
    Dato Token //el token
    //Pin string
}

/**
 * Respuesta del servidor con peticiones sobre entradas
 */
type RespEntrada struct {
    Ok bool // true -> correcto, false -> error
    Msg string // mensaje adicional
    Entradas map[int]Entrada
}
```

```
/**
 * Contraseña usuario: Hasheado 256 bits con clave maestra.
 * Se envía al servidor para calcular la salt y hacer un scrypt
 */
type Usuario struct{
    Email string
    Password string
}

/**
 * Token del usuario actual
 */
type Token struct{
    Dato2 string
}

/**
 * Estructura usuario cuando se modifica
 */
type UsuarioMod struct{
    Email string
    Password string
    //Salt string
    Entradas map[int] Entrada
}
```

```
/**
 * Contraseña entrada: cifrado con AES/CTR desde el cliente.
 */
type Entrada struct {
    Login string
    Password string
    Web string
    Descripcion string
}
```

Estructuras Servidor

```
/**
 * respuesta despues de comprobar si el usuario esta en la base de datos
 */
type RespLogin struct {
    Ok bool // true -> correcto, false -> error
    Msg string // mensaje adicional
    //Dato Token // el token
    //Pin string
}

/**
 * respuesta por defecto del servidor
 */
type Resp struct {
    Ok bool // true -> correcto, false -> error
    Msg string // mensaje adicional
    Dato Token // el token
    //Pin string
}
```

```
/**
 * respuesta del servidor con peticiones sobre entradas
 */
type RespEntrada struct {
    Ok bool // true -> correcto, false -> error
    Msg string // mensaje adicional
    Entradas map[int]Entrada
}

/**
 * Estructura usuario con todos sus componentes
 */
type Usuario struct{
    Email string
    Password string
    //Pin string
    Salt string
    Entradas map[int] Entrada
}
```

```
/* Estructura entrada con todos sus componentes
*/
type Entrada struct {
    Login string
    Password string
    Web string
    Descripcion string
}

/**
* Estructura entrada con todos sus componentes
*/
type Token struct{
    Dato2 string
}

/**
* Estructura sesion con el tiempo límite a comparar
*/
type Sesion struct {
    Email string
    TiempoLimite time.Time
    Dato Token
}

/**
* Estructura Sesión Pin con el tiempo límite
*/
type SesionPin struct {
    TiempoLimite time.Time
}
```

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:

```
/**
 * Codificamos en JSON una estructura cualquiera y
 * devolvemos codificado el JSON en base64
 */
func codificarStructToJSONBase64(estructura interface{})(string){
    //codificamos en JSON
    respJSON, error := json.Marshal(&estructura)
    checkError(error)
    //codificamos en base64 para que no dé problemas al enviar al servidor
    respuesta := base64.StdEncoding.EncodeToString(respJSON)
    return respuesta
}

/**
 * Decodifica un json en base 64 (viene del cliente así)
 * y lo pasa a []byte que es un json simple que hay que des-serializar
 */
func decodificarJSONBase64ToJSON(cadenaCodificada string)([]byte){
    //Decodificamos el base64
    cadena, error := base64.StdEncoding.DecodeString(cadenaCodificada)
    checkError(error)

    //Pasamos a []byte de JSON
    respuesta := []byte(cadena)
    return respuesta
}
```

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.

```
The Go Playground  Run Format Imports Share
1 package main
2
3 import (
4     "fmt"
5     "regexp"
6 )
7
8 func validateEmail(email string) bool {
9     Re := regexp.MustCompile(`^[a-z0-9](\.[a-z0-9]){5,}@g(oogle)?mail\.com$`)
10    return Re.MatchString(email)
11 }
12
13 func main() {
14
15     if(validateEmail("manuel@gmail.com")){
16         fmt.Println("Este email es admitido")
17     }
18
19     if(validateEmail("manuel@hotmail.com")){
20         fmt.Println("Este email no debe ser correcto")
21     }
22 }
23
24
25
```

Este email es admitido

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.

```

* Método que genera una contraseña aleatoria en base a formular a unas preguntas
* al usuario.
*/
func generarContraseñaAleatoria() string{
    var numeroCaracteres string
    var password string
    var respuestaNumeros string
    fmt.Println("¿Nº de caracteres?")
    numeroCaracteres = leerStringConsola()
    for{
        fmt.Println("¿Añadir números? S/N")
        respuestaNumeros = leerStringConsola()
        if strings.EqualFold(respuestaNumeros, "s") || strings.EqualFold(respuestaNumeros, "n"){
            break
        }else{
            fmt.Println("Hay que poner letra s/S o n/N")
        }
    }
    if strings.EqualFold(respuestaNumeros, "n"){
        i, error := strconv.Atoi(numeroCaracteres)
        checkError(error)
        password = randLetter(i)
    }else if strings.EqualFold(respuestaNumeros, "s"){
        i, error := strconv.Atoi(numeroCaracteres)
        checkError(error)
        password = randLettersNumbers(i)
    }
    return password
}

```

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.".

```

r = Resp{Ok: false, Msg: "El usuario ya existe, vuelve a intentarlo con otros datos."} // formateamos respuesta

```

```
-----Elige opción [1-2] o 'q' para salir-----
[1] Registro
[2] Login
[q] Salir

Se ha elegido registro
Introduce email:
garciamenarguez@gmail.com
Generar contraseña aleatoria? S/N
N
Introduce contraseña:
El usuario ya existe, vuelve a intentarlo con otros datos.
-----Elige opción [1-2] o 'q' para salir-----
[1] Registro
[2] Login
[q] Salir
```

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”.

```
r = Resp{Ok: true, Msg: "Registrado con éxito. Inicia sesión para empezar."} // formateamos respuesta
```

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.


```
-----GESTOR DE CONTRASEÑAS-----  
-----Elige opción [1-2] o 'q' para salir-----  
[1] Registro  
[2] Login  
[q] Salir  
2  
Se ha elegido login  
Introduce email:  
garciamenarguez@gmail.com  
Introduce contraseña:  
true  
Introduce pin enviado por correo:
```

El menú principal al acceder es el siguiente:

```
-----¡Bienvenido!-----  
-----Elige opción [1-4] o 'q' para cerrar sesión-----  
[1] Listar entradas  
[2] Añadir entrada  
[3] Editar entrada  
[4] Borrar entrada  
[5] Modificar contraseña usuario  
[6] Dar baja usuario  
[q] Cerrar sesión
```

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”:

```
Se ha elegido listar entradas
----- Entrada 1 -----
Login: manuel
Password (base64 + AES-CTR): gq0BF95U1uHqKbdgkTAWYdUPzq8KsQ== - Claro: garcia
Web: www.garna.com
Descripción: Esta es mi web de referencia de programación
----- FIN Entrada 1 -----

----- Entrada 2 -----
Login: manuelgarcia
Password (base64 + AES-CTR): 6o6P9iLpDc0Gv9p3x1Gmu8J4dD8nfw== - Claro: manuel
Web: jaja.com
Descripción: Esta web es de risa
----- FIN Entrada 2 -----
```

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.

```
//Comprobamos que el usuario existe en la base de datos
if existeUsuario(usuario.Email){
    //Ahora comprobamos si Email y Contraseña enviada
    //desde cliente coincide con lo que tenemos de dicho usuario en la bbdd
    //se comprueba si la sesion del pin, todavia esta activada
    fechaHoraActual := time.Now()
    if fechaHoraActual.Before(sesionPin.TiempoLimite) {
        if usuarios[usuario.Email].Email == usuario.Email && pin == string(cadenaPin) {
            //se genera el token
            var token Token
            token.Dato2 = crearToken(usuario.Email)
            //Se crea la sesión con tiempo actual + 90 segundos de tiempo límite
            sesion := Sesion{Email: usuario.Email, TiempoLimite: time.Now().Add(time.Hour * time.Duration(0) +
                time.Minute * time.Duration(1) +
                time.Second * time.Duration(30)), Dato : token }
            sesiones[usuario.Email] = sesion
            r = Resp{Ok: true, Msg: "El pin introducido es correcto.", Dato: token } // formateamos respuesta
            log.Println("Usuario "+ usuario.Email + " ha puesto pin correcto")
        }else{
            r = Resp{Ok: false, Msg: "El pin no es correcto. Vuelve a intentarlo.", Dato: Token{}} // formateamos respuesta
            log.Println("Usuario "+ usuario.Email + " ha puesto pin incorrecto")
        }
    }
}
```

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.

```
/**
 * Cifra la contraseña de la entrada con AES-CTR (invocando al método encrypt)
 */
func cifrarContraseñaEntrada(pass string)(string){
    return base64.StdEncoding.EncodeToString(encrypt([]byte(pass), claveMaestra))
}

/**
 * Descifra la contraseña de la entrada cifrada en AES-CTR (invocando al método decrypt)
 */
func descifrarContraseñaEntrada(pass string)(string){
    decode, error := base64.StdEncoding.DecodeString(pass)
    checkError(error)
    cadena := string(decrypt(decode, claveMaestra))
    return cadena
}

/**
 * Función para cifrar (con AES en este caso), pone el IV al principio
 */
func encrypt(data, key []byte) (out []byte) {
    out = make([]byte, len(data)+16) // reservamos espacio para el IV al principio
    rand.Read(out[:16]) // generamos el IV
    blk, err := aes.NewCipher(key) // cifrador en bloque (AES), usa key
    checkError(err) // comprobamos el error
    ctr := cipher.NewCTR(blk, out[:16]) // cifrador en flujo: modo CTR, usa IV
    ctr.XORKeyStream(out[16:], data) // ciframos los datos
    return
}

/**
 * Función para descifrar (con AES en este caso)
 */
func decrypt(data, key []byte) (out []byte) {
    out = make([]byte, len(data)-16) // la salida no va a tener el IV
    blk, err := aes.NewCipher(key) // cifrador en bloque (AES), usa key
    checkError(err) // comprobamos el error
    ctr := cipher.NewCTR(blk, data[:16]) // cifrador en flujo: modo CTR, usa IV
    ctr.XORKeyStream(out, data[16:]) // desciframos (doble cifrado) los datos
    return
}
```

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 sscript, 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.

```
if esEmailLogueado(usuario.Email){
    salt := make([]byte, 32)
    _, error2 := io.ReadFull(rand.Reader, salt)
    checkError(error2)
    //Recibimos del cliente base64(SHA256(passwordIntroducidaConsola))
    //Se realiza ahora -> sscript(decodebase64(SHA256(passwordIntroducidaConsola)), salt)
    pass, error3 := base64.StdEncoding.DecodeString(usuario.Password)
    checkError(error3)
    hash, error4 := sscript.Key(pass, salt, 16384, 8, 1, 32)
    checkError(error4)
    usuario.Password = base64.StdEncoding.EncodeToString(hash)
    usuario.Salt = base64.StdEncoding.EncodeToString(salt)
    //Lo agregamos al mapa global de usuarios
    usuarios[usuario.Email] = usuario
    r = Resp{Ok: true, Msg: "Contraseña modificada con éxito."} // formateamos respuesta
    log.Println("Usuario "+ usuario.Email + " modificar contraseña con éxito")
    //Llamada a la funcion para reiniciar la sesion
    reiniciarSesion(usuario)
} else {
    r = Resp{Ok: false, Msg: "Operación no puede completarse, el usuario ha perdido la sesión."}
    log.Println("Usuario "+ usuario.Email + " solicita modificar entrada, cuando ha perdido la sesión")
}
```

6- 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).

```
if usuarios[usuario.Email].Password == passIntroducidoCliente {  
    //generar el pin para mandarselo al usuario por correo  
    pin = randLettersNumbers(10)  
    senMail(usuario.Email, pin)  
    //crear una sesion del pin mandado al usuario por correo  
    sessionPin = SessionPin(TiempoLimite: time.Now().Add(time.Hour * time.Duration(1) +  
        time.Minute * time.Duration(5) +  
        time.Second * time.Duration(00)) )  
  
    r = RespLogin(Ok: true, Msg: "El login y la contraseña son correctos, en breve recibirás un pin para introducir." } // formateamos respuesta  
    log.Println("Usuario "+ usuario.Email + " se ha logeado correctamente")  
} else {  
    r = RespLogin(Ok: false, Msg: "La contraseña no es correcta. Vuelva a intentarlo.") // formateamos respuesta  
    log.Println("Usuario "+ usuario.Email + " ha introducido contraseña incorrecta")  
}
```

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 incluido.

El correo en concreto es: 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.

```
for i, m := range respuesta.Entradas {
    entradas[i] = Entrada{m.Login, descifrarContraseñaEntrada(m.Password), m.Web, m.Descripcion}
    fmt.Println(entradas[i])
}

//Generamos el hash del password a partir del password para enviarla al servidor
//ya con dicho hash
claveCliente := sha512.Sum512([]byte(password))
passwordHash := base64.StdEncoding.EncodeToString(claveCliente[0:32]) // una mitad para cifrar datos (256 bits)
claveMaestra = claveCliente[32:64] // una mitad para cifrar datos (256 bits)
//fmt.Println(passwordHash, claveMaestra)

for i, m := range entradas {
    entradas[i] = Entrada{m.Login, cifrarContraseñaEntrada(m.Password), m.Web, m.Descripcion}
}

//Generamos los parámetros a enviar al servidor
parametros2 := url.Values{}
parametros2.Set("opcion", "10")
//Pasamos el parámetro a la estructura Usuario
usuario := UsuarioMod{Email: usuarioActual.Email, Password: passwordHash, Entradas: entradas}
```

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.

```

25 2017/05/26 00:21:44 listen: http: Server closed
26 2017/05/26 00:22:22 Usuario infoouadi@gmail.com se le ha enviado pin para login con exito
27 2017/05/26 00:22:22 Usuario infoouadi@gmail.com se ha logeado correctamente
28 2017/05/26 00:22:54 Usuario infoouadi@gmail.com ha tardado en poner el pin
29 2017/05/26 00:23:16 Usuario infoouadi@gmail.com se le ha enviado pin para login con exito
30 2017/05/26 00:23:16 Usuario infoouadi@gmail.com se ha logeado correctamente
31 2017/05/26 00:23:28 Usuario infoouadi@gmail.com ha puesto pin correcto
32 2017/05/26 00:23:30 Usuario infoouadi@gmail.com solicita sus entradas
33 2017/05/26 00:23:33 Usuario infoouadi@gmail.com ha cerrado sesion
34 2017/05/26 00:26:27 Volcando datos recopilados en la BBDD...
35 2017/05/26 00:26:27 Apagando servidor ...
36 2017/05/26 00:26:27 Servidor detenido correctamente
37 2017/06/02 20:09:27 Usuario isiana96@gmail.com se ha registrado correctamente
38
39 2017/06/02 20:09:38 Usuario isiana96@gmail.com se le ha enviado pin para login con exito
40 2017/06/02 20:09:38 Usuario isiana96@gmail.com se ha logeado correctamente
41 2017/06/02 20:13:05 Usuario isiana96@gmail.com ha tardado en poner el pin
42 2017/06/02 20:13:52 Usuario isiana96@gmail.com se le ha enviado pin para login con exito
43 2017/06/02 20:13:52 Usuario isiana96@gmail.com se ha logeado correctamente
44 2017/06/02 20:15:10 Usuario isiana96@gmail.com ha tardado en poner el pin
45 2017/06/02 20:16:25 Usuario garciamenarguez@gmail.com se ha registrado correctamente
46
47 2017/06/02 20:16:36 Usuario garciamenarguez@gmail.com se le ha enviado pin para login con exito
48 2017/06/02 20:16:36 Usuario garciamenarguez@gmail.com se ha logeado correctamente
49 2017/06/02 20:16:49 Usuario garciamenarguez@gmail.com ha puesto pin incorrecto
50 2017/06/02 20:18:07 Volcando datos recopilados en la BBDD...
51 2017/06/02 20:18:07 Apagando servidor ...
52 2017/06/02 20:18:07 Servidor detenido correctamente
53 2017/06/02 20:18:07 listen: http: Server closed
54 2017/06/02 20:18:39 Usuario garciamenarguez@gmail.com se le ha enviado pin para login con exito
55 2017/06/02 20:18:39 Usuario garciamenarguez@gmail.com se ha logeado correctamente
56 2017/06/02 20:19:34 Usuario garciamenarguez@gmail.com ha puesto pin correcto
57 2017/06/02 20:37:15 Usuario garciamenarguez@gmail.com se le ha enviado pin para login con exito
58 2017/06/02 20:37:15 Usuario garciamenarguez@gmail.com se ha logeado correctamente
59 2017/06/02 20:37:29 Usuario garciamenarguez@gmail.com ha puesto pin correcto

```

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

```

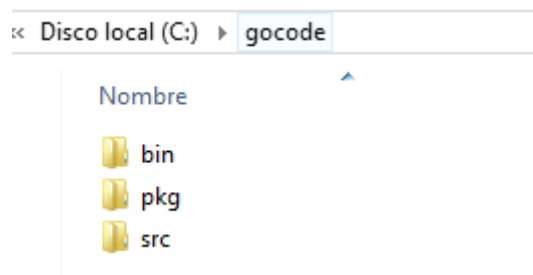
log.Println("Usuario "+ usuario.Email + " ha puesto pin correcto")
}else{
    r = Resp{Ok: false, Msg: "El pin no es correcto. Vuelve a intentarlo.", Dato: Token{}} // formateamos respuesta
    log.Println("Usuario "+ usuario.Email + " ha puesto pin incorrecto")
}
}else{
    r = Resp{Ok: false, Msg: "Se ha caducado el tiempo de sesion del pin.", Dato: Token{}} // formateamos respuesta
    log.Println("Usuario "+ usuario.Email + " ha tardado en poner el pin")
}

```

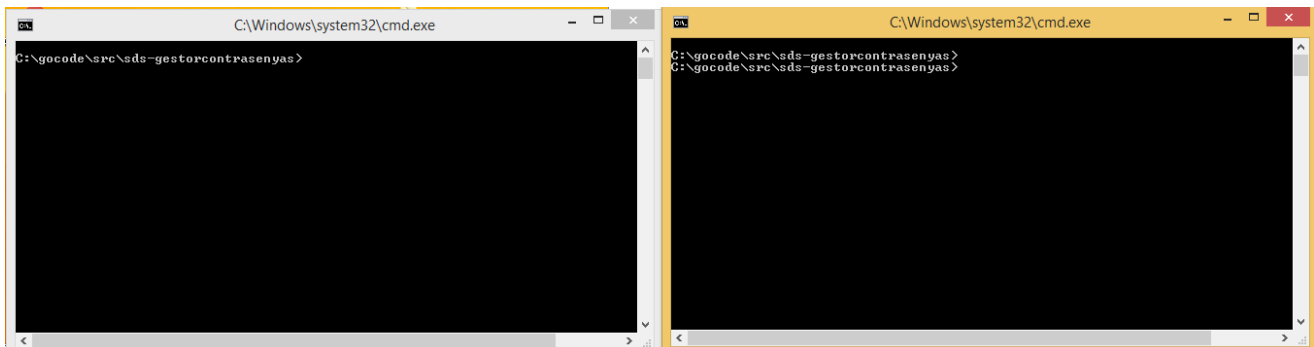

7- Despliegue de la aplicación

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

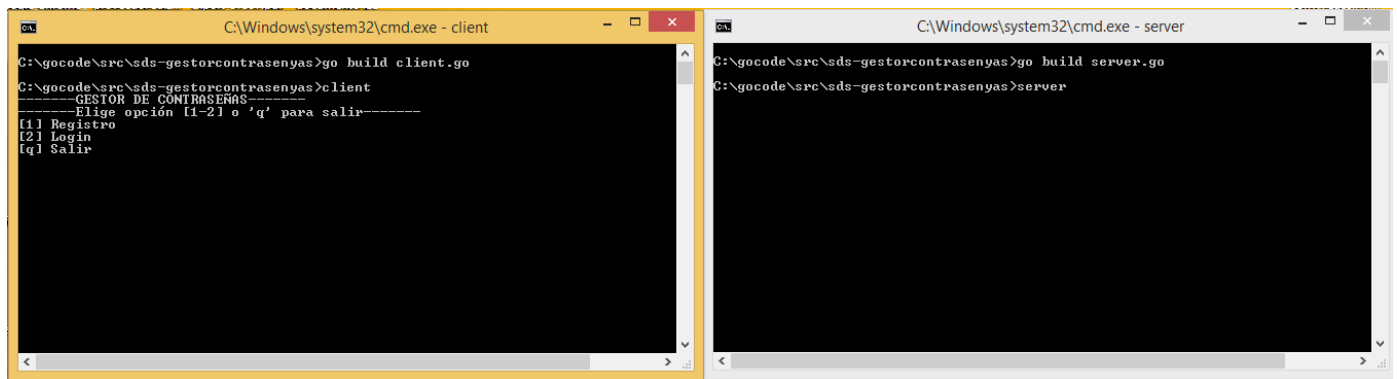
1. Suponiendo que el sistema operativo es Windows 8.1 o, en su defecto, cualquier Windows.
2. Instalación de Go 1.8.
3. Crear carpeta "C:\gocode" que será la carpeta donde se creen los proyectos desarrollados, en nuestro caso, la práctica en sí. Crear dentro de ella las tres subcarpetas: bin, pkg y src.



4. Crear/Modificar variables de entorno:
 - a. Crear variable "GOROOT" con valor "C:\Go\".
 - b. Crear variable "GOPATH" con valor "C:\gocode".
 - c. Modificar variable "Path" y añadir al final: ";C:\Go\bin;".
5. Copiar carpeta de la práctica "sds-gestorcontrasenyas" en la subcarpeta "src" antes creada, quedando: "C:\gocode\src\sds-gestorcontrasenyas".
6. 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`:



7. Compilar y ejecutar:



```
C:\Windows\system32\cmd.exe - client
C:\gocode\src\sds-gestorcontrasenyas>go build client.go
C:\gocode\src\sds-gestorcontrasenyas>client
-----GESTOR DE CONTRASEÑAS-----
-----Elige opción [1-2] o 'q' para salir-----
[1] Registro
[2] Login
[q] Salir

C:\Windows\system32\cmd.exe - server
C:\gocode\src\sds-gestorcontrasenyas>go build server.go
C:\gocode\src\sds-gestorcontrasenyas>server
```

8- 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.

9- 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 |