UNIVERSIDAD CASTILLA
LA-MANCHA

INGENIERÍA INFORMÁTICA

SEGURIDAD EN REDES

LABORATORIO 3

Cristina Serrano Trujillo

ÍNDICE

1. INTRODUCCIÓN

En esta práctica, se nos plantea el desafío de implementar un prototipo de base de datos como servicio, específicamente orientado a la gestión de documentos en formato JSON. El sistema será accesible a través de una API RESTful, lo que permitirá a los usuarios almacenar y gestionar sus documentos de forma sencilla. El servidor se ejecutará en el endpoint https://myserver.local:5000, donde myserver.local resolverá a 127.0.0.1.

Los principales objetivos de esta práctica son: conocer e implementar una API RESTful básica, implementar mecanismos de identificación y autenticación de usuarios, y asegurar la confidencialidad de las comunicaciones mediante el uso de HTTPS.

Cada usuario contará con un espacio personal para almacenar sus documentos JSON. Los usuarios podrán registrarse o iniciar sesión en el sistema, y posteriormente, podrán interactuar con la base de datos utilizando un token temporal, el cual se generará durante el proceso de inicio de sesión o registro.

Para la gestión de la autenticación, se utiliza un archivo .shadow, que simula el comportamiento de un sistema Linux, almacenando el nombre de usuario y la contraseña de forma segura. La estructura de almacenamiento en este archivo es la siguiente: usuario:salt:hash, donde salt es el código utilizado para generar el hash de la contraseña, garantizando así una capa adicional de seguridad.

Esta práctica consta de un único archivo Go denominado main.go, cinco scripts: version.sh, signup.sh, login.sh, doc_id.sh y alldocs.sh; y un Makefile para facilitar la compilacion de este proyecto.

2. CERTIFICADOS SSL/TLS

2.1. CONFIGURACIÓN BÁSICA DEL SERVIDOR

Existen varias herramientas para generar certificados SSL/TLS, entre las más comunes se encuentran mkcert y openssl.

En este proyecto, opté por utilizar mkcert en lugar de openssl debido a su simplicidad y eficacia en entornos de desarrollo local. mkcert permite generar certificados SSL/TLS válidos localmente con un solo comando y automáticamente configura una autoridad certificadora (CA) local, lo que asegura la confianza del certificado en la máquina donde se genera. Este enfoque elimina la necesidad de configuraciones manuales complejas, lo que acelera y facilita el proceso de desarrollo.

Por otro lado, openssl es una herramienta más flexible y adecuada para entornos de producción, pero requiere una comprensión más pasos adicionales, como la importación del certificado a los sistemas de los clientes y la configuración de una cadena de confianza. Por estas razones, openssl puede ser más complicado de usar en proyectos de desarrollo donde la prioridad es la rapidez y la simplicidad.

A continuación, se describen los comandos utilizados con ambas herramientas y una breve explicación de su funcionamiento:

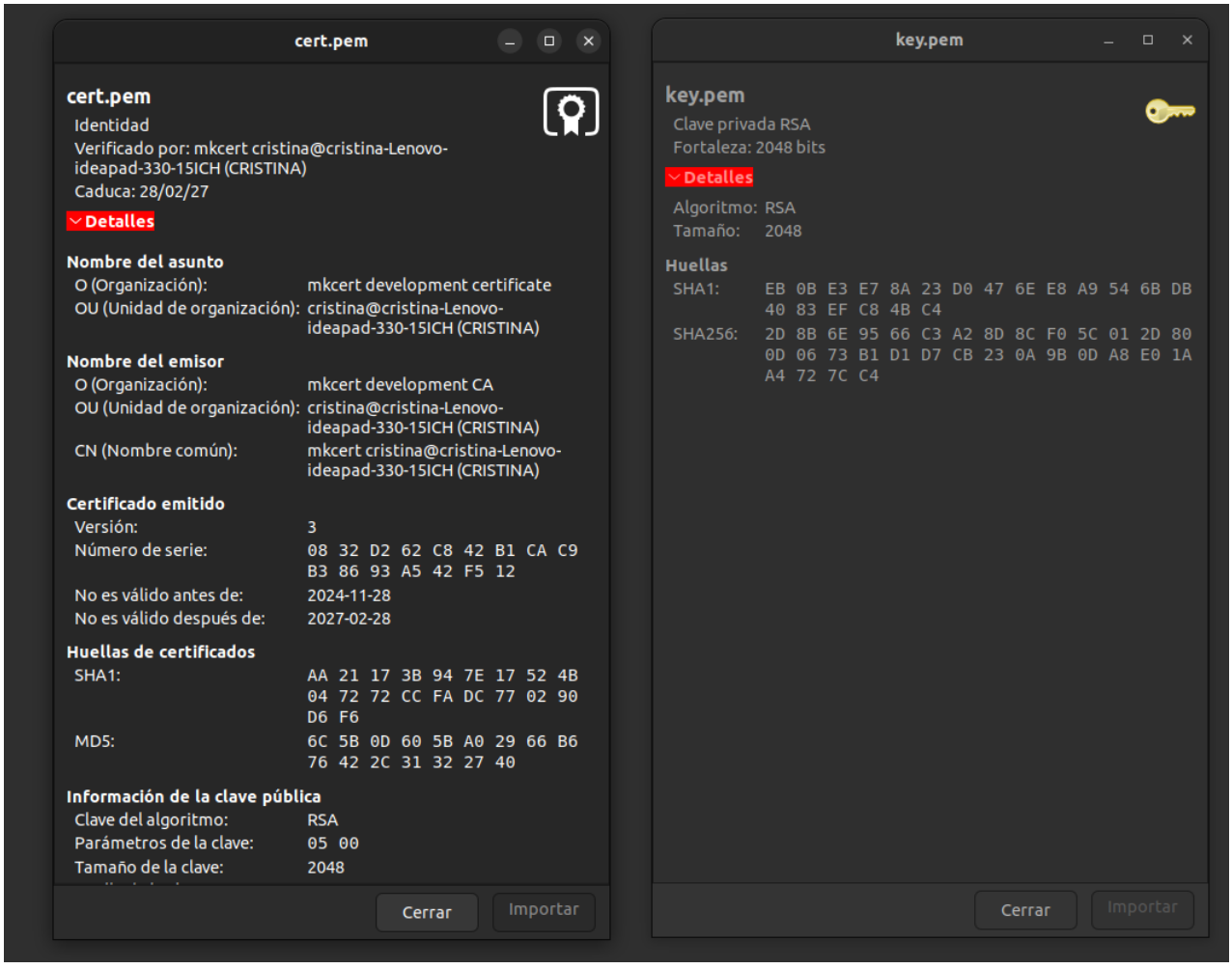
COMANDO MKCERT:

mkcert -key-file certs/key.pem -cert-file certs/cert.pem myserver.local

Este comando genera un certificado SSL/TLS autofirmado para un servidor local.

- key.pem: La clave privada del servidor.
- cert.pem: El certificado público del servidor.
- myserver.local: El nombre del servidor para el cual se genera el certificado.

Explicación: mkcert crea automáticamente una CA local y firma el certificado con ella, lo que hace que el certificado sea confiable solo en la máquina local. Este enfoque es ideal para entornos de desarrollo, donde no es necesario que el certificado sea válido a nivel global.



COMANDO OPENSSL:

openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout certs/server.key -out certs/server.crt

Este comando genera un certificado SSL/TLS autofirmado utilizando openssl.

- server.key: La clave privada del servidor.
- server.crt: El certificado público del servidor.

Explicación: openssl genera un certificado autofirmado, lo que significa que no está firmado por una CA reconocida, por lo que los navegadores mostrarán advertencias de seguridad. Este comando permite personalizar aspectos del certificado, como el tamaño de la clave y su duración. Para usar este certificado en un entorno de producción, es necesario importar la CA o configurar una cadena de confianza. Durante el uso de este comando, se deben completar varios pasos manualmente, lo que puede resultar tedioso, aunque se puede automatizar con un script. Adjunto foto de la clave privada que me generó:

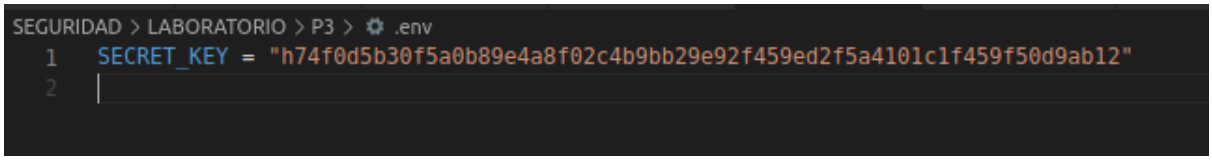

```
-----BEGIN PRIVATE KEY-----
MIIEvQIBADANBgkqhkiG9w0BAQEFAASCBKcwggSjAgEAAoIBAQCgMlc/170ZcsT8
eO3rn/5+HszsPzCCK2Ksk9sEfkbUjNT6XIKkD8sXrXJOxOt3jtIFKHiIFq6UFjEM
wyOR009VNFWpxHZwL/yGqkeS9ZDliEBx6lUCAWQs81Xmnz5nxY8RkPuxu7bxu6aY
k0VYS36oQ+Df5vIgvKyJDYzyXK3+r8eZGr3uYMqV6j3RXnXUYhOUY42AF8UThoU3
EiruZMe1aFGMCYD79Rp9esnHXtlxamVEiD0yFAGdBqP8MbGDqdxOyXFL0SzYMjHy
sQGnLpqX9wieac/UFekVsa5VBL5Ls9m76yVKh0VR9Mi78wB/mCB0jrtWzE7QTPtY
b8hDRSfzAgMBAAECggEAALiZSfFHqaCpoFVPBNuAgsLSu1GBf2SAmjRELR4AZZ6I
quPVtrVXfbDLLe2W/b/H1DwdZsdrW8FJ0TvjohloZwLIS+5VdiXRJi4ICnJOh6n4
WErsvQiSkd06isI2vrLV9QdJCsDEjwwAMPmgu0gDtvAvR/K7LkiLIkbMVp7yLY5N
K/2KAgSOK8Bypm8rOqnthnvuXEwBSQwE0X1UmLAlekhcWJEFOXWnpFWsDi/ap3Ky
qXcVww4VdBLMysO+wPAPkG6EPoNV420ObPNKPavtoF2y2Is8pJM7DL3cK8EiiFwh
A5o1REMdbp8WtxCN0gqZCT+p0Kc/QvEPbTN4hJJKcQKBgQDeGcMILZm75qwB6ZOU
tJQFL2PUu+BBUSXtXNQ0qAzNgbQJ4Qn1Y/b6dqIQ4iPZlnij+IWfFKBSZwbyvPq7
Tcm7Vu3SUGlHKNry31s+1FCDOwnVCrNupHZTgkQVABrNtitMeB8rwjMiA+NJmbrA
m/KWUgPpuvmW254uu/h7Zb4dqQKBgQC4pcbKGM3WAuHmGIvShFVEtU7TeGa8/R+g
FtvTi/2VKkx3db6kV+FmU87WhdnxtVLaUk3rOjkZ0D8og03xR2R3UgnlP3VvAPYu
lOItmJQR5p/IoKhVMzRNkqN6bPJeXWuXCzJ9B5BH+yz5XCDrMkDO0vc88NAwamyG
9fAq/BECOwKBgCmseczqCYOZV8MQGdH4RCHo73l2OzDmUhCEr4il06kn4ilMYXr6
49fBNM2oMQPd2QsjUac0zpRHoqUfUoicovKU8BDbZ7Wjc9nIkS1r1yOYi1K/LEXh
dBZdRfsOxvi8aslxbJ/gU4nZYudnggQr/su33eYYYEavNLIeMRSuVYrpAoGARU/t
bEWdVr2kQLRsIC0OuQj6OwWSQ1UdPwH1qNZ+7TSAmcM4OgbHFJtXJE4Afuwa2ttU
Zn1nm9oBK1sGshCjlOrVzhlhIrcQsnGu6YDB7GpPBofQSJM9CfWGqlklLekRjxRj
8m0ZavvNaRl9PeySSVEF//lBnTsF3C+L5QbT96sCgYEAmYkrQZfLn3iYJL3bTN0p
zi12pIl/UhEPwoSE9QhkEsgeaNXpjzAJEWlZ+C6is4nc1HZaQvyzzU6351jMsGpP
B6gNtSKsJw436kqdnCXJHjTLdhuzVTBBshd0W5kx5wC+iJaJgzoSetxVaFPL56ee
MU+2bW7FtFoJj5TfBv3g8jo=
-----END PRIVATE KEY-----
```


Dado que mkcert simplifica considerablemente el proceso de generación del certificado y es más adecuado para pruebas locales, se decidió utilizar esta herramienta para este proyecto. En este caso, la validez global del certificado no es una prioridad, lo que hace que mkcert sea la opción más conveniente para un entorno de desarrollo.

3. ARCHIVO .ENV

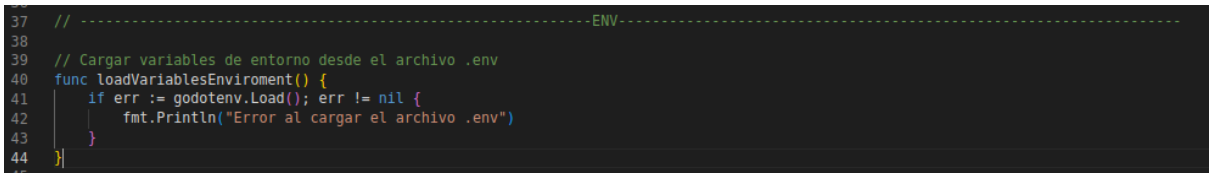
En esta sección explicaré cómo cargar y utilizar variables de entorno, específicamente la clave secreta, de manera segura desde un archivo .env. El uso de un archivo .env ayuda a mantener la clave privada fuera del código fuente, evitando que sea visible para todos los desarrolladores o en sistemas de control de versiones como Git. Esto mejora la seguridad de las aplicaciones al asegurar que las claves sensibles no estén expuestas.

En primer lugar cree un archivo .env y definí el valor SECRET_KEY tal como se ve en la siguiente imagen:



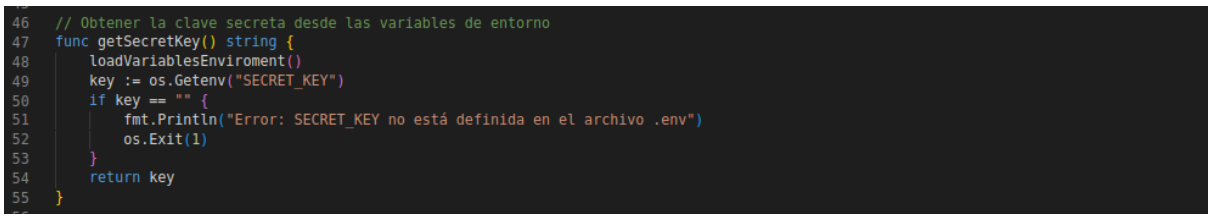
A continuación, procedo a explicar la implementación realizada para este archivo:

3.1. LoadVariablesEnviroment():



La función loadVariablesEnviroment() carga las variables de entorno definidas en el archivo .env, en este caso SECRET_KEY, utilizando la librería godotenv. Esta librería lee el archivo .env y establece la variable de entorno en el entorno de ejecución de la aplicación.

3.2. GetSecretKey():



La función **getSecretKey()** primero llama a **loadVariablesEnviroment()** para cargar las variables del archivo .env. Luego, obtiene el valor de la clave secreta usando *os.Getenv("SECRET_KEY")*. Si la clave no está definida en el archivo .env, la función muestra un mensaje de error y detiene la ejecución con *os.Exit(1)*. Si la clave existe, la devuelve.

4. MIDDLEWARE

En este código, he implementado un middleware que verifica la autenticación de los usuarios antes de que sus solicitudes lleguen a los manejadores de las rutas que requieren autenticación. De esta manera, me aseguro de que solo los usuarios autenticados y autorizados puedan acceder a recursos protegidos, mientras que se permite el acceso sin autenticación a rutas como el inicio de sesión y registro.

4.1. IsValidusername():

```go
// ----------------------------------------------------------MIDDLEWARE----------------------------------------------------------

// Función para validar el formato del username
func isValidusername(username string) bool {
    if len(username) == 0 {
        return false
    }
    for _, char := range username {
        if !unicode.IsLetter(char) && !unicode.IsDigit(char) && char != '-' {
            return false
        }
    }
    return true
}
```

Esta función valida el formato del nombre de usuario proporcionado. Asegura que el nombre de usuario no esté vacío y que solo contenga caracteres alfabéticos, numéricos o el guion (-). Si el nombre de usuario no cumple con estos requisitos, la función retorna false.
La validación del nombre de usuario es crucial para evitar entradas maliciosas que puedan contener caracteres no válidos o peligrosos, previniendo posibles inyecciones o manipulaciones de los datos de entrada.

4.2. AuthorizationMiddleware():

```go
// Middleware que comprueba la cabecera Authorization para asegurarse de que el usuario está autorizado a realizar la petición.
func AuthorizationMiddleware() gin.HandlerFunc {
    return func(c *gin.Context) {
        username := c.Param("username")
        fmt.Println("Valor de username recibido:", username)

        // Permitir acceso a las rutas de login y signup sin token
        if c.FullPath() == "/login" || c.FullPath() == "/signup" {
            c.Next()
            return
        }

        // Verificar si el username está presente y tiene un formato válido
        username = strings.TrimSpace(username)
        if !isValidusername(username) {
            fmt.Println("ID de usuario no válido o ausente:", username)
            c.JSON(404, gin.H{"Mensaje": "ID de usuario no válido o ausente"})
            c.Abort()
            return
        }

        // Comprobar cabecera Authorization y verificar el token
        if checkAuthorizationHeader(c, username) {
            c.Next()
        } else {
            fmt.Println("Autorización fallida para el usuario:", username)
            c.JSON(401, gin.H{"Mensaje": "Acceso no autorizado"})
            c.Abort()
        }
    }
}
```

El middleware **AuthorizationMiddleware** tiene la responsabilidad de validar la autenticación del usuario antes de que la solicitud sea procesada por las rutas protegidas.

1. Si la ruta solicitada es /login o /signup, el middleware permite el paso sin necesidad de autenticación.
2. Si la ruta no es de inicio de sesión o registro, el middleware verifica si el nombre de usuario es válido (utilizando la función **isValidusername**). Si el nombre de usuario no es válido, se aborta la solicitud y se responde con un mensaje de error.
3. Si el nombre de usuario es válido, se procede a verificar la cabecera *Authorization* y el *token* de autenticación a través de la función **checkAuthorizationHeader**. Si la verificación es exitosa, la solicitud continúa; si no, se responde con un error de autorización.

Este middleware proporciona una capa de seguridad adicional al asegurar que solo los usuarios autenticados, con un token válido, puedan acceder a los recursos protegidos. Evita el acceso no autorizado a rutas sensibles.

4.3. CheckAuthorizationHeader():

```
105  // Comprueba la cabecera Authorization y verifica el token del usuario.
106  func checkAuthorizationHeader(c *gin.Context, username string) bool {
107      signup := Signup{}
108      // Obtiene la cabecera Authorization de la petición
109      authHeader := c.GetHeader("Authorization")
110      fmt.Println("Cabecera de Autorización:", authHeader)
111      header := strings.Split(authHeader, " ")
112      if len(header) != 2 || strings.ToLower(header[0]) != "token" {
113          c.JSON(400, gin.H{"Mensaje": "Formato de cabecera no válido"})
114          return false
115      }
116
117      // Extrae el token de la cabecera
118      token := header[1]
119
120      // Verifica la autenticidad del token
121      if signup.VerifyToken(username, token, c) {
122          return true
123      } else {
124          c.JSON(401, gin.H{"Mensaje": "Token no válido o ausente"})
125          return false
126      }
127  }
```

La función **checkAuthorizationHeader** verifica que la solicitud incluya una cabecera *Authorization* con un *token* válido.

1. Extrae la cabecera *Authorization* de la solicitud, separa el valor del tipo de autorización (*token*) y el propio *token*.
2. Si la cabecera no tiene el formato correcto (por ejemplo, falta el token o el tipo no es *token*), devuelve un error con el mensaje "Formato de cabecera no válido".
3. Si la cabecera tiene el formato correcto, extrae el *token* y verifica su validez utilizando el método **VerifyToken** de la estructura **Signup**.
4. Si el *token* es válido, permite que la solicitud continúe; si no, responde con un error indicando que el *token* es inválido o ausente.

Este paso es fundamental para la autenticación basada en **tokens**.

5. VERSION DE LA API

Este fragmento de código se encarga de gestionar la información sobre la versión actual de la API. La función permite que los usuarios puedan consultar la versión de la API a través de una solicitud HTTP.

```
129  // ------------------------------------------------------------VERSION-------------------------------------------------------------
130
131  // VERSION: Contiene la versión actual de la API
132  func (v *Version) Get(c *gin.Context) {
133      fmt.Println("Version: ", VERSION)
134      c.JSON(200, gin.H{"Version:": VERSION})
135  }
136
```

La función Get responde a las solicitudes realizadas para obtener la versión de la API. Al recibir la solicitud, la función imprime en la consola la versión actual de la API (VERSION), lo que puede ser útil para depuración o registros de actividad. Luego, la función responde con un código de estado HTTP 200 (OK) y devuelve la versión de la API en formato JSON. El objeto JSON tiene una clave "Version:" que contiene el valor de la versión actual de la API.

Esta función no contiene validaciones de seguridad, ya que solo devuelve información pública sobre la versión de la API. No se requiere autenticación ni autorización para acceder a esta ruta.

6. GESTIÓN DE REGISTRO DE USUARIO (SIGNUP):

Aquí se implementa la lógica para registrar usuarios en un sistema utilizando un archivo .shadow para almacenar credenciales y un sistema de autenticación basado en tokens JWT. A continuación, explicaré los métodos clave.

6.1. CreateUserSpace():

```
137  // ------------------------------------------------------------SIGNUP--------------------------------------------------------------
138
139  // Create a d   irectory for the user
140  func (s *Signup) CreateUserSpace(username string, c *gin.Context) {
141      if err := os.MkdirAll("users/"+username, 0755); err != nil {
142          c.JSON(500, gin.H{"Error": "Error al crear el espacio del usuario"})
143          return
144      }
145      fmt.Println("¡Espacio de usuario creado correctamente!")
146  }
147
```

Crea un directorio para el usuario en el sistema de archivos. Username es el nombre del usuario para el cual se crea el directorio, y c es el contexto de Gin para gestionar la respuesta HTTP.

La salida es una respuesta HTTP con un código 500 en caso de error y mensaje de éxito si el directorio es creado correctamente.

6.2. RegisterUser():

```go
// Registrar un usuario en el archivo shadow
func (s *Signup) RegisterUser(username, password string, c *gin.Context) {
    shadowFile, err := os.OpenFile(SHADOW_FILE, os.O_APPEND|os.O_WRONLY|os.O_CREATE, 0644)
    if err != nil {
        c.JSON(500, gin.H{"Error": "Error al abrir el archivo shadow"})
        return
    }
    defer shadowFile.Close()

    // Generar un salt aleatorio
    salt := uuid.New().String()

    // Añadir un salto de línea si el archivo no está vacío
    if _, err := shadowFile.Stat(); err == nil {
        _, _ = shadowFile.WriteString("\n")
    }

    // Encriptar la contraseña usando el salt generado
    hashedPassword := s.EncryptPassword(salt, password)

    // Escribir la información en el archivo shadow
    if _, err := shadowFile.WriteString(fmt.Sprintf("%s:%s:%s\n", username, salt, hashedPassword)); err != nil {
        c.JSON(500, gin.H{"Error": "Error al escribir en el archivo shadow"})
        return
    }

    fmt.Println("¡Usuario registrado correctamente!")
}
```

Registra un nuevo usuario y su contraseña en un archivo .shadow. La contraseña es encriptada con un "salt" aleatorio antes de ser almacenada

En cuanto a sus parámetros: Username es el nombre del usuario, password es la contraseña del usuario y c es el contexto de Gin para gestionar la respuesta HTTP.

La salida generada es una respuesta HTTP con un código 500 si hay un error al abrir o escribir en el archivo, o un mensaje de éxito si el registro es exitoso, como en el método anterior.

6.3. CheckUsername():

```go
// Método para comprobar si un usuario ya está registrado
func (s *Signup) CheckUsername(username string) bool {

    // Abre el archivo shadow
    shadowFile, err := os.Open(SHADOW_FILE)
    if err != nil {
    return false
    }
    defer shadowFile.Close()

    // Lee el archivo línea por línea
    scanner := bufio.NewScanner(shadowFile)
    for scanner.Scan() {
        line := scanner.Text()
        if strings.Split(line, ":")[0] == username {
            return true
        }
    }
    return false
}
```

Verifica si un nombre de usuario ya está registrado en el archivo shadow. Devuelve true si el usuario ya está registrado y false si no lo está.

6.4. EncryptPassword():

```go
// Encriptar la contraseña con SHA-256 y añadir un salt
func (s *Signup) EncryptPassword(salt, password string) string {

    // Crear un hash SHA-256
    hash := sha256.New()

    // Concatena el salt y la contraseña y hashea el resultado
    hash.Write([]byte(salt + password))

    // Devuelve el hash en formato hexadecimal
    return hex.EncodeToString(hash.Sum(nil))
}
```

Encripta la contraseña del usuario con el algoritmo SHA-256, añadiendo un "salt" para mayor seguridad. Los parámetros de este método son: Salt que es el valor aleatorio utilizado para encriptar la contraseña y password que es la contraseña a encriptar.

La salida que genera es la contraseña encriptada en formato hexadecimal.

6.5. GenerateAccessToken():

```go
// Generar un token de acceso para el usuario
func (s *Signup) GenerateAccessToken(username string, c *gin.Context) string {

    // Genera un token con un tiempo de expiración de 5 minutos
    exp := time.Now().Add(time.Minute * TIME_EXPIRATION).Unix()

    // Crea un JWT con el nombre de usuario y el tiempo de expiración
    claims := jwt.StandardClaims{
        Subject:   username,
        ExpiresAt: exp,
    }

    // Crea un token firmado con el algoritmo HS256 y la clave secreta
    token := jwt.NewWithClaims(jwt.SigningMethodHS256, claims)
    signedToken, err := token.SignedString([]byte(getSecretKey()))
    if err != nil {
        c.JSON(500, gin.H{"Error": "Error al firmar el token"})
        return ""
    }
    return signedToken
}
```

Genera un token JWT para un usuario con un tiempo de expiración definido (5 minutos en este caso). Devuelve un token JWT firmado que puede ser utilizado para autenticación.

6.6. SignUp():

```go
// Método para registrar un usuario
func (s *Signup) SignUp(c *gin.Context) {
    var jsonInput map[string]string
    if err := c.BindJSON(&jsonInput); err != nil {
        c.JSON(400, gin.H{"Error": "Formato inválido."})
        return
    }

    // Comprueba si los campos 'Usuario' y 'Contraseña' están presentes
    username, uOk := jsonInput["Usuario"]
    password, pOk := jsonInput["Contraseña"]
    if !uOk || !pOk {
        c.JSON(400, gin.H{"Error": "Los argumentos deben ser 'Usuario' or 'Contraseña'"})
        return
    }

    // Comprueba si el usuario ya está registrado
    if s.CheckUsername(username) {
        c.JSON(409, gin.H{"Error": fmt.Sprintf("Error, el usuario %s ya existe. Por favor, pruebe de nuevo.", username)})
        return
    }

    // Registra al usuario y crea su espacio
    s.RegisterUser(username, password, c)
    s.CreateUserSpace(username, c)

    // Genera un token de acceso y lo almacena en el diccionario
    token := s.GenerateAccessToken(username, c)
    TOKENS_DICT[username] = token

    c.JSON(200, gin.H{"access_token": token})
}
```

Maneja el proceso de registro de un nuevo usuario, recibe los datos del usuario en formato JSON y verifica que los campos "Usuario" y "Contraseña" estén presentes. Además, comprueba si el usuario ya está registrado. Si no lo está registra al usuario y crea su espacio en el sistema de archivos. Por último, genera un token JWT y lo devuelve al usuario.

6.7. VerifyToken():

```go
// Método para verificar un token
func (s *Signup) VerifyToken(username, tokenString string, c *gin.Context) bool {
    token, err := jwt.ParseWithClaims(tokenString, &jwt.StandardClaims{}, func(token *jwt.Token) (interface{}, error) {
        // Proporciona la clave utilizada para firmar el token
        return []byte(getSecretKey()), nil
    })

    if err != nil {
        var message string
        if err == jwt.ErrSignatureInvalid {
            message = "Firma del token inválida"
        } else if ve, ok := err.(*jwt.ValidationError); ok && ve.Errors == jwt.ValidationErrorExpired {
            message = "El token ha expirado"
        } else {
            message = "Error al analizar el token"
        }
        c.JSON(401, gin.H{"Error": message})
        return false
    }


    // Comprueba si el token es válido y si los claims coinciden con el nombre de usuario proporcionado
    if claims, ok := token.Claims.(*jwt.StandardClaims); ok && token.Valid {
        if claims.Subject == username {
            return true
        } else {
            c.JSON(401, gin.H{"Error": "El token no coincide con el usuario"})
            return false
        }
    }

    c.JSON(401, gin.H{"Error": "Token inválido"})
    return false
}
```

Verifica la validez de un token JWT. Es decir, verifica si la firma es válida y si el token ha expirado. Si el token es válido y corresponde al usuario indicado, devuelve true, de lo contrario, responde con un error.

7. GESTIÓN DE INICIO DE SESIÓN (LOGIN)

Aquí se implementa la lógica para iniciar sesión en el sistema utilizando el archivo .shadow para verificar las credenciales del usuario y generar un token JWT. Procedo a explicar los metodos que lo componen:

7.1. CheckCredentials():

```go
// ----------------------------------------------------------------LOGIN-----------------------------------------------------------------

// Comprueba las credenciales del usuario
func (l *Login) CheckCredentials(username, password string, c *gin.Context) bool {

    signup := Signup{}

    // Abre el archivo shadow
    shadowFile, err := os.Open(SHADOW_FILE)
    if err != nil {
        c.JSON(500, gin.H{"error": "Error opening shadow file"})
        return false
    }
    defer shadowFile.Close()

    // Lee el archivo línea por línea
    scanner := bufio.NewScanner(shadowFile)
    for scanner.Scan() {
        line := scanner.Text()
        credentials := strings.Split(line, ":")
        if credentials[0] == username {
            // Encripta la contraseña proporcionada con la salt almacenada en el archivo shadow
            hashedPassword := signup.EncryptPassword(credentials[1], password)
            if err != nil {
                c.JSON(500, gin.H{"error": "Error encrypting password"})
                return false
            }
            // Comprueba si la contraseña encriptada coincide con la almacenada en el archivo shadow
            if strings.TrimSpace(credentials[2]) == hashedPassword {
                return true
            }
        }
    }
    return false
}
```

Verifica las credenciales del usuario. Compara el nombre de usuario y la contraseña proporcionada con la información almacenada en el archivo .shadow. La contraseña proporcionada se encripta utilizando el "salt" almacenado en el archivo y se compara con la contraseña encriptada guardada.

Devuelve true si las credenciales son válidas, false si no lo son. Además, si hay un error al abrir el archivo o al encriptar la contraseña, se devuelve un mensaje de error y false.

7.2. Login():

```go
// Inicia sesión en el sistema
func (l *Login) Login(c *gin.Context) {
    var jsonInput map[string]string
    if err := c.BindJSON(&jsonInput); err != nil {
        c.JSON(400, gin.H{"Error": "Formato inválido."})
        return
    }

    username, uOk := jsonInput["Usuario"]
    password, pOk := jsonInput["Contraseña"]
    if !uOk || !pOk {
        c.JSON(400, gin.H{"Error": "Los argumentos deben ser 'Usuario' o 'Contraseña'"})
        return
    }

    if l.CheckCredentials(username, password, c) {
        signup := Signup{}
        token, exists := TOKENS_DICT[username]
        if !exists || !signup.VerifyToken(username, token, c) {
            token = signup.GenerateAccessToken(username, c)
            TOKENS_DICT[username] = token
        }
        c.JSON(200, gin.H{"token_acceso": token})
    } else {
        c.JSON(401, gin.H{"Error": "Credenciales inválidas"})
    }
}
```

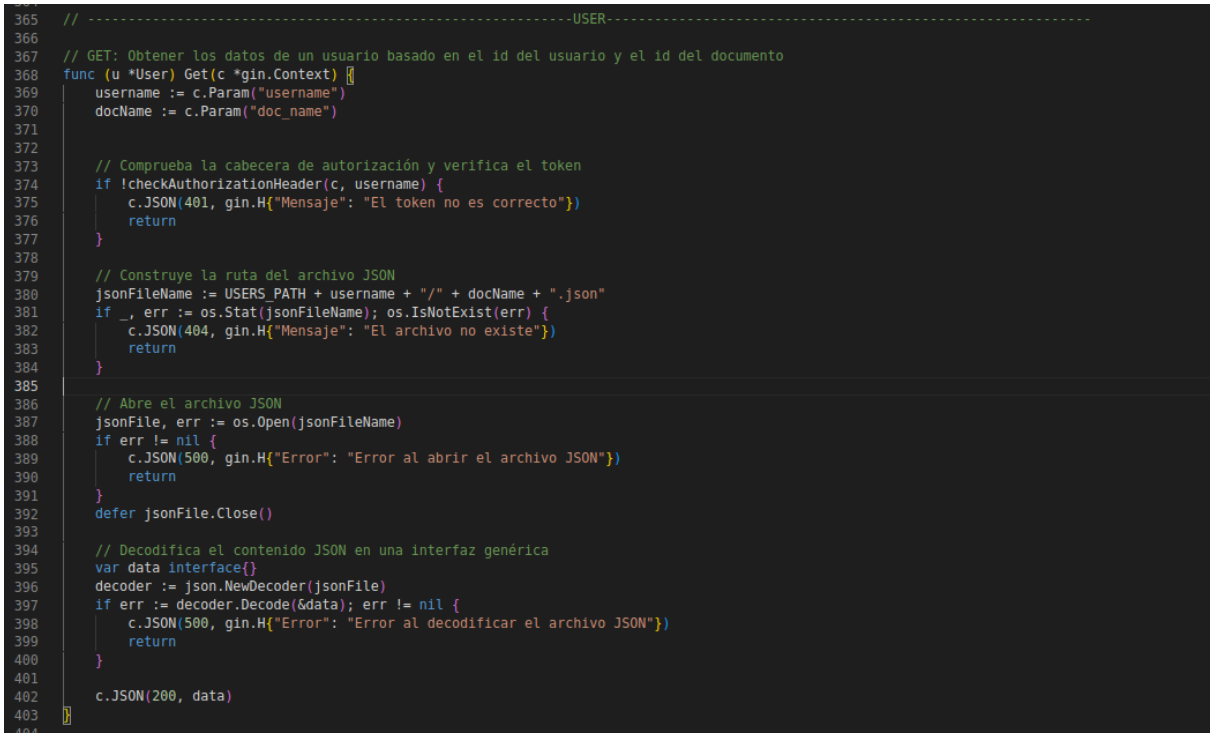
Este método maneja la autenticación del usuario y la generación de un token de acceso. Primero valida que los campos "Usuario" y "Contraseña" sean proporcionados. Luego, verifica las credenciales utilizando el método **CheckCredentials**. Si las credenciales son correctas, el sistema genera un token de acceso utilizando **GenerateAccessToken** y lo almacena en un diccionario (*TOKENS_DICT*). Si el token ya existe, se verifica su validez con **VerifyToken** antes de devolverlo.

Devuelve un token de acceso en formato JSON si el inicio de sesión es exitoso. Por otro lado, si las credenciales son inválidas, devuelve un error 401 con el mensaje "Credenciales inválidas", y si el formato de la solicitud es incorrecto, devuelve un error 400 con un mensaje apropiado.

8. GESTIÓN DE DOCUMENTOS (USER):

Esta sección implementa varias rutas en un servidor utilizando el framework Gin de Go. Las rutas están relacionadas con la gestión de documentos para un usuario, y las operaciones son GET, POST, PUT y DELETE. Procedo a explicar brevemente las funcionalidades de cada una:

8.1. GET():



```
// ------------------------------------------------------------USER-------------------------------------------------------------

// GET: Obtener los datos de un usuario basado en el id del usuario y el id del documento
func (u *User) Get(c *gin.Context) {
    username := c.Param("username")
    docName := c.Param("doc_name")


    // Comprueba la cabecera de autorización y verifica el token
    if !checkAuthorizationHeader(c, username) {
        c.JSON(401, gin.H{"Mensaje": "El token no es correcto"})
        return
    }

    // Construye la ruta del archivo JSON
    jsonFileName := USERS_PATH + username + "/" + docName + ".json"
    if _, err := os.Stat(jsonFileName); os.IsNotExist(err) {
        c.JSON(404, gin.H{"Mensaje": "El archivo no existe"})
        return
    }

    // Abre el archivo JSON
    jsonFile, err := os.Open(jsonFileName)
    if err != nil {
        c.JSON(500, gin.H{"Error": "Error al abrir el archivo JSON"})
        return
    }
    defer jsonFile.Close()

    // Decodifica el contenido JSON en una interfaz genérica
    var data interface{}
    decoder := json.NewDecoder(jsonFile)
    if err := decoder.Decode(&data); err != nil {
        c.JSON(500, gin.H{"Error": "Error al decodificar el archivo JSON"})
        return
    }

    c.JSON(200, data)
}
```

Obtiene un archivo JSON asociado a un usuario y un documento específico. Primero verifica el token de autorización en la cabecera. Si es correcto, abre y lee el archivo JSON, luego devuelve su contenido.

En cuanto a sus errores, devuelve un código de error 401 si el token no es válido, 404 si el archivo no existe, o 500 si hay un problema al abrir o leer el archivo.

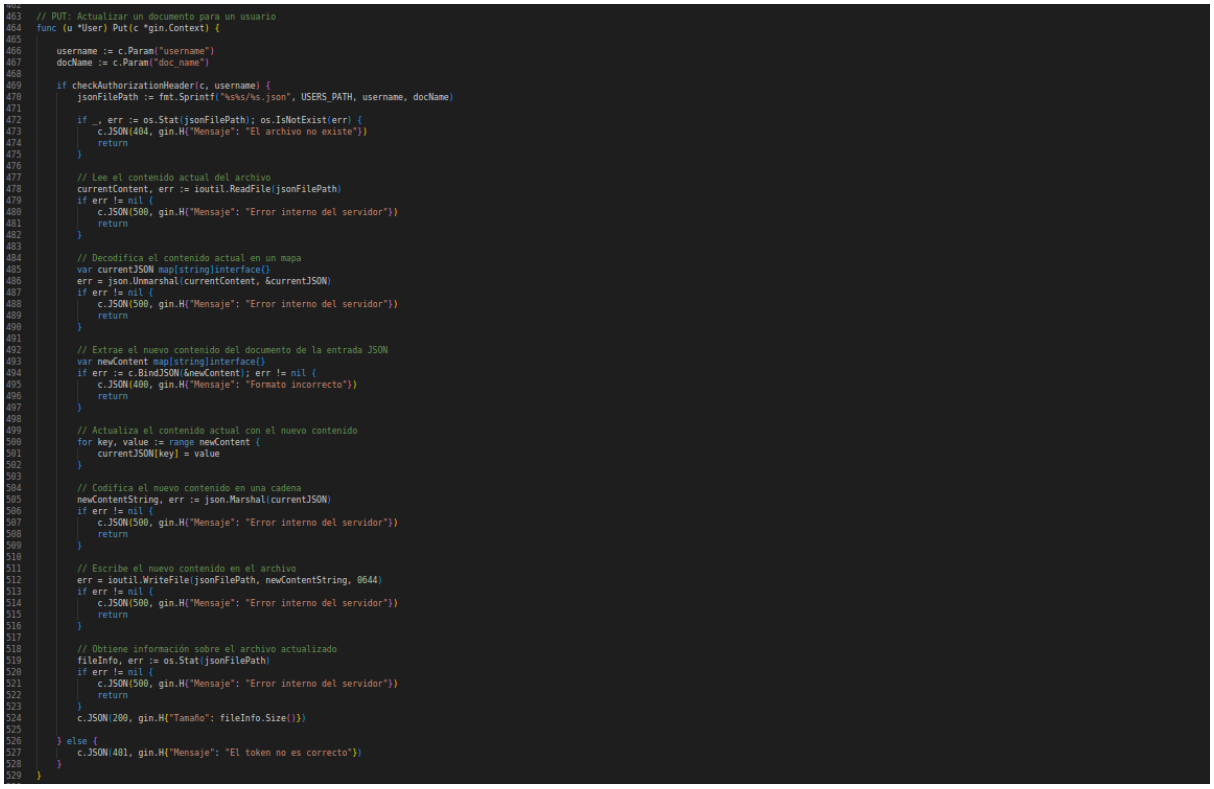
8.2. POST():

```go
// POST: Crear un nuevo documento para un usuario
func (u *User) Post(c *gin.Context) {

    username := c.Param("username")
    docName := c.Param("doc_name")

    // Comprueba la cabecera de autorización y verifica el token
    if checkAuthorizationHeader(c, username) {
        if _, err := os.Stat(USERS_PATH + username + "/" + docName + ".json"); !os.IsNotExist(err) {
            c.JSON(405, gin.H{"Mensaje": "El archivo ya existe. Utilice PUT para actualizar el archivo"})
            return
        }

        // Extrae el contenido del documento de la entrada JSON
        var jsonInput map[string]interface{}
        if err := c.BindJSON(&jsonInput); err != nil {
            c.JSON(400, gin.H{"Mensaje": "Formato incorrecto"})
            return
        }

        // Construye la ruta del archivo JSON
        docContent, exists := jsonInput["doc_content"]
        if !exists {
            c.JSON(400, gin.H{"Mensaje": "El campo 'doc_content' es obligatorio"})
            return
        }

        // Construye la ruta del archivo JSON
        jsonFileName := USERS_PATH + username + "/" + docName + ".json"

        // Codifica el contenido JSON en una cadena
        jsonString, err := json.Marshal(docContent)
        if err != nil {
            c.JSON(500, gin.H{"Mensaje": "Error interno del servidor"})
            return
        }

        // Escribe el contenido JSON en el archivo
        err = ioutil.WriteFile(jsonFileName, jsonString, 0644)
        if err != nil {
            c.JSON(500, gin.H{"Mensaje": "Error interno del servidor"})
            return
        }

        // Obtiene información sobre el archivo creado
        fileInfo, err := os.Stat(jsonFileName)
        if err != nil {
            c.JSON(500, gin.H{"Mensaje": "Error interno del servidor"})
            return
        }
        c.JSON(200, gin.H{"Tamaño": fileInfo.Size()})

    } else {
        c.JSON(401, gin.H{"Mensaje": "El token no es correcto"})
    }
}
```

Permite crear un nuevo documento para un usuario. Verifica que el archivo no exista ya, recibe el contenido del documento como JSON, lo guarda como un archivo .json y devuelve el tamaño del archivo.

Los errores que devuelve son: Un código 401 si el token no es válido, 400 si el formato del JSON es incorrecto o si falta el campo contenido_documento, 405 si el archivo ya existe.

8.3. PUT():



```go
// PUT: Actualizar un documento para un usuario
func (u *User) Put(c *gin.Context) {

    username := c.Param("username")
    docName := c.Param("doc_name")

    if checkAuthorizationHeader(c, username) {
        jsonFilePath := fmt.Sprintf("%s%s/%s.json", USERS_PATH, username, docName)

        if _, err := os.Stat(jsonFilePath); os.IsNotExist(err) {
            c.JSON(404, gin.H{"Mensaje": "El archivo no existe"})
            return
        }

        // Lee el contenido actual del archivo
        currentContent, err := ioutil.ReadFile(jsonFilePath)
        if err != nil {
            c.JSON(500, gin.H{"Mensaje": "Error interno del servidor"})
            return
        }

        // Decodifica el contenido actual en un mapa
        var currentJSON map[string]interface{}
        err = json.Unmarshal(currentContent, &currentJSON)
        if err != nil {
            c.JSON(500, gin.H{"Mensaje": "Error interno del servidor"})
            return
        }

        // Extrae el nuevo contenido del documento de la entrada JSON
        var newContent map[string]interface{}
        if err := c.BindJSON(&newContent); err != nil {
            c.JSON(400, gin.H{"Mensaje": "Formato incorrecto"})
            return
        }

        // Actualiza el contenido actual con el nuevo contenido
        for key, value := range newContent {
            currentJSON[key] = value
        }

        // Codifica el nuevo contenido en una cadena
        newContentString, err := json.Marshal(currentJSON)
        if err != nil {
            c.JSON(500, gin.H{"Mensaje": "Error interno del servidor"})
            return
        }

        // Escribe el nuevo contenido en el archivo
        err = ioutil.WriteFile(jsonFilePath, newContentString, 0644)
        if err != nil {
            c.JSON(500, gin.H{"Mensaje": "Error interno del servidor"})
            return
        }

        // Obtiene información sobre el archivo actualizado
        fileInfo, err := os.Stat(jsonFilePath)
        if err != nil {
            c.JSON(500, gin.H{"Mensaje": "Error interno del servidor"})
            return
        }
        c.JSON(200, gin.H{"Tamaño": fileInfo.Size()})

    } else {
        c.JSON(401, gin.H{"Mensaje": "El token no es correcto"})
    }
}
```

Actualiza un documento de un usuario. Verifica que el archivo ya exista, lee su contenido, recibe los nuevos datos a actualizar, los combina con el contenido existente y luego guarda el archivo actualizado.

Devuelve un código 401 si el token no es válido, 404 si el archivo no existe, 500 si hay un problema interno al leer o escribir el archivo.

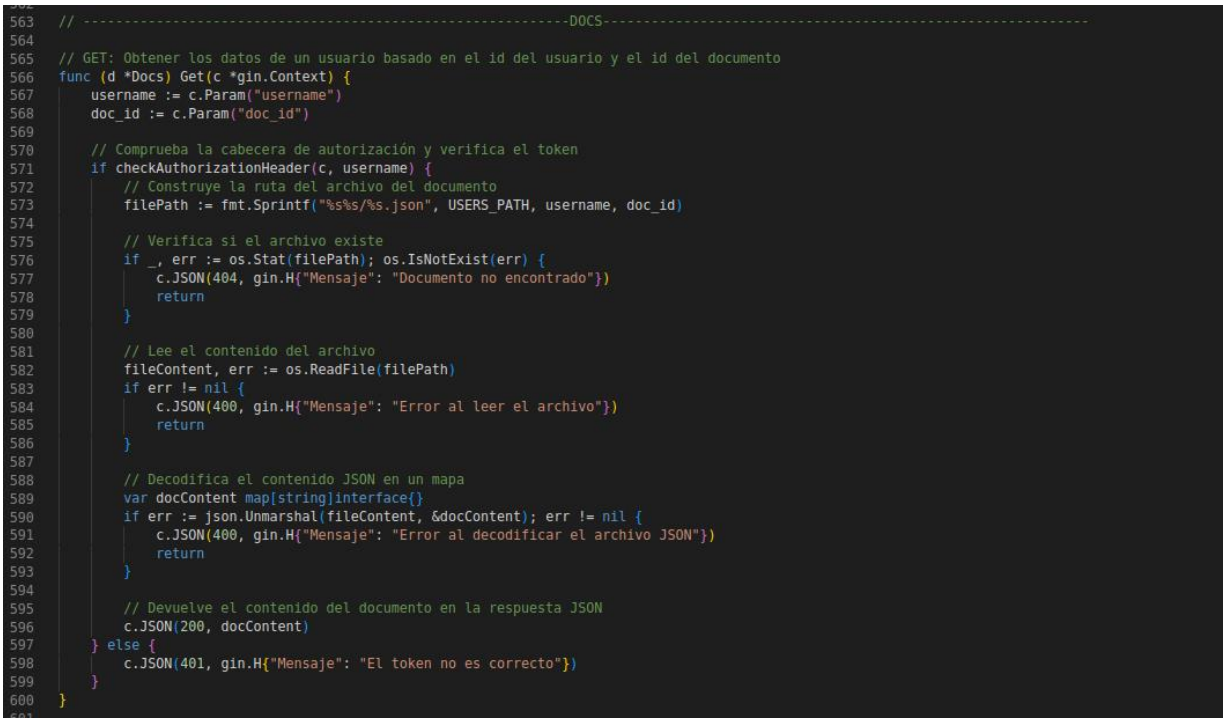
8.4. DELETE():

```go
// DELETE: Eliminar un documento para un usuario
func (u *User) Delete(c *gin.Context) {

    username := c.Param("username")
    docName := c.Param("doc_name")

    // Comprueba la cabecera de autorización y verifica el token
    if checkAuthorizationHeader(c, username) {

        // Construye la ruta del archivo JSON
        jsonFilePath := fmt.Sprintf("%s%s/%s.json", USERS_PATH, username, docName)

        // Comprueba si el archivo existe
        if _, err := os.Stat(jsonFilePath); os.IsNotExist(err) {
            c.JSON(404, gin.H{"Mensaje": "El archivo no existe"})
            return
        }

        // Elimina el archivo JSON
        err := os.Remove(jsonFilePath)
        if err != nil {
            c.JSON(400, gin.H{"Mensaje": "Error al eliminar el archivo"})
            return
        }
        c.JSON(200, gin.H{})

    } else {
        c.JSON(401, gin.H{"Mensaje": "El token no es correcto"})
    }
}
```

Elimina un archivo de documento de un usuario. Verifica que el archivo exista y luego lo elimina. Este método devuelve un código 401 si el token no es válido, 404 si el archivo no existe, 400 si hay un error al eliminar el archivo.

9. DOCS():

9.1. GET():



```go
// ------------------------------------------------------------DOCS------------------------------------------------------------

// GET: Obtener los datos de un usuario basado en el id del usuario y el id del documento
func (d *Docs) Get(c *gin.Context) {
    username := c.Param("username")
    doc_id := c.Param("doc_id")

    // Comprueba la cabecera de autorización y verifica el token
    if checkAuthorizationHeader(c, username) {
        // Construye la ruta del archivo del documento
        filePath := fmt.Sprintf("%s%s/%s.json", USERS_PATH, username, doc_id)

        // Verifica si el archivo existe
        if _, err := os.Stat(filePath); os.IsNotExist(err) {
            c.JSON(404, gin.H{"Mensaje": "Documento no encontrado"})
            return
        }

        // Lee el contenido del archivo
        fileContent, err := os.ReadFile(filePath)
        if err != nil {
            c.JSON(400, gin.H{"Mensaje": "Error al leer el archivo"})
            return
        }

        // Decodifica el contenido JSON en un mapa
        var docContent map[string]interface{}
        if err := json.Unmarshal(fileContent, &docContent); err != nil {
            c.JSON(400, gin.H{"Mensaje": "Error al decodificar el archivo JSON"})
            return
        }

        // Devuelve el contenido del documento en la respuesta JSON
        c.JSON(200, docContent)
    } else {
        c.JSON(401, gin.H{"Mensaje": "El token no es correcto"})
    }
}
```

Aquí se implementa un manejador de la ruta **GET** para obtener los datos de un documento basado en el usuario y el identificador del documento.
Este método permite obtener el contenido de un documento almacenado en un archivo JSON. Está diseñado para asegurar que solo usuarios autenticados puedan acceder a los documentos que les corresponden.

Extrae los parámetros username (nombre del usuario) y doc_id (identificador del documento) desde la URL. Llama a la función **checkAuthorizationHeader** para validar que el token de autorización en la cabecera es correcto y que el usuario tiene permiso de acceso. Después, genera la ruta del archivo del documento en el formato: *USERS_PATH/username/doc_id.json.*

Por otro lado, utiliza *os.Stat* para verificar si el archivo existe en la ruta especificada. Si no existe, devuelve un código **404** con el mensaje: {"Mensaje": "Documento no encontrado"}. Si el archivo existe, lo lee completamente usando os.ReadFile y si ocurre un error durante la lectura, devuelve un código **400** con el mensaje: {"Mensaje": "Error al leer el archivo"}.

En cuanto a su decodificación, intenta decodificar el contenido del archivo como un objeto JSON. Si falla la decodificación, devuelve un código **400** con el mensaje: {"Mensaje": "Error al decodificar el archivo JSON"}. Si todo es exitoso, devuelve un código **200** con el contenido del archivo como un objeto JSON.

Por último, si el token de autorización es inválido, responde con un código **401** y el mensaje:{"Mensaje": "El token no es correcto"}.

10. MAIN():

```go
func main() {
    fmt.Println("Práctica 3 - Seguridad en Redes - Cristina Serrano Trujillo")

    // Instancias
    signup := Signup{}
    login := Login{}
    version := Version{}
    user := User{}
    docs := Docs{}


    // Configurar router
    router := gin.Default()
    router.SetTrustedProxies([]string{"127.0.0.1"})


    // Definir rutas
    router.GET("/version", version.Get)
    router.POST("/signup", signup.SignUp)
    router.POST("/login", login.Login)

    router.GET("/users/:username", user.Get)
    router.GET("/users/:username/docs/:doc_id", docs.Get)
    router.POST("/users/:username/docs/:doc_name", user.Post)
    router.PUT("/users/:username/docs/:doc_name", user.Put)
    router.DELETE("/users/:username/docs/:doc_name", user.Delete)

    // Iniciar servidor
    router.RunTLS("myserver.local:5000", "certs/cert.pem", "certs/key.pem")
}
```

El método main permite gestionar usuarios y documentos. Inicialmente, se imprime un mensaje descriptivo para confirmar la ejecución del programa y se crean instancias de las estructuras encargadas de manejar las diferentes funcionalidades, como registro de usuarios, inicio de sesión, gestión de usuarios y documentos, y consulta de la versión de la API. A continuación, se configura el router de la aplicación, estableciendo como confiables solo las solicitudes provenientes de la dirección 127.0.0.1, lo que mejora la seguridad al limitar la aceptación de peticiones externas.

Posteriormente, se definen las rutas principales de la API, organizadas en rutas generales, como la consulta de la versión de la API y las operaciones de registro e inicio de sesión, y rutas específicas para gestionar usuarios y documentos, que incluyen operaciones para obtener, crear, actualizar y eliminar documentos asociados a un usuario. Finalmente, se inicia el servidor en modo seguro utilizando HTTPS con certificados TLS, escuchando en el dominio myserver.local y el puerto 5000. Este diseño garantiza la seguridad en las comunicaciones y una organización clara y funcional de las rutas y controladores.

11. EJECUCIÓN

11.1. REQUISITOS PREVIOS

Para poder hacer uso del proyecto es necesario instalar algunas dependencias de GO, para facilitar esta instalación he creado un metodo en Makefile que ejecutando make dependecias se instalarán automaticamente.

11.2. EJECUCIÓN DEL SERVIDOR

Para agilizar el arranque del servidor se puede hacer uso de:

make run

SI quisiésemos hacerlo manualmente sería con los siguientes comandos:

go build
main.gogo run
main.go

Una vez arrancado el servidor hay dos formas de probarlo: Haciendo uso de los tests generados para agilizar las pruebas o haciendo uso de alguna herramienta de prueba de APIs como por ejemplo *postman* y realizar las pruebas con ella de forma manual.

11.3. EJECUCIÓN TESTS

Para facilitar la comprobación de la robustez e integridad del sistema, he implementaddo al- gunos archivos .sh que contienen pruebas automatizadas que cubren las funcionalidades del programa.

Se han creado varios tipos de prueba ubicados en la carpeta *tests/*, para ello primero hay quedarle permisos de ejecución a los que queramos ejecutar, con el siguiente comando:

chmod +x script test.sh

Una vez otorgados los permisos tu ejecución es muy simple y entraré en detalle para cada uno de ellos. Lo primero para poder realizar el lanzamiento de las pruebas es necesario tener instalado la herramienta *curl* y *jq*, para la instalación hay que hacer uso del siguiente comando:

sudo apt-get install curl
sudo apt-get install jq

Por último para poder realizar los tests, es necesario tener lanzado el servidor en terminal, mencionado anteriormente. Tambiém haciendo uso de make test podemos realizar la prueba del sistema automaticamente.

11.3.1. Test signup()

Realiza el registro del usuario

./signup.sh <username><password>

SEGURIDAD > LABORATORIO > P3 > tests > $ signup.sh

```bash
#!/bin/bash

echo "=== Registrando nuevo usuario ==="

read -p "Nombre de usuario: " username
read -s -p "Contraseña: " password
echo
read -s -p "Confirme la contraseña: " password_confirm
echo
read -p "ID de usuario: " user_id

if [[ "$password" != "$password_confirm" ]]; then
  echo "Error: Las contraseñas no coinciden."
  exit 1
fi

response=$(curl -s -X POST "https://myserver.local:5000/signup" \
  -H "Content-Type: application/json" \
  -d "{\"Usuario\": \"$username\", \"Contraseña\": \"$password\", \"ID\": \"$user_id\"}" -k)

token=$(echo "$response" | jq -r '.access_token // empty')

if [[ -n "$token" ]]; then
  echo "Usuario registrado con éxito. {token: $token}"

  # Crear la carpeta del usuario si no existe
  user_dir="users/$username"
  mkdir -p "$user_dir"

  # Guardar el username, password, token y ID en un archivo JSON en la carpeta del usuario
  echo "{\"username\":\"$username\", \"password\":\"$password\", \"token\":\"$token\", \"ID\":\"$user_id\"}" > "$user_dir/$user_id.json"
  echo "Datos almacenados en $user_dir/$user_id.json."

  echo "Nota: Este token expira en 5 minutos. Deberá volver a iniciar sesión para obtener un nuevo token."
else
  echo "Error al registrar usuario. Respuesta: $response"
  exit 1
fi
```

11.3.2. Test login()

Para iniciar sesión con un usuario existente.

./login.sh <username><password>

SEGURIDAD > LABORATORIO > P3 > tests > $ login.sh

```bash
#!/bin/bash

echo "=== Iniciando sesión ==="

read -p "Nombre de usuario: " username
read -s -p "Contraseña: " password
echo

response=$(curl -s -X POST "https://myserver.local:5000/login" \
  -H "Content-Type: application/json" \
  -d "{\"Usuario\": \"$username\", \"Contraseña\": \"$password\"}" -k)

# Intentar extraer el token de 'access_token' o 'token_acceso'
token=$(echo "$response" | jq -r '.access_token // .token_acceso // empty')

if [[ -n "$token" ]]; then
  echo "Inicio de sesión exitoso. {token: $token}"

  # Crear la carpeta del usuario si no existe
  user_dir="users/$username"
  mkdir -p "$user_dir"

  # Solicitar el ID del usuario
  read -p "Ingrese el ID del usuario: " user_id

  # Guardar el username, password, token y ID en un archivo JSON en la carpeta del usuario
  user_file="$user_dir/$user_id.json"
  echo "{\"username\":\"$username\", \"password\":\"$password\", \"token\":\"$token\", \"ID\":\"$user_id\"}" > "$user_file"
  echo "Datos guardados en $user_file."

  # Guardar el nombre de usuario en un archivo
  echo "$username" > "current_user.txt"

  echo "Nota: Este token expira en 5 minutos. Deberá volver a iniciar sesión para obtener un nuevo token."
else
  echo "Error al iniciar sesión. Respuesta: $response"
  exit 1
fi
```

11.3.3. Test doc_id

Con este test probaremos las funciones que puede hacer el usuario con sus documentos, *GET,PUT y DELETE.*

./user.sh <username><password≥<doc id>

11.3.4. Test all_docs

Muestra todos los documentos JSON del usuario

./test/all docs.sh <username><password>

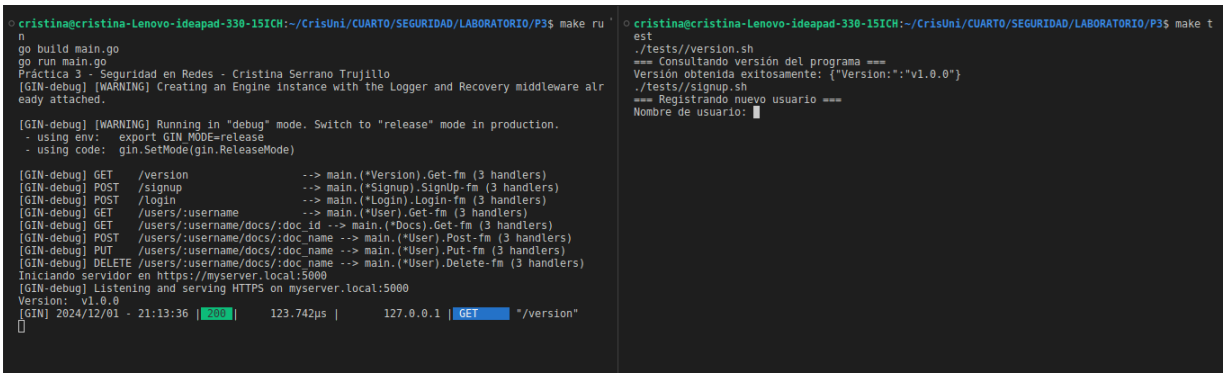
SEGURIDAD > LABORATORIO > P3 > tests > $ all_docs.sh

```bash
#!/bin/bash

# GET: Obtener todos los documentos de un usuario
# Argumento: username

if [ $# -ne 1 ]; then
    echo "Uso: $0 <username>"
    exit 1
fi

username=$1

# Leer el token desde el archivo JSON del usuario
user_dir="users/$username"
id_file="$user_dir/$username.json"

if [[ ! -f "$id_file" ]]; then
    echo "Error: No se encontró el archivo de autenticación para el usuario $username."
    exit 1
fi

token=$(jq -r '.token' "$id_file")

# Verifica que el token no esté vacío
if [ -z "$token" ]; then
    echo "Se requiere un token de autenticación. Usa /login o /signup para obtenerlo."
    exit 1
fi

response=$(curl -s -o /dev/null -w "%{http_code}" -X GET "https://myserver.local:5000/users/$username/_all_docs" \
    -k \
    -H "Authorization: token $token")

if [ "$response" -eq 200 ]; then
    echo "Operación realizada con éxito."
else
    echo "Error en la operación. Código de respuesta: $response"
fi
```

12. PRUEBAS

En primer lugar ejecutamos **make run** en una consola que será el servidor y **make test** en la otra que será el cliente.

Lo primero que nos mostrará en pantalla será la versión de la API, ya que en el Makefile está puesto en ese orden para que se ejecuten todos los scripts en concordancia.



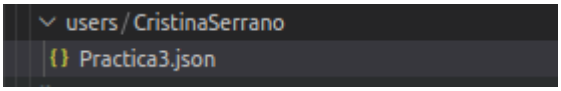
Después se ejecuta el siguiente script, que es el signup.sh. Aquí nos piden un nombre de usuario con el que registrarnos y una contraseña que no estará visible para mayor seguridad. Posteriormente nos pedirá una confirmación de dicha contraseña acompañada del ID.

Este ID será el nombre con el que se generará el archivo.json, el cual se creará bajo una carpeta llamada igual que el nombre de usuario que pongamos por teclado. Una vez hecho esto, el servidor nos manda el token de acceso.

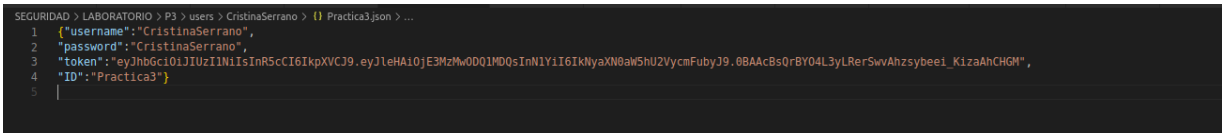
```
cristina@cristina-Lenovo-ideapad-330-15ICH:~/CrisUni/CUARTO/SEGURIDAD/LABORATORIO/P3$ make ru
n
go build main.go
go run main.go
Práctica 3 - Seguridad en Redes - Cristina Serrano Trujillo
[GIN-debug] [WARNING] Creating an Engine instance with the Logger and Recovery middleware alr
eady attached.

[GIN-debug] [WARNING] Running in "debug" mode. Switch to "release" mode in production.
 - using env:   export GIN_MODE=release
 - using code:  gin.SetMode(gin.ReleaseMode)

[GIN-debug] GET    /version                  --> main.(*Version).Get-fm (3 handlers)
[GIN-debug] POST   /signup                   --> main.(*Signup).SignUp-fm (3 handlers)
[GIN-debug] POST   /login                    --> main.(*Login).Login-fm (3 handlers)
[GIN-debug] GET    /users/:username          --> main.(*User).Get-fm (3 handlers)
[GIN-debug] GET    /users/:username/docs/:doc_id --> main.(*Docs).Get-fm (3 handlers)
[GIN-debug] POST   /users/:username/docs/:doc_name --> main.(*User).Post-fm (3 handlers)
[GIN-debug] PUT    /users/:username/docs/:doc_name --> main.(*User).Put-fm (3 handlers)
[GIN-debug] DELETE /users/:username/docs/:doc_name --> main.(*User).Delete-fm (3 handlers)
Iniciando servidor en https://myserver.local:5000
[GIN-debug] Listening and serving HTTPS on myserver.local:5000
Version:  v1.0.0
[GIN] 2024/12/01 - 21:13:36 | 200 |     123.742µs |       127.0.0.1 | GET      "/version"
¡Usuario registrado correctamente!
¡Espacio de usuario creado correctamente!
[GIN] 2024/12/01 - 21:16:44 | 200 |     645.014µs |       127.0.0.1 | POST     "/signup"
```

```
cristina@cristina-Lenovo-ideapad-330-15ICH:~/CrisUni/CUARTO/SEGURIDAD/LABORATORIO/P3$ make t
est
./tests//version.sh
=== Consultando versión del programa ===
Versión obtenida exitosamente: {"Version:":"v1.0.0"}
./tests//signup.sh
=== Registrando nuevo usuario ===
Nombre de usuario: CristinaSerrano
Contraseña:
Confirme la contraseña:
ID de usuario: Practica3
Usuario registrado con éxito. {token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE3MzMwO
DQ1MDQsInN1YiI6IkNyaXN0aW5hU2VycmFubyJ9.0BAAcBsQrBYO4L3yLRerSwvAhzsybeei_KizaAhCHGM}
Datos almacenados en users/CristinaSerrano/Practica3.json.
Nota: Este token expira en 5 minutos. Deberá volver a iniciar sesión para obtener un nuevo t
oken.
./tests//login.sh
=== Iniciando sesión ===
Nombre de usuario:
```

Adjunto foto del .json generado:



La carpeta raíz donde se crearán todos los usuarios se creará automáticamente en el caso de no estar creada ya, al ejecutar el Makefile. El contenido del .json es el siguiente:

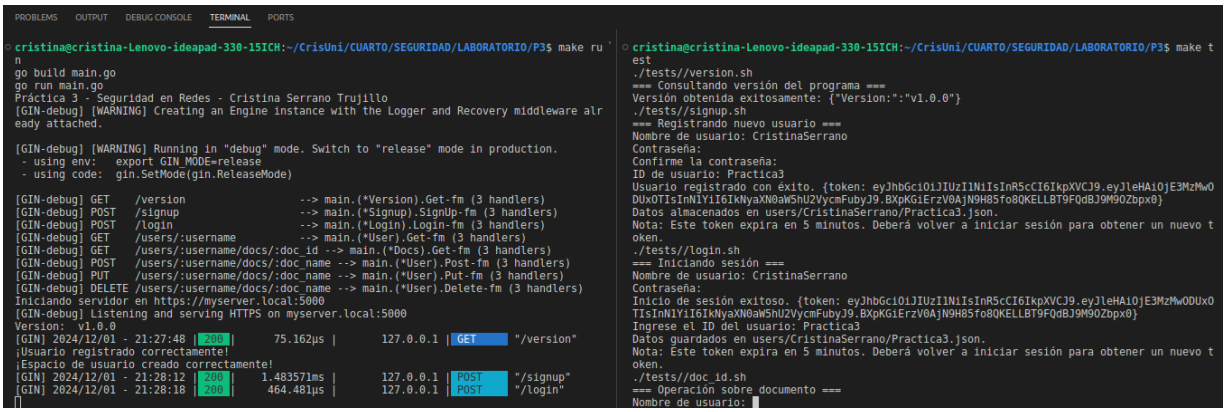


Siguiendo con la ejecución, el siguiente script es el login.sh para iniciar sesion con el usuario creado anteriormente.

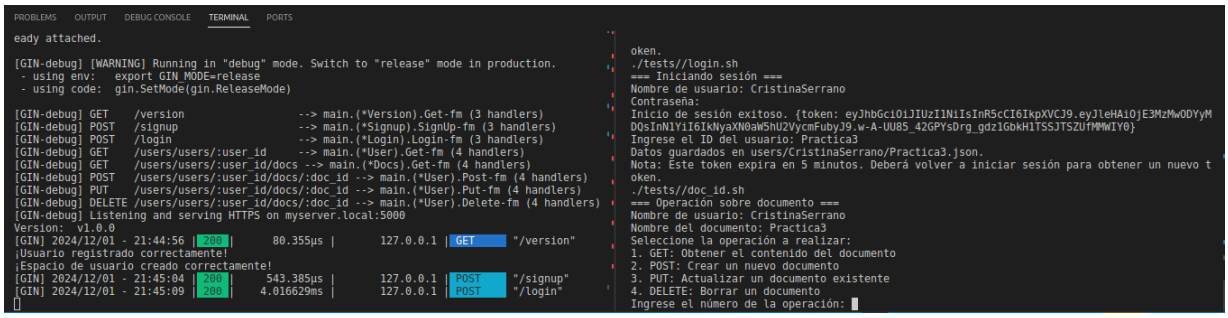
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

cristina@cristina-Lenovo-ideapad-330-15ICH:~/CrisUni/CUARTO/SEGURIDAD/LABORATORIO/P3$ make ru
n
go build main.go
go run main.go
Práctica 3 - Seguridad en Redes - Cristina Serrano Trujillo
[GIN-debug] [WARNING] Creating an Engine instance with the Logger and Recovery middleware alr
eady attached.

[GIN-debug] [WARNING] Running in "debug" mode. Switch to "release" mode in production.
 - using env:   export GIN_MODE=release
 - using code:  gin.SetMode(gin.ReleaseMode)

[GIN-debug] GET    /version                  --> main.(*Version).Get-fm (3 handlers)
[GIN-debug] POST   /signup                   --> main.(*Signup).SignUp-fm (3 handlers)
[GIN-debug] POST   /login                    --> main.(*Login).Login-fm (3 handlers)
[GIN-debug] GET    /users/:username          --> main.(*User).Get-fm (3 handlers)
[GIN-debug] GET    /users/:username/docs/:doc_id --> main.(*Docs).Get-fm (3 handlers)
[GIN-debug] POST   /users/:username/docs/:doc_name --> main.(*User).Post-fm (3 handlers)
[GIN-debug] PUT    /users/:username/docs/:doc_name --> main.(*User).Put-fm (3 handlers)
[GIN-debug] DELETE /users/:username/docs/:doc_name --> main.(*User).Delete-fm (3 handlers)
Iniciando servidor en https://myserver.local:5000
[GIN-debug] Listening and serving HTTPS on myserver.local:5000
Version:  v1.0.0
[GIN] 2024/12/01 - 21:27:48 | 200 |      75.162µs |       127.0.0.1 | GET      "/version"
¡Usuario registrado correctamente!
¡Espacio de usuario creado correctamente!
[GIN] 2024/12/01 - 21:28:12 | 200 |    1.483571ms |       127.0.0.1 | POST     "/signup"
[GIN] 2024/12/01 - 21:28:18 | 200 |     464.481µs |       127.0.0.1 | POST     "/login"
```

```
cristina@cristina-Lenovo-ideapad-330-15ICH:~/CrisUni/CUARTO/SEGURIDAD/LABORATORIO/P3$ make t
est
./tests//version.sh
=== Consultando versión del programa ===
Versión obtenida exitosamente: {"Version:":"v1.0.0"}
./tests//signup.sh
=== Registrando nuevo usuario ===
Nombre de usuario: CristinaSerrano
Contraseña:
Confirme la contraseña:
ID de usuario: Practica3
Usuario registrado con éxito. {token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE3MzMwO
DUxOTIsInN1YiI6IkNyaXN0aW5hU2VycmFubyJ9.BXpKGiErzV0AjN9H85fo8QKELLBT9FQdBJ9M9OZbpx0}
Datos almacenados en users/CristinaSerrano/Practica3.json.
Nota: Este token expira en 5 minutos. Deberá volver a iniciar sesión para obtener un nuevo t
oken.
./tests//login.sh
=== Iniciando sesión ===
Nombre de usuario: CristinaSerrano
Contraseña:
Inicio de sesión exitoso. {token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE3MzMwODUxO
TIsInN1YiI6IkNyaXN0aW5hU2VycmFubyJ9.BXpKGiErzV0AjN9H85fo8QKELLBT9FQdBJ9M9OZbpx0}
Ingrese el ID del usuario: Practica3
Datos guardados en users/CristinaSerrano/Practica3.json.
Nota: Este token expira en 5 minutos. Deberá volver a iniciar sesión para obtener un nuevo t
oken.
./tests//doc_id.sh
=== Operación sobre documento ===
Nombre de usuario:
```

Como podemos observar, nos vuelve a pedir el usuario, la contraseña y el ID, devolviéndonos además el mismo token que nos dio al registrarnos.



Lo siguiente que se ejecuta es el doc_id.sh. Los parámetros que te piden son el nombre de usuario y el nombre del documento. El nombre del documento es el ID que metimos más arriba, ya que este será el nombre con el que se cree el archivo.json. Después nos da un menú a elegir entre las posibles opciones para operar con los documentos generados: GET, POST, PUT y DELETE.



Por último ejecutaremos **make clean** para eliminar la carpeta /users y el archivo .shadow.