

# Sistema de Gestión de Políticas de Contraseñas Seguras

Proyecto Final – Programación en Red

## 1. Introducción

En este proyecto se implementa un Sistema de Gestión de Políticas de Contraseñas Seguras orientado a entornos empresariales. El objetivo principal es centralizar y automatizar la aplicación de políticas de seguridad sobre contraseñas de usuarios, evitando contraseñas débiles, forzando la caducidad periódica y proporcionando mecanismos de notificación al usuario final.

La solución se diseñó siguiendo el enfoque propuesto en el enunciado del proyecto final de la asignatura Programación en Red, basado en una arquitectura de tres servidores lógicos (S1, S2, S3), clientes web heterogéneos y la integración con un servicio de mensajería externo para notificar la proximidad de caducidad de contraseñas.

## 2. Objetivos del sistema

### 2.1 Objetivo general

Desarrollar un sistema distribuido que gestione políticas de contraseñas robustas, valide nuevas contraseñas de acuerdo con dichas políticas, almacene de forma segura los hashes y notifique al usuario cuando su contraseña esté próxima a caducar.

### 2.2 Objetivos específicos

- Implementar un Servidor HTTP/REST (S1) capaz de autenticar administradores y usuarios mediante login y JWT, administrar el CRUD de políticas de contraseñas y el CRUD de usuarios.
- Implementar un Servidor RPC (S2) que reciba peticiones de validación de contraseñas desde S1 por gRPC, aplique la política activa, valide contraseñas débiles y utilice bcrypt para el hashing.
- Implementar un Servidor de Recursos/REST (S3) que gestione el acceso a la base de datos de usuarios y almacene el hash de la contraseña y la fecha de último cambio.
- Desarrollar dos clientes web heterogéneos: un panel de administración para políticas y usuarios y un portal de usuario para cambio de contraseña y consulta de estado.
- Integrar un servicio de mensajería externo (Telegram Bot API) para enviar notificaciones de caducidad de contraseña.
- Demostrar alta disponibilidad lógica: S2 conserva una copia en caché de la política activa para seguir validando la complejidad incluso si S3 falla, aunque no pueda persistir cambios.

### 3. Marco teórico (resumen)

#### 3.1 Políticas de contraseñas

Una política de contraseñas define reglas mínimas de seguridad, por ejemplo: longitud mínima, inclusión de mayúsculas, minúsculas, dígitos y símbolos, prohibición de contraseñas comunes y caducidad periódica. Estas políticas reducen la probabilidad de ataques de fuerza bruta y reutilización de contraseñas débiles.

#### 3.2 Hashing seguro (bcrypt)

En lugar de almacenar contraseñas en texto plano, se almacena un hash obtenido con un algoritmo lento y con sal (salt), como bcrypt. Esto dificulta los ataques de diccionario y hace que, incluso si se filtra la base de datos, resulte costoso recuperar las contraseñas originales.

#### 3.3 JWT (JSON Web Token)

Un JWT es un token firmado que contiene información sobre el usuario (claims). Se utiliza para mantener sesiones sin guardar estado en el servidor y para autorizar el acceso a endpoints protegidos.

#### 3.4 TLS / HTTPS

El cifrado TLS garantiza la confidencialidad e integridad de los datos en tránsito entre cliente y servidor. El Servidor 1 (S1) expone una interfaz HTTPS, usando un certificado (autofirmado en este entorno de laboratorio).

### 4. Descripción general de la arquitectura

La solución se compone de:

- S1 – Servidor HTTP/REST (Node.js + Express + HTTPS): expone la API principal, gestiona login, JWT, políticas y usuarios, y sirve los clientes web (admin y usuario).
- S2 – Servidor RPC (Node.js + gRPC): motor de validación de contraseñas. Recibe solicitudes de S1 y aplica la política activa, revisa contra la lista de contraseñas débiles y usa bcrypt.
- S3 – Servidor de Recursos/REST (Node.js + Express + MongoDB): API REST para usuarios, políticas e historial.
- Base de datos MongoDB: almacena usuarios, políticas e historial de contraseñas.
- Servicio externo: Telegram Bot API para notificaciones de caducidad de contraseña.
- Clientes Web: Cliente Admin Web (panel de administración) y Portal de Usuario.

### 5. Detalle de componentes

#### 5.1 Servidor 1 (S1) – HTTP/REST + HTTPS

Tecnologías: Node.js, Express, HTTPS, JWT.

Funcionalidades principales:

- Login de administrador y usuarios, generación de JWT.
- Gestión de políticas (CRUD): crear, obtener, actualizar y eliminar políticas de contraseñas.
- Gestión de usuarios (CRUD): alta, baja, modificación de usuarios, incluyendo

telegramChatId.

- Gestión de contraseña del usuario: consulta de estado y cambio de contraseña.
- Servir el Cliente Admin Web en /admin y el Portal de Usuario en /user.

## 5.2 Servidor 2 (S2) – RPC / gRPC

Tecnologías: Node.js, gRPC, bcrypt.

Responsabilidades:

- Exponer un servicio gRPC para validar contraseñas.
- Aplicar la lógica de política: longitud mínima, complejidad y diccionario de contraseñas débiles.
- Hashear la contraseña con bcrypt si es válida.
- Mantener en memoria una caché de la política activa para tolerancia a fallos.

## 5.3 Servidor 3 (S3) – Recursos/REST + MongoDB

Tecnologías: Node.js, Express, MongoDB (Mongoose).

Modelos principales:

- User: email, rol, passwordHash, passwordLastChangedAt, passwordExpiresAt, telegramChatId, notificationSent, lastNotificationDate.
- PasswordPolicy: minLength, requireUppercase, requireLowercase, requireDigits, requireSymbols, maxPasswordAgeDays, isActive, description.
- PasswordHistory: historial de hashes anteriores.

Endpoints típicos: GET/POST/PUT/DELETE de usuarios y políticas.

# 6. Clientes web heterogéneos

## 6.1 Cliente Admin Web

Aplicación HTML/CSS/JS que permite:

- Login de administrador.
- Gestión de usuarios (alta, edición, eliminación).
- Gestión de políticas de contraseña.

Todas las peticiones se realizan contra S1 usando JWT.

## 6.2 Portal de Usuario

Interfaz web separada para el usuario final que permite:

- Login de usuario.
- Consultar el estado de la contraseña (días restantes, fecha de último cambio).
- Cambiar la contraseña cumpliendo la política activa.

# 7. Integración con servicio tercero (Telegram)

Como servicio de mensajería externo se utiliza Telegram Bot API. El sistema funciona de la siguiente forma:

1. El usuario tiene configurado su telegramChatId en la base de datos.

2. S1 calcula los días restantes hasta la caducidad de la contraseña.
3. Si los días restantes son menores o iguales a 7 y aún no se ha enviado notificación, S1 invoca el servicio interno de Telegram.
4. El sistema realiza una petición HTTPS a `api.telegram.org/bot<TOKEN>/sendMessage` con el `chat_id` y el texto.
5. Si la petición es exitosa se actualizan los campos `notificationSent` y `lastNotificationDate`.

## 8. Implementación de seguridad

### 8.1 Autenticación y autorización con JWT

Los endpoints protegidos requieren un JWT válido enviado en el header `Authorization: Bearer <token>`. Los middlewares verifican la firma del token, extraen el rol del usuario y solo permiten el acceso si el token es válido y no ha expirado.

### 8.2 HTTPS / TLS en S1

El servidor S1 expone un puerto HTTPS (por ejemplo 3001) utilizando certificados y llaves privadas. En el entorno de laboratorio se emplean certificados autofirmados, pero el canal de comunicación entre cliente y servidor está cifrado.

### 8.3 Hash lento y salado (bcrypt)

Las contraseñas se almacenan mediante `bcrypt` con un costo adecuado de rondas. Nunca se guarda la contraseña en texto plano; únicamente se almacena el hash resultante en la base de datos.

## 9. Alta disponibilidad y tolerancia a fallos

El sistema implementa un mecanismo de tolerancia a fallos a nivel lógico:

- S2 mantiene en memoria la política activa recibida desde S1.
- Si S3 falla, S2 aún puede validar la complejidad de las contraseñas basándose en la política en caché.
- Sin embargo, mientras S3 esté caído, no es posible persistir el cambio de contraseña; S1 informa al usuario del error de almacenamiento.
- Cada servidor expone un endpoint `/health` que permite comprobar su estado.

## 10. Despliegue con Docker

El proyecto se despliega mediante Docker Compose, con contenedores separados para S1, S2, S3 y MongoDB. Pasos generales para el despliegue:

1. Configurar el archivo `.env` con `MONGO_URI`, `JWT_SECRET`, `TELEGRAM_BOT_TOKEN`, etc.
2. Ejecutar el script de despliegue `./scripts/run-all.sh`.
3. Verificar los contenedores con `docker compose ps`.
4. Acceder al panel admin en `https://IP_DEBIAN:3001/admin` y al portal usuario en `https://IP_DEBIAN:3001/user`.

## 11. Pruebas realizadas

Entre las pruebas realizadas se incluyen:

- Prueba de login de administrador y acceso al panel admin.

- Prueba de CRUD de políticas de contraseña.
- Prueba de CRUD de usuarios.
- Prueba de cambio de contraseña (contraseña débil vs fuerte).
- Prueba de notificación por Telegram al acercarse la fecha de caducidad.
- Prueba de tolerancia a fallos con S3 caído: validación de política activa pero error al guardar cambios.

## 12. Conclusiones

El sistema desarrollado cumple con los requisitos planteados para un Sistema de Gestión de Políticas de Contraseñas Seguras, al implementar tres servidores lógicos con responsabilidades bien definidas, un motor de validación con políticas configurables y hashing seguro, clientes web heterogéneos, integración con un servicio externo de mensajería (Telegram) y mecanismos de alta disponibilidad lógica.

El uso de JWT y HTTPS proporciona seguridad adicional para la autenticación y la protección de datos en tránsito, mientras que Docker Compose facilita la orquestación y despliegue del entorno completo.