Memoria del Proyecto
Servicios en Red y Programación Segura
RA4 y RA5

Aitor Jury Rodríguez.
2º DAM.
Programación de Servicios y Procesos.
RA 4-5.

Índice

Índice **2**
Introducción **3**
Objetivo del Proyecto **4**
Herramientas y Librerías **5**
Especificación de clases **6**
HttpServerMain 6
AuthHandler 6
HttpClientMain 6
DatabaseManager 6
HashUtil 6
FtpClientService 6
Diseño y Organización **7**
Organización física del proyecto 7
Protocolo de Comunicación (Capa de Aplicación) 7
Mecanismos de Concurrencia **8**
Medidas de Seguridad Implementadas **9**
Mejoras Realizadas **10**
Manual de Uso Básico **11**
Pruebas Realizadas **12**
Captura 1: Servidor iniciado y esperando conexiones 12
Captura 2: Registro de un nuevo usuario y comprobación en MySQL 12
Captura 3: Intento de login con contraseña errónea 12
Captura 4: Descarga con éxito del archivo mediante la opción FTP 13
Captura 5: Uso de la opción GET para ver el Hash y la Sal del usuario desde el cliente 13
Conclusión **14**

Introducción

En este proyecto se ha desarrollado un ecosistema de servicios en red basado en la arquitectura Cliente-Servidor utilizando el protocolo HTTP para la gestión de usuarios y el protocolo FTP para la transferencia de ficheros.

La aplicación permite el registro, login, consulta, modificación y borrado de usuarios (CRUD completo) de forma segura, aplicando algoritmos criptográficos modernos para la protección de la información sensible en tránsito y en reposo.

Objetivo del Proyecto

El objetivo principal es cumplir con los Resultados de Aprendizaje (RA) del módulo:

- RA4: Desarrollar servicios en red eficientes, disponibles y concurrentes.
- RA5: Proteger los datos mediante técnicas de programación segura y criptografía.

Herramientas y Librerías

- Lenguaje: Java SE 17/21.
- Entorno de Desarrollo: Visual Studio Code.
- Gestor de Base de Datos: MySQL 8.0.
- Librerías Externas:
    - mysql-connector-j: Conector JDBC para la persistencia en MySQL.
    - commons-net-3.11.1: Implementación del protocolo FTP.
    - commons-io-2.18.0: Soporte para el manejo de flujos de datos (Streams).
- Librerías Nativas:
    - com.sun.net.httpserver: Para el servidor.
    - java.net.http.HttpClient: Para el cliente.

Especificación de clases

HttpServerMain

Clase principal del servidor. Configura el puerto 8080, el contexto /auth y el pool de hilos para la concurrencia.

AuthHandler

Controlador que gestiona la semántica HTTP. Procesa los métodos POST, GET, PUT y DELETE, interactuando con la lógica de negocio.

HttpClientMain

Interfaz de usuario por consola. Gestiona el ciclo de vida de la sesión y las peticiones al servidor.

DatabaseManager

Capa de persistencia (DAO). Gestiona la conexión JDBC y las consultas SQL parametrizadas.

HashUtil

Utilidad criptográfica. Genera sales aleatorias y hashes SHA-256.

FtpClientService

Cliente especializado en la descarga de recursos mediante el protocolo FTP.

Diseño y Organización

Organización física del proyecto

```
HttpSecureProject/
├── lib/ (Librerías .jar externas)
├── src/
│   ├── com/auth/db/DatabaseManager.java
│   ├── com/auth/util/HashUtil.java
│   ├── com/server/HttpServerMain.java
│   ├── com/server/AuthHandler.java
│   ├── com/client/HttpClientMain.java
│   ├── com/client/FtpClientService.java
└── resources/schema.sql
```

Protocolo de Comunicación (Capa de Aplicación)

Se utiliza HTTP/1.1 sobre TCP. El intercambio de datos se realiza mediante el formato application/x-www-form-urlencoded:

- Registro/Login (POST): Cuerpo user=nombre&pass=clave. El servidor decide si crea o valida.
- Consulta (GET): Parámetro en URL ?user=nombre. Devuelve auditoría del Hash.
- Modificación (PUT): Cuerpo oldUser=nombre&newUser=nuevoNombre.
- Borrado (DELETE): Cuerpo user=nombre.

Mecanismos de Concurrencia

Para garantizar la disponibilidad (RA4), el servidor implementa un ThreadPoolExecutor mediante Executors.newCachedThreadPool(). Esto permite que el servidor no sea bloqueante: cada nueva conexión de un cliente es delegada a un hilo del pool, permitiendo que múltiples usuarios realicen operaciones CRUD o descargas simultáneas sin interferir entre ellos.

Medidas de Seguridad Implementadas

1. No almacenamiento en texto plano: Las contraseñas se procesan mediante el algoritmo SHA-256.
2. Salting Criptográfico: Se genera un Salt único de 16 bytes por usuario con SecureRandom, mitigando ataques de diccionario y Rainbow Tables.
3. Prevención de SQL Injection: Uso estricto de PreparedStatement en todas las consultas a la base de datos MySQL.
4. Validación de entradas: El cliente y servidor filtran campos vacíos y nulos para evitar estados inconsistentes.

Mejoras Realizadas

1. CRUD Completo: Implementación de los 4 métodos HTTP principales (POST, GET, PUT, DELETE) superando el mínimo obligatorio.
2. Persistencia en MySQL: Uso de una base de datos relacional real en lugar de ficheros de texto.
3. Sistema de "SALIR" Global: Implementación de una palabra clave interceptora que permite cerrar la aplicación en cualquier momento de la entrada de datos.
4. Cliente FTP Robusto: Mejora opcional para descargar archivos desde servidores internacionales (Mirror de la Univ. de Berlín/Trellian).
5. Validación de Datos: Control de errores para evitar el envío de campos vacíos.

Manual de Uso Básico

1. Preparación: Ejecutar el script schema.sql en MySQL para crear la base de datos secure_auth.
2. Lanzamiento: Ejecutar primero HttpServerMain.java para iniciar la escucha en el puerto 8080.
3. Interacción: Ejecutar HttpClientMain.java.
4. Introducir usuario y contraseña para entrar.
5. Navegar por el menú numérico (1-6).
6. Escribir "SALIR" en cualquier momento para finalizar.

Pruebas Realizadas

Captura 1: Servidor iniciado y esperando conexiones

```
aitor@2DAM-008:/media/aitor/AITOR JURY/Programacion_de_Se
/bin/java @/tmp/cp_4g1c2qirkg78tngxxnnuiqtgb.argfile com.
================================================
[SISTEMA] Servidor de Autenticación Segura iniciado
[ESTADO] Escuchando en el puerto: 8080
[URL] Punto de acceso: http://localhost:8080/auth
================================================
```

Captura 2: Registro de un nuevo usuario y comprobación en MySQL

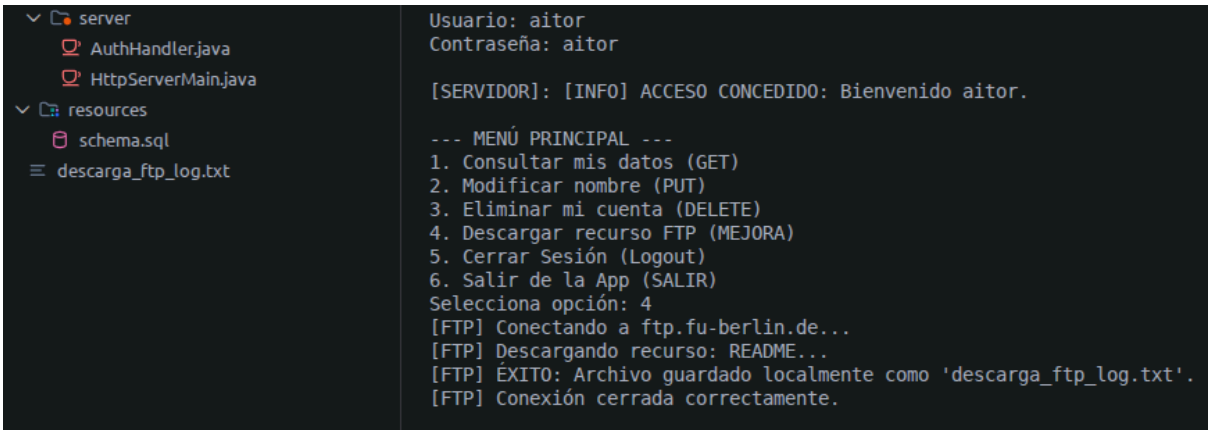
```
=== SISTEMA DE GESTIÓN SEGURA ===
(Escribe 'SALIR' en cualquier momento para finalizar)

Usuario: aitor
Contraseña: aitor

[SERVIDOR]: [INFO] NUEVA CUENTA: Usuario 'aitor' registrado con éxito.

--- MENÚ PRINCIPAL ---
1. Consultar mis datos (GET)
2. Modificar nombre (PUT)
3. Eliminar mi cuenta (DELETE)
4. Descargar recurso FTP (MEJORA)
5. Cerrar Sesión (Logout)
6. Salir de la App (SALIR)
Selecciona opción:
```

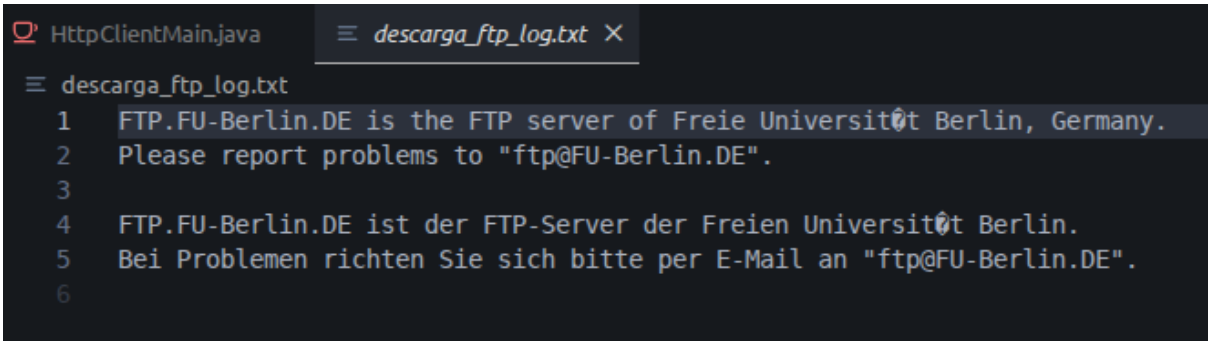
```
mysql> SELECT * FROM users;
+----+----------+--------------------------------------------+--------------------------+
| id | username | password_hash                              | salt                     |
+----+----------+--------------------------------------------+--------------------------+
|  1 | juan     | /1YhoOgpLAcxbze5qSSkduquiD+t7GghSGsOmjldTnQ= | uU55G6Rd07lZyEHF/qkfXw== |
|  2 | aitor    | 3tPYBYtlIgzCNUXA0/6bymqIzqZ3b/R3TO3cM9VD3Zk= | HMOCGwu3yivg3xnGZIonPw== |
+----+----------+--------------------------------------------+--------------------------+
2 rows in set (0,00 sec)
```

Captura 3: Intento de login con contraseña errónea

```
=== SISTEMA DE GESTIÓN SEGURA ===
(Escribe 'SALIR' en cualquier momento para finalizar)

Usuario: aitor
Contraseña: malacontra

[SERVIDOR]: [ERROR] Credenciales incorrectas.

Usuario:
```

Captura 4: Descarga con éxito del archivo mediante la opción FTP





Captura 5: Uso de la opción GET para ver el Hash y la Sal del usuario desde el cliente

Conclusión

El desarrollo de este proyecto ha consolidado los conocimientos sobre arquitecturas distribuidas. Se ha logrado un sistema que no solo comunica procesos en red de forma eficiente, sino que prioriza la integridad y privacidad de la información del usuario, simulando los estándares de seguridad de una aplicación profesional moderna.