Programación de servicios y procesos

S5
UT3

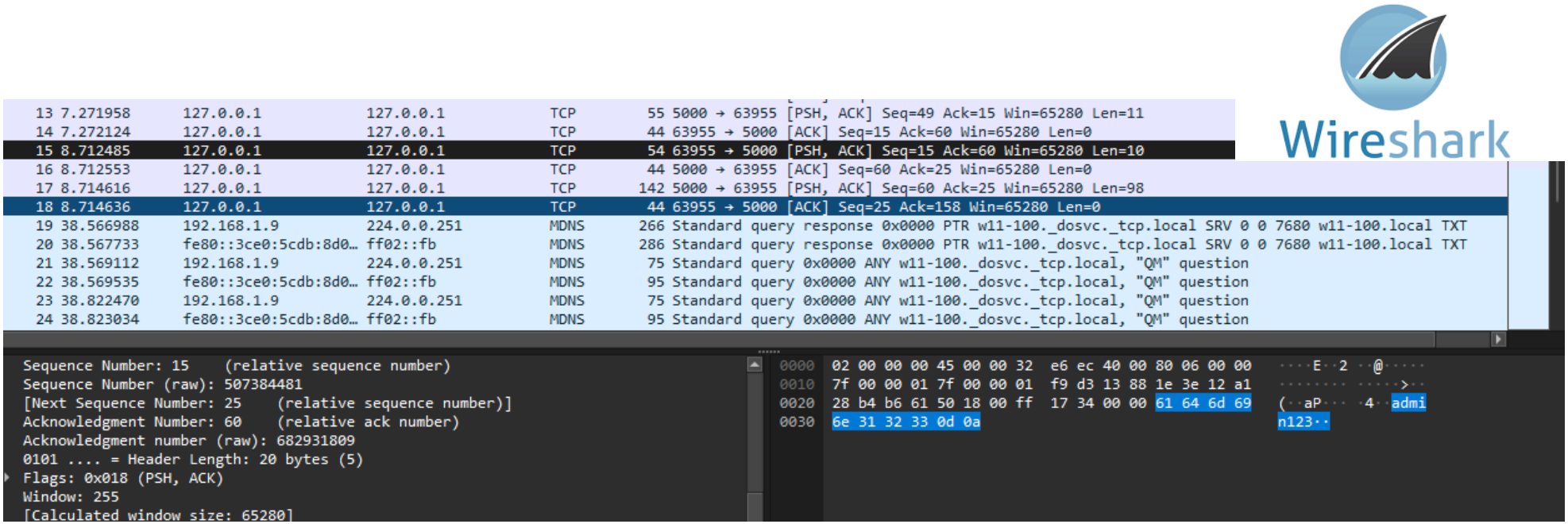
Programación y comunicaciones seguras

SOCKETS SEGUROS

- Tenemos una aplicación cliente-servidor concurrente funcionando.
- Ahora vamos a analizar si es segura.

➢ Cliente envía datos (líneas de texto), el servidor lo procesa.
➢ Todo viaja en texto plano.

¿Qué pasaría si alguien intercepta la comunicación?



SOCKETS SEGUROS


```
11 7.270914   127.0.0.1   127.0.0.1   TCP    51 63955 → 5000 [PSH, ACK] Seq=8 Ack=49 Win=65280 Len=7
12 7.270945   127.0.0.1   127.0.0.1   TCP    44 5000 → 63955 [ACK] Seq=49 Ack=15 Win=65280 Len=0
13 7.271958   127.0.0.1   127.0.0.1   TCP    55 5000 → 63955 [PSH, ACK] Seq=49 Ack=15 Win=65280 Len=11
14 7.272124   127.0.0.1   127.0.0.1   TCP    44 63955 → 5000 [ACK] Seq=15 Ack=60 Win=65280 Len=0
15 8.712485   127.0.0.1   127.0.0.1   TCP    54 63955 → 5000 [PSH, ACK] Seq=15 Ack=60 Win=65280 Len=10
16 8.712553   127.0.0.1   127.0.0.1   TCP    44 5000 → 63955 [ACK] Seq=60 Ack=25 Win=65280 Len=0
17 8.714616   127.0.0.1   127.0.0.1   TCP    142 5000 → 63955 [PSH, ACK] Seq=60 Ack=25 Win=65280 Len=98
18 8.714636   127.0.0.1   127.0.0.1   TCP    44 63955 → 5000 [ACK] Seq=25 Ack=158 Win=65280 Len=0
19 38.566988  192.168.1.9  224.0.0.251 MDNS   266 Standard query response 0x0000 PTR w11-100._dosvc._tcp.local SRV 0 0 7680 w11-100.local TXT
20 38.567733  fe80::3ce0:5cdb:8d0… ff02::fb MDNS 286 Standard query response 0x0000 PTR w11-100._dosvc._tcp.local SRV 0 0 7680 w11-100.local TXT
21 38.569112  192.168.1.9  224.0.0.251 MDNS   75 Standard query 0x0000 ANY w11-100._dosvc._tcp.local, "QM" question
22 38.569535  fe80::3ce0:5cdb:8d0… ff02::fb MDNS 95 Standard query 0x0000 ANY w11-100._dosvc._tcp.local, "QM" question
23 38.822470  192.168.1.9  224.0.0.251 MDNS   75 Standard query 0x0000 ANY w11-100._dosvc._tcp.local, "QM" question
24 38.823034  fe80::3ce0:5cdb:8d0… ff02::fb MDNS 95 Standard query 0x0000 ANY w11-100._dosvc._tcp.local, "QM" question

[TCP Segment Len: 7]
Sequence Number: 8    (relative sequence number)
Sequence Number (raw): 507384474
[Next Sequence Number: 15    (relative sequence number)]
Acknowledgment Number: 49    (relative ack number)

0000  02 00 00 00 45 00 00 2f  e6 e8 40 00 80 06 00 00   ····E··/ ··@·····
0010  7f 00 00 01 7f 00 00 01  f9 d3 13 88 1e 3e 12 9a   ········ ·····>··
0020  28 b4 b6 56 50 18 00 ff  4c aa 00 00 61 64 6d 69   (··VP··· L···admi
0030  6e 0d 0a                                          n··
```

```
12 7.270945   127.0.0.1   127.0.0.1   TCP    44 5000 → 63955 [ACK] Seq=49 Ack=15 Win=65280 Len=0
13 7.271958   127.0.0.1   127.0.0.1   TCP    55 5000 → 63955 [PSH, ACK] Seq=49 Ack=15 Win=65280 Len=11
14 7.272124   127.0.0.1   127.0.0.1   TCP    44 63955 → 5000 [ACK] Seq=15 Ack=60 Win=65280 Len=0
15 8.712485   127.0.0.1   127.0.0.1   TCP    54 63955 → 5000 [PSH, ACK] Seq=15 Ack=60 Win=65280 Len=10
16 8.712553   127.0.0.1   127.0.0.1   TCP    44 5000 → 63955 [ACK] Seq=60 Ack=25 Win=65280 Len=0
17 8.714616   127.0.0.1   127.0.0.1   TCP    142 5000 → 63955 [PSH, ACK] Seq=60 Ack=25 Win=65280 Len=98
18 8.714636   127.0.0.1   127.0.0.1   TCP    44 63955 → 5000 [ACK] Seq=25 Ack=158 Win=65280 Len=0
19 38.566988  192.168.1.9  224.0.0.251 MDNS   266 Standard query response 0x0000 PTR w11-100._dosvc._tcp.local SRV 0 0 7680 w11-100.local TXT
20 38.567733  fe80::3ce0:5cdb:8d0… ff02::fb MDNS 286 Standard query response 0x0000 PTR w11-100._dosvc._tcp.local SRV 0 0 7680 w11-100.local TXT
21 38.569112  192.168.1.9  224.0.0.251 MDNS   75 Standard query 0x0000 ANY w11-100._dosvc._tcp.local, "QM" question
22 38.569535  fe80::3ce0:5cdb:8d0… ff02::fb MDNS 95 Standard query 0x0000 ANY w11-100._dosvc._tcp.local, "QM" question
23 38.822470  192.168.1.9  224.0.0.251 MDNS   75 Standard query 0x0000 ANY w11-100._dosvc._tcp.local, "QM" question
24 38.823034  fe80::3ce0:5cdb:8d0… ff02::fb MDNS 95 Standard query 0x0000 ANY w11-100._dosvc._tcp.local, "QM" question

[TCP Segment Len: 11]
Sequence Number: 49    (relative sequence number)
Sequence Number (raw): 682931798
[Next Sequence Number: 60    (relative sequence number)]
Acknowledgment Number: 15    (relative ack number)
Acknowledgment number (raw): 507384481
0101 .... = Header Length: 20 bytes (5)
Flags: 0x018 (PSH, ACK)
Window: 255
[Calculated window size: 65280]

0000  02 00 00 00 45 00 00 33  e6 ea 40 00 80 06 00 00   ····E··3 ··@·····
0010  7f 00 00 01 7f 00 00 01  13 88 f9 d3 28 b4 b6 56   ········ ····(··V
0020  1e 3e 12 a1 50 18 00 ff  a1 c4 00 00 50 61 73 73   ·>··P··· ····Pass
0030  77 6f 72 64 3a 0d 0a                              word:··
```

```
13 7.271958   127.0.0.1   127.0.0.1   TCP    55 5000 → 63955 [PSH, ACK] Seq=49 Ack=15 Win=65280 Len=11
14 7.272124   127.0.0.1   127.0.0.1   TCP    44 63955 → 5000 [ACK] Seq=15 Ack=60 Win=65280 Len=0
15 8.712485   127.0.0.1   127.0.0.1   TCP    54 63955 → 5000 [PSH, ACK] Seq=15 Ack=60 Win=65280 Len=10
16 8.712553   127.0.0.1   127.0.0.1   TCP    44 5000 → 63955 [ACK] Seq=60 Ack=25 Win=65280 Len=0
17 8.714616   127.0.0.1   127.0.0.1   TCP    142 5000 → 63955 [PSH, ACK] Seq=60 Ack=25 Win=65280 Len=98
18 8.714636   127.0.0.1   127.0.0.1   TCP    44 63955 → 5000 [ACK] Seq=25 Ack=158 Win=65280 Len=0
19 38.566988  192.168.1.9  224.0.0.251 MDNS   266 Standard query response 0x0000 PTR w11-100._dosvc._tcp.local SRV 0 0 7680 w11-100.local TXT
20 38.567733  fe80::3ce0:5cdb:8d0… ff02::fb MDNS 286 Standard query response 0x0000 PTR w11-100._dosvc._tcp.local SRV 0 0 7680 w11-100.local TXT
21 38.569112  192.168.1.9  224.0.0.251 MDNS   75 Standard query 0x0000 ANY w11-100._dosvc._tcp.local, "QM" question
22 38.569535  fe80::3ce0:5cdb:8d0… ff02::fb MDNS 95 Standard query 0x0000 ANY w11-100._dosvc._tcp.local, "QM" question
23 38.822470  192.168.1.9  224.0.0.251 MDNS   75 Standard query 0x0000 ANY w11-100._dosvc._tcp.local, "QM" question
24 38.823034  fe80::3ce0:5cdb:8d0… ff02::fb MDNS 95 Standard query 0x0000 ANY w11-100._dosvc._tcp.local, "QM" question

Sequence Number: 15    (relative sequence number)
Sequence Number (raw): 507384481
[Next Sequence Number: 25    (relative sequence number)]
Acknowledgment Number: 60    (relative ack number)
Acknowledgment number (raw): 682931809
0101 .... = Header Length: 20 bytes (5)
Flags: 0x018 (PSH, ACK)
Window: 255
[Calculated window size: 65280]

0000  02 00 00 00 45 00 00 32  e6 ec 40 00 80 06 00 00   ····E··2 ··@·····
0010  7f 00 00 01 7f 00 00 01  f9 d3 13 88 1e 3e 12 a1   ········ ·····>··
0020  28 b4 b6 61 50 18 00 ff  17 34 00 00 61 64 6d 69   (··aP··· ·4··admi
0030  6e 31 32 33 0d 0a                                 n123··
```

SOCKETS SEGUROS

- Crear keystore del servidor (autofirmado para pruebas).
- C:\Program Files\Java\jdk-XX\bin\keytool.exe

➢ keytool -genkeypair -alias servidorpgvsat -keyalg RSA -keysize 2048 -validity 365 -storetype PKCS12 -keystore servidor-keystore.p12 -storepass servidorpgvsat -keypass servidorpgvsat -dname "CN=localhost, OU=PGV, O=PGV, L=Canarias, C=ES"

```
Windows PowerShell
PS C:\Users\Ruymán\Documents\javaworkspace\ut2_21_SATv4_sem_tls> keytool -genkeypair -alias servidorpgvsat -keyalg RSA -keysize 2048
-validity 365 -storetype PKCS12 -keystore servidor-keystore.p12 -storepass servidorpgvsat -keypass servidorpgvsat -dname "CN=localhos
t, OU=PGV, O=PGV, L=Canarias, C=ES"
Generando par de claves RSA de 2.048 bits para certificado autofirmado (SHA256withRSA) con una validez de 365 días
        para: CN=localhost, OU=PGV, O=PGV, L=Canarias, C=ES
PS C:\Users\Ruymán\Documents\javaworkspace\ut2_21_SATv4_sem_tls>
```

➢ ls servidor-keystore.p12

```
PS C:\Users\Ruymán\Documents\javaworkspace\ut2_21_SATv4_sem_tls> ls servidor-keystore.p12


    Directorio: C:\Users\Ruymán\Documents\javaworkspace\ut2_21_SATv4_sem_tls


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        21/01/2026     11:51           2712 servidor-keystore.p12
```

SOCKETS SEGUROS

- -alias satserver: Es el nombre interno que le das a la llave dentro del archivo. Como si fuera una "etiqueta" para encontrarla luego.
- -keyalg RSA: Especifica el algoritmo matemático para el cifrado. RSA es el estándar más compatible.
- -keysize 2048: El tamaño de la llave. 2048 bits es el estándar de seguridad actual (más pequeño sería inseguro, más grande sería lento).
- -validity 365: El certificado caducará en un año (365 días).
- -storetype PKCS12: Define el formato del archivo. PKCS12 es el estándar moderno que usan casi todos los sistemas (Windows, Java, navegadores).
- -keystore servidor-keystore.p12: El nombre del archivo físico que se va a crear en el directoio.
- -storepass *******: La contraseña para abrir el archivo completo.
- -keypass *****: La contraseña específica para la llave privada.

SOCKETS SEGUROS

- Guardamos los certificados en un directorio específico “/.certs”.
- Ponemos dentro el fichero servidor-keystore.p12.

- Exportamos el certificado. Generamos el fichero servidorpgvsat.cer.
- ➢ keytool -exportcert -alias servidorpgvsat -keystore .certs/servidor-keystore.p12 -storepass servidorpgvsat -rfc -file .certs/servidorpgvsat.cer

```
PS C:\Users\Ruymán\Documents\javaworkspace\ut2_21_SATv4_sem_tls> keytool -exportcert -alias servidorpgvsat -keystore .certs/servidor-
keystore.p12 -storepass servidorpgvsat -rfc -file .certs/servidorpgvsat.cer
Certificado almacenado en el archivo <.certs/servidorpgvsat.cer>
PS C:\Users\Ruymán\Documents\javaworkspace\ut2_21_SATv4_sem_tls>
```

- Añadimos al truststore del cliente, para que confíe en él.
- ➢ keytool -importcert -alias servidorpgvsat -file .certs/servidorpgvsat.cer -storetype PKCS12 -keystore .certs/cliente-truststore.p12 -storepass servidorpgvsat -noprompt

```
PS C:\Users\Ruymán\Documents\javaworkspace\ut2_21_SATv4_sem_tls> keytool -importcert -alias servidorpgvsat -file .certs/servidorpgvsa
t.cer -storetype PKCS12 -keystore .certs/cliente-truststore.p12 -storepass servidorpgvsat -noprompt
Se ha agregado el certificado al almacén de claves
PS C:\Users\Ruymán\Documents\javaworkspace\ut2_21_SATv4_sem_tls>
```

SOCKETS SEGUROS

- Comprobamos.


```
    Directorio: C:\Users\Ruymán\Documents\javaworkspace\ut2_21_SATv4_sem_tls\.certs


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         21/01/2026     12:05           1222 cliente-truststore.p12
-a----         21/01/2026     11:51           2712 servidor-keystore.p12
-a----         21/01/2026     12:05           1212 servidorpgvsat.cer
```

SOCKETS SEGUROS

- Cambios en el código fuente en el servidor.

- `import javax.net.ssl.SSLServerSocket;`
- `import javax.net.ssl.SSLServerSocketFactory;`
- `import javax.net.ssl.SSLSocket;`

```
System.setProperty("javax.net.ssl.keyStore", ".certs/servidor-keystore.p12");
System.setProperty("javax.net.ssl.keyStorePassword", "servidorpgvsat");
System.setProperty("javax.net.ssl.keyStoreType", "PKCS12");
SSLServerSocketFactory ssf = (SSLServerSocketFactory) SSLServerSocketFactory.getDefault();
SSLServerSocket servidor = (SSLServerSocket) ssf.createServerSocket(PUERTO);
// Abrimos el puerto
//ServerSocket servidor = new ServerSocket(PUERTO);


….


//Socket socketCliente = servidor.accept();
SSLSocket socketCliente = (SSLSocket) servidor.accept();
```

SOCKETS SEGUROS

- Cambios en el código fuente en el cliente.

- `import javax.net.ssl.SSLSocket;`
- `import javax.net.ssl.SSLSocketFactory;`

```
final String HOST = "localhost";
final int PUERTO = 5000;


//Configuración SSL/TLS del cliente
System.setProperty("javax.net.ssl.trustStore", ".certs/cliente-truststore.p12");
System.setProperty("javax.net.ssl.trustStorePassword", "servidorpgvsat");
System.setProperty("javax.net.ssl.trustStoreType", "PKCS12");
try (
//Socket socket = new Socket(HOST, PUERTO);
SSLSocket socket = (SSLSocket) ((SSLSocketFactory) SSLSocketFactory.getDefault())
.createSocket(HOST, PUERTO);
```

aaaa

- Resultado, codificado.


```
     4 0.029118    127.0.0.1     127.0.0.1     RSL      472 HANDOver DETection
     5 0.029253    127.0.0.1     127.0.0.1     TCP       44 5000 → 62015 [ACK] Seq=1 Ack=429 Win=65024 Len=0
     6 0.031400    127.0.0.1     127.0.0.1     RSL      171 unknown 122
     7 0.031415    127.0.0.1     127.0.0.1     TCP       44 62015 → 5000 [ACK] Seq=429 Ack=128 Win=65280 Len=0
     8 0.032416    127.0.0.1     127.0.0.1     RSL       50 DATA REQuest [Malformed Packet: length of contained item exceeds length of containing item]
     9 0.032427    127.0.0.1     127.0.0.1     TCP       44 62015 → 5000 [ACK] Seq=429 Ack=134 Win=65280 Len=0
    10 0.032964    127.0.0.1     127.0.0.1     RSL      114 Location Information [Malformed Packet: length of contained item exceeds length of containing
    11 0.032971    127.0.0.1     127.0.0.1     TCP       44 62015 → 5000 [ACK] Seq=429 Ack=204 Win=65280 Len=0
    12 0.033173    127.0.0.1     127.0.0.1     RSL      934 ip.access MDCX NACK
    13 0.033181    127.0.0.1     127.0.0.1     TCP       44 62015 → 5000 [ACK] Seq=429 Ack=1094 Win=64256 Len=0
    14 0.034483    127.0.0.1     127.0.0.1     RSL      346 MODE MODIFY REQuest
    15 0.034493    127.0.0.1     127.0.0.1     TCP       44 62015 → 5000 [ACK] Seq=429 Ack=1396 Win=64000 Len=0
    16 0.034619    127.0.0.1     127.0.0.1     RSL      134 unknown 85
    17 0.034624    127.0.0.1     127.0.0.1     TCP       44 62015 → 5000 [ACK] Seq=429 Ack=1486 Win=64000 Len=0
    18 0.053500    127.0.0.1     127.0.0.1     RSL       50 DATA REQuest [Malformed Packet: length of contained item exceeds length of containing item]

▸ Frame 7: Packet, 44 bytes on wire (352 bits), 44 bytes captured (352 bits) on interf
▸ Null/Loopback
▸ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

0000  02 00 00 00 45 00 00 28  e7 63 40 00 80 06 00 00   ····E··(  ·c@·····
0010  7f 00 00 01 7f 00 00 01  f2 3f 13 88 6d 34 37 27   ········  ·?··m47'
0020  46 16 04 f3 50 10 00 ff  bb a6 00 00              F···P···  ····
```

NORMALIZACIÓN Y VALIDACIÓN

NORMALIZACIÓN

- Normalizar comandos.
- Asegurar los strings que recibimos o mandamos.

```
//#########################################################################################
// NORMALIZAMOS EL COMANDO QUE NOS LLEGA CON EL READLINE
//#########################################################################################
String cmd = normalizarComando(comando);
if (cmd == null) {
salida.println("ERROR Comando invalido");
Logger.warning("Comando invalido recibido. cliente id=" + idCliente + " valor='" + comando + "'");
continue;
}
```

```
//lista de comandos correctos
private static final Set<String> COMANDOS_VALIDOS =
        Set.of("SALIR", "ALTA", "LISTAR", "CLIENTES", "LOGIN");
```

```
//#########################################################################################
// Método para normalizar los comandos que nos llegan del cliente
//#########################################################################################
private String normalizarComando(String comando) {

    if (comando == null) return null;

    //1)Quitamos espacios y pasamos a mayúsculas
    String cmd = comando.trim().toUpperCase();
    //2)No permitimos vacío
    if(cmd.isEmpty()) return null;
    //3)Longitud máxima para evitar cosas extrañas
    if (cmd.length() > 12) return null;
    //4)Solo letras (lista blanca de formato)
    if (!cmd.matches("[A-Z]+")) return null;
    //5)Lista blanca de comandos permitidos
    /*
    switch (cmd) {
        case "SALIR":
        case "ALTA":
        case "LISTAR":
        case "CLIENTES":
            return cmd;
        default:
            return null;
    }
    */
    if (!COMANDOS_VALIDOS.contains(cmd)) return null;
    return cmd;
}
```

NORMALIZACIÓN

- Normalizar descripción.

```java
//creamos un bloque nuevo por restricciones de java
//las creadas dentro de los case se comparten y si se crea más tarde da error
String descripcion = entrada.readLine();
//if (descripcion == null) return; // Desconexión inesperada
//Limpiamos la descrición
String descLimpia = normalizarDescripcion(descripcion);
if (descLimpia == null) {
salida.println("ERROR Descripcion invalida");
logger.warning("ALTA con descripcion invalida. cliente id=" + idCliente);
break;
}
```

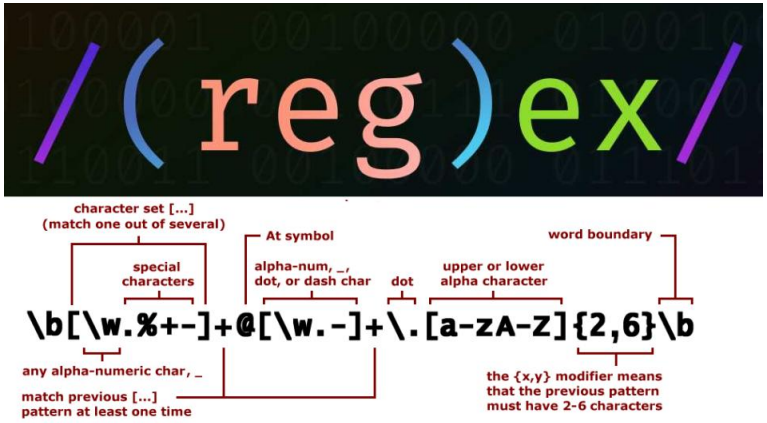
```java
//##########################################################################################
// Comprobamos que la descripción es adecuada
//##########################################################################################
private String normalizarDescripcion(String descripcion) {

    if (descripcion == null) return null;

    String des = descripcion.trim();

    if (des.isEmpty()) return null;
    if (des.length() > 200) return null;
    if (!des.matches("[A-Za-z0-9 áéíóúÁÉÍÓÚñÑ.,;:()/_\\-]+")) return null;

    return des;
}
```

NORMALIZACIÓN

- if (!des.matches("[A-Za-z0-9 áéíóúÁÉÍÓÚñÑ.,;:()/_\\-]+")) return null;

- matches() es un método de String.
- Comprueba si toda la cadena cumple una expresión regular (regex).
- Devuelve:
  - ❑ true texto válido
  - ❑ false texto no válido

Permitimos letras, números, espacios, acentos, ñ y signos básicos; se rechaza cualquier otro carácter.



NORMALIZACIÓN

- **REGEX**

➢ Se valida la entrada del usuario para permitir únicamente caracteres válidos, evitando errores y mejorando la seguridad y estabilidad de la aplicación. EJEMPLO:

- Error en impresora
- Fallo módulo_1
- Revisión equipo portátil
- Incidencia (urgente)
- Problema red-local
- Temperatura 25 grados

- Error@impresora
- <script>alert(1)</script>
- DROP TABLE usuarios
- hola|admincontraseña=1234
- comando && rm -rf

```
String descripcion = "Error en impresora HP";

if (!descripcion.matches("[A-Za-z0-9 áéíóúÁÉÍÓÚñÑ.,;:()/_\\-]+")) {
    System.out.println("Texto no válido");
}
```

NORMALIZACIÓN

- matches() es el método que usa Java para comprobar una expresión regular (regex).
- La regex es la regla, y matches() es quien la aplica.

- ❑ Regex (expresión regular) Define qué está permitido y qué no.
- ❑ matches() Comprueba si un texto cumple exactamente esa regla.

```
String texto = "Hola123";

boolean valido = texto.matches("[A-Za-z0-9]+");
```

- [A-Za-z0-9]+ es la expresión regular
- matches(...) comprueba si el texto cumple esa expresión
- Si cumple: true
- Si no cumple false

NORMALIZACIÓN

- Filtrar en vez de rechazar.

```
String texto = "Error@impresora#HP!";

String filtrado = texto.replaceAll("[^A-Za-z0-9 áéíóúÁÉÍÓÚñÑ.,;:()/_\\-]", "");

>>> Resultado: "ErrorimpresoraHP"
```

➢ [^ ...] Todo lo que esté dentro del conjunto.

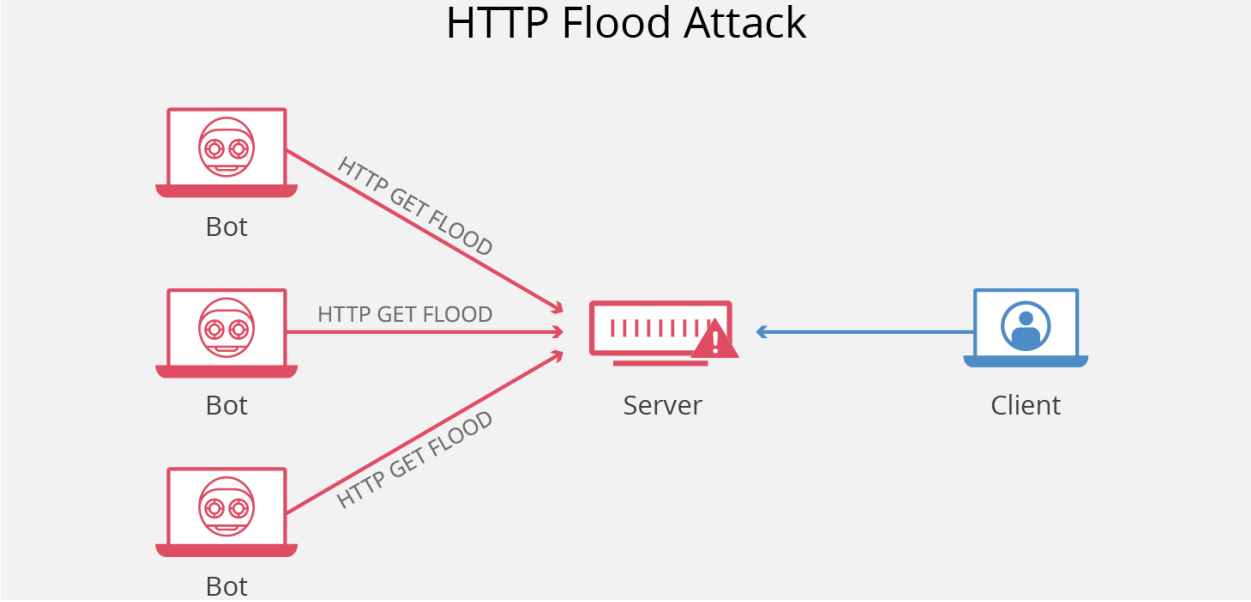
```
//Primero filtro y después si no quedó nada válido retorno null

String limpio = texto.replaceAll("[^A-Za-z0-9 áéíóúÁÉÍÓÚñÑ.,;:()/_\\-]", "");

if (limpio.isBlank()) {
    return null;
}
```

PROTECCIÓN FLOODS PETICIONES

PROTECCIÓN FRENTE A FLOODS

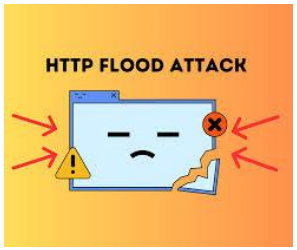
- Se ha implementado una protección frente a floods para evitar que un cliente envíe un número excesivo de peticiones en un corto periodo de tiempo.

- Este mecanismo permite limitar el uso indebido del sistema, mejorar la estabilidad del servidor y prevenir posibles bloqueos derivados de un uso abusivo de la conexión.



PROTECCIÓN FRENTE A FLOODS

- Limitamos el número de peticiones por usuario.
- Solo contamos las peticiones si no son salir o login (ejemplo)




```
// ---- Límite de peticiones por sesión (solo comandos)
private static final int MAX_PETICIONES = 20;
private int peticionesRealizadas = 0;
```

```
//Limitamos por sesion: en produccion se haria por tiempo (peticiones/minuto) y/o por IP.
if (!cmd.equals("LOGIN") && !cmd.equals("SALIR")) {
peticionesRealizadas++;
}
```

```
        //Limitamos por sesión porque es simple y efectivo: en producción se haría por tiempo (peticiones/minuto) y/o por IP."
        peticionesRealizadas++;

        if (peticionesRealizadas > MAX_PETICIONES) {
            salida.println("ERROR Limite de peticiones alcanzado (" + MAX_PETICIONES + "). Conexion cerrada.");
            logger.warning("Limite de peticiones alcanzado. cliente id=" + idCliente + " peticiones=" + peticionesRealizadas);
            return; // corta la sesión (finally cerrará socket y limpiará)
        }
```

PROTECCIÓN CONEXIONES USUARIO CONECTADOS

USUARIOS MÁXIMOS CONECTADOS

- Se utiliza un semáforo para limitar el número máximo de clientes conectados simultáneamente al servidor.

- Si no hay permisos disponibles, la conexión se rechaza de forma controlada, se informa al cliente y se libera el socket, evitando la saturación del servidor.

- **tryAcquire()** intenta obtener un permiso del semáforo sin bloquear el hilo.
- Si hay permisos disponibles devuelve true y el cliente es aceptado, si no los hay devuelve false y la conexión se rechaza de forma inmediata

```
//semáforo para limitar clientes en server
if (!semaforoClientes.tryAcquire()) {
Logger.warning("Servidor saturado: rechazando cliente desde=" +
socketCliente.getRemoteSocketAddress());
try (PrintWriter out = new PrintWriter(socketCliente.getOutputStream(), true)) {
out.println("ERROR Servidor ocupado. Intenta mas tarde.");
} catch (IOException ignored) {}
socketCliente.close();
continue;
}
```

IDENTIFICACIÓN DE USUARIOS
LOGIN

IDENTIFICACIÓN LOGIN

- Necesitamos realizar un login en nuestro servidor.
- Haremos uso de un login simulado con usuario admin y técnico predefinido.

admin – admin123
tecnico – tecnico123

```
//clase interna para los datos solamente
public static class AuthResult {
    public boolean login;
    public String usuario;
    public String rol;
    public String token;
    public String error;
    public String rawBody;
}
```

```
private String generarToken() {
    return java.util.UUID.randomUUID().toString();
}
```

```
private AuthResult autenticarSimulado(String user, String pass) {
    AuthResult r = new AuthResult();

    // Normalización aquí (un solo sitio)
    if (user != null) user = user.trim();
    if (pass != null) pass = pass.trim();
    // Seguridad extra
    if (user == null || pass == null) {
        r.autenticado = false;
        return r;
    }
    // Simulación: luego esto será llamada a tu API PHP
    // Usuarios de ejemplo:
    // admin / admin123  -> ADMIN
    // tecnico / tecnico123 -> TECNICO

    if ("admin".equals(user) && "admin123".equals(pass)) {
        r.autenticado = true;
        r.usuario = "admin";
        r.rol = "ADMIN";
        r.token = generarToken();
        return r;
    }

    if ("tecnico".equals(user) && "tecnico123".equals(pass)) {
        r.autenticado = true;
        r.usuario = "tecnico";
        r.rol = "TECNICO";
        r.token = generarToken();
        return r;
    }

    r.autenticado = false;
    return r;
}
```

IDENTIFICACIÓN LOGIN

- Ejemplo del login. **admin – admin123** **tecnico – tecnico123**

```
case "LOGIN":
    salida.println("Usuario:");
    String userIn = entrada.readLine();

    salida.println("Password:");
    String passIn = entrada.readLine();

    if (!usuarioValido(userIn) || !passwordValida(passIn)) {
        logger.warning("LOGIN formato invalido. cliente id=" + idCliente);
        break;
    }

    AuthResult r = autenticarSimulado(userIn, passIn);

    if (!r.autenticado) {
        logger.warning("LOGIN fallido. cliente id=" + idCliente);
        break;
    }

    // Guardamos sesión
    autenticado = true;
    usuario = r.usuario;
    rol = r.rol;
    token = r.token;

    logger.info("LOGIN ok. cliente id=" + idCliente + " user=" + usuario + " rol=" + rol);
    break;
```

VALIDACIÓN

```java
//#########################################################################################
//validar el user
//#########################################################################################
private boolean usuarioValido(String u) {
if (u == null) return false;
u = u.trim();
return !u.isEmpty()
&& u.length() <= 20
&& u.matches("[a-zA-Z0-9_]+"); // simple y seguro
}
//#########################################################################################
//validar el pass
//#########################################################################################
private boolean passwordValida(String p) {
if (p == null) return false;
p = p.trim();
return !p.isEmpty()
&& p.length() <= 30
&& p.matches("[a-zA-Z0-9@#%_\\-!.]+"); // ejemplo básico
}
```

IDENTIFICACIÓN CON API EXTERNA JSON Y APIKEY

JSON

- JSON (*JavaScript Object Notation*) es un formato ligero de intercambio de datos, basado en texto, que permite representar información de forma estructurada mediante pares clave Y valor.

- JSON se utiliza de forma mayoritaria en las APIs actuales porque:
  - ❑ Es fácil de leer y escribir, tanto para personas como para máquinas.
  - ❑ Es independiente del lenguaje (funciona igual en Java, Python, JavaScript, etc.)
  - ❑ Tiene poco tamaño, lo que mejora el rendimiento en red.
  - ❑ Se integra de forma natural con servicios web y APIs REST

En este proyecto, JSON se utiliza para intercambiar información estructurada (credenciales, respuestas del servidor, datos de estado) de forma clara y estandarizada entre cliente y servidor o entre el sistema y una API externa.

JSON

- Añadir librería si usamos Maven.
- La descarga y la guarda en el equipo.
- La añade al classpath.
- Nos permite usarla.
- No descargamos JAR manualmente.
- Edición del POM (Project Object Model).

```
<dependency>
   <groupId>com.fasterxml.jackson.core</groupId>
   <artifactId>jackson-databind</artifactId>
</dependency>
```

JSON

ut3_1_vlogindemojson/pom.xml

```
    </scm>
    <properties>
        <java.version>17</java.version>
    </properties>
    <dependencies> Add Spring Boot Starters...
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>

        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
        </dependency>

    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
```

JSON

```
//las clases importantes para poder usa JSON
import com.fasterxml.jackson.annotation.JsonProperty;
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;


//le decimos a jackson que si los campos que no tenemos que los ignore, para evitar
posibles errores
//Si no existe en la clase no haya ningún error, por si el server añade alguno extra
//que todavía no hemos implementado
@JsonIgnoreProperties(ignoreUnknown = true)
```

JSON

ENVIAMOS

```
{
  "user": "admin",
  "password": “admin123"
}
```

RECIBIMOS

```
{
  “login": true,
  "user": "admin",
  "rol": "ADMIN",
  "token": "abc123xyz"
}
```

HTTP PETICIÓN/RESPUESTA

Ejemplo en el Campus

```java
//las clases importantes para poder usa JSON
import com.fasterxml.jackson.annotation.JsonProperty;



//le decimos a jacksion que si los campos que no tenemos que los ignoe, para evitar posibles errpores
//Si no existe en la clase no haya ningún error, por si el server añade alguno extra
//que todavía no hemos implementado
@JsonIgnoreProperties(ignoreUnknown = true)


//clase sin logica solo datos!

public class AuthResultado {

    //Respuesta del server
    public boolean login;

    //Caso en el que el json trae "user" y en nuestro código tenemos "usuario"
    @JsonProperty("user") //para a la variable usuario
    public String usuario;

    public String rol;
    public String token;
    public String error;

    //campo extra para guardar el JSON original para verlo en los logs
    public String rawBody;//lo rellenamos manualmente

    //funademantal si no da error al deserializar
    //importante: Jackson necesita un constructor vacío (que Java crea por defecto si no hay otros)
    public AuthResultado() {
    }
}
```

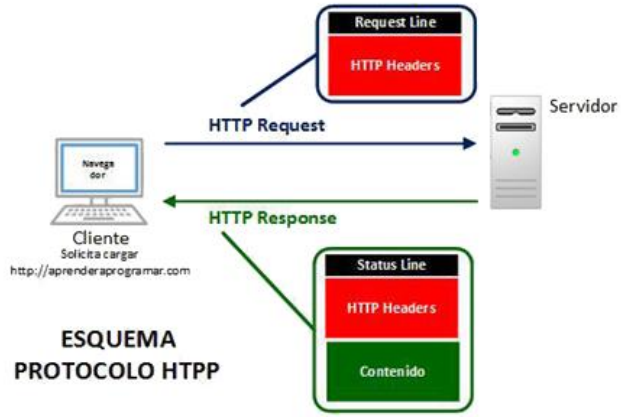
MODELO HTTP

- HTTP funciona con un modelo petición - respuesta:
  1) El cliente envía una petición al servidor.
  2) El servidor procesa la petición.
  3) El servidor devuelve una respuesta.

- Cada comunicación es independiente y el servidor no mantiene estado entre peticiones (**stateless**).

- **Una petición HTTP incluye:**
  - ❑ Método (GET, POST, PUT, DELETE…).
  - ❑ URL del recurso.
  - ❑ Cabeceras (headers) con información adicional.
  - ❑ Cuerpo (body), opcional (por ejemplo JSON).

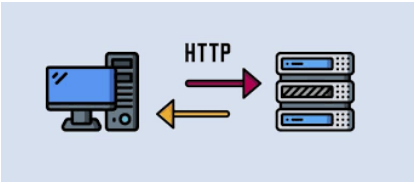
```
POST /login
Headers:
  Authorization: ApiKey abc123
Body:
{
  "user": "admin",
  "password": "admin123"
}
```



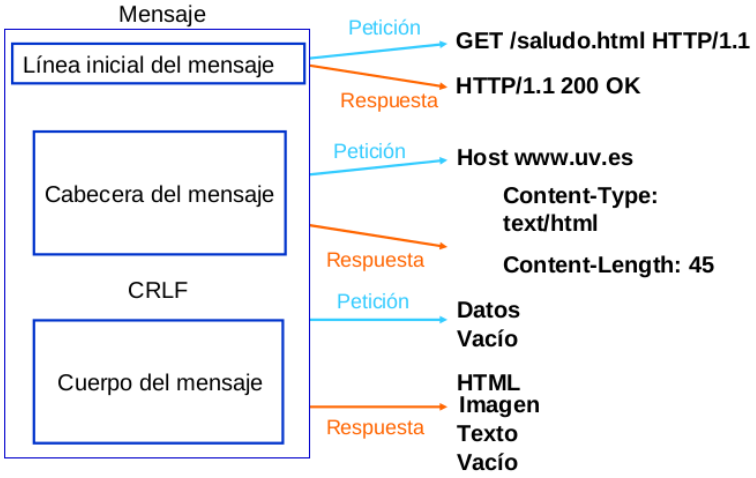
MODELO HTTP

- La API Key sirve para identificar y autorizar al cliente que realiza la petición.

- **Se envía en el header porque:**
  - ❑ No se mezcla con los datos del cuerpo
  - ❑ No aparece en la URL
  - ❑ Es el lugar estándar para credenciales
  - ❑ Facilita su validación en el servidor



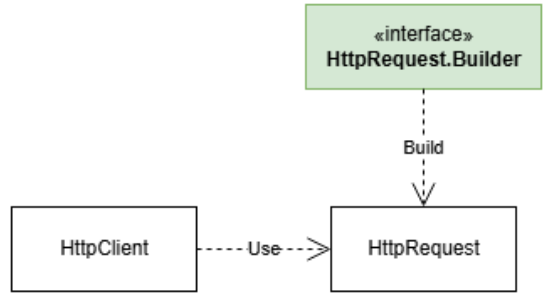
El cliente envía una petición HTTP con los datos en formato JSON y una clave de acceso en la cabecera. El servidor valida la clave, procesa la petición y devuelve una respuesta.



HTTP PETICIÓN/RESPUESTA

```
//Especial API, ruta raíz, creamos un objeto nuevo
//Desde la clase servidor
ApiCliente api = new ApiCliente("https://pgv.ciclofp.es");
```

```
public class ApiCliente {

    //Creamos un cliente HTTP para toda la clase, lo reutilizamos
    //En la construcción le damos un timeout personalizado de 3s
    private final HttpClient http = HttpClient.newBuilder()
    .connectTimeout(Duration.ofSeconds(3))
    .build();
```




```
//construimos la petición con los diferentes parámetros
HttpRequest req = HttpRequest.newBuilder()
.uri(URI.create(baseUrl + "/api/logindemo"))
.header("Content-Type", "application/json")
.header("X-APP-KEY", "MI_CLAVE_APP_123")
.timeout(Duration.ofSeconds(5))
.POST(HttpRequest.BodyPublishers.ofString(jsonReq)) //método post y el body
.build();//finalizamos la petición


//enviamos la petición, la enviamos de forma síncrona. el body como string
HttpResponse<String> res =
http.send(req, HttpResponse.BodyHandlers.ofString());
```

REGISTRO DE LOGS CON NIVELES

LOGS

- Programar pensando en seguridad
- Validar datos recibidos
- Gestionar errores correctamente
- **Registrar lo que ocurre (logs)**

- En lugar de usar System.out.println(), se utiliza un sistema de logging porque permite:
  - ❑ Clasificar los mensajes por nivel de importancia
  - ❑ Activar o desactivar mensajes según el contexto
  - ❑ Analizar errores sin modificar el código
  - ❑ Mantener un registro del funcionamiento del sistema

❖ **Ejemplos**


```
Logger logger = Logger.getLogger("ServidorSAT");
//private static final Logger logger =
Logger.getLogger(ServidorPersistenteSAT.class.getName());

logger.setLevel(Level.INFO);

logger.info("Servidor iniciado");
logger.warning("Cliente rechazado");
logger.severe("Error al acceder a fichero");
```

TAREA UT3

TAREA UT3 – OPCIONAL PUNTUABLE

- Base:
  1) Sockets con SSL/TLS.
  2) Identificación, login (nuevo comando). Simulado.
  3) Normalización de textos, comandos, etc.

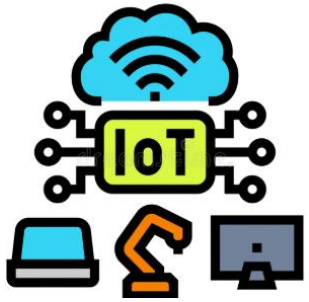
- Suma adicional de puntos:
  1) Validación de usuario contraseña con normalización.
  2) Protección frente a floods.
  3) Usuarios máximos conectados.
  4) Uso de logs con diferentes y eliminar los mensajes en consola directos.
  5) Identificación externa, consumo de API con KEY, envío y recepción de JSON.

PROYECTOS

PROYECTOS FINALES DE PGV

- Incidencias.
- Continuación de lo que tenemos
- Info de clientes (+user)
- Listado de clientes solo admin
- Roles e Identificación
- Limite de clientes
- Validar y normalizar
- Sockets seguros
- Editar y cancelar.

- ❖ API Login APIKEY
- ❖ Logs niveles
- ❖ Persistencia (autoapren.)



- Dispositivos IoT.
- Registro de datos de un dispo con id.
- Listas datos de un dispositivo o de los dispositivos.
- Listado clientes conectados.
- Roles e Identificación
- Limite de clientes
- Validar y normalizar
- Sockets seguros

- ❖ API Login APIKEY
- ❖ Logs niveles
- ❖ Persistencia (autoapren.)

- Reservas con una línea texto.
- Estados, cuando se inserta activa Listado clientes, solo admin.
- Roles e Identificación.
- Limite de clientes.
- Validar y normalizar.
- Sockets seguros.
- Editar y cancelar.

- ❖ API Login APIKEY
- ❖ Logs niveles
- ❖ Persistencia (autoapren.)

AÑADIR INFORMACIÓN EXTRA QUE SE LES OCURRA SUMA PUNTOS

Programación de servicios y procesos

S5
UT3

Programación y comunicaciones seguras