PSP\Tarea4PSP\src\tarea4pspsolucion\ServidorHTTPS.java

```java
package tarea4pspsolucion;

import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.nio.charset.StandardCharsets;
import java.security.KeyStore;
import java.util.concurrent.ConcurrentHashMap;
import java.util.logging.FileHandler;
import java.util.logging.Logger;
import javax.net.ssl.KeyManagerFactory;
import javax.net.ssl.SSLContext;
import javax.net.ssl.SSLServerSocket;
import javax.net.ssl.SSLServerSocketFactory;
import javax.net.ssl.SSLSocket;

/**
 * Servidor HTTPS que maneja juegos interactivos.
 *
 * Rutas disponibles: - /adivina: Juega a "Adivina el Número". - /dados: Juega a
 * "Lanza Dados". - /ppt: Juega a "Piedra, Papel o Tijera".
 */
public class ServidorHTTPS {

    private static final ConcurrentHashMap<String, SesionJuego> sesiones = new ConcurrentHashMap<>();
    private static final Logger logger = Logger.getLogger("logErrores");
    private static FileHandler fh;

    public static void main(String[] args) throws Exception {

        // Establecemos el fichero que vamos a utilizar y que
        // se vayan añadiendo las lineas
```

```
            fh = new FileHandler("logErrores.txt", true);

            // Utilizamos el método de configuración de la clase Log
            GestionLog.configurarLog(fh, logger);

            // Cargar el almacén de claves (keystore)
            KeyStore keyStore = KeyStore.getInstance("JKS");
            try (FileInputStream keyFile = new FileInputStream("AlmacenSSL")) {
                keyStore.load(keyFile, "123456".toCharArray());
            }

            // Inicializar el gestor de claves con el keystore
            KeyManagerFactory keyManagerFactory = KeyManagerFactory.getInstance("SunX509");
            keyManagerFactory.init(keyStore, "123456".toCharArray()); // Usa la misma contraseña

            // Inicializar el contexto SSL con el gestor de claves
            SSLContext sslContext = SSLContext.getInstance("TLS");
            sslContext.init(keyManagerFactory.getKeyManagers(), null, null);

            // Declara objeto tipo Factory para crear socket SSL servidor
            SSLServerSocketFactory factory = sslContext.getServerSocketFactory();

            // Crea un socket servidor seguro
            SSLServerSocket socketServidorSsl = (SSLServerSocket) factory.createServerSocket(8444);
            System.out.println("Servidor SSL escuchando en el puerto " + 8444);

            while (true) {
                // Acepta conexiones de clientes
                SSLSocket socketSsl = (SSLSocket) socketServidorSsl.accept();
                System.out.println("Cliente conectado");
                // Crea un nuevo hilo para manejar al cliente.
                Thread hiloCliente = new HiloCliente(socketSsl);
                hiloCliente.start(); // Inicia el hilo.
            }
        }

        /**
```

```
     * Clase interna que implementa la lógica de manejar un cliente. Extiende la
     * clase Thread y sobrescribe el método run.
     */
    private static class HiloCliente extends Thread {

        private static final GestionPeticiones gPeticiones = new GestionPeticiones(sesiones, logger);
        private final SSLSocket cliente;

        public HiloCliente(SSLSocket cliente) {
            this.cliente = cliente; // Asocia el socket del cliente al hilo.
        }

        @Override
        public void run() {
            try (BufferedReader entrada = new BufferedReader(
                            new InputStreamReader(cliente.getInputStream()));
                 PrintWriter salida = new PrintWriter(cliente.getOutputStream(), true, StandardCharsets.UTF_8)) {
                // Lee la primera línea de la petición HTTP.
                String peticion = entrada.readLine();
                if (peticion == null || (!peticion.startsWith("GET") && !peticion.startsWith("POST"))) {
                    return; // Ignora la petición si no es GET o POST.
                }
                System.out.println("peticion: " + peticion);
                String ruta = peticion.split(" ")[1]; // Extrae la ruta solicitada.

                // Leer encabezados HTTP. Determina la sesionID y el tamaño del cuerpo.
                String[] metadatos = new String[2];
                metadatos = obtenerMetadatos(entrada);
                String sessionId = metadatos[0];

                SesionJuego sesion = null;
                if (sessionId != null) { // Si en los metadatos encontramos una cookie, cargamos su sesión
                    sesion = sesiones.get(sessionId);
                }

                int contentLength = Integer.parseInt(metadatos[1]);
```

```
            System.out.println("linea: vacia");

            // Leer el cuerpo si es un POST.
            StringBuilder cuerpo = new StringBuilder(); // Para almacenar el cuerpo de la solicitud.
            if (peticion.startsWith("POST") && contentLength > 0) {
                char[] buffer = new char[contentLength];
                entrada.read(buffer, 0, contentLength);
                cuerpo.append(buffer);
            }

            String respuesta; // Contendrá¡ la respuesta generada por el servidor.

            if (ruta.equals("/")) {
                respuesta = gPeticiones.construirRespuesta(200,
                        (sesion != null ? Paginas.html_index : Paginas.html_LoginRegistro("")), sessionId);
            } else if (ruta.startsWith("/login")) {
                respuesta = gPeticiones.manejarLogin(cuerpo.toString(), sessionId);
            } else if (ruta.startsWith("/registro")) {
                respuesta = gPeticiones.manejarRegistro(cuerpo.toString(), sessionId);
            } else if (ruta.startsWith("/adivina")) {
                respuesta = gPeticiones.manejarAdivina(cuerpo.toString(), sesion, sessionId);
            } else if (ruta.startsWith("/dados")) {
                respuesta = gPeticiones.manejarDados(cuerpo.toString(), sesion, sessionId);
            } else if (ruta.startsWith("/ppt")) {
                respuesta = gPeticiones.manejarPPT(cuerpo.toString(), sesion, sessionId);
            } else if (ruta.startsWith("/logout")) {
                respuesta = gPeticiones.manejarLogout(sessionId);
            } else {
                respuesta = gPeticiones.construirRespuesta(404, Paginas.html_noEncontrado, sessionId);
            }

            salida.println(respuesta); // Envía la respuesta al cliente.
            cliente.close(); // Cierro conexión
        } catch (IOException e) {
            //e.printStackTrace(); // Muestra errores en la consola.
        }
    }
```

```
        private String[] obtenerMetadatos(BufferedReader entrada) throws IOException {
            String linea;
            String[] metadatos = new String[2];
            String sessionId = null;
            String contentLength = "0";

            while (!(linea = entrada.readLine()).isBlank()) {
                System.out.println("Metadato: " + linea);
                if (linea.startsWith("Cookie: ")) {
                    String[] cookies = linea.substring(8).split("; ");
                    for (String cookie : cookies) {
                        if (cookie.startsWith("sessionId=")) {
                            sessionId = cookie.substring(10);
                        }
                    }
                } else if (linea.startsWith("Content-Length: ")) {
                    contentLength = linea.substring(16);
                }
            }

            metadatos[0] = sessionId;
            metadatos[1] = contentLength;
            return metadatos;
        }
    }
}
```