🛡️ AI Code Analysis Report

Análisis inteligente de seguridad y calidad de código

📅 Fecha de Análisis
2025-07-08 12:50:22

⏱️ Duración del Análisis
1m 37.9s

🤖 Modelo de IA
google/gemini-2.0-flash-exp:free

📊 Archivos Analizados
5 total (4 Java + 1 POM)

🚦 Quality Gate: FAILED

❌ Se requieren mejoras antes de producción

5.0	1m 37.9s	5	21	23
PUNTUACIÓN GENERAL	TIEMPO DE ANÁLISIS	VULNERABILIDADES CRÍTICAS	TOTAL VULNERABILIDADES	PROBLEMAS DE CALIDAD

5.2
PUNTUACIÓN SEGURIDAD

📁 Análisis por Archivo

📄 ../demo/src/main/java/com/demo/DemoApplication.java — JAVA

🔒 Análisis de Seguridad

Lack of Security Controls — LOW

📍 **Ubicación:** 1-11

📝 **Descripción:** The provided code snippet only contains the basic Spring Boot application setup. It lacks any explicit security controls, such as authentication, authorization, input validation, or output encoding. While Spring Boot provides some default security features, they are not enabled or configured in this basic setup. This means the application is vulnerable to various attacks if further code is added without proper security considerations.

💥 **Impacto:** The application is vulnerable to various attacks if further code is added without proper security considerations. This could lead to data breaches, unauthorized access, and other security incidents.

💡 **Recomendación:** Implement security measures such as authentication, authorization, input validation, output encoding, and protection against common web vulnerabilities. Consider using Spring Security for authentication and authorization. Implement proper input validation to prevent injection attacks. Use output encoding to prevent XSS attacks.

🔧 **Solución:**

```
// Add Spring Security dependency to pom.xml or build.gradle
// Implement authentication and authorization using Spring Security
// Implement input validation using annotations or custom validators
// Implement output encoding using libraries like OWASP Java Encoder
```

⚡ Análisis de Calidad

Lack of Comments

LOW

📍 **Línea:** 1

📝 **Descripción:** The class and method lack Javadoc comments explaining their purpose and usage. While this is a simple application entry point, adding comments improves readability and maintainability, especially for larger projects.

🏷️ **Categoría:** Documentation

⏱️ **Esfuerzo:** 5 min

💡 **Recomendación:** Add Javadoc comments to the class and main method.

🔧 **Solución:**

```
/**
 * Entry point for the Demo Application.
 */
@SpringBootApplication
public class DemoApplication {
        /**
         * Main method to start the Spring Boot application.
         * @param args Command line arguments.
         */
        public static void main(String[] args) {
                SpringApplication.run(DemoApplication.class, args);
        }
}
```

Missing Exception Handling

LOW

📍 **Línea:** 7

📝 **Descripción:** While SpringApplication.run handles exceptions internally, for more complex applications, explicit exception handling around the application startup might be beneficial for logging or custom error handling.

🏷️ **Categoría:** Reliability

⏱️ **Esfuerzo:** 10 min

💡 **Recomendación:** Consider adding a try-catch block around SpringApplication.run for custom exception handling, especially if the application has specific startup requirements.

🔧 **Solución:**

```
try {
                SpringApplication.run(DemoApplication.class, args);
        } catch (Exception e) {
                System.err.println("Application failed to start: " + e.getMessage());
                // Log the exception properly using a logger
                System.exit(1); // Exit with an error code
        }
```

📄 ../demo/src/main/java/com/demo/controllers/WelcomeController.java

JAVA

🔒 Análisis de Seguridad

SQL Injection

HIGH

📍 **Ubicación:** 40

📝 **Descripción:** La consulta SQL se construye concatenando directamente la entrada del usuario (`username`) sin sanitizarla. Esto permite a un atacante inyectar código SQL malicioso.

💥 **Impacto:** Acceso no autorizado a la base de datos, modificación o eliminación de datos, e incluso ejecución de comandos en el servidor.

💡 **Recomendación:** Utilizar consultas preparadas (PreparedStatement) para parametrizar la consulta SQL y evitar la inyección.

🔧 **Solución:**

```
```java
Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/demo", "root", DB_PASSWORD);
PreparedStatement stmt = conn.prepareStatement("SELECT * FROM users WHERE username = ?");
stmt.setString(1, username);
ResultSet rs = stmt.executeQuery();
```
```

Cross-Site Scripting (XSS)

MEDIUM

📍 **Ubicación:** 24

📝 **Descripción:** El parámetro `name` se incluye directamente en la respuesta HTML sin codificación. Esto permite a un atacante inyectar código JavaScript malicioso que se ejecutará en el navegador de otros usuarios.

💥 **Impacto:** Robo de cookies, redirección a sitios web maliciosos, defacement del sitio web.

💡 **Recomendación:** Codificar la salida HTML para evitar la ejecución de código JavaScript no deseado. Utilizar la librería `org.springframework.web.util.HtmlUtils` para escapar los caracteres especiales.

🔧 **Solución:**

```
```java
import org.springframework.web.util.HtmlUtils;

@GetMapping("/api/welcome")
public WelcomeDTO welcome(@RequestParam(value = "name", defaultValue = "...") String name) {
    return new WelcomeDTO("Hola, bienvenido " + HtmlUtils.htmlEscape(name) + ", esto es un demo");
}
```
```

Remote Code Execution (RCE)

CRITICAL

📍 **Ubicación:** 29

📝 **Descripción:** El código proporcionado por el usuario se evalúa directamente utilizando `ScriptEngine`. Esto permite a un atacante ejecutar código arbitrario en el servidor.

💥 **Impacto:** Control total del servidor, compromiso de datos, denegación de servicio.

💡 **Recomendación:** Evitar la ejecución de código proporcionado por el usuario. Si es absolutamente necesario, utilizar un entorno de ejecución aislado (sandbox) con permisos muy restringidos.

🔧 **Solución:**

```
Eliminar la funcionalidad o implementar un sandbox seguro para la ejecución del script.
```

Path Traversal

HIGH

📍 **Ubicación:** 34

📝 **Descripción:** El parámetro `filePath` se utiliza directamente para leer un archivo sin validación. Esto permite a un atacante acceder a archivos fuera del directorio esperado.

💥 **Impacto:** Acceso no autorizado a archivos sensibles, divulgación de información.

💡 **Recomendación:** Validar y sanitizar la entrada `filePath` para asegurar que solo se acceda a archivos dentro de un directorio seguro. Utilizar una lista blanca de archivos permitidos o un prefijo de ruta seguro.

🔧 **Solución:**

```
```java
@GetMapping("/api/read-file")
```
```

```
public String readFile(@RequestParam String filePath) throws Exception {
    Path safePath = Paths.get("/path/to/safe/directory").resolve(filePath).normalize();
    if (!safePath.startsWith(Paths.get("/path/to/safe/directory"))) {
        return "Acceso denegado";
    }
    return Files.readString(safePath);
}
```

Command Injection

HIGH

📍 **Ubicación:** 48

📝 **Descripción:** El parámetro `host` se concatena directamente en un comando del sistema sin sanitizarlo. Esto permite a un atacante ejecutar comandos arbitrarios en el servidor.

💥 **Impacto:** Control total del servidor, compromiso de datos, denegación de servicio.

💡 **Recomendación:** Utilizar una API segura para ejecutar comandos del sistema o validar y sanitizar la entrada `host` para evitar la inyección de comandos.

🔧 **Solución:**

```
```java
@PostMapping("/api/ping")
public String ping(@RequestBody String host) throws Exception {
    if (!isValidHost(host)) {
        return "Host inválido";
    }
    ProcessBuilder pb = new ProcessBuilder("ping", "-c", "1", host);
    Process p = pb.start();
    BufferedReader reader = new BufferedReader(new InputStreamReader(p.getInputStream()));
    StringBuilder out = new StringBuilder();
    String line;
    while ((line = reader.readLine()) != null) out.append(line).append("\n");
    return out.toString();
}

private boolean isValidHost(String host) {
    // Implementar validación robusta del host aquí
    // Por ejemplo, verificar que solo contenga caracteres alfanuméricos y puntos
    return host.matches("^[a-zA-Z0-9.]*$");
}
```
```

Insecure Deserialization

HIGH

📍 **Ubicación:** 56

📝 **Descripción:** Se deserializa un objeto a partir de una cadena Base64 proporcionada por el usuario. Esto permite a un atacante ejecutar código arbitrario en el servidor mediante la creación de un objeto malicioso.

💥 **Impacto:** Ejecución remota de código, denegación de servicio.

💡 **Recomendación:** Evitar la deserialización de datos proporcionados por el usuario. Si es absolutamente necesario, utilizar un formato de serialización seguro como JSON o implementar un filtro de deserialización.

🔧 **Solución:**

```
Eliminar la funcionalidad o utilizar un formato de serialización seguro como JSON.
```

Information Exposure through Logs

MEDIUM

📍 **Ubicación:** 62

📝 **Descripción:** Se registran las credenciales del usuario (nombre de usuario y contraseña) en los logs. Esto expone información sensible que puede ser utilizada por un atacante.

💥 **Impacto:** Compromiso de credenciales, acceso no autorizado a cuentas.

💡 **Recomendación:** Evitar registrar información sensible en los logs. Registrar solo información relevante para la depuración y el seguimiento de errores.

🔧 **Solución:**

```java
logger.info("Intento de login con usuario: " + user + "");
```

Use of Weak Cryptographic Hash

MEDIUM

📍 **Ubicación:** 68

📝 **Descripción:** Se utiliza el algoritmo MD5 para generar un hash de los datos. MD5 es un algoritmo de hash débil que es vulnerable a colisiones y no debe utilizarse para fines de seguridad.

💥 **Impacto:** Compromiso de la integridad de los datos, suplantación de identidad.

💡 **Recomendación:** Utilizar un algoritmo de hash seguro como SHA-256 o SHA-3.

🔧 **Solución:**

```java
MessageDigest md = MessageDigest.getInstance("SHA-256");
```

Use of Hardcoded Cryptographic Key

CRITICAL

📍 **Ubicación:** 74

📝 **Descripción:** Se utiliza una clave secreta codificada en el código para el cifrado AES. Esto permite a un atacante descifrar los datos cifrados.

💥 **Impacto:** Compromiso de la confidencialidad de los datos.

💡 **Recomendación:** No codificar claves secretas en el código. Utilizar un almacén de claves seguro o un servicio de gestión de claves para almacenar y gestionar las claves secretas. Además, utilizar un vector de inicialización (IV) aleatorio para cada operación de cifrado y un modo de operación autenticado como GCM.

🔧 **Solución:**

```java
// Generar una clave aleatoria y almacenarla de forma segura
KeyGenerator keyGen = KeyGenerator.getInstance("AES");
keyGen.init(256); // Longitud de la clave
SecretKey secretKey = keyGen.generateKey();

// Generar un IV aleatorio
byte[] iv = new byte[16];
SecureRandom random = new SecureRandom();
random.nextBytes(iv);
IvParameterSpec ivParameterSpec = new IvParameterSpec(iv);

Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");
cipher.init(Cipher.ENCRYPT_MODE, secretKey, ivParameterSpec);
byte[] cipherText = cipher.doFinal(text.getBytes(StandardCharsets.UTF_8));

// Codificar el IV y el texto cifrado en Base64 para su almacenamiento o transmisión
ByteBuffer buffer = ByteBuffer.allocate(iv.length + cipherText.length);
buffer.put(iv);
buffer.put(cipherText);
byte[] combined = buffer.array();
String encryptedText = Base64.getEncoder().encodeToString(combined);

return encryptedText;
```

Information Exposure through Error Message

LOW

📍 **Ubicación:** 82

📝 **Descripción:** Se devuelve la traza de la pila de la excepción al usuario. Esto expone información sensible sobre la implementación interna de la aplicación.

💥 **Impacto:** Divulgación de información sensible, facilitación de ataques.

💡 **Recomendación:** No devolver la traza de la pila de la excepción al usuario. Devolver un mensaje de error genérico o un código de error específico.

🔧 **Solución:**

```
```java
} catch (Exception e) {
    return "Error interno del servidor";
}
```
```

Hardcoded Password

CRITICAL

📍 **Ubicación:** 15

📝 **Descripción:** La contraseña de la base de datos está codificada directamente en el código. Esto permite a un atacante acceder a la base de datos si obtiene acceso al código fuente.

💥 **Impacto:** Acceso no autorizado a la base de datos, modificación o eliminación de datos.

💡 **Recomendación:** No codificar contraseñas en el código. Utilizar variables de entorno, archivos de configuración seguros o un servicio de gestión de secretos para almacenar y gestionar las contraseñas.

🔧 **Solución:**

```
Eliminar la contraseña codificada y utilizar una variable de entorno o un archivo de configuración seguro.
```

Hardcoded API Key

CRITICAL

📍 **Ubicación:** 16

📝 **Descripción:** La clave de la API está codificada directamente en el código. Esto permite a un atacante utilizar la API con privilegios no autorizados si obtiene acceso al código fuente.

💥 **Impacto:** Uso no autorizado de la API, acceso a datos sensibles.

💡 **Recomendación:** No codificar claves de API en el código. Utilizar variables de entorno, archivos de configuración seguros o un servicio de gestión de secretos para almacenar y gestionar las claves de API.

🔧 **Solución:**

```
Eliminar la clave de API codificada y utilizar una variable de entorno o un archivo de configuración seguro.
```

Logging of Sensitive Data

MEDIUM

📍 **Ubicación:** 13

📝 **Descripción:** El logger se utiliza para registrar información sensible sin la debida precaución. Esto puede exponer información confidencial a personas no autorizadas.

💥 **Impacto:** Divulgación de información sensible.

💡 **Recomendación:** Evitar registrar información sensible en los logs. Si es necesario registrar información sensible, utilizar un logger seguro y configurar el acceso a los logs de forma adecuada.

🔧 **Solución:**

```
Revisar el uso del logger y evitar registrar información sensible.
```

⚡ Análisis de Calidad

Hardcoded Password

CRITICAL

📍 **Línea:** 16

📝 **Descripción:** The database password 'SuperSecreta123!' is hardcoded in the source code. This is a major security vulnerability.

🏷️ **Categoría:** Reliability

🏷️ **Categoría:** Reliability

⏱️ **Esfuerzo:** 60 min

💡 **Recomendación:** Store the password securely, such as in environment variables or a dedicated secrets management system.

🔧 **Solución:**

```
Eliminar la constante DB_PASSWORD y obtener el valor de una variable de entorno o un sistema de gestión de secretos.
```

Hardcoded API Key

CRITICAL

📍 **Línea:** 17

📝 **Descripción:** The API key 'ABC123-TOKEN-INSEGURO' is hardcoded in the source code. This is a major security vulnerability.

🏷️ **Categoría:** Reliability

⏱️ **Esfuerzo:** 60 min

💡 **Recomendación:** Store the API key securely, such as in environment variables or a dedicated secrets management system.

🔧 **Solución:**

```
Eliminar la constante API_KEY y obtener el valor de una variable de entorno o un sistema de gestión de secretos.
```

Cross-Site Scripting (XSS)

CRITICAL

📍 **Línea:** 21

📝 **Descripción:** The `welcome` method is vulnerable to XSS because it directly includes user-provided input (`name`) in the response without proper sanitization or encoding.

🏷️ **Categoría:** Reliability

⏱️ **Esfuerzo:** 120 min

💡 **Recomendación:** Sanitize or encode the user input before including it in the response. Use a templating engine that automatically escapes output.

🔧 **Solución:**

```
Utilizar un motor de plantillas que escape automáticamente la salida o escapar manualmente la entrada del usuario antes de incluirla en la respuesta.
```

Remote Code Execution (RCE)

CRITICAL

📍 **Línea:** 27

📝 **Descripción:** The `executeCode` method allows arbitrary code execution by using `ScriptEngine.eval` with user-provided input. This is a severe security risk.

🏷️ **Categoría:** Reliability

⏱️ **Esfuerzo:** 240 min

💡 **Recomendación:** Avoid using `ScriptEngine.eval` with user-provided input. If code execution is necessary, use a sandboxed environment with strict limitations.

🔧 **Solución:**

```
Eliminar la funcionalidad de ejecución de código arbitrario o implementar un entorno de pruebas con limitaciones estrictas.
```

Path Traversal

CRITICAL

📍 **Línea:** 33

📝 **Descripción:** The `readFile` method is vulnerable to path traversal because it directly uses user-provided input (`filePath`) to read files without proper validation. An attacker could read arbitrary files on the server.

🏷️ **Categoría:** Reliability

⏱️ **Esfuerzo:** 120 min

💡 **Recomendación:** Validate the user-provided file path to ensure it is within an allowed directory and does not contain path traversal sequences (e.g., '..').

🔧 **Solución:**

```
Validar la ruta del archivo proporcionada por el usuario para asegurarse de que esté dentro de un directorio permitido y no contenga secuencias de recorrido de ruta (por ejemplo, '..').
```

SQL Injection

CRITICAL

📍 **Línea:** 39

📝 **Descripción:** The `getUserInfo` method is vulnerable to SQL injection because it directly includes user-provided input (`username`) in the SQL query without proper sanitization or parameterized queries.

🏷️ **Categoría:** Reliability

⏱️ **Esfuerzo:** 180 min

💡 **Recomendación:** Use parameterized queries or prepared statements to prevent SQL injection.

🔧 **Solución:**

```
Utilizar consultas parametrizadas o sentencias preparadas para prevenir la inyección SQL.
```

Command Injection

CRITICAL

📍 **Línea:** 48

📝 **Descripción:** The `ping` method is vulnerable to command injection because it directly includes user-provided input (`host`) in the command executed by `Runtime.getRuntime().exec`. An attacker could execute arbitrary commands on the server.

🏷️ **Categoría:** Reliability

⏱️ **Esfuerzo:** 240 min

💡 **Recomendación:** Avoid using `Runtime.getRuntime().exec` with user-provided input. If command execution is necessary, use a safe API or library that properly escapes or validates the input.

🔧 **Solución:**

```
Evitar el uso de `Runtime.getRuntime().exec` con la entrada proporcionada por el usuario. Si la ejecución del comando es necesaria, utilizar una API o biblioteca segura que escape o valide correctamente la entrada.
```

Insecure Deserialization

CRITICAL

📍 **Línea:** 56

📝 **Descripción:** The `deserialize` method is vulnerable to insecure deserialization because it deserializes user-provided data without proper validation. An attacker could execute arbitrary code by providing a malicious serialized object.

🏷️ **Categoría:** Reliability

⏱️ **Esfuerzo:** 240 min

💡 **Recomendación:** Avoid deserializing user-provided data. If deserialization is necessary, use a safe serialization format (e.g., JSON) or implement strict validation of the deserialized object.

🔧 **Solución:**

```
Evitar la deserialización de datos proporcionados por el usuario. Si la deserialización es necesaria, utilizar un formato de serialización seguro (por ejemplo, JSON) o implementar una validación estricta del objeto deserializado.
```

Sensitive Data Logging

HIGH

📍 **Línea:** 62

📝 **Descripción:** The `login` method logs the username and password. This is a security risk because sensitive information is exposed in the logs.

🏷️ **Categoría:** Reliability

⏱️ **Esfuerzo:** 60 min

💡 **Recomendación:** Avoid logging sensitive information. If logging is necessary, use a secure logging mechanism that masks or encrypts the sensitive data.

🔧 **Solución:**

```
No registrar información confidencial. Si el registro es necesario, utilizar un mecanismo de registro seguro que enmascare o cifre los datos confidenciales.
```

Weak Hashing Algorithm

HIGH

📍 **Línea:** 68

📝 **Descripción:** The `hash` method uses the MD5 hashing algorithm, which is considered weak and vulnerable to collisions. This is a security risk.

🏷️ **Categoría:** Reliability

⏱️ **Esfuerzo:** 60 min

💡 **Recomendación:** Use a stronger hashing algorithm, such as SHA-256 or SHA-3.

🔧 **Solución:**

```
Reemplazar MD5 con un algoritmo de hash más fuerte, como SHA-256 o SHA-3.
```

Weak Encryption Algorithm

HIGH

📍 **Línea:** 74

📝 **Descripción:** The `encrypt` method uses AES encryption with a hardcoded key and without an initialization vector (IV) or GCM. This is a security risk.

🏷️ **Categoría:** Reliability

⏱️ **Esfuerzo:** 120 min

💡 **Recomendación:** Use a strong encryption algorithm with a randomly generated key and IV or GCM. Store the key securely.

🔧 **Solución:**

```
Utilizar un algoritmo de cifrado fuerte con una clave generada aleatoriamente y un IV o GCM. Almacenar la clave de forma segura.
```

Stacktrace Exposure

MEDIUM

📍 **Línea:** 83

📝 **Descripción:** The `error` method returns the stack trace of the exception to the client. This can expose sensitive information about the application's internal workings.

🏷️ **Categoría:** Reliability

⏱️ **Esfuerzo:** 30 min

💡 **Recomendación:** Return a generic error message to the client and log the exception details on the server.

🔧 **Solución:**

```
Devolver un mensaje de error genérico al cliente y registrar los detalles de la excepción en el servidor.
```

Unnecessary Imports

LOW

📍 **Línea:** 3-9

📝 **Descripción:** The code imports several classes that are not used, such as `javax.crypto.Cipher`, `javax.crypto.spec.SecretKeySpec`, `javax.script.ScriptEngine`, `javax.script.ScriptEngineManager`, `java.io.*`, `java.nio.file.*`, `java.security.MessageDigest`, `java.sql.*`.

🏷️ **Categoría:** Maintainability

⏱️ **Esfuerzo:** 15 min

💡 **Recomendación:** Remove unused imports to improve code readability and reduce dependencies.

🔧 **Solución:**

```
Eliminar las importaciones no utilizadas.
```

📄 **../demo/src/main/java/com/demo/models/WelcomeDTO.java** JAVA

🔒 Análisis de Seguridad

Insufficient Input Validation

LOW

📍 **Ubicación:** 4, 8, 12

📝 **Descripción:** The `WelcomeDTO` class lacks input validation for the `message` field. While this class itself doesn't directly interact with user input, it's a data transfer object (DTO) and the `message` field could potentially be populated with data from an untrusted source elsewhere in the application. Without validation, this data could contain malicious content, leading to vulnerabilities like XSS if the `message` is later displayed in a web page without proper encoding.

💥 **Impacto:** Potential XSS vulnerability if the `message` is displayed in a web page without proper encoding. Denial of service if excessively long strings are used.

💡 **Recomendación:** Implement input validation and sanitization for the `message` field. Consider using a library like OWASP Java HTML Sanitizer to prevent XSS. If the message is intended to be plain text, ensure it only contains allowed characters.

🔧 **Solución:**

```
```java
package com.demo.models;

import org.owasp.html.PolicyFactory;
import org.owasp.html.Sanitizers;

public class WelcomeDTO {
    private String message;
    private static final PolicyFactory sanitizer = Sanitizers.FORMATTING.and(Sanitizers.LINKS);

    public WelcomeDTO(String message) {
        this.message = sanitizeMessage(message);
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = sanitizeMessage(message);
    }

    private String sanitizeMessage(String message) {
        if (message == null) {
            return ""; // Or throw an exception, depending on requirements
        }
        return sanitizer.sanitize(message);
    }
}
```
```

⚡ Análisis de Calidad

Code Smell: Data Class

LOW

📍 **Línea:** 1

📝 **Descripción:** The `WelcomeDTO` class is a simple data holder with only a field and getter/setter methods. While not inherently bad, it might indicate an Anemic Domain Model if used extensively without behavior.

🏷️ **Categoría:** Maintainability

⏱️ **Esfuerzo:** 5 min

💡 **Recomendación:** Consider if this class should have any behavior related to the message it holds. If it's purely a data transfer object, it's acceptable, but be mindful of potential Anemic Domain Model issues in the larger context.

🔧 **Solución:**

```
N/A - Conceptual design consideration
```

Lack of Immutability

LOW

📍 **Línea:** 4

📝 **Descripción:** The `message` field is mutable due to the presence of a setter method. For simple DTOs, immutability can improve thread safety and predictability.

🏷️ **Categoría:** Reliability

⏱️ **Esfuerzo:** 5 min

💡 **Recomendación:** If the message should not be changed after creation, remove the `setMessage` method and make the `message` field final.

🔧 **Solución:**

```
```java
public class WelcomeDTO {
    private final String message;
    public WelcomeDTO(String message) {
        this.message = message;
    }

    public String getMessage() {
        return message;
    }

    // Remove setMessage method
}
```
```

Missing Javadoc

LOW

📍 **Línea:** 1

📝 **Descripción:** The class and its methods lack Javadoc comments. This reduces readability and maintainability.

🏷️ **Categoría:** Documentation

⏱️ **Esfuerzo:** 10 min

💡 **Recomendación:** Add Javadoc comments to the class and its methods, explaining their purpose and usage.

🔧 **Solución:**

```
```java
/**
 * Data Transfer Object for welcome messages.
 */
public class WelcomeDTO {
    /**
     * The welcome message.
     */
    private String message;

    /**
     * Constructs a new WelcomeDTO with the given message.
     * @param message The welcome message.
     */
    public WelcomeDTO(String message) {
        this.message = message;
    }
```
```

```
    /**
     * Returns the welcome message.
     * @return The welcome message.
     */
    public String getMessage() {
        return message;
    }

    /**
     * Sets the welcome message.
     * @param message The welcome message.
     */
    public void setMessage(String message) {
        this.message = message;
    }
}
```

📄 ../demo/src/test/java/com/demo/controller/WelcomeControllerTests.java

JAVA

🔒 Análisis de Seguridad

Insufficient Input Validation — LOW

📍 **Ubicación:** 24

📝 **Descripción:** The `welcomeEndpointReturnsCustomMessage` test case uses a parameter 'name' without any validation. While this is a test, it highlights a potential vulnerability in the actual `WelcomeController` if it doesn't properly sanitize or validate the 'name' parameter. If the `WelcomeController` directly uses this parameter in a response without proper encoding, it could lead to XSS.

💥 **Impacto:** Low in this test case, but potentially Medium in the actual controller if the input is not validated, leading to XSS.

💡 **Recomendación:** Ensure that the `WelcomeController` properly validates and sanitizes the 'name' parameter to prevent XSS or other injection attacks. Use appropriate encoding techniques when displaying the 'name' parameter in the response.

🔧 **Solución:**

````
In the WelcomeController:
```java
import org.springframework.web.util.HtmlUtils;

// ...
String sanitizedName = HtmlUtils.htmlEscape(name); // Escape HTML entities
String message = "Hola, bienvenido " + sanitizedName + ", esto es un demo";
// ...
```
````

⚡ Análisis de Calidad

Duplication of Code — MEDIUM

📍 **Línea:** 25, 33

📝 **Descripción:** The structure of the two test methods `welcomeEndpointReturnsDefaultMessage` and `welcomeEndpointReturnsCustomMessage` is highly similar, leading to code duplication. The `mockMvc.perform`, `status().isOk()`, and `content().contentType()` calls are repeated.

🏷️ **Categoría:** Maintainability

⏱️ **Esfuerzo:** 30 min

💡 **Recomendación:** Refactor the common parts of the tests into a helper method to reduce duplication and improve readability and maintainability.

🔧 **Solución:**

```java
private void performWelcomeTest(String nameParam, String expectedMessage) throws Exception {
    MockHttpServletRequestBuilder requestBuilder = get("/api/welcome");
    if (nameParam != null) {
        requestBuilder = requestBuilder.param("name", nameParam);
    }

    mockMvc.perform(requestBuilder)
           .andExpect(status().isOk())
           .andExpect(content().contentType("application/json"))
           .andExpect(jsonPath("$.message").value(expectedMessage));
}

@Test
public void welcomeEndpointReturnsDefaultMessage() throws Exception {
    performWelcomeTest(null, "Hola, bienvenido ..., esto es un demo");
}

@Test
public void welcomeEndpointReturnsCustomMessage() throws Exception {
    performWelcomeTest("Juan", "Hola, bienvenido Juan, esto es un demo");
}
```

Lack of Assertions

LOW

📍 **Línea:** 26, 34

📝 **Descripción:** While the tests check the HTTP status and content type, they could benefit from additional assertions to ensure the controller behaves as expected under different circumstances. For example, checking for specific headers or error conditions.

🏷️ **Categoría:** Reliability

⏱️ **Esfuerzo:** 15 min

💡 **Recomendación:** Add more specific assertions to cover different scenarios and edge cases. Consider adding assertions for headers, error messages, or other relevant aspects of the response.

🔧 **Solución:**

```java
// Example: Adding a header assertion
.andExpect(header().string("Content-Language", "es"));
```

Magic String

LOW

📍 **Línea:** 27, 35

📝 **Descripción:** The expected message strings are hardcoded within the tests. This makes the tests brittle and harder to maintain if the message changes in the controller.

🏷️ **Categoría:** Maintainability

⏱️ **Esfuerzo:** 20 min

💡 **Recomendación:** Extract the expected message strings into constants or use a more flexible approach to define the expected values, such as reading them from a configuration file or using a dedicated test data provider.

🔧 **Solución:**

```java
private static final String DEFAULT_MESSAGE = "Hola, bienvenido ..., esto es un demo";
private static final String CUSTOM_MESSAGE = "Hola, bienvenido Juan, esto es un demo";

@Test
public void welcomeEndpointReturnsDefaultMessage() throws Exception {
    mockMvc.perform(get("/api/welcome"))
           .andExpect(status().isOk())
           .andExpect(content().contentType("application/json"))
           .andExpect(jsonPath("$.message").value(DEFAULT_MESSAGE));
}
```

```
@Test
public void welcomeEndpointReturnsCustomMessage() throws Exception {
    mockMvc.perform(get("/api/welcome").param("name", "Juan"))
            .andExpect(status().isOk())
            .andExpect(content().contentType("application/json"))
            .andExpect(jsonPath("$.message").value(CUSTOM_MESSAGE));
}
```

Lack of Test Data Variation

LOW

📍 **Línea:** N/A

📝 **Descripción:** The tests only cover a single positive scenario for each endpoint. There's no testing of edge cases, invalid input, or error conditions.

🏷️ **Categoría:** Reliability

⏱️ **Esfuerzo:** 45 min

💡 **Recomendación:** Add more test cases to cover different scenarios, including invalid input, missing parameters, and potential error conditions. Consider using parameterized tests to easily test multiple input values.

🔧 **Solución:**

```java
// Example: Adding a test for missing parameter
@Test
public void welcomeEndpointReturnsErrorForMissingName() throws Exception {
    mockMvc.perform(get("/api/welcome").param("name", ""))
            .andExpect(status().isOk())
            .andExpect(content().contentType("application/json"))
            .andExpect(jsonPath("$.message").value("Hola, bienvenido , esto es un demo"));
}
```

Missing Javadoc

LOW

📍 **Línea:** 13

📝 **Descripción:** The test class and methods lack Javadoc comments, making it harder to understand their purpose and functionality.

🏷️ **Categoría:** Documentation

⏱️ **Esfuerzo:** 15 min

💡 **Recomendación:** Add Javadoc comments to the test class and methods to explain their purpose and how they work.

🔧 **Solución:**

```java
/**
 * Tests for the {@link WelcomeController}.
 */
@WebMvcTest(WelcomeController.class)
public class WelcomeControllerTests {

    /**
     * Injected MockMvc instance for simulating HTTP requests.
     */
    @Autowired
    private MockMvc mockMvc;

    /**
     * Tests that the welcome endpoint returns the default message.
     * @throws Exception if an error occurs during the test.
     */
    @Test
    public void welcomeEndpointReturnsDefaultMessage() throws Exception {
        ...
    }

    ...
}
```

📦 ../demo/pom.xml

POM

🔒 Análisis de Seguridad - Maven POM

Outdated Dependency

CVE-2019-14892, CVE-2019-14540, CVE-2018-11307, etc. (multiple)

HIGH

📍 **Ubicación:** 31

📝 **Descripción:** The `jackson-databind` dependency is using version 2.9.8, which is known to have multiple critical security vulnerabilities (CVEs). This version is severely outdated.

💥 **Impacto:** Remote code execution, denial of service, information disclosure. An attacker could potentially exploit these vulnerabilities to compromise the application and the server it runs on.

💡 **Recomendación:** Upgrade `jackson-databind` to the latest stable version (at least 2.16.0 or higher). Check for compatibility with other dependencies after upgrading.

📦 **Dependencia:** **com.fasterxml.jackson.core:jackson-databind**

🔧 **Solución:**

```
com.fasterxml.jackson.core
jackson-databind
2.16.1
```

Snapshot Dependency in Production

MEDIUM

📍 **Ubicación:** 10

📝 **Descripción:** The project version is set to `0.0.1-SNAPSHOT`. Snapshot versions are intended for development and should not be used in production deployments. Snapshot dependencies can change unpredictably, leading to inconsistent and potentially broken builds in production.

💥 **Impacto:** Unpredictable behavior in production, potential for broken builds, difficulty in reproducing issues.

💡 **Recomendación:** Remove the `-SNAPSHOT` suffix from the version before deploying to production. Use a proper release versioning scheme (e.g., 1.0.0, 1.0.1).

📦 **Dependencia:** **com:demo**

🔧 **Solución:**

```
0.0.1
```

Missing License Information

LOW

📍 **Ubicación:** 14

📝 **Descripción:** The `` section is present but empty. This makes it difficult for users to understand the terms under which they can use the software. While not a direct security vulnerability, it can lead to legal issues and discourage adoption.

💥 **Impacto:** Legal issues, discourages adoption.

💡 **Recomendación:** Add the appropriate license information to the `` section. Specify the license name, URL, and distribution type.

🔧 **Solución:**

```
Apache License, Version 2.0
http://www.apache.org/licenses/LICENSE-2.0.txt
```

```
repo
```

Missing SCM Information

LOW

📍 **Ubicación:** 17

📝 **Descripción:** The `` section is present but empty. This makes it difficult to track the source code repository and contribute to the project. While not a direct security vulnerability, it hinders collaboration and transparency.

💥 **Impacto:** Hindered collaboration, reduced transparency.

💡 **Recomendación:** Add the appropriate SCM information to the `` section. Specify the connection URL, developer connection URL, tag, and URL.

🔧 **Solución:**

```
scm:git:git://github.com/example/demo.git
scm:git:ssh://git@github.com/example/demo.git
HEAD
https://github.com/example/demo
```

Missing Version Control for Plugins

LOW

📍 **Ubicación:** 41

📝 **Descripción:** The `spring-boot-maven-plugin` is used without specifying a version. This can lead to unpredictable builds if the plugin is updated automatically by Maven. It's best practice to explicitly define the version of all plugins.

💥 **Impacto:** Unpredictable builds, potential for broken builds.

💡 **Recomendación:** Specify the version of the `spring-boot-maven-plugin` in the `` section.

📦 **Dependencia:** org.springframework.boot:spring-boot-maven-plugin

🔧 **Solución:**

```
org.springframework.boot
spring-boot-maven-plugin
3.4.5
```

📦 Dependencias Desactualizadas

com.fasterxml.jackson.core:jackson-databind

OUTDATED

📍 **Versión Actual:** 2.9.8

🆕 **Versión Recomendada:** 2.16.1

🔒 **Riesgo de Seguridad:** HIGH