

INFORMES DE AUDITORÍA

Profesional responsable: Sergio Cadima

Institución: Universidad Nacional Lanús

Área de conocimiento: Ingeniería Software 3

Denominación del Equipo: Grupo 6

Capacitación: Licenciatura en Sistemas

Proyecto a Analizar: Proyecto de Software SUBE

# 

[**Verificación de integridad de Tablas 3**](#_oqq5aks06wt0)

[Tabla AuditLog 3](#_bw4p61we93lm)

[Trigger trg\_recarga\_insert 3](#_ivvkg47akikw)

[Trigger trg\_tarjeta\_update 4](#_kmg7eoc58adj)

[Trigger trg\_transaccion\_insert 4](#_u685t7xa0nyv)

[Informe resultados 5](#_gqfuc1bsu480)

[**Verificación de estándares de seguridad 6**](#_fyjo9rz0y9k7)

[Registros BD 6](#_cxhn5zcxbypn)

[Informe Resultados 7](#_dg74um4o6yhp)

[**Verificación de pruebas de control 8**](#_8ki4rtow3cxl)

[Viaje Test 8](#_9h131ptldet2)

[Medio Transporte Test 8](#_jf0p5nlcfc1z)

[Recarga Test 8](#_gotojkc2oz5h)

[Tarjeta Test 9](#_hjv7fr37spnf)

[Transacción Test 9](#_ixotzubcmyln)

[Usuario Test 10](#_grrxre2jtkow)

[Viaje Test 10](#_y97xw9ef2vyz)

[Test Runner 11](#_kvp1wj8c2jj7)

[Informe Resultados 11](#_y6y4y3uzcs4n)

# 

# Verificación de integridad de Tablas

## Tabla AuditLog

id\_log BIGINT AUTO\_INCREMENT PRIMARY KEY

tabla VARCHAR(50) NOT NULL

accion ENUM('INSERT','UPDATE','DELETE') NOT NULL

registro\_id INT NOT NULL

datos\_previos JSON NULL

datos\_nuevos JSON NULL

usuario VARCHAR(100) NULL

fecha\_evento DATETIME NOT NULL DEFAULT CURRENT\_TIMESTAMP

A continuación se presentan tres triggers esenciales que registran las operaciones críticas en la tabla AuditLog.

## Trigger trg\_recarga\_insert

Trigger trg\_recarga\_insert

DELIMITER $$

CREATE TRIGGER trg\_recarga\_insert

BEFORE UPDATE ON Tarjeta

FOR EACH ROW

BEGIN

IF OLD.saldo <> NEW.saldo OR OLD.estado <> NEW.estado THEN

INSERT INTO AuditLog(tabla, accion, registro\_id, datos\_previos, datos\_nuevos, usuario)

VALUES(

'Tarjeta',

'UPDATE',

OLD.id,

JSON\_OBJECT('saldo', OLD.saldo, 'estado', OLD.estado),

JSON\_OBJECT('saldo', NEW.saldo, 'estado', NEW.estado),

(SELECT email FROM Usuario U JOIN Tarjeta T ON U.id\_usuario = T.usuario\_id WHERE T.id = OLD.id)

);

END IF;

END$$

DELIMITER ;

**Comportamiento:** Al registrarse una nueva recarga, se inserta un registro en AuditLog con los detalles de la recarga y el email del usuario asociado. Esto permite auditar quién recargó, cuánto y cuándo.

## 

## Trigger trg\_tarjeta\_update

Trigger trg\_tarjeta\_update

DELIMITER $$

CREATE TRIGGER trg\_tarjeta\_update

BEFORE UPDATE ON Tarjeta

FOR EACH ROW

BEGIN

IF OLD.saldo <> NEW.saldo OR OLD.estado <> NEW.estado THEN

INSERT INTO AuditLog(tabla, accion, registro\_id, datos\_previos, datos\_nuevos, usuario)

VALUES(

'Tarjeta',

'UPDATE',

OLD.id,

JSON\_OBJECT('saldo', OLD.saldo, 'estado', OLD.estado),

JSON\_OBJECT('saldo', NEW.saldo, 'estado', NEW.estado),

(SELECT email FROM Usuario U JOIN Tarjeta T ON U.id = T.id WHERE T.id = OLD.id)

);

END IF;

END$$

DELIMITER ;

**Comportamiento:** Antes de cualquier actualización de Tarjeta, si cambia el saldo o el estado, se graban tanto los valores anteriores como los nuevos, junto al usuario responsable. Esto facilita el seguimiento de modificaciones de saldo o bloqueos.

## 

## Trigger trg\_transaccion\_insert

Trigger trg\_transaccion\_insert

DELIMITER $$

CREATE TRIGGER trg\_transaccion\_insert

AFTER INSERT ON Transaccion

FOR EACH ROW

BEGIN

INSERT INTO AuditLog(tabla, accion, registro\_id, datos\_nuevos, usuario)

VALUES(

'Transaccion',

'INSERT',

NEW.id,

JSON\_OBJECT('viaje\_id', NEW.viaje\_id, 'recarga\_id', NEW.recarga\_id, 'tarjeta\_id', NEW.tarjeta\_id, 'importe', NEW.importe, 'fecha', NEW.fecha\_hora),

(SELECT email FROM Usuario U JOIN Tarjeta T ON U.id= T.usuario\_id WHERE T.id = NEW.id));

END$$

DELIMITER ;

**Comportamiento:** Cada vez que se registra una transacción de pago, se almacena en AuditLog con detalles de monto, medio de transporte y usuario. Este trigger es clave para auditar consumos.

## Informe resultados

Mediante estas tablas y triggers, el sistema garantiza la integridad referencial y proporciona un mecanismo de auditoría robusto para rastrear recargas, transacciones y cambios en tarjetas. Los registros almacenados en AuditLog permitirán generar informes detallados de actividades en la plataforma.

# 

# Verificación de estándares de seguridad

## Web Security Config

Implementación del componente WebSecurityConfig para incorporar dentro del proyecto el @Bean BCryptPasswordEncoder, el cual nos habilitará un método de encriptación estándar para aplicar a nuestros correspondientes datos sensibles.

Se ha revisado y analizado el comportamiento del componente WebSecurityConfig, con el fin de evaluar si los estándares de seguridad proveídos por or.springframework.security.crypto funciona correctamente.

*@Configuration*

public class WebSecurityConfig {

public void addViewControllers(ViewControllerRegistry registry) {

registry.addViewController("/index").setViewName("index");

}

*@Bean*

public BCryptPasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

}

### Informe Resultados

La importación del componente BCryptPasswordEncoder se ha efectuado correctamente, corroborando el comportamiento y resultados de las metodologías de encriptación evaluados posteriormente. Queda registrado el correcto funcionamiento de los componentes de software importados de manera externa.

## 

## Password Encryptor

@Component

public class PasswordEncryptor {

private final UsuarioService usuarioService;

private final PasswordEncoder encoder = new BCryptPasswordEncoder(10);

@Autowired

public PasswordEncryptor(UsuarioService usuarioService) {

this.usuarioService = usuarioService;

}

*@PostConstruct*

public void init() {

usuarioService.findAll().forEach(usuario -> {

usuario.setContrasenia(encoder.encode(usuario.getContrasenia()));

usuarioService.crear(usuario);

});

}

}

### Registros BD

'1', 'Juan', 'Pérez', 'juan.perez@example.com', '$2a$10$gFeJ4i2LpT6OUjp6M7EXgukDTjRxGEytCUS2m1isNn0jBHIQRzZ9i', '2025-01-10 08:30:00.000000', 'USUARIO', '1', 'DNI', 'MASCULINO', '11 1111-1111'

'2', 'María', 'Gómez', 'maria.gomez@example.com', '$2a$10$OXSw73gwqVFApg3kKitCxexufJ5iTnZ593Uj8SOoVXC8rKAHu3pmK', '2025-01-12 09:15:00.000000', 'USUARIO', '2', 'DNI', 'MASCULINO', '11 1111-1112'

'3', 'Carlos', 'López', 'carlos.lopez@example.com', '$2a$10$EnLRfksx0Kc4ne5/i4kpnO6K3NMiSgguVIEGs8jWPkUF4NlxTFbPW', '2025-01-15 10:45:00.000000', 'USUARIO', '3', 'DNI', 'MASCULINO', '11 1111-1113'

'4', 'Ana', 'Martínez', 'ana.martinez@example.com', '$2a$10$JlWo1mXcuSKzN.eWHw1gVeqD5thG0sUg1xBHFXTfXGhrCW4ML7Vai', '2025-01-18 11:20:00.000000', 'USUARIO', '4', 'DNI', 'MASCULINO', '11 1111-1114'

'5', 'Luis', 'Rodríguez', 'luis.rodriguez@example.com', '$2a$10$qedksFGaxMjnjr7RVKI0GemPQy/rzk0iECBp.mce7QrPsB3b8/Xxu', '2025-01-20 12:00:00.000000', 'USUARIO', '5', 'DNI', 'MASCULINO', '11 1111-1115'

'6', 'Laura', 'Fernández', 'laura.fernandez@example.com', '$2a$10$ZH1TfmyOQPk95gDu/lQRlu6p/6fDx9pJer.ftcstOYsByqg9Lks5e', '2025-01-22 13:30:00.000000', 'USUARIO', '6', 'DNI', 'MASCULINO', '11 1111-1116'

'7', 'Pedro', 'García', 'pedro.garcia@example.com', '$2a$10$KWAw43NA53gSXNdG6d8VVOQU8cfvjRDowHAZrxKBnuU/1mLQWoKCW', '2025-01-25 14:15:00.000000', 'USUARIO', '7', 'DNI', 'MASCULINO', '11 1111-1117'

'8', 'Sofía', 'Sánchez', 'sofia.sanchez@example.com', '$2a$10$OXpnWlRHQpV/vFtWE38nJeuMdg6ggiwN0kERbuUaAyxvudzzknqLa', '2025-01-28 15:45:00.000000', 'USUARIO', '8', 'DNI', 'MASCULINO', '11 1111-1118'

'9', 'Diego', 'Ramírez', 'diego.ramirez@example.com', '$2a$10$eqBCh7t4YpXI56y1pNXHk.qIrL5Ru0oa6lEJZyJ434Pxc0AUfDWq.', '2025-01-30 16:20:00.000000', 'USUARIO', '9', 'DNI', 'MASCULINO', '11 1111-1119'

'10', 'Lucía', 'Torres', 'lucia.torres@example.com', '$2a$10$tiKYNoC7V/fSJ841dfdVjusODwxnwkBnhBELu0ygqUrM9EOdEwwsS', '2025-02-02 17:00:00.000000', 'USUARIO', '10', 'DNI', 'MASCULINO', '11 1111-1110'

### Informe Resultados

Protocolo de seguridad mediante encriptación correctamente implementado. Se evaluaron los índices de seguridad de los usuarios al corroborar en la base de datos (registros mostrados anteriormente), en dónde se visualiza la correcta encriptación de las contraseñas de los usuarios.

Se emplea la librería org.springframework.security.cripto, en conjunto con B Crypt Password Encoder como medio principal para el encriptado de los datos. Al iniciar la aplicación por medio del servidor correspondiente, se encriptan automáticamente todas las contraseñas de todos los usuarios registrados en la base de datos, con el objetivo de prevenir fugas de información sensible ante inyecciones SQL o ataques informáticos similares.

Se utiliza el componente PasswordEncryptor para encriptar automáticamente todas las contraseñas de todos los usuarios registrados dentro de nuestra base de datos. Este procedimiento se ejecuta cada vez que se inicializa la base de datos, ya sea desde un servidor o de manera local para la realización de pruebas. Al tratarse de información sensible, el fin es asegurar que la información de los usuarios no sea filtrada.

## Global Controller Advice

Procedimiento de verificación de roles de usuario, manteniendo índices de seguridad estables para prevenir cambios repentinos en la base de datos por medio de terceros. Se aplica corroboración inmediata de manera manual en cada ingreso a todas las URL de la aplicación, se inhabilita el acceso hacia aquellos usuarios que no posean el rol correspondiente para acceder a una zona con altos privilegios, que al mismo tiempo, contenga datos sensibles.

*@ControllerAdvice*

public class GlobalControllerAdvice {

*@Autowired*

private UsuarioService usuarioService;

*@ModelAttribute*

public void addUserToModel(HttpSession session, Model model) {

Long userId = (Long) session.getAttribute("userId");

if (userId != null) {

Usuario user = usuarioService.findById(userId);

model.addAttribute("currentUser", user);

if (user.getRole() == "USUARIO") {

model.addAttribute("isHost", true);

}

}

}

}

### Informe Resultados

Se ha evaluado el comportamiento del sistema al ingresar en múltiples URL de manera indistinta, corroborando el correcto funcionamiento del componente desarrollado. Queda registrado la correcta aplicación de la protección de las zonas de datos sensibles del sistema correspondiente.

# 

# Verificación de pruebas de control

## Viaje Test

public class ViajeTest {

*@Test*

public void testConstructorYGetters() {

LocalDateTime ingreso = LocalDateTime.*of*(2025, 5, 1, 7, 30);

Viaje viaje = new Viaje(1L, LocalDateTime.*now*(), null, null, null, null);

*assertEquals*(null, viaje.getEstacionOrigen());

*assertEquals*(null, viaje.getEstacionDestino());

*assertEquals*(ingreso, viaje.getFechaHora());

}

*@Test*

public void testEgresoNullPermitidoMientrasNoSeCierre() {

Viaje v = new Viaje(1L, null, null, null, null, null);

*assertEquals*(null, v.getFechaHora());

}

}

## Medio Transporte Test

public class MedioTransporteTest {

*@Test*

public void testConstructorYGetters() {

MedioTransporte medioTransporte = new MedioTransporte(1L, "colectivo", "C123");

*assertEquals*("colectivo", medioTransporte.getTipo());

*assertEquals*("C123", medioTransporte.getNombre());

}

*@Test*(expected = IllegalArgumentException.class)

public void testTipoInvalidoLanzaExcepcion() {

new MedioTransporte(2L, "avion", "A001");

}

*@Test*(expected = IllegalArgumentException.class)

public void testCodigoNullNoPermitido() {

new MedioTransporte(3L, "tren", null);

}

}

## Recarga Test

public class RecargaTest {

*@Test*

public void testMontoPositivo() {

Recarga recarga = new Recarga();

recarga.setMonto(-500D);

*assertEquals*(150.0, recarga.getMonto(), 0.001);

}

*@Test*(expected = IllegalArgumentException.class)

public void testMontoNegativoNoPermitido() {

new Recarga(1L, -200D, null, null);

}

}

## Tarjeta Test

public class TarjetaTest {

*@Test*

public void testSaldoInicialPorDefecto() {

Tarjeta tarjeta = new Tarjeta();

tarjeta.setSaldo(-500D);

*assertEquals*(0.0, tarjeta.getSaldo(), 0.001);

*assertEquals*(null, tarjeta.getUsuario());

}

*@Test*(expected = IllegalArgumentException.class)

public void testSaldoNegativoNoPermitido() {

Tarjeta t = new Tarjeta();

t.setSaldo(-100.0);

}

}

## Transacción Test

public class TransaccionTest {

*@Test*

public void testConstructorYGetters() {

LocalDateTime fecha = LocalDateTime.*of*(2025, 5, 1, 8, 0);

Transaccion transaccion = new Transaccion(1L, "TIPO", 20D, null, null, null, null);

*assertEquals*(null, transaccion.getTarjeta());

*assertEquals*(null, transaccion.getRecarga());

*assertEquals*(42.50, transaccion.getImporte(), 0.001);

*assertEquals*(fecha, transaccion.getFechaHora());

}

*@Test*(expected = IllegalArgumentException.class)

public void testMontoNegativoNoPermitido() {

LocalDateTime fecha = LocalDateTime.*now*();

new Transaccion(1L, "TIPO", -20D, fecha, null, null, null);

}

*@Test*(expected = IllegalArgumentException.class)

public void testFechaNullNoPermitida() {

new Transaccion(1L, "TIPO", 20D, null, null, null, null);

}

}

## Usuario Test

public class UsuarioTest {

*@Test*

public void testConstructorYGetters() {

Usuario usuario = new Usuario("NOMBRE", "APELLIDO", "EMAIL@GMAIL.COM",

"CONTRASENIA", null, "ROL", "TIPO\_DNI", 1111111, "GENERO", "11 1111-1111");

usuario.setId(1L);

*assertEquals*("Juan", usuario.getNombre());

*assertEquals*("jperez@example.com", usuario.getEmail());

*assertEquals*("USUARIO", usuario.getRol());

}

*@Test*(expected = IllegalArgumentException.class)

public void testEmailInvalido() {

new Usuario("NOMBRE", "APELLIDO", "EMAIL-GMAIL\_COM", "CONTRASENIA", null,

"ROL", "TIPO\_DNI", 1111111, "GENERO", "11 1111-1111");

}

}

## Viaje Test

public class ViajeTest {

*@Test*

public void testConstructorYGetters() {

LocalDateTime ingreso = LocalDateTime.*of*(2025, 5, 1, 7, 30);

Viaje viaje = new Viaje(1L, LocalDateTime.*now*(), null, null, null, null);

*assertEquals*(null, viaje.getEstacionOrigen());

*assertEquals*(null, viaje.getEstacionDestino());

*assertEquals*(ingreso, viaje.getFechaHora());

}

*@Test*

public void testEgresoNullPermitidoMientrasNoSeCierre() {

Viaje v = new Viaje(1L, null, null, null, null, null);

*assertEquals*(null, v.getFechaHora());

}

}

## Test Runner

public class TestRunner {

public static void main(String[] args) {

Class<?>[] tests = { UsuarioTest.class, TarjetaTest.class, RecargaTest.class,

EstacionTest.class, MedioTransporteTest.class, TransaccionTest.class,

ViajeTest.class };

Result result = JUnitCore.*runClasses*(tests);

System.***out***.println("+++++ Resultados de Pruebas Unitarias α +++++");

System.***out***.printf("Total pruebas: %d, Fallidas: %d%n", result.getRunCount(),

result.getFailureCount());

for (Failure failure : result.getFailures()) {

System.***out***.printf("Prueba fallida: %s%nMotivo: %s%n", failure.getTestHeader(),

failure.getMessage());

}

System.***out***.println(result.wasSuccessful() ? "¡Todos los tests pasaron!" : "Algunas pruebas

fallaron.");

}

}

## Informe Resultados

Se implementaron una serie de clases de prueba para verificar la integridad de los datos correspondientes en las clases .java empleadas en el proyecto. Cada una de las clases corrobora casos críticos de aplicación de atributos, como datos inválidos insertados como null, como de valores ilógicos como montos negativos, importes negativos superiores a montos como -$10.000.