NOMBRE DE LA INSTITUCIÓN:
Tecnológico de Monterrey

Planeación:
Pruebas - Sitio WEB

MATERIA:
TC3002C.101 Ciberseguridad informática II

Miembros:
Axel Ariel Grande Ruiz A01611811
Carlos Eugenio Saldaña Tijerina A01285600
Humberto Jasso Silva A01771184
Isaac Hernández Pérez A01198674
Víctor Misael Escalante Alvarado A01741176

NOMBRE DEL PROFESOR:
Luis Alberto Terrazas
Jessica Izquierdo Alvarez

FECHA:
8 noviembre del 2025

Introducción

El presente documento interne mostrar un listado de las pruebas a realizar sobre la página MARCO definida en entregables anteriores. A continuación se enlistará cada prueba acorde a una de las siguientes verticales :

- Pruebas de frontend
- Pruebas de backend
- Pruebas de integridad y seguridad de acceso

Cada una de estas pruebas será descrita y se definirá la respuesta esperada por la página a desarrollar (aceptada o rechazada). En este caso se ha tomado un enfoque al rededor de las vulnerabilidades comunes encontradas en los entornos de desarrollo web y móvil de la iniciativa OWASP (se incluyen las vulnerabilidades móviles debido a que se planea generar una aplicación nativa en móvil como fue descrito en el entregable anterior), en base a estas vulnerabilidades descritas también se plantea Lina sección de manejo de riesgos en base a las pruebas y vulnerabilidades a cubrir.

Listado de pruebas a realizar en la página

Pruebas de Frontend (Web React y App React Native)

Identificador de la prueba	Descripción de la prueba	Resultados esperados
PF-01: Validación de campos de entrada en formulario de login	Verificar que el frontend implementa validaciones de formato, longitud mínima/máxima y caracteres permitidos en campos de usuario y contraseña antes de enviar datos al backend. Se debe validar restricción de caracteres especiales para prevenir XSS (OWASP A03: Injection).	El sistema rechaza entradas con formato inválido mostrando mensajes de error claros sin procesar la petición.
PF-02: Protección contra ataques de clickjacking	Evaluar que la aplicación web implementa headers X-Frame-Options o Content-Security-Policy frame-ancestors para prevenir que la página sea embebida en iframes maliciosos (OWASP A05: Security Misconfiguration).	La aplicación rechaza ser cargada en contextos de iframe no autorizados.
PF-03: Almacenamiento seguro de tokens en cliente	Verificar que tokens de sesión o JWT no se almacenan en localStorage sino en httpOnly cookies (web) o en Keychain/Keystore seguros (móvil), protegiendo contra robo mediante XSS (OWASP Mobile M1: Improper Credential Usage).	Los tokens no son accesibles mediante JavaScript y están cifrados en almacenamiento móvil.

PF-04: Implementación de rate limiting visual	Comprobar que después de múltiples intentos fallidos de login, el frontend muestra CAPTCHA o implementa delays progresivos antes de permitir nuevos intentos (OWASP A07: Authentication Failures).	El sistema implementa controles anti-automatización después de 5 intentos fallidos.
PF-05: Validación de certificados SSL/TLS en app móvil	Verificar que la aplicación React Native implementa certificate pinning o validación estricta de certificados SSL para prevenir ataques MITM (OWASP Mobile M3: Insecure Authentication).	La app rechaza conexiones con certificados inválidos o auto-firmados.

Pruebas de Backend

Identificador de la prueba	Descripción de la prueba	Resultados esperados
PB-01: Implementación de hash seguro de contraseñas	Verificar que el backend almacena contraseñas utilizando algoritmos seguros como bcrypt, Argon2 o PBKDF2 con salt único por usuario, nunca en texto plano o con hashes reversibles (OWASP A02: Cryptographic Failures).	Las contraseñas almacenadas en base de datos son irreversibles y resistentes a ataques de rainbow tables.
PB-02: Protección contra SQL Injection en queries de autenticación	Evaluar que todas las consultas de autenticación utilizan prepared statements o ORMs parametrizados, nunca concatenación de strings con input del usuario (OWASP A03: Injection).	El sistema rechaza payloads de SQLi como `' OR '1'='1` sin ejecutar código malicioso.

PB-03: Implementación de política de contraseñas robusta	Verificar que el backend valida requisitos mínimos de complejidad: longitud mínima 12 caracteres, combinación de mayúsculas, minúsculas, números y símbolos, rechazo de contraseñas comunes mediante lista de diccionario (OWASP A07: Authentication Failures).	El sistema rechaza contraseñas débiles como "Password123" durante registro o cambio.
PB-04: Validación de expiración y renovación de tokens	Comprobar que los tokens JWT o de sesión tienen tiempo de vida limitado (15-30 minutos para access tokens), implementan refresh tokens con rotación y se invalidan en logout (OWASP A01: Broken Access Control).	Tokens expirados son rechazados y se requiere reautenticación, tokens antiguos no pueden reutilizarse.
PB-05: Protección contra ataques de fuerza bruta	Verificar implementación de rate limiting a nivel servidor que bloquea temporalmente IPs después de múltiples intentos fallidos, con incremento exponencial de penalización (OWASP A07: Authentication Failures).	Después de 5 intentos fallidos en 5 minutos, la IP es bloqueada por 30 minutos.

Pruebas de Integridad, Autenticación y Acceso

Identificador de la prueba	Descripción de la prueba	Resultados esperados
PIA-01: Verificación de control de acceso basado en roles	Evaluar que el sistema implementa correctamente RBAC, verificando que usuarios autenticados solo acceden a recursos según su rol (visitante, miembro, administrador) sin posibilidad de escalación de privilegios (OWASP A01: Broken Access Control).	Usuario con rol "visitante" recibe HTTP 403 al intentar acceder a endpoints administrativos.
PIA-02: Validación de sesión en cada petición	Verificar que todas las rutas protegidas validan token de sesión en cada request, rechazando peticiones con tokens manipulados, expirados o inexistentes (OWASP A07: Authentication Failures)	Peticiones sin token válido reciben HTTP 401 Unauthorized sin exponer información sensible.
PIA-03: Protección contra Cross-Site Request Forgery (CSRF)	Verificar que formularios críticos implementan tokens CSRF únicos por sesión, validados en backend antes de procesar acciones sensibles como cambio de contraseña o actualización de datos personales (OWASP A01: Broken Access Control).	Peticiones sin token CSRF válido son rechazadas con HTTP 403.
PIA-05: Validación de integridad de tokens JWT	Evaluar que tokens JWT están firmados con algoritmo seguro (RS256 o HS256 con secret fuerte), el backend verifica firma en cada petición y rechaza tokens con claims modificados o algoritmo	Tokens con firma inválida o payload modificado son rechazados inmediatamente.

	"none" (OWASP A02: Cryptographic Failures).	
PIA-07: Auditoría y logging de intentos de autenticación	Comprobar que el sistema registra todos los intentos de login (exitosos y fallidos) con timestamp, IP origen, user agent y resultado, sin almacenar contraseñas en logs (OWASP A09: Security Logging Failures).	Los registros contienen información forense suficiente para análisis de incidentes sin exponer credenciales.

Remediaciones

Insufficient Cryptography (criptografía insuficiente)

A. Contraseñas: hashing correcto y migración sin fricción

Qué hacer: reemplazar cualquier hash débil (MD5/SHA-1/"casero") por un **KDF seguro**: ideal **Argon2id**; si no, **bcrypt** con *cost* 10–14. Cada contraseña debe llevar **sal aleatoria** (la añade el KDF) y opcionalmente un **pepper** (secreto del servidor guardado en tu gestor de secretos).

Cómo migrar sin romper logins:

1. Mantén el campo password_hash y agrega hash_version.
2. Al iniciar sesión, detecta la versión antigua: si el hash "viejo" coincide, **rehash** con Argon2/bcrypt y actualiza hash_version.
3. Nunca guardes contraseñas en texto plano ni sal/pepper junto al hash.

B. Cifrado de datos sensibles y gestión de claves

Qué hacer: si guardas **datos sensibles** (p. ej., datos personales que no sean contraseñas), usa **cifrado autenticado**: **AES-256-GCM** o **ChaCha20-Poly1305**. Cada registro/valor debe llevar su **IV/nonce único** y debes almacenar también el **auth tag** y la **versión del esquema** (para poder rotar algoritmos en el futuro).

Gestión de claves:

- La clave **no** va en el código ni en .env compartido. Úsala desde **Vault/KMS/Secrets Manager** y define **rotación** (p. ej., cada 90 días o tras incidente).
- Documenta: algoritmo, longitud, procedencia de la clave, rotaciones y versión.

Qué validas al final: si re-cifras el mismo valor dos veces, los ciphertexts son distintos (nonce único); al modificar 1 bit del ciphertext, la verificación GCM falla (integridad).

C. Protección en tránsito y aleatoriedad

Qué hacer:

- **TLS 1.2+ (ideal 1.3)** en toda la app, redirección firme a https:// y **HSTS**.
- Tokens/IDs/IVs generados con **CSPRNG** del lenguaje (no Math.random).
- Tokens con **expiración corta** y revocación (listas de deny o rotación de claves JWT si aplica).

Qué validas al final: devtools o un escáner confirman solo peticiones HTTPS; encabezados como Strict-Transport-Security, X-Content-Type-Options están presentes; los tokens expiran y se renuevan correctamente.

SQL Injection

A. Consultas parametrizadas siempre (nada de concatenar strings)

Qué hacer: usa **prepared statements**/placeholders (o un ORM/query builder que lo haga por ti). Prohíbe expresamente concatenar entradas del usuario en SQL.

Ejemplo (pg nativo):

```
// ❌ Mal (concatena):
// const q = `SELECT * FROM exhibitions WHERE title ILIKE '%${search}%'`;

// ✅ Bien (parametrizado):
const q = 'SELECT * FROM exhibitions WHERE title ILIKE $1 LIMIT $2';
const { rows } = await pool.query(q, [`%${search}%`, 20]);
```

Ejemplo (Knex):

```
const rows = await knex('exhibitions')
  .whereILike('title', `%${search}%`)
  .limit(20);
```

B. Validación en backend y control de acceso a datos

Qué hacer: valida **tipo y rango** (allowlist) en el backend para *cada* parámetro (números, UUID, enums). Además de evitar inyecciones, evita **IDOR**: que un usuario no pueda leer/editar recursos que no son suyos.

Patrón simple:

1. Valida esquema (p. ej., con **zod/joi**) antes de tocar la BD.
2. Aplica **ownership**: filtra por WHERE owner_id = $currentUserId.
3. Devuelve **403** cuando el recurso no pertenece al usuario.

Qué validas al final: con un token de "Usuario A" y un :id de "Usuario B", la API responde 403 o vacío; nunca devuelve datos de otro usuario.

C. Endurecimiento de base de datos y errores

Qué hacer:

- **Mínimos privilegios**: un usuario **solo-lectura** para endpoints GET y otro de **lectura/escritura** para POST/PUT/DELETE. Nada de permisos DDL para la app.
- **Manejo de errores**: responde mensajes genéricos ("Solicitud inválida") y loggea detalles en el servidor (sin datos sensibles).
- Opcional, pero útil en demos: **WAF** o reglas básicas para SQLi, y alertas cuando detectes patrones sospechosos.

Qué validas al final: con el usuario de app no puedes DROP TABLE; errores de BD no "filtran" nombres de tablas o SQL completo; los intentos quedan en logs.

Conclusiones

El presente plan de pruebas establece un marco estructurado de validación de seguridad para la plataforma MARCO, abordando más críticas identificadas por marco OWASP tanto para entornos web como móviles. La selección de estas pruebas responde directamente a los riesgos más prevalentes en aplicaciones modernas que manejan autenticación de usuarios y transacciones sensibles como la compra de boletos en línea. Es gracias a estas pruebas que podemos seguir una implementación correcta de controles de seguridad técnicos y validar cumplimiento de principios fundamentales como defensa en profundidad, mínimo privilegio y seguridad por diseño gracias a que integramos pruebas de seguridad desde el momento de desarrollo.

La incorporación de remediaciones específicas para criptografía insuficiente e inyección SQL fungen como guías concretas de implementación que el equipo de desarrollo (suponiendo que se plantea desarrollar este proyecto ficticio), podrá seguir durante la construcción de la plataforma. Dando lugar al desarrollador seguro.