
Plan de Pruebas Integral – Enfoque DevOps/DevSecOps

1. Introducción

El objetivo de este plan de pruebas es garantizar la calidad, seguridad y desempeño de la aplicación mediante una estrategia de pruebas que abarque desde pruebas unitarias hasta pruebas de seguridad avanzadas, integradas en el pipeline de CI/CD. Se aplicará el concepto de **Shift Left Testing** para detectar y corregir defectos lo antes posible y se incluirán pruebas de ciberseguridad para detectar vulnerabilidades comunes (inyección SQL, CSRF, XSS, etc.) y realizar pentesting.

---

2. Alcance y Objetivos

Alcance

- **Funcionalidades críticas:**
    - Gestión de evaluaciones (registro, redirección y visualización en historial).
    - Eliminación segura de registros mediante validación CSRF.
- **Seguridad:**
    - Validación de entradas y protección contra inyecciones.
    - Verificación de tokens CSRF en formularios.
    - Ejecución de pruebas de pentesting básicas (simulación de inyección SQL, XSS, etc.).
- **Rendimiento:**
    - Medición de tiempos de respuesta y estabilidad bajo carga.
- **Integración y despliegue:**
    - Automatización de pruebas en cada commit (CI/CD).

Objetivos

- **Funcional:** Confirmar que cada endpoint y flujo de usuario (desde la evaluación hasta la eliminación en historial) se comporta según lo esperado.
- **Seguridad:** Detectar vulnerabilidades en el código (por ejemplo, inyección SQL, CSRF, XSS) y garantizar que las medidas de seguridad (análisis SAST, DAST, IAST y pentesting) estén implementadas.
- **Rendimiento:** Asegurar que la aplicación responda de forma óptima bajo escenarios de carga y estrés.
- **DevOps/DevSecOps:** Integrar todas las pruebas en un pipeline automatizado que incluya análisis de seguridad y monitoreo continuo.

---

3. Estrategia de Pruebas

3.1 Tipos de Pruebas

A. Pruebas Funcionales

- **Unitarias:**
  - Validar funciones individuales (ej. determinarCategoriaRiesgo, validadores, helpers de SQL).
  - Herramientas: Jest, Mocha.
- **Integración:**
  - Verificar la comunicación entre módulos (rutas, controladores y base de datos).
  - Herramientas: Supertest, Postman.
- **Sistema y Aceptación (End-to-End):**
  - Simular el flujo completo del usuario (registro de evaluación, redirección, visualización en historial y eliminación).
  - Herramientas: Cypress, Selenium o Puppeteer.

B. Pruebas No Funcionales

- **Rendimiento y Carga:**
  - Medir tiempos de respuesta y estabilidad con usuarios concurrentes.
  - Herramientas: JMeter, k6, Artillery.
- **Usabilidad:**
  - Evaluar la experiencia del usuario mediante revisiones manuales (para detectar mejoras en la interfaz y flujo de usuario).

C. Pruebas de Seguridad (Ciberseguridad)

- **SAST (Static Application Security Testing):**
  - Analizar el código fuente para detectar vulnerabilidades antes de ejecutar la aplicación.
  - Herramientas: SonarQube, ESLint (con plugins de seguridad).
- **DAST (Dynamic Application Security Testing):**
  - Simular ataques externos para identificar vulnerabilidades en tiempo de ejecución.
  - Herramientas: OWASP ZAP, Burp Suite.
- **Pentesting Básico:**
  - Simular ataques de inyección SQL, XSS, CSRF, entre otros, en los formularios y endpoints críticos.
  - Herramientas: OWASP ZAP, pruebas manuales utilizando payloads comunes (ej. "' OR 1=1 --").
- **IAST/ RASP (Opcional):**
  - Integrar herramientas que monitoreen el comportamiento en tiempo real para detectar y prevenir ataques durante la ejecución.

3.2 Integración en CI/CD (DevOps y DevSecOps)

- **Pipeline Automatizado:**
  - Cada commit dispara pruebas unitarias, integración y análisis de seguridad.
  - Herramientas sugeridas: GitHub Actions, GitLab CI o Jenkins.
- **Análisis de Seguridad Automatizado:**
  - Integrar escáneres SAST y DAST en el pipeline.
  - Configurar alertas y fallos en el pipeline si se detectan vulnerabilidades críticas.
- **Despliegue Canary y Monitoreo:**
  - Utilizar feature flags para liberar cambios de forma controlada y monitorear en producción con herramientas de logging y APM (por ejemplo, ELK, Datadog).

---

4. Herramientas y Recursos

Tipo de Prueba	Herramienta Sugerida
Pruebas Unitarias	Jest, Mocha
Pruebas de Integración	Supertest, Postman
Pruebas E2E/Funcionales	Cypress, Selenium, Puppeteer
Análisis SAST	SonarQube, ESLint con plugins de seguridad
Análisis DAST/Pentesting	OWASP ZAP, Burp Suite
Pruebas de Carga/Rendimiento	JMeter, k6, Artillery
CI/CD	GitHub Actions, GitLab CI, Jenkins
Monitoreo	ELK Stack (Elastic, Logstash, Kibana), Datadog

---

5. Escenarios y Casos de Prueba

5.1 Casos de Prueba Funcionales

- **CPF-01: Evaluación Exitosa – Cuestionario de Riesgos**
  - **Precondición:** El usuario ingresa un nombre y número de episodios válidos, y responde a todas las preguntas.
  - **Entrada:** Datos válidos en /formulario1.
  - **Resultado Esperado:** Respuesta JSON con success: true y redirección a /Cuestionario_Niveles?paciente=....
  - **Automatización:** Prueba con Jest/Supertest.

- **CPF-02: Validación de Datos Incorrectos**
    - **Precondición:** El usuario deja campos vacíos o introduce datos fuera de rango.
    - **Entrada:** Nombre muy corto o episodio fuera del rango permitido.
    - **Resultado Esperado:** Respuesta con errores de validación (código 400).
    - **Automatización:** Pruebas unitarias de validadores.
- **CPF-03: Flujo Completo de Evaluación**
    - **Flujo:** Desde el formulario de riesgos, redirección a niveles, envío de evaluación detallada y registro en historial.
    - **Resultado Esperado:** Visualización en el historial y correcta eliminación si se solicita.

5.2 Casos de Prueba de Seguridad (Ciberseguridad)

- **CPS-01: Prueba de Inyección SQL**
    - **Precondición:** Ingresar payloads maliciosos en los formularios (ej. "' OR 1=1 --").
    - **Entrada:** Datos maliciosos en campos críticos.
    - **Resultado Esperado:** El sistema debe rechazar la entrada, no procesar la inyección y registrar el intento.
    - **Automatización:** Ejecución de OWASP ZAP y pruebas manuales.
- **CPS-02: Prueba de CSRF**
    - **Precondición:** Realizar peticiones sin token CSRF o con token inválido.
    - **Entrada:** Simular envío de formularios sin el token requerido.
    - **Resultado Esperado:** Rechazo de la petición (HTTP 403 o error de validación).
    - **Automatización:** Scripts en Postman o pruebas con herramientas DAST.
- **CPS-03: Prueba de XSS (Cross-Site Scripting)**
    - **Precondición:** Ingresar scripts en campos de texto.
    - **Entrada:** Payloads típicos de XSS en el campo "nombre" u otros campos de entrada.
    - **Resultado Esperado:** Los scripts no se ejecutan y se muestran como texto plano; validación o sanitización de entradas.
    - **Automatización:** Herramientas DAST y pruebas manuales.
- **CPS-04: Escaneo de Vulnerabilidades de Dependencias**
    - **Precondición:** Utilizar herramientas para revisar dependencias del proyecto.
    - **Entrada:** Ejecución de comandos como npm audit y análisis con Snyk.
    - **Resultado Esperado:** Identificar y mitigar vulnerabilidades en paquetes de terceros.

5.3 Casos de Prueba de Rendimiento y Carga

- **CPR-01: Prueba de Carga en el Endpoint de Historial**
    - **Precondición:** Base de datos poblada con evaluaciones.
    - **Entrada:** Simulación de múltiples usuarios accediendo al historial concurrentemente.
    - **Resultado Esperado:** Tiempos de respuesta dentro de los parámetros definidos (<300 ms) y estabilidad del servidor.
    - **Automatización:** JMeter o k6.
- **CPR-02: Prueba de Estrés**
    - **Precondición:** Simular una carga extrema para identificar puntos de fallo.
    - **Entrada:** Incremento progresivo de usuarios hasta alcanzar el límite.
    - **Resultado Esperado:** Identificar el umbral máximo antes de la degradación del servicio y planificar mejoras.

---

6. Entorno de Pruebas

- **Local:**
    - Servidor de desarrollo con configuración de Node.js, Express y MySQL.
    - Base de datos de pruebas (seeding inicial para escenarios controlados).
- **CI/CD:**
    - Entorno de integración continua donde se ejecutan las pruebas automatizadas (por ejemplo, en GitHub Actions o GitLab CI).
- **Staging:**
    - Entorno similar a producción para pruebas de sistema, carga y seguridad antes del despliegue final.

---

7. Ejecución y Cronograma

1. **Configuración Inicial y Seeding:**
    - Configurar variables de entorno, seeding de base de datos y herramientas de análisis.
2. **Desarrollo de Casos de Prueba:**
    - Escribir y automatizar casos de prueba unitarias, de integración y funcionales.
    - Crear scripts para pruebas de seguridad (ej. payloads de inyección, CSRF, XSS).
3. **Integración en Pipeline CI/CD:**
    - Configurar el pipeline para ejecutar pruebas en cada commit/push.
    - Integrar análisis SAST/DAST y escaneos de dependencias.
4. **Pruebas en Staging:**
    - Ejecución de pruebas de sistema, carga y seguridad en entorno de preproducción.
5. **Despliegue y Monitoreo en Producción (Canary):**
    - Liberar versiones gradualmente con monitoreo activo.

---

8. Roles y Responsabilidades

- **Desarrollador/Tester:**
    - Diseñar y automatizar los casos de prueba.
    - Configurar y gestionar el pipeline CI/CD con integración de análisis de seguridad.
    - Ejecutar pruebas manuales y remediar defectos identificados.
    - Documentar los resultados y realizar seguimiento de incidencias (por ejemplo, usando GitHub Issues o Jira).

---

9. Métricas y KPIs

- **Cobertura de Código:** Objetivo > 80% en pruebas unitarias e integración.
- **Tiempo de Respuesta:** API y endpoints con tiempos menores a 300 ms.
- **Vulnerabilidades Críticas:** 0 defectos críticos detectados en escaneos SAST/DAST.
- **Número de Incidentes de Seguridad:** Seguimiento y reducción progresiva.
- **Estabilidad Bajo Carga:** Disponibilidad del sistema ≥ 99.9% en pruebas de rendimiento.

---

10. Reporte y Seguimiento de Defectos

- **Registro de Defectos:**
    - Utilizar una herramienta de gestión de incidencias (por ejemplo, GitHub Issues o Jira) para documentar y hacer seguimiento a los defectos encontrados.
- **Revisión y Retroalimentación:**
    - Documentar resultados de cada fase de prueba y realizar reuniones de revisión (incluso de manera individual) para ajustar el plan.
- **Notificaciones en el Pipeline:**
    - Configurar alertas automáticas en caso de fallo en el pipeline de CI/CD.

---

11. Conclusión

Este plan de pruebas integral está diseñado para cubrir todos los aspectos críticos del sistema, integrando pruebas funcionales tradicionales y de ciberseguridad dentro de un entorno DevOps/DevSecOps. La automatización y el enfoque en seguridad desde las primeras etapas (Shift Left) garantizarán que el software se entregue con altos estándares de calidad y resiliencia frente a ataques.