DOCUMENTO DE TOLERANCIA A FALLOS DEL SISTEMA DE GESTIÓN DE PRÉSTAMOS ONLINE DE BIBLIOTECA

Curso de Construcción de Software

(Banchon Salas Roger Adonis, Loza Orozco Cristopher Mauricio, Macas Jimenez David Alejandro, Martinez Gamarra Daniel Andres, Valencia Mora Carlos Alexander)

Contenido

[1. Introducción 3](#_Toc197561440)

[2. Fallos Identificados 3](#_Toc197561441)

[3. Mecanismos de Tolerancia a Fallos Implementados 4](#_Toc197561442)

[4. Prevención de Defectos 4](#_Toc197561443)

[5. Uso de Redundancia y Diversidad 5](#_Toc197561444)

[6. Caracteristicas del Proceso de Desarrollo Confiable 6](#_Toc197561445)

[7. Escenarios de Tolerancia a Fallos 6](#_Toc197561446)

# Introducción

En el desarrollo de sistemas de software críticos, como el Sistema de Gestión de Préstamos Online de Biblioteca, es fundamental garantizar que la aplicación sea capaz de responder de forma controlada ante condiciones inesperadas, errores o fallos operativos. La tolerancia a fallos se refiere a la capacidad del sistema para mantener su funcionamiento, evitando caídas o comportamientos erráticos, incluso cuando se presentan errores internos o entradas incorrectas por parte del usuario.

Este documento detalla los mecanismos implementados en cada módulo del sistema para manejar errores comunes y prevenir fallos críticos. Se describen las validaciones aplicadas, las respuestas controladas frente a entradas inválidas y las decisiones de diseño que permiten mejorar la robustez del sistema. El análisis se basa en cinco componentes clave: conexión a base de datos, login, búsqueda de libros, gestión de préstamos y gestión de usuarios.

# Fallos Identificados

Tabla de Fallos Identificados

|  |  |  |
| --- | --- | --- |
| Módulo | Tipo de Fallo | Situación |
| database.php | Fallo de conexión a BD | Cuando las credenciales de conexión son incorrectas o la BD está caída. |
| LoginController.php | Credenciales vacías o incorrectas | Cuando el usuario no ingresa datos o lo hace mal. |
| LoginController.php | Rol no reconocido | Cuando el usuario tiene un rol\_id que no está contemplado. |
| LibroController.php | Búsqueda vacía | Cuando se intenta buscar sin ingresar un término. |
| PrestamoController.php | Datos faltantes al crear solicitud | Cuando no se envía alguno de los datos requeridos. |
| PrestamoController.php | Solicitud de préstamo no encontrada | Cuando se intenta aceptar un préstamo con ID inválido o inexistente. |
| UserController.php | Registro con campos incompletos | Cuando se omite algún campo obligatorio del formulario. |
| UserController.php | Registro duplicado (cédula ya registrada) | Al intentar registrar un usuario que ya existe. |
| UserController.php | Cambio de contraseña con contraseña actual incorrecta | Cuando el usuario ingresa mal su contraseña anterior. |

# Mecanismos de Tolerancia a Fallos Implementados

|  |  |  |
| --- | --- | --- |
| Módulo | Mecanismo aplicado | Descripción |
| database.php | try-catch en conexión PDO | Captura fallos al conectar a la base de datos y detiene la ejecución con un mensaje seguro. |
| LoginController.php | Validación de campos vacíos | Impide continuar si usuario o contraseña están vacíos. |
| LoginController.php | Verificación de usuario en base de datos | Impide iniciar sesión con credenciales inválidas. |
| LoginController.php | Manejo de roles desconocidos | Detiene la ejecución si el rol del usuario no está contemplado. |
| LibroController.php | Validación de búsqueda vacía | Responde con array vacío si no se proporciona término. |
| LibroController.php | Respuesta estructurada en JSON | Siempre responde en formato JSON, incluso si no hay resultados. |
| PrestamoController.php | Validación de parámetros en creación de solicitud | Se revisa que todos los campos estén presentes antes de continuar. |
| PrestamoController.php | Uso de operador ?? para evitar errores por índice inexistente en $\_POST | Se establecen valores por defecto si no se recibe algún parámetro. |
| UserController.php | Validación de campos en registro | Recorre y verifica que todos los campos estén llenos. |
| UserController.php | Verificación de duplicados | Revisa si la cédula ya existe antes de registrar. |
| UserController.php | Validación de contraseña anterior en cambio de clave | No permite cambiar la contraseña si la actual no es correcta. |

# Prevención de Defectos

**Validación de entradas:**

* Se revisa que los campos requeridos estén completos en formularios como registro y login.
* Se evita que se procesen datos vacíos o inválidos desde el inicio.

**Aplicación de reglas de negocio:**

* Se comprueba si la cédula del usuario ya está registrada para evitar duplicados.
* Solo se permite registrar usuarios nuevos con datos únicos.

**Control de tipos de datos:**

* Se verifica que los valores ingresados correspondan al tipo esperado (números, fechas, textos).
* Se previenen errores por tipos incompatibles que afectarían consultas o registros.

**Uso de interfaces controladas:**

* Se utilizan campos select (menús desplegables) y controles de fecha en los formularios.
* Se restringen las opciones del usuario para minimizar el error humano.

**Manejo de sesiones:**

* Se controla que el usuario esté logueado antes de permitir acceso a ciertas funciones.
* Se protege el flujo del sistema contra accesos no autorizados.

**Retroalimentación inmediata al usuario:**

* Se muestran mensajes claros (en pantalla o vía $\_SESSION o JSON) ante errores.
* El usuario puede corregir los problemas sin afectar el sistema.

**Separación por capas (MVC):**

* La lógica del sistema está organizada en controladores, modelos y vistas.
* Esto facilita detectar errores durante el desarrollo y mejora el mantenimiento.

# Uso de Redundancia y Diversidad

**Redundancia lógica de validación:**

* Los datos se validan tanto en el frontend (por el formulario) como en el backend (por los controladores).
* Esto garantiza que, aunque el usuario manipule el formulario, no pueda enviar datos inválidos al servidor.

**Redundancia de control de errores:**

* En varios métodos, si ocurre un fallo (por ejemplo, datos faltantes), se corta la ejecución y se notifica el error, evitando que se siga procesando algo incorrecto.

**Diversidad de mecanismos de respuesta:**

* El sistema responde con diferentes formatos según el contexto: mensajes HTML, sesiones (para redirección) o JSON (para AJAX).
* Esto permite manejar errores tanto en vistas tradicionales como en llamadas asincrónicas.

**Diversidad de módulos con control independiente:**

* Cada controlador (usuarios, libros, préstamos) maneja sus propios errores y validaciones.
* No dependen unos de otros para reaccionar ante fallos, lo cual mejora la estabilidad por componente.

**Redundancia de seguridad en sesiones:**

* El sistema valida y destruye sesiones en logout, y controla los accesos según el rol.
* Así se evita que usuarios no autorizados accedan a otras secciones.

# Caracteristicas del Proceso de Desarrollo Confiable

**Separación en capas (MVC):**

* El sistema está dividido claramente en modelos, controladores y vistas.
* Esto facilita el mantenimiento, el testeo por partes y la localización de errores.

**Validaciones sistemáticas:**

* Se valida la entrada del usuario en todos los módulos antes de ejecutar acciones.
* Esto evita el procesamiento de datos corruptos o incompletos.

**Manejo controlado de errores:**

* Ante condiciones inválidas, el sistema responde de forma predecible: corta la ejecución, muestra mensajes claros o devuelve JSON estructurado.
* Esto reduce el riesgo de fallos inesperados.

**Uso de sesiones para feedback:**

* Se utiliza $\_SESSION para conservar mensajes de error, éxito o datos viejos entre peticiones.
* Esto mejora la experiencia del usuario sin exponer información técnica.

**Autenticación y control de acceso:**

* El sistema diferencia claramente entre roles de usuario (general y bibliotecario).
* Esto evita accesos indebidos a funcionalidades que no corresponden.

# Escenarios de Tolerancia a Fallos

**Inicio de sesión con datos vacíos:**

* Si el usuario no escribe ni usuario ni contraseña, el controlador muestra un mensaje claro ("Usuario y contraseña requeridos") y detiene la ejecución.

**Inicio de sesión con credenciales incorrectas:**

* El sistema valida las credenciales con el modelo y, si son incorrectas, responde con el mensaje “Credenciales incorrectas”, sin exponer información interna.

**Registro de usuario con datos faltantes:**

* Si algún campo obligatorio (como nombre, cédula, email, etc.) está vacío, el sistema interrumpe el registro, almacena el mensaje de error en sesión y redirige al formulario.

**Registro de usuario con cédula duplicada:**

* Antes de crear un nuevo usuario, se verifica si la cédula ya existe en la base de datos. Si es así, se notifica el error y se conservan los datos anteriores en sesión para no perderlos.

**Error al conectar con la base de datos:**

* Si por algún motivo falla la conexión PDO, el sistema lanza una excepción y responde con el mensaje “Error de conexión a BD”, sin mostrar detalles sensibles.

**Cambio de contraseña con clave antigua incorrecta:**

* El sistema verifica la contraseña anterior antes de actualizar la nueva. Si no coincide, responde con un mensaje JSON controlado, sin continuar el proceso.

**Solicitud de préstamo con datos incompletos:**

* Si falta alguno de los campos (libro, usuario, fechas), la solicitud no se crea. Se responde en formato JSON con éxito: false y un mensaje explicativo.

**Consultas AJAX vacías o sin resultados:**

* Aunque una búsqueda no tenga coincidencias, siempre se responde con JSON válido, evitando fallos en la vista.