**Tolerancia a Fallos en la Aplicación Web PWA de Registro de asistencia**

**Introducción**

La tolerancia a fallos es la capacidad de un sistema para continuar funcionando de manera aceptable ante errores, caídas o condiciones imprevistas. En la aplicación Web PWA de Registro de Asistencia, esta característica es esencial debido a su enfoque offline-first y la necesidad de asegurar una experiencia robusta para el usuario, incluso en condiciones de conectividad inestable o nula.

Este documento analiza los mecanismos aplicados para tolerancia a fallos, así como las prácticas de **prevención de defectos** y elementos relacionados con el desarrollo de **software confiable**, según el enfoque planteado por Sommerville.

**Fallos Identificados**

|  |  |
| --- | --- |
| **Tipo de Fallo** | **Descripción** |
| **Fallo de conexión a Internet** | El usuario intenta registrar asistencia sin acceso a red. |
| **Pérdida de datos en caché** | Los datos guardados en localStorage pueden borrarse accidentalmente. |
| **Ingreso incompleto de materias** | El usuario intenta registrar asistencia sin haber definido materias. |
| **Registro fuera de horario** | El usuario intenta registrar asistencia en un horario incorrecto. |
| **Recarga inesperada** | Se pierde la edición de una materia si la página se recarga. |
| **Errores de validación** | El formulario acepta datos vacíos o mal estructurados. |

**Mecanismos de Tolerancia a Fallos Implementados**

|  |  |
| --- | --- |
| **Fallo Previsto** | **Mecanismo de tolerancia implementado** |
| **Sin conexión a Internet** | Uso de Service Worker para cachear recursos y localStorage para guardar datos temporalmente. |
| **Datos incompletos o inválidos** | Validaciones en JavaScript (frontend): no se permite guardar materias con campos vacíos. |
| **Registro sin materias** | Al abrir el módulo de asistencia, si no hay materias registradas, se muestra un mensaje de advertencia. |
| **Registro fuera de horario** | Lógica que valida el día y la hora actual para permitir o bloquear el registro. |
| **Recarga inesperada** | Los datos temporales de edición se limpian y se notifica al usuario para evitar confusión. |
| **Interface adaptable** | Diseño responsive que previene errores visuales en pantallas pequeñas o móviles. |

**Prevención de Defectos**

Para aumentar la confiabilidad, se han tomado medidas de **prevención de errores en fases tempranas**:

* **Validaciones de entrada:** Se bloquea el registro de datos incompletos o erróneos desde el frontend.
* **Modularidad:** La aplicación se estructura por módulos funcionales, lo que reduce el riesgo de errores cruzados.
* **Controles visuales:** Se utiliza retroalimentación visual (alertas, mensajes, colores) para evitar acciones erróneas.
* **Prototipado y pruebas exploratorias:** Se realizaron pruebas básicas con usuarios simulados antes de avanzar al backend.

**Uso de Redundancia y Diversidad**

Aunque la aplicación aún no está conectada a un backend, ya se han considerado mecanismos de **redundancia y diversidad**:

* **Redundancia de almacenamiento:** Se utiliza localStorage (Service-Worker) como almacenamiento temporal ante fallos de conexión.
* **Diversidad en validaciones:** Se prevé aplicar validaciones tanto en **frontend como backend** para evitar fallos en una sola capa del sistema.
* **Redundancia funcional futura:** Al implementar el backend, se planea incluir backups automáticos y múltiples puntos de sincronización.

**Características del Proceso de Desarrollo Confiable**

De acuerdo con Sommerville, el desarrollo de software confiable incluye procesos y prácticas que reducen el riesgo de fallos:

* **Desarrollo incremental:** La app se construyó por etapas (materias, horarios, asistencia) permitiendo ajustes tempranos.
* **Control de versiones:** El código se gestiona con controlador de versiones de git y GitHub para permitir recuperaciones rápidas.
* **Pruebas frecuentes:** Cada módulo se valida manualmente en distintos dispositivos y navegadores.
* **Diseño centrado en el usuario:** La interfaz fue probada con estudiantes para asegurar facilidad de uso y minimizar errores.

**Escenarios de Tolerancia a Fallos**

**Escenario 1: Usuario intenta registrar asistencia sin conexión**

* **Situación:** El estudiante entra a la app sin conexión a Internet.
* **Resultado esperado:** La app carga desde el cache permite registrar asistencia y guarda la respuesta localmente.
* **Manejo de fallo:** Se notifica que la app está sin conexión; los datos serán sincronizados cuando se restablezca la conexión.

**Escenario 2: No hay materias registradas**

* **Situación:** El usuario quiere marcar asistencia pero no ha ingresado materias.
* **Resultado esperado:** Se muestra un mensaje informativo: “No se han registrado materias, por favor agregue una primero.”
* **Manejo de fallo:** El botón de asistencia está deshabilitado hasta que se registren materias.

**Escenario 3: El usuario intenta registrar asistencia fuera del horario de clases**

* **Situación:** Intenta registrar una clase el sábado, pero su horario solo tiene lunes a viernes.
* **Resultado esperado:** El sistema impide el registro.
* **Manejo de fallo:** Se muestra una alerta indicando que no hay clases en ese horario.

**Escenario 4: El usuario recarga la página durante la edición de una materia**

* **Situación:** Mientras edita una materia, accidentalmente actualiza el navegador.
* **Resultado esperado:** Los datos no guardados se pierden.
* **Manejo de fallo:** Se limpia el formulario y se muestra un mensaje para evitar confusión. En futuras versiones podría mantenerse un borrador en localStorage.

**Escenario 5: Campos vacíos o inválidos en el formulario**

* **Situación:** El usuario intenta guardar una materia sin completar todos los campos.
* **Resultado esperado:** Se bloquea el envío del formulario.
* **Manejo de fallo:** Aparecen mensajes de error indicando qué campos deben completarse correctamente.

**Limitaciones y Mejoras Futuras**

Actualmente, el sistema aún no cuenta con manejo de errores a nivel de servidor o base de datos, ya que estas funcionalidades están previstas para la segunda etapa del proyecto. Una vez integrados, se podrán aplicar mecanismos adicionales de tolerancia como:

* Reintentos automáticos de sincronización.
* Backups programados.
* Logs de errores del servidor.
* Alertas de error con detalles técnicos.