Proyecto Integrador

Integrantes: Julian Torres, Pablo Yanez

Descripción del Proyecto

Diagnóstico de la Problemática (Global)

En muchas comunidades urbanas y conjuntos residenciales, existe una ineficiencia en el uso de recursos domésticos. Los residentes compran herramientas costosas (taladros, sierras, escaleras) que utilizan esporádicamente, generando un gasto innecesario y problemas de almacenamiento. Simultáneamente, no existen canales de confianza seguros para pedir prestado estos objetos a los vecinos, lo que fomenta el aislamiento social y el consumismo desmedido. No hay una forma centralizada de saber "quién tiene qué" ni garantías de devolución.

Funcionalidad del Proyecto (Core)

Veci-Herramientas es una plataforma web de economía colaborativa barrial que permite:

1. **Gestión de Inventario:** Los usuarios pueden publicar herramientas propias, indicando si son para **Préstamo** (gratuito/trueque) o **Venta**.
2. **Ciclo de Préstamos:** Un flujo completo de solicitud, aprobación por parte del dueño, entrega (cambio de estado a "Activo") y confirmación de devolución.
3. **Sistema de Reputación:** Calificación y reseñas entre vecinos para construir un ecosistema de confianza.

Restricciones Identificadas:

- **Geográfica:** El sistema está diseñado para funcionar en entornos cerrados o barrios específicos (radio limitado) para facilitar la logística física.
- **Validación:** La entrega física y la verificación del estado de la herramienta se realizan *offline* (cara a cara); el sistema solo registra el cambio de estados.
- **Conectividad:** Requiere conexión a internet activa (no funciona offline).

Alcance y Limitaciones

Alcance:

- Aplicativo Web Frontend (SPA) en React.
- Backend API REST con seguridad JWT.
- Roles: Administrador (moderación de herramientas) y Usuario Vecino.
- Chat básico para coordinación.

Limitaciones:

- No incluye pasarela de pagos en línea (los pagos de ventas son en efectivo contra entrega).
- No incluye seguimiento GPS en tiempo real de la herramienta.

Planteamiento de Alternativas de Solución

- **Alternativa A: Grupos de WhatsApp/Telegram.**
    - *Desventaja:* Información desorganizada, difícil búsqueda, sin historial de reputación, mensajes perdidos.
- **Alternativa B: App Móvil Nativa.**
    - *Desventaja:* Alto costo de desarrollo, fricción para que el usuario descargue una app solo para uso esporádico.
- **Alternativa C: Web App (SPA) Responsive (Solución Seleccionada).**
    - *Justificación:* Es la mejor solución por su **accesibilidad universal** (funciona en cualquier navegador móvil o PC sin instalación), bajo costo de mantenimiento y facilidad de actualización. Permite una experiencia fluida similar a una app (PWA) sin las barreras de entrada de las tiendas de aplicaciones.

2. Diagramas de Diseño

Diagrama de Clases

Diagrama de Estados

3. Mejoras SOLID y Patrones de Diseño

Identificación del Problema (Código Inicial)

En una implementación ingenua, es común encontrar **Controladores Monolíticos** donde la lógica de validación, acceso a datos y respuesta HTTP están mezcladas en un solo método createLoan(). También es común ver múltiples if/else anidados para manejar si una herramienta es de "Venta" o de "Préstamo".

Solución Propuesta

1. **Principio de Responsabilidad Única (SRP):** Separar la lógica en Capas: Controller (solo HTTP), Service (Lógica de negocio), Repository (Acceso a BD).
2. **Patrón Strategy (Comportamiento):** Para manejar la diferencia entre "Venta" y "Préstamo". En lugar de if (type == 'sale'), tener una interfaz ITransactionStrategy con implementaciones SaleStrategy y LoanStrategy.

3. **Patrón Factory:** Para la creación de notificaciones. Cuando cambia el estado de un préstamo, una NotificationFactory decide si enviar un email o una alerta en la app.

[Diagrama de clases pequeño mostrando la interfaz Strategy y sus dos clases concretas]

4. Pruebas Funcionales (Documentación)

Caso de Prueba 1: Solicitud de Préstamo Exitosa

- **Precondición:** Usuario A logueado, Usuario B tiene herramienta "Taladro" disponible.
- **Pasos:**
    1. Usuario A busca "Taladro".
    2. Clic en "Solicitar", selecciona fechas y confirma.
- **Resultado Esperado:** Se crea un registro de préstamo con estado "Pendiente". El Usuario B recibe notificación.
- **Estado:** Aprobado

Caso de Prueba 2: Restricción de Rol Admin

- **Precondición:** Usuario con rol "User" intenta acceder a /admin/tools.
- **Pasos:**
    1. Usuario intenta navegar vía URL a la ruta protegida.
- **Resultado Esperado:** El sistema redirige al Dashboard o muestra "Acceso Denegado (403)".
- **Estado:** Aprobado

5. Propuesta de API (JSON)

Endpoint: GET /api/tools

Descripción: Obtener catálogo de herramientas disponibles.

JSON:


```
{
  "data": [
    {
      "id": "t-101",
```

```
      "name": "Taladro Percutor Bosch",
      "category": "Carpintería",
      "type": "loan",
      "status": "available",
      "owner": {
        "id": "u-55",
        "name": "Julian Torres",
        "reputation": 4.8
      }
    },
    {
      "id": "t-102",
      "name": "Juego de Llaves Inglesas",
      "category": "Mecánica",
      "type": "sale",
      "price": 25.00,
      "status": "available",
      "owner": {
        "id": "u-22",
        "name": "Hamilton Lugmaña",
        "reputation": 5.0
      }
    }
  ],
  "meta": {
    "total": 2,
    "page": 1
  }
}
```

Implementación en Código

1. Aplicativo basado en Framework JS nuevo

Todo el código esta sesión está hecho en **React con Vite**.

- Si tu proyecto anterior era en Vue.js o Angular, **React** cuenta como el framework nuevo.
- El diseño **Neubrutalista** que implementamos le da un valor agregado visual único.

2. Implementación de API y Seguridad (Backend)

El Back en Node.js

- **Middleware de Autenticación:** Que verifique el Token JWT en los headers (Authorization: Bearer ...).
- **Guardias de Roles:** En el endpoint de Admin, verificar if (user.role !== 'admin') return 403;.

3. Patrones en el Frontend (React)

Estamos usando buenas prácticas y patrones implícitos:

- **Provider Pattern:** Usamos AuthProvider, ToolsContext, LoansContext para inyectar dependencias y estado global.
- **Custom Hooks:** Separamos la lógica de la vista (ej. useAuth(), useTools()).
- **Component Composition:** Reutilizamos componentes como la tarjeta de herramienta o los badges.