Script de Desarrollo Local

Script de Automatización para Entorno de Desarrollo

Propósito General

Este script shell (levantar.sh) automatiza el proceso de configuración y ejecución del entorno de desarrollo local para la aplicación IoT. Proporciona una forma consistente y confiable de levantar tanto el backend como el frontend con verificación de dependencias y manejo graceful de procesos.

Estructura del Script

Shebang y Configuración Inicial

bash

```
set -e
```

Modo estricto: El script termina inmediatamente si cualquier comando falla

Robustez: Previene continuar con estados inconsistentes

Funciones Auxiliares

Verificación de Comandos

bash

```
check_cmd() {
  command -v "$1" >/dev/null 2>&1 || {
    echo "Falta '$1'. Instalalo y volvé a probar.";
    exit 1;
  }
}
```

Propósito: Valida que comandos esenciales estén disponibles en el sistema

Comportamiento: Termina el script con mensaje claro si falta algún comando

Salida silenciosa: Redirección a /dev/null para output limpio

Limpieza de Procesos

bash

```
cleanup() {
    echo ""
    echo " 🛑 Deteniendo servidores..."
    jobs -p | xargs -r kill
```

```
    echo "✅ Servidores detenidos"
    exit 0
}
```

Manejo de señales: Captura SIGINT (Ctrl+C) y SIGTERM

Terminación graceful: Mata todos los procesos hijos del script

Feedback visual: Informa al usuario el estado de la terminación

Configuración de Traps

bash

```
trap cleanup SIGINT SIGTERM
```

Interceptación: Captura señales de interrupción y terminación

Limpieza garantizada: Ejecuta cleanup sin importar cómo termine el script

Flujo Principal de Ejecución

1. Verificación de Prerrequisitos

bash

```
echo "Verificando Node.js y npm..."
check_cmd node
check_cmd npm
```

Node.js: Runtime esencial para el backend

npm: Gestor de paquetes para dependencias JavaScript

Validación temprana: Falla rápido si faltan dependencias críticas

2. Instalación de Dependencias del Backend

bash

```
cd backend
if [ ! -f "package.json" ]; then
    echo "❌ Error: No se encontró package.json en backend/"
    exit 1
fi
npm install
```

Verificación de estructura: Confirma que el directorio backend tiene configuración npm

Instalación limpia: npm install sin flags para instalar exactamente lo definido en package-lock.json

Manejo de errores: Verifica código de salida del comando npm

3. Instalación de Dependencias del Frontend (Condicional)

bash

```
cd ../frontend
if [ -f "package.json" ]; then
   echo "📦 Instalando dependencias del frontend..."
   npm install
   echo "✅ Dependencias del frontend instaladas"
else
   echo "ℹ️  No se encontró package.json en frontend/, continuando..."
fi
```

Verificación existencial: Solo instala si existe package.json

Tolerante a fallos: Continúa incluso si el frontend no está presente

Mensajes informativos: Indica claramente el estado de la operación

4. Verificación de Configuración

bash

```
cd ..
if [ ! -f ".env" ]; then
   echo "❌ Error: No se encontró el archivo 'env' en la raíz del proyecto"
   echo "   Crea el archivo env con las variables de configuración necesarias"
   exit 1
fi
echo "✅ Archivo de configuración encontrado"
```

Configuración crítica: El archivo .env es esencial para el funcionamiento

Mensaje de ayuda: Indica al usuario cómo resolver el problema

Validación de ruta: Verifica en el directorio raíz del proyecto

Levantamiento de Servidores

Backend en Segundo Plano

bash

```
(cd backend && npm start) &
```

Subshell aislada: Cambia directorio sin afectar el contexto principal

Ejecución en background: & permite continuar con el frontend

npm start: Comando estándar para iniciar aplicación Node.js

Frontend con Live Server

bash

```
(cd frontend && npx live-server public --host=127.0.0.1 --port=8080 --no-browser) &
```

Servidor de desarrollo: live-server con recarga automática

Host específico: 127.0.0.1 para evitar problemas de red

Puerto definido: 8080 para consistencia

Sin navegador: --no-browser evita apertura automática

Información al Usuario

bash

```
echo ""
echo "====================================="
echo " Ambos servidores corriendo en local "
echo " Backend: http://127.0.0.1:${PORT:-3000}"
echo " Frontend: http://127.0.0.1:8080 (por default live-server)"
echo " Presioná Ctrl+C para frenar todo."
echo "====================================="
```

URLs accesibles: Muestra endpoints exactos para acceder

Variable con fallback: ${PORT:-3000} usa PORT de .env o 3000 por defecto

Instrucción clara: Indica cómo detener los servidores

Mantenimiento del Script

bash

```
wait
```

Espera activa: Mantiene el script vivo hasta que reciba señal

Procesos hijos: Permite que los servidores continúen ejecutándose

Receptivo a señales: Permite que trap capture Ctrl+C

Características de Usabilidad

Mensajes Claros y Formateados

Emojis: Mejoran la legibilidad y experiencia de usuario

Separadores visuales: Organizan la salida en secciones lógicas

Colores implícitos: Usa símbolos (✅ , ❌ , ℹ️) para estados

Manejo Robusto de Errores

Validaciones tempranas: Falla rápido con mensajes descriptivos

Códigos de salida: Usa exit 1 para errores, exit 0 para terminación normal

Verificación de comandos: Confirma que todos los requisitos estén presentes

Compatibilidad y Portabilidad

Shell estándar: Usa características POSIX-compliant

Rutas relativas: Funciona independientemente del directorio de ejecución

Comandos universales: Utiliza herramientas disponibles en cualquier entorno Node.js

Proceso de Configuración Previo

Permisos de Ejecución

bash

chmod +x levantar.sh

Habilitación: Hace el script ejecutable en sistemas Unix/Linux

Un solo paso: Solo necesario la primera vez

Ejecución

bash

./levantar.sh

Directo: Ejecuta el script desde el directorio del proyecto

Autónomo: No requiere parámetros o configuración adicional

Este script proporciona una experiencia de desarrollo optimizada que reduce la fricción en el setup inicial y garantiza consistencia entre diferentes entornos de desarrollo.