🔧 Errores de Deploy Resueltos

Lista de Errores Conocidos y Soluciones

Error #1: Alpine Linux 3.20/3.21 - Repositorios Rotos

Síntoma: `fetch https://dl-cdn.alpinelinux.org/alpine/v3.21/main/x86_64/APKINDEX.tar.gz`
- Error: Connection timeout o repositorios no disponibles

Solución: Usar Alpine 3.19 en Dockerfile

```
FROM node:18-alpine3.19 AS base
```

Error #2: TypeScript - Enums de Prisma no exportados

Síntoma:

```
TS2305: Module '@prisma/client' has no exported member 'UserRole'
```

Solución: Generar Prisma client ANTES de `yarn build`

```
RUN ./node_modules/.bin/prisma generate --schema=./prisma/schema.prisma && \
    yarn build
```

Error #3: yarn.lock Faltante o Corrupto

Síntoma:

```
yarn install --frozen-lockfile fails
```

Solución: Verificar que yarn.lock existe y es válido

```
if [ ! -f "app/yarn.lock" ]; then
    print_error "yarn.lock no existe"
    exit 1
fi
```

Error #4: Prisma CLI No Encontrado

Síntoma:

```
bash: prisma: command not found
```

Solución: Usar path directo a binario

```
RUN ./node_modules/.bin/prisma generate
```

Error #5: Permisos en Alpine - addgroup/adduser

Síntoma:

```
addgroup: unrecognized option '--system'
```

Solución: Usar flags correctos para Alpine

```
RUN addgroup --system --gid 1001 nodejs && \
    adduser --system --uid 1001 nextjs
```

Error #6: node_modules/.bin No en PATH

Síntoma:

```
prisma: command not found (even with PATH=/app/node_modules/.bin:$PATH)
```

Solución: Usar path directo en lugar de confiar en PATH

```
RUN ./node_modules/.bin/prisma generate
```

Error #7: Yarn Ya Preinstalado

Síntoma:

```
error Package "yarn@1.22.22" is already installed globally
```

Solución: NO reinstalar yarn en node:18-alpine

```
# yarn ya viene preinstalado en node:18-alpine3.19
# No es necesario instalar yarn nuevamente
```

Error #8: yarn.lock es Symlink Roto

Síntoma:

```
COPY app/yarn.lock* ./
⚠ @prisma not found in node_modules
ERROR: "/app/node_modules": not found
```

Causa: yarn.lock es un symlink que apunta a `/opt/hostedapp/node/root/app/yarn.lock` (no existe en Docker)

Solución: Regenerar yarn.lock como archivo real

```
cd app
rm yarn.lock  # Eliminar symlink
touch yarn.lock
yarn install  # Regenera yarn.lock
git add yarn.lock
git commit -m "fix: reemplazar symlink roto con archivo real"
git push
```

Verificación:

```
ls -lh app/yarn.lock
# -rw-r--r-- 1 ubuntu ubuntu 447K Nov 17 04:40 yarn.lock  ✓
# NO debe ser: lrwxrwxrwx (symlink)
```

Script de Verificación Automática

El script `pre-deploy-check.sh` verifica todos estos problemas antes del push:

```
bash pre-deploy-check.sh
```

Verifica:
- ✓ yarn.lock es archivo válido (no symlink)
- ✓ Prisma schema tiene enums requeridos
- ✓ Dockerfile usa Alpine 3.19
- ✓ Dockerfile genera Prisma client antes de build
- ✓ Dockerfile usa path directo a prisma CLI
- ✓ Scripts tienen permisos de ejecución

Pre-Push Hook Automático

El repositorio tiene un pre-push hook que ejecuta las verificaciones automáticamente:

```
.git/hooks/pre-push
```

Si alguna verificación falla, el push se bloquea y muestra el problema.

Última actualización: 2025-11-17 (Error #8 resuelto)

Error #11: Incompatibilidad Yarn Berry vs Yarn Classic

Síntoma:

```
yarn install --frozen-lockfile
error Your lockfile needs to be updated
⚠️ @prisma not found in node_modules
ERROR: "/app/node_modules": not found
```

Causa:
- yarn.lock generado con Yarn 4 (Berry) - `__metadata: version: 8`
- node:18-alpine tiene Yarn 1.x (Classic) preinstalado
- Yarn 1 no puede leer el formato de lockfile de Yarn 4

Solución: Cambiar a npm que es más compatible

```
# Antes (fallaba):
COPY app/yarn.lock ./
RUN yarn install --frozen-lockfile

# Después (funciona):
COPY app/package-lock.json ./
RUN npm ci
```

Verificación:

```
head -5 app/package-lock.json
# {
#   "name": "app",
#   "lockfileVersion": 3,  ✓
```

Última actualización: 2025-11-17 (Error #11 resuelto)

Error #12: Conflicto Peer Dependencies TypeScript ESLint

Síntoma:

```
npm ci
ERESOLVE could not resolve
@typescript-eslint/eslint-plugin@7.0.0 requires @typescript-eslint/parser@^6.0.0
Found: @typescript-eslint/parser@7.0.0
```

Causa:
- Conflicto entre versiones de @typescript-eslint/parser (7.0.0) y @typescript-eslint/eslint-plugin (7.0.0)
- package-lock.json tiene dependencias que no se resuelven con strict mode

Solución: Usar `--legacy-peer-deps`

```
RUN npm ci --legacy-peer-deps
```

Última actualización: 2025-11-17 (Error #12 resuelto)

Error #14: Test Enum Node.js Innecesario

Síntoma:

```
node -e "const { UserRole, StatusCuenta } = require('@prisma/client'); ..."
exit code: 1
```

Causa:
- Test de importación de enums con Node.js puede fallar en entorno Docker
- El grep ya verifica que los enums existen en index.d.ts
- Test redundante e innecesario

Solución: Eliminar test de Node.js, mantener solo grep

```
grep -c "export type UserRole" node_modules/.prisma/client/index.d.ts
```

Última actualización: 2025-11-17 (Error #14 resuelto)

Error #15: Grep Exit Code 1 (Validación Enums)

Síntoma:

```
grep -c "export type UserRole" node_modules/.prisma/client/index.d.ts
exit code: 1
```

Causa:
- `grep -c` retorna exit code 1 si no encuentra coincidencias
- Esto hace fallar todo el RUN command
- Validación de enums innecesariamente estricta

Solución: Simplificar validación - solo verificar que directorio existe

```
if [ -d "node_modules/.prisma/client/" ]; then
    ls -la node_modules/.prisma/client/ | head -10
    echo "✅ Prisma client directory exists"
else
    echo "❌ ERROR: Prisma client directory not found!"
    exit 1
fi
```

Última actualización: 2025-11-17 (Error #15 resuelto)
Total errores resueltos: 15

Error #16: package.json No Encontrado + Enums Prisma

Síntoma:

```
npm error path /app/package.json
npm error errno -2
npm error enoent Could not read package.json

error TS2305: Module '"@prisma/client"' has no exported member 'UserRole'
error TS2305: Module '"@prisma/client"' has no exported member 'StatusCuenta'
```

Causa:
- En stage builder: package.json no copiado (removido en fix #13)
- Cliente Prisma generado en RUN anterior, pero no disponible en siguiente RUN
- Docker RUN commands no comparten archivos generados entre layers
- Enums de Prisma no accesibles para TypeScript

Solución:
1. Copiar package.json y package-lock.json al stage builder
2. Regenerar cliente Prisma ANTES de `npm run build`

```
# Stage builder
COPY app/package.json app/package-lock.json ./
...
RUN ./node_modules/.bin/prisma generate && \
    npm run build
```

Última actualización: 2025-11-17 (Error #16 resuelto)
Total errores resueltos: 16

Error #17: Enums Prisma Persistentemente No Exportados

Síntoma:

```
error TS2305: Module '"@prisma/client"' has no exported member 'UserRole'
error TS2305: Module '"@prisma/client"' has no exported member 'StatusCuenta'
(Persiste después del fix #16)
```

Causa:
- npm ci ejecuta postinstall de Prisma automáticamente
- Prisma genera cliente ANTES de copiar schema.prisma al stage builder
- Cliente generado sin schema = sin enums
- Regeneración posterior no refresca correctamente por cache/conflicto

Solución:
1. Limpiar cliente generado viejo (node_modules/.prisma)
2. NO tocar runtime (@prisma/client instalado por npm)

3. Generar cliente limpio con schema correcto
4. Verificar enums en index.d.ts

```
# Limpiar solo el cliente generado
rm -rf node_modules/.prisma

# Generar limpio con schema completo
./node_modules/.bin/prisma generate --schema=./prisma/schema.prisma

# Verificar enums existen
grep "export.*UserRole" node_modules/.prisma/client/index.d.ts
```

Última actualización: 2025-11-17 (Error #17 resuelto)
Total errores resueltos: 17

Error #18: Prisma Genera en Ruta Absoluta Incorrecta

Síntoma:

```
✔ Generated Prisma Client to ./../home/ubuntu/muebleria_la_economica/app/
node_modules/.prisma/client
❌ ERROR: Prisma client directory not found!
```

Causa:
- schema.prisma tiene `output` hardcodeado a ruta absoluta del host
- En Docker, estamos en `/app`, no en `/home/ubuntu/muebleria_la_economica/app`
- Prisma genera cliente en ubicación incorrecta
- Verificación busca en `node_modules/.prisma/client/` (relativo) pero no existe

Solución:
Remover el `output` hardcodeado y dejar que Prisma use ubicación por defecto:

```
generator client {
    provider = "prisma-client-js"
    binaryTargets = ["native", "linux-musl-arm64-openssl-3.0.x"]
    # NO incluir output - usar default: node_modules/.prisma/client
}
```

Resultado: Cliente generado en `node_modules/.prisma/client/` (relativo, correcto)

Última actualización: 2025-11-17 (Error #18 resuelto)
Total errores resueltos: 18