Scripts de Validación Pre-Deploy

Scripts automatizados para detectar los 18 errores comunes documentados en `ERRORES-DEPLOY-RESUELTOS.md`.

📋 Scripts Disponibles

1. `pre-deploy-check.sh`

Validación completa antes de push/deploy (18 checks)

```
# Modo básico (solo errores críticos)
./pre-deploy-check.sh

# Modo estricto (incluye check de archivos sin commitear)
./pre-deploy-check.sh --strict

# Con verificación de conectividad GitHub
./pre-deploy-check.sh --check-remote
```

Salidas posibles:
- ✅ `exit 0` - Todo OK, listo para deploy
- ⚠️ `exit 0` - OK con advertencias (puedes continuar)
- ❌ `exit 1` - Errores críticos encontrados (NO deployar)

---

2. `install-git-hooks.sh`

Instala pre-push hook automático

```
./install-git-hooks.sh
```

Una vez instalado, cada `git push` ejecutará automáticamente `pre-deploy-check.sh`.

Para saltar validaciones (NO recomendado):

```
git push --no-verify
```

---

3. `cleanup-legacy-files.sh`

Remueve archivos legacy de Yarn

```
./cleanup-legacy-files.sh
git commit -m "chore: remover archivos legacy de yarn"
git push origin main
```

🔍 Validaciones Implementadas

Dependencias y Lock Files

1. ✅ `package-lock.json` existe y no es symlink (#13)
2. ⚠️ `yarn.lock` no existe (#12, #13, #14)
3. ⚠️ `.yarn/` directory no existe (#15)

Prisma

1. ✅ `schema.prisma` existe
2. ✅ NO tiene `output` hardcodeado (#18)
3. ✅ Todos los enums críticos presentes (`UserRole` , `StatusCuenta` , etc.)

Dockerfile

1. ✅ Usa Alpine 3.19 (NO 3.21) (#2, #3)
2. ✅ Usa `npm ci` (NO yarn) (#12, #13, #14)
3. ✅ NO instala yarn globalmente (#11)
4. ✅ NO copia `.yarn/` (#15)
5. ✅ Copia `package.json` al builder stage (#16)
6. ✅ Limpia prisma client antes de regenerar (#17)

Archivos Críticos

1. ✅ `next.config.js` existe
2. ✅ Imports de Prisma en `types.ts` (#16, #17, #18)

Git

1. ✅ Branch es `main`
2. ⚠️ No hay cambios sin commitear (solo con `--strict`)
3. ✅ Conectividad con GitHub (solo con `--check-remote`)

🚀 Workflow Recomendado

Opción A: Validación Manual

```
# Antes de cada push
./pre-deploy-check.sh

# Si todo OK, push a GitHub
git push origin main

# Coolify detecta cambios y hace deploy automático
```

Opción B: Validación Automática (Recomendado)

```
# Instalar hook una vez
./install-git-hooks.sh

# Ahora cada push ejecuta validaciones automáticamente
git push origin main
```

📊 Ejemplo de Salida

```
🔍 Pre-deploy check (18 validaciones)...
✓ package-lock.json existe (528K)
⚠ yarn.lock existe (ignorado en Docker) - ejecuta cleanup-legacy-files.sh
⚠ .yarn/ directory existe (ignorado en Docker) - ejecuta cleanup-legacy-files.sh
✓ Prisma schema OK (5 enums)
✓ Prisma schema sin output hardcodeado
✓ Todos los enums críticos presentes
✓ Dockerfile usa Alpine 3.19
✓ Dockerfile usa npm ci
✓ Dockerfile no instala yarn globalmente
✓ Dockerfile no copia .yarn/
✓ Dockerfile copia package.json al builder
✓ Dockerfile limpia prisma client antes de regenerar
✓ next.config.js existe
✓ Imports de Prisma en types.ts (5 enums)
✓ Branch: main

═══════════════════════════════════════
⚠️  VALIDACIÓN COMPLETADA CON ADVERTENCIAS
   Errores: 0 | Advertencias: 2
   Puedes continuar, pero revisa las advertencias
```

🐛 Troubleshooting

“yarn.lock existe” / “.yarn/ directory existe”

Solución:

```
./cleanup-legacy-files.sh
git commit -m "chore: remover archivos legacy de yarn"
git push origin main
```

“Dockerfile usa Alpine 3.21”

Solución: Cambiar a Alpine 3.19 en Dockerfile (repositorios 3.21 están rotos)

“Prisma schema tiene output hardcodeado”

Solución: Remover línea `output = "..."` del generator client en schema.prisma

Otros errores

Consulta `ERRORES-DEPLOY-RESUELTOS.md` para soluciones detalladas de cada error.

📝 Notas

- Los scripts están sincronizados con `ERRORES-DEPLOY-RESUELTOS.md` (18 errores)
- Advertencias (⚠️) no bloquean el deploy
- Errores (❌) DEBEN resolverse antes de deployar
- Hook de Git se puede saltar con `--no-verify` (NO recomendado)

Última actualización: 2025-11-17
Errores documentados: 18
Validaciones implementadas: 17