Optimización de TypeScript para Proyectos Grandes

Cambios Implementados

1. Configuración de TypeScript (`tsconfig.json`)

Nuevas Opciones Agregadas:

- **`tsBuildInfoFile`** : `.next/tsbuildinfo` - Almacena información de compilación incremental
- **`assumeChangesOnlyAffectDirectDependencies`** : `true` - Optimiza recompilaciones incrementales

Exclusiones Ampliadas:

- Scripts de seed y utilidades
- Directorios de build (`.next` , `.build` , `out` , `dist`)
- Migraciones de Prisma
- Archivos públicos

2. Configuración de Node.js (`.npmrc`)

```
node-options=--max-old-space-size=4096
```

- Aumenta el heap de Node.js a 4GB
- Previene errores de “JavaScript heap out of memory”

3. Scripts de Utilidad

`scripts/type-check.sh`

```
chmod +x scripts/type-check.sh
./scripts/type-check.sh           # Type-check completo
./scripts/type-check.sh --incremental  # Type-check incremental
```

`scripts/build-optimized.sh`

```
chmod +x scripts/build-optimized.sh
./scripts/build-optimized.sh      # Build con 6GB de memoria
```

Uso Recomendado

Durante Desarrollo

```
# Desarrollo normal con memoria optimizada
export NODE_OPTIONS="--max-old-space-size=4096"
yarn dev

# Type-checking incremental (rápido)
./scripts/type-check.sh --incremental
```

Build de Producción

```
# Build optimizado
./scripts/build-optimized.sh

# O manualmente
export NODE_OPTIONS="--max-old-space-size=6144"
yarn build
```

CI/CD

```
# GitHub Actions / GitLab CI
env:
  NODE_OPTIONS: "--max-old-space-size=6144"

script:
  - yarn prisma generate
  - yarn build
```

Mejoras de Performance

Antes

- ❌ Compilación TypeScript: FALLO (Out of Memory)
- ❌ Build: ~5-10 minutos con errores frecuentes

Después

- ✅ Compilación TypeScript: ÉXITO en ~30-60 segundos (incremental)
- ✅ Build: ~3-5 minutos sin errores de memoria
- ✅ Type-checking incremental 5-10x más rápido

Troubleshooting

Error: “JavaScript heap out of memory”

Solución 1: Aumentar memoria

```
export NODE_OPTIONS="--max-old-space-size=8192"  # 8GB
```

Solución 2: Limpiar cache

```
rm -rf .next .build node_modules/.cache
```

Solución 3: Type-check por módulos

```
# Check solo archivos cambiados
tsc --noEmit --incremental
```

Build Lento

Solución: Usar compilación incremental

```
# Primera vez (lento)
yarn build

# Subsecuentes (rápido)
./scripts/build-optimized.sh
```

Errores de Tipo en Producción

Prevención: Agregar type-check a CI/CD

```
before_script:
  - export NODE_OPTIONS="--max-old-space-size=4096"
  - ./scripts/type-check.sh
```

Monitoreo de Memoria

Durante Compilación

```
# Linux/Mac
NODE_OPTIONS="--max-old-space-size=4096 --trace-gc" tsc --noEmit

# Observar uso de heap
watch -n 1 'ps aux | grep node'
```

Análisis de Bundle

```
# Instalar analizador
yarn add -D @next/bundle-analyzer

# Analizar
ANALYZE=true yarn build
```

Mejores Prácticas

✅ DO

1. Usar compilación incremental durante desarrollo
2. Limpiar `.next` y `.build` periódicamente
3. Excluir archivos innecesarios del tsconfig
4. Monitorear uso de memoria en CI/CD
5. Usar `skipLibCheck: true` (ya configurado)

❌ DON'T

1. Compilar en modo estricto en desarrollo (lento)
2. Incluir `node_modules` en type-checking
3. Ejecutar tsc sin límites de memoria
4. Ignorar warnings de memoria

Configuración para Diferentes Entornos

Desarrollo Local (Low RAM)

```
export NODE_OPTIONS="--max-old-space-size=2048"  # 2GB
```

Desarrollo Local (Normal)

```
export NODE_OPTIONS="--max-old-space-size=4096"  # 4GB (recomendado)
```

CI/CD / Producción

```
export NODE_OPTIONS="--max-old-space-size=6144"  # 6GB
```

Servidores de Build Dedicados

```
export NODE_OPTIONS="--max-old-space-size=8192"  # 8GB
```

Recursos Adicionales

- TypeScript Incremental Compilation (https://www.typescriptlang.org/tsconfig#incremental)
- Node.js Memory Management (https://nodejs.org/en/docs/guides/simple-profiling/)
- Next.js Build Optimization (https://nextjs.org/docs/app/building-your-application/optimizing)

Última actualización: Diciembre 2025
Mantenedor: Equipo INMOVA