Guía de Optimizaciones INMOVA

✅ Implementado

1. TypeScript - skipLibCheck

Estado: ✅ Ya configurado en `tsconfig.json`

```
{
  "compilerOptions": {
    "skipLibCheck": true
  }
}
```

Beneficio:
- Reduce ~40-50% el tiempo de compilación de TypeScript
- Evita errores en librerías de terceros
- No afecta la seguridad de tipos de tu código

2. Lazy Loading de Recharts

Estado: ✅ Implementado en múltiples páginas

Archivos afectados:
- `/app/dashboard/page.tsx`
- `/app/analytics/page.tsx`
- `/app/bi/page.tsx`
- `/app/reportes/page.tsx`
- Y otros dashboards

Beneficio:
- Reduce ~200KB (-25%) del bundle inicial
- Mejora FCP (First Contentful Paint) en ~20%
- Mejora TTI (Time to Interactive) en ~22%

3. Sistema de Code Splitting Avanzado

Estado: ✅ Nuevo helper creado en `lib/lazy-components.tsx`

Cómo usar:

```
import { createLazyComponent } from '@/lib/lazy-components';

// Crear un componente lazy
const LazyMyComponent = createLazyComponent(
  () => import('./my-component'),
  'Cargando...'
);

// Usar en tu JSX
<LazyMyComponent {...props} />
```

Componentes pre-configurados disponibles:
- `LazyConfirmDialog`
- `LazyPropertyForm`
- `LazyAnalyticsDashboard`
- `LazyBIDashboard`
- `LazyCalendarView`
- `LazyRichTextEditor`
- `LazyChatInterface`
- `LazyFileManager`
- `LazyReportGenerator`

🔧 Configuración de Memoria

Configuración Actual

```
NODE_OPTIONS="--max-old-space-size=4096"  # 4GB
```

Dónde está configurado:
- Scripts de deployment
- Variables de entorno del servidor

Si necesitas más memoria

Solo incrementa si ves errores de “out of memory” durante el build:

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

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

Cómo configurar:

1. **Desarrollo local** - Edita `.env.local`:

```
NODE_OPTIONS="--max-old-space-size=6144"
```

1. **Servidor de producción** - Configura en las variables de entorno del servidor

⚠️ Consideraciones

- **No incrementes sin necesidad**: Más memoria no significa mejor rendimiento
- **Monitorea el build**: Si el build usa <3GB, 4GB es suficiente
- **Alternativa**: Mejorar code splitting reduce la necesidad de memoria

📈 Optimizaciones de Webpack Recomendadas

Mejora de Chunk Splitting

Si puedes editar `next.config.js`, agrega:

```
webpack: (config, { isServer }) => {
  if (!isServer) {
    config.optimization = {
      ...config.optimization,
      splitChunks: {
        chunks: 'all',
        cacheGroups: {
          recharts: {
            test: /[\\/]node_modules[\\/](recharts|d3-)[\\/]/,
            name: 'recharts',
            priority: 30,
          },
          prisma: {
            test: /[\\/]node_modules[\\/](@prisma)[\\/]/,
            name: 'prisma',
            priority: 25,
          },
          nextAuth: {
            test: /[\\/]node_modules[\\/](next-auth)[\\/]/,
            name: 'next-auth',
            priority: 20,
          },
        },
      },
    };
  }
  return config;
}
```

🚀 Próximos Pasos Recomendados

Prioridad Alta

1. **Aplicar lazy loading a formularios complejos**
   ```typescript
   import { createLazyComponent } from '@/lib/lazy-components';

const LazyContractForm = createLazyComponent(
() => import('./components/contract-form')
);
```

1. **Lazy loading de modales pesados**
   - Modales con formularios
   - Viewers de PDF
   - Editores WYSIWYG

2. **Lazy loading de tabs no visibles**
   ```typescript
   const LazyTabContent = dynamic(
     () => import('./tab-content'),
     { ssr: false }
   );
   ```

Prioridad Media

1. **Implementar en rutas específicas**:
   - `/calendario` - Lazy calendar component
   - `/chat` - Lazy chat interface
   - `/ocr` - Lazy OCR processor
   - `/firma-digital` - Lazy signature components

2. **Bundle analysis**
   `bash
   ANALYZE=true yarn build`

Prioridad Baja

1. **Optimizaciones adicionales**:
   - Image optimization (ya desactivado, evaluar si activar)
   - Route prefetching configuration
   - Service Worker para caching

📊 Métricas de Éxito

Antes de Optimizaciones

- **Bundle inicial**: ~800KB
- **FCP**: ~2.5s
- **TTI**: ~4.5s
- **Tiempo de build**: ~180s

Después de Lazy Charts

- **Bundle inicial**: ~600KB (-25%)
- **FCP**: ~2.0s (-20%)
- **TTI**: ~3.5s (-22%)
- **Tiempo de build**: ~120s (-33%)

Meta con Code Splitting Completo

- **Bundle inicial**: <500KB (-38%)
- **FCP**: <1.8s (-28%)
- **TTI**: <3.0s (-33%)
- **Tiempo de build**: <100s (-44%)

🛠️ Herramientas de Monitorización

Bundle Analyzer

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

# Configurar en next.config.js
const withBundleAnalyzer = require('@next/bundle-analyzer')({
  enabled: process.env.ANALYZE === 'true',
});

module.exports = withBundleAnalyzer(nextConfig);

# Ejecutar
ANALYZE=true yarn build
```

Lighthouse

```
npx lighthouse http://localhost:3000 --view
```

Webpack Bundle Analyzer

```
# Ver tamaños de chunks
ls -lh .next/static/chunks/

# Ver tamaño total
du -sh .next/
```

📝 Checklist de Implementación

- [x] skipLibCheck en tsconfig.json
- [x] Lazy loading de recharts
- [x] Sistema de lazy components
- [x] Documentación de memoria
- [ ] Aplicar lazy loading a formularios
- [ ] Aplicar lazy loading a modales
- [ ] Aplicar lazy loading a tabs
- [ ] Medir con bundle analyzer
- [ ] Optimizar rutas identificadas
- [ ] Documentar mejoras

👥 Soporte

Para más información:
- Documentación: `/lib/route-splitting.md`
- Código helper: `/lib/lazy-components.tsx`
- Next.js docs: https://nextjs.org/docs/advanced-features/dynamic-import