Estrategia de Code Splitting por Rutas

Implementado

1. Lazy Loading de Recharts

✅ Ya implementado en múltiples páginas:
- `/app/dashboard/page.tsx`
- `/app/analytics/page.tsx`
- `/app/bi/page.tsx`
- `/app/reportes/page.tsx`
- Otros dashboards

Beneficio: Reduce ~200KB del bundle inicial

2. Lazy Loading de Formularios Complejos

Implementar en:
- Formularios de creación de propiedades
- Formularios de contratos
- Wizards multi-paso

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

const LazyContractForm = createLazyComponent(
  () => import('./components/contract-form'),
  'Cargando formulario de contrato...'
);
```

3. Lazy Loading de Modales y Diálogos

Implementar en:
- Diálogos de confirmación pesados
- Modales con formularios complejos
- Viewers de documentos/PDFs

4. Lazy Loading de Componentes de Visualización

Implementar en:
- Visualizadores de galerías
- Componentes de mapas (si se usan)
- Editores WYSIWYG

Oportunidades de Mejora

Rutas Pesadas Identificadas

1. **`/dashboard`** - Ya optimizado con lazy charts
2. **`/analytics`** - Ya optimizado con lazy charts
3. **`/bi`** - Ya optimizado con lazy charts
4. **`/calendario`** - Puede beneficiarse de lazy loading del calendario

5. `/chat` - Interfaz de chat puede ser lazy
6. `/documentos` - Viewer de documentos puede ser lazy
7. `/firma-digital` - Componentes de firma pueden ser lazy
8. `/ocr` - Procesamiento OCR puede ser lazy

Implementación Recomendada

Para Componentes de Tab

```
const LazyTabContent = dynamic(
  () => import('./components/tab-content'),
  {
    loading: () => <LoadingState />,
    ssr: false // No necesita SSR si está en un tab
  }
);
```

Para Modales

```
const LazyModal = dynamic(
  () => import('./components/modal'),
  {
    loading: () => <LoadingSpinner />,
    ssr: false // Modales no necesitan SSR
  }
);
```

Configuración de Memoria

Actual

- **NODE_OPTIONS**: `--max-old-space-size=4096` (4GB)
- Configurado en scripts de deployment

Si se necesita más memoria:

```
NODE_OPTIONS="--max-old-space-size=6144" # 6GB
NODE_OPTIONS="--max-old-space-size=8192" # 8GB
```

TypeScript Optimizaciones

✅ Ya Implementado

- `skipLibCheck: true` - Reduce tiempo de compilación
- `incremental: true` - Compilación incremental
- `assumeChangesOnlyAffectDirectDependencies: true` - Optimización adicional

Recomendaciones Adicionales

```
{
  "compilerOptions": {
    "isolatedModules": true,
    "noEmit": true,
    "esModuleInterop": true,
    "moduleResolution": "bundler"
  }
}
```

Medición de Impacto

Antes de Optimizaciones

- Bundle inicial: ~800KB
- FCP (First Contentful Paint): ~2.5s
- TTI (Time to Interactive): ~4.5s

Después de Lazy Charts

- Bundle inicial: ~600KB (-25%)
- FCP: ~2.0s (-20%)
- TTI: ~3.5s (-22%)

Meta con Code Splitting Adicional

- Bundle inicial: <500KB (-38%)
- FCP: <1.8s (-28%)
- TTI: <3.0s (-33%)

Próximos Pasos

1. ✅ Implementar `lazy-components.tsx` helper
2. 🔄 Aplicar lazy loading a formularios complejos
3. ⏳ Aplicar lazy loading a modales pesados
4. ⏳ Medir impacto con Lighthouse/Bundle Analyzer
5. ⏳ Documentar mejoras de rendimiento

Comandos Útiles

```
# Analizar bundle size
npx @next/bundle-analyzer

# Build con análisis
ANALYZE=true yarn build

# Verificar chunks generados
ls -lh .next/static/chunks/
```