Guía de Optimización de Rendimiento - INMOVA

Estado Actual

✅ Optimizaciones Ya Implementadas

1. **Lazy Loading de Componentes Pesados**
   - ✅ Charts: `@/components/ui/lazy-charts-extended`
   - ✅ Dialogs: `@/components/ui/lazy-dialog`
   - ✅ Tabs: `@/components/ui/lazy-tabs`

2. **Optimización de Imágenes**
   - ✅ Uso de Next.js Image component
   - ✅ Aspect ratio containers
   - ✅ Lazy loading de imágenes fuera del viewport

3. **Memoización de Componentes**
   - ✅ KPICard component memoizado
   - ✅ DataTable con MemoizedTableRow
   - ✅ Componentes de tarjetas en listas

4. **Optimización de Base de Datos**
   - ✅ 8 índices compuestos agregados a Prisma schema
   - ✅ Queries optimizadas con select/include específicos

🎯 Objetivos de Rendimiento

Lighthouse Performance Goals

Métrica	Objetivo	Estado
Performance Score	> 80	⏳ En progreso
First Contentful Paint (FCP)	< 1.8s	⏳ En progreso
Time to Interactive (TTI)	< 3.8s	⏳ En progreso
Bundle Size (gzipped)	< 500KB	⏳ En progreso
API Response Time	< 500ms	⏳ En progreso

🚀 Plan de Implementación

Fase 1: Sistema de Caché con Redis (Completado)

Archivos Creados:
- `lib/redis.ts` - Cliente Redis y funciones de caché
- `lib/cache-helpers.ts` - Helpers para caché de recursos específicos

Funcionalidades:
- ✅ Conexión Redis con reconexión automática
- ✅ Caché con TTL configurable
- ✅ Invalidación de caché por patrón
- ✅ Helpers para dashboard, buildings, units, tenants, contracts, payments

Configuración Requerida:

```
# Añadir a .env
REDIS_URL=redis://localhost:6379
# o para Redis Cloud
REDIS_URL=redis://username:password@hostname:port
```

Instalación de Redis Local (Desarrollo):

```
# macOS
brew install redis
brew services start redis

# Ubuntu/Debian
sudo apt-get install redis-server
sudo systemctl start redis-server

# Docker
docker run -d -p 6379:6379 redis:alpine
```

Fase 2: Optimización de API Routes

APIs a Optimizar (Prioridad Alta)

1. **`/api/dashboard`** ⭐⭐⭐
   - Múltiples queries pesadas
   - Se accede en cada visita al dashboard
   - **Caché TTL recomendado:** 60 segundos

2. **`/api/buildings`** ⭐⭐⭐
   - Lista completa de edificios con unidades
   - **Caché TTL recomendado:** 300 segundos (5 min)

3. **`/api/units`** ⭐⭐⭐
   - Lista de unidades con relaciones
   - **Caché TTL recomendado:** 300 segundos

4. `/api/analytics` ⭐⭐
   - Cálculos complejos de estadísticas
   - **Caché TTL recomendado:** 1800 segundos (30 min)

5. `/api/payments` ⭐⭐
   - Consultas frecuentes para dashboard de pagos
   - **Caché TTL recomendado:** 60 segundos

Ejemplo de Implementación

Antes:

```
export async function GET(request: NextRequest) {
  const buildings = await prisma.building.findMany({
    where: { companyId },
    include: { units: true },
  });
  return NextResponse.json(buildings);
}
```

Después:

```
import { cachedBuildings } from '@/lib/cache-helpers';

export async function GET(request: NextRequest) {
  const buildings = await cachedBuildings(companyId, async () => {
    return prisma.building.findMany({
      where: { companyId },
      include: { units: true },
    });
  });
  return NextResponse.json(buildings);
}
```

Fase 3: Invalidación Inteligente de Caché

Cuándo invalidar el caché:

1. **Al crear/actualizar/eliminar recursos**

```
import { invalidateResourceCache } from '@/lib/cache-helpers';

// Después de crear un edificio
await prisma.building.create({ ... });
await invalidateResourceCache(companyId, 'buildings');
await invalidateResourceCache(companyId, 'dashboard');
```

1. **Recursos a invalidar por operación:**

Operación	Caché a Invalidar
Crear/Editar Edificio	`buildings` , `dashboard`
Crear/Editar Unidad	`units` , `buildings` , `dashboard`
Crear/Editar Contrato	`contracts` , `units` , `dashboard`
Crear/Editar Pago	`payments` , `dashboard`
Crear/Editar Inquilino	`tenants` , `dashboard`

Fase 4: Optimización de Queries Prisma

Malas Prácticas a Evitar

❌ **Cargar relaciones innecesarias:**

```
// Mal - carga todo
const buildings = await prisma.building.findMany({
  include: {
    units: true,  // Puede ser cientos de unidades
  },
});
```

✅ **Seleccionar solo lo necesario:**

```
// Bien - solo campos necesarios
const buildings = await prisma.building.findMany({
  select: {
    id: true,
    nombre: true,
    direccion: true,
    _count: {
      select: { units: true },
    },
  },
});
```

Queries Críticas a Optimizar

1. **Dashboard - Evitar múltiples queries:**

```
// Mal - múltiples queries secuenciales
const buildings = await prisma.building.findMany();
const units = await prisma.unit.findMany();
const contracts = await prisma.contract.findMany();

// Bien - usar Promise.all para paralelizar
const [buildings, units, contracts] = await Promise.all([
  prisma.building.findMany({ select: { id: true, nombre: true } }),
  prisma.unit.count({ where: { estado: 'ocupada' } }),
  prisma.contract.count({ where: { estado: 'activo' } }),
]);
```

1. **Agregar índices faltantes:**

```
// En schema.prisma - Añadir índices para queries frecuentes
model Payment {
  // ...
  @@index([estado, fechaVencimiento])
  @@index([contractId, fechaVencimiento])
}
```

Fase 5: Reducción de Bundle Size

Estrategias

1. **Dynamic Imports para rutas:**

```
// app/admin/page.tsx
import dynamic from 'next/dynamic';

const AdminDashboard = dynamic(() => import('@/components/admin/Dashboard'), {
  loading: () => <LoadingState />,
  ssr: false,
});
```

1. **Análisis de Bundle:**

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

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

module.exports = withBundleAnalyzer(nextConfig);

# Ejecutar análisis
ANALYZE=true yarn build
```

1. **Tree Shaking de librerías:**

```
// Mal
import _ from 'lodash';

// Bien
import debounce from 'lodash/debounce';
import throttle from 'lodash/throttle';
```

Fase 6: Optimización de Imágenes

Checklist de Imágenes

- [ ] Todas las imágenes usan Next.js Image component
- [ ] Imágenes optimizadas en formato WebP cuando sea posible
- [ ] Tamaños apropiados para diferentes breakpoints
- [ ] Lazy loading habilitado (por defecto en Next.js Image)
- [ ] Placeholder blur para mejor UX

```
import Image from 'next/image';

<Image
  src={imageSrc}
  alt="Descripción"
  fill
  className="object-cover"
  sizes="(max-width: 768px) 100vw, (max-width: 1200px) 50vw, 33vw"
  placeholder="blur"
  blurDataURL="data:image/jpeg..."
/>
```

Fase 7: Service Worker y PWA

Caché de Assets Estáticos

```
// public/sw.js
const CACHE_NAME = 'inmova-v1';
const urlsToCache = [
  '/',
  '/dashboard',
  '/static/css/main.css',
  '/static/js/main.js',
];

self.addEventListener('install', (event) => {
  event.waitUntil(
    caches.open(CACHE_NAME)
      .then((cache) => cache.addAll(urlsToCache))
  );
});

self.addEventListener('fetch', (event) => {
  event.respondWith(
    caches.match(event.request)
      .then((response) => response || fetch(event.request))
  );
});
```

📊 Métricas y Monitoreo

Herramientas de Medición

1. **Lighthouse CI**

```
# Instalar
yarn add -D @lhci/cli

# lighthouserc.json
{
  "ci": {
    "collect": {
      "url": ["http://localhost:3000/"],
      "numberOfRuns": 3
    },
    "assert": {
      "assertions": {
        "categories:performance": ["error", {"minScore": 0.8}],
        "first-contentful-paint": ["error", {"maxNumericValue": 1800}]
      }
    }
  }
}

# Ejecutar
lhci autorun
```

1. **Performance Monitoring en Producción**

```tsx
// app/layout.tsx
import { Analytics } from '@vercel/analytics/react';

export default function RootLayout({ children }) {
  return (
    <html>
      <body>
        {children}
        <Analytics />
      </body>
    </html>
  );
}
```

1. **API Response Time Tracking**

```tsx
// middleware.ts
import { NextResponse } from 'next/server';

export function middleware(request: Request) {
  const start = Date.now();
  const response = NextResponse.next();
  const duration = Date.now() - start;

  response.headers.set('X-Response-Time', `${duration}ms`);

  if (duration > 500) {
    console.warn(`Slow API: ${request.url} took ${duration}ms`);
  }

  return response;
}
```

🔧 Configuración de Redis

Opción 1: Redis Local (Desarrollo)

```bash
# Instalar y ejecutar
brew services start redis  # macOS
sudo systemctl start redis # Linux

# Verificar conexión
redis-cli ping
# Debe responder: PONG
```

Opción 2: Redis Cloud (Producción)

1. Crear cuenta en Redis Cloud (https://redis.com/try-free/)
2. Crear una base de datos
3. Copiar la URL de conexión
4. Agregar a `.env`:

```
REDIS_URL=redis://default:password@redis-12345.c1.us-east-1.amazonaws.com:12345
```

Opción 3: Upstash (Serverless)

1. Crear cuenta en Upstash (https://upstash.com/)
2. Crear base de datos Redis
3. Configurar:

```
REDIS_URL=https://your-db.upstash.io
REDIS_TOKEN=your-token
```

✅ Checklist de Implementación

Semana 1: Infraestructura

- [x] Instalar Redis
- [x] Crear `lib/redis.ts`
- [x] Crear `lib/cache-helpers.ts`
- [ ] Configurar variables de entorno
- [ ] Probar conexión Redis

Semana 2: APIs Críticas

- [ ] Optimizar `/api/dashboard`
- [ ] Optimizar `/api/buildings`
- [ ] Optimizar `/api/units`
- [ ] Optimizar `/api/analytics`
- [ ] Implementar invalidación de caché

Semana 3: Queries y Bundle

- [ ] Auditar y optimizar queries Prisma
- [ ] Analizar bundle size
- [ ] Implementar code splitting
- [ ] Optimizar imports de librerías

Semana 4: Medición y Ajustes

- [ ] Ejecutar Lighthouse CI
- [ ] Configurar monitoreo en producción
- [ ] Ajustar TTLs de caché según uso real
- [ ] Documentar mejoras y métricas

📈 Resultados Esperados

Mejoras de Rendimiento Estimadas

Métrica	Antes	Después	Mejora
Lighthouse Score	~65	>80	+23%
FCP	~2.5s	<1.8s	-28%
TTI	~5.0s	<3.8s	-24%
API Response (Dashboard)	~1200ms	<300ms	-75%
Bundle Size	~800KB	<500KB	-37%

Beneficios Esperados

1. **Experiencia de Usuario**
   - Carga de páginas más rápida
   - Transiciones suaves
   - Menor latencia en acciones

2. **Costos de Infraestructura**
   - Menor carga en base de datos
   - Reducción de queries redundantes
   - Mejor uso de recursos del servidor

3. **SEO y Core Web Vitals**
   - Mejor posicionamiento en búsquedas
   - Cumplimiento de Core Web Vitals
   - Mayor tasa de conversión

🆘 Troubleshooting

Redis no conecta

```
# Verificar que Redis está corriendo
redis-cli ping

# Ver logs de Redis
tail -f /usr/local/var/log/redis.log

# Reiniciar Redis
brew services restart redis  # macOS
sudo systemctl restart redis # Linux
```

Caché no se invalida

```
// Verificar invalidación manual
import { invalidateCache } from '@/lib/redis';

// Invalidar todo el caché de una empresa
await invalidateCache('company:COMPANY_ID:*');

// Verificar keys en Redis
// redis-cli
// KEYS company:*
```

API sigue lenta

1. Verificar logs de Redis (HIT vs MISS)
2. Revisar queries Prisma con `.$queryRaw`
3. Usar herramienta de profiling:

```
import { performance } from 'perf_hooks';

const start = performance.now();
// tu código
const end = performance.now();
console.log(`Tiempo: ${end - start}ms`);
```

📚 Referencias

- Next.js Performance (https://nextjs.org/docs/app/building-your-application/optimizing)
- Redis Best Practices (https://redis.io/docs/manual/patterns/)
- Prisma Performance (https://www.prisma.io/docs/guides/performance-and-optimization)
- Web Vitals (https://web.dev/vitals/)
- Lighthouse CI (https://github.com/GoogleChrome/lighthouse-ci)

🎯 Próximos Pasos

1. **Configurar Redis** en desarrollo y producción
2. **Implementar caché** en los 5 endpoints más críticos
3. **Medir baseline** con Lighthouse antes de optimizaciones
4. **Iterar y mejorar** basado en métricas reales
5. **Documentar hallazgos** y compartir con el equipo

Última actualización: Diciembre 2024
Versión: 1.0