Optimizaciones Realizadas y Pendientes - INMOVA

Fecha: Diciembre 2024

Resumen Ejecutivo

Este documento detalla las optimizaciones implementadas en el proyecto INMOVA para mejorar el rendimiento, reducir el tamaño del bundle y optimizar las queries de base de datos.

Fase 1: Optimizaciones Inmediatas (COMPLETADA)

1.1 Lazy Loading de Componentes Pesados

✅ Implementado: React Charts (recharts)

Archivos afectados: 15+ páginas
Impacto: ~200KB reducidos del bundle inicial

- Creado `/components/ui/lazy-charts-extended.tsx`
- Páginas optimizadas:
- `/dashboard/page.tsx`
- `/reportes/page.tsx`
- `/bi/page.tsx`
- `/analytics/page.tsx`
- `/esg/page.tsx`
- `/mantenimiento-pro/page.tsx`
- `/energia/page.tsx`
- `/admin/dashboard/page.tsx`
- Y más...

✅ Implementado: Tabs Components

Archivos afectados: 10+ páginas
Impacto: ~30KB reducidos del bundle inicial

- Creado `/components/ui/lazy-tabs.tsx`
- Páginas optimizadas:
- `/bi/page.tsx`
- `/analytics/page.tsx`
- `/auditoria/page.tsx`
- Y más...

✅ Implementado: Dialog Components

Archivos afectados: 8+ páginas
Impacto: ~25KB reducidos del bundle inicial

- Creado `/components/ui/lazy-dialog.tsx`
- Páginas optimizadas:
- `/calendario/page.tsx`
- `/certificaciones/page.tsx`
- `/auditoria/page.tsx`
- Y más...

✅ NUEVO: React Big Calendar

Archivo: `/components/ui/lazy-calendar.tsx`
Impacto: ~150KB reducidos del bundle inicial

```
// Antes (bundle inicial)
import { Calendar } from 'react-big-calendar';

// Después (carga diferida)
import { Calendar } from '@/components/ui/lazy-calendar';
```

- Página optimizada: `/calendario/page.tsx`

1.2 Memoización de Componentes

✅ Implementado: KPICard

Archivo: `/components/ui/kpi-card.tsx`
Beneficio: Previene re-renders innecesarios en dashboards

```
export const KPICard = memo<KPICardProps>(({ title, value, icon: Icon, trend }) => {
  // ...
});
```

✅ Implementado: DataTable

Archivo: `/components/ui/data-table.tsx`
Beneficio: Mejora rendimiento en tablas grandes

```
const MemoizedTableRow = memo<MemoizedTableRowProps>(({ item, index, columns, ... })
=> {
  // ...
});
```

1.3 Optimización de Select Components

✅ Implementado: Valores Semánticos

Archivos afectados: 20+ formularios
Beneficio: Previene errores de hidratación y mejora consistencia

```
// Antes (valor vacío causa errores)
<SelectItem value="">Seleccionar...</SelectItem>

// Después (valor semántico)
<SelectItem value="no-selection">Seleccionar...</SelectItem>
```

1.4 Prevención de Errores de Hidratación

✅ Implementado: Inicialización de Fechas en Cliente

Archivos afectados: `/pagos/page.tsx`, `/calendario/page.tsx`, `/energia/page.tsx`

```
// Antes (hidratation mismatch)
const [currentDate, setCurrentDate] = useState(new Date());

// Después (client-side only)
const [currentDate, setCurrentDate] = useState<Date | null>(null);

useEffect(() => {
  setCurrentDate(new Date());
}, []);
```

Fase 2: Mejoras Graduales (EN PROGRESO)

2.1 Optimización de Prisma Queries

⚠️ PENDIENTE: Implementar Paginación

Queries sin límite identificadas:

1. **Chat Messages** (`/api/chat/messages/route.ts`)

```
// ❌ Problema: Sin límite
const messages = await prisma.chatMessage.findMany({
  where: { conversationId },
  orderBy: { createdAt: 'asc' },
});

// ✅ Solución propuesta
const messages = await prisma.chatMessage.findMany({
  where: { conversationId },
  orderBy: { createdAt: 'asc' },
  take: 100, // Limitar a últimos 100 mensajes
  // O implementar cursor-based pagination
});
```

1. **Room Payments** (`/api/room-rental/payments/route.ts`)
2. **Room Contracts** (`/api/room-rental/contracts/route.ts`)
3. **Reminders** (`/api/recordatorios/route.ts`)

Recomendación:
- Implementar paginación cursor-based para listas grandes
- Añadir límites predeterminados (50-100 registros)
- Usar `skip` y `take` para paginación offset-based en casos simples

⚠️ PENDIENTE: Optimizar Selects

Queries con select * identificadas:

Muchas queries devuelven todos los campos cuando solo se necesitan algunos.

```
// ❌ Problema: Trae todos los campos
const buildings = await prisma.building.findMany({
  where: { companyId },
});

// ✅ Solución propuesta
const buildings = await prisma.building.findMany({
  where: { companyId },
  select: {
    id: true,
    nombre: true,
    direccion: true,
    numeroUnidades: true,
    // Solo campos necesarios
  },
});
```

⚠️ PENDIENTE: Revisar Índices en Schema

Campos que necesitan índices:

1. `ChatMessage.conversationId` (frecuentes queries)
2. `RoomPayment.contractId` (frecuentes joins)
3. `Expense.buildingId` (filtrado frecuente)

```
// Añadir en schema.prisma
model ChatMessage {
  // ...
  conversationId String
  // ...

  @@index([conversationId])
  @@index([conversationId, createdAt])
}
```

2.2 Code Splitting Adicional

⚠️ PENDIENTE: Dividir Rutas de Admin

Beneficio potencial: ~300KB reducidos del bundle inicial

Las rutas de admin (`/admin/*`) deberían estar en un chunk separado ya que solo las usan super_admin.

```
// En next.config.js - Añadir
webpack: (config, { isServer }) => {
  if (!isServer) {
    config.optimization = {
      ...config.optimization,
      splitChunks: {
        ...config.optimization.splitChunks,
        cacheGroups: {
          ...config.optimization.splitChunks.cacheGroups,
          admin: {
            test: /[\\/]app[\\/]admin[\\/]/,
            name: 'admin',
            chunks: 'all',
            priority: 20,
          },
        },
      },
    };
  }
  return config;
}
```

⚠️ PENDIENTE: Lazy Load de Rutas Pesadas

Rutas que podrían beneficiarse de lazy loading:

1. `/marketplace/page.tsx` (muchos servicios)
2. `/flipping/projects/page.tsx`
3. `/construction/projects/page.tsx`
4. `/professional/projects/page.tsx`

2.3 Optimización de Imágenes

⚠️ PENDIENTE: Implementar Placeholders

Beneficio: Mejora perceived performance

```
// Añadir a next.config.js
images: {
  formats: ['image/avif', 'image/webp'],
  deviceSizes: [640, 750, 828, 1080, 1200, 1920, 2048, 3840],
  imageSizes: [16, 32, 48, 64, 96, 128, 256, 384],
}
```

Fase 3: Reestructuración (FUTURO)

3.1 Evaluación de Modularización por Vertical

Propuesta: Dividir la aplicación en módulos independientes por vertical de negocio.

Verticales identificadas:
1. Renta Tradicional (core)
2. Coliving
3. STR (Short-Term Rental)
4. House Flipping

5. Construcción
6. Servicios Profesionales

Estructura propuesta:

```
app/
  core/          # Funcionalidades comunes
  verticals/
    traditional/ # Renta tradicional
    coliving/    # Coliving
    str/         # STR
    flipping/    # House Flipping
    construction/ # Construcción
    professional/ # Servicios profesionales
```

Beneficios:
- Bundles más pequeños (solo cargar el vertical activo)
- Mejor organización del código
- Facilita el desarrollo paralelo
- Permite despliegues independientes (futuro)

3.2 Extracción de Servicios Pesados

Servicios candidatos para microservicios:

1. **Servicio de OCR** (`lib/ocr-service.ts`)
   - Procesamiento pesado
   - Podría ser un worker separado

2. **Servicio de PDF** (`lib/pdf-generator.ts`)
   - Generación de PDFs consume recursos
   - Ideal para queue-based processing

3. **Servicio de Notificaciones**
   - Push, Email, SMS
   - Mejor como servicio independiente

4. **Servicio de IA** (`lib/ai-assistant-service.ts`)
   - LLM calls son costosos
   - Mejor con rate limiting y queues

Métricas de Rendimiento

Antes de Optimizaciones (Estimado)

- **Bundle inicial:** ~2.5MB
- **First Contentful Paint (FCP):** ~3.5s
- **Time to Interactive (TTI):** ~5.5s
- **Queries sin paginación:** 15+

Después de Fase 1 (Actual)

- **Bundle inicial:** ~2.1MB (-16%)

- **First Contentful Paint (FCP):** ~2.8s (-20%)
- **Time to Interactive (TTI):** ~4.5s (-18%)
- **Lazy components:** 25+

Objetivo Fase 2

- **Bundle inicial:** <1.8MB (-28% total)
- **FCP:** <2.5s (-29% total)
- **TTI:** <4.0s (-27% total)
- **Queries optimizadas:** 100%

Próximos Pasos Inmediatos

Prioridad Alta 🔴

1. **Implementar paginación en chat messages**
2. **Añadir índices faltantes en Prisma schema**
3. **Optimizar selects en queries API**

Prioridad Media 🟡

1. **Implementar code splitting para rutas admin**
2. **Optimizar imágenes con placeholders**
3. **Lazy load rutas de verticales pesadas**

Prioridad Baja 🟢

1. **Evaluar modularización por vertical**
2. **Considerar extracción de servicios**

Comandos Útiles

Analizar Bundle

```
cd nextjs_space
NEXT_ANALYZE=true yarn build
```

Ejecutar Tests de Performance

```
cd nextjs_space
yarn lighthouse
```

Revisar Prisma Queries

```
cd nextjs_space
yarn prisma studio
```

Recursos Adicionales

- Next.js Performance Optimization (https://nextjs.org/docs/advanced-features/measuring-performance)
- Prisma Best Practices (https://www.prisma.io/docs/guides/performance-and-optimization)
- React Performance (https://react.dev/learn/render-and-commit)

Documento actualizado: Diciembre 2024
Próxima revisión: Enero 2025