Guía de Análisis y Optimización de Bundles

Introducción

Esta guía explica cómo analizar y optimizar los bundles de JavaScript de la aplicación Inmova para mejorar el rendimiento.

Herramientas de Análisis

1. Next.js Bundle Analyzer

Ya instalado y configurado en el proyecto.

Ejecutar Análisis

```
# Analizar ambos bundles (cliente y servidor)
yarn analyze

# Analizar solo bundle del servidor
yarn analyze:server

# Analizar solo bundle del cliente
yarn analyze:browser
```

Esto generará:
- Archivo HTML interactivo con visualización de bundles
- Se abrirá automáticamente en el navegador
- Datos en `.next/analyze/`

2. Next.js Build Output

Cada build muestra tamaños de rutas:

```
yarn build
```

Output ejemplo:

```
Route (app)                              Size     First Load JS
┌─ ○ /                                    142 B          87.4 kB
├─ ○ /dashboard                          5.43 kB        92.8 kB
├─ ○ /buildings                          12.1 kB        99.4 kB
├─ λ /api/buildings                      0 B                0 B
└─ λ /api/contracts                      0 B                0 B

○  (Static)  prerendered as static content
λ  (Dynamic) server-rendered on demand
```

3. Lighthouse

```
# Instalar CLI
npm install -g @lhci/cli

# Ejecutar análisis
lighthouse https://inmova.app --view
```

Interpretar Resultados

Bundle Analyzer

🔴 Problemas Comunes

1. **Módulos Duplicados**
   - Mismo paquete aparece múltiples veces
   - **Solución**: Consolidar imports, usar alias

2. **Dependencias Pesadas**
   - Librerías > 100KB sin comprimir
   - **Solución**: Buscar alternativas más ligeras o lazy load

3. **Imports Innecesarios**
   - Importar librerías completas cuando solo se usan partes
   - **Solución**: Tree shaking, imports específicos

Tamaños Objetivo

Excelente ✅

- **First Load JS**: < 100 KB
- **Total Page Size**: < 500 KB
- **JavaScript**: < 200 KB

Aceptable ⚠️

- **First Load JS**: 100-150 KB
- **Total Page Size**: 500-800 KB
- **JavaScript**: 200-350 KB

Necesita Optimización 🔴

- **First Load JS**: > 150 KB
- **Total Page Size**: > 800 KB
- **JavaScript**: > 350 KB

Técnicas de Optimización

1. Tree Shaking

❌ Mal

```
// Importa TODO lodash (~70KB)
import _ from 'lodash';
const result = _.map(array, fn);
```

✅ **Bien**

```
// Solo importa map (~2KB)
import map from 'lodash/map';
const result = map(array, fn);
```

2. Dynamic Imports

❌ **Mal**

```
import Chart from 'chart.js'; // Siempre en bundle
```

✅ **Bien**

```
const Chart = dynamic(() => import('chart.js')); // Solo cuando se usa
```

3. Code Splitting por Ruta

Next.js lo hace automáticamente:

```
app/
  (public)/
    page.tsx        → Chunk separado
  dashboard/
    page.tsx        → Chunk separado
  buildings/
    page.tsx        → Chunk separado
```

4. Externals

Para dependencias que se cargan vía CDN:

```
// next.config.js
module.exports = {
  webpack: (config) => {
    config.externals = {
      ...config.externals,
      'react': 'React',
      'react-dom': 'ReactDOM',
    };
    return config;
  },
};
```

5. Remover console.log en Producción

```
// next.config.js
module.exports = {
  compiler: {
    removeConsole: process.env.NODE_ENV === 'production',
  },
};
```

6. Compresión

```
// next.config.js
module.exports = {
  compress: true, // Habilitar gzip
};
```

Dependencias Pesadas Comunes

Identificar Dependencias Pesadas

```
# Analizar tamaño de node_modules
npx cost-of-modules
```

Alternativas Ligeras

Pesada	Ligera	Ahorro
moment.js (232KB)	date-fns (20KB)	90%
lodash (71KB)	lodash-es (24KB)	66%
axios (13KB)	fetch nativo (0KB)	100%
chart.js (240KB)	recharts (100KB)	58%

Ejemplo: Reemplazar moment.js

Antes

```
import moment from 'moment';
const formatted = moment(date).format('DD/MM/YYYY');
```

Después

```
import { format } from 'date-fns';
const formatted = format(date, 'dd/MM/yyyy');
```

Ahorro: ~210KB

Optimización de Imágenes

1. Usar next/image

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

<Image
  src="/building.jpg"
  width={800}
  height={600}
  alt="Building"
  loading="lazy"
  quality={85}
/>
```

2. Formatos Modernos

```
// next.config.js
module.exports = {
  images: {
    formats: ['image/avif', 'image/webp'],
  },
};
```

Ahorro: 40-60% en tamaño de imagen

3. Responsive Images

```
<Image
  src="/hero.jpg"
  sizes="(max-width: 768px) 100vw, 50vw"
  fill
  priority
/>
```

Monitorización Continua

1. Performance Budget

Crear `.lighthouserc.json`:

```
{
  "ci": {
    "assert": {
      "assertions": {
        "first-contentful-paint": ["error", {"maxNumericValue": 2000}],
        "speed-index": ["error", {"maxNumericValue": 3000}],
        "interactive": ["error", {"maxNumericValue": 3500}],
        "total-byte-weight": ["error", {"maxNumericValue": 500000}]
      }
    }
  }
}
```

2. Webpack Bundle Analysis en CI

```yaml
# .github/workflows/bundle-analysis.yml
name: Bundle Analysis
on: [pull_request]

jobs:
  analyze:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Analyze bundle
        run: |
          yarn install
          yarn analyze
      - name: Upload bundle report
        uses: actions/upload-artifact@v2
        with:
          name: bundle-report
          path: .next/analyze/
```

3. Size Limit

Instalar:

```
yarn add --dev @size-limit/preset-next
```

Configurar `.size-limit.json`:

```json
[
  {
    "path": ".next/static/chunks/*.js",
    "limit": "200 KB"
  }
]
```

Ejecutar:

```
size-limit
```

Checklist de Optimización

Antes de Cada Release

- [ ] Ejecutar `yarn analyze` y revisar bundles
- [ ] Verificar que no hay dependencias duplicadas
- [ ] Confirmar que imágenes usan next/image
- [ ] Verificar lazy loading de componentes pesados
- [ ] Ejecutar Lighthouse y verificar score > 90
- [ ] Verificar First Load JS < 150KB
- [ ] Confirmar que console.log se remueve en producción
- [ ] Verificar compresión gzip habilitada

Optimizaciones Específicas de Inmova

Alta Prioridad

1. **Gráficos**: Lazy load Chart.js/Recharts
   ```
   typescript
   const RevenueChart = dynamic(() => import('@/components/charts/Revenue'));
   ```
2. **Mapas**: Lazy load y solo cliente
   ```
   typescript
   const MapView = dynamic(() => import('@/components/MapView'), {
     ssr: false,
   });
   ```
3. **PDF**: Lazy load jsPDF/PDFKit
   ```
   typescript
   const generatePDF = async () => {
     const { jsPDF } = await import('jspdf');
     // ...
   };
   ```
4. **Iconos**: Tree shake lucide-react
   ```typescript
   // ❌ Mal
   import * as Icons from 'lucide-react';

   // ✅ Bien
   import { Home, User, Settings } from 'lucide-react';
   ```

Media Prioridad

1. **Lodash**: Imports específicos
2. **Date libraries**: Usar date-fns en lugar de moment
3. **Forms**: Code split Formik/React Hook Form
4. **Tables**: Virtualizar tablas grandes

Recursos

- Next.js Bundle Analyzer (https://www.npmjs.com/package/@next/bundle-analyzer)
- Bundle Phobia (https://bundlephobia.com/) - Verificar tamaño de paquetes
- Import Cost (https://marketplace.visualstudio.com/items?itemName=wix.vscode-import-cost) - VSCode extension
- Lighthouse CI (https://github.com/GoogleChrome/lighthouse-ci)

Resultados Esperados

Antes de Optimizaciones

- First Load JS: 250 KB
- Total Bundle Size: 1.2 MB
- Lighthouse Score: 65

Después de Optimizaciones

- First Load JS: 120 KB (-52%)
- Total Bundle Size: 450 KB (-63%)
- Lighthouse Score: 92 (+42%)