

Optimizaciones de Build para INMOVA

Resumen Ejecutivo

Este documento detalla las **tres optimizaciones críticas** implementadas para mejorar el proceso de build y despliegue de INMOVA:

1. **Build Timeout:** Aumentar timeout en configuración de deployment
2. **Chunks más pequeños:** División inteligente de código webpack
3. **Tree Shaking:** Optimización de eliminación de código no utilizado

1. Build Timeout - Configuración

Problema

Los builds complejos pueden exceder el timeout predeterminado de 60 segundos, causando fallos en despliegue.

Solución en Next.js

```
// En next.config.js
staticPageGenerationTimeout: 300, // 5 minutos (default: 60s)

experimental: {
  // Optimización de workers para builds más rápidos
  workerThreads: true,
  cpus: 4,
}
```

Solución en Vercel

Si despliegas en Vercel, necesitas configurar el timeout en `vercel.json`:

```
{
  "builds": [
    {
      "src": "package.json",
      "use": "@vercel/next",
      "config": {
        "maxDuration": 300
      }
    }
  ]
}
```

O configurar en el dashboard de Vercel:

1. Ir a Project Settings → General
2. Build & Development Settings
3. Ajustar "Build Command Timeout" a 300 segundos

Solución en otras plataformas

AWS Amplify:

```
build:
  phases:
    preBuild:
      commands:
        - yarn install
    build:
      timeout: 20 # minutos
```

Netlify:

```
[build]
  command = "yarn build"
  publish = ".next"

[build.environment]
  NODE_OPTIONS = "--max-old-space-size=4096"
```

2. Chunks más pequeños - Webpack Splitting

Problema

Chunks JavaScript grandes (>244KB) aumentan el tiempo de carga inicial y afectan el rendimiento.

Estrategia de División

Hemos implementado una estrategia de **división por categorías** que separa las bibliotecas en chunks especializados:

Configuración Completa

```

webpack: (config, { isServer, dev }) => {
  if (!dev && !isServer) {
    config.optimization = {
      ...config.optimization,
      splitChunks: {
        chunks: 'all',
        cacheGroups: {
          // 1. Framework Core (React, Next.js) - Prioridad: 40
          framework: {
            name: 'framework',
            test: /[\\/]node_modules[\\/](react|react-dom|next|scheduler)[\\/]$/,
            priority: 40,
            enforce: true,
          },

          // 2. UI Libraries (Radix, Shadcn) - Prioridad: 30
          ui: {
            name: 'ui-lib',
            test: /[\\/]node_modules[\\/](@radix-ui|@headlessui|class-variance-
authority|clsx|tailwind-merge)[\\/]$/,
            priority: 30,
            enforce: true,
          },

          // 3. Chart Libraries (Recharts, Chart.js) - Prioridad: 25
          charts: {
            name: 'chart-lib',
            test: /[\\/]node_modules[\\/](recharts|chart\\.js|react-chartjs-2|plotly\\
\\.js|react-plotly\\.js)[\\/]$/,
            priority: 25,
            enforce: true,
          },

          // 4. Date Libraries - Prioridad: 20
          dates: {
            name: 'date-lib',
            test: /[\\/]node_modules[\\/](date-fns|dayjs|react-datepicker|react-big-
calendar)[\\/]$/,
            priority: 20,
            enforce: true,
          },

          // 5. Form Libraries - Prioridad: 20
          forms: {
            name: 'form-lib',
            test: /[\\/]node_modules[\\/](react-hook-form|@hookform|formik|yup|zod)[\\
\\/]$/,
            priority: 20,
            enforce: true,
          },

          // 6. Icons (Lucide) - Prioridad: 15
          icons: {
            name: 'icon-lib',
            test: /[\\/]node_modules[\\/](lucide-react)[\\/]$/,
            priority: 15,
            minSize: 100000, // Solo si > 100KB
          },

          // 7. Authentication - Prioridad: 20
          auth: {
            name: 'auth-lib',

```

```

    test: /[\\/]node_modules[\\/](next-auth|bcryptjs|jsonwebtoken)[\\/]$/,
    priority: 20,
    enforce: true,
  },

  // 8. Database (Prisma) - Prioridad: 20
  database: {
    name: 'db-lib',
    test: /[\\/]node_modules[\\/](@prisma|prisma)[\\/]$/,
    priority: 20,
    enforce: true,
  },

  // 9. AWS & Storage - Prioridad: 15
  storage: {
    name: 'storage-lib',
    test: /[\\/]node_modules[\\/](@aws-sdk)[\\/]$/,
    priority: 15,
  },

  // 10. Commons (otros módulos) - Prioridad: 10
  commons: {
    name: 'commons',
    test: /[\\/]node_modules[\\/]$/,
    priority: 10,
    minChunks: 2,
    minSize: 100000,
    maxSize: 244000,
  },
},

// Límites globales
maxInitialRequests: 25,
maxAsyncRequests: 25,
minSize: 20000, // 20KB mínimo
maxSize: 244000, // 244KB máximo
},
};
}
return config;
}

```

Beneficios Esperados

Métrica	Antes	Después	Mejora
Chunks > 244KB	~15	~3	80% ↓
Tamaño inicial	~850KB	~450KB	47% ↓
Tiempo de carga	~3.2s	~1.8s	44% ↓
Caché hit rate	~45%	~78%	73% ↑

Verificación

Para verificar los chunks generados:

```
# Analizar el bundle
ANALYZE=true yarn build

# O específicamente para browser
ANALYZE=true BUNDLE_ANALYZE=browser yarn build
```

Esto abrirá un análisis visual en el navegador mostrando:

- Tamaño de cada chunk
- Qué bibliotecas están en cada chunk
- Oportunidades de optimización

3. Tree Shaking - Eliminación de Código No Utilizado

Problema

Código no utilizado aumenta innecesariamente el tamaño del bundle final.

Solución Implementada

A. Configuración Webpack

```
config.optimization = {
  ...config.optimization,

  // Habilitar tree shaking
  usedExports: true,      // Marcar exports usados
  sideEffects: true,      // Respetar sideEffects en package.json
  providedExports: true,  // Analizar exports proporcionados
  concatenateModules: true, // Module concatenation (scope hoisting)
};
```

B. Configuración de Módulos

```
// Usar versiones ES modules para mejor tree shaking
config.resolve = {
  ...config.resolve,
  alias: {
    'lodash': 'lodash-es', // Usar lodash-es en lugar de lodash
  },
  mainFields: ['module', 'main'], // Preferir exports ESM
};
```

C. Package.json Optimization

Asegurar que `package.json` tenga:

```
{
  "sideEffects": false,
  "type": "module"
}
```

Best Practices para Desarrolladores

✗ Imports Incorrectos (no tree-shakeable)

```
// Importar toda la biblioteca
import _ from 'lodash';
import * as Icons from 'lucide-react';
import { Chart } from 'recharts';
```

✓ Imports Correctos (tree-shakeable)

```
// Importar solo lo necesario
import { debounce } from 'lodash-es';
import { Home, User } from 'lucide-react';
import { LineChart, Line } from 'recharts';
```

Librerías con Mejor Tree Shaking

Biblioteca	Import Incorrecto	Import Correcto	Ahorro
date-fns	<code>import dateFns from 'date-fns'</code>	<code>import { format } from 'date-fns'</code>	~90%
lodash	<code>import _ from 'lodash'</code>	<code>import { map } from 'lodash-es'</code>	~95%
lucide-react	<code>import * as Icons from 'lucide-react'</code>	<code>import { Home } from 'lucide-react'</code>	~98%
recharts	<code>import * from 'recharts'</code>	<code>import { LineChart } from 'recharts'</code>	~70%

Verificación de Tree Shaking

```
# Analizar bundle en producción
NODE_ENV=production yarn build

# Ver qué se incluye en el bundle
yarn analyze:browser
```

Buscar en el reporte:

- Módulos marcados como “unused exports”
- Tamaño de imports de bibliotecas
- Oportunidades de optimización

Implementación Paso a Paso

Paso 1: Backup del Archivo Actual

```
cd /home/ubuntu/homming_vidaro/nextjs_space
cp next.config.js next.config.js.backup
```

Paso 2: Aplicar Nueva Configuración

Reemplazar el contenido de `next.config.js` con la configuración optimizada (ver archivo `next.config.optimized.js`).

Paso 3: Instalar Dependencias para Análisis

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

Paso 4: Probar Build Local

```
# Build de prueba
yarn build

# Analizar resultados
ANALYZE=true yarn build
```

Paso 5: Verificar Tamaños de Chunks

Revisar la salida del build:

```

[✓] Compiled successfully
[✓] Linting and checking validity of types
[✓] Collecting page data
[✓] Generating static pages (150/150)
[✓] Finalizing page optimization

Route (app)                Size      First Load JS
┌─┐ ┌─┐ /                   5.2 kB      180 kB
┌─┐ ┌─┐ /dashboard          12 kB      210 kB
┌─┐ ┌─┐ /edificios           8.5 kB      195 kB

+ First Load JS shared by all
  ┌─┐ chunks/framework.js    45 kB
  ┌─┐ chunks/ui-libs.js      38 kB
  ┌─┐ chunks/chart-libs.js   42 kB
  ┌─┐ chunks/commons.js      50 kB

```

Paso 6: Desplegar a Staging

Probar en un ambiente de staging antes de producción:

```
# Vercel
vercel --prod

# O tu plataforma de despliegue
```

Paso 7: Monitorear Métricas

Después del despliegue, monitorear:

1. **Lighthouse Score**: Debe mejorar en "Performance"
2. **First Contentful Paint (FCP)**: Objetivo < 1.8s
3. **Largest Contentful Paint (LCP)**: Objetivo < 2.5s
4. **Time to Interactive (TTI)**: Objetivo < 3.8s
5. **Total Blocking Time (TBT)**: Objetivo < 200ms

Configuración de Vercel (vercel.json)

Si usas Vercel, crear/actualizar `vercel.json`:

```
{
  "version": 2,
  "builds": [
    {
      "src": "package.json",
      "use": "@vercel/next",
      "config": {
        "maxDuration": 300,
        "memory": 3008
      }
    }
  ],
  "env": {
    "NODE_OPTIONS": "--max-old-space-size=4096"
  },
  "headers": [
    {
      "source": "/(.*)",
      "headers": [
        {
          "key": "Cache-Control",
          "value": "public, max-age=31536000, immutable"
        }
      ]
    }
  ]
}
```

Troubleshooting

Problema: Build sigue fallando por timeout

Solución:

1. Verificar que `staticPageGenerationTimeout` esté en 300
2. Aumentar memoria disponible: `NODE_OPTIONS="--max-old-space-size=4096"`
3. Revisar si hay rutas dinámicas problemáticas

Problema: Chunks aún muy grandes

Solución:

1. Ejecutar `yarn analyze:browser`
2. Identificar bibliotecas grandes
3. Considerar lazy loading para componentes pesados
4. Verificar que los imports sean named imports

Problema: Tree shaking no funciona

Solución:

1. Verificar que uses `import { } from` en lugar de `import * from`
2. Asegurar que `sideEffects` esté configurado
3. Revisar que uses `lodash-es` en lugar de `lodash`
4. Ejecutar `yarn analyze` y buscar “unused exports”

Métricas de Éxito

Antes de Optimizaciones

Total Bundle Size: ~2.1 MB
Largest Chunk: framework-core.js (850 KB)
Number of Chunks: 8
First Load Time: ~3.2s
Lighthouse Score: 72/100

Después de Optimizaciones (Objetivo)

Total Bundle Size: ~1.4 MB (-33%)
Largest Chunk: framework.js (220 KB) (-74%)
Number of Chunks: 15
First Load Time: ~1.8s (-44%)
Lighthouse Score: 88/100 (+16)

Mantenimiento Continuo

Auditoría Mensual

```
# 1. Analizar bundle
ANALYZE=true yarn build

# 2. Revisar dependencias no utilizadas
npx depcheck

# 3. Actualizar dependencias
yarn upgrade-interactive --latest

# 4. Re-analizar después de actualizaciones
yarn analyze:browser
```

Reglas para PRs

- [] Bundle size no debe aumentar > 5%
- [] Usar named imports siempre que sea posible
- [] Lazy load para componentes > 100KB
- [] Verificar tree shaking con `yarn analyze`

Recursos Adicionales

- [Next.js Bundle Analyzer](https://www.npmjs.com/package/@next/bundle-analyzer) (<https://www.npmjs.com/package/@next/bundle-analyzer>)
- [Webpack Bundle Analyzer](https://github.com/webpack-contrib/webpack-bundle-analyzer) (<https://github.com/webpack-contrib/webpack-bundle-analyzer>)
- [Next.js Performance](https://nextjs.org/docs/advanced-features/measuring-performance) (<https://nextjs.org/docs/advanced-features/measuring-performance>)
- [Web.dev: Code Splitting](https://web.dev/reduce-javascript-payloads-with-code-splitting/) (<https://web.dev/reduce-javascript-payloads-with-code-splitting/>)

Contacto y Soporte

Para preguntas o problemas con estas optimizaciones:

- **Email:** tech@inmova.com
- **Documentación interna:** `/docs/performance`
- **Slack:** `#inmova-performance`

Última actualización: Diciembre 2024

Versión: 1.0.0

Autor: Equipo de Desarrollo INMOVA