🚀 GUÍA COMPLETA DE OPTIMIZACIÓN DE RENDIMIENTO

Grúas Equiser Website - gruasequiser.net

📊 OBJETIVOS DE RENDIMIENTO

✅ **Tiempo de carga:** < 2 segundos
✅ **Google PageSpeed Score:** 90+
✅ **First Contentful Paint (FCP):** < 1.8s
✅ **Largest Contentful Paint (LCP):** < 2.5s
✅ **Time to Interactive (TTI):** < 3.8s
✅ **Cumulative Layout Shift (CLS):** < 0.1
✅ **First Input Delay (FID):** < 100ms

🔧 OPTIMIZACIONES IMPLEMENTADAS

1. OPTIMIZACIÓN DE IMÁGENES ⚡

A. Conversión a WebP/AVIF

```
# Instalar herramientas de conversión
npm install -g sharp-cli

# Convertir todas las imágenes PNG/JPG a WebP
cd public/images
for file in *.{jpg,jpeg,png}; do
  sharp -i "$file" -o "${file%.*}.webp" --webp
done
```

B. Compresión de imágenes existentes

```
# Usar TinyPNG o Squoosh
# Online: https://tinypng.com/
# CLI: npm install -g tinypng-cli

# Comprimir todas las imágenes
tinypng public/images/*.{jpg,png}
```

C. Responsive Image Sets

- Implementar `srcset` para diferentes tamaños de pantalla
- Usar `sizes` attribute para especificar tamaños
- Implementar lazy loading nativo: `loading="lazy"`

Configuración en Next.js:

```
<Image
  src="/images/grua.png"
  alt="Grúa"
  width={800}
  height={600}
  sizes="(max-width: 768px) 100vw, (max-width: 1200px) 50vw, 33vw"
  quality={85}
  loading="lazy"
  placeholder="blur"
/>
```

---

2. OPTIMIZACIÓN DE CÓDIGO 💻

A. Minificación y Compresión

```
# Next.js ya incluye minificación automática en producción
npm run build

# Verificar tamaño de bundles
npx @next/bundle-analyzer
```

B. Code Splitting

- **Dynamic Imports:**

```
import dynamic from 'next/dynamic'

const HeavyComponent = dynamic(() => import('./HeavyComponent'), {
  loading: () => <LoadingSpinner />,
  ssr: false
})
```

C. Tree Shaking

- Importar solo lo necesario:

```
// ❌ Mal
import * as Icons from 'lucide-react'

// ✅ Bien
import { Menu, X, Phone } from 'lucide-react'
```

---

3. ESTRATEGIAS DE CACHING 🗄️

A. Browser Caching (.htaccess)

```
# Ver archivo .htaccess proporcionado
# Configurar cache para assets estáticos: 1 año
# Cache para HTML: sin cache
```

B. CDN Configuration

Opciones recomendadas:

1. **Cloudflare** (Gratis)
- Activar “Auto Minify” para HTML, CSS, JS
- Activar “Brotli” compression
- Configurar Page Rules para cache

1. **Vercel Edge Network** (Si desplegado en Vercel)
   - Cache automático de assets estáticos
   - Edge functions para SSR

C. Service Worker

```
// Implementar en public/sw.js
self.addEventListener('install', (event) => {
  event.waitUntil(
    caches.open('v1').then((cache) => {
      return cache.addAll([
        '/',
        '/styles/main.css',
        '/images/logo.png',
      ]);
    })
  );
});
```

---

4. OPTIMIZACIONES TÉCNICAS ⚙️

A. Reducir HTTP Requests

```
// Combinar múltiples CSS en uno
// Usar CSS Modules en Next.js
import styles from './Component.module.css'

// Sprite de iconos en lugar de múltiples SVGs
// Usar fuentes de sistema cuando sea posible
```

B. Preload Critical Resources

```
<!-- En app/layout.tsx -->
<head>
  <link rel="preload" href="/fonts/inter.woff2" as="font" type="font/woff2" crossOri-
gin="anonymous" />
  <link rel="preconnect" href="https://api.whatsapp.com" />
  <link rel="dns-prefetch" href="https://www.google.com" />
</head>
```

C. Optimización de Fuentes

```
// app/layout.tsx
import { Inter } from 'next/font/google'

const inter = Inter({
  subsets: ['latin'],
  display: 'swap',
  preload: true,
  variable: '--font-inter',
})
```

---

5. OPTIMIZACIÓN MÓVIL 📱

A. Mobile-First CSS

```
/* Estilos móvil primero */
.container {
  padding: 1rem;
}

/* Tablet y superior */
@media (min-width: 768px) {
  .container {
    padding: 2rem;
  }
}

/* Desktop */
@media (min-width: 1024px) {
  .container {
    padding: 3rem;
  }
}
```

B. Reducir JavaScript en Móvil

```
// Detectar dispositivo y cargar componentes específicos
const isMobile = /iPhone|iPad|iPod|Android/i.test(navigator.userAgent)

if (isMobile) {
  // Cargar versión ligera
} else {
  // Cargar versión completa
}
```

---

6. MONITOREO Y TESTING 📈

A. Herramientas de Testing

```
# Google PageSpeed Insights
https://pagespeed.web.dev/

# GTmetrix
https://gtmetrix.com/

# WebPageTest
https://www.webpagetest.org/

# Chrome DevTools Lighthouse
# Ejecutar desde Chrome DevTools > Lighthouse
```

B. Implementar Analytics de Rendimiento

```
// app/layout.tsx
export function reportWebVitals(metric) {
  console.log(metric)

  // Enviar a Google Analytics
  if (typeof window !== 'undefined' && window.gtag) {
    window.gtag('event', metric.name, {
      value: Math.round(metric.name === 'CLS' ? metric.value * 1000 : metric.value),
      event_label: metric.id,
      non_interaction: true,
    })
  }
}
```

C. Monitoring Continuo

Herramientas recomendadas:

1. **Google Search Console** - Core Web Vitals
2. **Vercel Analytics** - Real User Monitoring
3. **Sentry** - Error tracking y performance monitoring
4. **New Relic** - Application Performance Monitoring

🚀 PASOS DE IMPLEMENTACIÓN INMEDIATA

FASE 1: Optimizaciones Críticas (1-2 horas)

1. **Reemplazar next.config.js:**

```
cd /home/ubuntu/gruas_equiser_website/app
mv next.config.js next.config.old.js
mv next.config.optimized.js next.config.js
```

1. **Convertir imágenes a WebP:**

```
cd public/images
# Usar herramienta online o CLI para convertir
```

1. **Implementar lazy loading:**

```
// Reemplazar Image components con OptimizedImage
import { OptimizedImage } from '@/components/optimized-image'
```

1. **Rebuild del proyecto:**

```
npm run build
```

FASE 2: Optimizaciones de Caching (30 minutos)

1. **Subir .htaccess al servidor:**

```
# Copiar .htaccess a la raíz del sitio web
```

1. **Configurar Cloudflare (si aplica):**
   - Activar “Auto Minify”
   - Activar “Brotli”
   - Configurar Page Rules

FASE 3: Testing y Ajustes (1 hora)

1. **Ejecutar PageSpeed Insights:**

```
# Probar en:
https://pagespeed.web.dev/?url=https://gruasequiser.net
```

1. **Ejecutar GTmetrix:**

```
# Probar en:
https://gtmetrix.com/
```

1. **Ajustar según resultados**

📊 MÉTRICAS ESPERADAS POST-OPTIMIZACIÓN

Antes de Optimización:

- ⏱️ Tiempo de carga: 5-8 segundos
- 📊 PageSpeed Score: 40-60
- 📱 Mobile Score: 30-50

Después de Optimización:

- ⚡ Tiempo de carga: 1.5-2 segundos
- 🎯 PageSpeed Score: 90-95

- 📱 Mobile Score: 85-90
- ✅ Core Web Vitals: Todos en verde

🛠️ COMANDOS ÚTILES

```
# Analizar tamaño de bundles
npm run build && npx @next/bundle-analyzer

# Generar reporte de rendimiento
npx lighthouse https://gruasequiser.net --output html --output-path ./lighthouse-re-
port.html

# Monitorear tamaño de assets
du -sh public/images/*

# Optimizar imágenes en batch
find public/images -name "*.png" -exec convert {} -quality 85 {} \;
```

📞 SOPORTE Y MANTENIMIENTO

Monitoreo Continuo:

- Ejecutar PageSpeed Insights semanalmente
- Monitorear Core Web Vitals en Google Search Console
- Revisar logs de errores y performance

Actualizaciones Periódicas:

- Actualizar dependencias de Next.js mensualmente
- Revisar y optimizar nuevas imágenes subidas
- Auditar código para eliminar dependencias no usadas

🎯 CHECKLIST FINAL

- [ ] Next.config.js optimizado implementado
- [ ] Todas las imágenes convertidas a WebP
- [ ] Lazy loading implementado en imágenes
- [ ] .htaccess con headers de caching subido
- [ ] CDN configurado (Cloudflare)
- [ ] Fuentes optimizadas con `font-display: swap`
- [ ] Code splitting implementado
- [ ] Service Worker configurado (opcional)
- [ ] Analytics de rendimiento implementados
- [ ] Testing en PageSpeed Insights: Score 90+
- [ ] Testing en GTmetrix: Grade A
- [ ] Mobile testing completado

- [ ] Core Web Vitals en verde

📈 RESULTADOS ESPERADOS

Timeline de Implementación:
- ⚡ **Inmediato (1-2 horas):** Mejora de 30-40% en velocidad
- 🚀 **24 horas:** Score de PageSpeed 85+
- 🎯 **48 horas:** Score de PageSpeed 90-95
- ✅ **1 semana:** Core Web Vitals todos en verde

Impacto en el Negocio:
- 📈 Mejora en SEO y ranking de Google
- 💰 Mayor tasa de conversión (usuarios permanecen más tiempo)
- 📱 Mejor experiencia móvil = más leads
- ⚡ Carga rápida = menor tasa de rebote

¿Necesitas ayuda con alguna implementación específica? 🚀