MISIÓN CRÍTICA: Optimización Final de Bundle JavaScript
Score actual PageSpeed: 87 → Meta: 95-100
ESTADO ACTUAL DEL BUILD:
✅ vendor-react: 136 KB (React + ReactDOM) - Optimizado
✅ vendor-query: 3 KB (TanStack Query) - Optimizado
✅ vendor-icons: 13 KB (Lucide) - Optimizado
⚠️ vendor-charts: 257 KB (Recharts) - NECESITA LAZY LOADING
⚠️ vendor-misc: 395 KB (framer-motion + date-fns + otros) - NECESITA SEPARACIÓN
📦 index: 131 KB - Aceptable
OBJETIVOS DE ESTA OPTIMIZACIÓN:
1. Lazy load de componentes que usan Recharts (gráficos)
2. Separar framer-motion en chunk independiente
3. Separar date-fns en chunk independiente
4. Reducir vendor-misc a <100 KB
5. Reducir bundle inicial (index) a <100 KB
═══
TAREA 1: IDENTIFICAR COMPONENTES QUE USAN RECHARTS
═══
PASO 1.1: Ejecuta este comando y muéstrame el output completo:

```bash
grep -r "Chart\|recharts" client/src --include="*.tsx" --include="*.ts" -n | grep -v "node_modules"
```

PASO 1.2: Lista los archivos que importan Chart:

```
grep -r "import.*Chart" client/src --include="*.tsx" -l
```

PASO 1.3: Muestra el contenido de client/src/components/ui/chart.tsx (primeras 50 líneas):

```
head -50 client/src/components/ui/chart.tsx
```

═══ TAREA 2: APLICAR LAZY LOADING A COMPONENTES CON GRÁFICOS
═══
Basándote en los resultados de TAREA 1:
PASO 2.1: Identifica TODOS los componentes (.tsx) que importan el Chart UI component PASO 2.2: Para CADA archivo que importa Chart:

- Si el componente NO se usa en la página inicial (above-the-fold)
- Convierte el import a lazy loading con React.lazy()

EJEMPLO DE CONVERSIÓN:
ANTES (import estático):

```
import { SomeChartComponent } from "@/components/stats-chart";
export default function Dashboard() {
 return (
   <div>
     <SomeChartComponent data={data} />
   </div>
 );
}
```

DESPUÉS (lazy loading):

```
import { lazy, Suspense } from 'react';
const SomeChartComponent = lazy(() =>
import("@/components/stats-chart"));
export default function Dashboard() {
 return (
   <div>
     <Suspense fallback={<div className="h-96 animate-pulse bg-gray-100
dark:bg-gray-800 rounded-lg" />}>
       <SomeChartComponent data={data} />
     </Suspense>
   </div>
 );
}
```

IMPORTANTE: Solo aplica lazy loading si el componente:

- NO está en la sección hero
- NO está visible inmediatamente al cargar la página
- Es parte del dashboard de admin o secciones de estadísticas

═══════════════════════════════════ TAREA 3: ACTUALIZAR vite.config.ts CON SEPARACIÓN ADICIONAL ═══════════════════════════════════

PASO 3.1: Localiza la sección "manualChunks" en vite.config.ts
PASO 3.2: AGREGA estos chunks adicionales ANTES de la línea "if (id.includes('node_modules'))":

```
// 🔥 Framer Motion (animaciones - solo si se detecta uso)
if (id.includes('framer-motion')) {
 return 'vendor-animations';
}
// 🔥 Date-fns (fechas)
if (id.includes('date-fns')) {
 return 'vendor-dates';
}
// 🔥 React Icons (solo si se usa)
if (id.includes('react-icons')) {
 return 'vendor-react-icons';
}
// 🔥 Lodash (utilidades)
if (id.includes('lodash')) {
 return 'vendor-utils';
}
```

PASO 3.3: Guarda el archivo y ejecuta:

```
npm run build
```

PASO 3.4: Muestra el resultado de los chunks vendor:

```
ls -lh dist/public/assets/vendor-*.js
```

═══════════════════════════════════ TAREA 4: OPTIMIZAR IMPORTACIONES DE DATE-FNS (TREE-SHAKING)

```
═══════════════════════════════════════════════════════════════════════
PASO 4.1: Busca todas las importaciones de date-fns:
grep -r "from ['\"]date-fns" client/src --include="*.tsx"
--include="*.ts" -n
PASO 4.2: Para CADA archivo que importa date-fns:
```

- Verifica que use importaciones específicas (no import *)
- Asegúrate que sea: import { format } from 'date-fns';
- NO permitir: import * as dateFns from 'date-fns';

```
PASO 4.3: Si encuentras importaciones tipo "import *", corrígelas a importaciones específicas.
═══════════════════════════════════════════════════════════════════════
═══════ TAREA 5: VERIFICAR FRAMER-MOTION
═══════════════════════════════════════════════════════════════════════
PASO 5.1: Busca uso de framer-motion:
grep -r "framer-motion" client/src --include="*.tsx" -l | head -10
PASO 5.2: Si framer-motion se usa en componentes below-the-fold:
```

- Aplica lazy loading similar a los componentes con Charts
- Si solo se usa para animaciones pequeñas, considera usar CSS animations

```
═══════════════════════════════════════════════════════════════════════
═══════ TAREA 6: BUILD FINAL Y REPORTE
═══════════════════════════════════════════════════════════════════════
PASO 6.1: Ejecuta el build final:
npm run build
PASO 6.2: Genera reporte de todos los chunks:
ls -lh dist/public/assets/*.js | grep -E "(vendor|index)" | awk '{print
$5, $9}'
PASO 6.3: Calcula el tamaño total del bundle inicial:
du -ch dist/public/assets/index-*.js
dist/public/assets/vendor-react-*.js
dist/public/assets/vendor-query-*.js
dist/public/assets/vendor-icons-*.js | tail -1
PASO 6.4: Genera un archivo OPTIMIZATION-REPORT.md en la raíz con:
# Reporte de Optimización de Performance - grupotranservica.com
## Fecha: [FECHA ACTUAL]
## Chunks Generados (Build Final)
| Chunk | Tamaño | Tipo | Carga |
|-------|--------|------|-------|
| index-*.js | XX KB | Principal | Inmediata |
| vendor-react-*.js | XX KB | React Core | Inmediata |
| vendor-query-*.js | XX KB | TanStack | Inmediata |
| vendor-icons-*.js | XX KB | Lucide | Inmediata |
| vendor-charts-*.js | XX KB | Recharts | Lazy (bajo demanda) |
| vendor-animations-*.js | XX KB | Framer Motion | Lazy (bajo demanda)
|
| vendor-dates-*.js | XX KB | Date-fns | Lazy (bajo demanda) |
| vendor-misc-*.js | XX KB | Otros | Condicional |
| **TOTAL INICIAL** | **XX KB** | - | - |
```

```
## Mejoras Aplicadas
- ✅ Code splitting granular por librería
- ✅ Lazy loading de componentes con gráficos (Recharts)
- ✅ Lazy loading de todas las secciones below-the-fold
- ✅ Tree-shaking de date-fns
- ✅ Separación de framer-motion
- ✅ Minificación con Terser (2 passes)
- ✅ CSS code splitting
## Optimizaciones Restantes (Si Aplica)
[Lista cualquier optimización que no pudiste aplicar y por qué]
## Próximos Pasos
1. Deploy a producción
2. Esperar 3-5 minutos para propagación de caché
3. Test PageSpeed: https://pagespeed.web.dev/
4. Score esperado: 94-97 (antes 87)
```

═══════════════════════ RESTRICCIONES Y REGLAS ═══════════════════════

❌ NO toques archivos de configuración del servidor (server/*) ❌ NO modifiques vite.config.ts más allá de la sección manualChunks ❌ NO apliques lazy loading a Header, Footer, HeroSection ✅ SÍ aplica lazy loading a componentes de dashboard/admin ✅ SÍ mantén Suspense con fallbacks visuales apropiados ✅ SÍ verifica que el build no tenga errores antes de reportar

═══════════════════════ CRITERIOS DE ÉXITO ═══════════════════════

1. ✅ Bundle inicial (index + vendor-react + vendor-icons) < 170 KB total
2. ✅ vendor-charts carga solo cuando se necesita (no en bundle inicial)
3. ✅ vendor-misc reducido a < 150 KB
4. ✅ Build completa sin errores ni warnings críticos
5. ✅ OPTIMIZATION-REPORT.md generado con métricas reales

═══════════════════════ OUTPUT ESPERADO ═══════════════════════

Al finalizar, proporciona:

1. Lista de archivos modificados
2. Contenido de OPTIMIZATION-REPORT.md
3. Output completo de `ls -lh dist/public/assets/vendor-*.js`
4. Confirmación de que el build funciona sin errores
5. Tamaño total del bundle inicial

EJECUTA TODAS LAS TAREAS EN SECUENCIA. COMIENZA AHORA.

```
---
## 📋 DESPUÉS DE QUE EL AGENTE TERMINE
Cuando el agente complete, **tú deberás:**
1. **Revisar el OPTIMIZATION-REPORT.md** que genera
```

2. **Verificar los tamaños** de los chunks
3. **Hacer el deploy** a producción
4. **Esperar 3-5 minutos** para propagación
5. **Test PageSpeed:** https://pagespeed.web.dev/
6. **Reportarme el nuevo score**