🔍 AUDITORÍA TÉCNICA EXHAUSTIVA - INMOVA

Análisis de Problemas de Deployment y Memoria

📋 RESUMEN EJECUTIVO

🚨 Problema Principal Identificado

ERROR CRÍTICO DE BUILD: El deployment está fallando debido a un error de TypeScript que impide la compilación. Este es el motivo por el cual no se está desplegando la última versión.

💾 Problemas de Memoria Detectados

El proyecto tiene **130MB+ de dependencias pesadas** que están causando problemas durante el build:
- **Plotly.js**: 99MB (crítico)
- **PDF-parse**: 21MB
- **Recharts**: 7.8MB
- **Tesseract.js**: 1.7MB

🔴 PROBLEMAS CRÍTICOS (PRIORIDAD MÁXIMA)

1. Error de TypeScript que Impide el Build

Ubicación:
- `app/api/approvals/route.ts`
- `app/api/approvals/[id]/route.ts`

Error:

```
Type '{ userId: string; tipo: "alerta_sistema"; titulo: string; mensaje: string;
leida: false; }'
is not assignable to type 'NotificationCreateInput'
```

Causa: Falta el campo `companyId` requerido en el schema de Prisma.

Solución Inmediata:

```
// Agregar companyId al crear notificaciones:
await prisma.notification.create({
  data: {
    companyId: approval.companyId, // ✅ AGREGAR ESTE CAMPO
    userId: approval.createdById,
    tipo: 'alerta_sistema',
    titulo: '...',
    mensaje: '...',
    leida: false,
    prioridad: 'medio',
  }
});
```

Impacto: 🔴 **BLOQUEANTE** - Sin arreglar esto, no se puede hacer deployment.

---

2. Plotly.js (99MB) - Causa Principal de Problemas de Memoria

Ubicación: Instalado como dependencia pero aparentemente no utilizado en el código

Análisis:
- Búsqueda en código: 0 archivos encontrados usando plotly
- Peso: 99MB
- Está agregando ~25% del peso total del node_modules

Solución:

```
# Verificar si realmente se usa:
grep -r "plotly" app/ --include="*.tsx" --include="*.ts"

# Si no se usa, remover:
yarn remove plotly.js react-plotly.js @types/plotly.js @types/react-plotly.js
```

Impacto de Remoción:
- ✅ Reducción de 99MB en node_modules
- ✅ Menor uso de memoria durante build (~25% menos)
- ✅ Build más rápido

Impacto: 🔴 **CRÍTICO** - Principal causa de problemas de memoria

---

🟠 PROBLEMAS IMPORTANTES (PRIORIDAD ALTA)

3. Recharts sin Lazy Loading (6 archivos)

Archivos Afectados:
1. `app/dashboard/components/advanced-analytics.tsx`
2. `app/flipping/dashboard/page.tsx`
3. `app/admin/dashboard/page.tsx`
4. `app/admin/metricas-uso/page.tsx`
5. `app/str/dashboard/page.tsx`
6. `app/str/page.tsx`

Problema: Importación directa de recharts (7.8MB) aumenta el bundle inicial y el uso de memoria durante SSR.

Solución:

```
// ❌ ANTES:
import { LineChart, Line, BarChart, Bar } from 'recharts';

// ✅ DESPUÉS:
import dynamic from 'next/dynamic';

const RechartsComponents = dynamic(
  () => import('recharts').then((mod) => ({
    LineChart: mod.LineChart,
    Line: mod.Line,
    BarChart: mod.BarChart,
    Bar: mod.Bar,
    XAxis: mod.XAxis,
    YAxis: mod.YAxis,
    CartesianGrid: mod.CartesianGrid,
    Tooltip: mod.Tooltip,
    Legend: mod.Legend,
    ResponsiveContainer: mod.ResponsiveContainer,
  })),
  { ssr: false, loading: () => <div>Cargando gráfico...</div> }
);
```

Beneficios:
- ✅ Reduce bundle inicial en ~7.8MB
- ✅ Mejora tiempo de carga inicial
- ✅ Reduce uso de memoria en SSR

Impacto: 🟠 **ALTO** - Afecta rendimiento y uso de memoria

---

4. Memory Leaks Potenciales - useEffect sin Cleanup

Archivos con Potenciales Fugas:

Archivo	Effects	Cleanups	Riesgo
`app/chat/page.tsx`	4	1	🔴 Alto
`app/mantenimiento/page.tsx`	5	1	🔴 Alto
`app/admin/salud-sistema/page.tsx`	3	1	🟠 Medio
`app/admin/alertas/page.tsx`	3	1	🟠 Medio
`app/portal-proveedor/chat/page.tsx`	4	2	🟡 Bajo
`app/portal-inquilino/chat/page.tsx`	4	2	🟡 Bajo

Patrones Comunes de Fugas:

```
// ❌ FUGA: Interval sin limpiar
useEffect(() => {
  const interval = setInterval(() => {
    // código
  }, 1000);
  // ❌ Falta: return () => clearInterval(interval);
}, []);

// ❌ FUGA: Listener sin remover
useEffect(() => {
  window.addEventListener('resize', handleResize);
  // ❌ Falta: return () => window.removeEventListener('resize', handleResize);
}, []);

// ❌ FUGA: Subscription sin cancelar
useEffect(() => {
  const subscription = observable.subscribe();
  // ❌ Falta: return () => subscription.unsubscribe();
}, []);
```

Solución General:

```
// ✅ CORRECTO: Cleanup apropiado
useEffect(() => {
  const interval = setInterval(() => {
    // código
  }, 1000);

  return () => clearInterval(interval); // ✅ Cleanup
}, []);
```

Impacto: 🟠 **ALTO** - Puede causar consumo creciente de memoria en sesiones largas

---

5. Archivos Extremadamente Grandes

Top 10 Archivos Más Grandes:

Archivo	Líneas	Complejidad	Recomendación
`app/landing/page.tsx`	1,834	🔴 Crítica	Dividir en componentes
`app/admin/clientes/page.tsx`	1,364	🔴 Crítica	Dividir en módulos
`app/marketplace/page.tsx`	1,285	🔴 Crítica	Refactorizar
`app/votaciones/page.tsx`	1,239	🔴 Crítica	Extraer lógica
`app/mantenimiento/page.tsx`	1,229	🔴 Crítica	Dividir features
`app/contabilidad/page.tsx`	1,083	🟠 Alta	Modularizar
`app/admin/reportes-programados/page.tsx`	1,073	🟠 Alta	Refactorizar
`app/calendario/page.tsx`	1,019	🟠 Alta	Simplificar
`lib/bankinter-integration-service.ts`	949	🟠 Alta	Dividir servicios
`lib/room-rental-service.ts`	941	🟠 Alta	Extraer módulos

Problemas de Archivos Grandes:
- 🔴 Mayor uso de memoria durante transpilación
- 🔴 Dificultad de mantenimiento
- 🔴 Mayor probabilidad de conflictos en merge
- 🔴 Tiempo de compilación más lento

Recomendación: Priorizar refactorización de archivos >1000 líneas

Impacto: 🟠 **MEDIO-ALTO** - Afecta mantenibilidad y build time

---

🟡 PROBLEMAS MENORES (OPTIMIZACIONES)

6. Dependencias Pesadas Adicionales

PDF-parse (21MB)
- Uso: Procesamiento de PDFs
- Alternativa ligera: `pdf-lib` (4MB) si solo se necesita extracción básica
- Recomendación: Revisar si se puede lazy-load o usar alternativa

Tesseract.js (1.7MB)
- Uso: OCR de imágenes
- Recomendación: Lazy-load solo cuando se necesite OCR

Mammoth (en pdf-parse dependencies)
- Uso: Conversión de documentos Word
- Recomendación: Verificar si es necesario, considerar alternativas

---

7. Configuración de Build Subóptima

Actual `next.config.js`:

```
const nextConfig = {
  distDir: process.env.NEXT_DIST_DIR || '.next',
  output: process.env.NEXT_OUTPUT_MODE,
  experimental: {
    outputFileTracingRoot: path.join(__dirname, '../'),
  },
  eslint: { ignoreDuringBuilds: true },
  typescript: { ignoreBuildErrors: false }, // ✅ Bueno
  images: { unoptimized: true },
};
```

Optimizaciones Recomendadas:

```
const nextConfig = {
  // ... config existente ...

  // ✅ Agregar optimizaciones de webpack
  webpack: (config, { isServer }) => {
    // Optimizar bundle splitting
    if (!isServer) {
      config.optimization = {
        ...config.optimization,
        splitChunks: {
          chunks: 'all',
          cacheGroups: {
            recharts: {
              name: 'recharts',
              test: /[\\/]node_modules[\\/]recharts[\\/]/,
              priority: 10,
            },
            vendors: {
              name: 'vendors',
              test: /[\\/]node_modules[\\/]/,
              priority: -10,
            },
          },
        },
      };
    }
    return config;
  },

  // ✅ Optimizar imágenes (si es posible en producción)
  images: {
    unoptimized: false,
    domains: ['abacusai-apps-030d8be4269891ba0e758624-us-west-2.s3.us-
west-2.amazonaws.com'],
  },

  // ✅ Agregar compresión
  compress: true,

  // ✅ SWC minification (más rápido)
  swcMinify: true,
};
```

📊 ANÁLISIS DE IMPACTO Y PRIORIZACIÓN

Matriz de Prioridades

Problema	Impacto	Esfuerzo	Prioridad	ROI
1. Error TypeScript Build	🔴🔴🔴	🟢 Bajo	**P0**	⭐⭐⭐⭐⭐
2. Remover Plotly.js	🔴🔴🔴	🟢 Bajo	**P0**	⭐⭐⭐⭐⭐
3. Lazy Load Re-charts	🟠🟠	🟡 Medio	**P1**	⭐⭐⭐⭐
4. Fix Memory Leaks	🟠🟠	🟡 Medio	**P1**	⭐⭐⭐⭐
5. Refactorizar Archivos Grandes	🟠	🔴 Alto	**P2**	⭐⭐⭐
6. Optimizar De-pendencias	🟡	🟡 Medio	**P2**	⭐⭐⭐
7. Mejorar next.config	🟡	🟢 Bajo	**P3**	⭐⭐

✅ PLAN DE ACCIÓN INMEDIATO

Fase 1: Desbloqueador Crítico (1-2 horas)

✓ Paso 1: Arreglar Error de Build

```
# 1. Localizar archivos:
cd /home/ubuntu/homming_vidaro/nextjs_space
grep -r "alerta_sistema" app/api/approvals/

# 2. Agregar companyId en ambos archivos
# Ver solución en sección "Problema 1"

# 3. Verificar build:
yarn build
```

✓ Paso 2: Remover Plotly.js (si no se usa)

```
# 1. Verificar uso:
grep -r "plotly" app/ --include="*.tsx" --include="*.ts"

# 2. Si no hay resultados, remover:
yarn remove plotly.js react-plotly.js @types/plotly.js @types/react-plotly.js

# 3. Rebuild:
yarn build
```

Resultado Esperado:
- ✅ Build exitoso
- ✅ Reducción de ~99MB en memoria
- ✅ Deployment desbloqueado

Fase 2: Optimizaciones de Rendimiento (4-6 horas)

✓ Paso 3: Implementar Lazy Loading de Recharts

Para cada uno de los 6 archivos:
1. `app/dashboard/components/advanced-analytics.tsx`
2. `app/flipping/dashboard/page.tsx`
3. `app/admin/dashboard/page.tsx`
4. `app/admin/metricas-uso/page.tsx`
5. `app/str/dashboard/page.tsx`
6. `app/str/page.tsx`

Aplicar el patrón de lazy loading documentado en “Problema 3”.

✓ Paso 4: Fix Memory Leaks Críticos

1. `app/chat/page.tsx` (4 effects, 1 cleanup) 🔴
2. `app/mantenimiento/page.tsx` (5 effects, 1 cleanup) 🔴

Revisar y agregar cleanup functions apropiadas.

Resultado Esperado:
- ✅ Bundle inicial reducido
- ✅ Mejor tiempo de carga
- ✅ Menos fugas de memoria

Fase 3: Refactorización Gradual (2-4 semanas)

✓ Paso 5: Dividir Archivos Grandes

Priorizar top 5:
1. `app/landing/page.tsx` (1,834 líneas)
2. `app/admin/clientes/page.tsx` (1,364 líneas)
3. `app/marketplace/page.tsx` (1,285 líneas)
4. `app/votaciones/page.tsx` (1,239 líneas)
5. `app/mantenimiento/page.tsx` (1,229 líneas)

Estrategia:
- Extraer componentes reutilizables
- Mover lógica de negocio a hooks custom
- Dividir en sub-páginas si es necesario

✓ Paso 6: Optimizar next.config.js

Implementar configuración optimizada de la sección “Problema 7”.

📈 MÉTRICAS DE ÉXITO

Antes de las Optimizaciones

- 🔴 Build: **FALLANDO**
- 🔴 node_modules: ~400MB estimado
- 🔴 Memory leaks: 7 archivos con riesgo
- 🔴 Bundle inicial: Grande (recharts incluido)
- 🔴 Archivos >1000 líneas: 10+

Después de Fase 1

- ✅ Build: **EXITOSO**
- ✅ node_modules: ~300MB (-25%)
- 🟠 Memory leaks: 7 archivos con riesgo
- 🟠 Bundle inicial: Grande (recharts incluido)
- 🔴 Archivos >1000 líneas: 10+

Después de Fase 2

- ✅ Build: **EXITOSO**
- ✅ node_modules: ~300MB
- ✅ Memory leaks: 5 archivos con riesgo (-29%)
- ✅ Bundle inicial: Optimizado (-7.8MB)
- 🔴 Archivos >1000 líneas: 10+

Después de Fase 3

- ✅ Build: **EXITOSO Y OPTIMIZADO**
- ✅ node_modules: ~280MB (-30% total)
- ✅ Memory leaks: 2 archivos con riesgo (-71%)
- ✅ Bundle inicial: Altamente optimizado
- ✅ Archivos >1000 líneas: 5 (-50%)

🔧 COMANDOS ÚTILES PARA MONITOREO

Verificar Tamaño de Build

```
cd /home/ubuntu/homming_vidaro/nextjs_space
yarn build
du -sh .next
du -sh .next/static
```

Analizar Bundle

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

# En next.config.js:
const withBundleAnalyzer = require('@next/bundle-analyzer')({
  enabled: process.env.ANALYZE === 'true',
});

module.exports = withBundleAnalyzer(nextConfig);

# Analizar:
ANALYZE=true yarn build
```

Monitorear Memoria Durante Build

```
# Linux:
NODE_OPTIONS="--max-old-space-size=4096" yarn build

# Ver uso real:
/usr/bin/time -v yarn build 2>&1 | grep "Maximum resident"
```

Verificar Memory Leaks en Dev

```
# Iniciar con profiler:
node --inspect node_modules/.bin/next dev

# Abrir Chrome DevTools y hacer heap snapshots
```

📝 NOTAS ADICIONALES

Sobre el Deployment Actual

El script de deployment usa:

```
NODE_OPTIONS="--max-old-space-size=4096" yarn build
```

Esto significa:
- 4GB de heap máximo para Node.js durante build
- Si el build excede esto, falla con “Out of Memory”
- Con las optimizaciones propuestas, este límite debería ser suficiente

Consideraciones de Arquitectura

1. **Code Splitting**: Next.js ya hace code splitting automático por ruta
2. **Dynamic Imports**: Usar para componentes pesados (recharts, charts, etc.)
3. **Server Components**: Considerar migrar a App Router y usar RSC cuando sea posible
4. **Edge Runtime**: Para rutas que no necesitan Node.js completo

Recursos de Monitoreo Recomendados

- **Sentry**: Ya configurado, monitorear errores de memoria
- **Next.js Analytics**: Considerar activar para métricas de rendimiento
- **Lighthouse CI**: Agregar a CI/CD para detectar regresiones

---

🎯 CONCLUSIÓN

Problema Principal

El deployment está fallando por un **error de TypeScript** en la creación de notificaciones (falta campo `companyId`). Este es el bloqueador principal.

Problema Secundario

El uso de **Plotly.js (99MB)** está causando problemas de memoria durante el build, especialmente cuando se combina con otros componentes pesados.

Solución Inmediata

1. ✅ Arreglar error de TypeScript en approvals API
2. ✅ Remover Plotly.js si no se usa
3. ✅ Hacer deploy para verificar

Optimizaciones Recomendadas

1. 🔄 Lazy load recharts en 6 archivos
2. 🔄 Arreglar memory leaks en archivos críticos
3. 🔄 Refactorizar archivos grandes gradualmente

Tiempo estimado para desbloquear deployment: 1-2 horas
Tiempo estimado para optimizaciones completas: 2-3 semanas

📞 CONTACTO Y SEGUIMIENTO

Esta auditoría fue generada el: **5 de Diciembre, 2024**

Para implementar las soluciones o discutir prioridades, revisar este documento con el equipo técnico.

Próximos pasos recomendados:
1. ✅ Implementar Fase 1 (crítico)
2. ⏳ Planificar Fase 2 (alto impacto)
3. 📅 Roadmap para Fase 3 (mantenibilidad)

Fin del reporte técnico