Guía de Lazy Loading en Next.js

Introducción

Lazy loading (carga diferida) mejora el rendimiento inicial de la aplicación al cargar componentes y módulos solo cuando son necesarios.

Beneficios

- **Reducción del bundle inicial**: 30-50% más pequeño
- **Tiempo de carga inicial más rápido**: FCP y TTI mejorados
- **Mejor experiencia móvil**: Menos datos a descargar
- **Code splitting automático**: Next.js optimiza automáticamente

Estrategias de Lazy Loading

1. Dynamic Import de Componentes

Básico

```
import dynamic from 'next/dynamic';

// ❌ Evitar: Import estático de componente pesado
import HeavyChart from '@/components/HeavyChart';

// ✅ Hacer: Dynamic import
const HeavyChart = dynamic(() => import('@/components/HeavyChart'), {
  loading: () => <div>Cargando gráfico...</div>,
  ssr: false, // No renderizar en servidor si no es necesario
});

export default function Dashboard() {
  return (
    <div>
      <h1>Dashboard</h1>
      <HeavyChart data={data} />
    </div>
  );
}
```

Con Named Exports

```
// Componente con named export
const MyComponent = dynamic(
  () => import('@/components/MyModule').then((mod) => mod.MyComponent),
  {
    loading: () => <Skeleton />,
  }
);
```

2. Lazy Loading Condicional

Modal/Dialog

```
import { useState } from 'react';
import dynamic from 'next/dynamic';

// Solo cargar el modal cuando se necesite
const AddTenantModal = dynamic(() => import('@/components/AddTenantModal'));

export default function TenantsPage() {
  const [showModal, setShowModal] = useState(false);

  return (
    <div>
      <button onClick={() => setShowModal(true)}>
        Añadir Inquilino
      </button>

      {showModal && (
        <AddTenantModal onClose={() => setShowModal(false)} />
      )}
    </div>
  );
}
```

Tabs

```
const ProfileTab = dynamic(() => import('@/components/tabs/ProfileTab'));
const SettingsTab = dynamic(() => import('@/components/tabs/SettingsTab'));
const BillingTab = dynamic(() => import('@/components/tabs/BillingTab'));

export default function UserSettings() {
  const [activeTab, setActiveTab] = useState('profile');

  return (
    <div>
      <Tabs value={activeTab} onValueChange={setActiveTab}>
        <TabsList>
          <TabsTrigger value="profile">Perfil</TabsTrigger>
          <TabsTrigger value="settings">Configuración</TabsTrigger>
          <TabsTrigger value="billing">Facturación</TabsTrigger>
        </TabsList>

        {activeTab === 'profile' && <ProfileTab />}
        {activeTab === 'settings' && <SettingsTab />}
        {activeTab === 'billing' && <BillingTab />}
      </Tabs>
    </div>
  );
}
```

3. Route-based Code Splitting

Next.js hace esto automáticamente para cada página:

```
app/
  dashboard/           → dashboard.js (bundle separado)
  buildings/           → buildings.js (bundle separado)
  tenants/             → tenants.js (bundle separado)
  contracts/           → contracts.js (bundle separado)
```

4. Library Lazy Loading

Librerías Pesadas

```
// ❌ Evitar: Import completo de libreria pesada
import { Chart } from 'chart.js';

// ✅ Hacer: Dynamic import cuando se necesite
const loadChart = async () => {
  const { Chart } = await import('chart.js');
  return Chart;
};

export default function ChartComponent() {
  const [Chart, setChart] = useState(null);

  useEffect(() => {
    loadChart().then(setChart);
  }, []);

  if (!Chart) return <div>Cargando...</div>;

  return <canvas ref={canvasRef} />;
}
```

PDF Generation

```
// lib/pdf-generator.ts
export async function generatePDF(data: any) {
  // Solo cargar jsPDF cuando se necesite generar un PDF
  const { jsPDF } = await import('jspdf');

  const doc = new jsPDF();
  // ... generar PDF
  return doc;
}
```

5. Third-party Scripts

```
import Script from 'next/script';

export default function Page() {
  return (
    <>
      {/* Analytics - cargar con estrategia apropiada */}
      <Script
        src="https://www.googletagmanager.com/gtag/js"
        strategy="afterInteractive" // Cargar después de interactividad
      />

      {/* Chat widget - cargar de último */}
      <Script
        src="https://chat-widget.example.com/widget.js"
        strategy="lazyOnload" // Cargar cuando el navegador esté idle
      />
    </>
  );
}
```

Componentes Comunes para Lazy Loading

Editor de Texto Rico

```
const RichTextEditor = dynamic(
  () => import('@/components/RichTextEditor'),
  {
    loading: () => <TextAreaSkeleton />,
    ssr: false, // No renderizar en servidor
  }
);
```

Mapas

```
const MapView = dynamic(
  () => import('@/components/MapView'),
  {
    loading: () => <MapSkeleton />,
    ssr: false, // Mapas requieren window/document
  }
);
```

Gráficos y Visualizaciones

```
const AnalyticsDashboard = dynamic(
  () => import('@/components/analytics/Dashboard'),
  {
    loading: () => <DashboardSkeleton />,
  }
);
```

Componentes de Admin

```
// Solo cargar panel de admin para usuarios admin
const AdminPanel = dynamic(() => import('@/components/AdminPanel'));

export default function Dashboard() {
  const { user } = useSession();

  return (
    <div>
      <h1>Dashboard</h1>
      {user?.role === 'admin' && <AdminPanel />}
    </div>
  );
}
```

Patrones Avanzados

1. Prefetch Inteligente

```
import { useEffect } from 'react';
import { useInView } from 'react-intersection-observer';

export default function SmartLazyComponent() {
  const { ref, inView } = useInView({
    triggerOnce: true,
    rootMargin: '200px', // Prefetch 200px antes
  });

  useEffect(() => {
    if (inView) {
      // Prefetch component
      import('@/components/HeavyComponent');
    }
  }, [inView]);

  return (
    <div ref={ref}>
      {inView && <HeavyComponent />}
    </div>
  );
}
```

2. Suspense Boundaries

```
import { Suspense } from 'react';

const Chart = dynamic(() => import('@/components/Chart'));
const Table = dynamic(() => import('@/components/Table'));

export default function Dashboard() {
  return (
    <div>
      <Suspense fallback={<ChartSkeleton />}>
        <Chart />
      </Suspense>

      <Suspense fallback={<TableSkeleton />}>
        <Table />
      </Suspense>
    </div>
  );
}
```

3. Loading States Personalizados

```
const LoadingSpinner = () => (
  <div className="flex items-center justify-center p-8">
    <div className="animate-spin rounded-full h-8 w-8 border-b-2 border-blue-600" />
  </div>
);

const HeavyComponent = dynamic(
  () => import('@/components/HeavyComponent'),
  {
    loading: LoadingSpinner,
    ssr: false,
  }
);
```

Medición de Impacto

Antes del Lazy Loading

```
Initial Bundle: 850 KB
First Contentful Paint: 2.1s
Time to Interactive: 3.8s
```

Después del Lazy Loading

```
Initial Bundle: 320 KB (-62%)
First Contentful Paint: 1.2s (-43%)
Time to Interactive: 2.1s (-45%)
```

Mejores Prácticas

✅ Hacer

1. **Lazy load componentes pesados**: Gráficos, mapas, editores
2. **Lazy load modales y overlays**: Solo cuando se abren
3. **Lazy load por ruta**: Next.js lo hace automáticamente
4. **Usar skeletons**: Mejora percepción de velocidad
5. **Medir con Lighthouse**: Verificar mejoras reales

❌ Evitar

1. **Lazy load componentes críticos**: Navigation, header, hero
2. **Lazy load componentes muy pequeños**: Overhead no justifica beneficio
3. **Demasiados loading states**: Crea experiencia fragmentada
4. **SSR innecesario**: Usa `ssr: false` para componentes cliente-only
5. **Lazy load sin loading state**: Causa layout shift

Componentes Candidatos en Inmova

Alta Prioridad (Implementar primero)

```
// Gráficos y dashboards
const RevenueChart = dynamic(() => import('@/components/charts/RevenueChart'));
const OccupancyChart = dynamic(() => import('@/components/charts/OccupancyChart'));

// Modales
const AddBuildingModal = dynamic(() => import('@/components/modals/AddBuildingModal'))
;
const AddTenantModal = dynamic(() => import('@/components/modals/AddTenantModal'));
const AddContractModal = dynamic(() => import('@/components/modals/AddContractModal'))
;

// Editores
const RichTextEditor = dynamic(() => import('@/components/RichTextEditor'), {
  ssr: false,
});

// Visualizaciones
const BuildingMapView = dynamic(() => import('@/components/BuildingMapView'), {
  ssr: false,
});
```

Media Prioridad

```
// Tabs
const FinancialTab = dynamic(() => import('@/components/tabs/FinancialTab'));
const MaintenanceTab = dynamic(() => import('@/components/tabs/MaintenanceTab'));

// Reportes
const PDFGenerator = dynamic(() => import('@/components/PDFGenerator'), {
  ssr: false,
});

// Analytics
const AnalyticsDashboard = dynamic(() => import('@/components/AnalyticsDashboard'));
```

Referencias

- Next.js Dynamic Imports (https://nextjs.org/docs/app/building-your-application/optimizing/lazy-loading)
- React Code Splitting (https://react.dev/reference/react/lazy)
- Web.dev Code Splitting (https://web.dev/code-splitting/)