🔧 Guía de Integración de Optimizaciones

🎯 Resumen

Esta guía te muestra cómo integrar las optimizaciones en tu código existente.

1️⃣ Rate Limiting en APIs

Patrón Básico

```
import { applyRateLimit } from '@/lib/rate-limit';

export async function POST(request: NextRequest) {
  // SIEMPRE aplicar rate limiting PRIMERO
  const rateLimitResponse = await applyRateLimit(request, 'write');
  if (rateLimitResponse) {
    return rateLimitResponse; // 429 Too Many Requests
  }

  // Tu lógica aquí
  // ...
}
```

Tipos de Rate Limit

```
// Autenticación (login, registro): 5 intentos / 15 min
await applyRateLimit(request, 'auth');

// APIs estándar: 60 requests / minuto
await applyRateLimit(request, 'api');

// Lecturas: 100 requests / minuto
await applyRateLimit(request, 'read');

// Escrituras: 30 requests / minuto
await applyRateLimit(request, 'write');

// Operaciones costosas: 10 requests / hora
await applyRateLimit(request, 'expensive');
```

¿Dónde Aplicar?

✅ SÍ aplicar en:
- `/api/auth/*` - Autenticación (tipo: `auth`)
- APIs públicas sin autenticación (tipo: `api`)
- Endpoints de creación/actualización (tipo: `write`)
- Operaciones costosas como exports, reportes (tipo: `expensive`)

❌ NO aplicar en:
- Webhooks de terceros (Stripe, etc.)

- Health checks (`/api/health`)
- Assets estáticos

2️⃣ Logging Estructurado

Importar

```
import logger, {
  logError,
  logApiRequest,
  logPerformance,
  logSecurityEvent
} from '@/lib/logger';
```

Uso Básico

```
// Información general
logger.info('Usuario creado exitosamente', { userId, companyId });

// Advertencias
logger.warn('Intento de acceso a recurso no autorizado', { userId, resource });

// Errores
logger.error('Error al procesar pago', { error: error.message, paymentId });

// Debug (solo en desarrollo)
logger.debug('Procesando solicitud', { step: 1, data });
```

Funciones Especializadas

```
// Errores con contexto
try {
  await someOperation();
} catch (error) {
  logError(error as Error, {
    userId: session.user.id,
    action: 'create-building',
    buildingData: data,
  });
  // Manejar error...
}

// Requests de API
const start = Date.now();
// ... tu lógica
logApiRequest(
  'POST',
  '/api/buildings',
  session.user.id,
  Date.now() - start
);

// Performance
const start = Date.now();
const result = await expensiveOperation();
logPerformance('expensive-operation', Date.now() - start, {
  recordsProcessed: result.length,
});

// Eventos de seguridad
logSecurityEvent('failed-login-attempt', email, {
  ip: request.headers.get('x-forwarded-for'),
  userAgent: request.headers.get('user-agent'),
});
```

Patrón Completo en API

```
export async function POST(request: NextRequest) {
  const start = Date.now();
  const session = await getServerSession(authOptions);

  try {
    // Tu lógica
    const result = await createBuilding(data);

    // Log exitoso
    logApiRequest(
      'POST',
      '/api/buildings',
      session?.user?.id,
      Date.now() - start
    );

    return NextResponse.json(result);
  } catch (error) {
    // Log de error
    logError(error as Error, {
      endpoint: '/api/buildings',
      userId: session?.user?.id,
      data,
    });

    return NextResponse.json(
      { error: 'Error al crear edificio' },
      { status: 500 }
    );
  }
}
```

3️⃣ Caché con Redis

Configuración

```
# .env
REDIS_URL=redis://localhost:6379
# O Redis Cloud:
REDIS_URL=redis://user:password@host:port
```

Importar

```
import {
  withCache,
  getCached,
  setCached,
  deleteCached,
  CACHE_TTL
} from '@/lib/redis';
```

Uso Simple con `withCache`

```
// Automáticamente cachea y retorna
const stats = await withCache(
  'dashboard:stats:company-123',
  async () => {
    // Esta función solo se ejecuta si no hay caché
    return await fetchStatsFromDatabase();
  },
  CACHE_TTL.MEDIUM // 5 minutos
);
```

Uso Manual

```
// Obtener de caché
const cached = await getCached<DashboardStats>('dashboard:stats:company-123');

if (cached) {
  return cached; // Hit
}

// Miss - obtener de BD
const stats = await fetchStatsFromDatabase();

// Guardar en caché
await setCached('dashboard:stats:company-123', stats, CACHE_TTL.MEDIUM);

return stats;
```

Invalidación de Caché

```
// Eliminar una caché específica
await deleteCached('dashboard:stats:company-123');

// Eliminar todas las cachés de una compañía
await deletePattern('dashboard:*:company-123');

// Eliminar todas las cachés de dashboard
await deletePattern('dashboard:*');
```

¿Qué Cachear?

✅ **SÍ cachear:**
- Estadísticas de dashboard
- Listas de opciones (ej: tipos de edificio)
- Datos de configuración
- Reportes generados
- Datos que cambian poco

❌ **NO cachear:**
- Datos financieros críticos
- Información en tiempo real
- Datos de sesión
- Operaciones de escritura

TTL Recomendados

```
CACHE_TTL.SHORT      // 1 min  - Datos que cambian frecuentemente
CACHE_TTL.MEDIUM     // 5 min  - Dashboard, estadísticas
CACHE_TTL.LONG       // 30 min - Listas de opciones
CACHE_TTL.VERY_LONG  // 1 hora - Configuración, datos casi estáticos
CACHE_TTL.DAY        // 24h    - Datos históricos
```

Patrón Completo

```
export async function GET(request: NextRequest) {
  const session = await getServerSession(authOptions);
  const companyId = session?.user?.companyId;

  const stats = await withCache(
    `dashboard:stats:${companyId}`,
    async () => {
      logger.info('Cache miss - fetching from DB', { companyId });

      const [buildings, units, tenants] = await Promise.all([
        prisma.building.count({ where: { companyId } }),
        prisma.unit.count({ where: { building: { companyId } } }),
        prisma.tenant.count({ where: { companyId } }),
      ]);

      return { buildings, units, tenants };
    },
    CACHE_TTL.MEDIUM
  );

  return NextResponse.json(stats);
}

// Invalidar cuando se crean/actualizan/eliminan registros
export async function POST(request: NextRequest) {
  // ... crear edificio

  // Invalidar caché del dashboard
  await deleteCached(`dashboard:stats:${companyId}`);

  return NextResponse.json(building);
}
```

4️⃣ Health Checks

Endpoints Disponibles

```
// Health check completo
GET /api/health

// Liveness (está vivo?)
GET /api/health/liveness

// Readiness (listo para tráfico?)
GET /api/health/readiness
```

Integración con Monitoring

```
// Configurar en tu sistema de monitoreo:
const response = await fetch('https://tu-app.com/api/health');
const health = await response.json();

if (health.status !== 'healthy') {
  // Enviar alerta
  alerting.send('App unhealthy', health);
}
```

Docker Health Check

```
HEALTHCHECK --interval=30s --timeout=3s --start-period=40s --retries=3 \
  CMD curl -f http://localhost:3000/api/health/liveness || exit 1
```

Kubernetes Probes

```
livenessProbe:
  httpGet:
    path: /api/health/liveness
    port: 3000
  initialDelaySeconds: 30
  periodSeconds: 10

readinessProbe:
  httpGet:
    path: /api/health/readiness
    port: 3000
  initialDelaySeconds: 5
  periodSeconds: 5
```

5️⃣ Ejemplo Completo: API Optimizada

```
import { NextRequest, NextResponse } from 'next/server';
import { getServerSession } from 'next-auth';
import { authOptions } from '@/lib/auth-options';
import { applyRateLimit } from '@/lib/rate-limit';
import { withCache, CACHE_TTL, deleteCached } from '@/lib/redis';
import { prisma } from '@/lib/db';
import logger, { logError, logApiRequest, logPerformance } from '@/lib/logger';

export const dynamic = 'force-dynamic';

// GET: Listar edificios (con caché)
export async function GET(request: NextRequest) {
  const startTime = Date.now();

  // 1. Rate limiting
  const rateLimitResponse = await applyRateLimit(request, 'read');
  if (rateLimitResponse) return rateLimitResponse;

  // 2. Autenticación
  const session = await getServerSession(authOptions);
  if (!session?.user?.companyId) {
    return NextResponse.json({ error: 'No autorizado' }, { status: 401 });
  }

  const { companyId, id: userId } = session.user;

  try {
    // 3. Caché
    const buildings = await withCache(
      `buildings:list:${companyId}`,
      async () => {
        logger.info('Fetching buildings from database', { companyId });

        return await prisma.building.findMany({
          where: { companyId },
          include: { units: { select: { id: true } } },
          orderBy: { createdAt: 'desc' },
        });
      },
      CACHE_TTL.MEDIUM
    );

    // 4. Log de performance
    logPerformance('get-buildings', Date.now() - startTime, {
      companyId,
      count: buildings.length,
    });

    return NextResponse.json(buildings);
  } catch (error) {
    logError(error as Error, { endpoint: '/api/buildings', userId });
    return NextResponse.json(
      { error: 'Error al obtener edificios' },
      { status: 500 }
    );
  }
}

// POST: Crear edificio
export async function POST(request: NextRequest) {
  const startTime = Date.now();
```

```
  // 1. Rate limiting (más estricto para escrituras)
  const rateLimitResponse = await applyRateLimit(request, 'write');
  if (rateLimitResponse) return rateLimitResponse;

  // 2. Autenticación
  const session = await getServerSession(authOptions);
  if (!session?.user?.companyId) {
    return NextResponse.json({ error: 'No autorizado' }, { status: 401 });
  }

  const { companyId, id: userId } = session.user;

  try {
    const body = await request.json();

    // 3. Validación
    if (!body.nombre || !body.direccion) {
      return NextResponse.json(
        { error: 'Nombre y dirección son requeridos' },
        { status: 400 }
      );
    }

    // 4. Crear edificio
    const building = await prisma.building.create({
      data: {
        ...body,
        companyId,
      },
    });

    // 5. Invalidar caché
    await Promise.all([
      deleteCached(`buildings:list:${companyId}`),
      deleteCached(`dashboard:stats:${companyId}`),
    ]);

    // 6. Log
    logger.info('Building created', { buildingId: building.id, companyId, userId });
    logApiRequest('POST', '/api/buildings', userId, Date.now() - startTime);

    return NextResponse.json(building, { status: 201 });
  } catch (error) {
    logError(error as Error, {
      endpoint: '/api/buildings',
      userId,
      body: request.body,
    });

    return NextResponse.json(
      { error: 'Error al crear edificio' },
      { status: 500 }
    );
  }
}
```

6️⃣ Checklist de Integración

Por cada API endpoint:

- [ ] ✅ Rate limiting aplicado
- [ ] ✅ Logging de requests
- [ ] ✅ Logging de errores con contexto
- [ ] ✅ Logging de performance
- [ ] ✅ Caché para GETs (si aplica)
- [ ] ✅ Invalidación de caché para POST/PUT/DELETE
- [ ] ✅ Manejo de errores apropiado
- [ ] ✅ Autenticación verificada

Configuración:

- [ ] Variables de entorno configuradas
- [ ] Sentry configurado
- [ ] Redis configurado (opcional)
- [ ] Scripts de package.json agregados
- [ ] Índices de BD agregados
- [ ] Tests ejecutándose correctamente

7️⃣ Recursos Adicionales

Archivos de Ejemplo

- `app/api/auth/login/rate-limited-example.ts` - Rate limiting en auth
- `app/api/dashboard/stats-cached-example/route.ts` - Dashboard con caché

Documentación

- `OPTIMIZACIONES_IMPLEMENTADAS.md` - Documentación completa
- `AGREGAR_INDICES_DB.md` - Guía de índices de BD

Testing

```
# Ejecutar tests
yarn test

# Con cobertura
yarn test:coverage

# Optimizar BD
yarn tsx scripts/optimize-database.ts
```

¡Listo! Ahora tienes todas las herramientas para una aplicación robusta y optimizada. 🚀