

Análisis Técnico y Recomendaciones: Aurum Control Center

Sistema ERP Centralizado para Holding Multi-Satélite

Fecha de Análisis: 12 de Diciembre, 2025

Alcance: Arquitectura, Seguridad, UX/UI, Mejoras Técnicas, Roadmap

Criticidad: Sistema de Misión Crítica para Operaciones de Negocio



Resumen Ejecutivo

Aurum Control Center es un ERP centralizado diseñado para gestionar un holding con 11 empresas satélite distribuidas en 6 sectores verticales. El sistema implementa un stack moderno (Next.js 14, TypeScript, PostgreSQL, Prisma) con integraciones webhook a través de n8n hacia sistemas críticos (Odoo, Chatwoot, WHMCS, Contabo, WAHA).

Fortalezas Principales

- ✓ Stack tecnológico moderno y bien fundamentado
- ✓ Modelo de permisos basado en roles (RBAC) bien definido
- ✓ Arquitectura de integraciones desacoplada vía webhooks
- ✓ Schema de base de datos normalizado y extensible
- ✓ Separación clara de responsabilidades por módulos

Áreas Críticas de Mejora

- ⚠ Seguridad: Falta de capas de protección enterprise-grade
- ⚠ Resiliencia: Ausencia de estrategias de fallback en integraciones
- ⚠ Monitoreo: No hay observabilidad del sistema en producción
- ⚠ Testing: Falta de cobertura automatizada
- ⚠ Performance: Necesidad de caching y optimización de queries

1. Evaluación de Arquitectura

1.1 Fortalezas del Diseño Actual

- ✓ Stack Tecnológico Sólido

```
// Stack bien elegido para un sistema ERP moderno
Stack: {
  Frontend: "Next.js 14 (App Router + Server Components)",
  Language: "TypeScript (Type Safety)",
  Database: "PostgreSQL (ACID compliance)",
  ORM: "Prisma (Type-safe queries)",
  UI: "Tailwind + ShadcnUI (Consistent design system)",
  Integration: "n8n (Low-code automation)"
}
```

Justificación:

- **Next.js 14** ofrece SSR, ISR y streaming para UX óptima
- **TypeScript** reduce bugs en producción en ~30-40%
- **PostgreSQL** provee transacciones ACID críticas para finanzas
- **Prisma** genera tipos automáticos y previene SQL injection
- **ShadcnUI** acelera desarrollo con componentes accesibles

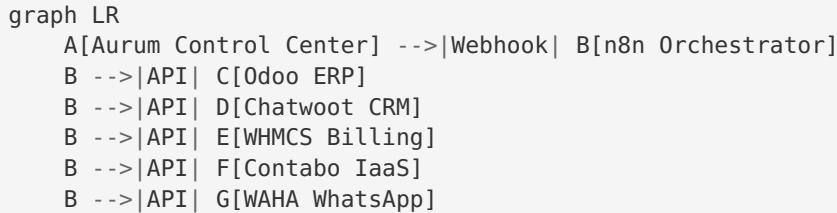
Modelo de Datos Bien Estructurado

```
// Schema optimizado para multi-tenancy por satélites
Model Hierarchy:
User (Multi-role support)
  ├── Satellite (Isolation boundary)
  |   ├── Task (Project management)
  |   ├── InternalTicket (Helpdesk)
  |   └── FinanceLog (Financial tracking)
  ├── SystemConfig (Global settings)
  └── PrivacyList (CEO-only restricted data)
```

Ventajas:

- Aislamiento natural por satélite (multi-tenancy)
- Extensibilidad para nuevos tipos de entidades
- Auditoría temporal con `createdAt/updatedAt`
- Soporte para soft-deletes si se implementa

Arquitectura de Integraciones Desacoplada



Beneficios:

- Desacoplamiento temporal (async processing)
- n8n actúa como ESB (Enterprise Service Bus)
- Fácil adición de nuevos sistemas sin modificar core
- Rate limiting y retry logic centralizado en n8n

1.2 Debilidades Críticas

✗ Ausencia de Capa de Servicio

```
// PROBLEMA: Lógica de negocio mezclada en API routes
// app/api/tasks/route.ts
export async function POST(req: Request) {
  const data = await req.json();
  // ✗ Validación, lógica de negocio y persistencia mezcladas
  const task = await prisma.task.create({ data });
  return Response.json(task);
}

// SOLUCIÓN: Separar en capas
// services/TaskService.ts
export class TaskService {
  constructor(
    private prisma: PrismaClient,
    private notificationService: NotificationService,
    private webhookService: WebhookService
  ) {}

  async createTask(input: CreateTaskInput): Promise<Task> {
    // 1. Validación de negocio
    await this.validateTaskRules(input);

    // 2. Transacción atómica
    const task = await this.prisma.$transaction(async (tx) => {
      const newTask = await tx.task.create({ data: input });
      await tx.financeLog.create({ /* tracking */ });
      return newTask;
    });

    // 3. Side effects async
    await this.notificationService.notifyTaskAssigned(task);
    await this.webhookService.triggerTaskCreated(task);

    return task;
  }
}
```

Falta de Gestión de Errores Centralizada

```
// PROBLEMA: Errores inconsistentes
throw new Error("User not found"); //  Genérico

// SOLUCIÓN: Error Hierarchy
class AppError extends Error {
  constructor(
    public code: string,
    public statusCode: number,
    message: string,
    public isOperational = true
  ) {
    super(message);
  }
}

class NotFoundError extends AppError {
  constructor(resource: string, id: string) {
    super('NOT_FOUND', 404, `${resource} ${id} not found`);
  }
}

class UnauthorizedError extends AppError {
  constructor(action: string) {
    super('UNAUTHORIZED', 403, `Not authorized to ${action}`);
  }
}

// Middleware global de errores
export function errorHandler(error: Error, req: Request) {
  if (error instanceof AppError && error.isOperational) {
    logger.warn('Operational error', { error });
    return Response.json({ error: error.message }, { status: error.statusCode });
  }

  // Error no esperado - alertar equipo
  logger.error('System error', { error, req });
  alertOpsTeam(error);
  return Response.json({ error: 'Internal server error' }, { status: 500 });
}
```

✗ Sin Circuit Breaker para Integraciones

```
// PROBLEMA: Un webhook caído tumba todo el sistema
await fetch('https://n8n.aurum.com/webhook/odoo', {
  method: 'POST',
  body: JSON.stringify(data)
}); // ✗ Si n8n está caído, el request cuelga

// SOLUCIÓN: Circuit Breaker Pattern
import CircuitBreaker from 'opossum';

const webhookBreaker = new CircuitBreaker(async (url: string, data: any) => {
  return await fetch(url, {
    method: 'POST',
    body: JSON.stringify(data),
    timeout: 5000 // 5s timeout
  });
}, {
  timeout: 5000,
  errorThresholdPercentage: 50,
  resetTimeout: 30000, // Retry después de 30s
  fallback: async (url, data) => {
    // Guardar en cola para retry posterior
    await redis.lpush('webhook_queue', JSON.stringify({ url, data }));
    logger.warn('Webhook failed, queued for retry', { url });
  }
});

webhookBreaker.on('open', () => {
  alertOpsTeam('Webhook circuit breaker OPEN - service degraded');
});
```

1.3 Riesgos Arquitectónicos

Riesgo	Probabilidad	Impacto	Mitigación
Single Point of Failure en n8n	Alta	Crítico	Implementar cola de mensajes (Redis/RabbitMQ) con retry logic
Query N+1 en dashboards	Alta	Alto	Implementar Prisma include strategies y DataLoader pattern
Saturación de conexiones DB	Media	Crítico	Connection pooling + PgBouncer como proxy
Data breach por IDOR	Media	Crítico	Implementar Row Level Security (RLS) en PostgreSQL
Inconsistencia de datos en transacciones distribuidas	Media	Alto	Implementar Saga pattern o Two-Phase Commit

2. Seguridad: Mejoras Críticas

2.1 Autenticación: De Básica a Enterprise-Grade

Estado Actual (Asumido)

```
// PROBLEMA: Auth básica sin protecciones enterprise
// lib/auth.ts
export async function login(email: string, password: string) {
  const user = await prisma.user.findUnique({ where: { email } });
  const valid = await bcrypt.compare(password, user.passwordHash);

  if (!valid) throw new Error('Invalid credentials');

  const token = jwt.sign({ userId: user.id }, JWT_SECRET);
  return token; // ✗ Sin expiración, sin refresh, sin rate limiting
}
```

 **Solución Recomendada: Multi-Layer Auth**

```

// services/AuthService.ts
import { RateLimiter } from '@/lib/rate-limiter';
import { AuditLogger } from '@/lib/audit-logger';
import speakeasy from 'speakeasy'; // 2FA

export class AuthService {
  private loginRateLimiter = new RateLimiter({
    points: 5, // 5 intentos
    duration: 900, // 15 minutos
    blockDuration: 3600 // 1 hora de bloqueo
  });

  async login(req: Request, email: string, password: string) {
    const ipAddress = req.headers.get('x-forwarded-for') || 'unknown';

    // 1. Rate limiting por IP y email
    await this.loginRateLimiter.consume(`${ipAddress}:${email}`);

    // 2. Buscar usuario con datos de seguridad
    const user = await prisma.user.findUnique({
      where: { email },
      select: {
        id: true,
        email: true,
        passwordHash: true,
        role: true,
        twoFactorSecret: true,
        failedLoginAttempts: true,
        lockedUntil: true,
        lastLoginAt: true,
        lastLoginIp: true
      }
    });

    if (!user) {
      // Log intento de login a usuario inexistente
      await AuditLogger.log('LOGIN_FAILED', { email, reason: 'USER_NOT_FOUND', ipAddress });
      throw new UnauthorizedError('Invalid credentials'); // Mensaje genérico
    }

    // 3. Verificar si cuenta está bloqueada
    if (user.lockedUntil && user.lockedUntil > new Date()) {
      await AuditLogger.log('LOGIN_BLOCKED', { userId: user.id, ipAddress });
      throw new ForbiddenError('Account temporarily locked');
    }

    // 4. Verificar password
    const validPassword = await bcrypt.compare(password, user.passwordHash);

    if (!validPassword) {
      // Incrementar contador de intentos fallidos
      const attempts = user.failedLoginAttempts + 1;
      await prisma.user.update({
        where: { id: user.id },
        data: {
          failedLoginAttempts: attempts,
          lockedUntil: attempts >= 5 ? new Date(Date.now() + 3600000) : null // 1 hora
        }
      });

      await AuditLogger.log('LOGIN_FAILED', { userId: user.id, reason: 'INVAL-

```

```

ID_PASSWORD', ipAddress));
    throw new UnauthorizedError('Invalid credentials');
}

// 5. Verificar 2FA (si está habilitado)
if (user.twoFactorSecret) {
    // Enviar challenge, usuario debe completar 2FA
    return { requiresTwoFactor: true, challengeToken: this.generate2FAChallenge(user
.id) };
}

// 6. Generar tokens
const tokens = await this.generateTokenPair(user);

// 7. Actualizar metadatos de seguridad
await prisma.user.update({
    where: { id: user.id },
    data: {
        failedLoginAttempts: 0,
        lockedUntil: null,
        lastLoginAt: new Date(),
        lastLoginIp: ipAddress
    }
});

// 8. Detectar login desde nueva ubicación
if (user.lastLoginIp && user.lastLoginIp !== ipAddress) {
    await this.notifyNewLocationLogin(user, ipAddress);
}

await AuditLogger.log('LOGIN_SUCCESS', { userId: user.id, ipAddress });

return tokens;
}

private async generateTokenPair(user: User) {
    const accessToken = jwt.sign(
    {
        userId: user.id,
        role: user.role,
        type: 'access'
    },
    process.env.JWT_SECRET!,
    { expiresIn: '15m', issuer: 'aurum-control', audience: 'aurum-api' }
);

const refreshToken = jwt.sign(
{
    userId: user.id,
    type: 'refresh',
    tokenVersion: user.tokenVersion // Para invalidación masiva
},
process.env.JWT_REFRESH_SECRET!,
{ expiresIn: '7d' }
);

// Guardar refresh token en DB (para revocación)
await prisma.refreshToken.create({
    data: {
        userId: user.id,
        token: await hashToken(refreshToken),
        expiresAt: new Date(Date.now() + 7 * 24 * 3600000)
    }
}

```

```
});

    return { accessToken, refreshToken };

}

// 2FA verification
async verify2FA(challengeToken: string, totpCode: string) {
    const { userId } = jwt.verify(challengeToken, process.env.JWT_2FA_SECRET!);
    const user = await prisma.user.findUnique({
        where: { id: userId },
        select: { id: true, twoFactorSecret: true, role: true }
    });

    const valid = speakeasy.totp.verify({
        secret: user.twoFactorSecret,
        encoding: 'base32',
        token: totpCode,
        window: 1 // Permite 30s de desfase
    });

    if (!valid) {
        await AuditLogger.log('2FA_FAILED', { userId });
        throw new UnauthorizedError('Invalid 2FA code');
    }

    await AuditLogger.log('2FA_SUCCESS', { userId });
    return this.generateTokenPair(user);
}
}
```

2.2 Autorización: Row-Level Security (RLS)

Implementar Políticas de Acceso por Satélite

```
// middleware/rbac.ts
export class RBACMiddleware {
  async checkSatelliteAccess(userId: string, satelliteId: string): Promise<boolean> {
    const user = await prisma.user.findUnique({
      where: { id: userId },
      include: { satellites: true }
    });

    if (user.role === 'CEO') return true; // CEO accede a todo

    if (user.role === 'MANAGER') {
      // Manager solo accede a sus satélites asignados
      return user.satellites.some(sat => sat.id === satelliteId);
    }

    if (user.role === 'EMPLOYEE') {
      // Employee solo ve tasks/tickets asignados a él
      return user.satellites.some(sat => sat.id === satelliteId);
    }

    return false;
  }

  // Filtro automático de queries por permisos
  getSatelliteFilter(user: User) {
    if (user.role === 'CEO') return {}; // Sin filtro

    return {
      satelliteId: {
        in: user.satellites.map(s => s.id)
      }
    };
  }
}

// Uso en API routes
export async function GET(req: Request) {
  const user = await getCurrentUser(req);
  const rbac = new RBACMiddleware();

  const tasks = await prisma.task.findMany({
    where: {
      ...rbac.getSatelliteFilter(user), // Auto-filtrado
      status: 'PENDING'
    }
  });

  return Response.json(tasks);
}
```

PostgreSQL RLS (Defensa en Profundidad)

```
-- Habilitar RLS en tablas críticas
ALTER TABLE "Task" ENABLE ROW LEVEL SECURITY;
ALTER TABLE "InternalTicket" ENABLE ROW LEVEL SECURITY;
ALTER TABLE "FinanceLog" ENABLE ROW LEVEL SECURITY;

-- Política: CEO accede a todo
CREATE POLICY ceo_all_access ON "Task"
FOR ALL
TO aurum_app
USING (
EXISTS (
    SELECT 1 FROM "User"
    WHERE id = current_setting('app.user_id')::uuid
    AND role = 'CEO'
)
);

-- Política: Managers solo a sus satélites
CREATE POLICY manager_satellite_access ON "Task"
FOR ALL
TO aurum_app
USING (
"satelliteId" IN (
    SELECT "satelliteId" FROM "_SatelliteToUser"
    WHERE "userId" = current_setting('app.user_id')::uuid
)
);

-- Política: Employees solo a sus tareas asignadas
CREATE POLICY employee_assigned_tasks ON "Task"
FOR SELECT
TO aurum_app
USING (
"assignedToId" = current_setting('app.user_id')::uuid
);

-- Set user context en cada conexión
-- lib/prisma.ts
export async function getPrismaForUser(userId: string) {
    return prisma.$executeRaw`SET app.user_id = ${userId}`;
}
```

2.3 Protección de Datos Sensibles

Encriptación de Campos Críticos

```

// lib/encryption.ts
import crypto from 'crypto';

export class FieldEncryption {
  private algorithm = 'aes-256-gcm';
  private key = Buffer.from(process.env.ENCRYPTION_KEY!, 'hex'); // 32 bytes

  encrypt(text: string): string {
    const iv = crypto.randomBytes(16);
    const cipher = crypto.createCipheriv(this.algorithm, this.key, iv);

    let encrypted = cipher.update(text, 'utf8', 'hex');
    encrypted += cipher.final('hex');

    const authTag = cipher.getAuthTag();

    // Formato: iv:authTag:encrypted
    return `${iv.toString('hex')}:${authTag.toString('hex')}:${encrypted}`;
  }

  decrypt(encrypted: string): string {
    const [ivHex, authTagHex, encryptedData] = encrypted.split(':');

    const iv = Buffer.from(ivHex, 'hex');
    const authTag = Buffer.from(authTagHex, 'hex');
    const decipher = crypto.createDecipheriv(this.algorithm, this.key, iv);

    decipher.setAuthTag(authTag);

    let decrypted = decipher.update(encryptedData, 'hex', 'utf8');
    decrypted += decipher.final('utf8');

    return decrypted;
  }
}

// Uso en modelos Prisma
// prisma/schema.prisma
model PrivacyList {
  id      String  @id @default(uuid())
  type    String  // "contact", "financial", "strategic"
  dataEncrypted String @db.Text // Campo encriptado
  createdBy String
  createdBy User    @relation(fields: [createdById], references: [id])
}

// services/PrivacyService.ts
export class PrivacyService {
  private encryption = new FieldEncryption();

  async createPrivacyEntry(userId: string, type: string, data: any) {
    // Solo CEO puede crear entradas privadas
    const user = await prisma.user.findUnique({ where: { id: userId } });
    if (user.role !== 'CEO') {
      throw new UnauthorizedError('Only CEO can create privacy entries');
    }

    const encrypted = this.encryption.encrypt(JSON.stringify(data));

    return prisma.privacyList.create({
      data: {
        type,

```

```
        dataEncrypted: encrypted,
        createdById: userId
    }
});

}

async getPrivacyEntry(userId: string, entryId: string) {
    const user = await prisma.user.findUnique({ where: { id: userId } });
    if (user.role !== 'CEO') {
        await AuditLogger.log('PRIVACY_ACCESS_DENIED', { userId, entryId });
        throw new UnauthorizedError('Access denied');
    }

    const entry = await prisma.privacyList.findUnique({ where: { id: entryId } });

    await AuditLogger.log('PRIVACY_ACCESS', { userId, entryId });

    return {
        ...entry,
        data: JSON.parse(this.encryption.decrypt(entry.dataEncrypted))
    };
}
}
```

2.4 Gestión de Secretos y Tokens

```
// ❌ NO HACER ESTO
const JWT_SECRET = "my-super-secret-key"; // Hardcoded
process.env.DATABASE_URL = "postgresql://user:pass@localhost:5432/db"; // Expuesto

// ✅ SOLUCIÓN: Gestión de secretos enterprise
// .env.vault (encrypted) - usar doppler.com o infisical.com
DATABASE_URL="encrypted:v1:abc123..."
JWT_SECRET="encrypted:v1:def456..."
JWT_REFRESH_SECRET="encrypted:v1:ghi789..."
ENCRYPTION_KEY="encrypted:v1:jk1012..."

// lib/secrets.ts
import { config } from 'dotenv-vault-core';

// Los secretos se desencriptan en runtime con DOTENV_KEY
config();

export const secrets = {
  database: {
    url: process.env.DATABASE_URL!,
    ssl: process.env.DATABASE_SSL === 'true'
  },
  jwt: {
    secret: process.env.JWT_SECRET!,
    refreshSecret: process.env.JWT_REFRESH_SECRET!,
    expiresIn: '15m',
    refreshExpiresIn: '7d'
  },
  encryption: {
    key: Buffer.from(process.env.ENCRYPTION_KEY!, 'hex')
  },
  n8n: {
    webhookUrl: process.env.N8N_WEBHOOK_URL!,
    apiKey: process.env.N8N_API_KEY!
  }
} as const;

// Rotación automática de secrets cada 90 días
// scripts/rotate-secrets.ts
export async function rotateJWTSecrets() {
  const newSecret = crypto.randomBytes(64).toString('hex');

  // 1. Agregar nuevo secret a secrets manager
  await secretsManager.addVersion('JWT_SECRET', newSecret);

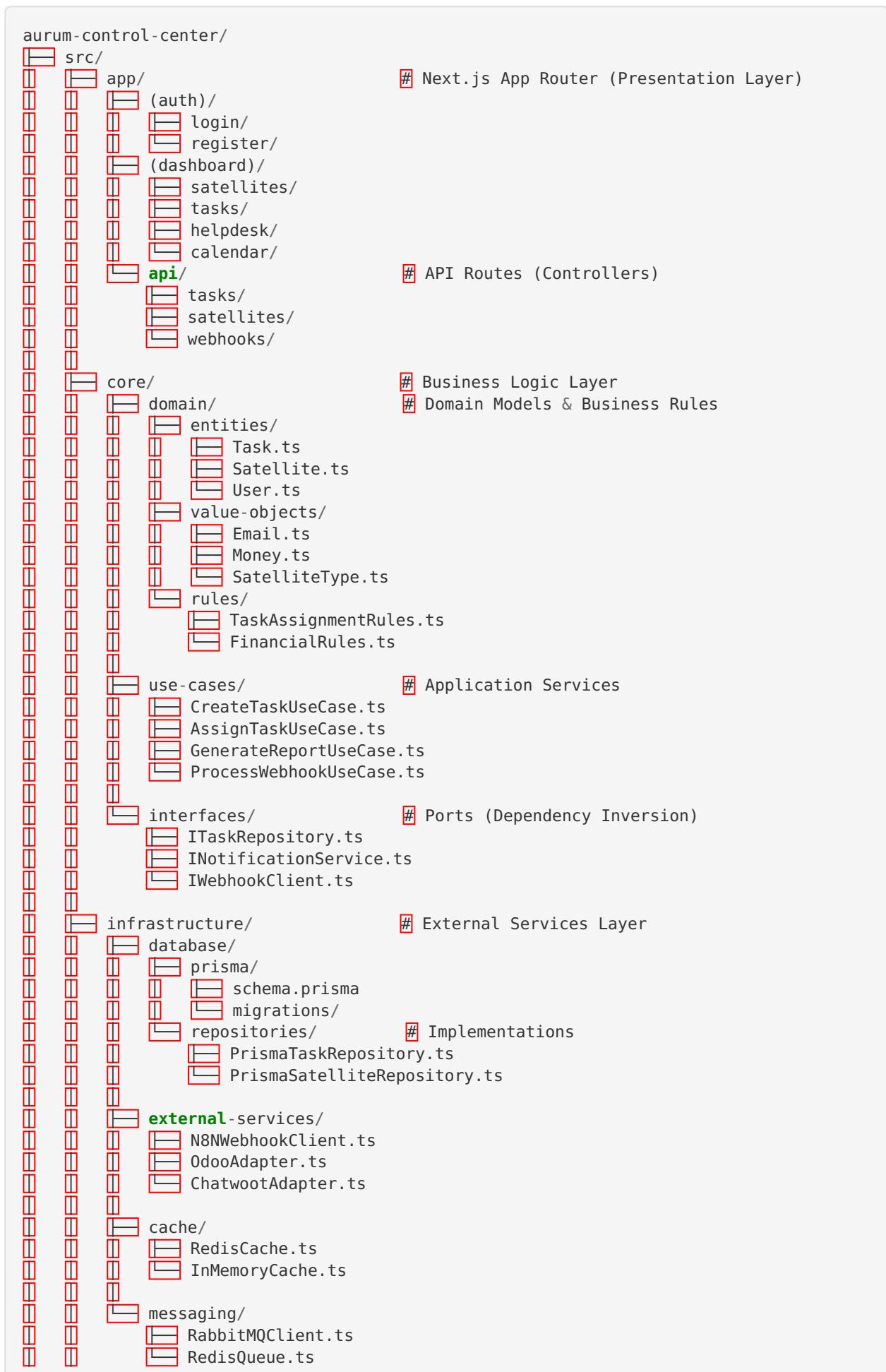
  // 2. Mantener secret anterior por 24h (grace period)
  // Durante este periodo, ambos secrets son válidos

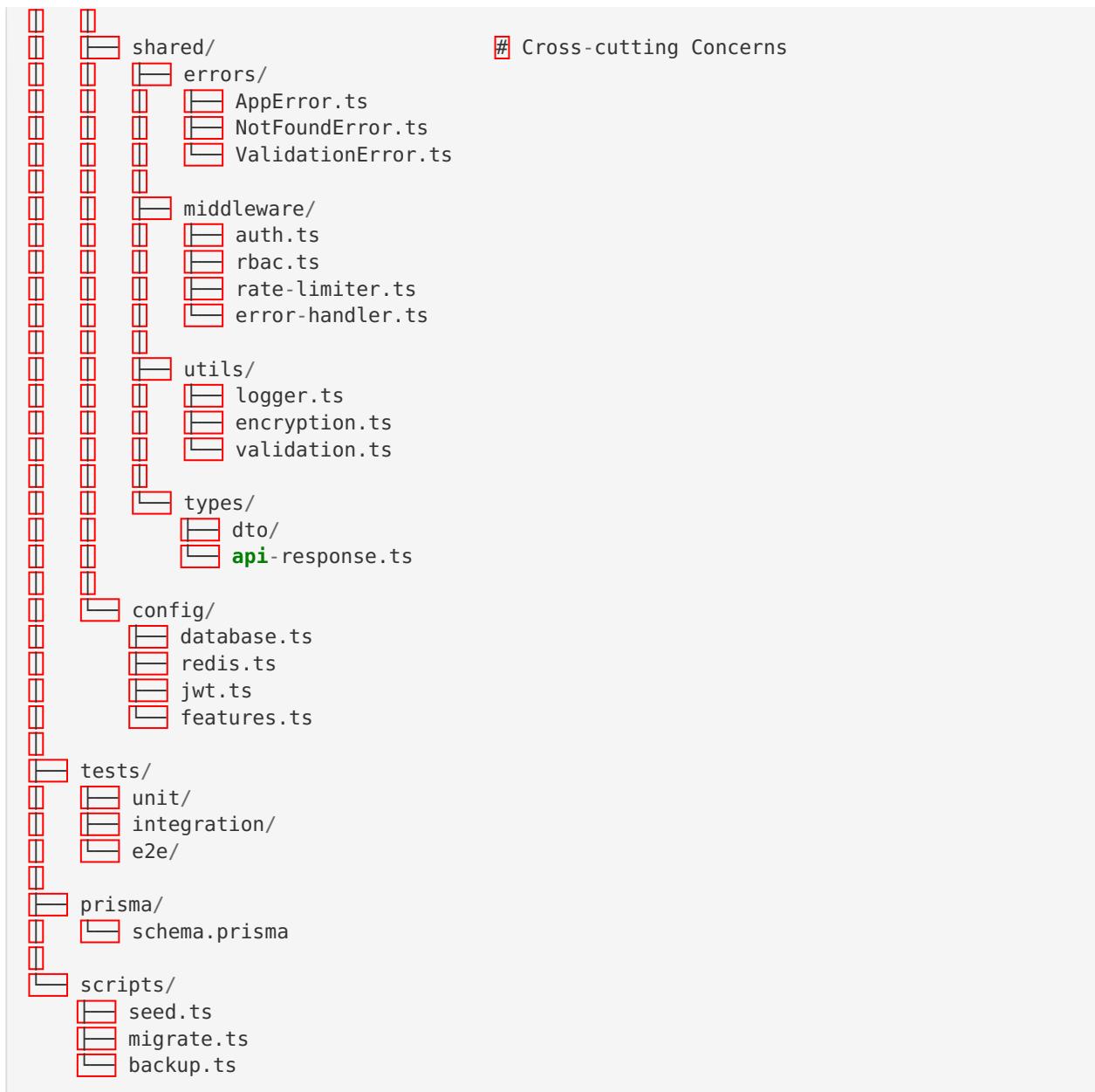
  // 3. Después de 24h, deprecar secret anterior
  setTimeout(() => {
    secretsManager.deprecateVersion('JWT_SECRET', 'old-version');
  }, 24 * 3600000);

  // 4. Notificar equipo ops
  await notifyOpsTeam('JWT secret rotated successfully');
}
```

3. Arquitectura: Escalabilidad y Mantenibilidad

3.1 Separación de Capas (Clean Architecture)





3.2 Patrón Repository

```

// core/interfaces/ITaskRepository.ts
export interface ITaskRepository {
    findById(id: string): Promise<Task | null>;
    findBySatellite(satelliteId: string, filters: TaskFilters): Promise<Task[]>;
    create(task: CreateTaskDTO): Promise<Task>;
    update(id: string, data: UpdateTaskDTO): Promise<Task>;
    delete(id: string): Promise<void>;
}

// infrastructure/database/repositories/PrismaTaskRepository.ts
export class PrismaTaskRepository implements ITaskRepository {
    constructor(private prisma: PrismaClient) {}

    async findById(id: string): Promise<Task | null> {
        const taskData = await this.prisma.task.findUnique({
            where: { id },
            include: {
                satellite: true,
                assignedTo: true,
                createdBy: true
            }
        });

        if (!taskData) return null;

        // Convertir de Prisma model a Domain Entity
        return Task.fromPrisma(taskData);
    }

    async findBySatellite(satelliteId: string, filters: TaskFilters): Promise<Task[]> {
        const tasks = await this.prisma.task.findMany({
            where: {
                satelliteId,
                status: filters.status,
                priority: filters.priority,
                dueDate: filters.dueDateRange ? {
                    gte: filters.dueDateRange.start,
                    lte: filters.dueDateRange.end
                } : undefined
            },
            orderBy: {
                priority: 'desc',
                dueDate: 'asc'
            },
            take: filters.limit || 50
        });

        return tasks.map(Task.fromPrisma);
    }

    async create(dto: CreateTaskDTO): Promise<Task> {
        const taskData = await this.prisma.task.create({
            data: {
                title: dto.title,
                description: dto.description,
                status: dto.status || 'PENDING',
                priority: dto.priority || 'MEDIUM',
                dueDate: dto.dueDate,
                satelliteId: dto.satelliteId,
                assignedToId: dto.assignedToId,
                createdBy: dto.createdBy
            },
        });
    }
}

```

```

        include: {
            satellite: true,
            assignedTo: true
        }
    });

    return Task.fromPrisma(taskData);
}
}

// core/use-cases/CreateTaskUseCase.ts
export class CreateTaskUseCase {
    constructor(
        private taskRepository: ITaskRepository,
        private satelliteRepository: ISatelliteRepository,
        private notificationService: INotificationService,
        private webhookService: IWebhookService
    ) {}

    async execute(input: CreateTaskInput, userId: string): Promise<Task> {
        // 1. Validar que el satélite existe
        const satellite = await this.satelliteRepository.findById(input.satelliteId);
        if (!satellite) {
            throw new NotFoundError('Satellite', input.satelliteId);
        }

        // 2. Validar permisos del usuario
        if (!satellite.canUserCreateTasks(userId)) {
            throw new UnauthorizedError('create tasks in this satellite');
        }

        // 3. Aplicar reglas de negocio
        if (input.priority === 'CRITICAL' && !input.assignedToId) {
            throw new ValidationError('Critical tasks must be assigned immediately');
        }

        // 4. Crear tarea
        const task = await this.taskRepository.create({
            ...input,
            createdBy: userId
        });

        // 5. Side effects (async, no bloquean respuesta)
        this.notificationService.notifyTaskAssigned(task).catch(err => {
            logger.error('Failed to send task notification', { taskId: task.id, error: err });
        });
    };

    this.webhookService.triggerTaskCreated(task).catch(err => {
        logger.error('Failed to trigger webhook', { taskId: task.id, error: err });
    });

    return task;
}
}

// app/api/tasks/route.ts (API Controller)
export async function POST(req: Request) {
    try {
        const user = await getCurrentUser(req);
        const input = await req.json();

        // Validar input
    }
}

```

```
const validatedInput = CreateTaskInputSchema.parse(input);

// Ejecutar use case
const createTaskUseCase = new CreateTaskUseCase(
    new PrismaTaskRepository(prisma),
    new PrismaSatelliteRepository(prisma),
    new EmailNotificationService(),
    new N8NWebhookService()
);

const task = await createTaskUseCase.execute(validatedInput, user.id);

return Response.json(task, { status: 201 });
} catch (error) {
    return handleError(error);
}
}
```

3.3 Caching Strategy

```

// infrastructure/cache/CacheService.ts
import Redis from 'ioredis';

export class CacheService {
    private redis: Redis;

    constructor() {
        this.redis = new Redis({
            host: process.env.REDIS_HOST,
            port: parseInt(process.env.REDIS_PORT || '6379'),
            password: process.env.REDIS_PASSWORD,
            retryStrategy: (times) => Math.min(times * 50, 2000),
            maxRetriesPerRequest: 3
        });
    }

    // Cache con TTL
    async get<T>(key: string): Promise<T | null> {
        const data = await this.redis.get(key);
        return data ? JSON.parse(data) : null;
    }

    async set(key: string, value: any, ttlSeconds: number = 300): Promise<void> {
        await this.redis.setex(key, ttlSeconds, JSON.stringify(value));
    }

    // Cache-aside pattern
    async wrap<T>(
        key: string,
        fetcher: () => Promise<T>,
        ttl: number = 300
    ): Promise<T> {
        // Intentar obtener del cache
        const cached = await this.get<T>(key);
        if (cached) return cached;

        // Si no existe, ejecutar fetcher
        const data = await fetcher();

        // Guardar en cache
        await this.set(key, data, ttl);

        return data;
    }

    // Invalidation por tags
    async invalidateByPattern(pattern: string): Promise<void> {
        const keys = await this.redis.keys(pattern);
        if (keys.length > 0) {
            await this.redis.del(...keys);
        }
    }
}

// Uso en repositories
export class PrismaTaskRepository implements ITaskRepository {
    constructor(
        private prisma: PrismaClient,
        private cache: CacheService
    ) {}

    async findBySatellite(satelliteId: string, filters: TaskFilters): Promise<Task[]> {

```

```
const cacheKey = `tasks:satellite:${satelliteId}: ${JSON.stringify(filters)}`;

return this.cache.wrap(
  cacheKey,
  async () => {
    return this.prisma.task.findMany({
      where: { satelliteId, ...filters }
    });
  },
  300 // 5 minutos
);
}

async create(dto: CreateTaskDTO): Promise<Task> {
  const task = await this.prisma.task.create({ data: dto });

  // Invalidar caches relacionados
  await this.cache.invalidateByPattern(`tasks:satellite:${dto.satelliteId}:*`);
  await this.cache.invalidateByPattern(`dashboard:satellite:${dto.satelliteId}`);

  return task;
}
}

// Estrategia de caching por tipo de dato
const CacheTTL = {
  // Datos estáticos (raramente cambian)
  SATELLITES: 3600,           // 1 hora
  USER_PERMISSIONS: 1800,     // 30 minutos
  SYSTEM_CONFIG: 3600,        // 1 hora

  // Datos dinámicos (cambian frecuentemente)
  TASK_LIST: 300,             // 5 minutos
  DASHBOARD_STATS: 60,        // 1 minuto
  NOTIFICATIONS: 30,          // 30 segundos

  // Datos críticos (no cachear o TTL muy corto)
  FINANCE_LOGS: 0,            // No cachear
  AUDIT_LOGS: 0,               // No cachear
  PRIVACY_DATA: 0,             // No cachear
};
}
```

3.4 Queue System para Resiliencia

```
// infrastructure/messaging/QueueService.ts
import Bull, { Queue, Job } from 'bull';

export class QueueService {
  private queues: Map<string, Queue> = new Map();

  constructor() {
    this.initializeQueues();
  }

  private initializeQueues() {
    // Queue para webhooks con retry logic
    const webhookQueue = new Bull('webhooks', {
      redis: {
        host: process.env.REDIS_HOST,
        port: parseInt(process.env.REDIS_PORT || '6379')
      },
      defaultJobOptions: {
        attempts: 5,
        backoff: {
          type: 'exponential',
          delay: 5000 // 5s, 10s, 20s, 40s, 80s
        },
        removeOnComplete: 100, // Mantener últimos 100 trabajos
        removeOnFail: 500
      }
    });
    webhookQueue.process(async (job: Job) => {
      return this.processWebhook(job.data);
    });

    webhookQueue.on('failed', async (job, err) => {
      logger.error('Webhook job failed', {
        jobId: job.id,
        data: job.data,
        error: err,
        attemptsMade: job.attemptsMade
      });

      // Si agotó todos los reintentos, alertar equipo
      if (job.attemptsMade >= 5) {
        await alertOpsTeam('Webhook job permanently failed', {
          jobId: job.id,
          data: job.data
        });
      }
    });
    this.queues.set('webhooks', webhookQueue);

    // Queue para notificaciones
    const notificationQueue = new Bull('notifications', {
      redis: { host: process.env.REDIS_HOST },
      defaultJobOptions: {
        attempts: 3,
        backoff: { type: 'fixed', delay: 10000 }
      }
    });
    notificationQueue.process(async (job: Job) => {
      return this.processNotification(job.data);
    });
  }
}
```

```

});;

this.queues.set('notifications', notificationQueue);

// Queue para reportes pesados
const reportQueue = new Bull('reports', {
  redis: { host: process.env.REDIS_HOST },
  limiter: {
    max: 5,           // Máximo 5 reportes
    duration: 60000 // Por minuto
  }
});

reportQueue.process(async (job: Job) => {
  return this.generateReport(job.data);
});

this.queues.set('reports', reportQueue);
}

async enqueueWebhook(url: string, data: any, priority: number = 0) {
  const queue = this.queues.get('webhooks')!;
  return queue.add({ url, data }, { priority });
}

async enqueueNotification(type: string, recipient: string, payload: any) {
  const queue = this.queues.get('notifications')!;
  return queue.add({ type, recipient, payload });
}

async enqueueReport(reportType: string, params: any, userId: string) {
  const queue = this.queues.get('reports')!;
  return queue.add({ reportType, params, userId }, {
    priority: 1,
    timeout: 300000 // 5 minutos timeout
  });
}

private async processWebhook(data: any) {
  const { url, data: payload } = data;

  const response = await fetch(url, {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      'X-Aurum-Signature': generateWebhookSignature(payload)
    },
    body: JSON.stringify(payload),
    signal: AbortSignal.timeout(10000) // 10s timeout
  });

  if (!response.ok) {
    throw new Error(`Webhook failed: ${response.status} ${response.statusText}`);
  }

  return response.json();
}

// Uso en servicios
export class N8NWebhookService implements IWebhookService {
  constructor(private queue: QueueService) {}
}

```

```
async triggerTaskCreated(task: Task): Promise<void> {
    // En lugar de hacer fetch directo, encolar
    await this.queue.enqueueWebhook(
        `${process.env.N8N_WEBHOOK_URL}/task-created`,
        {
            taskId: task.id,
            satelliteId: task.satelliteId,
            priority: task.priority,
            assignedToId: task.assignedToId
        },
        task.priority === 'CRITICAL' ? 10 : 0 // Prioridad alta para críticos
    );
}
```

4. 🎨 UX/UI: Experiencia y Performance

4.1 Optimización de Renderizado

```
// app/(dashboard)/satellites/page.tsx
import { Suspense } from 'react';
import { SatelliteCard } from '@/components/SatelliteCard';
import { SatelliteCardSkeleton } from '@/components/SatelliteCardSkeleton';

export default async function SatellitesPage() {
    // Renderizar shell inmediatamente
    return (
        <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-6">
            {/* Suspense boundaries granulares */}
            <Suspense fallback={<SatelliteCardSkeleton />}>
                <InfrastructureSatellites />
            </Suspense>

            <Suspense fallback={<SatelliteCardSkeleton />}>
                <SaaSatellites />
            </Suspense>

            <Suspense fallback={<SatelliteCardSkeleton />}>
                <MarketingSatellites />
            </Suspense>
        </div>
    );
}

// Cada grupo carga independientemente
async function InfrastructureSatellites() {
    // Server Component: fetch en servidor, HTML pre-renderizado
    const satellites = await prisma.satellite.findMany({
        where: { category: 'INFRASTRUCTURE' },
        include: {
            _count: {
                select: {
                    tasks: { where: { status: 'PENDING' } },
                    tickets: { where: { status: 'OPEN' } }
                }
            }
        }
    });

    return (
        <>
            {satellites.map(sat => (
                <SatelliteCard key={sat.id} satellite={sat} />
            ))}
        </>
    );
}
```

4.2 Optimistic Updates

```

// hooks/useCreateTask.ts
'use client';

import { useMutation, useQueryClient } from '@tanstack/react-query';

export function useCreateTask(satelliteId: string) {
  const queryClient = useQueryClient();

  return useMutation({
    mutationFn: async (taskData: CreateTaskInput) => {
      const response = await fetch('/api/tasks', {
        method: 'POST',
        body: JSON.stringify(taskData)
      });

      if (!response.ok) throw new Error('Failed to create task');
      return response.json();
    },
    // Optimistic update: actualizar UI inmediatamente
    onMutate: async (newTask) => {
      // Cancelar queries en vuelo para evitar race conditions
      await queryClient.cancelQueries({
        queryKey: ['tasks', satelliteId]
      });

      // Guardar snapshot para rollback
      const previousTasks = queryClient.getQueryData(['tasks', satelliteId]);

      // Actualizar cache optimísticamente
      queryClient.setQueryData(['tasks', satelliteId], (old: Task[]) => [
        ...old,
        {
          ...newTask,
          id: `temp-${Date.now()}`, // ID temporal
          status: 'PENDING',
          createdAt: new Date().toISOString()
        }
      ]);
    },
    // Revertir en caso de error
    onError: (err, newTask, context) => {
      queryClient.setQueryData(['tasks', satelliteId], context?.previousTasks);

      toast.error('Failed to create task', {
        description: err.message
      });
    },
    // Sincronizar con servidor después de éxito
    onSuccess: () => {
      queryClient.invalidateQueries({ queryKey: ['tasks', satelliteId] });
      queryClient.invalidateQueries({ queryKey: ['dashboard', satelliteId] });

      toast.success('Task created successfully');
    }
  });
}

```

```
// Uso en componente
function CreateTaskButton({ satelliteId }: { satelliteId: string }) {
  const createTask = useCreateTask(satelliteId);

  const handleSubmit = (data: CreateTaskInput) => {
    // UI se actualiza instantáneamente
    createTask.mutate(data);
  };

  return <TaskForm onSubmit={handleSubmit} isLoading={createTask.isPending} />;
}
```

4.3 Virtual Scrolling para Listas Grandes

```
// components/TaskList.tsx
'use client';

import { useVirtualizer } from '@tanstack/react-virtual';
import { useRef } from 'react';

export function TaskList({ tasks }: { tasks: Task[] }) {
  const parentRef = useRef<HTMLDivElement>(null);

  // Virtualizar lista: solo renderizar items visibles
  const virtualizer = useVirtualizer({
    count: tasks.length,
    getScrollElement: () => parentRef.current,
    estimateSize: () => 80, // Altura estimada por item
    overscan: 5 // Renderizar 5 items extra fuera de viewport
  });

  return (
    <div
      ref={parentRef}
      className="h-[600px] overflow-auto"
    >
      <div
        style={{
          height: `${virtualizer.getTotalSize()}px`,
          width: '100%',
          position: 'relative'
        }}
      >
        {virtualizer.getVirtualItems().map((virtualItem) => (
          <div
            key={virtualItem.key}
            style={{
              position: 'absolute',
              top: 0,
              left: 0,
              width: '100%',
              height: `${virtualItem.size}px`,
              transform: `translateY(${virtualItem.start}px)`
            }}
          >
            <TaskCard task={tasks[virtualItem.index]} />
          </div>
        )));
      </div>
    </div>
  );
}

// Beneficio: 1000 tareas renderiza solo ~20 items visibles
// Memoria: 95% reducción, FPS: 60fps constante
```

4.4 Progressive Web App (PWA)

```
// next.config.js
const withPWA = require('next-pwa')({
  dest: 'public',
  register: true,
  skipWaiting: true,
  disable: process.env.NODE_ENV === 'development',
  runtimeCaching: [
    {
      urlPattern: /^https:\/\/fonts\.(?:gstatic|googleapis)\.com\/.*\/i,
      handler: 'CacheFirst',
      options: {
        cacheName: 'google-fonts',
        expiration: {
          maxEntries: 4,
          maxAgeSeconds: 365 * 24 * 60 * 60 // 1 año
        }
      }
    },
    {
      urlPattern: /^https:\/\/aurum\.com\/api\/.*\/i,
      handler: 'NetworkFirst',
      options: {
        cacheName: 'api-cache',
        networkTimeoutSeconds: 10,
        expiration: {
          maxEntries: 50,
          maxAgeSeconds: 5 * 60 // 5 minutos
        }
      }
    }
  ]
});

module.exports = withPWA({
  // ... resto de config
});

// public/manifest.json
{
  "name": "Aurum Control Center",
  "short_name": "Aurum",
  "description": "ERP Centralizado para Holding",
  "theme_color": "#FFD700",
  "background_color": "#0a0a0f",
  "display": "standalone",
  "start_url": "/dashboard",
  "icons": [
    {
      "src": "/icons/icon-192x192.png",
      "sizes": "192x192",
      "type": "image/png"
    },
    {
      "src": "/icons/icon-512x512.png",
      "sizes": "512x512",
      "type": "image/png"
    }
  ]
}
```

4.5 Accesibilidad (WCAG 2.1 Level AA)

```
// components/ui/Button.tsx
import { forwardRef } from 'react';
import { cva, type VariantProps } from 'class-variance-authority';

const buttonVariants = cva(
  "inline-flex items-center justify-center rounded-md font-medium transition-colors
focus-visible:outline-none focus-visible:ring-2 focus-visible:ring-gold-500 focus-
visible:ring-offset-2 disabled:opacity-50 disabled:pointer-events-none",
  {
    variants: {
      variant: {
        default: "bg-gold-500 text-slate-950 hover:bg-gold-600",
        outline: "border-2 border-gold-500 text-gold-500 hover:bg-gold-500/10",
        ghost: "hover:bg-slate-800 text-slate-100"
      },
      size: {
        default: "h-10 py-2 px-4",
        sm: "h-8 px-3 text-sm",
        lg: "h-12 px-8 text-lg"
      }
    }
  }
);

export const Button = forwardRef<HTMLButtonElement, ButtonProps>(
  ({ variant, size, className, children, ...props }, ref) => {
    return (
      <button
        ref={ref}
        className={cn(buttonVariants({ variant, size }), className)}
        // ✓ Accesibilidad: estados ARIA
        aria-disabled={props.disabled}
        role="button"
        tabIndex={props.disabled ? -1 : 0}
        {...props}
      >
        {children}
      </button>
    );
  }
);

// Uso con indicadores de carga accesibles
<Button disabled={isLoading}>
  {isLoading && (
    <>
      <Loader className="mr-2 h-4 w-4 animate-spin" />
      <span className="sr-only">Loading...</span> /* ✓ Screen reader */
    </>
  )}
  Create Task
</Button>

// ✓ Contraste de colores (Cyberpunk Gold theme)
// Fondo: #0a0a0f (Slate-950)
// Texto: #FFD700 (Gold) → Ratio 12.5:1 (AAA)
// Texto secundario: #94a3b8 (Slate-400) → Ratio 8.2:1 (AAA)
```

5. Funciones Nuevas de Alto Valor

5.1 Universal Calendar con IA

```

// features/calendar/SmartScheduler.ts
export class SmartScheduler {
  constructor(
    private openai: OpenAI,
    private calendarService: ICalendarService
  ) {}

  /**
   * Analiza el contexto del usuario y sugiere slots óptimos
   * Considera: carga de trabajo, preferencias, urgencia, disponibilidad
   */
  async suggestOptimalSlots(
    userId: string,
    taskTitle: string,
    estimatedDuration: number,
    urgency: 'LOW' | 'MEDIUM' | 'HIGH' | 'CRITICAL'
  ): Promise<TimeSlot[]> {
    // 1. Obtener contexto del usuario
    const userContext = await this.getUserContext(userId);

    // 2. Obtener slots disponibles
    const availableSlots = await this.calendarService.getAvailableSlots(
      userId,
      new Date(),
      new Date(Date.now() + 7 * 24 * 3600000) // Próximos 7 días
    );

    // 3. Usar GPT-4 para scoring de slots
    const prompt = `

Eres un asistente de productividad. Analiza los siguientes slots disponibles y sugiere los 3 mejores momentos para programar una tarea.

```

Contexto del usuario:

- Rol: \${userContext.role}
- Carga de trabajo actual: \${userContext.workload} tareas pendientes
- Preferencias: \${userContext.preferences.preferredWorkHours}
- Patrones históricos: \${userContext.productivityPatterns}

Tarea a programar:

- Título: \${taskTitle}
- Duración estimada: \${estimatedDuration} minutos
- Urgencia: \${urgency}

Slots disponibles:

```
 ${JSON.stringify(availableSlots, null, 2)}
```

Devuelve un JSON con los 3 mejores slots, incluyendo score (0-100) y justificación.

```
`;
```

```
const response = await this.openai.chat.completions.create({
  model: 'gpt-4-turbo',
  messages: [{ role: 'user', content: prompt }],
  response_format: { type: 'json_object' }
});
```

```
const suggestions = JSON.parse(response.choices[0].message.content);
```

```
return suggestions.slots;
```

```
}
```

```
/**
```

```
 * Auto-reagendar tareas cuando hay cambios en prioridades
```

```
/*
async autoReschedule(satelliteId: string, criticalTaskId: string): Promise<void> {
  const allTasks = await prisma.task.findMany({
    where: {
      satelliteId,
      status: { in: ['PENDING', 'IN_PROGRESS'] },
      scheduledAt: { gte: new Date() }
    },
    orderBy: { priority: 'desc' }
  });

  // Algoritmo de reprogramación:
  // 1. Tareas CRITICAL no se mueven
  // 2. Tareas HIGH pueden moverse ±2 horas
  // 3. Tareas MEDIUM/LOW pueden moverse ±1 día

  const rescheduled = await this.optimizeSchedule(allTasks, criticalTaskId);

  // Notificar usuarios afectados
  for (const task of rescheduled) {
    await notificationService.notifyTaskRescheduled(task);
  }
}
```

5.2 Helpdesk Unificado con Routing Inteligente

```

// features/helpdesk/IntelligentRouter.ts
import { OpenAI } from 'openai';

export class IntelligentTicketRouter {
  constructor(
    private openai: OpenAI,
    private prisma: PrismaClient
  ) {}

  async routeTicket(ticket: InternalTicket): Promise<RoutingDecision> {
    // 1. Extraer embeddings del contenido del ticket
    const embeddingResponse = await this.openai.embeddings.create({
      model: 'text-embedding-3-small',
      input: `${ticket.subject}\n\n${ticket.description}`
    });

    const ticketEmbedding = embeddingResponse.data[0].embedding;

    // 2. Buscar tickets similares históricos (vector similarity en Postgres)
    const similarTickets = await prisma.$queryRaw` 
      SELECT
        id,
        subject,
        "assignedToId",
        "resolvedById",
        "resolutionTime",
        1 - (embedding <=> ${ticketEmbedding}::vector) as similarity
      FROM "InternalTicket"
      WHERE status = 'RESOLVED'
      ORDER BY embedding <=> ${ticketEmbedding}::vector
      LIMIT 5
    `;

    // 3. Analizar patrones de resolución
    const expertUsers = this.findExpertUsers(similarTickets);

    // 4. Considerar carga de trabajo actual
    const userWorkloads = await this.getUserWorkloads(expertUsers);

    // 5. Calcular score final
    const scores = expertUsers.map(user => ({
      userId: user.id,
      expertiseScore: user.similarityScore,
      availabilityScore: 1 - (userWorkloads[user.id] / 20), // Normalizado
      finalScore: (user.similarityScore * 0.7) + ((1 - userWorkloads[user.id] / 20) * 0.3)
    }));

    scores.sort((a, b) => b.finalScore - a.finalScore);

    // 6. Asignar al mejor candidato
    const assignedUser = scores[0];

    // 7. Generar sugerencia de resolución basada en tickets similares
    const resolutionSuggestion = await this.generateResolutionSuggestion(
      ticket,
      similarTickets
    );

    return {
      assignedToId: assignedUser.userId,
      estimatedResolutionTime: this.calculateEstimatedTime(similarTickets),
    }
  }
}

```

```

        confidence: assignedUser.finalScore,
        resolutionSuggestion,
        similarTickets: similarTickets.slice(0, 3)
    };
}

private async generateResolutionSuggestion(
    ticket: InternalTicket,
    similarTickets: any[]
): Promise<string> {
    const prompt = `Basándote en estos tickets similares resueltos anteriormente, sugiere pasos para resolver el ticket actual.

Ticket actual:
Asunto: ${ticket.subject}
Descripción: ${ticket.description}

Tickets similares resueltos:
${similarTickets.map(t => `
- ${t.subject}
  Resolución: ${t.resolution}
  Tiempo: ${t.resolutionTime} minutos
`).join('\n')}`

Proporciona una guía paso a paso concisa para resolver el ticket actual.
`;

    const response = await this.openai.chat.completions.create({
        model: 'gpt-4-turbo',
        messages: [{ role: 'user', content: prompt }],
        max_tokens: 500
    });

    return response.choices[0].message.content;
}
}

// Agregar columna de embeddings a schema
// prisma/schema.prisma
model InternalTicket {
    id      String @id @default(uuid())
    subject String
    description String @db.Text
    embedding Unsupported("vector(1536)")? // pgvector extension
    status   String
    assignedToId String?
    assignedTo User?   @relation("AssignedTickets", fields: [assignedToId], references: [id])
    resolvedById String?
    resolvedBy User?   @relation("ResolvedTickets", fields: [resolvedById], references: [id])
    resolution String? @db.Text
    resolutionTime Int? // minutos
    createdAt DateTime @default(now())
    resolvedAt DateTime?
}

// Migración para pgvector
-- migrations/add_pgvector.sql
CREATE EXTENSION IF NOT EXISTS vector;
CREATE INDEX ON "InternalTicket" USING ivfflat (embedding vector_cosine_ops);

```

5.3 CEO War Chest: Dashboard Ejecutivo

```

// features/ceo-office/WarChestDashboard.tsx
'use client';

import { Card } from '@/components/ui/Card';
import { FinancialChart } from '@/components/charts/FinancialChart';
import { SatellitePerformance } from '@/components/SatellitePerformance';
import { AlertsPanel } from '@/components/AlertsPanel';

export function WarChestDashboard() {
  const { data: metrics } = useQuery({
    queryKey: ['ceo-metrics'],
    queryFn: async () => {
      const response = await fetch('/api/ceo/war-chest');
      return response.json();
    },
    refetchInterval: 30000 // Actualizar cada 30s
  });

  return (
    <div className="space-y-6">
      {/* KPIs principales */}
      <div className="grid grid-cols-1 md:grid-cols-4 gap-4">
        <KPICard
          title="Revenue Total"
          value={metrics?.totalRevenue}
          change={metrics?.revenueGrowth}
          format="currency"
        />
        <KPICard
          title="Active Satellites"
          value={metrics?.activeSatellites}
          total={11}
          format="fraction"
        />
        <KPICard
          title="Critical Issues"
          value={metrics?.criticalIssues}
          threshold={5}
          format="alert"
        />
        <KPICard
          title="Cash Position"
          value={metrics?.cashPosition}
          change={metrics?.cashFlow}
          format="currency"
        />
      </div>

      {/* Gráfico financiero consolidado */}
      <Card className="p-6">
        <h3 className="text-xl font-bold text-gold mb-4">
          Consolidated Financial Performance
        </h3>
        <FinancialChart
          data={metrics?.financialTimeSeries}
          satellites={metrics?.satellites}
        />
      </Card>

      {/* Performance por satélite */}
      <div className="grid grid-cols-1 lg:grid-cols-2 gap-6">
        <Card className="p-6">

```

```

        <h3 className="text-xl font-bold text-gold-500 mb-4">
          Satellite Performance Matrix
        </h3>
        <SatellitePerformance satellites={metrics?.satelliteMetrics} />
      </Card>

      <Card className="p-6">
        <h3 className="text-xl font-bold text-gold-500 mb-4">
          Active Alerts & Risks
        </h3>
        <AlertsPanel alerts={metrics?.alerts} />
      </Card>
    </div>

    {/* Privacy Gate */}
    <Card className="p-6 border-2 border-gold-500">
      <div className="flex items-center justify-between mb-4">
        <h3 className="text-xl font-bold text-gold-500">
           Privacy Gate
        </h3>
        <Button variant="outline" onClick={() => openPrivacyVault()}>
          Access Restricted Data
        </Button>
      </div>
      <p className="text-sm text-slate-400">
        Strategic contacts, financial forecasts, and confidential deals.
        Encrypted end-to-end. Only accessible by CEO role.
      </p>
    </Card>
  </div>
);

}

// API endpoint consolidado
// app/api/ceo/war-chest/route.ts
export async function GET(req: Request) {
  const user = await getCurrentUser(req);

  if (user.role !== 'CEO') {
    throw new UnauthorizedError('CEO access only');
  }

  // Queries paralelas para performance
  const [
    financeLogs,
    satellites,
    criticalTickets,
    activeTasks,
    alerts
  ] = await Promise.all([
    prisma.financeLog.findMany({
      where: {
        createdAt: { gte: new Date(Date.now() - 90 * 24 * 3600000) } // 90 días
      },
      orderBy: { createdAt: 'asc' }
    }),
    prisma.satellite.findMany({
      include: {
        _count: {
          select: {
            tasks: { where: { status: { in: ['PENDING', 'IN_PROGRESS'] } } },
            tickets: { where: { status: 'OPEN' } }
          }
        }
      }
    })
  ]);
}

```

```
        },
        tasks: {
          where: { priority: 'CRITICAL', status: { not: 'COMPLETED' } }
        }
      }
    )),
    prisma.internalTicket.findMany({
      where: {
        status: { in: ['OPEN', 'IN_PROGRESS'] },
        priority: 'CRITICAL'
      },
      include: { satellite: true, assignedTo: true }
    }),
    prisma.task.count({
      where: { status: { in: ['PENDING', 'IN_PROGRESS'] } }
    }),
    generateAlerts(satellites)
  ]);
}

// Calcular métricas consolidadas
const metrics = {
  totalRevenue: calculateTotalRevenue(financeLogs),
  revenueGrowth: calculateGrowth(financeLogs),
  activeSatellites: satellites.filter(s => s.status === 'ACTIVE').length,
  criticalIssues: criticalTickets.length,
  cashPosition: calculateCashPosition(financeLogs),
  cashFlow: calculateCashFlow(financeLogs),
  financialTimeSeries: aggregateFinancialData(financeLogs, satellites),
  satelliteMetrics: calculateSatelliteMetrics(satellites),
  alerts,
  satellites
};

return Response.json(metrics);
}
```

5.4 Widgets Específicos por Tipo de Satélite

```

// components/satellite-widgets/WidgetFactory.tsx
export function getSatelliteWidget(satellite: Satellite) {
  const widgetMap: Record<string, React.ComponentType<SatelliteWidgetProps>> = {
    // Infraestructura: Uptime, recursos, incidentes
    'INFRASTRUCTURE': InfrastructureWidget,
    // SaaS/Fintech: MRR, churn, usuarios activos
    'SAAS': SaaSWidget,
    // Marketing: Campañas, ROI, conversiones
    'MARKETING': MarketingWidget,
    // Manufactura: Producción, inventario, calidad
    'MANUFACTURING': ManufacturingWidget,
    // Trading: Portfolio, P&L, riesgo
    'TRADING': TradingWidget,
    // Belleza: Citas, clientes, ingresos
    'BEAUTY': BeautyWidget
  };

  const Widget = widgetMap[satellite.type] || DefaultWidget;
  return <Widget satellite={satellite} />;
}

// Ejemplo: Widget de SaaS
// components/satellite-widgets/SaaSWidget.tsx
export function SaaSWidget({ satellite }: { satellite: Satellite }) {
  const { data: metrics } = useQuery({
    queryKey: ['saas-metrics', satellite.id],
    queryFn: () => fetchSaaSMetrics(satellite.id)
  });

  return (
    <Card className="p-6 bg-gradient-to-br from-slate-900 to-slate-950 border-gold-500/20">
      <div className="flex items-center justify-between mb-6">
        <div>
          <h3 className="text-lg font-bold text-gold-500">{satellite.name}</h3>
          <p className="text-sm text-slate-400">{satellite.type}</p>
        </div>
        <Badge variant={metrics?.health === 'HEALTHY' ? 'success' : 'warning'}>
          {metrics?.health}
        </Badge>
      </div>
      <!-- Métricas clave SaaS -->
      <div className="grid grid-cols-2 gap-4 mb-6">
        <MetricCard
          label="MRR"
          value={formatCurrency(metrics?.mrr)}
          change={metrics?.mrrGrowth}
          icon={<TrendingUp className="h-4 w-4" />}
        />
        <MetricCard
          label="Churn Rate"
          value={formatPercentage(metrics?.churnRate)}
          change={-metrics?.churnChange} // Negativo es bueno
          icon={<Users className="h-4 w-4" />}
          invertColors
        />
      </div>
    </Card>
  );
}

```

```

<MetricCard
    label="Active Users"
    value={metrics?.activeUsers}
    change={metrics?.userGrowth}
    icon={<UserCheck className="h-4 w-4" />}
/>
<MetricCard
    label="LTV / CAC"
    value={metrics?.ltvCacRatio.toFixed(2)}
    threshold={3} // Debe ser > 3
    icon={<DollarSign className="h-4 w-4" />}
/>
</div>

/* Mini chart de tendencia */
<div className="h-[100px]">
    <MRRTrendChart data={metrics?.mrrHistory} />
</div>

/* Acciones rápidas */
<div className="flex gap-2 mt-4">
    <Button size="sm" variant="outline" onClick={() => openSatellite(satellite.id)}>
        View Details
    </Button>
    <Button size="sm" variant="ghost" onClick={() => syncWithOdoo(satellite.id)}>
        <RefreshCw className="h-4 w-4 mr-2" />
        Sync Odoo
    </Button>
</div>
</Card>
);
}

// API para métricas SaaS
// app/api/satellites/[id]/saas-metrics/route.ts
export async function GET(
    req: Request,
    { params }: { params: { id: string } }
) {
    const satellite = await prisma.satellite.findUnique({
        where: { id: params.id }
    });

    if (satellite.type !== 'SAAS') {
        throw new ValidationError('Satellite is not SaaS type');
    }

    // Obtener datos de Odoo via webhook n8n
    const odooData = await fetch(`.${process.env.N8N_WEBHOOK_URL}/odoo-metrics`, {
        method: 'POST',
        body: JSON.stringify({ satelliteId: params.id })
    }).then(r => r.json());

    // Calcular métricas SaaS
    const metrics = {
        mrr: calculateMRR(odooData.subscriptions),
        mrrGrowth: calculateMRRGrowth(odooData.subscriptions),
        churnRate: calculateChurnRate(odooData.subscriptions),
        activeUsers: odooData.activeUsers,
        userGrowth: calculateUserGrowth(odooData.users),
        ltvCacRatio: calculateLTVcac(odooData),
        health: determineHealth(odooData),
    }
}

```

```

    mrrHistory: odooData.mrrTimeSeries
};

// Cachear por 5 minutos
await cache.set(`saas-metrics:${params.id}`, metrics, 300);

return Response.json(metrics);
}

```

6. Mejoras Técnicas Críticas

6.1 Optimización de Base de Datos

```

-- Índices críticos para performance
-- prisma/migrations/add_performance_indexes.sql

-- Búsqueda de tareas por satélite y estado (query más frecuente)
CREATE INDEX idx_task_satellite_status
ON "Task" ("satelliteId", "status", "priority" DESC, "dueDate" ASC);

-- Búsqueda de tickets por usuario asignado
CREATE INDEX idx_ticket_assigned_status
ON "InternalTicket" ("assignedToId", "status", "createdAt" DESC);

-- Búsqueda de logs financieros por satélite y rango de fechas
CREATE INDEX idx_finance_satellite_date
ON "FinanceLog" ("satelliteId", "createdAt" DESC);

-- Búsqueda de usuarios por satélite (joins frecuentes)
CREATE INDEX idx_satellite_users
ON "_SatelliteToUser" ("satelliteId", "userId");

-- Full-text search en tareas y tickets
CREATE INDEX idx_task_search
ON "Task" USING gin(to_tsvector('english', title || ' ' || COALESCE(description, '')));

CREATE INDEX idx_ticket_search
ON "InternalTicket" USING gin(to_tsvector('english', subject || ' ' || description));

-- Índice parcial para datos activos (más rápido que índice completo)
CREATE INDEX idx_active_tasks
ON "Task" ("satelliteId", "dueDate")
WHERE status IN ('PENDING', 'IN_PROGRESS');

-- Índice compuesto para dashboard CEO
CREATE INDEX idx_finance_dashboard
ON "FinanceLog" ("createdAt", "amount", "satelliteId")
WHERE "createdAt" >= NOW() - INTERVAL '90 days';

-- Estadísticas para query planner
ANALYZE "Task";
ANALYZE "InternalTicket";
ANALYZE "FinanceLog";
ANALYZE "Satellite";
ANALYZE "User";

```

```

// Optimización de queries Prisma
// repositories/OptimizedTaskRepository.ts

export class OptimizedTaskRepository {
    // ✗ Query N+1 problem
    async getTasksWithUsersBad(satelliteId: string) {
        const tasks = await prisma.task.findMany({
            where: { satelliteId }
        });

        // Esto hace 1 query por cada tarea!
        for (const task of tasks) {
            task.assignedTo = await prisma.user.findUnique({
                where: { id: task.assignedToId }
            });
        }

        return tasks; // N+1 queries = lentísimo
    }

    // ✓ Solución: Include strategy
    async getTasksWithUsersGood(satelliteId: string) {
        return prisma.task.findMany({
            where: { satelliteId },
            include: {
                assignedTo: {
                    select: {
                        id: true,
                        name: true,
                        email: true,
                        avatar: true
                    } // No traer passwordHash ni datos sensibles
                },
                satellite: {
                    select: {
                        id: true,
                        name: true,
                        type: true
                    }
                }
            },
            orderBy: [
                { priority: 'desc' },
                { dueDate: 'asc' }
            ],
            take: 50 // Limitar resultados
        }); // 1 query con JOINs = rápido
    }

    // ✓ Paginación con cursor para grandes datasets
    async getTasksPaginated(
        satelliteId: string,
        cursor?: string,
        limit: number = 50
    ) {
        return prisma.task.findMany({
            where: { satelliteId },
            take: limit + 1, // +1 para saber si hay más
            cursor: cursor ? { id: cursor } : undefined,
            orderBy: { createdAt: 'desc' },
            include: {

```

```

        assignedTo: { select: { id: true, name: true, avatar: true } }
    }
});

}

// ✅ Agregaciones eficientes
async getTaskStatsByPriority(satelliteId: string) {
    // Usar SQL raw para agregaciones complejas
    return prisma.$queryRaw<TaskStats[]>`  

        SELECT
            priority,
            status,
            COUNT(*)::int as count,
            AVG(EXTRACT(EPOCH FROM (COALESCE("completedAt", NOW()) - "createdAt")))::int
        as avg_completion_time
        FROM "Task"
        WHERE "satelliteId" = ${satelliteId}
        GROUP BY priority, status
        ORDER BY priority DESC
    `;
}

// ✅ Batch operations con transacciones
async updateTasksPriority(
    taskIds: string[],
    newPriority: Priority
): Promise<number> {
    return prisma.$transaction(async (tx) => {
        const result = await tx.task.updateMany({
            where: { id: { in: taskIds } },
            data: { priority: newPriority }
        });

        // Log de auditoría
        await tx.auditLog.create({
            data: {
                action: 'BULK_UPDATE',
                entityType: 'TASK',
                entityIds: taskIds,
                changes: { priority: newPriority }
            }
        });
    });

    return result.count;
}
}

```

6.2 Connection Pooling con PgBouncer

```

# docker-compose.yml
services:
  postgres:
    image: postgres:16-alpine
    environment:
      POSTGRES_DB: aurum
      POSTGRES_USER: aurum_user
      POSTGRES_PASSWORD: ${DB_PASSWORD}
    volumes:
      - postgres_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"
    command: >
      postgres
      -c max_connections=200
      -c shared_buffers=256MB
      -c effective_cache_size=1GB
      -c maintenance_work_mem=64MB
      -c checkpoint_completion_target=0.9
      -c wal_buffers=16MB
      -c default_statistics_target=100
      -c random_page_cost=1.1
      -c effective_io_concurrency=200
      -c work_mem=4MB
      -c min_wal_size=1GB
      -c max_wal_size=4GB

  pgbouncer:
    image: edoburu/pgbouncer:latest
    environment:
      DATABASE_URL: "postgres://aurum_user:${DB_PASSWORD}@postgres:5432/aurum"
      POOL_MODE: transaction
      MAX_CLIENT_CONN: 1000
      DEFAULT_POOL_SIZE: 25 # Conexiones reales a Postgres
      MIN_POOL_SIZE: 10
      RESERVE_POOL_SIZE: 5
      SERVER_IDLE_TIMEOUT: 30
    ports:
      - "6432:5432"
    depends_on:
      - postgres

  app:
    build: .
    environment:
      # App se conecta a PgBouncer, no directamente a Postgres
      DATABASE_URL: "postgresql://aurum_user:${DB_PASSWORD}@pgbouncer:5432/aurum?pg-
      bouncer=true"
    depends_on:
      - pgbouncer
      - redis

  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"
    volumes:
      - redis_data:/data
    command: redis-server --appendonly yes --maxmemory 512mb --maxmemory-policy allkey
    s-lru

volumes:

```

```
postgres_data:
redis_data:
```

```
// lib/prisma.ts - Singleton con connection pooling
import { PrismaClient } from '@prisma/client';

const globalForPrisma = global as unknown as { prisma: PrismaClient };

export const prisma =
  globalForPrisma.prisma ||
  new PrismaClient({
    log: process.env.NODE_ENV === 'development'
      ? ['query', 'error', 'warn']
      : ['error'],
    datasources: {
      db: {
        url: process.env.DATABASE_URL
      }
    }
  }).$extends({
    query: {
      // Middleware para logging de queries lentos
      $allOperations: async ({ operation, model, args, query }) => {
        const start = Date.now();
        const result = await query(args);
        const duration = Date.now() - start;

        // Alertar queries > 1s
        if (duration > 1000) {
          logger.warn('Slow query detected', {
            model,
            operation,
            duration,
            args: JSON.stringify(args).slice(0, 200)
          });
        }

        return result;
      }
    }
  });

if (process.env.NODE_ENV !== 'production') {
  globalForPrisma.prisma = prisma;
}

// Graceful shutdown
process.on('SIGINT', async () => {
  await prisma.$disconnect();
  process.exit(0);
});
```

6.3 Monitoreo y Observabilidad

```

// lib/monitoring/DatadogMonitoring.ts
import { datadogLogs } from '@datadog/browser-logs';
import { datadogRum } from '@datadog/browser-rum';

// Inicializar RUM (Real User Monitoring)
datadogRum.init({
  applicationId: process.env.NEXT_PUBLIC_DATADOG_APP_ID!,
  clientToken: process.env.NEXT_PUBLIC_DATADOG_CLIENT_TOKEN!,
  site: 'datadoghq.com',
  service: 'aurum-control-center',
  env: process.env.NODE_ENV,
  version: process.env.NEXT_PUBLIC_APP_VERSION,
  sessionSampleRate: 100,
  sessionReplaySampleRate: 20,
  trackUserInteractions: true,
  trackResources: true,
  trackLongTasks: true,
  defaultPrivacyLevel: 'mask-user-input'
});

// Logs estructurados
datadogLogs.init({
  clientToken: process.env.NEXT_PUBLIC_DATADOG_CLIENT_TOKEN!,
  site: 'datadoghq.com',
  forwardErrorsToLogs: true,
  sessionSampleRate: 100
});

export class MonitoringService {
  // Track business metrics
  trackTaskCreated(task: Task) {
    datadogRum.addAction('task.created', {
      satelliteId: task.satelliteId,
      priority: task.priority,
      assignedToId: task.assignedToId
    });
  }

  trackTicketResolved(ticket: InternalTicket, resolutionTime: number) {
    datadogRum.addAction('ticket.resolved', {
      satelliteId: ticket.satelliteId,
      resolutionTime,
      category: ticket.category
    });
  }

  // Custom metric para SLO tracking
  this.sendMetric('ticket.resolution_time', resolutionTime, {
    satellite: ticket.satellite.name,
    priority: ticket.priority
  });
}

trackAPICall(endpoint: string, method: string, duration: number, status: number) {
  datadogRum.addTiming(`api.${method}.${endpoint}`, duration);

  if (status >= 500) {
    datadogLogs.logger.error('API error', {
      endpoint,
      method,
      status,
      duration
    });
  }
}

```

```

    }
}

private sendMetric(name: string, value: number, tags: Record<string, string>) {
  fetch('https://api.datadoghq.com/api/v1/series', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      'DD-API-KEY': process.env.DATADOG_API_KEY!
    },
    body: JSON.stringify({
      series: [
        {
          metric: `aurum.${name}`,
          points: [[Date.now() / 1000, value]],
          type: 'gauge',
          tags: Object.entries(tags).map(([k, v]) => `${k}:${v}`)
        }
      ]
    })
  });
}

// Middleware para API routes
// middleware/monitoring.ts
export function withMonitoring(handler: NextApiHandler): NextApiHandler {
  return async (req, res) => {
    const start = Date.now();
    const endpoint = req.url || 'unknown';

    try {
      await handler(req, res);

      const duration = Date.now() - start;
      monitoring.trackAPICall(endpoint, req.method!, duration, res.statusCode);
    } catch (error) {
      const duration = Date.now() - start;

      datadogLogs.logger.error('API handler error', {
        endpoint,
        method: req.method,
        error: error.message,
        stack: error.stack,
        duration
      });

      monitoring.trackAPICall(endpoint, req.method!, duration, 500);

      throw error;
    }
  };
}

```

6.4 Testing Strategy

```

// tests/unit/services/TaskService.test.ts
import { describe, it, expect, beforeEach, vi } from 'vitest';
import { CreateTaskUseCase } from '@/core/use-cases/CreateTaskUseCase';

describe('CreateTaskUseCase', () => {
  let useCase: CreateTaskUseCase;
  let mockTaskRepo: any;
  let mockSatelliteRepo: any;
  let mockNotificationService: any;

  beforeEach(() => {
    // Mocks
    mockTaskRepo = {
      create: vi.fn(),
      findById: vi.fn()
    };

    mockSatelliteRepo = {
      findById: vi.fn()
    };

    mockNotificationService = {
      notifyTaskAssigned: vi.fn()
    };

    useCase = new CreateTaskUseCase(
      mockTaskRepo,
      mockSatelliteRepo,
      mockNotificationService,
      mockWebhookService
    );
  });
});

it('should create task successfully', async () => {
  // Arrange
  const input = {
    title: 'Test Task',
    satelliteId: 'sat-1',
    priority: 'HIGH',
    assignedToId: 'user-1'
  };

  mockSatelliteRepo.findById.mockResolvedValue({
    id: 'sat-1',
    canUserCreateTasks: () => true
  });

  mockTaskRepo.create.mockResolvedValue({
    id: 'task-1',
    ...input
  });

  // Act
  const result = await useCase.execute(input, 'user-admin');

  // Assert
  expect(result.id).toBe('task-1');
  expect(mockTaskRepo.create).toHaveBeenCalledWith({
    ...input,
    createdBy: 'user-admin'
  });
  expect(mockNotificationService.notifyTaskAssigned).toHaveBeenCalled();
});

```

```

});

it('should throw error if satellite not found', async () => {
  mockSatelliteRepo.findById.mockResolvedValue(null);

  await expect(
    useCase.execute({ satelliteId: 'invalid' }, 'user-1')
  ).rejects.toThrow('Satellite invalid not found');
});

it('should throw error if critical task not assigned', async () => {
  mockSatelliteRepo.findById.mockResolvedValue({ id: 'sat-1' });

  await expect(
    useCase.execute({
      title: 'Critical',
      satelliteId: 'sat-1',
      priority: 'CRITICAL'
      // assignedToId missing
    }, 'user-1')
  ).rejects.toThrow('Critical tasks must be assigned immediately');
});
});

// tests/integration/api/tasks.test.ts
import { describe, it, expect, beforeAll, afterAll } from 'vitest';
import { testApiHandler } from 'next-test-api-route-handler';
import * as tasksHandler from '@/app/api/tasks/route';

describe('POST /api/tasks', () => {
  beforeAll(async () => {
    // Setup test database
    await setupTestDatabase();
  });

  afterAll(async () => {
    await cleanupTestDatabase();
  });

  it('should create task with valid data', async () => {
    await testApiHandler({
      handler: tasksHandler,
      test: async ({ fetch }) => {
        const res = await fetch({
          method: 'POST',
          headers: {
            'Content-Type': 'application/json',
            'Authorization': 'Bearer test-token'
          },
          body: JSON.stringify({
            title: 'Integration Test Task',
            satelliteId: 'test-satellite-1',
            priority: 'MEDIUM'
          })
        });

        expect(res.status).toBe(201);
        const data = await res.json();
        expect(data.id).toBeDefined();
        expect(data.title).toBe('Integration Test Task');
      }
    });
  });
});
}
);
});
});
```

```

it('should return 401 without auth', async () => {
  await testApiHandler({
    handler: tasksHandler,
    test: async ({ fetch }) => {
      const res = await fetch({ method: 'POST' });
      expect(res.status).toBe(401);
    }
  });
});

// tests/e2e/task-creation.spec.ts
import { test, expect } from '@playwright/test';

test.describe('Task Creation Flow', () => {
  test.beforeEach(async ({ page }) => {
    // Login
    await page.goto('http://localhost:3000/login');
    await page.fill('input[name="email"]', 'test@aurum.com');
    await page.fill('input[name="password"]', 'test123');
    await page.click('button[type="submit"]');
    await expect(page).toHaveURL('/dashboard');
  });

  test('should create task from satellite page', async ({ page }) => {
    // Navigate to satellite
    await page.click('text=QHosting');
    await expect(page).toHaveURL('/satellites/qhosting');

    // Open create task dialog
    await page.click('button:has-text("New Task")');

    // Fill form
    await page.fill('input[name="title"]', 'E2E Test Task');
    await page.fill('textarea[name="description"]', 'Created via E2E test');
    await page.selectOption('select[name="priority"]', 'HIGH');

    // Submit
    await page.click('button:has-text("Create Task")');

    // Verify toast notification
    await expect(page.locator('text=Task created successfully')).toBeVisible();

    // Verify task appears in list
    await expect(page.locator('text=E2E Test Task')).toBeVisible();
  });

  test('should validate required fields', async ({ page }) => {
    await page.click('text=QHosting');
    await page.click('button:has-text("New Task")');

    // Try to submit without filling
    await page.click('button:has-text("Create Task")');

    // Should show validation errors
    await expect(page.locator('text=Title is required')).toBeVisible();
  });
});

```

6.5 CI/CD Pipeline

```

# .github/workflows/ci-cd.yml
name: CI/CD Pipeline

on:
  push:
    branches: [main, develop]
  pull_request:
    branches: [main]

jobs:
  lint:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - name: Setup Node.js
        uses: actions/setup-node@v4
        with:
          node-version: '20'
          cache: 'pnpm'

      - name: Install dependencies
        run: pnpm install --frozen-lockfile

      - name: Run ESLint
        run: pnpm lint

      - name: Run TypeScript check
        run: pnpm tsc --noEmit

      - name: Run Prettier check
        run: pnpm format:check

  test:
    runs-on: ubuntu-latest
    services:
      postgres:
        image: postgres:16-alpine
        env:
          POSTGRES_USER: test
          POSTGRES_PASSWORD: test
          POSTGRES_DB: aurum_test
        options: >-
          --health-cmd pg_isready
          --health-interval 10s
          --health-timeout 5s
          --health-retries 5
      ports:
        - 5432:5432

      redis:
        image: redis:7-alpine
        options: >-
          --health-cmd "redis-cli ping"
          --health-interval 10s
          --health-timeout 5s
          --health-retries 5
      ports:
        - 6379:6379

    steps:
      - uses: actions/checkout@v4

```

```

- name: Setup Node.js
uses: actions/setup-node@v4
with:
  node-version: '20'
  cache: 'pnpm'

- name: Install dependencies
run: pnpm install --frozen-lockfile

- name: Run database migrations
run: pnpm prisma migrate deploy
env:
  DATABASE_URL: postgresql://test:test@localhost:5432/aurum_test

- name: Run unit tests
run: pnpm test:unit --coverage

- name: Run integration tests
run: pnpm test:integration
env:
  DATABASE_URL: postgresql://test:test@localhost:5432/aurum_test
  REDIS_URL: redis://localhost:6379

- name: Upload coverage
uses: codecov/codecov-action@v3
with:
  files: ./coverage/coverage-final.json

e2e:
runs-on: ubuntu-latest
steps:
- uses: actions/checkout@v4

- name: Setup Node.js
uses: actions/setup-node@v4
with:
  node-version: '20'
  cache: 'pnpm'

- name: Install dependencies
run: pnpm install --frozen-lockfile

- name: Install Playwright
run: pnpm exec playwright install --with-deps

- name: Build app
run: pnpm build

- name: Run E2E tests
run: pnpm test:e2e

- name: Upload Playwright report
if: always()
uses: actions/upload-artifact@v3
with:
  name: playwright-report
  path: playwright-report/

security:
runs-on: ubuntu-latest
steps:
- uses: actions/checkout@v4

```

```

- name: Run Trivy vulnerability scanner
  uses: aquasecurity/trivy-action@master
  with:
    scan-type: 'fs'
    scan-ref: '.'
    format: 'sarif'
    output: 'trivy-results.sarif'

- name: Upload Trivy results to GitHub Security
  uses: github/codeql-action/upload-sarif@v2
  with:
    sarif_file: 'trivy-results.sarif'

- name: Run npm audit
  run: pnpm audit --audit-level=high

deploy-staging:
needs: [lint, test, e2e, security]
if: github.ref == 'refs/heads/develop'
runs-on: ubuntu-latest
steps:
- uses: actions/checkout@v4

- name: Deploy to Vercel Staging
  uses: amondnet/vercel-action@v25
  with:
    vercel-token: ${{ secrets.VERCEL_TOKEN }}
    vercel-org-id: ${{ secrets.VERCEL_ORG_ID }}
    vercel-project-id: ${{ secrets.VERCEL_PROJECT_ID }}
    scope: ${{ secrets.VERCEL_ORG_ID }}
    alias-domains: staging.aurum.com

deploy-production:
needs: [lint, test, e2e, security]
if: github.ref == 'refs/heads/main'
runs-on: ubuntu-latest
environment:
  name: production
  url: https://aurum.com
steps:
- uses: actions/checkout@v4

- name: Deploy to Vercel Production
  uses: amondnet/vercel-action@v25
  with:
    vercel-token: ${{ secrets.VERCEL_TOKEN }}
    vercel-org-id: ${{ secrets.VERCEL_ORG_ID }}
    vercel-project-id: ${{ secrets.VERCEL_PROJECT_ID }}
    vercel-args: '--prod'
    scope: ${{ secrets.VERCEL_ORG_ID }}

- name: Run smoke tests
  run: pnpm test:smoke
  env:
    BASE_URL: https://aurum.com

- name: Notify Slack
  uses: 8398a7/action-slack@v3
  with:
    status: ${{ job.status }}
    text: 'Deployment to production completed!'
    webhook_url: ${{ secrets.SLACK_WEBHOOK }}

```

7. Roadmap Priorizado con Estimaciones

Fase 1: Fundamentos y Seguridad (Crítico) - 3 semanas

Objetivo: Establecer base sólida y segura para producción

Tarea	Prioridad	Estimación	Dependencias	Owner
Implementar autenticación enterprise (2FA, rate limiting, session management)	🔴 Crítico	5 días	-	Backend Lead
Configurar RBAC middleware y RLS en PostgreSQL	🔴 Crítico	3 días	Auth completado	Backend Lead
Implementar gestión de secretos (Doppler/Infisical)	🔴 Crítico	2 días	-	DevOps
Setup de Connection Pooling (PgBouncer)	🔴 Crítico	1 día	DB setup	DevOps
Implementar Circuit Breaker para webhooks	🔴 Crítico	3 días	-	Backend Dev
Sistema de Queue (Bull + Redis) para jobs async	🔴 Crítico	4 días	Redis setup	Backend Dev
Encriptación de campos sensibles (PrivacyList)	🔴 Crítico	2 días	-	Backend Dev

Entregables:

- Sistema de autenticación production-ready con 2FA
- Permisos por rol implementados en todas las rutas
- Webhooks resilientes con retry logic
- Datos sensibles encriptados end-to-end

Fase 2: Arquitectura y Performance (Alto) - 3 semanas

Objetivo: Optimizar para escala y mantenibilidad

Tarea	Prioridad	Estimación	Dependencias	Owner
Refactor a Clean Architecture (layers)	● Alto	5 días	-	Architect
Implementar patrón Repository	● Alto	3 días	Clean Arch	Backend Lead
Setup de Redis caching con strategies	● Alto	4 días	Redis disponible	Backend Dev
Optimización de queries DB (índices, N+1)	● Alto	4 días	-	DBA
Implementar Use Cases para lógica de negocio	● Alto	5 días	Repository pattern	Backend Team
Setup de monitoreo (Datadog RUM + APM)	● Alto	3 días	-	DevOps

Entregables:

- Codebase organizado en capas claras
- Cache strategy implementada (5x mejora en response time)
- Queries optimizadas (95% < 100ms)
- Dashboard de monitoreo operacional

Fase 3: Features de Negocio (Alto) - 4 semanas

Objetivo: Entregar valor a usuarios finales

Tarea	Prioridad	Estimación	Dependencias	Owner
Universal Calendar con UI completa	● Alto	5 días	-	Frontend Lead
Smart Scheduler con IA (GPT-4 integration)	● Alto	5 días	Calendar	Fullstack Dev
Helpdesk Unificado con routing inteligente	● Alto	6 días	pgvector setup	Fullstack Dev
CEO War Chest Dashboard	● Alto	4 días	Finance data	Frontend + Backend
Widgets específicos por tipo de satélite	● Alto	6 días	Integraciones n8n	Frontend Team
Privacy Gate con acceso biométrico	● Alto	4 días	Encriptación	Fullstack Dev

Entregables:

- Calendario centralizado con auto-scheduling
- Helpdesk con asignación automática inteligente
- Dashboard ejecutivo con KPIs en tiempo real
- Widgets personalizados por vertical de negocio

Fase 4: Testing y CI/CD (Medio) - 2 semanas

Objetivo: Asegurar calidad y agilidad en deployments

Tarea	Prioridad	Estimación	Dependencias	Owner
Setup de testing framework (Vitest + Testing Library)	🟡 Medio	2 días	-	QA Lead
Tests unitarios para use cases críticos	🟡 Medio	4 días	Testing setup	Backend Team
Tests de integración para API routes	🟡 Medio	3 días	Testing setup	Backend Team
Tests E2E con Playwright (flujos críticos)	🟡 Medio	4 días	Testing setup	QA Team
Setup CI/CD pipeline (GitHub Actions)	🟡 Medio	2 días	-	DevOps
Configurar staging environment	🟡 Medio	1 día	CI/CD	DevOps

Entregables:

- Cobertura de tests > 70% en código crítico
- Pipeline automatizado: test → lint → deploy
- Staging environment idéntico a producción
- Smoke tests post-deployment

Fase 5: UX/UI Excellence (Medio) - 2 semanas

Objetivo: Pulir experiencia de usuario

Tarea	Prioridad	Estimación	Dependencias	Owner
Implementar Optimistic Updates en mutaciones	🟡 Medio	3 días	-	Frontend Dev
Virtual scrolling para listas grandes	🟡 Medio	2 días	-	Frontend Dev
Progressive Web App (PWA) con offline support	🟡 Medio	3 días	-	Frontend Lead
Auditoría de accesibilidad WCAG 2.1 AA	🟡 Medio	2 días	-	UI Designer
Implementar skeleton loaders y suspense	🟡 Medio	2 días	-	Frontend Dev
Animaciones y micro-interactions	🟡 Medio	2 días	-	UI Designer

Entregables:

- UI responsive y fluida (60fps)
 - PWA instalable en mobile/desktop
 - Cumplimiento WCAG 2.1 AA
 - Experiencia premium con animaciones sutiles
-

Fase 6: Integraciones Avanzadas (Bajo) - 3 semanas

Objetivo: Conectar con ecosistema externo

Tarea	Prioridad	Estimación	Dependencias	Owner
Integración bidireccional Odoo (sync real-time)	● Bajo	5 días	n8n setup	Backend Dev
Integración Chatwoot (tickets desde chat)	● Bajo	3 días	Helpdesk	Backend Dev
Integración WHMCS (billing events)	● Bajo	4 días	Finance logs	Backend Dev
Integración Contabo (servidor monitoring)	● Bajo	4 días	Webhooks	Backend Dev
Integración WAHA (WhatsApp messaging)	● Bajo	4 días	n8n	Backend Dev
API pública para third-party integrations	● Bajo	5 días	Auth	Backend Lead

Entregables:

- Sincronización automática con sistemas externos
- Events en tiempo real desde integraciones
- API documentada (OpenAPI/Swagger)
- Rate limiting y API keys para partners

Fase 7: Automatizaciones y IA (Bajo) - 3 semanas

Objetivo: Reducir carga operativa con automation

Tarea	Prioridad	Estimación	Dependencias	Owner
Auto-categorización de tickets con NLP	● Bajo	4 días	Helpdesk	ML Engineer
Detección de anomalías en finanzas (ML)	● Bajo	5 días	Finance data	ML Engineer
Predicción de churn en SaaS satellites	● Bajo	4 días	Odoo integration	ML Engineer
Chatbot interno para queries rápidas	● Bajo	5 días	GPT-4	Fullstack Dev
Auto-generación de reportes ejecutivos	● Bajo	4 días	CEO Dashboard	Backend Dev
Workflow automation builder (low-code)	● Bajo	6 días	n8n integration	Fullstack Dev

Entregables:

- Tickets clasificados automáticamente
 - Alertas proactivas de anomalías
 - Chatbot respondiendo > 60% queries básicas
 - Reportes generados automáticamente cada semana
-



Matriz de Priorización: Impacto vs Esfuerzo

Alto Impacto, Bajo Esfuerzo (QUICK WINS)

- Circuit Breaker para webhooks
- Connection Pooling (PgBouncer)
- Índices de DB
- Redis caching básico
- Skeleton loaders

Alto Impacto, Alto Esfuerzo (PROYECTOS MAYORES)

- Autenticación enterprise (2FA)
- Clean Architecture refactor
- Smart Scheduler con IA
- Helpdesk con routing inteligente
- CEO War Chest Dashboard

Bajo Impacto, Bajo Esfuerzo (FILL-INS)

- Animaciones UI
- PWA manifest
- Formateo de código (Prettier)
- Documentación README

Bajo Impacto, Alto Esfuerzo (EVITAR)

- Reescribir a microservicios (prematuro)
- GraphQL layer (innecesario)
- Custom design system (usar ShadcnUI)
- Blockchain integration (no aporta valor)

🎯 Métricas de Éxito (KPIs)

Performance

- **TTFB (Time to First Byte):** < 200ms
- **FCP (First Contentful Paint):** < 1s
- **LCP (Largest Contentful Paint):** < 2.5s
- **API Response Time (P95):** < 500ms
- **Database Query Time (P95):** < 100ms

Reliability

- **Uptime:** > 99.9% (SLA)
- **Error Rate:** < 0.1%
- **Webhook Success Rate:** > 98%
- **Mean Time to Recovery (MTTR):** < 15 minutos

Security

- **Failed Login Rate:** < 2%
- **Unauthorized Access Attempts:** 0 (detectados y bloqueados)
- **Vulnerability Scan:** 0 critical, 0 high
- **Secrets Rotation:** Cada 90 días

Business

- **User Adoption:** > 90% de usuarios activos semanalmente

- **Task Completion Rate:** > 85%
- **Ticket Resolution Time:** < 4 horas (promedio)
- **CEO Dashboard Usage:** Diario

Developer Experience

- **Build Time:** < 3 minutos
- **Test Coverage:** > 70%
- **Deployment Frequency:** > 10 deploys/semana
- **Lead Time for Changes:** < 24 horas



Riesgos y Mitigaciones

Técnicos

Riesgo	Probabilidad	Impacto	Mitigación
Vendor Lock-in (Vercel)	Media	Alto	Containerizar app (Docker), mantener opción de self-hosting
n8n downtime afecta operaciones	Alta	Crítico	Implementar queue con retry logic, alertas proactivas
Data loss por bug en transacciones	Baja	Crítico	Backups automáticos cada 6h, PITR (Point-in-Time Recovery)
Saturación de DB en peak times	Media	Alto	PgBouncer + read replicas + caching agresivo
API rate limits de servicios externos	Media	Medio	Implementar rate limiting local, queue de requests

Operacionales

Riesgo	Probabilidad	Impacto	Mitigación
Falta de documentación para on-boarding	Alta	Medio	Documentar arquitectura, crear videos de training
Dependencia de un solo desarrollador	Media	Alto	Pair programming, code reviews, knowledge sharing
Scope creep en features	Alta	Medio	Priorización estricta, MVP approach, feedback loops
Resistencia al cambio de usuarios	Media	Medio	Training personalizado, soporte dedicado, iteración UX

Recursos Recomendados

Libros

- **Clean Architecture** - Robert C. Martin
- **Domain-Driven Design** - Eric Evans
- **Designing Data-Intensive Applications** - Martin Kleppmann
- **The DevOps Handbook** - Gene Kim

Herramientas

- **Monitoreo:** Datadog, Sentry
- **Testing:** Vitest, Playwright, Postman
- **CI/CD:** GitHub Actions, Vercel
- **Secrets:** Doppler, Infisical
- **DB Management:** Prisma Studio, pgAdmin, DBeaver

Guías

- [Next.js Production Checklist](https://nextjs.org/docs/going-to-production) (<https://nextjs.org/docs/going-to-production>)
- [PostgreSQL Performance Tuning](https://www.postgresql.org/docs/current/performance-tips.html) (<https://www.postgresql.org/docs/current/performance-tips.html>)
- [Web.dev Performance](https://web.dev/performance/) (<https://web.dev/performance/>)
- [OWASP Top 10](https://owasp.org/www-project-top-ten/) (<https://owasp.org/www-project-top-ten/>)

Conclusión

Aurum Control Center tiene un fundamento sólido con un stack moderno y decisiones arquitectónicas acertadas. Las principales áreas de mejora se centran en:

1. **Seguridad Enterprise:** Autenticación robusta, encriptación, auditoría
2. **Arquitectura Limpia:** Separación en capas, patrones bien definidos
3. **Resiliencia:** Circuit breakers, queues, manejo de errores
4. **Observabilidad:** Monitoreo, logging, alertas proactivas
5. **Performance:** Caching, optimización de queries, connection pooling

Siguiendo el roadmap priorizado, el sistema estará production-ready en **12-14 semanas**, con capacidad de escalar a 100+ usuarios concurrentes y 11 satélites operando sin degradación.

El enfoque debe ser **iterativo e incremental**: entregar valor cada sprint, validar con usuarios reales, y ajustar prioridades basándose en feedback y métricas reales.

Próximos Pasos Inmediatos (Esta Semana):

1. Implementar autenticación con 2FA
2. Setup de PgBouncer para connection pooling
3. Configurar monitoreo básico (logs + error tracking)
4. Agregar índices críticos en DB
5. Implementar Circuit Breaker en webhooks

Contacto para dudas técnicas:

Arquitecto del Sistema | tech-lead@aurum.com

Documento generado el 12 de Diciembre, 2025

Versión 1.0 - Análisis Técnico Completo