

Arquitectura de NEO

Visión General de la Arquitectura

NEO está construido siguiendo una arquitectura moderna de aplicación web full-stack con Next.js 13+, utilizando el nuevo App Router y patrones de diseño escalables.

Arquitectura del Sistema



Componentes Principales

1. Frontend Layer

App Router (Next.js 13+)

- Utiliza el nuevo sistema de routing basado en archivos
- Server Components por defecto para mejor rendimiento
- Client Components solo cuando es necesario
- Layouts anidados para estructura consistente

Componentes UI

```
components/  
├── ui/                                # Componentes base (shadcn/ui)  
│   ├── button.tsx  
│   ├── input.tsx  
│   ├── dialog.tsx  
│   └── ...  
├── layout/                            # Componentes de estructura  
│   ├── header.tsx  
│   ├── sidebar.tsx  
│   └── main-layout.tsx  
├── pages/                             # Componentes de página completa  
│   ├── home-page.tsx  
│   ├── tasks-page.tsx  
│   └── ...  
└── [feature]/                         # Componentes específicos por funcionalidad  
    ├── task-board.tsx  
    ├── calendar-grid.tsx  
    └── ...
```

Gestión de Estado

- **React Context:** Para estado global (proyecto actual, idioma)
- **React Hook Form:** Para gestión de formularios
- **Local State:** Para estado de componentes específicos
- **Server State:** Manejado por Next.js y Prisma

2. API Layer

Estructura de APIs

```

app/api/
├── auth/
│   ├── [...nextauth]/route.ts    # Autenticación OAuth
│   └── signup/route.ts           # Registro de usuarios
├── tasks/
│   ├── route.ts                  # CRUD de tareas
│   ├── [id]/route.ts            # Operaciones específicas
│   ├── bulk-upload/route.ts      # Carga masiva
│   └── validate-csv/route.ts     # Validación CSV
├── projects/
│   ├── route.ts                  # CRUD de proyectos
│   └── [id]/route.ts            # Operaciones específicas
├── meetings/
│   ├── route.ts                  # CRUD de reuniones
│   └── [id]/process/route.ts     # Procesamiento
├── files/
│   ├── route.ts                  # Listado de archivos
│   ├── upload/route.ts           # Subida de archivos
│   └── [id]/process/route.ts     # Procesamiento
└── reports/
    └── generate/route.ts         # Generación de reportes

```

Patrones de API

- **RESTful Design:** Endpoints consistentes y predecibles
- **Type Safety:** Validación con Zod schemas
- **Error Handling:** Manejo centralizado de errores
- **Authentication:** Middleware de autenticación en todas las rutas protegidas

3. Data Layer

Modelo de Datos (Prisma)

```

model User {
  id      String    @id @default(cuid())
  email   String    @unique
  name    String?
  password String
  projects Project[]
  tasks   Task[]
  meetings Meeting[]
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
}

model Project {
  id      String    @id @default(cuid())
  name    String
  description String?
  status  String    @default("active")
  userId  String
  user    User      @relation(fields: [userId], references: [id])
  tasks   Task[]
  meetings Meeting[]
  files    File[]
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
}

model Task {
  id      String    @id @default(cuid())
  title   String
  description String?
  status  String    @default("todo")
  priority String    @default("medium")
  dueDate DateTime?
  projectId String
  userId  String
  project Project @relation(fields: [projectId], references: [id])
  user    User    @relation(fields: [userId], references: [id])
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
}

```

Relaciones de Datos

- **User → Projects:** Un usuario puede tener múltiples proyectos
- **Project → Tasks:** Un proyecto puede tener múltiples tareas
- **Project → Meetings:** Un proyecto puede tener múltiples reuniones
- **Project → Files:** Un proyecto puede tener múltiples archivos
- **Task → Dependencies:** Las tareas pueden tener dependencias entre sí

Flujo de Datos

1. Flujo de Autenticación

Usuario → Login Form → NextAuth → Database → Session → Protected Routes

2. Flujo de Operaciones CRUD

UI Component → Form Submission → API Route → Prisma ORM → Database → Response → UI Update

3. Flujo de Carga de Archivos

File Input → FormData → Upload API → File System → Database Record → UI Feedback

Seguridad

Autenticación y Autorización

- **NextAuth.js:** Manejo seguro de sesiones
- **JWT Tokens:** Para autenticación stateless
- **Password Hashing:** Bcrypt para contraseñas
- **CSRF Protection:** Protección contra ataques CSRF

Validación de Datos

- **Zod Schemas:** Validación en tiempo de ejecución
- **Type Safety:** TypeScript en toda la aplicación
- **Input Sanitization:** Limpieza de datos de entrada
- **SQL Injection Prevention:** Prisma ORM previene inyecciones

Protección de Rutas

```
// Middleware de autenticación
export async function authMiddleware(request: NextRequest) {
  const token = await getToken({ req: request })

  if (!token) {
    return NextResponse.redirect(new URL('/auth/login', request.url))
  }

  return NextResponse.next()
}
```

Rendimiento

Optimizaciones Frontend

- **Server Components:** Renderizado en servidor por defecto
- **Code Splitting:** Carga lazy de componentes
- **Image Optimization:** Next.js Image component

- **Bundle Analysis:** Análisis de tamaño de bundles

Optimizaciones Backend

- **Database Indexing:** Índices en campos frecuentemente consultados
- **Query Optimization:** Consultas eficientes con Prisma
- **Caching:** Estrategias de cache para datos estáticos
- **Connection Pooling:** Pool de conexiones a base de datos

Métricas de Rendimiento

- **Core Web Vitals:** LCP, FID, CLS optimizados
- **Time to Interactive:** Tiempo de carga interactiva
- **Bundle Size:** Tamaño optimizado de JavaScript
- **Database Query Time:** Tiempo de respuesta de consultas



Herramientas de Desarrollo

Linting y Formateo

- **ESLint:** Análisis estático de código
- **Prettier:** Formateo automático
- **TypeScript:** Verificación de tipos
- **Husky:** Git hooks para calidad de código

Testing (Preparado para implementar)

- **Jest:** Framework de testing
- **React Testing Library:** Testing de componentes
- **Cypress:** Testing end-to-end
- **MSW:** Mock Service Worker para APIs

Monitoreo y Debugging

- **Next.js DevTools:** Herramientas de desarrollo
- **Prisma Studio:** Interface visual para base de datos
- **React DevTools:** Debugging de componentes
- **Network Monitoring:** Análisis de requests



Escalabilidad

Arquitectura Modular

- **Feature-based Structure:** Organización por funcionalidades
- **Reusable Components:** Componentes reutilizables
- **Separation of Concerns:** Separación clara de responsabilidades
- **Plugin Architecture:** Extensible mediante plugins

Preparado para Microservicios

- **API First:** APIs diseñadas para ser independientes
- **Database per Service:** Cada módulo puede tener su propia DB
- **Event-Driven:** Preparado para comunicación asíncrona
- **Container Ready:** Dockerización para despliegue

Estrategias de Crecimiento

- **Horizontal Scaling:** Múltiples instancias de la aplicación
- **Database Sharding:** Particionado de datos
- **CDN Integration:** Distribución de contenido estático
- **Load Balancing:** Distribución de carga



Monitoreo y Métricas

Métricas de Aplicación

- **User Engagement:** Tiempo en aplicación, páginas visitadas
- **Feature Usage:** Uso de diferentes funcionalidades
- **Performance Metrics:** Tiempo de carga, errores
- **Business Metrics:** Proyectos creados, tareas completadas

Logging y Debugging

- **Structured Logging:** Logs estructurados para análisis
- **Error Tracking:** Seguimiento de errores en producción
- **Performance Monitoring:** Monitoreo de rendimiento
- **User Analytics:** Análisis de comportamiento de usuarios

Esta arquitectura proporciona una base sólida, escalable y mantenible para NEO, permitiendo el crecimiento futuro y la adición de nuevas funcionalidades de manera eficiente.