Sistema de Gestión Inteligente de Emails

Guía completa y fuente de verdad para todo el desarrollo del sistema.

Última actualización: 9 de Noviembre, 2025
Versión del documento: v2.0.0
Próxima revisión: 16 de Noviembre, 2025

Índice

📋 Reglas del Agente (Instrucciones Fijas)

> **Estas son metainstrucciones críticas que DEBE seguir cualquier IA o desarrollador que trabaje en este proyecto. Son reglas no negociables.**

⚠️ Reglas Críticas de Código

R1. TypeScript Estricto:

- ✅ **SIEMPRE** usa TypeScript en todos los archivos
- ❌ **NUNCA** uses `any` - utiliza tipos específicos o `unknown`
- ✅ Define interfaces para todas las estructuras de datos
- ✅ Exporta tipos desde archivos dedicados en `src/types/`

R2. Arquitectura Smart Actions:

- ✅ **SIEMPRE** usa Server Actions para lógica de backend (carpeta `actions/`)
- ✅ **SIEMPRE** marca Server Actions con `"use server"` al inicio del archivo
- ✅ **SIEMPRE** marca componentes interactivos con `"use client"`
- ❌ **NUNCA** crees endpoints API tradicionales (`/api/`) sin justificación

R3. Sistema de Diseño:

- ✅ **SIEMPRE** usa variables CSS definidas en `globals.css`
- ✅ **SIEMPRE** usa componentes UI del sistema (`src/components/ui/`)
- ✅ **SIEMPRE** usa clases de badge predefinidas (`.badge-categoria-*`, `.badge-prioridad-*`)
- ❌ **NUNCA** hardcodees colores - usa `var(--color-*)`

📁 Reglas de Estructura y Organización

R4. Convenciones de Nomenclatura:

- ✅ Componentes React: `PascalCase.tsx` (ej: `EmailTable.tsx`)
- ✅ Páginas Next.js: `page.tsx`, `layout.tsx`
- ✅ Server Actions: `camelCase.ts` (ej: `emailActions.ts`)
- ✅ Interfaces: `PascalCase` + sufijo descriptivo (`EmailMock`, `ButtonProps`)
- ✅ Constantes: `UPPER_SNAKE_CASE` (ej: `PAGE_SIZE = 10`)

R5. Estructura de Carpetas:

- ✅ **SIEMPRE** respeta la estructura definida en Sección 4
- ✅ Mock data SOLO en `src/lib/mock-data/`
- ✅ Componentes específicos de dominio en carpetas propias (emails/, kanban/, dashboard/)
- ❌ **NUNCA** pongas componentes en carpetas incorrectas

R6. Imports y Dependencies:

- ✅ **SIEMPRE** agrupa imports: React, Next.js, terceros, propios
- ✅ **SIEMPRE** usa paths absolutos con `@/` desde `src/`
- ✅ **SIEMPRE** verifica que las dependencias estén en `package.json`

🔒 Reglas de Validación y Seguridad

R7. Validación de Datos:

- ✅ **ANTES** de procesar datos externos, valídalos con Zod schemas
- ✅ **ANTES** de guardar respuestas de IA, valídalas contra interfaces definidas
- ❌ **NUNCA** confíes en datos del cliente sin validación server-side

R8. Autenticación y Autorización:

- ✅ **SIEMPRE** verifica sesión de usuario en Server Actions
- ✅ **SIEMPRE** filtra datos por `userId` para aislamiento
- ❌ **NUNCA** expongas datos de otros usuarios

🎨 Reglas de UI y UX

R9. Accesibilidad:

- ✅ **SIEMPRE** incluye `aria-label` en botones y controles
- ✅ **SIEMPRE** usa roles semánticos (`button` , `navigation` , `main`)
- ✅ **SIEMPRE** prueba navegación por teclado (Tab, Enter, Escape)

R10. Responsive Design:

- ✅ **SIEMPRE** implementa Mobile First (móvil → tablet → desktop)
- ✅ **SIEMPRE** usa clases responsive (`.hide-mobile` , `.hide-desktop`)
- ✅ **SIEMPRE** testa en breakpoints: 640px, 768px, 1024px, 1280px

R11. Estados de Loading y Error:

- ✅ **SIEMPRE** muestra estados de carga con spinners o skeletons
- ✅ **SIEMPRE** maneja estados de error con `EmptyState`
- ✅ **SIEMPRE** proporciona fallbacks para datos vacíos

📋 Reglas de Planificación y Documentación

R12. Protocolo de Planificación:

- ✅ **ANTES** de desarrollar cualquier feature, sigue el protocolo de hitos
- ✅ **SIEMPRE** divide features en mínimo 3 hitos secuenciales
- ✅ **SIEMPRE** asegura que cada hito sea desplegable independientemente

R13. Actualización de Documentación:

- ✅ **CUALQUIER** cambio de arquitectura DEBE actualizarse en este Sistema Maestro
- ✅ **CUALQUIER** nueva regla DEBE agregarse a esta sección
- ✅ **CUALQUIER** componente nuevo DEBE documentarse en Sección 8
- ✅ **CUALQUIER** Server Action nueva DEBE documentarse en `src/actions/README.md` y actualizarse en Sección 7
- ✅ **CUALQUIER** cambio en base de datos DEBE documentarse en `prisma/README.md` y actualizarse en Sección 5
- ✅ **CUALQUIER** servicio externo nuevo DEBE documentarse en `src/services/README.md` y actualizarse en Sección 10
- ✅ **CUALQUIER** hook personalizado nuevo DEBE documentarse en `src/hooks/README.md` y actualizarse en Sección 13
- ✅ **CUALQUIER** tipo global nuevo DEBE documentarse en `src/types/README.md` y actualizarse en Sección 13
- ✅ **CUALQUIER** test nuevo DEBE documentarse en `src/tests/README.md` y actualizarse en Sección 16

⚡ Reglas de Performance y Optimización

R14. Optimización React:

- ✅ **SIEMPRE** usa `useMemo()` para cálculos costosos (ej: filtering, sorting)
- ✅ **SIEMPRE** usa `useCallback()` para funciones que son props de componentes hijos
- ❌ **NUNCA** hagas fetching de datos en useEffect innecesario

R15. Next.js Best Practices:

- ✅ **SIEMPRE** usa App Router (no Pages Router)
- ✅ **SIEMPRE** prefiere Server Components sobre Client Components
- ✅ Client Components SOLO cuando hay interactividad real (useState, onClick, etc.)

🧪 Reglas de Testing y Quality

R16. Testing (Pendiente Implementación):

- ✅ **ANTES** de marcar un hito como completo, ejecuta smoke tests

- ✅ **SIEMPRE** testa navegación crítica (login, tabla de emails, kanban)
- ✅ **SIEMPRE** valida responsive en móvil real, no solo DevTools

R17. Code Review:

- ✅ **ANTES** de commit, verifica que no haya errores TypeScript (`npm run build`)
- ✅ **ANTES** de commit, verifica que no haya console.log() o alerts de debug
- ✅ **SIEMPRE** usa nombres descriptivos en commits: `feat(emails): add filtering by category`

1. Resumen Ejecutivo y Visión General

1.1 Propósito del Proyecto

El Sistema de Gestión Inteligente de Emails resuelve la sobrecarga de comunicación que enfrentan los ejecutivos comerciales, quienes reciben 50-100 emails diarios mezclando solicitudes importantes con spam y comunicaciones de bajo valor. El sistema automatiza la clasificación mediante IA y organiza tareas implícitas en un tablero Kanban visual.

Problema identificado:

- Volumen abrumador (50-100 emails diarios)
- Pérdida de tiempo en clasificación manual (1-2 horas diarias)
- Gestión ineficiente: tareas implícitas olvidadas
- Falta de visibilidad entre pendientes urgentes vs. informativos

Impacto del problema:

- ❌ Oportunidades de negocio perdidas
- ❌ Clientes insatisfechos por falta de respuesta oportuna
- ❌ Caos operativo en gestión del día a día
- ❌ Estrés y sobrecarga de ejecutivos comerciales

1.2 Solución Propuesta

Sistema inteligente que:

1. **Procesa** emails automáticamente con IA
2. **Extrae** tareas mediante análisis semántico
3. **Organiza** todo en un tablero Kanban visual

1.3 Enfoque MVP vs. Versión Futura

Aspecto	MVP (Semana 1-2)	Versión Futura
Ingesta	Importación manual vía JSON	Integración directa con Gmail API
Procesamiento	Batch manual (usuario selecciona)	Automático + polling/webhooks
Visualización	Tablero Kanban básico	Dashboard avanzado con analytics
Notificaciones	No incluidas	Push notifications + email alerts

1.4 Objetivos del MVP (14 días)

- Validar concepto de clasificación automática
- Demostrar extracción de tareas con IA
- Implementar interfaz Kanban funcional
- Establecer base arquitectónica escalable

2. Stack Tecnológico y Dependencias

2.1 Frontend

Tecnología	Versión	Rol	Justificación / Características
Next.js	16.0.1	Framework Frontend/Backend	App Router, Server Actions, SSR optimizado
React	19.2.0	Componentes de UI	Biblioteca principal para componentes interactivos
TypeScript	5+	Tipado Estático	Type safety, prevención de errores en desarrollo
Tailwind CSS	4+	Estilos Utility-First	Sistema de diseño consistente, CSS moderno

2.2 Backend & Database

Tecnología	Versión	Rol	Justificación / Características
Prisma	6.19.0	ORM	Type-safe queries, migraciones automáticas
Next.js Server Actions	16.0.1	Backend Logic	Eliminación de endpoints API tradicionales
PostgreSQL	Latest	Base de Datos	(Pendiente de implementación - Semana 2)

2.3 UI & Estado

Tecnología	Versión	Rol	Justificación / Características
class-variance-authority	0.7.1	Variantes de Componentes	System de variantes type-safe
clsx	2.1.1	Utilidades CSS	Concatenación condicional de clases
lucide-react	0.552.0	Iconos	Iconografía moderna y consistente
zustand	5.0.8	Estado Global	Gestión de estado simple y performante
tailwind-merge	3.3.1	Merge de clases	Optimización de clases Tailwind

2.4 Inteligencia Artificial

Servicio/Librería	Versión	Propósito	Estado
OpenAI API	6.8.1	Procesamiento de emails y extracción de metadata	Pendiente implementación
Zod	4.1.12	Validación de respuestas IA	Implementado para validación general

2.5 Funcionalidad Específica

Tecnología	Versión	Rol	Estado
@dnd-kit/core	6.3.1	Drag & Drop Kanban	Instalado, implementación pendiente
@tanstack/react-table	8.21.3	Tablas avanzadas	Instalado, no implementado
next-auth	4.24.13	Autenticación OAuth	Instalado, implementación pendiente
notyf	3.10.0	Notificaciones toast	Instalado, toast simulados con alert()
react-loading-skeleton	3.5.0	Estados de carga	Instalado, no implementado
react-spinners	0.17.0	Spinners de carga	Instalado, no implementado

3. Arquitectura del Sistema

3.1 Arquitectura General

El sistema utiliza el **Smart Actions Pattern** de Next.js 15, aprovechando Server Funciones que eliminan la necesidad tradicional de endpoints API.

Capas principales:

- **Presentación**: Componentes React con data fetching directo
- **Lógica de Negocio**: Server Actions (carpeta `actions/`)
- **Servicios**: Integraciones externas (carpeta `services/`)
- **Datos**: Mock data (Semana 1) → Prisma + PostgreSQL (Semana 2+)

3.2 Patrones Implementados

Smart Actions Pattern (Next.js 15)

- ✅ Eliminación de endpoints API tradicionales
- ✅ Type-safety end-to-end

- ✅ Validación centralizada con Zod
- ✅ Revalidación automática de cache

Ventajas implementadas:

- ✅ Menos código boilerplate
- ✅ Validación de datos unificada
- ✅ Cache management transparente

Patrones pendientes de implementación:

- Principios SOLID
- Repository Pattern para servicios
- Observer Pattern para notificaciones en tiempo real

4. Estructura de Carpetas del Proyecto

```
/src
├── app/                      # Rutas y páginas (App Router)
│   ├── (auth)/              # Layouts/páginas de login
│   │   └── login/
│   │       └── page.tsx     # [IMPLEMENTADO] Página de login simulado
│   ├── (protected)/         # Rutas protegidas
│   │   ├── layout.tsx       # [IMPLEMENTADO] Layout con sidebar y header
│   │   ├── dashboard/       # [IMPLEMENTADO] Vista principal con métricas
│   │   │   └── page.tsx
│   │   ├── emails/          # [IMPLEMENTADO] Gestión de emails
│   │   │   ├── page.tsx     # Tabla de emails con filtros
│   │   │   └── [id]/        # [IMPLEMENTADO] Vista detalle de email
│   │   │       └── page.tsx
│   │   ├── kanban/          # [IMPLEMENTADO] Tablero Kanban
│   │   │   └── page.tsx
│   │   └── _playground/     # [IMPLEMENTADO] Área de pruebas de componentes
│   │       └── buttons/
│   │           └── page.tsx
│   ├── layout.tsx           # [IMPLEMENTADO] Root layout básico
│   ├── page.tsx             # [IMPLEMENTADO] Página de inicio
│   └── globals.css          # [IMPLEMENTADO] Sistema de diseño completo
│
├── actions/                 # Server Actions (lógica de negocio)
│   └── README.md            # [PENDIENTE] Implementación para Semana 2
│
├── services/                # Integraciones externas
│   └── README.md            # [PENDIENTE] OpenAI API integration
│
├── lib/                     # Utilidades centrales
│   ├── utils.ts             # [IMPLEMENTADO] Utilidades básicas (cn)
│   └── mock-data/           # [IMPLEMENTADO] Datos de desarrollo
│       ├── emails.ts        # 15 emails con metadata completa
│       ├── navigation.ts    # Configuración del menú
│       └── user.ts          # Usuario demo
│
├── components/              # Componentes UI
│   ├── ui/                  # [IMPLEMENTADO] Componentes base
│   │   └── button.tsx       # Botón con variantes y estados
│   ├── layout/              # [IMPLEMENTADO] Navegación y estructura
│   │   └── index.tsx        # Sidebar, Header, Breadcrumbs, UserMenu
```

```
│   ├── emails/             # [IMPLEMENTADO] Funcionalidad de emails
│   │   ├── EmailTable.tsx   # Tabla con filtros, paginación, ordenamiento
│   │   ├── EmailDetailView.tsx     # [IMPLEMENTADO] Vista detalle
│   │   └── EmailMetadataSidebar.tsx # [IMPLEMENTADO] Sidebar con metadata
│   ├── kanban/             # [IMPLEMENTADO] Tablero Kanban
│   │   ├── KanbanBoard.tsx  # Tablero principal
│   │   ├── KanbanColumn.tsx # Columnas del tablero
│   │   ├── TaskCard.tsx     # Cards de tareas
│   │   └── KanbanFilters.tsx # [IMPLEMENTADO] Filtros por categoría y prioridad
│   ├── dashboard/          # [IMPLEMENTADO] Métricas y dashboard
│   │   └── MetricCard.tsx   # Tarjetas de métricas clickeables
│   └── shared/             # [IMPLEMENTADO] Componentes reutilizables
│       ├── EmptyState.tsx   # Estados vacíos
│       └── SearchBar.tsx    # Barra de búsqueda
│
├── hooks/                  # Custom React hooks
│   └── README.md           # [PENDIENTE] Hooks personalizados
│
├── types/                  # Tipos TypeScript compartidos
│   └── README.md           # [PENDIENTE] Definiciones de tipos globales
│
├── prisma/                 # Base de datos
│   └── README.md           # [PENDIENTE] Schema y migraciones
│
└── tests/                  # Testing
    └── README.md           # [PENDIENTE] Pruebas unitarias e integración
```

4.1 Convenciones de Nomenclatura

Archivos y Carpetas:

- Componentes: PascalCase (`EmailTable.tsx`)
- Páginas: lowercase (`page.tsx` , `layout.tsx`)
- Utilidades: camelCase (`mock-data/` , `utils.ts`)
- Constantes: UPPER_SNAKE_CASE en el código

Componentes:

- Función principal: match del archivo (`EmailTable`)
- Props interfaces: `ComponentNameProps`
- Hooks personalizados: `useFeatureName`

5. Base de Datos y Modelado

5.1 Estado Actual

Semana 1 (Implementado):

- Uso exclusivo de mock data en archivos TypeScript
- No hay conexión a base de datos real
- Datos estructurados pero volátiles

5.2 Modelo de Datos Mock (Implementado)

```typescript
// src/lib/mock-data/emails.ts
interface EmailMock {
  id: string;                    // Identificador único
  from: string;                  // Email del remitente
  subject: string;               // Asunto del email
  body: string;                  // Contenido completo
  receivedAt: string;            // Fecha ISO format
  processed: boolean;            // Si fue procesado por IA
  category: 'cliente' | 'lead' | 'interno' | 'spam' | null;
  priority: 'alta' | 'media' | 'baja' | null;
  hasTask: boolean;              // Si contiene tarea detectada
  taskDescription: string | null; // Descripción de la tarea
  taskStatus: 'todo' | 'doing' | 'done' | null;
}
```

Datos mock implementados:

- 15 emails con variedad completa
- 5 emails sin procesar (processed: false)
- 10 emails procesados con metadata de IA
- 7 emails con tareas distribuidas en estados Kanban
- Categorías balanceadas (cliente, lead, interno, spam)
- Prioridades variadas (alta, media, baja)

5.3 Base de Datos Futura (Pendiente - Semana 2)

Stack planificado:

- PostgreSQL via Neon (producción)
- Prisma ORM para desarrollo
- Migraciones automáticas

Entidades principales identificadas:

- Users (usuarios del sistema)
- Emails (emails importados)
- EmailMetadata (metadata generada por IA)
- Tasks (tareas extraídas)
- EmailCategories (categorías)
- EmailPriorities (prioridades)

6. Autenticación y Autorización

6.1 Estado Actual (Semana 1)

Implementación simulada:

- Login básico sin autenticación real en `src/app/(auth)/login/page.tsx`
- Usuario mock definido en `src/lib/mock-data/user.ts`
- Navegación directa a rutas protegidas sin validación
- Logout simulado con `router.push("/login")`

```
// src/lib/mock-data/user.ts
export const mockUser: UserMock = {
  id: 'user-001',
  name: 'Usuario Demo',
  email: 'demo@emailkanban.com',
  avatar: null,
  role: 'Ejecutivo Comercial'
};
```

6.2 Autenticación Futura (Pendiente - Semana 2)

Stack planificado:

- NextAuth.js 4.24.13 (ya instalado)

- Google OAuth como proveedor principal
- Sesiones persistentes y renovación automática
- Middleware de protección de rutas

Flujo de autenticación planificado:

1. Login via Google OAuth
2. Validación de token y creación de sesión
3. Middleware intercepta rutas protegidas
4. Usuario ve únicamente sus datos (aislamiento por userId)

7. Servicios y Acciones del Backend

7.1 Estado Actual

Smart Actions Pattern:

- Carpeta `actions/` existe pero no implementada
- Lógica actualmente vive en componentes (data fetching mock)
- Validación básica mediante TypeScript interfaces

7.2 Server Actions Planificadas (Semana 2)

Basado en el patrón definido en `doc/GUIA_TRABAJO_CARPETAS.md`:

emailActions.ts (Pendiente):

```
"use server"
import { auth } from "@/lib/auth"
import { prisma } from "@/lib/prisma"
import { revalidatePath } from "next/cache"

export async function getEmails() {
  const session = await auth()
  if (!session) throw new Error("Unauthorized")

  const emails = await prisma.email.findMany({
    where: { userId: session.user.id }
  })

  return emails
}
```

Servicios Externos Planificados:

```
// services/aiService.ts (Pendiente)
export async function processEmailWithAI(email: string, body: string) {
  try {
    const response = await openai.chat.completions.create({
      model: "gpt-4",
      messages: [
        { role: "system", content: AI_PROMPT },
        { role: "user", content: `Email: ${body}` }
      ]
    })

    return parseAIResponse(response)
  } catch (error) {
    throw new AIServiceError("Failed to process email")
  }
}
```

7.3 Validación de Datos

Implementado:

- Zod 4.1.12 instalado para validación runtime
- TypeScript interfaces para desarrollo

Pendiente:

- Schemas Zod para validación de respuestas IA
- Validación de entradas en Server Actions
- Manejo de errores estructurado

8. Componentes UI y Sistema de Diseño

8.1 Sistema de Diseño Implementado

Archivo principal: `src/app/globals.css`

Paleta de colores basada en #607e9d (Slate Blue):

- Primario: `--color-primary-500: #607e9d`
- Secundario: `--color-secondary-500: #10b981` (Verde - Success/Lead)
- Peligro: `--color-danger-500: #ff646a` (Rojo - Spam/Alta Prioridad)
- Advertencia: `--color-warning-500: #f59e0b` (Amber - Media Prioridad)
- Neutros: Escala de grises para texto y bordes

Variables CSS Implementadas:

- Espaciado semántico (4px, 8px, 16px, 24px, 32px, 48px, 64px)
- Tipografía (Inter como fuente base)
- Radios (4px, 6px, 8px, 12px, 16px)
- Sombras (card, hover, dropdown, modal)
- Z-indexes organizados (1000-1070)

8.2 Componentes Base Implementados

Button Component (`src/components/ui/button.tsx`):

- 7 variantes: default, primary, secondary, outline, ghost, link, destructive
- 4 tamaños: sm, md, lg, icon
- Estados: hover, focus, active, disabled, loading
- Accesibilidad: aria-busy, aria-live, soporte teclado
- asChild prop para Next.js Link wrapper

8.3 Componentes Específicos del Dominio

Layout Components (`src/components/layout/index.tsx`):

- ✅ `Sidebar` : Navegación colapsable con estado persistente en localStorage
- ✅ `MobileSidebar` : Overlay responsive para móvil
- ✅ `Header` : Breadcrumbs + hamburger + menú usuario
- ✅ `Breadcrumbs` : Navegación contextual automática
- ✅ `UserMenu` : Dropdown con opciones de usuario
- ✅ `ProtectedShell` : Wrapper principal con todos los layouts

Email Components:

- ✅ `EmailTable` : Tabla con filtros, paginación, ordenamiento, selección múltiple
- ✅ `EmailDetailView` : Vista detallada de email individual
- ✅ `EmailMetadataSidebar` : Panel lateral con metadata IA

Kanban Components:

- ✅ `KanbanBoard` : Tablero principal con 3 columnas
- ✅ `KanbanColumn` : Columna individual con contador
- ✅ `TaskCard` : Card clickeable con navigation
- ✅ `KanbanFilters` : Filtros por categoría y prioridad

Dashboard Components:

- ✅ `MetricCard` : Tarjeta de métricas clickeable

Shared Components:

- ✅ `SearchBar` : Barra de búsqueda reutilizable
- ✅ `EmptyState` : Estados vacíos consistentes

8.4 Badges y Estados Semánticos

Implementado en CSS:

- `.badge-categoria-cliente` : Azul - emails de clientes existentes
- `.badge-categoria-lead` : Verde - prospectos nuevos
- `.badge-categoria-interno` : Gris - comunicaciones internas
- `.badge-categoria-spam` : Rojo - correos no deseados
- `.badge-prioridad-alta` : Rojo - urgente

- `.badge-prioridad-media` : Amarillo - importante
- `.badge-prioridad-baja` : Gris - normal
- `.badge-procesado` / `.badge-sin-procesar` : Estados de procesamiento IA

8.5 Responsive Design

Implementado:

- Breakpoints: 640px, 768px, 1024px, 1280px, 1536px
- Clases utilitarias: `.hide-mobile` , `.hide-tablet` , `.hide-desktop`
- Container responsive padding
- Stack layout en móvil con `.stack-mobile`
- Sidebar colapsable (desktop) → hamburger menu (móvil)

9. Flujos de Datos y Procesos Clave

9.1 Flujo Principal del Usuario (Implementado)

1. Autenticación (Simulada):

```
Login básico → router.push("/emails") → Layout protegido
```

2. Gestión de Emails (`src/components/emails/EmailTable.tsx`):

```
Mock data → Filtros (estado, categoría, búsqueda) → Paginación → Detalle (/emails/[id])
```

3. Visualización Kanban (`src/app/(protected)/kanban/page.tsx`):

```
Mock emails → Filter hasTask === true → Group by taskStatus → Render columns
```

4. Dashboard (`src/components/dashboard/MetricCard.tsx`):

```
Mock data → Calculate metrics → Display cards → Navigation on click
```

9.2 Procesamiento con IA (Planificado - Semana 2)

Estrategia técnica identificada:

- Procesamiento batch (máx 10 emails por tanda)
- Requests concurrentes limitados (3 simultáneos)
- Prompt estrictamente definido con respuesta JSON
- Validación con Zod antes de guardar
- Retry automático (máx 2 intentos por email)
- Estados: procesando → completado/error

9.3 Flujos de Navegación Implementados

Rutas principales:

- `/` → Dashboard principal (métricas + accesos rápidos)
- `/emails` → Lista con filtros y paginación
- `/emails/[id]` → Vista detalle individual
- `/kanban` → Tablero con tareas detectadas
- `/login` → Página de autenticación (simulada)

Navegación contextual:

- Click en email row → `/emails/[id]`
- Click en TaskCard → `/emails/[id]`
- Click en MetricCard → ruta correspondiente (emails, kanban)
- Breadcrumbs automáticos por ruta

10. Integraciones Externas

10.1 Integraciones Instaladas (Pendientes de Implementación)

OpenAI API (6.8.1):

- Propósito: Clasificación automática y extracción de tareas
- Estado: Instalado, no implementado
- Configuración pendiente: API key, prompts, rate limiting

NextAuth (4.24.13):

- Propósito: Autenticación OAuth con Google
- Estado: Instalado, configuración pendiente
- Flujo planificado: Google OAuth → session management

10.2 No Implementado (Fuera de Alcance MVP)

Integraciones con inbox real:

- Gmail API integration
- Outlook/Exchange API
- IMAP/POP3 genérico
- Webhooks automático
- Polling de nuevos emails

Notificaciones externas:

- Email notifications
- Push notifications
- SMS alerts
- Slack/Teams integration

11. Configuración y Despliegue

11.1 Scripts Disponibles (`package.json`)

```
{
  "dev": "next dev --webpack",
  "build": "next build --webpack",
  "start": "next start",
  "lint": "eslint"
}
```

11.2 Variables de Entorno (Pendiente)

Estructura planificada:

```
# Base de datos
DATABASE_URL=
DIRECT_URL=

# Autenticación
NEXTAUTH_SECRET=
NEXTAUTH_URL=
GOOGLE_CLIENT_ID=
GOOGLE_CLIENT_SECRET=

# OpenAI
OPENAI_API_KEY=

# App
NODE_ENV=
```

11.3 Despliegue (Planificado)

Plataformas identificadas:

- **Vercel**: Hosting principal (Next.js optimizado)
- **Neon PostgreSQL**: Base de datos producción
- **GitHub**: Control de versiones y CI/CD

URL objetivo: [Por configurar después del primer deploy]

12. Seguridad y Rendimiento

12.1 Seguridad Implementada

Nivel Frontend:

- TypeScript para validación en compilación
- Sanitización básica en componentes
- No exposure de datos sensibles en cliente

Pendiente (Semana 2+):

- Validación server-side con Zod

- Rate limiting en API routes
- CORS configuration
- SQL injection protection (Prisma)
- XSS protection
- CSRF tokens

12.2 Rendimiento Implementado

Optimizaciones UI:

- Next.js App Router con optimizaciones automáticas
- Lazy loading de componentes
- Memoización con `useMemo()` en componentes críticos
- CSS-in-CSS con variables para temas consistentes

Pendiente:

- Image optimization
- Bundle analysis
- Database query optimization
- Caching strategies (Redis)
- CDN asset delivery

13. Patrones y Convenciones de Código

13.1 Convenciones de Nomenclatura Implementadas

Archivos:

- Componentes React: `PascalCase.tsx` (`EmailTable.tsx`)
- Páginas Next.js: `page.tsx`, `layout.tsx`
- Utilidades: `camelCase.ts` (`utils.ts`)
- Carpetas: `kebab-case` (`mock-data/`)

Variables y Funciones:

```
// Interfaces: PascalCase + Props suffix
interface EmailTableProps { }

// Constantes: UPPER_SNAKE_CASE
const PAGE_SIZE = 10;

// Funciones: camelCase
function formatRelative(iso: string): string { }

// Componentes: PascalCase matching filename
export default function EmailTable() { }
```

13.2 Estructura de Componentes

Patrón implementado:

```tsx
"use client"; // Si requiere interactividad

import { useState, useMemo } from "react";
import { useRouter } from "next/navigation";
// ... otros imports ordenados

// Tipos e interfaces locales
type LocalType = "value1" | "value2";

// Constantes del componente
const COMPONENT_CONSTANT = 10;

// Funciones helper
function helperFunction(param: string): string { }

// Componente principal
export default function ComponentName() {
  const router = useRouter();

  // Estado
  const [state, setState] = useState();

  // Derivados con useMemo
  const derived = useMemo(() => {}, [deps]);

  // Handlers
  function handleAction() { }

  // Render
  return <div>JSX</div>;
}
```

13.3 Gestión de Estilos

CSS Custom Properties (`globals.css`):

```css
:root {
  --color-primary-500: #607e9d;
  --space-md: 1rem;
  --radius-lg: 0.75rem;
}
```

Clases Utilitarias:

```
.badge-categoria-cliente {
  background-color: var(--color-categoria-cliente-bg);
  color: var(--color-categoria-cliente-text);
}

.truncate-2-lines {
  display: -webkit-box;
  -webkit-line-clamp: 2;
  -webkit-box-orient: vertical;
  overflow: hidden;
}
```

13.4 Gestión de Estado

Patrón implementado:

- Estado local con `useState()` para UI state
- `useMemo()` para derivados costosos
- Props drilling para comunicación padre-hijo
- Event handlers para comunicación hijo-padre

Planificado (Zustand 5.0.8):

- Estado global para datos compartidos
- Store principal para user session
- Store secundario para UI preferences

14. Estado Actual y Roadmap

14.1 Funcionalidades Implementadas (Semana 1)

Módulo	Estado	% Implementado	Detalles
Layout y Navegación	✅ COMPLETADO	100%	Sidebar responsive, header, breadcrumbs

Módulo	Estado	% Implementado	Detalles
Sistema de Diseño	✅ COMPLETADO	100%	CSS variables, badges, responsive utilities
Mock Data System	✅ COMPLETADO	100%	15 emails, user, navigation config
Tabla de Emails	✅ COMPLETADO	95%	Filtros, paginación, ordenamiento, responsive
Vista Detalle Email	✅ COMPLETADO	90%	Contenido completo, metadata sidebar
Tablero Kanban	✅ COMPLETADO	85%	3 columnas, filtros, navegación
Dashboard Métricas	✅ COMPLETADO	80%	Cards clickeables, cálculos dinámicos
Componentes Base	✅ COMPLETADO	90%	Button, EmptyState, SearchBar

14.2 Roadmap por Semanas

Semana 1 (Completada - Frontend Only):

- ✅ Setup del proyecto y arquitectura base
- ✅ Sistema de diseño y componentes UI
- ✅ Mock data y navegación completa
- ✅ Todas las vistas principales implementadas
- ✅ Deploy en Vercel functioning

Semana 2 (Planificada - Backend Integration):

- 🔄 Implementar Server Actions
- 🔄 Configurar base de datos (Prisma + PostgreSQL)
- 🔄 Integración OpenAI API para procesamiento
- 🔄 Sistema de autenticación real (NextAuth + Google)
- 🔄 Drag & Drop funcional en Kanban

Semana 3+ (Futuro):

- ⏳ Notificaciones en tiempo real
- ⏳ Integración con Gmail API
- ⏳ Analytics y dashboard avanzado
- ⏳ Testing automatizado
- ⏳ Performance optimization

14.3 Issues Identificados Pendientes

Críticos (Semana 2):

- Mock data no persiste entre sesiones
- Falta validación de datos runtime (Zod)
- No hay manejo de errores estructurado
- Estados de carga simulados con alert()

Mediana Prioridad:

- Drag & Drop visual pero no funcional
- Falta optimización de imágenes
- Testing coverage inexistente
- No hay analytics de uso

Baja Prioridad:

- Dark mode implementado en CSS pero no togglable
- Algunos componentes no utilizan Zustand
- Falta documentación JSDoc en componentes
- Bundle optimization pendiente

15. Protocolo de Planificación

Este proyecto sigue un **protocolo de planificación por hitos** definido en
[`doc/Protocolo de Planificacion.md`](doc/Protocolo\ de\ Planificacion.md).

15.1 Principios Fundamentales

1. **Desarrollo incremental por hitos**: cada semana/feature se divide en hitos independientes pero secuenciales

2. **Entregables concretos**: cada hito produce valor tangible y funcional
3. **Independencia funcional**: cada hito puede existir de forma autónoma en producción
4. **Progresión secuencial**: desarrollo ordenado (Hito 1 → Hito 2 → Hito N)

15.2 Reglas del Protocolo

Cantidad de hitos: Mínimo 3 hitos por feature de complejidad media-alta

Independencia funcional: Cada hito DEBE poder desplegarse a producción independientemente

Desarrollo secuencial: Los hitos se desarrollan en orden estricto

Entregables tangibles: Cada hito debe producir:

- Funcionalidad visible para el usuario final
- Componente reutilizable para el sistema
- Mejora de infraestructura medible
- Documentación técnica completa

15.3 Plantilla de Hito Implementada

```
# HITO [NOMBRE]
## Descripción
- [Qué se logrará específicamente]

## Entregables
- [ ] [Entregable funcional 1]
- [ ] [Entregable funcional 2]

## Tareas
#### Backend/Frontend/Testing
- [ ] [Tarea específica con criterio de completitud]

## Dependencias
- **Internas:** [Hitos previos necesarios]
- **Externas:** [APIs, servicios, etc.]
```

15.4 Estado Actual de Planificación

Semana 1: Cumplió con protocolo de 4 hitos secuenciales

- Hito 1: Setup y diseño → ✅ COMPLETADO

- Hito 2: Mock data y navegación → ✅ COMPLETADO
- Hito 3: Componentes principales → ✅ COMPLETADO
- Hito 4: Integration y deploy → ✅ COMPLETADO

Semana 2: Planificada con 5 hitos identificados

- Hito 1: Base de datos y models
- Hito 2: Server Actions core
- Hito 3: OpenAI integration
- Hito 4: Authentication system
- Hito 5: Performance optimization

Nota: Este documento es vivo y centraliza todo el conocimiento actualizado del sistema.
Todo cambio significativo debe documentarse aquí de inmediato y comunicarse al equipo de desarrollo.

Próxima actualización planificada: Post-implementación Semana 2 (23 Noviembre, 2025)