Especificación Técnica del Frontend del Dashboard: Núcleo Administrativo Central

1. Introducción y Propósito

El presente documento tiene como objetivo principal detallar las especificaciones técnicas para la construcción del Frontend del "Dashboard de Escritorio", componente central del "Ecosistema Digital Inteligente para Cirugía Especial". Servirá como la guía definitiva para los equipos de diseño y desarrollo, asegurando una implementación coherente, de alta calidad y sin ambigüedades.

El Dashboard de Escritorio es el "Núcleo Administrativo Central" del ecosistema, diseñado para ser el corazón operativo del consultorio del Dr. Joel Sánchez García. Su propósito fundamental es centralizar toda la información crítica del consultorio en un solo lugar seguro y accesible, ofreciendo una visión general del estado actual y un acceso rápido a las funcionalidades clave. Su objetivo es optimizar la gestión de pacientes, mejorar la eficiencia operativa al reducir los procesos manuales y el riesgo de errores, y liberar tiempo valioso para el personal. Además, está concebido para ser un entorno digital potente, seguro y excepcionalmente intuitivo, reflejando la calidad y el prestigio de la práctica médica del Dr. Sánchez García.

2. Elección del Framework Frontend

Para el desarrollo del Frontend del Dashboard, se recomienda la utilización de un framework moderno y robusto, siendo React o Vue.js las opciones preferidas y viables. Estos frameworks son ampliamente adoptados en la industria debido a su popularidad, amplios ecosistemas de herramientas y librerías, y su eficiencia en la creación de interfaces de usuario complejas y reactivas. Permiten un desarrollo modular, facilitan la gestión del estado y ofrecen un rendimiento óptimo para aplicaciones dinámicas.

El Dashboard se empaquetará como una aplicación de escritorio utilizando Electron.js. Electron.js permite construir aplicaciones de escritorio nativas (compatibles con Windows y macOS) utilizando tecnologías web estándares como HTML, CSS y JavaScript. Esta elección es estratégica, ya que aprovecha la experiencia en desarrollo web para crear una aplicación de escritorio, lo que agiliza el proceso y garantiza una experiencia de usuario familiar y de alta calidad. La integración con el framework frontend elegido (React/Vue.js) es fluida, permitiendo que la lógica de negocio y la interfaz se desarrollen de manera cohesiva para el entorno de escritorio.

3. Arquitectura y Estructura de la Aplicación

3.1. Diagrama de Arquitectura de Alto Nivel del Frontend y su Interacción

El Frontend del Dashboard de Escritorio actúa como la capa de presentación que permite la interacción del usuario con la lógica de negocio y los datos alojados en la nube. Se comunica de forma segura con el Backend a través de APIs RESTful.

```
graph TD
    A[Aplicación de Escritorio (Frontend)] -- HTTPS (TLS 1.3) --> B(API Gateway / Load Balancer - GCP)
    B --> C[Servidor de Aplicación (Backend)]
    C -- Lógica de Negocio --> D[Base de Datos Relacional (Cloud SQL - GCP)]
    C -- Almacenamiento --> E[Almacenamiento de Archivos (Cloud Storage - GCP)]
    C -- Logs --> F[Servicio de Logs de Auditoría (Cloud Logging - GCP)]
    C -- Integración IA --> G[Modelo de IA (Gemma 3)]

    D -- Cifrado en Reposo --> D
    E -- Cifrado en Reposo --> E
    G -- Consultas/Generación --> C
```

Explicación del Flujo:

1. El Usuario inicia la Aplicación de Escritorio (Frontend).
2. La aplicación se conecta de forma segura a la infraestructura en la nube de Google Cloud Platform (GCP).
3. Todas las solicitudes del Frontend hacia el Backend pasan por el API Gateway / Load Balancer, que actúa como el punto de entrada seguro y distribuye el tráfico eficientemente.
4. La comunicación entre el Frontend y la nube se realiza a través de HTTPS (TLS 1.3) para cifrar los datos en tránsito, protegiendo la información sensible como las credenciales del usuario.
5. El Servidor de Aplicación (Backend) procesa las solicitudes, interactúa con la Base de Datos Relacional (Cloud SQL) para datos estructurados (pacientes, citas, usuarios) y con el Almacenamiento de Archivos (Cloud Storage) para documentos digitales (PDFs, imágenes, videos). Los datos sensibles en ambos servicios de almacenamiento están cifrados en

reposo.
6. Cada acción del usuario y cada intento de login (exitoso o fallido) se registra cronológicamente en el Servicio de Logs de Auditoría (Cloud Logging / Audit Logs), los cuales son inmutables.
7. El Backend también integra el modelo de Inteligencia Artificial Gemma 3 mediante APIs para potenciar funcionalidades de automatización y el chat interno.

3.2. Estructura Detallada de Directorios y Carpetas del Proyecto Frontend

Se propone la siguiente estructura de directorios, basada en principios de modularidad y separación de responsabilidades para facilitar la mantenibilidad, escalabilidad y colaboración del equipo:

```
src/
├── assets/                  # Contiene recursos estáticos como imágenes, iconos, fuentes
│   ├── images/
│   ├── icons/               # Iconos de estilo de línea (line-style) con grosor de trazo consistente [49-52]
│   └── fonts/               # Fuente Inter (sans-serif geométrica) [45, 50-54]
├── components/              # Componentes UI reutilizables y atómicos
│   ├── buttons/
│   ├── forms/
│   ├── inputs/
│   ├── layout/              # Componentes de layout (Header, Sidebar, MainContent) [52, 55-57]
│   └── common/              # Otros componentes generales
├── views/ (o pages/)        # Contiene las vistas principales o "pantallas" de la aplicación
│   ├── Auth/                # Pantalla de Login [3, 7, 28, 58-60]
│   ├── Dashboard/           # Pantalla Principal del Dashboard [4, 7, 8, 31, 36, 58]
│   ├── Patients/            # Gestión Integral del Expediente del Paciente [2, 61-66]
│   ├── Appointments/        # Módulo de Agenda y Citas [67-72]
│   ├── Documents/           # Gestión Documental Avanzada [2, 63, 72-75]
│   ├── Administration/      # Seguridad, Roles y Accesos Rápidos (Gestionar Usuarios, Ver Logs de Auditoría) [68, 71, 76-79]
│   ├── Reports/             # Módulo de Reportes [7, 58, 80-82]
│   └── AIAutomation/        # Inteligencia y Automatización Integrada (Chat Interno) [68, 71, 79, 83-85]
├── services/ (o api/)       # Lógica para interactuar con el Backend (APIs)
│   ├── auth.js              # Servicios de autenticación (Login, Logout) [86, 87]
│   ├── patients.js          # Servicios para la gestión de pacientes [88, 89]
│   ├── appointments.js      # Servicios para la gestión de citas [88, 89]
│   ├── documents.js         # Servicios para la gestión de documentos [88, 89]
│   ├── users.js             # Servicios para la gestión de usuarios y roles [88, 89]
│   └── ai.js                # Servicios para la interacción con la IA (Gemma 3) [88, 89]
├── styles/                  # Estilos globales y específicos
│   ├── globals.css          # Estilos CSS globales, incluyendo las Propiedades Personalizadas de CSS [49-51, 90]
│   ├── themes.css           # Definiciones de temas (Glassmorphism Oscuro) [91, 92]
│   ├── components.css       # Estilos para componentes reutilizables
│   └── views.css            # Estilos específicos para vistas/pantallas
├── utils/                   # Funciones de utilidad (ej. formateo de fechas, validación de formularios)
├── hooks/                   # Custom Hooks (si se usa React) para lógica reutilizable con estado
├── store/ (o context/)      # Gestión de estado global de la aplicación (ej. información del usuario logueado, roles)
└── App.js (o main.js)       # Punto de entrada principal de la aplicación
```

3.3. Descripción de la Lógica de Organización

- Modularidad y Reusabilidad: La aplicación se estructurará en módulos y componentes, lo que permite reutilizar el código en diferentes partes de la aplicación y facilita el desarrollo paralelo. Los componentes se diseñarán de forma atómica, encapsulando su propia lógica y estilos.
- Separación de Responsabilidades:
    - Lógica de UI en Componentes/Vistas: Los componentes y las vistas (carpetas components y views) serán responsables de la presentación visual y la interacción directa

con el usuario.
    ◦ Lógica de Negocio y Llamadas a API en Servicios: Toda la lógica de comunicación con el backend, incluyendo las llamadas a las APIs RESTful y el manejo de la autenticación/autorización, residirá en la carpeta services (o api). Esto asegura que la lógica de presentación esté desacoplada de la lógica de datos.
    ◦ Estilos Separados: Los estilos se gestionarán de forma centralizada en la carpeta styles, utilizando CSS Variables para mantener la coherencia visual del "Glassmorphism Oscuro" y facilitar futuras modificaciones. Esto evita la duplicación de código y asegura que el diseño sea adaptable y fácil de mantener.
• Manejo de Estado: Se implementará una estrategia de manejo de estado global (usando Context API/Redux para React, o Vuex para Vue.js) para gestionar datos críticos como la información del usuario logueado, su rol y los permisos de acceso, asegurando que esta información esté disponible en toda la aplicación de manera eficiente.
• Control de Acceso Basado en Roles (RBAC): El Frontend implementará la lógica para adaptar la interfaz de usuario (visibilidad de secciones, botones, etc.) basándose en el rol del usuario autenticado, cuyos permisos son validados por el Backend. Esto garantiza que solo se presenten las funcionalidades a las que el usuario tiene acceso según su perfil (Administrador Principal, Admin Secundario, Asistente, Invitado).

4.1. Layout y Contenedores
Estos componentes definen la estructura general de la aplicación y sus principales áreas de contenido.
a. Contenedor Principal (App Layout)
• Propósito y función: Sirve como el envoltorio de toda la aplicación de escritorio, estableciendo el fondo principal y la base para el efecto visual "Aurora".
• Variantes: N/A. Es el layout base.
• Estados visuales: N/A. Es un contenedor estático.
• Propiedades (props): children (para contener todos los demás componentes del Dashboard).
• Alineación visual con Glassmorphism: Presenta un fondo principal de gris carbón suave (#121212) para reducir la fatiga visual. Detrás de los paneles de vidrio, incorpora auras difusas de azul oceánico profundo (#4A69FF) y violeta vibrante (#8C52FF), esenciales para el efecto "Aurora" que proporciona el fondo dinámico para el Glassmorphism.
b. Barra de Navegación Lateral (Sidebar)
• Propósito y función: Permite al usuario navegar a las secciones principales del sistema, como "Pacientes", "Agenda", "Reportes", "Documentos" y "Administración". Su contenido se adapta al rol del usuario logueado.
• Variantes: N/A. Es una estructura fija.
• Estados visuales: N/A. Su contenido cambia dinámicamente según el rol.
• Propiedades (props): sections (un array de objetos que define las secciones y sus sub-opciones accesibles para el usuario actual), currentUserRole (para adaptar las opciones de navegación).
• Alineación visual con Glassmorphism: Actúa como un panel lateral que sigue la estética "Glassmorphism Oscuro". Aunque no se especifica explícitamente como "vidrio esmerilado" en sí, se infiere que mantendrá la paleta de colores y la tipografía para una coherencia integral.
c. Cabecera (Header)
• Propósito y función: Ubicada en la parte superior, muestra la información del usuario logueado (nombre y rol) y proporciona la opción para "Cerrar Sesión".
• Variantes: N/A.
• Estados visuales: N/A.
• Propiedades (props): userName, userRole, onLogoutClick (función para manejar el cierre de sesión).
• Alineación visual con Glassmorphism: El texto y los iconos son nítidos y brillantes, utilizando el color principal de texto blanco roto (#F5F5F5) para alta legibilidad sobre el fondo oscuro o semi-transparente.
d. Área de Contenido Principal (Main Content Area)
• Propósito y función: Es el espacio más grande y dinámico del Dashboard, donde se muestran los paneles de información clave ("Citas del Día", "Métricas Rápidas", "Pacientes Recientes") y donde se cargan las vistas de las secciones seleccionadas desde la barra lateral.
• Variantes: N/A. Es el contenedor de contenido principal.
• Estados visuales: N/A. Su contenido cambia dinámicamente.
• Propiedades (props): children (para cargar el contenido de las diferentes secciones del Dashboard).
• Alineación visual con Glassmorphism: Este área es donde los paneles flotantes de "vidrio esmerilado" (Glassmorphism) con esquinas redondeadas y un sutil borde luminoso cobran vida, mostrando la información clave de forma jerárquica y visualmente cómoda.
e. Paneles/Tarjetas (Glassmorphism Panels/Cards)
• Propósito y función: Son los contenedores visuales principales para la información estructurada dentro del Área de Contenido Principal (ej., "Citas del Día", "Métricas Rápidas", "Pacientes Recientes").
• Variantes: Pueden variar en tamaño (ej., pequeños para métricas rápidas, grandes para listados de pacientes).
• Estados visuales: N/A. Son contenedores de información.
• Propiedades (props): title, content (el JSX/componente interno que muestra la información específica), onClick (si el panel es interactivo, como un acceso directo).
• Alineación visual con Glassmorphism: Representan la esencia del diseño "Glassmorphism Oscuro". Son semi-transparentes y desenfocados (usando backdrop-filter: blur(24px)). Tienen esquinas suavemente redondeadas (border-radius: 16px) y un sutil borde luminoso de 1px de color blanco semitransparente que "atrapa" la luz del fondo, creando una sensación de profundidad y jerarquía.

4.2. Componentes de Navegación

Estos componentes permiten la interacción del usuario con la barra lateral.

a. Elemento de Navegación (Sidebar Item)

• Propósito y función: Representa cada una de las opciones en la barra de navegación lateral (ej., "Pacientes", "Agenda", "Reportes"). Permite la selección de la sección principal y, si aplica, la expansión para mostrar sub-opciones.

• Variantes:
    ◦ Con Sub-opciones: Se puede expandir y contraer para mostrar opciones anidadas (ej., "Pacientes" > "Nuevo Paciente", "Buscar Paciente").
    ◦ Sin Sub-opciones: Un elemento de menú directo (ej., "Documentos").

• Estados visuales:
    ◦ Normal: Estado por defecto, icono y texto con el color secundario (#A8A8A8).
    ◦ Hover: Un sutil cambio de color del texto y/o icono a blanco roto (#F5F5F5) o un ligero efecto de brillo/hundimiento al pasar el cursor.
    ◦ Activo/Seleccionado: El elemento actualmente visible en el área de contenido principal. Texto y/o icono con el color primario (#F5F5F5) y posiblemente una barra lateral sutil (1px solid rgba(255, 255, 255, 0.18)) o un fondo sutilmente resaltado.

• Propiedades (props): icon (para el icono de estilo de línea), label (texto de la opción), path (ruta de navegación), hasSubOptions (booleano), subOptions (array de sub-opciones), isActive (booleano para estado activo), onClick (función para manejar la navegación).

• Alineación visual con Glassmorphism: Los iconos son de estilo de línea con un grosor de trazo consistente y terminaciones/esquinas suavemente redondeadas, utilizando el color principal de texto (#F5F5F5) para los estados activos. La tipografía "Inter" se usa para las etiquetas, asegurando legibilidad. La retroalimentación visual al interactuar (hover/activo) es sutil pero clara, manteniendo la estética pulcra y de alta calidad del Glassmorphism.

4.3. Componentes Interactivos Básicos

Estos son los bloques de construcción fundamentales para la interacción del usuario.

a. Botones (Button)

• Propósito y función: Permiten al usuario ejecutar acciones específicas o navegar a otras secciones. Son el elemento principal para la interacción directa.

• Variantes:
    ◦ Primario: Para acciones principales y destacadas (ej., "Iniciar Sesión", "Guardar Paciente", "+ Nuevo Registro"). Utiliza un relleno sólido de color de acento.
    ◦ Secundario: Para acciones menos prominentes, acciones alternativas o botones de cancelación (ej., "Cancelar", "Reprogramar"). Podrían tener un fondo transparente o un borde sutil para menor impacto visual.
    ◦ Terciario/Icono: Para acciones muy discretas o solo con un icono (ej., menú de acciones en tabla, iconos de acceso rápido).

• Estados visuales:
    ◦ Normal: Apariencia por defecto, con colores y sombreado definidos.
    ◦ Hover: Un sutil aumento de brillo o un ligero hundimiento al pasar el cursor, ofreciendo retroalimentación clara.
    ◦ Activo/Focus: Un borde o sombra más prominente, indicando que el botón está seleccionado o en uso.
    ◦ Disabled: Reduce la opacidad y desactiva la interacción (no responde a clics), indicando que la acción no está disponible.
    ◦ Loading: Muestra un spinner o indicador de carga dentro del botón o junto a él, indicando que la acción está en progreso.

• Propiedades (props): onClick (función a ejecutar), label (texto del botón), variant (primario, secundario, etc.), disabled (booleano), loading (booleano), icon (opcional).

• Alineación visual con Glassmorphism:
    ◦ Relleno Primario: Utiliza accent-blue (#4A69FF) o button-primary-fill (#4A69FF).
    ◦ Texto: text-primary (#F5F5F5) para alta legibilidad.
    ◦ Sombra: Un box-shadow suave para un efecto de brillo que contribuye a la sensación de flotación del Glassmorphism.
    ◦ Esquinas: Suavemente redondeadas (ej., border-radius: 8px).

b. Campos de Entrada de Texto (Input / Textarea)

• Propósito y función: Permiten al usuario introducir texto, números u otros datos en la interfaz (ej., "Nombre de Usuario", "Contraseña", campos de perfil de paciente).

• Variantes:
    ◦ Input (tipo text, password, number, email, date, etc.) para campos cortos y de una sola línea.
    ◦ Textarea para campos multi-línea, como "Notas de Evolución".
• Estados visuales:
    ◦ Normal: Apariencia por defecto, con fondo sutilmente contrastante o translúcido.
    ◦ Focus: Un borde o resplandor sutil que indica que el campo está activo para la entrada de texto.
    ◦ Disabled: Desactiva la entrada y reduce la opacidad.
    ◦ Error: Un borde o texto en color de error (ej., rojo) y un mensaje de validación debajo del campo (ej., "Campo 'Nombre' requerido", "Formato de email inválido").
    ◦ Filled: Cuando el campo contiene datos.
• Propiedades (props): value, onChange (función de manejo de entrada), label (etiqueta del campo), placeholder, type (para inputs), rows (para textarea), disabled (booleano), error (booleano), errorMessage (string).
• Alineación visual con Glassmorphism:
    ◦ Fondo: Apariencia limpia y minimalista, posiblemente un fondo sutilmente contrastante o translúcido que mantenga la estética de vidrio.
    ◦ Texto: Las etiquetas y el texto introducido utilizan el color principal de texto (--color-text-primary: #F5F5F5) para una legibilidad óptima sobre el fondo oscuro.
    ◦ Bordes: Posiblemente bordes suaves o inexistentes en estado normal, y un borde sutilmente luminoso en el estado focus para indicar la interacción.

c. Checkboxes (Checkbox)
• Propósito y función: Permiten seleccionar o deseleccionar opciones (ej., selección múltiple de filas en una tabla para acciones en lote).
• Variantes: Individual, Maestro (para seleccionar/deseleccionar todos en una tabla).
• Estados visuales:
    ◦ Sin marcar: Estado por defecto.
    ◦ Marcado: Cuando la opción ha sido seleccionada.
    ◦ Disabled: No permite la interacción.
• Propiedades (props): checked (booleano), onChange (función de manejo de cambio), disabled (booleano).
• Alineación visual con Glassmorphism:
    ◦ Marcado: El color de acento interactivo (--color-accent-interactive: #34D1F3) para el relleno, sin borde.
    ◦ Sin marcar: Fondo transparente y un borde de 1px de color secundario (--color-text-secondary).
    ◦ El contraste visual asegura que los estados sean fácilmente distinguibles en el entorno oscuro.

d. Iconos (Icon)
• Propósito y función: Representan visualmente acciones, categorías o información, mejorando la usabilidad y la comprensión rápida de la interfaz (ej., iconos de navegación, iconos de acciones).
• Variantes: Diversos iconos según su propósito (casa para inicio, lupa para búsqueda, papelera para eliminar, gráfico de barras para reportes, etc.).
• Estados visuales:
    ◦ Normal: Utiliza el color secundario (#A8A8A8) o un tono más tenue.
    ◦ Hover/Activo: Cambia al color principal de texto (#F5F5F5) o a un color de acento, indicando interactividad o selección.
• Propiedades (props): name (nombre del icono), size (tamaño), color (color, que puede ser sobreescrito por el estado), onClick (si el icono es interactivo).
• Alineación visual con Glassmorphism:
    ◦ Estilo: Iconos de estilo de línea (line-style) con un grosor de trazo consistente (aproximadamente 1.5px) y terminaciones/esquinas suavemente redondeadas. Esto contribuye a la estética limpia y moderna.
    ◦ Coloración: Principalmente de un solo color (monocromático), utilizando --color-text-primary o --color-text-secondary para mantener la coherencia y legibilidad sobre los fondos oscuros o semi-transparentes.
    ◦ Biblioteca: Se recomienda el uso de bibliotecas de iconos consistentes como Iconify o Feather Icons para uniformidad y escalabilidad.

e. Elementos de Tipografía (Text / Heading / Label)
• Propósito y función: Estructuran y presentan todo el contenido textual de la interfaz, desde títulos y encabezados hasta párrafos y etiquetas de campos, asegurando legibilidad y

jerarquía.
• Variantes:
    ◦ Heading (H1, H2, etc.) para títulos y secciones importantes.
    ◦ Body para párrafos de texto y contenido principal.
    ◦ Label para etiquetas de campos de entrada o elementos más pequeños.
• Estados visuales: N/A para el componente en sí, pero su color y peso pueden cambiar contextualmente (ej., texto de error, texto deshabilitado).
• Propiedades (props): children (el contenido de texto), variant (H1, body, label), color (ej., primary, secondary, error), weight (bold, regular, etc.).
• Alineación visual con Glassmorphism:
    ◦ Fuente: Inter, una fuente sans-serif geométrica optimizada para UI, seleccionada por su legibilidad en entornos digitales.
    ◦ Colores:
        ▪ --color-text-primary: #F5F5F5 (blanco roto) para títulos, texto principal y elementos clave. Proporciona alto contraste sobre el vidrio oscuro y el fondo principal, superior a 4.5:1 (WCAG).
        ▪ --color-text-secondary: #A8A8A8 (gris claro) para texto secundario, descripciones o ayuda, manteniendo el confort visual.
    ◦ Escala Tipográfica: Definida con rem (ej., H1: 1.5rem/24px Bold; Body: 0.875rem/14px Regular/Bold; Label: 0.75rem/12px Medium).

4.4. Componentes de Visualización de Datos y Estructura Compleja

Estos componentes muestran información estructurada y permiten la interacción avanzada.

a. Tabla de Datos (DataTable)

• Propósito y función: Presenta información en un formato tabular organizado, como "Pacientes Recientes", permitiendo visualizar y gestionar grandes volúmenes de datos de manera eficiente.
• Variantes: Tabla estándar con columnas configurables.
• Estados visuales:
    ◦ Normal: Filas y celdas con estilos predeterminados.
    ◦ Fila Hover: Un ligero cambio de fondo o un sutil resplandor en la fila al pasar el cursor para indicar interactividad.
    ◦ Fila Seleccionada: Resaltado del fondo de la fila para indicar que ha sido seleccionada (ej., mediante un checkbox).
    ◦ Estado Vacío: Un mensaje claro (ej., "No se encontraron pacientes") y posiblemente un icono ilustrativo, en lugar de una tabla vacía.
    ◦ Cargando: Un spinner o una superposición de carga sobre la tabla.
• Propiedades (props): data (array de objetos con los datos a mostrar), columns (definición de columnas: key, label, renderCell, sortable), onRowClick (para ver detalles), onSelectionChange (para checkboxes), isLoading (booleano), emptyMessage (string), pagination (objeto de paginación).
• Alineación visual con Glassmorphism:
    ◦ Contenida dentro de un Panel/Tarjeta Glassmorphism.
    ◦ Encabezados de Columna: Utilizan el color text-primary (#F5F5F5) y la fuente Inter en el peso adecuado.
    ◦ Contenido de Celdas: El texto de datos principal utiliza text-primary. Información secundaria puede usar text-secondary (#A8A8A8).
    ◦ Columna de Identificación: "ID/Nombre del Paciente" en negrita y funciona como enlace para ver el detalle completo.
    ◦ Estados: Las píldoras de estado (State Pills) como "Active" (Verde) y "Inactive" (Rojo/Gris) utilizan colores acentuados para una identificación rápida.
    ◦ Checkbox de Selección: En cada fila y un maestro en el encabezado, siguiendo el estilo Glassmorphism del checkbox.
    ◦ Menú de Acciones por Fila: Un icono de tres puntos vertical (Icon) que al hacer clic despliega un menú contextual flotante (posiblemente un Modal o Panel Lateral con opciones como "Ver Detalles", "Editar Registro", "Archivar", "Eliminar (Suave)").

b. Paginación (Pagination)

• Propósito y función: Permite al usuario navegar a través de grandes conjuntos de datos presentados en tablas o listas, cargando el contenido en bloques (páginas).
• Variantes: Controles de navegación (anterior, siguiente, números de página, selector de tamaño de página).
• Estados visuales:
    ◦ Número de Página Activa: Resaltado con un color de acento o un efecto de brillo sutil para indicar la página actual.

    ◦ Controles Deshabilitados: (ej., "Anterior" en la primera página, "Siguiente" en la última página) con menor opacidad.
• Propiedades (props): currentPage, totalPages, onPageChange (función), pageSize (opcional), onPageSizeChange (opcional).
• Alineación visual con Glassmorphism: Botones y texto siguen el estilo de los demás componentes interactivos, con una retroalimentación visual clara y sutil.
c. Componente de Filtros (Filter Panel/Modal)
• Propósito y función: Proporciona opciones para que el usuario refine la información mostrada en tablas o vistas (ej., filtrar pacientes por fecha de última consulta, tipo de cita).
• Variantes:
    ◦ Panel desplegable: Que se abre in-situ desde un icono de filtro en la barra de acciones de la tabla.
    ◦ Modal: Una ventana emergente con opciones de filtro avanzadas.
    ◦ Panel Lateral: Un panel que se desliza desde un lado de la pantalla (similar a la barra lateral de navegación).
• Estados visuales:
    ◦ Abierto/Cerrado: Transiciones suaves para abrir y cerrar el panel/modal.
    ◦ Filtros Aplicados: Indicadores visuales (ej., un punto de color en el icono de filtro) que muestran que hay filtros activos.
• Propiedades (props): isOpen (booleano), onClose (función), filterOptions (definición de campos de filtro: tipo, label), onApplyFilters (función), onClearFilters (función).
• Alineación visual con Glassmorphism:
    ◦ El Panel o Modal contenedor seguiría la estética de "vidrio esmerilado" del Glassmorphism (semi-transparente, desenfocado, esquinas redondeadas, borde luminoso).
    ◦ Los campos de entrada (Input), checkboxes y botones dentro del componente de filtro seguirían sus respectivos estilos Glassmorphism.
d. Modales (Modal)
• Propósito y función: Presentan contenido importante o solicitan interacción del usuario en una ventana flotante sobre el contenido principal, sin que el usuario abandone la vista actual (ej., formularios de creación/edición de paciente, confirmaciones de eliminación).
• Variantes:
    ◦ Formulario: Para la creación o edición de registros.
    ◦ Confirmación: Para acciones críticas (ej., "¿Desea archivar los X registros seleccionados?").
    ◦ Alerta/Información: Para mensajes importantes o notificaciones.
• Estados visuales:
    ◦ Abierto: Visible sobre un fondo semi-oscurecido (backdrop).
    ◦ Cerrado: Oculto.
• Propiedades (props): isOpen (booleano), onClose (función), title (título del modal), children (contenido del modal), onConfirm (función), onCancel (función), confirmButtonLabel, cancelButtonLabel.
• Alineación visual con Glassmorphism:
    ◦ El modal en sí es un Panel/Tarjeta Glassmorphism: semi-transparente y desenfocado (backdrop-filter: blur(24px)), con esquinas suavemente redondeadas (border-radius: 16px) y un sutil borde luminoso de 1px.
    ◦ El fondo del modal (el backdrop) se oscurecería sutilmente para enfocar la atención en la ventana emergente, manteniendo la sensación de profundidad.
e. Paneles Laterales (Side Panel / Drawer)
• Propósito y función: Ofrecen un espacio adicional para contenido o controles que no requieren una vista de página completa, como detalles adicionales de un elemento seleccionado, formularios de edición o filtros avanzados.
• Variantes: Deslizante desde la izquierda o la derecha.
• Estados visuales:
    ◦ Abierto: Visible, con el contenido principal posiblemente encogido o superpuesto.
    ◦ Cerrado: Oculto.
• Propiedades (props): isOpen (booleano), onClose (función), title (título del panel), children (contenido).
• Alineación visual con Glassmorphism:
    ◦ Similar a un Panel/Tarjeta Glassmorphism, pero extendido verticalmente. Mantendrá la semi-transparencia, el desenfoque y los bordes luminosos.
    ◦ La transición de apertura/cierre podría incluir una sutil animación de deslizamiento.

4.5. Componentes Específicos de Módulos
Estos componentes están diseñados para funcionalidades especializadas.
a. Interfaz del Asistente Virtual (AI Assistant Interface)
• Propósito y función: Permite al personal interactuar con el modelo de IA (Gemma 3) para realizar consultas rápidas, obtener resúmenes de texto o redactar informes. Es un chat interno que solo funciona con texto.
• Variantes: N/A.
• Estados visuales:
    ◦ Normal: Área de visualización de mensajes y campo de entrada activos.
    ◦ Escribiendo: Indicador de que el usuario está introduciendo texto.
    ◦ Recibiendo Respuesta/Cargando: Un indicador visual (ej., puntos suspensivos o spinner) cuando la IA está procesando o generando una respuesta.
    ◦ Errores: Mensajes que indican un problema con la consulta o la IA.
• Propiedades (props): messages (array de objetos {sender, text}), onSendMessage (función), isLoading (booleano), suggestedActions (array de strings para botones de acción sugerida).
• Alineación visual con Glassmorphism:
    ◦ El área de visualización de mensajes y el campo de entrada podrían estar contenidos dentro de un Panel/Tarjeta Glassmorphism.
    ◦ El campo de entrada seguiría el estilo de Input Glassmorphism.
    ◦ Los botones de acción sugerida (ej., "te lo envío a tu correo o prefieres descargarlo?") seguirían el estilo de Button (quizás Secundario o con el color accent-interactive).
b. Componentes del Visor de Archivos Visuales (Visual File Viewer)
• Propósito y función: Permite visualizar y navegar por archivos visuales (imágenes y videos cortos) asociados a los expedientes de los pacientes, como RX o grabaciones de cirugías.
• Variantes:
    ◦ Carrusel Contenedor: Para mostrar miniaturas y navegar entre ellas.
    ◦ Elementos de Miniatura Individual: Representación visual de cada archivo.
    ◦ Vista Ampliada: El área principal donde se muestra la imagen o video seleccionado a tamaño completo.
    ◦ Flechas de Navegación: Para avanzar o retroceder en el carrusel/galería.
    ◦ Controles de Filtro de Origen/Tipo: Botones o checkboxes para filtrar por "Cloud Drives", "Photos", "Videos".
• Estados visuales:
    ◦ Normal: Archivos visibles y navegables.
    ◦ Cargando: Indicador de carga al abrir un archivo grande.
    ◦ Vacío: Mensaje si no hay archivos visuales asociados al paciente.
• Propiedades (props): files (array de objetos de archivo), currentFileIndex (índice del archivo actual), onNavigate (función), onFilterChange (función), selectedFilter (string).
• Alineación visual con Glassmorphism:
    ◦ El Carrusel o Visor principal podría ser un Panel/Tarjeta Glassmorphism, aprovechando la transparencia y el desenfoque para integrarse con el fondo Aurora.
    ◦ Las Miniaturas y la Vista Ampliada estarían contenidas dentro de este panel.
    ◦ Las Flechas de Navegación utilizarían Iconos de estilo de línea.
    ◦ Los Controles de Filtro (botones/checkboxes) seguirían el diseño Glassmorphism de sus respectivos componentes.
c. Componentes para Widgets de Servicios Externos (External Service Widgets)
• Propósito y función: Proporcionan acceso rápido a herramientas de uso común como Gmail, Google Drive, o aplicaciones de Office, integradas visualmente en la interfaz.
• Variantes: Botones con iconos, o pequeños paneles con un icono y una etiqueta.
• Estados visuales:
    ◦ Normal: Icono y etiqueta visibles.
    ◦ Hover: Un sutil brillo o cambio de color al pasar el cursor.
• Propiedades (props): serviceName (nombre del servicio), icon (icono del servicio), onClick (función para abrir el servicio/aplicación), url (opcional, si es una integración web).
• Alineación visual con Glassmorphism:
    ◦ Podrían ser pequeños Paneles/Tarjetas Glassmorphism flotantes, cada uno representando un acceso directo, o simplemente Iconos estilizados con etiquetas de texto.
    ◦ El uso de iconos de estilo de línea y el color text-primary asegurarían la consistencia.

4.6. Componentes de Feedback y Contenedores Generales

a. Componentes de Feedback Visual (Loading Spinner / Toast Notification)

- Propósito y función: Informan al usuario sobre el estado del sistema, como operaciones en curso (carga) o resultados de acciones (notificaciones de éxito/error).
- Variantes:
    - Spinner de Carga: Un indicador animado circular que aparece durante operaciones asíncronas.
    - Mensajes de Notificación/Toast: Pequeños banners que aparecen brevemente para confirmar una acción o mostrar un error (ej., "Guardado con éxito", "Error al subir el archivo").
- Estados visuales:
    - Visible: El componente de feedback se muestra.
    - Oculto: El componente de feedback no es visible.
- Propiedades (props):
    - Spinner: isVisible (booleano), size, color.
    - Toast: isVisible (booleano), message (string), type (success, error, info, warning), duration (tiempo en ms).
- Alineación visual con Glassmorphism:
    - Spinners: Podrían ser círculos sutilmente luminosos o con el color accent-blue o accent-interactive.
    - Toast: Aparecerían como pequeños Paneles/Tarjetas Glassmorphism transitorios, con el desenfoque y los bordes luminosos. El texto dentro utilizaría text-primary y el color del fondo del toast podría variar ligeramente según el tipo (ej., un toque de verde para éxito, rojo para error), manteniendo la opacidad y el desenfoque Glassmorphism.

b. Formularios (Form Container)

- Propósito y función: Agrupan un conjunto de campos de entrada y botones de acción relacionados para la recolección estructurada de datos (ej., "Creación de Perfil de Paciente", "Agendar Nueva Cita").
- Variantes: Formulario de creación, formulario de edición, formulario de búsqueda.
- Estados visuales:
    - Normal: Todos los campos y botones son visibles y están listos para la interacción.
    - Validando/Enviando: Puede deshabilitar los campos o mostrar un spinner en el botón de submit.
    - Error de Validación: Muestra mensajes de error junto a los campos correspondientes.
- Propiedades (props): onSubmit (función al enviar), children (los campos y botones internos), title (título del formulario).
- Alineación visual con Glassmorphism:
    - El formulario en sí estaría contenido dentro de un Panel/Tarjeta Glassmorphism o un Modal (que es un panel Glassmorphism), dependiendo de su ubicación y flujo.
    - Los Campos de Entrada, Checkboxes y Botones dentro del formulario seguirían sus respectivos estilos Glassmorphism, asegurando una experiencia visual coherente y agradable.

5. Implementación del Sistema de Diseño (Glassmorphism Oscuro)
El estilo visual "Glassmorphism Oscuro" es fundamental para la experiencia de usuario del ecosistema, especialmente en el Dashboard de Escritorio, y no es solo estético sino funcional, diseñado para ofrecer comodidad visual y una jerarquía de información clara.
• Estrategia de implementación de estilos: La estrategia de implementación de estilos se centra en el uso de Propiedades Personalizadas de CSS (CSS Variables), declaradas bajo el selector :root. Esta aproximación centraliza los valores de diseño, facilitando la mantenibilidad, los cambios globales y futuras tematizaciones del ecosistema. Aunque no se especifica explícitamente el uso de CSS Modules, SCSS o Styled Components, la adopción de variables CSS sugiere un enfoque modular y escalable para la gestión de estilos.
• Definición y uso de Variables CSS (Tokens de Diseño): Los "tokens de diseño" se codificarán como variables CSS para mantener la coherencia y facilitar su aplicación a través de todos los componentes.
    ◦ Paleta de Colores:
        ▪ --color-background-main: #121212 (Fondo principal, gris carbón suave).
        ▪ --color-surface-glass-base: 29, 35, 50 (Valores RGB para la base del vidrio, azul/gris oscuro desaturado, utilizado con opacidad).
        ▪ --color-border-glass-base: 255, 255, 255 (Valores RGB para el blanco, utilizado para bordes sutiles).
        ▪ --color-text-primary: #F5F5F5 (Texto principal, blanco roto para alta legibilidad).
        ▪ --color-text-secondary: #A8A8A8 (Texto secundario/ayuda, gris claro).
        ▪ --color-accent-blue: #4A69FF (Color de acento primario, azul oceánico profundo).
        ▪ --color-accent-purple: #8C52FF (Color de acento secundario, violeta vibrante).
        ▪ --color-accent-interactive: #34D1F3 (Color para elementos interactivos, azul brillante).
        ▪ --color-button-primary-fill: #4A69FF (Color de relleno para botones primarios).
    ◦ Escala Tipográfica (Fuente: Inter, sans-serif geométrica optimizada para UI):
        ▪ --font-size-h1: 1.5rem (24px), 700 (Bold).
        ▪ --font-size-h2: 1.125rem (18px), 600 (Semibold).
        ▪ --font-size-body: 0.875rem (14px), 700 (Bold) o 400 (Regular).
        ▪ --font-size-label: 0.75rem (12px), 500 (Medium).
        ▪ Los colores de texto principales y secundarios se definirán con --color-text-primary y --color-text-secondary respectivamente.
    ◦ Espaciado y border-radius:
        ▪ --border-radius-card: 16px (Para paneles flotantes, esquinas suaves).
        ▪ --border-radius-button: 8px (Para botones).
        ▪ Aunque no se definen variables específicas para un sistema de espaciado general como --space-unit, el relleno (padding: 24px) se utiliza de manera consistente en los paneles de vidrio.
• Implementación técnica detallada del efecto Glassmorphism en los componentes de panel/tarjeta: Los paneles y tarjetas del Dashboard, como el contenedor de login o las áreas de información clave, implementarán el efecto Glassmorphism utilizando las siguientes propiedades CSS:
    ◦ background-color: rgba(var(--color-surface-glass-base), 0.6): Establece un fondo semi-transparente con un 60% de opacidad sobre el color base de la superficie de vidrio.
    ◦ backdrop-filter: blur(24px): Aplica un desenfoque significativo al contenido que se encuentra detrás del panel, creando el efecto "esmerilado".
    ◦ -webkit-backdrop-filter: blur(24px): Incluye el prefijo de navegador para asegurar la compatibilidad con Safari.
    ◦ border-radius: var(--border-radius-card): Redondea suavemente las esquinas de los paneles.
    ◦ border: 1px solid rgba(var(--color-border-glass-base), 0.18): Delinea los paneles con un borde delicado de 1 píxel, de color blanco semi-transparente (18% de opacidad), que "atrapa" la luz del fondo y define la forma del panel.
    ◦ padding: 24px: Asegura un espaciado interno consistente dentro de los paneles.
• Implementación técnica del fondo Aurora UI: La Interfaz Aurora no es un adorno, sino un componente funcional indispensable que proporciona la "materia prima" visual para que el filtro backdrop-filter del Glassmorphism actúe. Los dos estilos son codependientes: la Aurora UI proporciona la "luz" y el Glassmorphism la "lente".
    ◦ Se creará un fondo dinámico utilizando auras difusas de azul oceánico profundo (#4A69FF) y violeta vibrante (#8C52FF).

    ◦ Técnicamente, esto se logra con elementos div posicionados absolutamente con un fuerte filter: blur() o con múltiples radial-gradients. Este fondo debe tener suficiente variación tonal y de color para que el efecto de vidrio sea visible y efectivo.
• Aplicación de estilos a los componentes reutilizables: El sistema de diseño garantiza una apariencia Glassmorphism coherente en todos los componentes interactivos:
    ◦ Botones (Iniciar Sesión):
        ▪ background-color: var(--color-button-primary-fill): Para un relleno sólido de alto contraste.
        ▪ border-radius: var(--border-radius-button): Esquinas suaves.
        ▪ padding: 12px 24px: Espaciado interno.
        ▪ color: var(--color-text-primary): Para la etiqueta de texto.
        ▪ box-shadow: 0 4px 12px rgba(var(--color-accent-blue), 0.3): Un suave efecto de brillo.
        ▪ La interacción al pulsarlo provocará un sutil aumento de brillo o un ligero hundimiento, ofreciendo una retroalimentación clara.
    ◦ Campos de Entrada (Nombre de Usuario, Contraseña):
        ▪ Diseñados con una apariencia limpia y minimalista, posiblemente con un fondo sutilmente contrastante o translúcido que mantenga la estética de vidrio.
        ▪ El texto de las etiquetas y el que el usuario introduce utilizará --color-text-primary (#F5F5F5) para una legibilidad óptima sobre el fondo oscuro.
    ◦ Casillas de Verificación (Checkboxes):
        ▪ Marcado: background-color: var(--color-accent-interactive), border: none.
        ▪ Sin marcar: background-color: transparent, border: 1px solid var(--color-text-secondary).
    ◦ Tablas (ej., PACIENTES RECIENTES): Aunque no se especifican propiedades Glassmorphism para las tablas en sí, los elementos de texto y los iconos dentro de ellas seguirán la paleta de colores y la tipografía definidas por las variables CSS, utilizando --color-text-primary y --color-text-secondary para mantener la coherencia y legibilidad.
• Uso de Iconografía:
    ◦ Estilo: Los iconos serán de estilo de línea (line-style), con un grosor de trazo consistente (aproximadamente 1.5px) y terminaciones/esquinas suavemente redondeadas.
    ◦ Color: Utilizarán un solo color, var(--color-text-primary), para una excelente integración con el modo oscuro.
    ◦ Recomendación: Se recomienda el uso de bibliotecas como Iconify o Feather Icons para asegurar consistencia y escalabilidad.
• Consideraciones específicas para el modo oscuro avanzado: El diseño del "Glassmorphism Oscuro" incorpora principios avanzados para asegurar la legibilidad y el confort visual en un tema oscuro:
    ◦ Evitar el negro puro: El fondo principal es un gris carbón muy oscuro (#121212), no negro puro. Esto reduce la fatiga visual y permite la percepción de profundidad y sombras sutiles.
    ◦ Colores de acento desaturados: Los tonos de azul (#4A69FF) y violeta (#8C52FF) utilizados para las "Auras" y acentos están desaturados. Esto previene la "vibración" óptica y mantiene un confort visual general, especialmente en exposiciones prolongadas.
    ◦ Texto de alto contraste pero suavizado: El texto principal utiliza un blanco roto o gris muy claro (#F5F5F5), lo que suaviza el contraste general sin comprometer la legibilidad. Esto ayuda a reducir el deslumbramiento y cumple con una relación de contraste superior a 4.5:1 (WCAG), un estándar clave de accesibilidad.
    ◦ Comunicación de profundidad sin sombras: En entornos oscuros, las sombras son ineficaces. El diseño utiliza un sistema multifacético para comunicar la elevación:
        ▪ La luz como indicador de elevación: Las superficies que "se acercan" al usuario se vuelven más claras.
        ▪ La superposición como señal de profundidad: El propio efecto de desenfoque del Glassmorfismo crea una separación perceptual entre el primer plano nítido y el fondo desenfocado.
        ▪ La definición del borde como sustituto de la sombra: El sutil borde luminoso define claramente el contorno del panel, separándolo del fondo donde una sombra no sería visible.
    ◦ Accesibilidad: Se aborda la accesibilidad mediante un enfoque dual de contraste:
        ▪ Contraste de Contenido (Alto): Asegura alta legibilidad entre el texto/iconos y su fondo inmediato (el panel de vidrio oscuro y semi-transparente).
        ▪ Contraste Estructural (Bajo): Mantiene el contraste entre el panel de vidrio y el fondo principal deliberadamente bajo para preservar el efecto flotante y esmerilado.

6. Gestión del Estado

La "Pantalla: Dashboard Principal (Vista General)" actúa como un centro de control dinámico que muestra datos relevantes en tiempo real y proporciona puntos de entrada intuitivos a todas las funcionalidades principales del sistema. Para que la interfaz de usuario (UI) se mantenga sincronizada y reactiva a las acciones del usuario, es fundamental una buena gestión del estado. Dado que el frontend del Dashboard se desarrollará con frameworks como React, Vue.js o similares, la elección de una estrategia de gestión del estado es vital.

Elección de una librería o patrón para la gestión del estado global de la aplicación: Aunque los documentos no especifican una librería concreta, la necesidad de una UI reactiva y sincronizada sugiere la implementación de patrones de gestión de estado robustos.

- Para un frontend basado en React, se podrían considerar:
  - Context API + useReducer: Una solución nativa de React adecuada para la gestión de estados globales de complejidad media. Es ideal para datos de usuario autenticado o estados de modales/paneles que necesitan ser accesibles por muchos componentes sin una prop drilling excesiva.
  - Redux (con Redux Toolkit) o Zustand: Para escenarios de mayor complejidad, donde la gestión de datos asíncronos (llamadas a la API) y la necesidad de un flujo de datos predecible sean prioritarias. Proporciona una "fuente única de verdad" para el estado global y herramientas potentes para depuración y escalabilidad.
- Para un frontend basado en Vue.js, se podrían considerar:
  - Vuex o Pinia: Los gestores de estado oficiales de Vue.js. Proporcionan un almacén centralizado de estado que facilita la reactividad y la depuración, siendo esenciales para aplicaciones Vue a gran escala.

La elección específica dependerá de la complejidad exacta de la lógica de negocio y el volumen de datos que se esperen manejar en tiempo real. Para un "Núcleo Administrativo Central", que es la "columna vertebral y la base de datos central de todo el ecosistema", una solución más estructurada como Redux/Vuex/Pinia podría ofrecer mayor mantenibilidad y escalabilidad a largo plazo.

Descripción de las partes principales del estado que necesitarán ser gestionadas globalmente: Para el Dashboard, varias piezas de información son críticas y requerirán un estado global para garantizar la coherencia y la reactividad:

- Datos del Usuario Autenticado y su Rol: La cabecera del dashboard mostrará el usuario logueado y su rol, y la barra de navegación lateral presentará las secciones a las que tiene acceso según sus permisos. Esta información es fundamental para la adaptación de la interfaz y el control de acceso basado en roles.
- Estado de Carga y Error Global de APIs: Las comunicaciones con el backend pueden presentar "problemas de conectividad" o "errores de servidor". Es crucial tener un estado global para indicar cuándo se están realizando llamadas a la API (estado loading, para mostrar spinners o deshabilitar botones) y cuándo ha ocurrido un error (estado error, para mostrar mensajes de error genéricos).
- Datos de la Tabla de Pacientes y Otros Datos Dinámicos: La sección "PACIENTES RECIENTES" y las "CITAS DEL DÍA" muestran datos dinámicos que se actualizan en tiempo real o casi real. Estos datos, junto con las "MÉTRICAS RÁPIDAS", deben ser parte del estado global para que cualquier componente que los necesite pueda acceder a ellos de forma consistente.
- Estado de Visibilidad de Modales/Paneles: Si bien no se menciona explícitamente en los documentos, una interfaz de usuario compleja como el Dashboard a menudo utiliza modales o paneles laterales (como los de filtrado o creación de nuevos registros) cuya visibilidad puede ser gestionada a nivel global para una mejor coordinación entre componentes.
- Contenido de Texto Generado por IA: Las funcionalidades de Inteligencia y Automatización Integrada permiten generar texto (ej., borradores de correos, resúmenes de texto). Este contenido, antes de ser utilizado o guardado, podría residir temporalmente en el estado global.

Cómo se actualizará y accederá al estado desde diferentes componentes: La forma en que se actualiza y accede al estado dependerá del patrón o librería de gestión de estado elegida:

- Actualización:
  - Acciones/Despachos: En patrones como Redux o Vuex, los componentes "despachan" (dispatch) acciones. Estas acciones son objetos que describen lo que sucedió (ej., USER_LOGIN_SUCCESS, FETCH_PATIENTS_START, FETCH_PATIENTS_SUCCESS). Los "reducers" (en Redux) o "mutations" (en Vuex) son funciones puras que toman el estado actual y una acción, y devuelven un nuevo estado, asegurando un flujo de datos unidireccional y

predecible.
    ◦ Actualizadores de estado: Para Context API + useReducer, se utilizan funciones dispatch para enviar acciones al reducer. Para estados locales o de menor alcance, se usarán funciones setState de React o ref/reactive de Vue.
• Acceso:
    ◦ Selectores/Hooks: Los componentes accederán a porciones específicas del estado global mediante "selectores" (en Redux) o "hooks personalizados" (en React con Context/Zustand), o mediante funciones mapState o useStore (en Vuex/Pinia). Esto les permite "suscribirse" solo a la parte del estado que les interesa y re-renderizarse solo cuando esa parte cambia, optimizando el rendimiento.
7. Consumo de APIs del Backend
El Dashboard de Escritorio se comunica con el "Servidor de Aplicación (Backend)" a través de APIs RESTful. Este backend está alojado en Google Cloud Platform (GCP) y maneja la lógica de negocio, la interacción con la base de datos y otros servicios.
Estrategia para realizar las llamadas a los endpoints del backend desde el frontend: Se recomienda una estrategia que garantice la robustez, mantenibilidad y seguridad de las comunicaciones:
• Módulo de Servicios Dedicado: En lugar de realizar llamadas fetch o axios directamente desde los componentes, se creará un módulo de servicios o una capa de abstracción para las APIs. Este módulo contendría funciones específicas para cada endpoint (ej., api.getPatients(), api.createPatient(data), api.login(credentials)).
    ◦ Librería HTTP: Se puede utilizar fetch (API nativa del navegador) o una librería popular como axios. axios es preferible por su simplicidad, manejo automático de JSON, intercepción de solicitudes/respuestas y mejor manejo de errores.
• Endpoints: Se deben desarrollar endpoints de API robustos para las diferentes funcionalidades del Dashboard, como:
    ◦ GET /patients con parámetros de paginación, búsqueda y filtrado.
    ◦ POST /patients para crear nuevos registros.
    ◦ PUT /patients/{id} para actualizar.
    ◦ DELETE /patients/{id} para el borrado suave (soft delete).
    ◦ Endpoints específicos para la galería de archivos y las interacciones con el asistente virtual de IA.
    ◦ POST /api/v1/auth/login para la autenticación.
Cómo se manejarán los estados de carga (loading), error (error), y éxito (data) de las llamadas a la API en la interfaz de usuario: La gestión visual de estos estados es crucial para una buena experiencia de usuario:
• Estado de Carga (loading):
    ◦ Indicadores Visuales: Durante las solicitudes a la API, la UI mostrará indicadores de carga (ej., spinners, esqueletos de contenido, o deshabilitación de botones) para comunicar al usuario que la operación está en curso y que el sistema no está "congelado". Esto es especialmente relevante cuando hay "problemas de conectividad" o el "servidor no puede comunicarse".
• Estado de Error (error):
    ◦ Mensajes Clares y Concisos: En caso de fallos en la API (ej., credenciales incorrectas, campos vacíos, o errores de servidor), se mostrarán "mensajes de error claros y concisos" al usuario. Estos mensajes pueden ser notificaciones flotantes, textos debajo de los campos de formulario, o alertas modales.
    ◦ Registro en Logs: Los errores del frontend también podrían registrarse en el "Servicio de Logs de Auditoría" si son críticos, para monitorear problemas de rendimiento o seguridad.
• Estado de Éxito (data):
    ◦ Actualización Reactiva de la UI: Una vez que una llamada a la API es exitosa, los datos recibidos se utilizarán para actualizar el estado global o local pertinente. Esto provocará que los componentes relevantes se re-rendericen automáticamente, mostrando la información actualizada (ej., una nueva entrada en el historial clínico, un nuevo paciente en la tabla, o un cambio de estado en una cita).
Cómo se gestionarán los tokens de autenticación: La seguridad de las credenciales es una "preocupación de seguridad de datos" clave. El proceso de login autentica al usuario y aplica el control de acceso basado en roles. La comunicación se realiza mediante HTTPS (TLS 1.3) para cifrar los datos en tránsito.
• Generación del Token: Tras un login exitoso, el backend emitirá un token de autenticación (probablemente un JSON Web Token o JWT, que es un estándar de la industria). Este token representará la sesión del usuario y sus permisos de rol.

• Almacenamiento Seguro en el Frontend: La forma de almacenar el token es crítica para la seguridad:
    ◦ sessionStorage: Más seguro que localStorage porque los datos se borran cuando la sesión de la pestaña se cierra. Adecuado para sesiones más cortas.
    ◦ localStorage: Permite persistir la sesión entre cierres del navegador, lo que mejora la conveniencia del usuario. Sin embargo, es más vulnerable a ataques XSS (Cross-Site Scripting). Si se usa, es esencial tener estrictas medidas de seguridad XSS y considerar tiempos de expiración cortos para los tokens.
    ◦ Cookies HTTP-Only: Considerado el método más seguro para gestionar sesiones, especialmente si se usa con Secure y SameSite flags. El token se envía automáticamente en cada solicitud al backend sin que el JavaScript del frontend pueda acceder a él directamente, mitigando riesgos de XSS. El backend sería responsable de establecer estas cookies.
• Inclusión en los Headers de las Solicitudes a la API: Para cada solicitud subsiguiente al login que requiera autenticación, el token se incluirá en el encabezado Authorization como un token Bearer (ej., Authorization: Bearer <your_token>). Esto permitirá al backend verificar la identidad y los permisos del usuario antes de procesar la solicitud, garantizando el "control de acceso basado en roles".

8. Implementación de Funcionalidades de UI Complejas:
La interfaz del Dashboard de Escritorio, diseñada bajo la estética "Glassmorphism Oscuro", no es solo visualmente atractiva, sino que también integra lógicas complejas para ofrecer una experiencia de usuario robusta y eficiente. La implementación de estas funcionalidades requiere una atención meticulosa a la interacción entre el frontend (desarrollado con frameworks como React o Vue.js y empaquetado con Electron.js) y el backend (Python/Flask/Django o Node.js/Express en GCP).

- Tabla de Datos Interactiva (ej. Pacientes Recientes, Citas del Día):
  - Estructura y Elementos: La tabla de datos principal muestra información crítica como "ID/Nombre del Paciente", "Fecha de Última Consulta", "Diagnóstico Principal", "Médico Tratante" y un "Checkbox de Selección" por fila. También incluye un "Menú de Acciones" contextual por cada fila con opciones como "Ver Detalles", "Editar Registro", "Ver Historial Clínico Completo" o "Archivar/Eliminar (Suave)". La cabecera incluye un "Checkbox maestro" para selección/deselección de todo.
  - Paginación y Ordenamiento: Aunque no se especifica explícitamente en los detalles de la tabla, para manejar eficientemente la información en un "Núcleo Administrativo Central", es común que se implemente paginación. El ordenamiento podría realizarse en el frontend para conjuntos de datos pequeños, pero para grandes volúmenes de pacientes o citas, sería más eficiente delegar la lógica de ordenamiento (y filtrado complejo) al backend a través de parámetros en las peticiones API (ej., GET /pacientes?page=1&sort=nombre&order=asc).
  - Filtrado: La tabla cuenta con un "Icono de filtro" que despliega un panel con "opciones avanzadas para filtrar la tabla (por fecha, por estado, por tipo de consulta, etc.)". La lógica de filtrado se implementará en el frontend para construir dinámicamente los parámetros de consulta que se enviarán al backend a través de APIs RESTful. El backend procesará estos filtros contra la "Base de Datos Relacional (Cloud SQL)" para devolver solo los datos relevantes.
  - Selección Múltiple y Acciones en Lote: Los checkboxes permiten la selección múltiple. La lógica del frontend detectará los registros seleccionados y habilitará botones de acción en lote (ej., "Eliminar" para "borrado suave"). Al activar estas acciones, se enviará una petición API al backend con los IDs de los registros afectados, y el backend ejecutará la lógica de borrado suave (marcando is_deleted = true en la base de datos en lugar de eliminar permanentemente).
  - Menú de Acciones por Fila: Este menú contextual se mostrará al hacer clic en un "Icono de tres puntos". La lógica del frontend gestionará la visibilidad y las acciones asociadas a cada ítem del menú (ej., redirigir a una vista de detalle, abrir un formulario de edición).
- Interacción con el Asistente Virtual (Gemma 3):
  - Mecanismo de Interacción: El usuario interactúa con el chat interno a través de un "campo de entrada de texto". Al enviar el texto, la aplicación de escritorio (frontend) enviará esta consulta de usuario al Servidor de Aplicación (backend) a través de una API segura.
  - Procesamiento y Respuesta: El backend, donde se integra el modelo de IA "Gemma 3" (o similar a Gemini) mediante APIs, procesará la consulta del usuario. La IA generará una respuesta (ej., resúmenes de texto, redacción de informes, búsqueda inteligente de archivos). El backend devolverá esta respuesta al frontend, que la mostrará en tiempo real o casi real en el "área de visualización de conversación" del chat. Es importante recordar que este chat "solo funciona con texto y no genera archivos de ningún formato".
  - Automatización de Tareas: El Asistente Virtual también está "conectado a las APIs del sistema para ejecutar acciones como enviar correos, generar borradores de documentos, crear proyectos, etc.". Esto implica que, además de la interacción textual, la IA puede invocar otras funcionalidades del backend basadas en la intención del usuario.
- Implementación del Visor de Archivos Visuales:
  - Funcionalidad: Este widget permite visualizar imágenes y videos relacionados con los pacientes. El contenido se actualiza contextualmente según el paciente seleccionado o la búsqueda realizada.
  - Carga de Miniaturas y Vista Ampliada: Las miniaturas se cargarán de forma eficiente para el "Carrusel de Vistas Previas". Al seleccionar una miniatura, se cargará la vista ampliada del archivo. Los archivos digitales (PDF, imágenes, videos cortos) se almacenan en "Cloud Storage (S3/Blob) en GCP", y "estarán cifrados en reposo". La recuperación de estos archivos desde Cloud Storage se hará a través de APIs del backend.
  - Navegación y Filtros: Las flechas del carrusel (< y >) permitirán la navegación

entre archivos. Se implementarán filtros (ej., "Cloud Drives", "Photos", "Videos") para refinar la visualización. La lógica del frontend aplicará estos filtros a la lista de archivos obtenida del backend, o bien el backend los aplicará a nivel de base de datos/almacenamiento para optimizar la carga.

- Integración de Widgets de Servicios Externos (Gmail, Google Drive, Office):
  - Nivel de Integración: El Dashboard incluye "widgets para acceso rápido a Gmail y Google Drive, así como accesos directos para abrir aplicaciones de Office". Dada la descripción, la integración inicial se centrará en la conveniencia:
    - Gmail y Google Drive: Los widgets probablemente abrirán las respectivas aplicaciones web en el navegador predeterminado del usuario o en una ventana web integrada si Electron.js lo permite de forma segura. Si se requiere una integración más profunda (ej. listar correos, buscar archivos directamente en el Dashboard), se explorará el uso de las APIs de Google Workspace (como Gmail API, Google Drive API). Esto requeriría una gestión de OAuth 2.0 y tokens de acceso para la autenticación del usuario.
    - Aplicaciones de Office: Los "accesos directos" se interpretan como la capacidad de lanzar las aplicaciones de escritorio de Office (Word, Excel, PowerPoint) instaladas localmente en el sistema operativo del usuario, o abrir documentos de Office en la web si el usuario lo tiene configurado. No se implica una integración profunda a nivel de contenido o edición dentro del Dashboard.
  - Seguridad: Cualquier integración más allá de un simple enlace deberá asegurar la comunicación a través de HTTPS/TLS 1.3 y manejar las credenciales de forma segura, respetando las políticas de privacidad y los permisos de acceso del usuario.
- Manejo de Formularios (ej. Creación de Perfil de Paciente):
  - Validación del Lado del Cliente: Antes de enviar los datos al backend, el frontend implementará validaciones para asegurar que "campos obligatorios" no estén vacíos (ej., "Campo 'Nombre' requerido") y que el "formato" de los datos sea correcto (ej., "Formato de email inválido"). Esto se realizará mediante lógica JavaScript/TypeScript en el framework de frontend (React/Vue.js).
  - Validación del Lado del Servidor: Aunque se realice una validación inicial en el frontend, el backend siempre replicará y reforzará estas validaciones para garantizar la integridad y seguridad de los datos antes de persistirlos en la base de datos.
  - Retroalimentación al Usuario: Los "mensajes de validación" se mostrarán claramente en la interfaz de usuario, guiando al personal sobre cómo corregir los errores en los datos ingresados.

9. Consideraciones de Rendimiento y Optimización del Frontend:

Dado que el Dashboard de Escritorio es el "Núcleo Administrativo Central", su rendimiento y capacidad de respuesta son fundamentales para la eficiencia operativa del consultorio. Se implementarán diversas estrategias de optimización:

- Optimización de la Carga Inicial:
  - Code Splitting (División de Código): Se dividirá el "bundle" (paquete) de JavaScript, CSS y otros activos en módulos más pequeños. Esto permite que el navegador solo cargue el código necesario para la vista actual, reduciendo el tiempo de carga inicial. Por ejemplo, los módulos de "Administración" o "Reportes", que pueden no ser utilizados por todos los roles o con tanta frecuencia, pueden cargarse de forma diferida.
  - Lazy Loading (Carga Perezosa) de Componentes no Críticos: Los componentes de la interfaz de usuario que no son visibles de inmediato al cargar el Dashboard (ej., modales, secciones ocultas, widgets menos utilizados) se cargarán solo cuando sean necesarios. Esto mejora el tiempo de "Time to Interactive" (TTI), haciendo que la aplicación sea utilizable más rápidamente.
  - Minificación y Compresión: Todos los archivos de código y activos se minificarán (eliminando espacios en blanco y caracteres innecesarios) y se comprimirán (ej., con Gzip o Brotli) antes de su despliegue para reducir el tamaño de la transferencia de datos.
  - Uso Eficiente de CDN (Content Delivery Network): Aunque no se menciona explícitamente, los activos estáticos del frontend (imágenes, CSS, JS) podrían distribuirse a través de una CDN para reducir la latencia de carga para los usuarios, aprovechando los servidores más cercanos geográficamente.
- Optimización del Renderizado de Listas Grandes:
  - Virtualización de Listas/Tablas: Aunque las secciones como "Pacientes Recientes" o "Citas del Día" pueden no tener miles de filas inicialmente, para asegurar la escalabilidad a medida que el consultorio crezca, se considerará la implementación de la virtualización de listas. Esto implica renderizar solo las filas de la tabla que son visibles en la ventana de visualización del usuario, en lugar de renderizar todas las filas a la vez. Esto reduce significativamente el uso de memoria y mejora el rendimiento

del scroll para conjuntos de datos extensos.
    ◦ Memoización de Componentes: En frameworks como React o Vue.js, se utilizará la memoización (ej., React.memo o Vue.js keep-alive) para evitar el re-renderizado innecesario de componentes que no han cambiado sus propiedades (props) o estado.
• Manejo Eficiente de Imágenes y Videos:
    ◦ Optimización de Formatos y Compresión: Los archivos de imágenes y videos se almacenarán en "Cloud Storage" en formatos web optimizados (ej., WebP para imágenes, MP4 con codecs eficientes para videos) y se comprimirán adecuadamente para reducir su tamaño sin comprometer excesivamente la calidad.
    ◦ Carga Adaptativa (Responsive Images/Videos): Las imágenes y videos se servirán en diferentes resoluciones o calidades, eligiendo la más apropiada según el tamaño de la pantalla del usuario y la velocidad de su conexión a internet.
    ◦ Streaming y Carga Diferida de Videos: Los videos se cargarán mediante streaming, permitiendo que la reproducción comience antes de que se descargue el archivo completo. Para la funcionalidad del "Visor de Archivos Visuales", los videos solo se cargarán y reproducirán cuando el usuario interactúe con ellos.
    ◦ Caché del Navegador: Se configurarán encabezados de caché HTTP adecuados para los activos multimedia, permitiendo que el navegador los almacene localmente y los recupere más rápidamente en visitas posteriores.
• Requisito de Conectividad: Es fundamental enfatizar que "se recomienda una conexión a internet estable de al menos 250 Mbps para un rendimiento óptimo del ecosistema", ya que el sistema "depende de la comunicación constante con la infraestructura en la nube". Esta es una consideración clave para la expectativa del usuario sobre el rendimiento.
Estas implementaciones técnicas y estrategias de optimización garantizarán que el Dashboard de Escritorio sea no solo seguro y funcional, sino también rápido, fluido y escalable, proporcionando una experiencia de usuario de alta calidad.

10. Consideraciones Específicas de Electron

El "Núcleo Administrativo Central" o Dashboard de Escritorio del "Ecosistema Digital Inteligente para Cirugía Especial" es una aplicación de escritorio. Para su desarrollo, el frontend se construirá utilizando frameworks web como React o Vue.js, y se empaquetará con Electron.js. Electron.js es una tecnología que permite construir aplicaciones de escritorio multiplataforma (compatibles con Windows y macOS) utilizando tecnologías web como HTML, CSS y JavaScript.

Esto implica varias consideraciones específicas:

- Manejo de funcionalidades específicas de escritorio: Aunque la mayoría de la información crítica y los archivos se almacenarán en Cloud Storage con cifrado en reposo, minimizando la necesidad de acceso a archivos locales, Electron.js proporciona la capacidad de interactuar con el sistema operativo nativo. Esto podría ser útil para funcionalidades futuras, como la gestión de impresoras o la integración con software local si fuera necesario. Sin embargo, el enfoque principal de almacenamiento es la nube para seguridad y centralización.
- Comunicación entre procesos de Electron: Una aplicación Electron se compone de un proceso principal (que maneja la ventana y las interacciones del sistema operativo) y procesos de renderizado (donde se ejecuta la interfaz de usuario web) [No se especifica explícitamente en las fuentes, pero es inherente a Electron]. Una implementación efectiva debe asegurar una comunicación fluida y eficiente entre estos procesos, especialmente cuando la interfaz de usuario necesita invocar funcionalidades del sistema o recibir información de ellas.
- Proceso de empaquetado y distribución: El uso de Electron requiere un proceso para empaquetar la aplicación final en formatos ejecutables para Windows (.exe) y macOS (.dmg) [No se especifica explícitamente en las fuentes, pero es un paso necesario en el desarrollo de Electron]. Esto incluye la configuración de instaladores y el manejo de dependencias.
- Estrategia de actualizaciones de la aplicación de escritorio: Para mantener el sistema seguro y actualizado, es fundamental definir una estrategia clara para la distribución de nuevas versiones y parches [No se especifica explícitamente en las fuentes, pero es una buena práctica para aplicaciones de escritorio]. Electron admite mecanismos de auto-actualización que pueden simplificar este proceso para el usuario final.

La elección de Electron subraya el compromiso con una aplicación de escritorio robusta que aprovecha la agilidad del desarrollo web, facilitando la integración con el backend en la nube.

Siguiente paso: Podríamos explorar las ventajas y desventajas de Electron frente a un desarrollo nativo puro para aplicaciones de escritorio, en términos de costos de desarrollo y rendimiento, para una visión más completa.

11. Diseño Responsivo y Adaptabilidad

El diseño del Dashboard de Escritorio está profundamente anclado en el estilo visual "Glassmorphism Oscuro", que no es solo una cuestión estética, sino funcional, diseñada para ofrecer comodidad visual y una jerarquía de información clara.

Aunque el Dashboard es una aplicación de escritorio y no requiere un "diseño responsivo" en el sentido de adaptarse a pantallas de teléfonos móviles o tabletas (ya que no es su propósito principal), sí debe ser adaptable a diferentes resoluciones de monitor dentro del entorno de escritorio.

Esto se logra mediante la aplicación de los principios del Glassmorphism Oscuro y las buenas prácticas de diseño de interfaz de usuario (UI/UX):

• Coherencia Visual Integral: El diseño del Dashboard mantiene una coherencia visual con la pantalla de Login y el resto del ecosistema, asegurando una experiencia de usuario fluida y de alta calidad desde el primer punto de interacción.

• Comodidad Visual y Legibilidad Óptima: El fondo principal de gris carbón suave (#121212) está diseñado para reducir la fatiga visual, evitando el negro puro. El texto y los iconos son nítidos y brillantes, utilizando el color principal de texto blanco roto (#F5F5F5) para una alta legibilidad sobre el vidrio oscuro, cumpliendo con una relación de contraste superior a 4.5:1 (WCAG). Esta legibilidad es crucial para mantener la usabilidad en diversas configuraciones de pantalla de escritorio.

• Jerarquía de Información Clara: Los paneles flotantes de "vidrio esmerilado" y el enfoque multicapa (eje Z) crean una sensación de profundidad que ayuda a organizar y jerarquizar la información de manera intuitiva. Esta clara separación de elementos ayuda a que la interfaz se escale visualmente sin perder su estructura o la importancia de la información.

• Uso de CSS Variables: La implementación utilizará Propiedades Personalizadas de CSS para el sistema de color y tipografía. Esto facilita la mantenibilidad y asegura una coherencia visual integral, lo que indirectamente apoya la adaptabilidad, ya que los cambios en el tamaño o la disposición de los elementos pueden ajustarse de manera centralizada.

• Estrategias Implícitas para Layout: Aunque no se especifican directamente técnicas como flexbox o grid para el layout en las fuentes, el concepto de paneles flotantes y la necesidad de una "jerarquía espacial clara" sugieren que se emplearán principios de diseño de interfaz de usuario que permitan que los elementos se distribuyan y redimensionen de manera armoniosa en el espacio disponible, manteniendo el confort visual y la funcionalidad en diferentes tamaños de ventana de escritorio.

En resumen, la "adaptabilidad" del Dashboard se centra en mantener una experiencia de usuario consistente, cómoda y legible a través de las variaciones de tamaño de pantalla típicas de un entorno de escritorio, utilizando las propiedades inherentes del diseño Glassmorphism Oscuro y las buenas prácticas de desarrollo de UI.