Documento de Requisitos No Funcionales (RNF)

1. Definición

Los **Requisitos No Funcionales (RNF)** describen los atributos de calidad, rendimiento, seguridad y usabilidad que el sistema debe cumplir. Se basan en el diseño de la arquitectura técnica y las mejores prácticas de desarrollo.

2. Requisitos No Funcionales por Categoría

2.1. Categoría 1: Seguridad y Autorización (RNF-S)

RNF ID	Descripción Detallada	Referencia de Implementación	Categoría
RNF-S1	**Autenticación Fuerte:** El sistema debe utilizar **JWT** para la gestión de la sesión y **Bcrypt** para el *hash* de contraseñas de forma segura.	La verificación de credenciales está en backend/src/controllers/authController.js. *Middleware* authMiddleware en backend/src/middlewares/auth.middleware.js.	Seguridad
RNF-S2	**Control de Acceso (RBAC):** Se debe implementar el control de acceso basado en roles para **restringir el acceso** a rutas y funcionalidades administrativas (ej., GET /api/users).	Función checkRole aplicada a las rutas protegidas en el *backend*. La lógica *frontend* usa protectPage() en web/js/api.js.	Seguridad
RNF-S3	**Protección XSS:** El *frontend* debe **sanitizar** los datos de usuario al insertarlos en el DOM vía innerHTML para prevenir ataques XSS.	Función escapeHtml() definida en web/js/api.js. La auditoría técnica exige su uso en las tablas de datos.	Seguridad
RNF-S4	**Rate Limiting:** La API debe aplicar limitación de solicitudes (ej. **100 *requests* por 15 minutos**) para mitigar ataques de denegación de servicio (DoS).	Implementado en el *middleware* rateLimiter y aplicado en backend/src/server.js.	Seguridad

RNF ID	Descripción Detallada	Referencia de Implementación	Categoría
RNF-S5	**Transmisión Segura (HTTPS):** Todas las comunicaciones entre el *frontend* y el *backend* deben realizarse a través de **HTTPS**.	Requisito de configuración final en el servicio de *deployment* (Vercel).	Seguridad

2.2. Categoría 2: Rendimiento y Optimización (RNF-P)

RNF ID	Descripción Detallada	Referencia de Implementación	Categoría
RNF-P1	**Optimización de Assets:** Se deben eliminar las dependencias de **CDN** y utilizar el CSS compilado con *purge* para reducir el tamaño de descarga inicial.	La implementación requiere el *build* de **Tailwind CSS** con *purge* (configurado en tailwind.config.js).	Rendimiento
RNF-P2	**Tiempos de Carga (LCP):** El *Largest Contentful Paint* (LCP) de los *dashboards* debe ser óptimo, con un objetivo de **inferior a 2.5 segundos**.	*Métrica clave* para el rendimiento, justificada por la elección de un *frontend* ligero (JS Puro + Tailwind).	Rendimiento
RNF-P3	**Respuesta de API:** Las peticiones críticas (login, creación de citas, *CRUD* de usuarios) deben tener un tiempo de respuesta inferior a **500 ms**.	La elección de **Node.js** y **Express** está justificada por su alta velocidad en operaciones concurrentes, buscando este objetivo.	Rendimiento
RNF-P4	**Logging de Rendimiento:** El sistema debe registrar los tiempos de respuesta de las peticiones críticas para el monitoreo y auditoría de rendimiento.	*Requiere implementación de un sistema de logging en* backend/src/server.js.	Rendimiento

2.3. Categoría 3: Mantenibilidad y Escalabilidad (RNF-M)

RNF ID	Descripción Detallada	Referencia de Implementación	Categoría
RNF-M1	**Modularidad Frontend:** El código *frontend* debe estar organizado en **módulos ES6** (clases) con una clara separación de responsabilidades y reutilización de componentes.	Lógica de *Dashboards* implementada en archivos modulares como web/js/modules/patient-dashboard.js.	Mantenibilidad
RNF-M2	**Estructura Arquitectónica:** El *backend* debe seguir una arquitectura de **capas** clara y estandarizada (*Rutas, Controladores, Modelos*).	Estructura visible en backend/src/server.js y los subdirectorios (routes, controllers, models).	Mantenibilidad
RNF-M3	**Tecnología Consistente:** Usar JavaScript en todo el *stack* (*Node.js + Express + JS puro*) para facilitar la curva de aprendizaje y la gestión del desarrollo.	*Justificación tecnológica en* docs/Justificación tecnológica .pdf.	Mantenibilidad
RNF-M4	**Base de Datos NoSQL:** La base de datos debe ser flexible (MongoDB) para permitir la **evolución de los modelos** sin migraciones complejas.	Uso del ORM **Mongoose** con esquemas flexibles, como se define en backend/src/models/User.js.	Escalabilidad

2.4. Categoría 4: Usabilidad y Accesibilidad (RNF-A)

RNF ID	Descripción Detallada	Referencia de Implementación	Categoría
RNF-A1	**Accesibilidad (WCAG 2.1 AA):** La interfaz debe ser completamente accesible, cumpliendo los criterios **WCAG 2.1 Nivel AA**.	Módulo web/js/modules/accessibility.js gestiona *Skip Links*, *Focus Trap* en modales y *ARIA labels*.	Accesibilidad
RNF-A2	**Diseño Responsivo:** El diseño debe ser **100% responsivo** y *mobile-first*, adaptándose a cualquier dispositivo.	El uso de **Tailwind CSS** garantiza la implementación de *media queries* y *mobile-first*.	Usabilidad
RNF-A3	**Feedback al Usuario:** El sistema debe proporcionar **feedback visual inmediato** (mensajes *toast* de éxito/error) para todas las operaciones asíncronas.	Funciones showSuccessMessage / showErrorMessage disponibles globalmente en web/js/common.js y usadas en web/js/api.js.	Usabilidad
RNF-A4	**Navegación Intuitiva:** La navegación debe ser clara y permitir el acceso rápido a las funcionalidades clave según el **rol** del usuario.	Lógica en web/js/navigation-config.js que define menús específicos para 4 roles distintos.	Usabilidad