Manual Técnico del Proyecto: Sistema de Gestión de Casos MP (Sugerencia Refinada)

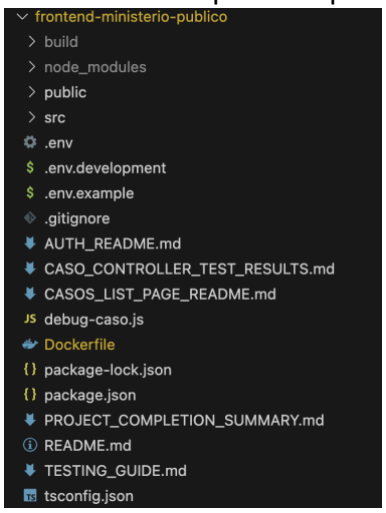
1. Introducción

- **1.1. Propósito del Proyecto:**
  El presente proyecto ha sido desarrollado como parte de la prueba técnica para aspirantes al equipo de desarrollo de software del Ministerio Público. Su objetivo principal es demostrar la capacidad de diseñar, desarrollar e implementar una aplicación de software funcional para la gestión y rastreo eficiente de casos, cumpliendo con los requisitos especificados. La aplicación facilita el registro, actualización, asignación y seguimiento de casos, además de la generación de informes básicos y la provisión de una interfaz de usuario segura e intuitiva para los fiscales.
- **1.2. Tecnologías Utilizadas:**
  - **Frontend:** React.js (con TypeScript), Axios, React Router DOM.
  - **Backend:** Node.js con Express.js, JSON Web Tokens (JWT) para autenticación.
  - **Base de Datos:** Microsoft SQL Server, con lógica de datos implementada mediante Procedimientos Almacenados.
  - **Contenerización:** Docker y Docker Compose para la orquestación de servicios.
  - **Control de Versiones:** Git.

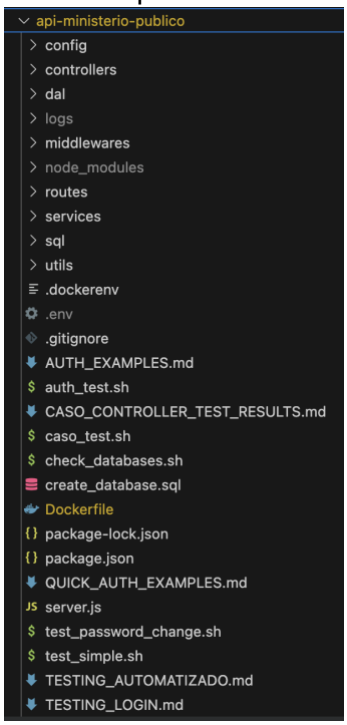
2. Arquitectura General

- **2.1. Visión General:**
  La aplicación sigue una arquitectura de N-Capas desacoplada, diseñada para promover la modularidad y escalabilidad. Los componentes principales son el Frontend (interfaz de usuario), el Backend (API RESTful y lógica de negocio) y la Base de Datos (persistencia de datos). Estos componentes interactúan como se describe en el diagrama de arquitectura.
- **2.2. Diagrama de Arquitectura:**

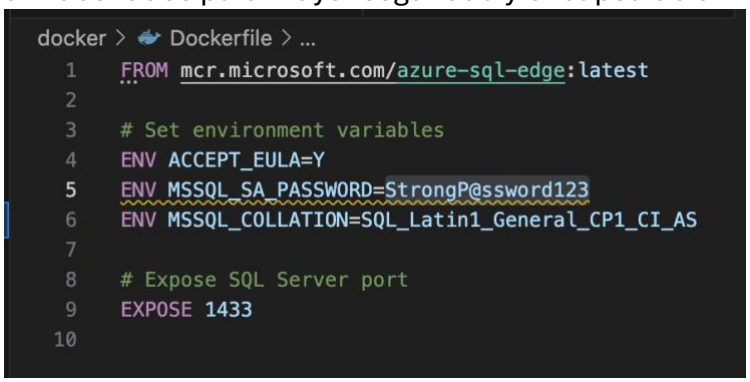
- **2.3. Componentes Principales:**
  - **Frontend (frontend-ministerio-publico):** Aplicación de una Sola Página (SPA) desarrollada en React.js, responsable de la presentación de la interfaz de usuario y la interacción con el usuario final. Consume los servicios expuestos por el Backend API.



  - **Backend (api-ministerio-publico):** API RESTful construida con Node.js y Express.js. Encapsula la lógica de negocio, gestiona la autenticación y autorización, y se comunica con la base de datos exclusivamente a través de procedimientos almacenados.



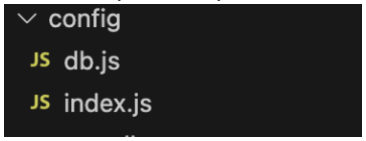
- **Base de Datos:** Sistema de Gestión de Base de Datos Microsoft SQL Server, donde se almacenan todos los datos de la aplicación. Toda la manipulación de datos (CRUD) se realiza mediante procedimientos almacenados para mayor seguridad y encapsulación.



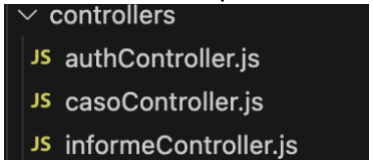
3. Backend (API - api-ministerio-publico)

- **3.1. Descripción General:**
  El backend es el núcleo de la aplicación, proporcionando una API RESTful segura para que el frontend interactúe con los datos y la lógica de negocio. Está diseñado para ser robusto y eficiente, utilizando procedimientos almacenados para todas las operaciones de base de datos.

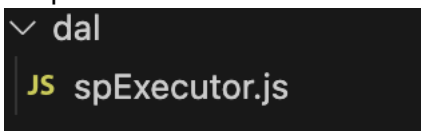
- **3.2. Estructura del Proyecto:**
  El proyecto backend sigue una estructura modular para facilitar el mantenimiento y la escalabilidad:
  - config/: Contiene la configuración de la aplicación, incluyendo la conexión a la base de datos (dbConfig.js) y la gestión de variables de entorno (dotenv).



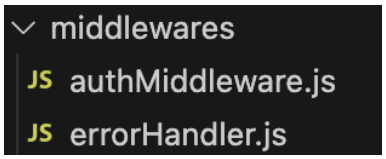
  - controllers/: Responsables de manejar las solicitudes HTTP entrantes, validar los datos de entrada (payloads, parámetros de consulta) y coordinar la respuesta, invocando a los servicios correspondientes.



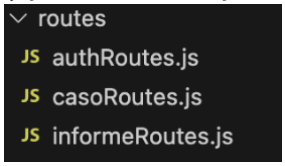
- dal/ (Data Access Layer): Módulos encargados de la interacción directa con la base de datos, ejecutando los procedimientos almacenados. Proporciona una abstracción sobre la capa de persistencia.



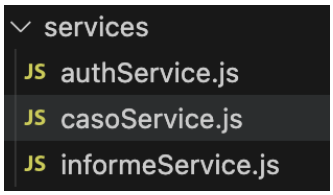
- middlewares/: Funciones que se ejecutan durante el ciclo de vida de la solicitud/respuesta. Incluye middlewares para la autenticación de tokens JWT (authMiddleware.js), validación de datos, y manejo global de errores.



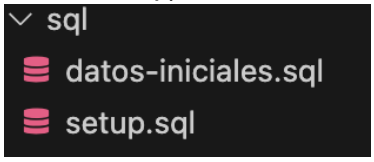
- routes/: Define los endpoints de la API y los asocia con los controladores y middlewares correspondientes (ej. authRoutes.js, casoRoutes.js).



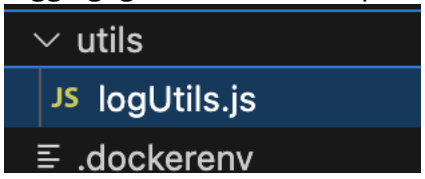
- services/: Encapsula la lógica de negocio específica de la aplicación, orquestando llamadas al DAL y aplicando reglas de negocio antes de interactuar con los controladores.



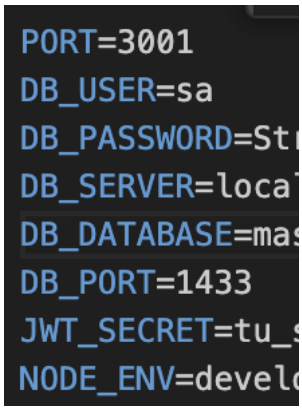
- sql/: Almacena los scripts SQL para la creación de la estructura de la base de datos (setup.sql), procedimientos almacenados (stored_procedures.sql), y la inserción de datos iniciales (datos-iniciales.sql).



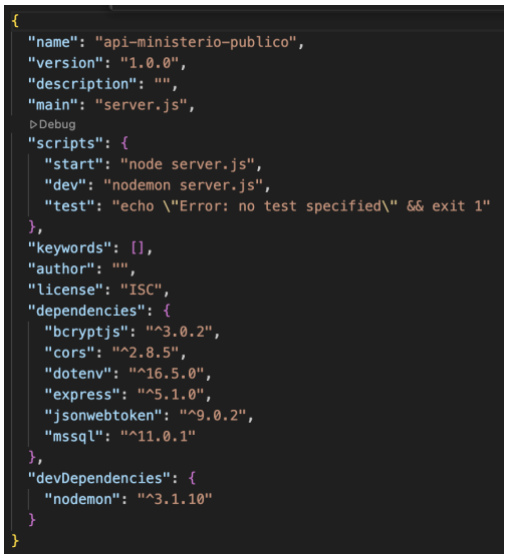
- utils/: Contiene funciones de utilidad genéricas, como helpers para logging, generación de respuestas HTTP estandarizadas, etc.



- **3.3. Endpoints de la API:**
  La API expone varios grupos de endpoints para gestionar los diferentes recursos de la aplicación:
  - **Autenticación (/api/auth):** Gestionado por routes/authRoutes.js. Incluye:
    - POST /register: Registro de nuevos usuarios.
    - POST /login: Inicio de sesión y generación de token JWT.
    - POST /logout: (Si aplica) Invalidación de token.
    - GET /verify: Verificación de la validez de un token.
    - GET /profile: Obtención del perfil del usuario autenticado.
    - POST /change-password: Cambio de contraseña del usuario autenticado.
    - POST /refresh: Refresco de token JWT.
  - **Casos (/api/casos):** Gestionado por routes/casoRoutes.js. Incluye:
    - GET /: Listado de casos (con paginación y filtros).
    - GET /mis-casos: Listado de casos asignados al fiscal autenticado.
    - POST /: Creación de un nuevo caso.
    - GET /:id: Obtención de detalles de un caso específico.
    - PUT /:id: Actualización de un caso existente.
    - PUT /:id/asignar (o /reasignar): Asignación/Reasignación de un fiscal a un caso, aplicando las reglas de negocio especificadas.
    - GET /:id/logs-reasignacion: (Si implementado) Obtención de logs de intentos fallidos de reasignación.
  - **Informes y Estadísticas (/api/informes o /api/casos/estadisticas):** Gestionado por routes/informeRoutes.js (o integrado en casoRoutes.js). Incluye:
    - GET /casos-por-estado: Estadísticas de casos agrupados por estado.
    - GET /casos-por-fiscal: Estadísticas de casos asignados por fiscal.
- **3.4. Configuración y Despliegue:**
  - **Variables de Entorno:** El backend requiere la configuración de variables de entorno definidas en un archivo .env. Estas incluyen credenciales de base de datos, secreto para JWT, puerto del servidor, etc.



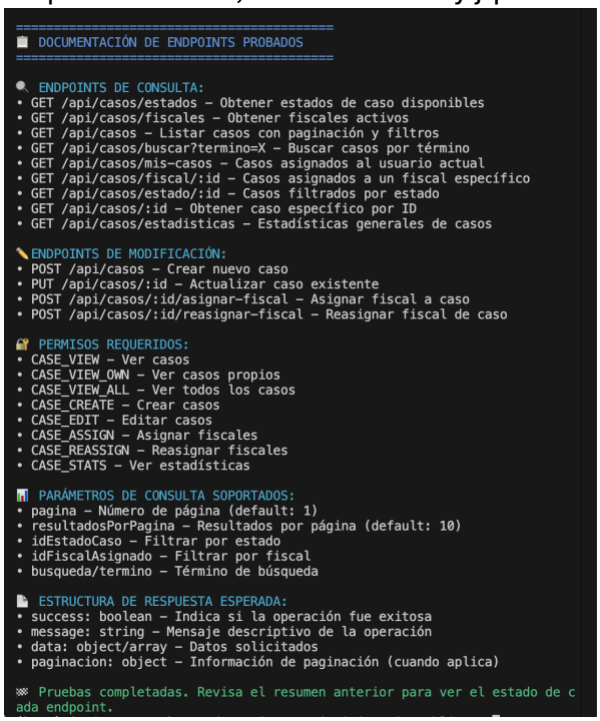
- **package.json:** Detalla las dependencias del proyecto y define scripts para iniciar (npm start, npm run dev), probar (npm test) y otras tareas de desarrollo.



- **Dockerfile:** (api-ministerio-publico/Dockerfile) Define los pasos para construir la imagen Docker del servicio backend, optimizada para producción.

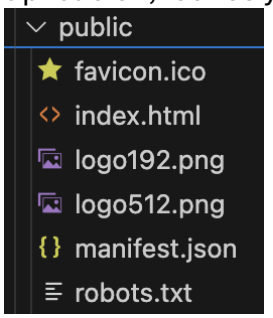
```
# Usa una imagen base de Node.js
FROM node:18-alpine

# Establece el directorio de trabajo en la aplicación
WORKDIR /usr/src/app

# Copia el package.json y package-lock.json (si existe)
COPY package*.json ./

# Instala las dependencias de la aplicación
RUN npm install

# Copia el resto de los archivos de la aplicación
COPY . .

# Copia el archivo .dockerenv como .env para las variables
COPY .dockerenv .env

# Asegura que todos los archivos tienen los permisos corre
RUN chmod -R 755 .

# Expone el puerto en el que corre la aplicación
EXPOSE 3001

# Comando para iniciar la aplicación
CMD [ "node", "server.js" ]
```

- **3.5. Pruebas del Backend:**
  Se han desarrollado scripts y documentación para facilitar las pruebas de la API:
  - auth_test.sh, caso_test.sh: Scripts Bash para ejecutar pruebas automatizadas contra los endpoints de autenticación y casos respectivamente, utilizando curl y jq.

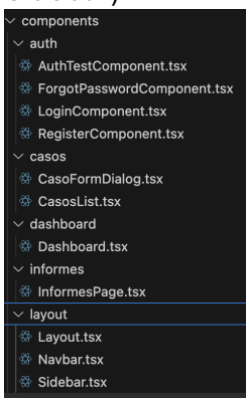


4. Frontend (frontend-ministerio-publico)

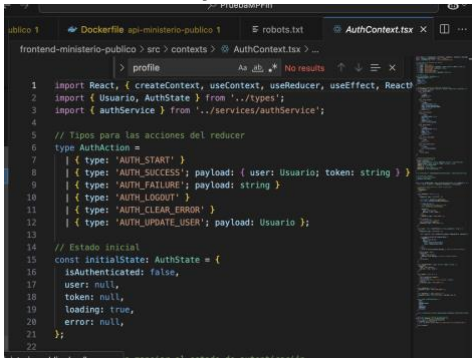
- **4.1. Descripción General:**
  El frontend es una aplicación React.js moderna y responsiva que proporciona una interfaz de usuario intuitiva para interactuar con el sistema de gestión de casos. Utiliza TypeScript para mejorar la robustez y mantenibilidad del código.
- **4.2. Estructura del Proyecto:**
  La aplicación frontend está organizada de la siguiente manera:
  - public/: Contiene el archivo index.html principal, el manifest de la aplicación, íconos y otros activos estáticos.



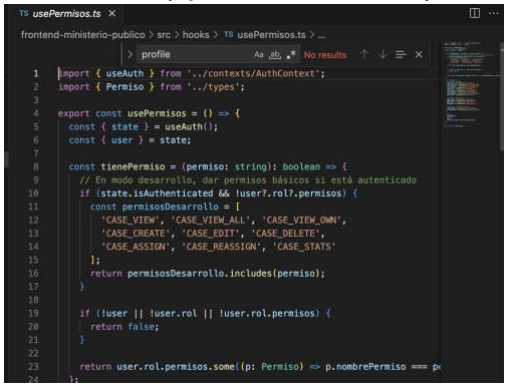
- src/: Directorio principal del código fuente de la aplicación React:
  - assets/: Recursos estáticos como imágenes, fuentes personalizadas.
  - components/: Componentes de UI reutilizables, divididos en common/ (genéricos como botones, inputs) y layout/ (Navbar, Sidebar).



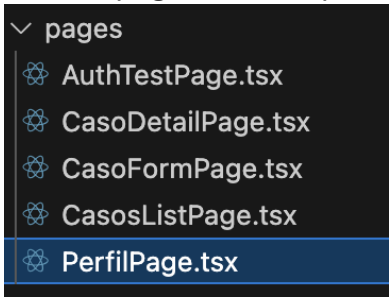
  - contexts/: Implementaciones de React Context API, como AuthContext.tsx para la gestión centralizada del estado de autenticación y la información del usuario.



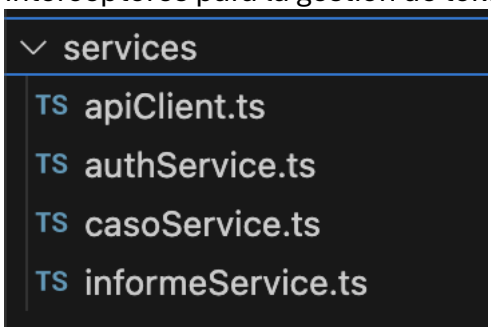
  - hooks/: Hooks personalizados para encapsular lógica reutilizable (ej. usePermisos.ts).



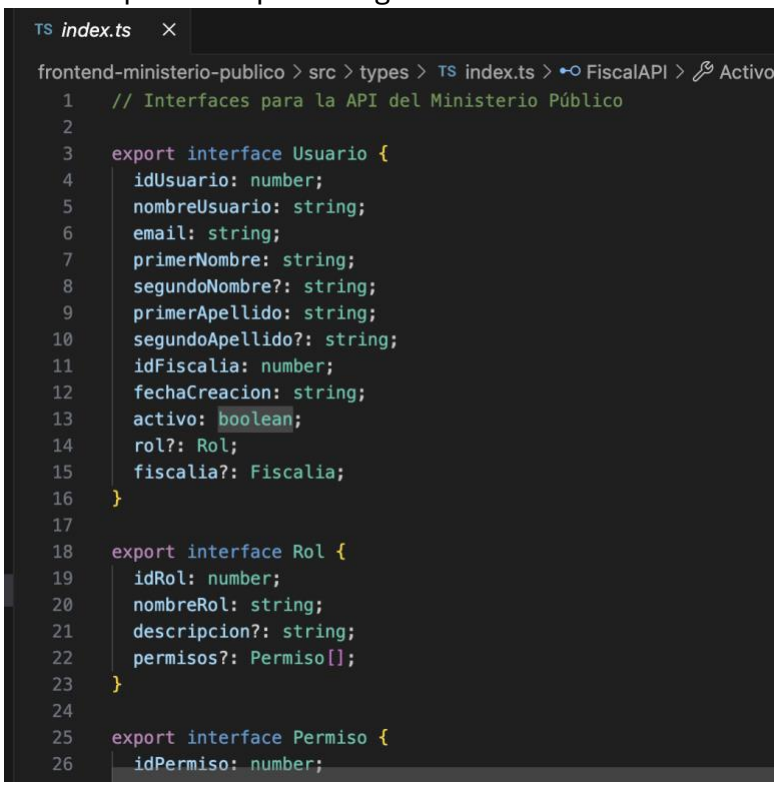
- pages/: Componentes de React que representan las diferentes vistas o páginas de la aplicación



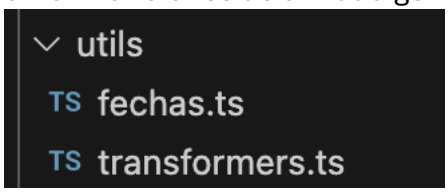
- services/: Módulos que encapsulan la lógica de comunicación con la API del backend
  utilizando axios (ej. authService.ts, casoService.ts). Incluyen interceptores para la gestión de tokens.



- types/: Definiciones de interfaces y tipos TypeScript utilizados en toda la aplicación para asegurar la consistencia de los datos.



- utils/: Funciones de utilidad generales para el frontend.



- App.tsx: Componente raíz que configura el enrutador principal y los proveedores de contexto.
- index.tsx: Punto de entrada de la aplicación que renderiza el componente App en el DOM.

- **4.3. Flujo de Autenticación:**

1. El usuario ingresa credenciales en la LoginPage.
2. Se realiza una solicitud al endpoint /api/auth/login del backend a través de authService.ts.
3. Si la autenticación es exitosa, el backend devuelve un token JWT y datos del usuario.
4. El AuthContext almacena el token (ej. en localStorage) y la información del usuario, actualizando el estado de autenticación global.
5. Las solicitudes subsiguientes a endpoints protegidos incluyen automáticamente el token JWT en el encabezado Authorization mediante un interceptor de axios.
6. El componente ProtectedRoute redirige a los usuarios no autenticados a la LoginPage.
7. La funcionalidad de "Cerrar Sesión" limpia el token almacenado y el estado de autenticación.

- **4.4. Gestión de Casos:**
  Las funcionalidades de gestión de casos se implementan a través de varias páginas y componentes que interactúan con casoService.ts:
  - **Listado de Casos:** Muestra casos con filtros y paginación.

    - **Creación/Edición de Casos:** Formularios con validación para ingresar o modificar datos de casos.
    - **Detalle del Caso:** Vista completa de la información de un caso.
    - **Reasignación de Fiscal:** Interfaz para reasignar un caso, comunicándose con el endpoint del backend que aplica las reglas de negocio (estado pendiente, misma fiscalía). El frontend muestra el feedback correspondiente (éxito o motivo del fallo si se generó un log).
- **4.5. Configuración y Despliegue:**
    - **Variables de Entorno:** La URL base de la API se configura mediante la variable
    - - REACT_APP_API_BASE_URL=http://localhost:3001/api
    - - REACT_APP_API_HOST=http://localhost:3001
    - **package.json:** Contiene las dependencias del proyecto y scripts para desarrollo (npm start o npm run dev), construcción (npm run build), y pruebas (npm test).
    - **Dockerfile:** (frontend-ministerio-publico/Dockerfile) Describe el proceso de construcción de la imagen Docker para el frontend, utilizando un multi-stage build con Nginx para servir los archivos estáticos optimizados.

5. Base de Datos

- **5.1. Modelo Entidad-Relación y Esquema:**
  El diseño de la base de datos se detalla en el Diagrama Entidad-Relación y su explicación correspondiente. Estos documentos ilustran las tablas, sus atributos, relaciones y claves.
    - Diagrama Visual: 4) DiagramaEntidadRelacion/diagramaEntidadRelacion.png
    - Explicación Detallada: 4) DiagramaEntidadRelacion/Explicacion Diagrama Entidad Relación.pdf
    - Proceso de Normalización: normalizacionBD.xlsx
- **5.2. Scripts de Inicialización y Configuración:**
  La configuración inicial de la base de datos se automatiza mediante los siguientes scripts:
    - api-ministerio-publico/sql/ setup.sql: Contiene la definición de todas las tablas y todos los procedimientos almacenados utilizados por la aplicación.
    - api-ministerio-publico/sql/datos-iniciales.sql: Script para poblar las tablas con datos iniciales necesarios para el funcionamiento y prueba de la aplicación.

6. Dockerización

- **6.1. Orquestación con docker-compose.yml:**
  El archivo docker-compose.yml en la raíz del proyecto define y orquesta los tres servicios principales de la aplicación:
  - **backend:** Construido a partir del Dockerfile en api-ministerio-publico/.
  - **frontend:** Construido a partir del Dockerfile en frontend-ministerio-publico/.
  - **database:** Utiliza una imagen oficial de Microsoft SQL Server, configurada para ejecutar los scripts de inicialización ubicados en docker/init-scripts/.
    El docker-compose.yml también gestiona las redes, volúmenes para persistencia de datos, y la configuración de variables de entorno para cada servicio.
- **6.2. Dockerfiles:**
  - **Backend (api-ministerio-publico/Dockerfile):** Define el entorno Node.js, instala dependencias, copia el código fuente y expone el puerto necesario para la API.
  - **Frontend (frontend-ministerio-publico/Dockerfile):** Utiliza un proceso de construcción multi-etapa. Primero, construye la aplicación React para generar archivos estáticos, y luego los sirve utilizando un servidor web Nginx optimizado.
  - **docker/Dockerfile:** Es una imagen base personalizada para SQL Server la cual crea una base de datos para que el api pueda conectarse.