Logotipo

Descripción generada automáticamente con confianza baja

Logotipo

Descripción generada automáticamente



Tabla de Contenido

[**1 Introducción 3**](#_heading=h.30j0zll)

[**2 Arquitectura del sistema 3**](#_heading=h.3znysh7)

[**3 Descripción de los módulos del frontend de Medical-AID 3**](#_heading=h.2et92p0)

[**4 Backend 9**](#_heading=h.tyjcwt)

[**5 Modelo de clasificación de síntomas: 17**](#_heading=h.3dy6vkm)

[**6 Conclusión 19**](#_heading=h.1t3h5sf)

1. **Introducción**

Este manual técnico tiene como objetivo entregar una visión detallada del sistema **Medical AID** a desarrolladores e integradores que deseen instalar, configurar y entender los módulos principales del sistema.

El sistema **Medical AID** es una herramienta diseñada para asistir en diagnósticos médicos utilizando tecnología de machine learning, una interfaz de usuario dinámica y una base de datos robusta.

**Medical-AID** está compuesto por tres pilares principales:

* **Frontend**: Basado en React, ofrece una interfaz intuitiva para que los usuarios interactúen con el sistema.
* **Backend**: Construido con Node.js y Express, gestiona las operaciones lógicas y la conexión con la base de datos.
* **Modelo de Machine Learning**: Implementado con Python y Scikit-learn, proporciona predicciones para los diagnósticos médicos.

1. **Arquitectura del sistema**

* **Frontend**: React + JavaScript, integrado con Axios para consumir APIs.
* **Backend**: Node.js, con endpoints REST creados con Express.
* **Base de Datos**: PostgreSQL, gestionada con PGAdmin para almacenar datos médicos y de usuarios.
* **Modelo ML**: Un servicio Python que utiliza Scikit-learn para cálculos de predicción.

1. **Descripción de los módulos del Frontend de Medical-AID**
2. **index.js**

**Descripción:** Archivo principal que sirve como punto de entrada de la aplicación React. Configura e inicializa la aplicación, estableciendo el entorno base para su ejecución.

**Funcionalidades Clave:**

* Renderiza el componente raíz App dentro del elemento DOM identificado como root.
* Integra bibliotecas esenciales para el correcto funcionamiento de la aplicación.
* Configura el entorno base, preparando la aplicación para interactuar con el frontend y otros servicios relacionados.

1. **App.js**

**Descripción:** Componente principal que actúa como el componente raíz de la aplicación. Proporciona la estructura general de la aplicación, configurando las rutas y manejando la navegación entre las diferentes secciones.

**Funcionalidades Clave:**

* Define y organiza las rutas de la aplicación utilizando bibliotecas de enrutamiento como react-router-dom.
* **Gestiona la navegación entre las principales secciones, tales como:**
  + - Panel de control: Vista principal con resúmenes y métricas del sistema.
    - Inicio de sesión: Página para autenticación de usuarios.
    - Biblioteca de diagnósticos: Repositorio de información sobre enfermedades, síntomas y precauciones.
* Implementa la estructura base de la aplicación, asegurando que todos los componentes secundarios se rendericen correctamente dentro del marco principal.

1. **LoginForm.jsx**

**Descripción:** Componente encargado de gestionar la autenticación de usuarios mediante un formulario de inicio de sesión. Se encarga de capturar las credenciales de los usuarios, validarlas y manejar la comunicación con el backend para realizar la autenticación.

**Funcionalidades Clave:**

* **Formulario de inicio de sesión:**
  + Captura de credenciales: Email y contraseña.
  + Validación básica de campos antes de enviar los datos al backend.
* **Comunicación con la API:**
  + Envía las credenciales de usuario al endpoint de autenticación en el backend.
  + Recibe y procesa la respuesta del backend para determinar si la autenticación fue exitosa o fallida.
* **Gestión de errores:**
  + Maneja errores de autenticación, como credenciales incorrectas o problemas de conexión, mostrando mensajes de error claros al usuario.
* **Estilos personalizados:**
  + Utiliza el archivo LoginForm.css para personalizar la apariencia del formulario, garantizando un diseño visualmente atractivo y responsivo en múltiples dispositivos.

1. **AdminPage.jsx**

**Descripción**: Componente diseñado para manejar tareas administrativas dentro del sistema. Ofrece una interfaz centralizada para la gestión de usuarios, datos y configuraciones específicas del sistema. Su uso está restringido a usuarios con roles de administrador.

**Funcionalidades Clave:**

* **Gestión de usuarios:**
  + Creación, edición y eliminación de perfiles de usuario.
  + Asignación y modificación de roles y permisos.
* **Administración de datos:**
  + Acceso a registros críticos del sistema para su revisión o modificación.
  + Herramientas para importar o exportar datos relevantes.
* **Funciones restringidas:**
  + Asegura que solo los administradores autorizados puedan acceder a las herramientas y vistas específicas.
  + Integra verificaciones de autenticación y roles para garantizar la seguridad.
* **Estilos personalizados:**
  + Utiliza el archivo AdminPage.css para proporcionar un diseño profesional y responsivo que facilite la usabilidad en dispositivos de diferentes tamaños.

1. **DashboardView.jsx**

**Descripción:** Componente principal que funciona como el panel de control del sistema. Ofrece una visión general del estado actual del sistema, presentando métricas clave y resúmenes de datos.

**Funcionalidades Clave:**

* **Visualización de métricas:**
  + Muestra gráficos dinámicos que resumen el uso del sistema.
  + Proporciona indicadores clave (KPIs) relevantes para los administradores y usuarios.
* **Resumen de datos:**
  + Presenta información consolidada de diagnósticos.
  + Ofrece filtros para personalizar la visualización de datos según necesidades específicas.
* **Interfaz amigable y responsiva:**
  + Utiliza el archivo DashboardView.css para garantizar un diseño atractivo, organizado y adaptado a diferentes dispositivos.

1. **MapView.jsx**

**Descripción**: Componente que utiliza mapas interactivos para mostrar datos geográficos relacionados con diagnósticos médicos o eventos relevantes. Permite a los usuarios visualizar ubicaciones clave de manera dinámica y visualmente atractiva.

**Funcionalidades Clave:**

* **Visualización de datos geográficos:**
  + Muestra puntos de interés, como áreas con mayor incidencia de diagnósticos o eventos médicos importantes.
  + Representación visual clara mediante marcadores, colores y etiquetas informativas.
* **Interactividad del mapa:**
  + Permite a los usuarios acercar, alejar y explorar diferentes áreas geográficas.
  + Proporciona detalles adicionales al interactuar con elementos del mapa (por ejemplo, haciendo clic en un marcador).
* **Integración de bibliotecas de mapas:**
  + Utiliza bibliotecas como Leaflet, Mapbox o Google Maps para renderizar y gestionar datos dinámicos.
* **Datos dinámicos:**
  + Obtiene datos geográficos como parámetros desde la vista que utilice este componente.
* **Estilos personalizados:**
  + Utiliza el archivo MapView.css para mantener la coherencia del diseño con el resto de la aplicación, asegurando que el mapa sea visualmente atractivo y fácil de usar.

1. **DiagnosisResultPage.jsx**

**Descripción:** Componente diseñado para presentar los resultados de un diagnóstico médico realizado en la plataforma. Muestra información relevante como la enfermedad posible, recomendaciones personalizadas y detalles adicionales para el usuario.

**Funcionalidades Clave:**

* **Visualización de resultados:**
  + Presenta la enfermedad o condición más probable basada en los síntomas ingresados por el usuario.
  + Muestra recomendaciones personalizadas, como sugerencias de atención médica o medidas preventivas.
* **Interacción con el backend:**
  + Realiza solicitudes al backend para obtener datos de diagnóstico y otras informaciones relacionadas.
  + Maneja respuestas y errores para garantizar que los datos se presenten correctamente al usuario.
* **Diseño intuitivo:**
  + Utiliza el archivo DiagnosisResultPage.css para garantizar que los resultados sean visualmente claros y organizados.
* **Compatibilidad responsiva:**
  + Diseñado para adaptarse a diferentes dispositivos y tamaños de pantalla, asegurando una experiencia de usuario consistente.

1. **DiagnosticLibraryPage.jsx**

**Descripción:** Componente que actúa como un repositorio interactivo de enfermedades y sus detalles. Permite a los usuarios buscar y explorar información sobre diversas condiciones médicas, incluyendo descripciones, síntomas asociados y precauciones recomendadas.

**Funcionalidades Clave:**

* **Visualización de detalles:**
  + Muestra descripciones completas de cada enfermedad, incluyendo sus características principales y factores de riesgo.
  + Lista de síntomas asociados y precauciones recomendadas para cada condición.
* **Estilo atractivo y accesible:**
  + Utiliza el archivo DiagnosticLibraryPage.css para aplicar animaciones suaves y un diseño que facilite la navegación.
  + Asegura que la biblioteca sea visualmente atractiva y compatible con dispositivos móviles.
* **Actualización dinámica:**
  + Obtiene datos de enfermedades desde el backend, asegurando que la información esté actualizada y refleje los últimos cambios o descubrimientos médicos.

1. **SymptomsForm.jsx**

**Descripción:** Componente que permite a los usuarios ingresar sus síntomas para obtener un diagnóstico. Actúa como la interfaz principal de entrada de datos, interactuando con el backend para procesar la información y proporcionar resultados médicos basados en los síntomas proporcionados.

**Funcionalidades Clave:**

* **Captura de datos:**
  + - Proporciona un formulario interactivo donde los usuarios pueden seleccionar o escribir los síntomas que están experimentando.
* **Interacción con el backend:**
  + Envía los datos ingresados al backend para su procesamiento.
* **Validación del formulario:**
  + Validar los campos obligatorios antes de enviar la solicitud al backend.
  + Muestra mensajes de error si no se ingresan datos o si ocurre un problema con la conexión.
* **Diseño intuitivo:**
  + Utiliza el archivo SymptomsForm.css para garantizar que el formulario sea limpio, organizado y fácil de usar.
  + Incluye ayudas visuales, como íconos o texto explicativo, para guiar al usuario durante la entrada de datos.
* **Compatibilidad responsiva:**
  + Adaptado para funcionar correctamente en dispositivos móviles y de escritorio, asegurando una experiencia fluida en diferentes tamaños de pantalla.

Cada módulo del frontend contribuye a las funcionalidades generales de la aplicación **Medical-AID**, abarcando desde la gestión de usuarios hasta la visualización de resultados de diagnóstico y datos geográficos. Los archivos CSS asociados juegan un papel clave al mantener una interfaz consistente y fácil de usar en toda la aplicación.

1. **Backend**

A continuación, se lista los módulos y funcionalidad principales del Backend de nuestro proyecto:

**1. index.js**

* **Descripción**: Archivo principal del backend que inicializa el servidor, configura las rutas y define los middlewares. Se encarga de levantar el servicio en el puerto especificado.
* **Funcionalidad Clave**: Define el servidor HTTPS, configura CORS y conecta controladores para manejar las operaciones de cada entidad.

**2. db.js**

* **Descripción**: Configuración básica de la conexión con la base de datos PostgreSQL.
* **Funcionalidad Clave**: Utiliza Sequelize para gestionar la conexión y manipulación de la base de datos.

**3. database.js**

* **Descripción**: Contiene configuraciones avanzadas para la inicialización de Sequelize y la conexión a la base de datos.
* **Funcionalidad Clave**: Configura la autenticación, inicializa modelos y sincroniza la estructura de la base de datos.

**4. config.js**

* **Descripción**: Archivo de configuración que contiene las variables de entorno, como la URL de la base de datos y claves secretas.
* **Funcionalidad Clave**: Proporciona un punto central para gestionar configuraciones globales.

**5. DiagnosesController.js**

* **Descripción**: Controlador para manejar las operaciones relacionadas con los diagnósticos médicos.
* **Funcionalidad Clave**: CRUD para diagnósticos, incluyendo la creación, actualización y eliminación de registros.

**6. DiseaseController.js**

* **Descripción**: Controlador que gestiona las operaciones relacionadas con las enfermedades.
* **Funcionalidad Clave**: Proporciona endpoints para listar, crear y modificar información de enfermedades.

**7. PredictionController.js**

* **Descripción**: Controlador que realiza predicciones basadas en un modelo de machine learning.
* **Funcionalidad Clave**: Integra predicciones usando Scikit-learn y envía resultados al frontend.

**8. subscribeNewsLetterController.js**

* **Descripción**: Controlador encargado de gestionar las suscripciones a boletines informativos.
* **Funcionalidad Clave**: Permite suscribirse, cancelar suscripciones y listar usuarios suscritos.

**9. authController.js**

* **Descripción**: Controlador para gestionar la autenticación de usuarios.
* **Funcionalidad Clave**: Registro, inicio de sesión y validación de tokens JWT.

**10. DashboardController.js**

* **Descripción**: Controlador que gestiona la administración del panel y reportes estadísticos.
* **Funcionalidad Clave**: Genera datos de analítica para el uso del sistema.

**11. SymptomsController.js**

* **Descripción**: Controlador para la gestión de síntomas.
* **Funcionalidad Clave**: CRUD para síntomas y relación con enfermedades o diagnósticos.

**12. userController.js**

* **Descripción**: Controlador para la gestión de usuarios.
* **Funcionalidad Clave**: Creación, actualización, eliminación y recuperación de usuarios en el sistema.

**13. authenticateUser.js**

* **Descripción**: Middleware para autenticar usuarios antes de acceder a ciertos endpoints.
* **Funcionalidad Clave**: Valida credenciales y roles asignados.

**14. authenticateToken.js**

* **Descripción**: Middleware para verificar la validez de tokens JWT en solicitudes.
* **Funcionalidad Clave**: Garantiza la autenticación segura antes de acceder a recursos protegidos.

**15. DiagnosisSymptoms.js**

* **Descripción**: Modelo que define la relación entre diagnósticos y síntomas.
* **Funcionalidad Clave**: Gestiona relaciones de muchos a muchos para estos registros.

**16. AlertGeoLocation.js**

* **Descripción**: Modelo que define alertas basadas en ubicaciones geográficas.
* **Funcionalidad Clave**: Configuración y activación de alertas en áreas específicas.

**17. Alerts.js**

* **Descripción**: Modelo que gestiona alertas en el sistema.
* **Funcionalidad Clave**: Registro y seguimiento de alertas activadas.

**18. AuditLogs.js**

* **Descripción**: Modelo para el registro de auditoría de operaciones en el sistema.
* **Funcionalidad Clave**: Guarda información de acciones realizadas por usuarios.

**19. Diagnoses.js**

* **Descripción**: Modelo que define la estructura y datos de los diagnósticos.
* **Funcionalidad Clave**: Almacena información clave sobre diagnósticos médicos realizados.

**20. DiagnosisDisease.js**

* **Descripción**: Modelo que define la relación entre diagnósticos y enfermedades.
* **Funcionalidad Clave**: Vincula diagnósticos con posibles enfermedades relacionadas.

**21. Users.js**

* **Descripción**: Modelo principal para la gestión de usuarios.
* **Funcionalidad Clave**: Almacena datos de credenciales, perfiles y roles de usuarios.

**22. UserRoles.js**

* **Descripción**: Modelo que define la relación entre usuarios y roles asignados.
* **Funcionalidad Clave**: Facilita la autorización basada en roles.

**23. UserProfiles.js**

* **Descripción**: Modelo para la gestión de perfiles de usuarios.
* **Funcionalidad Clave**: Define datos adicionales de usuarios, como información personal.

**24. NewsletterSubscribers.js**

* **Descripción**: Modelo que almacena información de suscriptores de boletines informativos.
* **Funcionalidad Clave**: Permite el envío de actualizaciones automáticas.

**25. Roles.js**

* **Descripción**: Modelo que define roles de usuario en el sistema.
* **Funcionalidad Clave**: Establece jerarquías y permisos.

**26. Symptoms.js**

* **Descripción**: Modelo que define la estructura de los síntomas.
* **Funcionalidad Clave**: Relación con enfermedades y diagnósticos.

**27. Alerts.js**

* **Descripción**: Modelo para gestionar las alertas activadas en el sistema.
* **Funcionalidad Clave**: Configuración, activación y seguimiento.

**28. SymptomsRoutes.js**

* **Descripción**: Define las rutas asociadas a síntomas.
* **Funcionalidad Clave**: Conecta las peticiones de usuarios con el controlador de síntomas.

**29. UserRoutes.js**

* **Descripción**: Rutas relacionadas con la gestión de usuarios.
* **Funcionalidad Clave**: Endpoints para operaciones CRUD y autenticación.

**30. DiseaseRoutes.js**

* **Descripción**: Rutas que conectan con el controlador de enfermedades.
* **Funcionalidad Clave**: Proporciona acceso a datos de enfermedades.

**31. PredictionRoutes.js**

* **Descripción**: Rutas para realizar predicciones.
* **Funcionalidad Clave**: Endpoints que integran predicciones con machine learning.

**32. DashboardRoutes.js**

* **Descripción**: Rutas para manejar los reportes del dashboard.
* **Funcionalidad Clave**: Generación de datos analíticos.

**33. DiagnosesRoutes.js**

* **Descripción**: Define las rutas para diagnósticos.
* **Funcionalidad Clave**: Permite acceso y modificación de diagnósticos.

**34. AuthRoutes.js**

* **Descripción**: Rutas para autenticación y registro.
* **Funcionalidad Clave**: Gestiona operaciones de inicio de sesión.

**35. subscribeNewsLetterRoutes.js**

* **Descripción**: Rutas asociadas a suscriptores de boletines.
* **Funcionalidad Clave**: Endpoints para suscribirse o cancelar suscripciones.

**36. init-models.js**

* **Descripción**: Archivo de inicialización para todos los modelos Sequelize.
* **Funcionalidad Clave**: Vincula y sincroniza los modelos.

**37. AlertGeoLocation.js**

* **Descripción**: Modelo para alertas geográficas.
* **Funcionalidad Clave**: Define alertas por ubicación.

**38. UserAlerts.js**

* **Descripción**: Modelo que relaciona usuarios con alertas.
* **Funcionalidad Clave**: Establece notificaciones personalizadas.

**39. Disease.js**

* **Descripción**: Modelo para almacenar datos de enfermedades.
* **Funcionalidad Clave**: Relación con diagnósticos y síntomas.

**40. auth.js**

* **Descripción**: Middleware de autenticación centralizada.
* **Funcionalidad Clave**: Verifica y autoriza usuarios en todos los módulos.

1. **Modelo de clasificación de síntomas:**

**Descripción**: Este archivo contiene la implementación principal del clasificador de síntomas. Utiliza un modelo entrenado, best\_rf\_model.pkl, para realizar predicciones basadas en entradas proporcionadas por el usuario. Proporciona una función que carga el modelo, transforma los datos de entrada y devuelve una predicción.

**Función Principal**:

* **load\_model()**: Carga el modelo pre entrenado desde el archivo best\_rf\_model.pkl.
* **predict(symptoms)**: Realiza predicciones basadas en los síntomas proporcionados como entrada.

**Archivo: symptom\_classifier\_proba.py**

* **Descripción**: Similar al archivo anterior, este script extiende la funcionalidad del clasificador para incluir probabilidades asociadas con cada predicción. Utiliza el modelo para calcular la probabilidad de cada categoría y devuelve estos valores junto con la predicción.
* **Función Principal**:
  + **predict\_proba(symptoms)**: Devuelve tanto la clase predicha como las probabilidades asociadas con cada clase para los síntomas ingresados.

**Archivo: test\_endpoint.py**

* **Descripción**: Este script se utiliza para probar el endpoint /predict del backend. Envía datos de prueba al endpoint y verifica la respuesta.
* **Detalles de prueba**:
  + **URL del endpoint**: http://localhost:5000/predict
  + **Datos de entrada**: Una lista de 131 valores booleanos que representan la presencia o ausencia de síntomas.
  + **Resultado esperado**: La respuesta del servidor debe contener una predicción basada en los datos ingresados.

**Archivo: test\_endpoint\_proba.py**

* **Descripción**: Este script realiza pruebas similares al anterior, pero está diseñado para el endpoint /predict\_proba. Este endpoint proporciona las probabilidades asociadas con cada clase predicha.
* **Detalles de prueba**:
  + **URL del endpoint**: http://localhost:5000/predict\_proba
  + **Datos de entrada**: Una lista de 131 valores booleanos.
  + **Resultado esperado**: La respuesta debe incluir tanto la clase predicha como las probabilidades asociadas.

**Archivo: best\_rf\_model.pkl**

* **Descripción**: Este archivo contiene el modelo de bosque aleatorio pre entrenado (Random Forest) que se utiliza para realizar predicciones. Es un archivo binario que almacena el modelo entrenado en formato serializado (pickle).
* **Propósito**: Proporcionar la lógica detrás de las predicciones de síntomas en los endpoints /predict y /predict\_proba.

1. **Conclusión**

Este documento ofrece una guía detallada para comprender la estructura modular del sistema y su implementación técnica, lo que permitirá a otros desarrolladores ampliar y mantener la solución de manera efectiva.