**Documentación del Proyecto: Identificador de Patologías Pulmonares**

**1. Resumen General del Proyecto**

Esta es una aplicación web construida con Next.js y React que permite a los usuarios subir imágenes de radiografías de pulmón para ser analizadas por un sistema de Inteligencia Artificial. La aplicación identifica posibles patologías (Normal, Opacidades, Neumonía, Covid) y muestra un diagnóstico junto con un porcentaje de confianza.

La interfaz de usuario está desarrollada con ShadCN UI y Tailwind CSS, lo que le da un aspecto moderno y profesional. La lógica de análisis de imágenes se gestiona a través de un backend externo (API), al que se accede de forma segura mediante Server Actions de Next.js.

**2. Estructura de Archivos y Tecnologías**

El proyecto sigue la estructura recomendada por Next.js 14, utilizando el App Router.

-   /src/app/: Contiene las páginas y layouts principales de la aplicación.

    -   `layout.tsx`: El diseño raíz que envuelve toda la aplicación.

    -   `page.tsx`: La página principal o "Analizador", donde el usuario sube la imagen.

    -   `history/page.tsx`: La página que muestra el historial de análisis.

    -   `actions.ts`: Define las "Server Actions", funciones que se ejecutan de forma segura en el servidor para interactuar con la API externa

<https://ensemble-api-qzpf.onrender.com>

esta API fue desarrollada usando modelos de Deep learning revisados en la materia de inteligencia artificial.

.

-   /src/components/: Almacena los componentes reutilizables de la interfaz.

    -   `header.tsx`: El encabezado de la aplicación.

    -   `/ui/`: Componentes base de ShadCN UI (botones, tarjetas, etc.).

-   /src/ai/: Contiene la lógica relacionada con la Inteligencia Artificial, utilizando Genkit.

    -   `flows/analyze-lung-xray.ts`: Define el flujo de Genkit que interactúa con el modelo de IA.

-   /public/: Para archivos estáticos (aunque en este proyecto no se usa para imágenes de usuario).

-   `package.json`: Lista las dependencias del proyecto (Next.js, React, ShadCN, Genkit, etc.).

-   `tailwind.config.ts`: Configuración de Tailwind CSS para los estilos.

**3. Manual Funcional de la Aplicación**

Esta sección describe cómo un usuario interactúa con la aplicación paso a paso.

**3.1. Pantalla Principal (Analizador)**

Al abrir la aplicación, el usuario ve la pantalla principal, diseñada para ser simple e intuitiva.

Funcionalidades:

-   Un título claro que describe el propósito de la herramienta.

-   Un área designada para cargar la imagen de la radiografía.



<https://uees-lung-x-ray.vercel.app/>

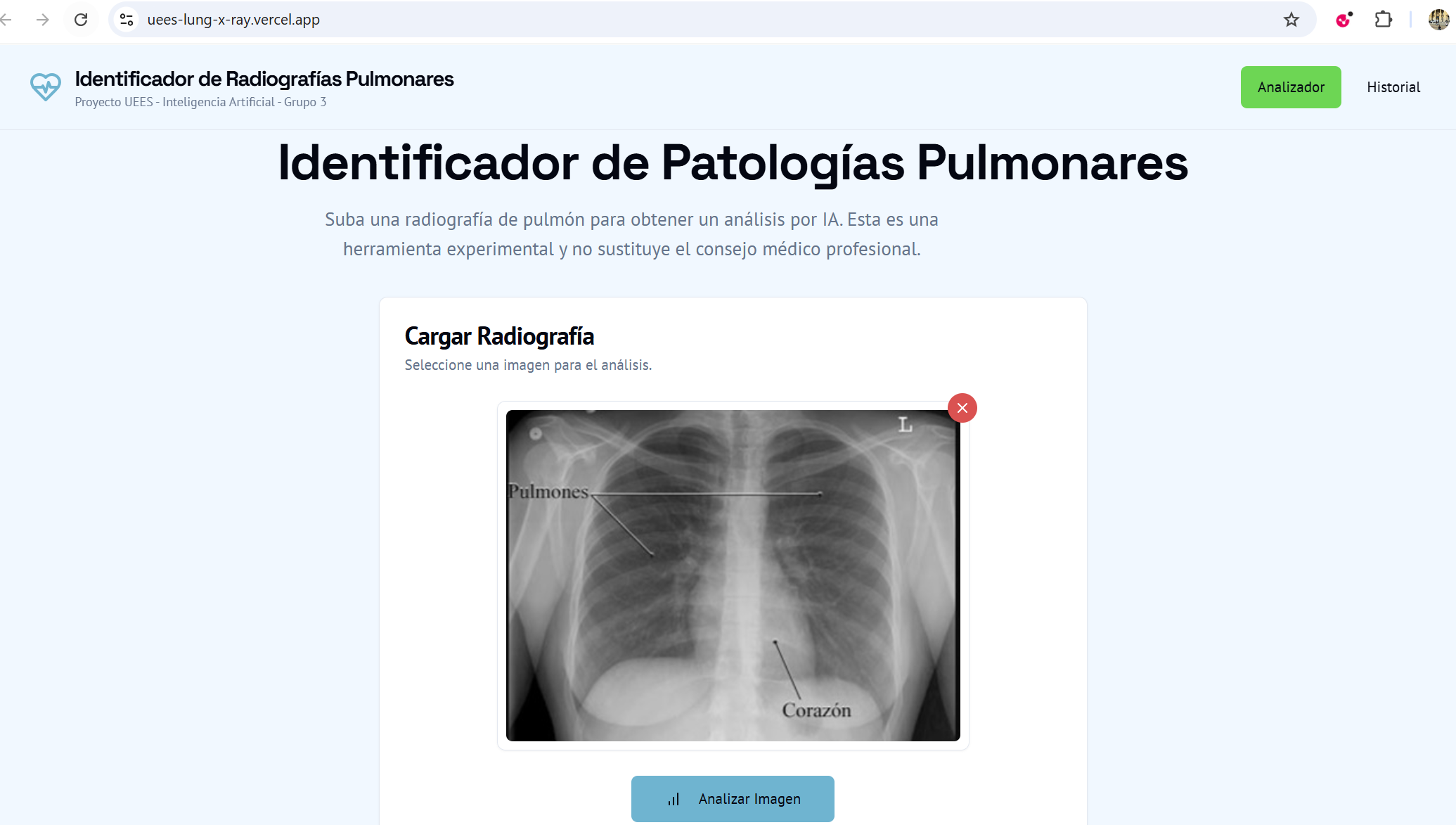
**Pasos de uso:**

1.  El usuario hace clic en el área que dice "Click para cargar" o arrastra un archivo de imagen directamente sobre esa zona.

2.  Se abre el selector de archivos del dispositivo. El usuario selecciona una imagen de radiografía (formatos comunes como PNG, JPG son aceptados).

**3.2. Vista Previa y Análisis**

Una vez que se selecciona una imagen, la interfaz cambia para mostrar una vista previa de la radiografía cargada.



**Pasos de uso:**

1.  El usuario puede verificar que ha subido la imagen correcta. Si se equivocó, puede hacer clic en el botón con la "X" para quitar la imagen y cargar otra.

2.  Si la imagen es correcta, el usuario hace clic en el botón "Analizar Imagen".

3.  La aplicación muestra un indicador de carga ("Analizando...") mientras envía la imagen al servidor y espera la respuesta de la IA.

**3.3. Visualización de Resultados**

Después de unos segundos, la aplicación muestra los resultados del análisis en una serie de tarjetas claras y fáciles de entender.



**Componentes del resultado:**

1.  Estado de los Pulmones: La tarjeta principal muestra el diagnóstico final (ej. "Opacidad Pulmonar", "Normal") y el nivel de confianza general que la IA tiene en ese diagnóstico. El color de la barra (rojo para patologías, verde para normal) ofrece una señal visual inmediata.

2.  Desglose por Modelo (Ensemble): Esta tarjeta muestra la predicción y confianza de cada modelo de IA individual que contribuyó al resultado final. Esto da una visión más detallada de cómo se llegó al diagnóstico.

3.  Botón para Nuevo Análisis: Al final, el botón "Analizar Otra Imagen" permite al usuario reiniciar el proceso para analizar una nueva radiografía.

**3.4. Historial de Análisis**

Cada análisis realizado se guarda automáticamente en el historial del navegador.

**Pasos de uso:**

1.  El usuario hace clic en el enlace "Historial" en la barra de navegación superior.

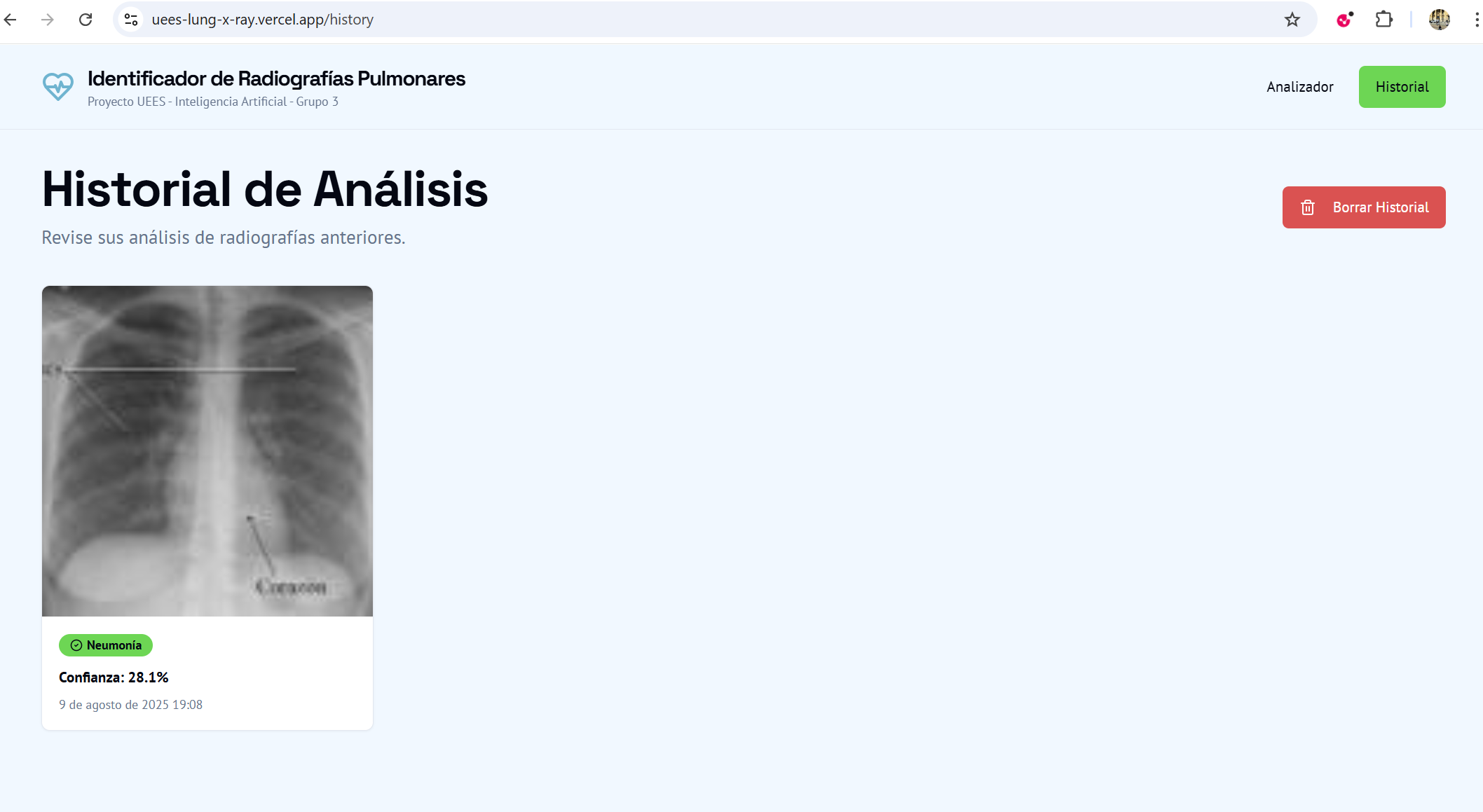
2.  Se muestra una galería con tarjetas de todos los análisis anteriores. Cada tarjeta incluye:

    - Una miniatura de la radiografía.

    - El diagnóstico obtenido.

    - La confianza del diagnóstico.

    - La fecha y hora del análisis.



**Funcionalidades Adicionales:**

-   Borrar Historial: Un botón en la esquina superior derecha permite eliminar todos los registros del historial con una confirmación para evitar borrados accidentales.

-   Navegación: Si el historial está vacío, se muestra un mensaje y un botón para volver a la página de análisis.

**4. Explicación Detallada del Código**

**Expuesto en GitHub :**

[**https://github.com/ElizaMarti/UEES-LungX-Ray/tree/main/src**](https://github.com/ElizaMarti/UEES-LungX-Ray/tree/main/src)

**4.1. Página Principal (Analizador) - `src/app/page.tsx`**

Este es el corazón de la interacción con el usuario. Es un Componente de Cliente (`"use client"`) porque necesita manejar estados y eventos del navegador.

-   Estado (`useState`):

    -   `preview`: Almacena la URL de la imagen para mostrar una vista previa.

    -   `imageBase64`: Guarda la imagen codificada en Base64. Esta es la clave para que la subida funcione en móviles (especialmente iPhone), ya que enviar texto (Base64) es más fiable que enviar un objeto `File`.

    -   `result`: Contiene la respuesta del análisis de la IA.

    -   `isLoading`: Controla si se muestra el indicador de "Analizando...".

-   Función `handleFileChange`:

    1.  Se activa cuando el usuario selecciona un archivo.

    2.  Utiliza `FileReader` para leer el archivo de imagen.

    3.  Cuando termina, convierte la imagen a una cadena Base64 y la guarda en los estados `preview` y `imageBase64`.

-   Función `handleAnalyze`:

    1.  Se ejecuta al hacer clic en "Analizar Imagen".

    2.  Establece `isLoading` a `true`.

    3.  Llama a la Server Action `analyzeImageAction`, pasándole la cadena Base64 de la imagen.

    4.  Manejo de Errores: Comprueba si la respuesta de la acción contiene un campo `error`. Si es así, muestra un `toast` (notificación) amigable al usuario en lugar de bloquear la aplicación.

    5.  Si el análisis es exitoso, actualiza el estado `result` con los datos y lo guarda en el historial del navegador (`localStorage`).

    6.  Finalmente, establece `isLoading` a `false`.

-   Renderizado Condicional: La interfaz cambia según el estado:

    -   Si no hay `preview`, muestra el área para subir archivos.

    -   Si hay `preview` pero no `result`, muestra la imagen y el botón "Analizar".

    -   Si hay `result`, muestra las tarjetas con el diagnóstico, la confianza y el desglose por modelo.

**4.2. Acciones del Servidor - `src/app/actions.ts`**

Este archivo es crucial para la seguridad y la fiabilidad. Define una `Server Action`, que es una función que, aunque se llama desde el cliente, se ejecuta exclusivamente en el servidor.

-   `'use server';`: Esta directiva indica a Next.js que el código de este archivo es una Server Action.

-   Función `analyzeImageAction`:

    1.  Recibe la imagen como una cadena Base64.

    2.  Seguridad: Valida la entrada usando la librería `zod` para asegurarse de que es una cadena de imagen Base64 válida.

    3.  Lógica del Servidor:

        -   Convierte la cadena Base64 de nuevo a un `Buffer` (datos binarios).

        -   Crea un objeto `Blob` a partir del Buffer.

        -   Construye un `FormData` en el servidor, que es el formato que la API externa (`ensemble-api-qzpf.onrender.com`) espera.

    4.  Llamada a la API externa:

        -   Usa `fetch` para enviar el `FormData` a la API de análisis.

    5.  Manejo de Errores Robusto:

        -   Verifica si la respuesta de la API fue exitosa (`!response.ok`).

        -   Maneja explícitamente errores comunes como `502` o `503` (API inactiva o sobrecargada), devolviendo un objeto de error JSON `{ error: '...' }` claro.

        -   Intenta parsear la respuesta como JSON. Si falla, devuelve otro error controlado. Esto evita que la aplicación se rompa si la API devuelve HTML u otro formato inesperado.

    6.  Si todo va bien, devuelve los datos del análisis al cliente.

**4.3. Página de Historial - `src/app/history/page.tsx`**

Muestra los análisis previos guardados en el navegador del usuario.

-   `'use client'`: Es un componente de cliente porque necesita acceder a `localStorage`, que solo existe en el navegador.

-   `useEffect` y `useState`:

    -   `useEffect` se usa para leer el historial desde `localStorage` de forma segura, solo después de que el componente se haya montado en el cliente. Esto evita errores de hidratación de Next.js.

    -   `useState` almacena la lista de items del historial.

-   Funcionalidad:

    -   Muestra los resultados en tarjetas (`<Card>`).

    -   Permite borrar todo el historial con una ventana de confirmación (`<AlertDialog>`).

    -   El historial guarda una versión pequeña de la imagen para no ocupar demasiado espacio en `localStorage`.

**4.4. Componentes y Estilos**

-   `src/components/header.tsx`: Define la barra de navegación superior. Usa el hook `usePathname` de Next.js para resaltar el enlace de la página activa.

-   `src/app/layout.tsx`: Es el componente principal que define la estructura HTML (etiquetas `<html>` y `<body>`). Importa las fuentes de Google Fonts y los estilos globales de `globals.css`. También incluye el componente `<Toaster>` para que las notificaciones funcionen en toda la aplicación.

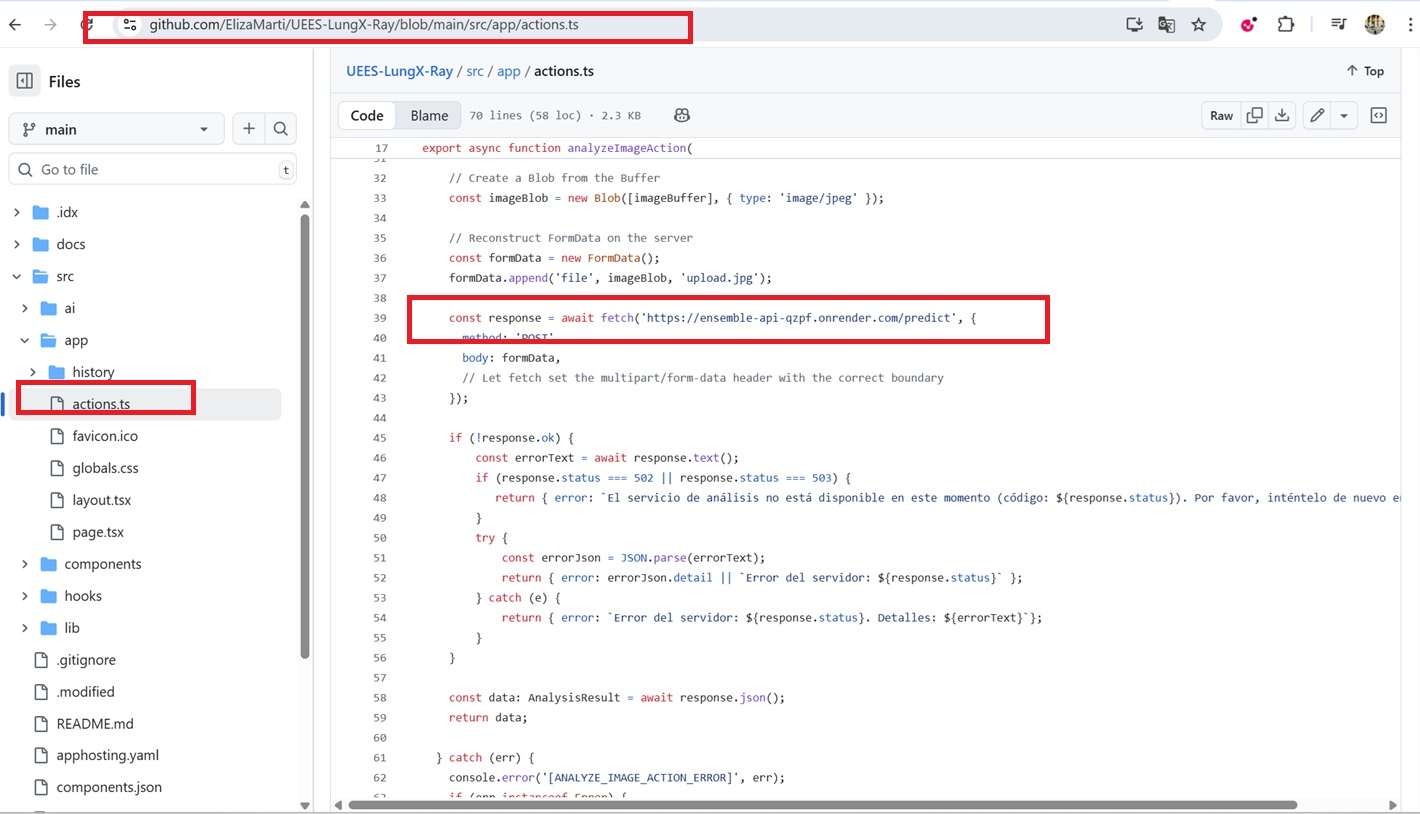
-   `src/app/globals.css` y `tailwind.config.ts`: Definen el tema visual de la aplicación. `globals.css` contiene las variables de color (en formato HSL) que ShadCN utiliza, como `--primary`, `--secondary`, `--destructive`, etc. `tailwind.config.ts` configura las fuentes y extiende el tema de Tailwind CSS.

**4.5. Integración con IA (Genkit) - `src/ai/`**

Aunque la llamada principal se hace a una API externa generada en base a los conocimientos adquirdos en clases , el proyecto también incluye una configuración de Genkit, un framework de Google para construir aplicaciones de IA.

**5. Conclusión**

El proyecto es un excelente ejemplo de una aplicación moderna full-stack construida con Next.js. Combina una interfaz de usuario reactiva y bien diseñada (React, ShadCN) con una lógica de backend segura y robusta (Server Actions), comsume la API <https://ensemble-api-qzpf.onrender.com> que contiene la lógica funcional de los resultados de las consultas de las radiografias.



La solución de convertir imágenes a Base64 en el cliente antes de enviarlas al servidor demuestra un enfoque práctico y efectivo para resolver problemas de compatibilidad con dispositivos móviles, lo que es crucial en el desarrollo web actual.