📌 Actividad 1: Creación del Proyecto del Chatbot

🧩 Historia de usuario

COMO estudiante de desarrollo web,
QUIERO crear un proyecto base en React que funcione correctamente en mi navegador,
PARA tener una base estable sobre la que construir mi chatbot en las siguientes actividades.

🧠 Tareas

🧱 Tarea 1: Creación del proyecto React

COMO desarrollador que va a construir un chatbot,
QUIERO generar un proyecto React desde cero con una herramienta moderna (por ejemplo Vite o Create React App),
PARA disponer de una aplicación inicial lista para ejecutarse con sus dependencias instaladas.

🖥️ Tarea 2: Ejecución y verificación del entorno

COMO estudiante que valida su entorno de trabajo,
QUIERO iniciar el servidor de desarrollo y abrir la aplicación en el navegador,
PARA asegurarme de que React se ejecuta sin errores y renderiza contenido en pantalla.

🎨 Tarea 3: Personalización inicial del proyecto

COMO creador de mi propio asistente,
QUIERO renombrar visualmente el proyecto y mostrar en pantalla el nombre o logotipo provisional de mi chatbot,
PARA identificar claramente que este proyecto es “mi chatbot” desde la primera versión y no una plantilla genérica.

📂 Tarea 4: Preparación de la estructura base

COMO estudiante que está preparando el chatbot para crecer,
QUIERO definir desde ya una estructura de carpetas organizada para componentes, servicios, estilos y recursos,
PARA tener una base escalable y ordenada que pueda reutilizar en las siguientes actividades (chat, conexión a APIs, etc.).

Estructura esperada:

```
src/
 ├── App.jsx              # Componente raíz de la aplicación
 ├── main.jsx             # Punto de entrada
 ├── components/              # Aquí irá el chatbot y sus partes visuales
 ├── services/               # Aquí irán las llamadas externas (por ejemplo POKEAPI)
 ├── styles/              # Hojas de estilo / estilos globales
public/
 └── assets/              # Logotipos e imágenes del chatbot
```

📸 Tarea 5: Evidencia del funcionamiento

COMO estudiante que entrega un primer hito funcional,
QUIERO capturar visualmente la app arrancada en el navegador mostrando el nombre o identidad del chatbot,
PARA poder demostrar que el proyecto ya corre en local y está listo para evolucionar en las siguientes actividades.

🧪 Pruebas funcionales de la actividad

✅ Prueba: Comprobación de arranque y estructura inicial del proyecto

1. Ejecutar el servidor de desarrollo del proyecto React.
2. Abrir el proyecto en el navegador y confirmar que se muestra en pantalla el nombre o logotipo provisional del chatbot.
3. Comprobar que no aparecen errores visibles en la aplicación durante la carga.
4. Mostrar la estructura de carpetas creada (`src/`, `components/`, `services/`, `styles/`, `public/`, `assets/`) para evidenciar que está preparada para el chatbot.
5. Registrar un GIF corto donde se vea:
    - el arranque del proyecto,
    - la vista en el navegador con la identidad del chatbot,
    - y la estructura del proyecto preparada para continuar en las siguientes actividades.

📌 Actividad 2: Personalización Visual del Chatbot

🧩 Historia de usuario

COMO estudiante que está dando forma a su chatbot,
QUIERO personalizar la apariencia inicial del proyecto sustituyendo todos los elementos visuales genéricos por la identidad de mi asistente (nombre, logo, colores básicos, iconos),
PARA tener una interfaz que represente mi chatbot real y no una plantilla por defecto.

🧠 Tareas

🎨 Tarea 1: Identidad visual del chatbot

COMO creador del chatbot,
QUIERO definir la identidad básica de mi asistente (nombre comercial, eslogan corto, logo provisional o icono identificable),
PARA que la aplicación tenga una marca clara desde el principio y cualquiera que la vea entienda “este es mi chatbot”.

🖼️ Tarea 2: Sustitución de elementos visuales por elementos propios

COMO estudiante que adapta la plantilla a su proyecto,
QUIERO reemplazar todos los recursos visuales genéricos que vienen por defecto en React (logos, iconos de ejemplo, textos placeholder tipo “React App”, etc.) por recursos que representen a mi chatbot,
PARA que la pantalla inicial muestre mi asistente y no referencias genéricas del framework.

Esto incluye:

- Cambiar el logo/imágenes por un recurso propio dentro de `public/assets/`.
- Cambiar textos genéricos visibles en pantalla por textos que describan el chatbot.
- Asegurar que ya no aparece ningún logotipo ni referencia visual a React por defecto.

🧭 Tarea 3: Ajuste visual mínimo de la pantalla de inicio

COMO usuario que va a interactuar con el chatbot,
QUIERO ver una pantalla inicial limpia donde aparezcan claramente el nombre del asistente, su identidad visual y una breve frase que explique qué hace,
PARA entender nada más abrir la app cuál es el propósito del chatbot (por ejemplo: “Asistente de ayuda”, “Buscador inteligente de Pokémon”, “Soporte técnico IA local”, etc.).

Esta pantalla será la base sobre la que luego se integrará la ventana de chat.

🗂️ Tarea 4: Organización de recursos gráficos

COMO estudiante que prepara su proyecto para escalar,
QUIERO guardar los recursos visuales personalizados (logos, iconos, imágenes del chatbot) dentro de una carpeta clara dentro de `public/assets/`,
PARA mantener ordenado el proyecto y reutilizar esos recursos visuales más adelante en el propio chatbot y en el futuro historial de chat.

Referente a la estructura de carpetas trabajada en la Actividad 1:

```
public/
 └── assets/
     ├── images/      # Logotipo / imagen principal del chatbot
     └── icons/       # Iconos que represente al chatbot, si existen
```

📝 Tarea 5: Reflejar la identidad en la interfaz

COMO responsable de producto de mi propio chatbot,
QUIERO que el nombre y la imagen del chatbot aparezcan visibles en la interfaz inicial de la aplicación React,
PARA poder enseñar la aplicación en ejecución y decir “esto es mi asistente” de forma clara y sin tener que explicarlo oralmente.

Esto implica que:

- El nombre del chatbot sea visible en la pantalla principal.
- El logotipo/imagen elegida aparezca también en pantalla.
- El eslogan o descripción corta esté presente (ejemplo: “Tu asistente local de consulta” o “Busco Pokémon por ti”).

🧪 Pruebas funcionales de la actividad

✅ Prueba: Comprobación de la identidad visual del chatbot en ejecución

1. Ejecutar la aplicación en el navegador.
2. Verificar que ya no aparece ningún elemento visual genérico del proyecto inicial (por ejemplo, el logo por defecto de React).
3. Confirmar que la pantalla inicial muestra:
    - Nombre del chatbot,
    - Imagen o icono representativo del chatbot (cargado desde `public/assets/`),
    - Breve descripción/eslogan de qué hace o qué hará el chatbot.
4. Registrar un GIF corto donde se vea la aplicación en ejecución mostrando la nueva identidad visual personalizada del chatbot.

📌 Actividad 3: Estructura del Proyecto y Modularización

🧩 Historia de usuario

COMO estudiante que está profesionalizando su chatbot,
QUIERO organizar mi proyecto React en una estructura clara con carpetas separadas para componentes, estilos, servicios y recursos,
PARA poder seguir creciendo (chat, conexión a APIs, historial, etc.) sin que el código se vuelva caótico.

🧠 Tareas

📂 Tarea 1: Estructurar el proyecto por responsabilidad

COMO desarrollador que va a mantener este chatbot más allá de hoy,
QUIERO separar el código en carpetas según su función (interfaz, lógica externa, estilos, recursos),
PARA tener una arquitectura mantenible y alineada con cómo se construyen aplicaciones React reales.

La estructura esperada tras esta actividad debe ser, como mínimo:

```
src/
 ├── App.jsx                # Punto de unión de toda la interfaz
 ├── main.jsx          # Punto de arranque de la aplicación React
 ├── components/            # Componentes visuales reutilizables
 │   └── Chatbot/           # Carpeta específica para el chatbot
 │       ├── ChatWindow.jsx       # Contenedor visual del chat
 │       ├── MessageList.jsx         # Listado de mensajes de la
conversación
 │       └── MessageInput.jsx     # Zona donde el usuario escribe
 ├── services/              # Conexiones externas / futuras APIs
 │   └── pokeapi.js  # Aquí irá la lógica para consultar la POKEAPI
 ├── styles/                     # Estilos globales y estilos por
componente
 │   ├── chatbot.css             # Estilos asociados al chatbot
 │   └── layout.css      # Estilos comunes de estructura
public/
 └── assets/                # Recursos estáticos del proyecto
     ├── images/             # Logos / ilustraciones del chatbot
     └── icons/              # Iconos personalizados del chatbot
```

🧱 Tarea 2: Aislar los componentes del chatbot dentro de su propio directorio

COMO responsable del chatbot dentro de la aplicación,
QUIERO agrupar los componentes relacionados con la interfaz de chat (ventana principal, lista de mensajes, cuadro de entrada, etc.) dentro de una carpeta dedicada,
PARA que todo lo que forma parte del chatbot esté localizado, sea fácil de encontrar y sea fácil de extender cuando más adelante añadamos cosas como historial, streaming de respuesta o avatar.

Esto implica:

- Tener un directorio propio del chatbot dentro de `src/components/`.
- Separar visualmente el contenedor del chat, la lista de mensajes y el input del usuario en piezas distintas.
- Evitar que todo el chatbot esté amontonado en un único componente enorme.

📜 Tarea 3: Preparar la carpeta `services/` para llamadas a datos

COMO desarrollador que va a consumir datos externos,
QUIERO crear una carpeta `services/` donde viva la lógica de obtención de datos externos (por ejemplo, las consultas a la POKEAPI en la siguiente actividad),
PARA no mezclar la vista (componentes visuales) con la lógica de obtención de datos (llamadas HTTP, parseo de respuestas, etc.).

Esto anticipa:

- Un archivo dedicado a la obtención de datos de Pokémon.
- Que los componentes del chatbot no conozcan los detalles de la API, sólo pidan “dame info de este Pokémon”.

🎨 Tarea 4: Centralizar estilos

COMO creador de la interfaz del chatbot,
QUIERO tener una carpeta `styles/` donde guardar los estilos globales de la aplicación y los estilos específicos del chatbot,
PARA mantener una línea visual consistente y poder localizar rápidamente dónde se modifica la apariencia del chat.

Esto incluye:

- Crear un fichero de estilos específico para el chatbot.
- Crear (o planificar) estilos comunes para la estructura general de la página (layout).

🧠 Tarea 5: Integrar todo en la aplicación raíz

COMO usuario que quiere ver el chatbot formar parte real de la app,
QUIERO asegurar que la app principal (por ejemplo el componente raíz `App.jsx`) conoce y utiliza la estructura creada (`components`, `services`, `styles`, `assets`),
PARA confirmar que la modularización no es solo teórica sino que el proyecto ya está preparado para mostrar un chatbot con su propia identidad visual y su propio espacio dentro de la página.

Esto significa:

- La aplicación debe estar ya preparada para mostrar una zona claramente identificada como “aquí vive el chatbot”.
- Se debe reflejar visualmente que el chatbot es parte central del proyecto, no un añadido sin contexto.

🧪 Pruebas funcionales de la actividad

✅ Prueba: Comprobación de estructura y renderizado del chatbot dentro de la app

1. Ejecutar la aplicación en el navegador.
2. Verificar que la interfaz principal de la app incluye ya una zona del chatbot reconocible (por ejemplo, un área diferenciada para conversación).
3. Comprobar que el proyecto está organizado con las carpetas indicadas (`components/Chatbot`, `services/`, `styles/`, `public/assets/`) y que no queda todo mezclado en un único archivo raíz.
4. Registrar un GIF en el que se muestre:
    - La aplicación corriendo, enseñando la sección reservada al chatbot.
    - La estructura de carpetas del proyecto abierta (por ejemplo en el explorador del editor), evidenciando que se ha aplicado la modularización.

📌 Actividad 4: Componentes mínimos del chatbot (inspirado en ChatGPT / Claude)

🧩 Historia de usuario

COMO usuario que va a interactuar con el chatbot dentro de la aplicación,
QUIERO disponer de una interfaz de chat que se parezca a una experiencia real (historial de mensajes, zona de escritura y área de respuesta del asistente),
PARA poder conversar con el chatbot dentro del navegador de forma clara, ordenada y comprensible.

🧠 Tareas

💬 Tarea 1: Ventana principal del chat

COMO usuario que quiere "ver" al chatbot,
QUIERO una ventana de chat visible en la aplicación con aspecto similar a las interfaces conocidas (por ejemplo ChatGPT o Claude), con un área dedicada a la conversación,
PARA entender visualmente que este espacio es el canal para hablar con el asistente.

Esta ventana debe estar integrada en la aplicación (no en otra pestaña ni escondida), y debe identificarse de forma clara como "Chatbot", "Asistente", "IA", etc.

🗂️ Tarea 2: Historial de mensajes

COMO persona que conversa con el chatbot,
QUIERO ver un listado de mensajes anteriores en orden cronológico (mis mensajes y las respuestas del asistente),
PARA poder seguir el contexto de la conversación sin perder quién ha dicho qué, igual que ocurre en herramientas tipo ChatGPT o Claude.

Esto implica:

- Diferenciar visualmente mensajes del usuario y mensajes del asistente.
- Presentar el contenido de cada mensaje como "burbujas" o bloques claramente separados.
- Mantener visible el historial completo dentro de la ventana de chat.

⌨️ Tarea 3: Área de entrada de mensaje

COMO usuario que quiere preguntar cosas al chatbot,
QUIERO tener un campo de entrada de texto donde pueda escribir mi mensaje y un control claro para enviarlo (por ejemplo, un botón “Enviar”),
PARA poder iniciar la interacción con el asistente igual que haría en cualquier otra interfaz conversacional moderna.

Este campo de entrada debe:

- Estar siempre accesible en la parte inferior de la ventana del chat.
- Limitarse a texto en esta fase (sin adjuntar archivos ni nada especial todavía).

🤖 Tarea 4: Respuesta simulada del asistente

COMO usuario que está probando la experiencia del chatbot,
QUIERO que cada vez que envío un mensaje aparezca después una respuesta de “asistente” en el historial (aunque sea provisional / simulada),
PARA sentir que estoy hablando con algo que me responde y visualizar el flujo conversación → respuesta, como en un chat real.

El objetivo aquí no es la inteligencia real todavía, sino el comportamiento de conversación.
Es decir: envío un mensaje, aparece mi mensaje en el historial, y luego aparece un mensaje del asistente como respuesta.

⏳ Tarea 5: Indicador de respuesta / “pensando…”

COMO usuario que espera una respuesta del asistente,
QUIERO ver algún tipo de indicador visual temporal (por ejemplo texto tipo “Pensando…” o un pequeño estado de carga) cuando el asistente está generando su respuesta,
PARA entender que el chatbot está procesando mi mensaje, igual que hacen asistentes tipo ChatGPT y Claude cuando muestran que están generando la respuesta.

Este indicador debe mostrarse en la propia zona del chat, no en otra parte de la pantalla, para que visualmente parezca que el asistente está escribiendo.

👁️ Tarea 6: Identidad visual del asistente en el chat

COMO usuario que usa este chatbot como producto propio,
QUIERO que la ventana de chat refleje la identidad definida en las actividades anteriores (nombre del asistente, icono/avatar si lo tengo, colores básicos),
PARA que no parezca un chat genérico sino que se reconozca quién me está respondiendo.

Esto acerca la experiencia a la de un producto real y no a una demo técnica.

🧪 Pruebas funcionales de la actividad

✅ Prueba: Comprobación de la interfaz conversacional del chatbot

1. Ejecutar la aplicación en el navegador.
2. Verificar que existe una ventana de chat claramente identificada como el asistente.
3. Escribir un mensaje en el área de entrada y “enviarlo”.
4. Comprobar que:
    - El mensaje enviado aparece en el historial como mensaje del usuario.
    - Se muestra algún estado de “pensando…” o equivalente.
    - Aparece después un mensaje del asistente dentro del mismo historial de chat.
5. Confirmar que en ese historial se distinguen visualmente “usuario” y “asistente” (por color, lado, etiqueta de autor, avatar, etc.), de forma similar a una interfaz tipo ChatGPT / Claude.
6. Grabar un GIF corto enseñando toda la interacción anterior en una única ejecución del navegador.

📌 Actividad 5: Búsqueda de Pokémon con POKEAPI desde el chatbot

🧩 Historia de usuario

COMO usuario que está usando el chatbot dentro de la aplicación,
QUIERO poder preguntarle por un Pokémon escribiendo su nombre o su número de la Pokédex nacional directamente en la ventana de chat,
PARA que el chatbot me responda con información básica de ese Pokémon obtenida desde una API real (POKEAPI), simulando así una utilidad práctica del asistente.

🧠 Tareas

🔍 Tarea 1: Interpretación de la consulta del usuario

COMO usuario que quiere información de un Pokémon,
QUIERO poder escribir en el chat cosas como “pikachu” o “25”,
PARA que el chatbot entienda que estoy pidiendo datos de ese Pokémon concreto y procese esa búsqueda.

Esto implica que el mensaje que se introduce en el área de entrada del chat se pueda usar como criterio de búsqueda, sin obligarme a seguir un formato especial.
Ejemplos de entrada válidos: nombre del Pokémon (“bulbasaur”), número en la Pokédex nacional (“1”).

🌐 Tarea 2: Consulta a la POKEAPI

COMO desarrollador que conecta el chatbot con datos reales,
QUIERO enviar una petición a la POKEAPI usando el valor que el usuario ha escrito (nombre o número),
PARA obtener la información oficial del Pokémon y usarla para construir la respuesta del asistente.

La lógica que consulta la POKEAPI debe estar preparada en un servicio dedicado dentro de la carpeta `services/` (por ejemplo, un módulo que centralice estas peticiones y devuelva datos ya “limpios”: nombre, imagen principal/sprite y tipo del Pokémon).
Este servicio actuará como punto único de acceso externo, en línea con la estructura creada en las actividades anteriores.

🧾 Tarea 3: Formato de la respuesta del chatbot

COMO usuario que hace la consulta,
QUIERO que el chatbot me responda en el propio historial de conversación con los datos más relevantes del Pokémon encontrado (por ejemplo, su nombre, su número en la Pokédex y su tipo principal),
PARA recibir una respuesta útil y legible directamente en el chat, igual que si el asistente fuera un buscador integrado.

La respuesta debe expresarse con claridad dentro de la misma conversación, igual que hace un asistente tipo “te lo cuento en texto”, no como una alerta aparte ni una consola oculta.

❌ Tarea 4: Manejo de errores y Pokémon no encontrados

COMO usuario que puede escribir mal o pedir algo que no existe,
QUIERO que el chatbot me avise si el Pokémon que he pedido no está disponible o el nombre/número no es válido,
PARA entender qué ha fallado sin que la aplicación se rompa ni deje el chat en blanco.

Esto incluye:

- Gestión de errores de red o errores al consultar la POKEAPI.
- Gestión de casos en los que el usuario pide un Pokémon que no existe.
- Mensaje claro en la propia conversación (“No encuentro ese Pokémon”, etc.), igual que haría un asistente real.

🔄 Tarea 5: Integración natural en el flujo de chat

COMO usuario que conversa con el asistente,
QUIERO que la respuesta con los datos del Pokémon aparezca como un mensaje más del asistente dentro del historial de chat (y no como algo separado),
PARA que la experiencia se sienta como una conversación continua, donde puedo preguntar por varios Pokémon uno detrás de otro y ver cada resultado en orden.

El comportamiento esperado es:

1. Yo escribo “25” → el mensaje aparece en el historial como mensaje del usuario.
2. El chatbot muestra visualmente que está “pensando…”.
3. El chatbot responde con los datos del Pokémon nº25 de la Pokédex nacional.
4. Todo queda registrado en el historial junto con las consultas anteriores (por ejemplo, si antes pregunté por “1” y luego por “25”, debo poder ver ambas respuestas listadas en orden).

🧠 **Tarea 6: Uso coherente de la estructura del proyecto**

COMO estudiante que quiere dejar el proyecto listo para siguientes sprints,
QUIERO que la lógica de conversación (componentes de interfaz del chat) y la lógica de datos (consulta a la POKEAPI) estén separadas en las carpetas ya definidas en la Actividad 3,
PARA mantener una arquitectura limpia donde el chatbot pueda más adelante llamar a otras utilidades externas (por ejemplo, más adelante LM Studio o utilidades internas), sin tener que reescribir toda la app.

Recordatorio de estructura trabajada hasta ahora:

```
src/
 ├── components/
 │   └── Chatbot/
 │        ├── ChatWindow.jsx        # Ventana principal del chat
 │        ├── MessageList.jsx       # Historial de mensajes (usuario
/ asistente)
 │        └── MessageInput.jsx      # Zona de escritura y envío de
mensaje
 ├── services/
 │   └── pokeapi.js                 # Lógica de obtención de datos
de la POKEAPI
 ├── styles/
 │    ├── chatbot.css                      # Apariencia del chat,
incluyendo estado "pensando..."
 │   └── layout.css
public/
 └── assets/
     ├── images/                   # Recursos visuales del chatbot
     └── icons/                    # Icono / avatar del asistente
```

🧪 Pruebas funcionales de la actividad

✅ Prueba 1: Búsqueda por nombre

1. Ejecutar la aplicación en el navegador.
2. Escribir en el chat el nombre de un Pokémon existente (por ejemplo, “pikachu”).
3. Verificar que el chatbot responde dentro del historial con la información básica del Pokémon consultado (por ejemplo nombre, número y tipo).
4. Confirmar que esa respuesta aparece etiquetada como mensaje del asistente.

✅ Prueba 2: Búsqueda por número de Pokédex nacional

1. Escribir en el chat un número válido de Pokédex nacional (por ejemplo, “1”, “25”, “150”).
2. Verificar que el chatbot obtiene la información del Pokémon asociado a ese número y la muestra en el historial de conversación como respuesta del asistente.
3. Confirmar que esta respuesta convive en el mismo historial junto con las consultas anteriores.

❌ Prueba 3: Manejo de error / Pokémon inexistente

1. Escribir en el chat un nombre inventado o un número que no corresponde a ningún Pokémon.
2. Verificar que el chatbot devuelve un mensaje de error entendible dentro del propio historial (por ejemplo, indicando que no encuentra ese Pokémon), sin romper la interfaz.
3. Confirmar que el resto de la interfaz del chat sigue funcionando con normalidad tras el error y que puedo seguir preguntando por otros Pokémon.