1.-- Explica con tus palabras que es una API:

Una Api es una interfaz de programación de aplicaciones en términos simples una Api es como un el menú de un restaurant (el cliente) se siente a la mesa y abre el menú

- El menú (Api) describe los platos ingredientes (servicios) que están disponibles en este caso (Datos)
- El cliente le pide un plato del menú al mesero
- El mesero trae la orden del cliente (del servidor)

En términos técnicos

- **Tú** eres el frontend (tu app web).
- **El menú** es la documentación de la API (qué endpoints existen y qué datos aceptan/devuelven).
- **El camarero** es el protocolo HTTP (fetch, axios…), que lleva tu petición al servidor y vuelve con la respuesta.
- **La cocina** es el back-end donde está la lógica y la base de datos.
- **El plato que te sirven** es el JSON con los datos que pediste (por ejemplo, un token de autenticación o la información de perfil).

Así seria la forma que explicaría que es una Api

2.- En caso de haber utilizado un framework de estilos, justifica su uso y porque elegiste ese:

**Tailwind CSS**

- Por qué lo escogí:
    - **Productividad:** Tailwind ofrece utilidades listas (paddings, colores, sombras…) que aceleran la maquetación sin escribir CSS a mano.
    - **Consistencia:** Al usar clases atómicas, el sistema de diseño (espaciados, paleta de colores, tipografías) se mantiene uniforme.
    - **Flexibilidad:** Permite personalizar temas (colores primarios, breakpoints) directamente en tailwind.config.js y generar sólo el CSS que usas

3.-En caso de haber utilizado un patron de componentes, justifica su uso y porque elegiste ese:

Patrón usado: **Atomic Components** (inputs, botones, tarjetas) + Page Composition (páginas Astro que ensamblan layout + React).

Por qué:

- **Reutilización:** Cada componente (Login, Input, Button, ProfileCard) vive en su propia carpeta y puede usarse o extenderse fácilmente.

- Responsabilidad única: Un componente hace una cosa (p.ej. <LoginForm /> sólo maneja UI y validación), mientras que la lógica de negocio (autenticación, sesión) vive en un hook separado (useAuth).

4.--¿Que patrones de diseno conoces?, cuales haz aplicado, explica porque lo utilizaste y como te ayudo a resolver un problema o tuvo una ventaja sobre otros.

Conozco lo que es:

- Singleton : sirve para instanciar un solo objeto global por ejemplo: un store de configuración o un cliente HTPP
- Hook Pattern: que es el encargado de extraer la lógica y efectos en hooks reutilizables(UseAuth)

Los que eh aplicado son :

- **Hook pattern:** useAuth encapsula todo lo relativo a autenticación/sesión. Esto separa UI de lógica y facilita las pruebas.
- **Composition:** Las páginas Astro (login.astro, profile.astro) componen <Layout> + <Login> o <Profile>, sin mezclar lógica interna.

5.- ¿Haz utilizado Sockets?. Si es asi, explica el porque lo utilizaste y sus ventajas:

En esta pequeña prueba no eh utilizado nada de Sockets por que la autentificación y perfil son flujos de petición/respuesta estáticos

Una de sus ventajas es:

- Baja latencia
- Conexión persistente
- Bidireccionalidad