Trabajo Practico – Integración PHP puro + React (Hooks) – Versión Conceptual

Respuestas a las Preguntas Conceptuales

1. Una API es una interfaz (endpoints HTTP) que expone datos y acciones del servidor. En React+PHP sirve para que el front pida/mande información (JSON) y el back procese la lógica y la DB.
2. “Desacoplados” significa que front y back son proyectos independientes que se comunican por la API. Ventaja: se desarrollan, despliegan y escalan por separado y se pueden reemplazar o reutilizar fácilmente.
3. Con PHP que devuelve la página ya armada recibís HTML listo; con JSON recibís sólo datos y React construye la vista en el cliente, permitiendo interactividad más dinámica y menores recargas completas.
4. JSON es más legible por humanos y nativo en JavaScript (objetos). Ejemplo simple: {"id":1,"titulo":"Comprar leche","completa":false}.
5. Pedís datos → GET; creas algo nuevo → POST; modificas → PUT/PATCH; borras → DELETE. Es el mapa conceptual de las operaciones con la API.
6. Ejemplos: lista de tareas (index), detalle/edición de una tarea, formulario de crear tarea, panel con filtros/estadísticas.
7. CORS es la política de mismo origen del navegador; bloquea peticiones entre puertos/dominios distintos si el servidor no responde con Access-Control-Allow-Origin, por seguridad.
8. Es un número HTTP (200, 404, 500) que indica el resultado de la petición; permite a la app decidir si mostrar éxito, advertencia o error y dar mensajes claros al usuario.
9. Mensajes ejemplo: éxito → “Tarea guardada correctamente.”; no encontrado → “No encontramos lo que pediste.”; faltan datos → “Falta completar: título.”
10. Mostrar feedback: spinner o skeleton, deshabilitar botones, texto “Cargando…” y, si aplica, mostrar datos cacheados o una barra de progreso para no dejar sensación de bloqueo.
11. useState guarda los valores del formulario para controlar inputs, validar en tiempo real y restablecer/mostrar errores sin recargar la página.
12. Usaría useEffect para traer la lista al montar la pantalla (componentDidMount) y también cuando cambio filtros/página; el “efecto” se dispara por la dependencia (por ejemplo, [filtro]).

13. Mostrar un banner claro: “Sin conexión, revisa tu red”, botón reintentar, y si existen, datos cacheados para que la app siga usable parcialmente.
14. Resaltar los campos inválidos con mensajes inline, marcar en rojo y situar foco en el primer campo faltante para que la persona sepa exactamente qué completar.
15. Porque perder lo escrito frustra al usuario; mantener los campos y rehabilitar el botón facilita corregir el error y reintentar sin reescribir todo.
16. Capturas útiles: lista de tareas con contenido real (flujo normal), pantalla de creación/edición tras “guardado exitoso”, y una pantalla mostrando un error de validación o de red.
17. README breve: prerrequisitos, pasos para instalar (npm, PHP, composer), variables de entorno, comandos para levantar front y back, migraciones/seed y ejemplo de peticiones.
18. Los mocks permiten desarrollar y probar UI rápidamente, simular errores y casos límite, y evitar depender de DB o servicios inestables durante el desarrollo.
19. Mostrar al usuario un mensaje amable: “Ocurrió un error en el servidor, intenta más tarde.”; dejar detalles técnicos (stack/error) en la consola y en los logs del servidor para que lo revise el equipo.
20. Aprendí que separar pantalla y datos hace el desarrollo más ágil, facilita testing y reutilización (móvil/web), pero exige buena definición de contratos (API), manejo de CORS y versiones. En la próxima versión mejoraría la documentación de la API, añadiría tipos compartidos (DTOs/TS), pruebas automáticas y manejo de errores más granular.