PRUEBA TÉCNICA – DESARROLLADOR(A) FULLSTACK MID REACT NATIVE (48 HORAS)

Hola
Gracias por tu interés. Esta prueba tiene como objetivo evaluar tus habilidades prácticas y tu dominio de las tecnologías principales del rol.

Dispones de 48 horas para enviarnos tu entrega completa.

Tu entrega debe incluir:

- Tu proyecto final (mini-app React Native)
- Código fuente solicitados en los ejercicios
- Respuestas a las preguntas teóricas
- Un README con instrucciones para correr el proyecto
- Enlace al repositorio GitHub o ZIP comprimido

SECCIÓN 1 — PREGUNTAS TEÓRICAS (RESPUESTA ESCRITA)

Responde de manera clara y breve.
No se requiere código en esta sección.

1.1 React / React Native

1. Explica la diferencia entre useEffect, useMemo y useCallback. ¿Cuándo usarías cada uno para mejorar el rendimiento en móviles?
2. Explica cómo manejarías:
    - Cache de datos con React Query o Apollo Client
    - Sincronización offline → online
    - Prevención de doble request en pantallas navegables
3. ¿Qué diferencia existe entre desarrollar una app con Expo versus React Native CLI?
   ¿Cuál usarías para un proyecto productivo con cámara, deep linking y builds automatizadas?

4. Describe cómo optimizarías el rendimiento de una app React Native en:
    - Renderizado de listas
    - Navegación
    - Tamaño del bundle
    - Imágenes y assets

1.2 Arquitectura y Backend

5. Describe (o dibuja) una arquitectura simple con:
    - App móvil React Native
    - API REST o GraphQL
    - Cache local + sincronización
    - Flujo de autenticación (JWT, OAuth o Firebase Auth)
6. Explica cómo estructurarías la arquitectura interna de tu app (carpetas, componentes, servicios, hooks, estados).
7. Explica cómo implementarías un flujo básico de CI/CD para React Native usando EAS Build o Fastlane (solo conceptual).

SECCIÓN 2 — EJERCICIOS DE CÓDIGO (ENVIAR ARCHIVOS .TS / .JS)

En esta sección sí debes enviar código funcional.
Puedes entregar archivos sueltos o una carpeta llamada **/ejercicios** dentro del repositorio.

2.1 JavaScript / TypeScript

Ejercicio 1 — Palíndromo

Implementa una función en TypeScript que determine si una palabra o frase es un palíndromo.
Debe ignorar mayúsculas, acentos y espacios.

```
function isPalindrome(word: string): boolean {
  // tu código aquí
}
```

Ejercicio 2 — Lista sin duplicados

Dado un arreglo de objetos, devuelve una nueva lista sin duplicados según el campo id.

```
const users = [
 { id: 1, name: "Ana" },
 { id: 2, name: "Carlos" },
 { id: 1, name: "Ana Duplicate" },
];


function uniqueUsers(list: typeof users) {
  // tu código aquí
}
```

Ejercicio 3 — Rotación de matriz

Dada una matriz NxN, retorna la matriz rotada 90° hacia la derecha sin usar librerías:

```
function rotateMatrix(matrix: number[][]): number[][] {
 // tu código aquí
}
```

2.2 Backend

Crea un endpoint simple (en Node.js, .NET, Python o cualquier tecnología que manejes) que retorne un JSON como este:

```
{
 "id": 1,
 "title": "Demo item",
 "completed": false
}
```

Debe ser consumido en el mini proyecto de la siguiente sección.
Entrega el archivo del endpoint o el enlace al repositorio.

SECCIÓN 3 — MINI PROYECTO (REACT NATIVE) – 48 HORAS

Debes construir una pequeña aplicación móvil llamada **TaskSync**.
Puedes usar **Expo** o **React Native CLI**.

Este mini-proyecto busca evaluar tu dominio real de React Native, integración de APIs, estado global, rendimiento, TypeScript y buenas prácticas.

3.1 Funcionalidad requerida

La app debe permitir:

1. **Listar tareas** obtenidas desde tu API creada en la sección anterior.
2. **Crear nuevas tareas** (título y estado).
3. **Editar o eliminar tareas**.
4. **Marcar una tarea como completada**.
5. Manejar correctamente estados de carga, error y vacíos.

Puedes guardar los cambios localmente o sincronizarlos con tu API (cualquiera es válido mientras lo expliques en el README).

3.2 Requerimientos técnicos obligatorios

Debes implementar:

1. Manejo de estado global

Puedes usar cualquiera de estos:

- React Context
- Redux Toolkit
- Zustand
- Jotai

Indica por qué elegiste ese manejador.

2. Integración con la API

- Debes consumir el endpoint creado por ti.
- Manejar errores y estados de carga.
- Añadir retry básico o lógica mínima de reintento.

3. Cache + Offline First

La app debe ser usable sin conexión.
Requisitos mínimos:

- Mostrar la lista de tareas sin internet (usando AsyncStorage o similar).
- Guardar acciones realizadas offline.
- Sincronizar las acciones cuando vuelva la conexión.

La implementación puede ser sencilla (no debe ser nivel producción).
Solo debe funcionar.

4. Uso de una funcionalidad nativa

Debes implementar al menos **una** de estas:

- Notificaciones locales (o push si tienes experiencia)
- Cámara (tomar foto y asignarla a una tarea)
- GPS (ubicación al crear una tarea)
- Deep linking (abrir una tarea desde un enlace)

Explica en el README cuál elegiste.

5. Buenas prácticas obligatorias

- Proyecto completo en **TypeScript**
- Estructura clara de carpetas
- Linter + Prettier configurado
- Componentes reutilizables

- Código limpio y comentado donde sea necesario

6. Testing

Incluye al menos **un test** con Jest + React Native Testing Library:

- Puede ser un test de un componente o de un hook.

7. Documentación

Incluye un **README** con:

- Cómo correr el proyecto
- Requerimientos (Node, Expo, etc.)
- Decisiones técnicas tomadas
- Arquitectura de carpetas
- Explicación de la funcionalidad nativa usada
- Posibles mejoras futuras

FORMA DE ENTREGA

Envíanos:

- Enlace a un repositorio GitHub **(obligatorio)**
- Video corto (máx 2 minutos) mostrando la app funcionando
- Carpeta o archivos de los ejercicios de código
- Respuestas de teoría en un documento o dentro del README

Puedes adjuntar un ZIP adicional si lo deseas, pero el repositorio es obligatorio.