Prueba Técnica – Rol Técnico Integral Aloja

Parte 1: Comprensión de Código y Resolución de Incidencias

Objetivo: Simular una incidencia operativa y su resolución.

Caso:

Un anfitrión reporta que al cargar su inmueble en la web, el botón “Publicar Inmueble” no realiza ninguna acción. La consola muestra este error:

```
TypeError: Cannot read properties of undefined (reading 'tipo')
```

Respuesta:

Indica que estamos intentando acceder a la propiedad 'tipo' de un objeto que es undefined. En ReactJS, esto suele ocurrir cuando:

- Estamos accediendo a una propiedad de un objeto que no ha sido inicializado correctamente.
- Los datos asíncronos no se han cargado todavía cuando el componente intenta renderizarse.
- Hay un error en la estructura de los datos recibidos del backend.

Una posible solución se pudiese dar de la siguiente manera:

```
// Solución común sería agregar protección contra undefined
const handlePublish = () => {
  // Antes: error si formData es undefined
  // if (formData.tipo === 'casa') {...}
  // Solución:
  if (formData?.tipo === 'casa') { //
    // lógica de publicación
  }
}
```

Para dar esta posible solución se deben de tomar las siguientes consideraciones:

- Identificar en qué parte del código se produce el error (usando stack trace de la consola)
- Verificar el componente que contiene el botón "Publicar Inmueble"
- Revisar cómo se maneja el estado del formulario
- Comprobar si hay datos asíncronos que se cargan después del renderizado.

Una de las posibles mejores prácticas a contemplar es hacer uso de:

- **Usar try-catch en operaciones riesgosas.**
- **Escribir tests unitarios para casos edge.**

Parte 2: Desarrollo Full Stack Ligero

Objetivo: Validar capacidad para realizar ajustes en componentes de frontend y backend.
Caso:

Se desea agregar un nuevo campo en el formulario de creación de inmueble llamado: Tipo de conexión a Internet con opciones: “Fibra Óptica”, “4G LTE”, “ADSL”, “No Disponible”.

Respuestas:

1. Pasos para implementar el nuevo campo

Frontend (ReactJS):

- Actualizar el esquema de validación del formulario
- Modificar el componente del formulario para incluir el nuevo campo
- Actualizar la función de submit para incluir el nuevo dato
- Actualizar las interfaces/types de TypeScript si se usan

Backend (NodeJS + Postgres):

- Crear una migración para agregar la nueva columna
- Actualizar el modelo/schema de la base de datos
- Modificar el endpoint API para aceptar el nuevo campo
- Actualizar la validación en el backend

2. Consideraciones para migración de BD

Migración segura:

- Crear backup antes de ejecutar migraciones
- Usar transacciones para migraciones complejas
- Planear migración en horario de bajo tráfico

Compatibilidad:

- Definir valor por defecto para registros existentes.
- Considerar si el campo será nullable o no.

Rendimiento:

- Evaluar necesidad de indexación
- Considerar longitud del campo si es texto

Una propuesta del cambio, tanto en Reactjs como en PostgreSQL es la siguiente:

Componente de Reacjs

```
import { Select, MenuItem } from '@mui/material';
function PropertyForm({ onSubmit }) {
  const [formData, setFormData] = useState({
    // otros campos...
    internetType: 'No Disponible' // valor por defecto
  });

  const internetOptions = [
    'Fibra Óptica',
    '4G LTE',
    'ADSL',
    'No Disponible'
  ];

  return (
    <form onSubmit={handleSubmit}>
      {/* otros campos del formulario */}

      <Select
        value={formData.internetType}
        onChange={(e) => setFormData({...formData, internetType: e.target.value})}
        label="Tipo de conexión a Internet"
        required
      >
        {internetOptions.map(option => (
          <MenuItem key={option} value={option}>{option}</MenuItem>
        ))}
```

```
      </Select>
      <button type="submit">Publicar Inmueble</button>
    </form>
  );
}
```

Migración SQL para Postgres:

Contenido sql:

```
-- migrations/20230501120000_add_internet_type_to_properties.sql
BEGIN;

ALTER TABLE properties
ADD COLUMN internet_type VARCHAR(20) NOT NULL DEFAULT 'No Disponible';

COMMIT;

-- Opcional: crear índice si se filtrará frecuentemente por este campo
--       CREATE       INDEX       idx_properties_internet_type       ON
properties(internet_type);
```

Parte 3: Diagnóstico de Integración

1. Pasos para diagnosticar

Verificar flujo de registro:

- Confirmar que la petición llega al backend.
- Revisar logs del servidor para ver si se procesó el registro.

Comprobar integración con Firebase:

- Verificar configuración de Firebase en el backend.
- Confirmar que las credenciales son correctas.
- Revisar si hay errores al inicializar Firebase.

Analizar código de envío:

- Localizar la función que envía el correo de confirmación
- Verificar que se está llamando correctamente
- Comprobar manejo de errores

Probar en diferentes entornos:

- Verificar si ocurre en desarrollo, staging y producción
- Probar con diferentes tipos de cuentas

2. Puntos del sistema a revisar

Frontend (React Native):

- Implementación del formulario de registro
- Manejo de la respuesta del API

Backend (NodeJS):

- Endpoint de registro de usuario.
- Integración con Firebase Auth.
- Servicio de envío de emails.

Infraestructura:

- Configuración de Firebase
- Credenciales y variables de entorno
- Reglas de seguridad/permisos

Monitorización:

- Logs del servidor
- Errores de Firebase
- Métricas de envío de emails.

3. Comunicación al equipo no técnico

Se debe redactar una comunicación para el usuario no técnico, a los correctivos a realizarse, como el siguiente:

"Estimado equipo,

Hemos identificado que el problema con los correos de confirmación se debe a un fallo en la conexión entre nuestro servidor y el servicio de Firebase. Específicamente, las credenciales de

autenticación no se están enviando correctamente cuando un nuevo usuario se registra desde la app móvil.

Impacto: *Los usuarios pueden registrarse pero no reciben el correo de confirmación, lo que podría afectar su capacidad para acceder a todas las funciones.*

Solución propuesta:

1. *Actualizar la configuración de conexión con Firebase*
2. *Implementar un sistema de reintentos para el envío de correos*
3. *Agregar notificaciones alternativas en la app mientras se resuelve*

Tiempo estimado: *2-4 horas para la corrección.*

Mantendremos informado al equipo sobre el progreso."

Parte 4: Bonus (Deseable)

Caso práctico opcional:

Query SQL para reporte de inmuebles:

```
SELECT

  COUNT(CASE WHEN estado = 'publicado'

        AND fecha_publicacion >= NOW() - INTERVAL '24 hours'

        THEN 1 ELSE NULL END) AS publicados_ultimas_24h,

  COUNT(CASE WHEN estado = 'pendiente' THEN 1 ELSE NULL END) AS
pendientes_aprobacion

FROM inmuebles;
```

Explicación:

- Usamos **COUNT** con **CASE WHEN** para contar de manera condicional
- **NOW() - INTERVAL '24 hours'** filtra los publicados en el último día
- **publicados_ultimas_24h** cuenta solo los inmuebles en estado 'publicado' en las últimas 24h
- **pendientes_aprobacion** cuenta todos los inmuebles con estado 'pendiente'

Versión alternativa con GROUP BY si se necesita más detalle:

```
SELECT

  estado,

  COUNT(*) as cantidad,

  SUM(CASE WHEN fecha_publicacion >= NOW() - INTERVAL '24 hours' THEN
1 ELSE 0 END) as ultimas_24h

FROM inmuebles

GROUP BY estado;
```