[ACT] S - 7. Comunicación asíncrona accesible

UNIVERSIDAD TECNOLÓGICA DE TEHUACÁN

NOMBRE DEL ALUMNO:

- Marco Antonio Aguilar Castillo
- Cleber Antonio Bolaños Moreno
- Osbaldo Alvarez Martinez
- Kevin Ricardo Simon Alfaro
- Samuel Jonathan Trujillo Bolaños

MATERIA:

- Desarrollo Web Profesional

PROFESOR:

- José Miguel Carrera Pacheco

GRADO Y GRUPO:

8° “B”

Miembro	Rol	Actividades Detalladas (Semana 7)	Entregable Clave
Osbaldo Alvarez	**Tech Lead (TL)**	Implementación del componente global `ErrorBoundary`, definición de la estrategia de reintentos (retry logic) para peticiones fallidas y revisión técnica de todos los Pull Requests (PRs).	**ErrorBoundary & Manual de Resiliencia**
Marco Antonio Aguilar	**Backend (BE)**	Desarrollo de API REST con Express, endpoints de autenticación JWT y empleos, middleware de manejo de errores centralizado y validación de esquemas.	**API REST & Sistema de Auth**
Cleber Antonio Bolaños	**Frontend (FE)**	Configuración de instancia de Axios, desarrollo del hook personalizado `useAsync` para estados de carga, maquetación de la Home y Dashboard con componentes accesibles.	**UI Funcional & Consumo de API**
Samuel J. Trujillo	**QA (Testing)**	Pruebas de funcionamiento, revisión de la navegación por teclado y creación de reportes de errores con evidencias.	**Reportes de Errores**
Kevin Ricardo Simon	**DevOps (DO)**	Configuración y ejecución del despliegue del proyecto, manejo de variables de entorno y habilitación de accesos entre plataformas.	

Contenido

2. INVESTIGACIÓN PROFUNDA

2.1 JavaScript Asíncrono (Async JS)

La asincronía permite que la aplicación no se bloquee mientras espera datos externos.

- **Promises:** Representan valores futuros con estados *Pending*, *Fulfilled* o *Rejected*.
- **Async/Await:** Sintaxis moderna que mejora la legibilidad y facilita el uso de bloques `try/catch`.
- **AbortController:** Permite cancelar peticiones si el usuario cambia de vista, optimizando recursos.

2.2 APIs REST y Consumo de Datos

- **Métodos:** GET (Lectura), POST (Creación), PUT/PATCH (Actualización) y DELETE (Borrado).
- **Fetch API vs Axios:** Mientras Fetch es nativo, Axios ofrece interceptores y manejo automático de JSON.
- **Validación:** Es crucial validar `response.ok` para manejar errores lógicos del servidor.

2.3 Estados de Carga Accesibles (UX Inclusiva)

- **Aria-live:** `polite` para anuncios no intrusivos y `assertive` para errores críticos.
- **Roles:** `role="status"` para mensajes de estado y `aria-busy="true"` para indicar carga en curso.
- **Skeletons:** Mejoran la percepción de velocidad al mostrar la estructura antes que los datos.

2.4 Manejo de Errores y Resiliencia

- **Categorías:** Errores de red (sin internet), de aplicación (4xx/5xx) y Timeouts.
- **Estrategias:** Botón de "Reintentar", mensajes humanizados y almacenamiento local temporal (localStorage).

3. IMPLEMENTACIÓN TÉCNICA

3.1 Arquitectura de Consumo de API

Se configuró una instancia centralizada de Axios para gestionar la comunicación:

```
// lib/api.js
import axios from 'axios';

const api = axios.create({
  baseURL: process.env.NEXT_PUBLIC_API_URL || 'http://localhost:3000/api',
  timeout: 10000,
  headers: { 'Content-Type': 'application/json' }
});

api.interceptors.request.use((config) => {
  const token = localStorage.getItem('token');
  if (token) config.headers.Authorization = `Bearer ${token}`;
  return config;
});
```

3.2 Hook Personalizado useAsync

Centraliza la lógica de estados asíncronos con accesibilidad integrada:

- **loading:** Activa aria-busy.
- **error:** Lanza alertas auditivas mediante regiones live.

3.3 Componente ErrorBoundary

Captura errores de renderizado de React:

- **Accesibilidad:** Uso de role="alert" y foco automático en el botón de recuperación.

4. ENDPOINTS DE API (BACKEND)

Desarrollado con **Express** y **MySQL**:

1. **Autenticación:**
    - POST /api/auth/register: Registro con hash bcrypt.
    - POST /api/auth/login: Generación de JWT .
2. **Empleos:**
    - GET /api/jobs: Listado paginado con filtros de ubicación y salario.
    - POST /api/jobs: Creación de vacantes (Reclutadores).
    - GET /api/jobs/:id: Detalle extendido.

5. ESTADOS DE CARGA ACCESIBLES

Regiones Live Implementadas

```
<div
  role="status"
  aria-live="polite"
  aria-atomic="true"
  className="sr-only"
>
  {loading && "Cargando vacantes disponibles..."}
  {error && `Error: ${error.message}`}
  {data && `Se encontraron ${data.length} vacantes`}
</div>
```

6. EVIDENCIAS DE PRUEBAS (QA)

Prueba	Resultado	Herramienta
Registro/Login	Exitoso	Postman
Navegación Teclado	100% Funcional	Teclado físico

7. DEPLOY PARCIAL

- **Frontend:** https://hubmar.fun

8. CONCLUSIONES Y LECCIONES APRENDIDAS

El equipo logró consolidar la comunicación asíncrona garantizando que la aplicación sea usable bajo condiciones de red inestables. Aprendimos que el manejo de errores no es solo técnico, sino una parte fundamental de la experiencia de usuario (UX). El uso de atributos ARIA y regiones live permite que **Conecta Tehuacán** sea un sistema inclusivo y profesional.