Prueba de habilidades

Alcance de la prueba:

Validar las habilidades y destrezas técnicas de los recursos de desarrollo.

Esta prueba comprende dos partes: FrontEnd y Backend.

Prueba de FrontEnd:

Alcance:

Crear una aplicación del tipo SPA que permita gestionar (Agregar / Actualizar / Eliminar) pokemons de un listado de favoritos.

Implementar el api de pokeapi la cual retorna las informaciones de los pokemons (solo se requiere implementar el api de listado de pokemons, se dará el detalle en los requerimientos técnicos).

Documentación oficial api de pokeApi https://pokeapi.co/

Requisitos funcionales:

1. Poder ver la lista paginada de Pokemons

   Notas:
   - En los requisitos técnicos se les entrega el api que retorna el listado paginado

   - Desplegar campos de nombre en el listado.

   - Agregar botón Agregar a Favoritos

   - Agregar paginación para permitir al usuario navegar entre las paginas

2. Poder agregar a favoritos los pokemons

   Notas:

3. Poder eliminar cualquier pokemon del listado de favoritos

4. Poder cambiar el alias al pokemon guardado en favoritos

Requisitos técnicos:

1. Usar Framework de angular 12.x con TypeScript https://angular.io/guide/setup-local

2. Para maquetar las pantallas usar el Framework de bootstrap (Introduction · Bootstrap v4.6 (getbootstrap.com)) para manejar los estilos.

3. Implementar tres componentes / Pantallas:

- Listado de pokemons paginado
- Listado de Favoritos
- Formulario de edición de favoritos

4. Usar validaciones para los campos del formulario todos los campos son requeridos.

Debe desplegar un mensaje al usuario si existen campos no completados o inválidos.

El único campo editable es el alias.

5. Implementar api listado de pokemons GET https://pokeapi.co/api/v2/pokemon?limit=10&offset=0

NOTA:
En el adjunto se agrega la colección de PostMan con el ejemplo de cómo consumirla

6. Guardar en la sesionStorage los datos de favoritos, se sugiere tener solo tres campos en el esquema:

```
interface favorito {
    name: string
    alias: string
    createdAt: Date
}


const favoritos: favorito[] = [
    {
        name: "bulbasaur",
        alias: "bulbasaur-Alidas",
        createdAt: new Date()
    }
]
```

6. Implementar mínimo dos rutas para desplegar las pantallas (Listado de Favoritos, Listado de Pokemons)

Prueba de BackEnd:

Alcance:

Crear una api que permita agregar y actualizar un listado de usuarios.

Requisitos Técnicos / Funcionales:

1. Crear api para el nuevo registro POST api/ v1/user

   Validar previamente por el correo del usuario si el mismo ya existe.

   Notas:
   - En caso de existir retornar error { success: false, data: null, message: “Este usuario ya está registrado” }

2. Crear api para actualizar un usuario PUT api/v1/user/{idUser: string}

   Notas:
   - En caso de no existir retornar error { success: false, data: null, message: “El usuario no existe” }

Requisitos técnicos:

1. Usar Framework de ExpressJs (https://expressjs.com/es/starter/hello-world.html) para construir las Apis (Se puede implementar con JavaScrip / TypeScript).

2. Implementar driver de mongo nativo o mongoose (`'mongodb' / mongoose`) para la fuente de datos. Se agrega a continuación los credenciales más ejemplo como se levanta la conexión usando mongodb.

```
// USER VERSION DE NODE 14.X en adelante
const express = require("express");
const { MongoClient } = require('mongodb');

(async () => {

    try {

        const dataBaseName = "develop";
        const uri =
`mongodb+srv://develop:develop@cluster0.q0i9x.mongodb.net/${dataBaseName}
?retryWrites=true&w=majority`;
        const client = new MongoClient(uri, { useNewUrlParser: true,
useUnifiedTopology: true });

        await client.connect();

        const db = client.db(dataBaseName);

        const app = express();

        const PORT = 3000;

        app.get('/', (req, res) => {
            res.status(201).json({ "develop": "Hello World" })
        })

        app.listen(PORT, () => {
            console.log(`Server listening in port ${PORT}`)
        });
    }
    catch (err) {
        console.error(err)
    }

})()
```

3. Implementar el siguiente modelo para el esquema el modelo de usuario (user)

```
interface user {
    _id: string
    createdDate: Date
    email: string
    firstName: string
    lastName: string
    password: string
    role?: string
}
```