**3214 – Master Desarrollo 2024 – 2025**

**PRACTICA OBLIGATORIA AZURE COSMOSBD**

Parte 1: Volcado de datos desde SQL Server (Management Studio) a Azure Cosmos DB

**Paso 1: Crear una cuenta de Cosmos DB en Azure**

1. Accede al portal de Azure.
2. Crea una nueva cuenta de **Azure Cosmos DB**.
   * Selecciona el modelo de API de **SQL (Core)**.
   * Configura el nombre, la región, y otras opciones de la cuenta.
3. Una vez creada la cuenta, obtendrás el Endpoint y la Primary Key que necesitarás para acceder desde la aplicación.

**Paso 2: Migrar los datos:**

Exporta los datos de la tabla Categories a un archivo CSV o JSON y luego súbelo manualmente a Cosmos DB

Parte 2: Backend en .NET Core (API CRUD para Cosmos DB)

**Objetivo:**

Desarrollar una API RESTful en **.NET Core** que se conecte a **Azure Cosmos DB** para realizar operaciones CRUD sobre los datos de la tabla Categories de la base de datos Northwind.

**Especificaciones:**

1. **Conexión a Cosmos DB:**
   * Utilizar el **SDK de Cosmos DB** para .NET (paquete Microsoft.Azure.Cosmos).
   * La API debe poder conectarse a una cuenta de **Cosmos DB** proporcionada.
   * Configurar la conexión utilizando el **Endpoint** y la **Primary Key** de la cuenta de Cosmos DB que se proporcionará.
   * El nombre de la base de datos será Northwind y la colección de contenedores será Categories.
2. **Modelo de Datos:**

* Crear una clase **Category** en el backend, que represente la tabla Categories de la base de datos Northwind.
* La clase **Category** debe tener las siguientes propiedades:
  + CategoryID: Identificador único de la categoría (tipo string).
  + CategoryName: Nombre de la categoría (tipo string).
  + Description: Descripción de la categoría (tipo string).
  + Picture: Imagen asociada a la categoría (tipo string, en formato base64 o URL).

1. **Operaciones CRUD:**

* **GET** /api/category: Devuelve todas las categorías almacenadas en Cosmos DB.
* **GET** /api/category/{id}: Devuelve una categoría específica por su CategoryID.
* **POST** /api/category: Permite crear una nueva categoría en Cosmos DB. El cuerpo de la solicitud debe contener los datos necesarios para crear una categoría.
* **PUT** /api/category/{id}: Permite actualizar una categoría existente por su CategoryID.
* **DELETE** /api/category/{id}: Permite eliminar una categoría por su CategoryID.

1. **Respuesta de la API:**

 Las respuestas deben ser claras y adecuadas (códigos de estado HTTP correctos).

 Usar los códigos de respuesta estándar como:

* 200 OK: Para solicitudes exitosas.
* 201 Created: Para la creación exitosa de un recurso.
* 404 Not Found: Si el recurso no se encuentra.
* 400 Bad Request: Para solicitudes mal formadas.
* 500 Internal Server Error: Si ocurre un error en el servidor.

1. **Autenticación y Seguridad (Opcional, pero recomendado):**

* Implementar alguna capa de seguridad básica, como la autenticación por **token JWT** o **API Key** si se requiere.

Parte 3: Frontend en React (Consumir la API)

**Objetivo:**

Desarrollar una aplicación en **React** que consuma la API creada en la Parte 2, permitiendo interactuar con las categorías en Cosmos DB a través de un frontend.

**Especificaciones:**

1. **Configuración del Proyecto React:**
   * Utilizar **Vite**  para crear la estructura del proyecto.
   * La aplicación debe estar configurada para hacer peticiones HTTP a la API del backend (que está corriendo en **.NET Core**).
2. **Comunicarse con la API:**
   * Utilizar **Axios** para hacer las peticiones HTTP a los endpoints de la API que se han implementado en el backend.
   * Realizar las siguientes operaciones desde el frontend:
     + **GET**: Obtener todas las categorías y mostrarlas en una lista.
     + **POST**: Crear una nueva categoría, enviando un formulario con los datos necesarios (como CategoryName, Description, etc.).
     + **PUT**: Permitir la edición de una categoría existente, mostrando un formulario con los datos actuales y permitiendo actualizarlos.
     + **DELETE**: Permitir eliminar una categoría.
3. **Interfaz de Usuario (UI):**
   * Mostrar una lista de todas las categorías obtenidas de la API en una tabla o lista con sus respectivos detalles.
   * Implementar formularios de entrada para agregar o editar categorías con campos como:
     + CategoryName (Texto)
     + Description (Texto largo)
     + Picture (Campo de imagen; puede ser un enlace o código base64)
   * Incluir botones para realizar las operaciones de **crear**, **editar**, **eliminar** y **ver detalles** de cada categoría.
   * Utilizar **estilos CSS** para que la interfaz sea simple y funcional. Se puede utilizar alguna librería de estilos como **Bootstrap**.
4. **Gestión del Estado:**
   * Utilizar el **estado local** de React (useState) para gestionar los datos de las categorías en la aplicación.
   * Asegurarse de que la lista de categorías se actualice automáticamente después de realizar cualquier operación CRUD.
5. **Validaciones:**
   * Validar los formularios para asegurarse de que los campos no estén vacíos y contengan valores correctos antes de enviar las solicitudes a la API.
6. **Manejo de Errores:**
   * Mostrar mensajes de error si alguna operación de la API falla (por ejemplo, si no se encuentra una categoría al intentar editarla o eliminarla).
   * Mostrar un mensaje de éxito cuando se realice una operación CRUD correctamente.
7. **Funcionalidades Extras (Opcionales):**
   * Implementar **paginación** si el número de categorías es grande.
   * Añadir un **sistema de búsqueda** para filtrar las categorías por nombre o descripción.
   * Implementar una pequeña **validación de imagen** para asegurarse de que el formato de la imagen sea válido.