Taller de Lenguajes II – 2025
Programador Universitario / Licenciatura en informática / Ingeniería en Informática
TP Nro 7

CONTENIDOS

- Base de Datos SQLite
- Sentencias SQL
- ADO Net
- Repositorios

INTRODUCCIÓN

Copie el siguiente enlace en su navegador: https://tinyurl.com/TL2-TP7-2025 esto creará el repositorio para poder subir el **Trabajo Práctico Nro. 7**.

Continuaremos con la implementación del generador de presupuestos de productos que será utilizado como prototipo para la página web de una Distribuidora de Insumos Informáticos.

Cree un nuevo proyecto Web Api

- Para ello use los siguientes comando
  - **dotnet new webapi --use-controllers**
- Incorpore en el presente proyecto la db "**tienda.db**" creada en el Trabajo Práctico Nro. 6.
- Instale dependencia para utilizar SQLite con ADO.Net, para ello ejecute el siguiente comando:
  - **dotnet add package Microsoft.Data.SQLite**

2) **Creando las clases modelo:**

Crear una carpeta llamada Models donde ubicará los modelos de su aplicación. A continuación se detallan los modelos que deben crear:

- **Productos**
  - int idProducto
  - string descripcion
  - int precio
- **Presupuestos**
  - int IdPresupuesto
  - string nombreDestinatario
  - date FechaCreacion
  - List<PresupuestoDetalle> detalle
  - Metodos
    - MontoPresupuesto ()
    - MontoPresupuestoConIva() //considerar iva 21
    - CantidadProductos () //contar total de productos (sumador de todas las cantidades del detalle)

- **PresupuestosDetalle**
  - Productos producto
  - int cantidad

3) **Creando repositorios:**

Para una mejor organización y separación de responsabilidades en su API en .NET, es recomendable utilizar el Patron Repositorio para gestionar las operaciones de acceso a la base de datos. Cree una carpeta llamada *Repositorios*, donde ubicará los repositorios a construir.

A continuación se detallan los repositorios y los métodos que deben crear:

Repositorio de Productos:

Crear un repositorio llamado ProductoRepository para gestionar todas las operaciones relacionadas con Productos. Este repositorio debe incluir métodos para:

- Crear un nuevo Producto. (recibe un objeto Producto)
- Modificar un Producto existente. (recibe un Id y un objeto Producto)
- Listar todos los Productos registrados. (devuelve un List de Producto)
- Obtener detalles de un Productos por su ID. (recibe un Id y devuelve un Producto)
- Eliminar un Producto por ID

Repositorio de Presupuestos:

Crear un repositorio llamado PresupuestosRepository para gestionar todas las operaciones relacionadas con Presupuestos. Este repositorio debe incluir métodos para:

- Crear un nuevo Presupuesto. (recibe un objeto Presupuesto)
- Listar todos los Presupuestos registrados. (devuelve un List de Presupuestos)
- Obtener detalles de un Presupuesto por su ID. (recibe un Id y devuelve un Presupuesto con sus productos y cantidades)
- Agregar un producto y una cantidad a un presupuesto (recibe un Id)
- Eliminar un Presupuesto por ID

4) Creando EndPoints:

A continuación, vamos a detallar los controladores y los distintos endpoints que se deberán agregar al sistema:

Crear un Controlador de Producto(ProductoController) que incluya los endpoints para:

- **POST** /api/Producto: Permite crear un nuevo Producto.
- **PUT** /api/Producto/{Id}: Permite modificar un nombre de un Producto.
- **GET** /api/Producto: Permite listar los Productos existentes.
- **GET** /api/Producto/{Id}: Obtener detalles de un Productos por su ID.
- **DELETE** /api/Producto: Eliminar un Producto por ID

Taller de Lenguajes II – 2025
Programador Universitario / Licenciatura en informática / Ingeniería en Informática
TP Nro 7
Crear un Controlador de Presupuestos (PresupuestosController) que incluya los endpoints para:

- **POST** /api/Presupuesto: Permite crear un Presupuesto.
- **POST** /api/Presupuesto/{id}/ProductoDetalle: Permite agregar un Producto existente y una cantidad al presupuesto.
- **GET** /api/Presupuesto/{id}: Obtener detalles de un Presupuesto por su ID.
- **GET** /api/presupuesto: Permite listar los presupuestos existentes.
- **DETELE** /api/Presupuesto/{id}: Permite eliminar un Presupuesto.

Nota:

La Estructura del Proyecto tendría que quedar de la siguiente manera

- /Models
  - Productos.cs
  - Presupuestos.cs
  - PresupuestosDetalle.cs
- /Repositorios
  - ProductosRepository.cs
  - PresupuestosRepository.cs
- /Controllers
  - ProductosController.cs
  - PresupuestosController.cs

Ej. Implementación de controller utilizando el patrón Repository

```
[ApiController]
[Route("[controller]")]
public class ProductosController: ControllerBase
{
    private ProductoRepository productoRepository;
    public ProductosController()
    {
        productoRepository= new ProductoRepository();
    }
//A partir de aquí van todos los Action Methods (Get, Post,etc.)
}
```

Ejemplo de cómo dar de alta un producto

```
 [HttpPost("AltaProducto")]
   public ActionResult<string> AltaProducto(Producto nuevoProducto)
   {
        productoRepository.Alta(nuevoProducto);
        return Ok("Producto dado de alta exitosamente");
   }
```

Taller de Lenguajes II – 2025
Programador Universitario / Licenciatura en informática / Ingeniería en Informática
TP Nro 7

Ejemplo de cómo listar los productos

```
  [HttpGet("Productos")]
   public ActionResult<List<Producto>> GetProductos()
   {
       List<Producto> listProductos;
       listProductos=productoRepository.GetAll();
       return Ok(listProductos);
   }
```

Ejemplo de cómo eliminar un producto

```
  [HttpDelete("{id}")]
   public ActionResult DeleteProducto(int id)
   {
      bool eliminado = productoRepository.Eliminar(id);
      if(eliminado)
      {
       return NoContent();// HTTP 204 (Eliminación exitosa)
      }
      else
      {
       return NotFound($"No se encontró el producto con ID {id} para
 eliminar.");
      }
   }
```