



Universidad Autónoma de Aguascalientes

Proyecto Final: Productos de una Taquería

Carrera:

Informática y Tecnologías Computacionales

Materia:

Desarrollo Web

Alumno:

Noel Alonso De Luna

Osmand Christian Gutiérrez Juárez

Tomas Abraham Hernández Rojas

Jesús Alfredo Pinedo Macías

Elías Alain Vital Delgadillo

Profesor:

Margarita Mondragón Arellano

Fecha:

26/Junio/2025

Índice

Documentación Técnica: Altas.....	3
Documentación Técnica: Bajas	9
Documentación Técnica: Actualizaciones	19
Documentación Técnica: Consultas	22
Creación e Instalación del Proyecto	30
- Configuración de Proyecto en “Blazor”	30
- Configuración del Proyecto en SQL Server	33
- Configuración del Diseño	35

Documentación Técnica: Altas

```
@page "/alta"  
@using Proyecto_Taqueria.Data.Modelos  
@using Proyecto_Taqueria.Data  
@using Microsoft.EntityFrameworkCore  
@using System.ComponentModel.DataAnnotations  
@inject ApplicationDbContext Contexto  
@rendermode InteractiveServer
```

Esta es la primera parte que establece la ruta de la página, importando las clases y librerías necesarias, inyectando el contexto de la base de datos y definiendo cómo se renderizará el componente para permitir la interacción del usuario.

@page "/alta"

Ruta que muestra el componente cuando visitas /alta.

@using Proyecto_Taqueria.Data.Modelos

Importa la clase Producto para usarla en la pagina.

@using Proyecto_Taqueria.Data

Permite usar el ApplicationDbContext que conecta con la base de datos.

@using Microsoft.EntityFrameworkCore

Importa funcionalidades de Entity Framework Core que se utiliza para interactuar con la base de datos para guardar, buscar o eliminar registros

@using System.ComponentModel.DataAnnotations

Importa atributos para la validación de datos como [Required] y [Range] que se usan en el modelo Producto.

@inject ApplicationDbContext Contexto

permite realizar operaciones de base de datos como agregar o consultar productos

@rendermode InteractiveServer

Es para manejar interacciones del usuario y actualizar la UI en el navegador, lo que permite interactividad en tiempo real.

```

EditForm Model="@nuevoProducto" OnValidSubmit="@ManejarEnvioFormulario">
  <DataAnnotationsValidator />
  <ValidationSummary />

  <div class="form-group row mb-2">
    <label for="productoId" class="col-sm-3 col-form-label">ID Producto:</label>
    <div class="col-sm-9">
      <InputText id="productoId" @bind-Value="idMostradoEnFormulario" class="form-control" readonly />
      <small class="form-text text-muted">Este ID se asigna automáticamente al guardar el producto.</small>
    </div>
  </div>

  <div class="form-group row mb-2">
    <label for="nombreProducto" class="col-sm-3 col-form-label">Nombre:</label>
    <div class="col-sm-9">
      <InputText id="nombreProducto" @bind-Value="nuevoProducto.Nombre" class="form-control" />
      <ValidationMessage For="@(( ) => nuevoProducto.Nombre)" />
    </div>
  </div>

  <div class="form-group row mb-2">
    <label for="categoriaProducto" class="col-sm-3 col-form-label">Categoría:</label>
    <div class="col-sm-9">
      <InputText id="categoriaProducto" @bind-Value="nuevoProducto.Categoria" class="form-control" />
      <ValidationMessage For="@(( ) => nuevoProducto.Categoria)" />
    </div>
  </div>

  <div class="form-group row mb-2">
    <label for="cantidadStock" class="col-sm-3 col-form-label">Cantidad en stock:</label>
    <div class="col-sm-9">
      <InputNumber id="cantidadStock" @bind-Value="nuevoProducto.CantidadStock" class="form-control" />
      <ValidationMessage For="@(( ) => nuevoProducto.CantidadStock)" />
    </div>
  </div>
</div>

```

EditForm Model="@nuevoProducto"
OnValidSubmit="@ManejarEnvioFormulario"

Crea un formulario ligado a nuevoProducto. Si pasa la validación, ejecuta ManejarEnvioFormulario().

<DataAnnotationsValidator />

Activa la validación automática con base en los atributos [Required], [Range], etc que estan definidos en el modelo.

<ValidationSummary />

Muestra todos los errores de validación juntos para que el usuario los vea fácilmente.

<label for="productoId">

Etiqueta visible asociada al campo respectivo.

<InputText @bind-Value="idMostradoEnFormulario" readonly />

Muestra el ID del producto. Está vinculado a la variable idMostradoEnFormulario y es de solo lectura en caso de que tenga readonly.

@bind-Value="nuevoProducto.Nombre"

enlaza el contenido del campo con la propiedad del campo del objeto nuevoProducto

<ValidationMessage For="@(() => nuevoProducto.Nombre)" />

Escucha los errores de validación asociados a la propiedad campo del modelo nuevoProducto y los muestra justo debajo del campo.

```
@if (!string.IsNullOrEmpty(mensajeFeedback))
{
    <div class="@mensajeCssClass mt-3" role="alert">
        @mensajeFeedback
    </div>
}
```

@if (!string.IsNullOrEmpty(mensajeFeedback))

Se ejecuta solo si mensajeFeedback tiene texto.

MensajeFeedback

Contiene el mensaje que se quiere mostrar por si se guarda un producto con éxito, si ocurre un error o si no se encuentran productos.

```
<h3>Productos Existentes</h3>

@if (productos == null)
{
    <p><em>Cargando productos...</em></p>
}
else if (!productos.Any())
{
    <p>No hay productos registrados.</p>
}
else
{
    <table class="table table-striped table-bordered mt-3">
        <thead class="thead-dark">
            <tr>
                <th>ID</th>
                <th>Nombre</th>
                <th>Categoria</th>
                <th>Stock</th>
                <th>Stock Mín.</th>
                <th>Stock Máx.</th>
                <th>Fecha Caducidad</th>
                <th>Fecha Llegada</th>
            </tr>
        </thead>
        <tbody>
            @foreach (var producto in productos)
            {
                <tr>
                    <td>@producto.Id</td>
                    <td>@producto.Nombre</td>
                    <td>@producto.Categoria</td>
                    <td>@producto.CantidadStock</td>
                    <td>@producto.MinimoStock</td>
                    <td>@producto.MaximoStock</td>
                    <td>@(producto.FechaCaducidad.HasValue ? producto.FechaCaducidad.Value.ToShortDateString() : "N/A")</td>
                    <td>@(producto.FechaLlegada.HasValue ? producto.FechaLlegada.Value.ToShortDateString() : "N/A")</td>
                </tr>
            }
        </tbody>
    </table>
}
```

@if (productos == null)

Los datos aún no llegan, muestra un mensaje de cargando productos.

else if (!productos.Any())

La lista llegó pero está vacía, muestra un mensaje de No hay productos registrados.

else

Mostrará la tabla en caso que las anteriores sean falsas

<table class="table table-striped table-bordered mt-3">

Crea una tabla

```
<tr>
  <th>ID</th>
  <th>Nombre</th>
  <th>Categoría</th>
  <th>Stock</th>
  <th>Fecha Llegada</th>
  <th>Fecha Caducidad</th>
</tr>
```

Tr crea filas y th columnas donde esta cada campo

@foreach (var producto in productos)

Este es un bucle que recorre la lista productos uno por uno en donde imprimirá los siguientes valores

```
<tr>
  <td>@producto.Id</td>
  <td>@producto.Nombre</td>
  <td>@producto.Categoria</td>
  <td>@producto.CantidadStock</td>
  <td>@producto.MinimoStock</td>
  <td>@producto.MaximoStock</td>
  <td>@(producto.FechaCaducidad.HasValue ?
producto.FechaCaducidad.Value.ToShortDateString() : "N/A")</td>
  <td>@(producto.FechaLlegada.HasValue ?
producto.FechaLlegada.Value.ToShortDateString() : "N/A")</td>
</tr>
```

HasValue

indica si ese campo contiene una fecha real.

ToShortDateString

se convierte a formato corto (dd/mm/yyyy)

```

@code {
    private Producto nuevoProducto = new Producto();
    private List<Producto>? productos;
    private string mensajeFeedback = string.Empty;
    private string mensajeCssClass = string.Empty;
    private string idMostradoEnFormulario = string.Empty;

    protected override async Task OnInitializedAsync()
    {
        await CargarProductos();
        idMostradoEnFormulario = string.Empty;
    }

    private async Task CargarProductos()
    {
        try
        {
            productos = await Contexto.Productos.OrderByDescending(p => p.Id).ToListAsync();
        }
        catch (Exception ex)
        {
            mensajeFeedback = $"Error al cargar productos: {ex.Message}";
            mensajeCssClass = "alert alert-danger";
            Console.WriteLine($"Error al cargar productos: {ex.Message}");
        }
    }

    private async Task ManejarEnvioFormulario()
    {
        mensajeFeedback = string.Empty;
        mensajeCssClass = string.Empty;

        await GuardarProducto();
    }

    private async Task GuardarProducto()
    {
        try
        {
            Contexto.Productos.Add(nuevoProducto);
            await Contexto.SaveChangesAsync();

            int idGenerado = nuevoProducto.Id;

            idMostradoEnFormulario = idGenerado.ToString();

            private async Task GuardarProducto()
            {
                try
                {
                    Contexto.Productos.Add(nuevoProducto);
                    await Contexto.SaveChangesAsync();

                    int idGenerado = nuevoProducto.Id;

                    idMostradoEnFormulario = idGenerado.ToString();
                    mensajeFeedback = $"Producto '{nuevoProducto.Nombre}' guardado correctamente. ID: {idGenerado}.";
                    mensajeCssClass = "alert alert-success";
                    Console.WriteLine($"Producto '{nuevoProducto.Nombre}' guardado correctamente. ID: {idGenerado}.");

                    nuevoProducto = new Producto();
                    idMostradoEnFormulario = string.Empty;
                    await CargarProductos();
                }
                catch (Exception ex)
                {
                    mensajeFeedback = $"Error al guardar el producto: {ex.Message}";
                    mensajeCssClass = "alert alert-danger";
                    Console.WriteLine($"Error al guardar: {ex.Message}");
                }
            }
        }
    }
}

```

private Producto nuevoProducto = new Producto();

espacio en memoria para que el formulario almacene temporalmente los datos que el usuario ingresa para un nuevo producto.

private List<Producto>? productos;

espacio en memoria con la lista cargada desde la base de datos para mostrar en la tabla.

private string mensajeFeedback = string.Empty;

espacio en memoria con texto dinámico mostrado como mensaje de éxito, error o ningún resultado.

private string mensajeCssClass = string.Empty;

espacio en memoria con clases css para estilizar el mensaje.

private string idMostradoEnFormulario = string.Empty;

espacio en memoria que guarda el ID del producto recién registrado.

await CargarProductos();

Carga la lista de productos desde la base de datos para mostrarlos en la tabla.

idMostradoEnFormulario = string.Empty;

Deja el campo del ID del producto en el formulario en blanco al inicio.

private async Task CargarProductos()

Este método se encarga de traer los productos de la base de datos.

**productos = await Contexto.Productos.OrderByDescending(p =>
p.Id).ToListAsync();**

Consulta la base de datos para obtener todos los productos, ordenándolos del más reciente al más antiguo por su ID, y los guarda en la variable productos.

private async Task ManejarEnvioFormulario()

Este método se activa cuando el usuario envía el formulario y todos los datos son correctos.

mensajeFeedback = string.Empty; y mensajeCssClass = string.Empty;

Limpia cualquier mensaje de anterior.

await GuardarProducto();

Llama a GuardarProducto() para iniciar el proceso de guardar el nuevo producto en la base de datos.

private async Task GuardarProducto()

proceso de guardar un nuevo producto.

Contexto.Productos.Add(nuevoProducto);

Prepara el nuevo producto para ser añadido a la base de datos.

await Contexto.SaveChangesAsync();

Envía y guarda el nuevo producto en la base de datos y se le asigna automáticamente un ID.

int idGenerado = nuevoProducto.Id;

Obtiene el ID que la base de datos acaba de asignar al producto guardado.

idMostradoEnFormulario = idGenerado.ToString();

Muestra el ID recién generado en el campo correspondiente del formulario.

mensajeFeedback = \$"Producto '{nuevoProducto.Nombre}' guardado correctamente. ID: {idGenerado}.";

Informa al usuario que el producto se guardó con éxito y registra la acción.

nuevoProducto = new Producto();

Crea un nuevo objeto Producto vacío, restableciendo así el formulario para que el usuario pueda ingresar el siguiente producto.

idMostradoEnFormulario = string.Empty;

Asegura que el campo de ID en el formulario esté vacío para la siguiente entrada.

await CargarProductos();

Vuelve a cargar la lista de productos de la base de datos para que el producto recién añadido aparezca en la tabla.

Documentación Técnica: Bajas

```
@page "/baja"
@using Proyecto-Taqueria.Data.Modelos
@using Proyecto-Taqueria.Data
@using Microsoft.EntityFrameworkCore
@inject ApplicationDbContext Contexto
@rendermode InteractiveServer
```

@page "/baja"

Ruta que muestra el componente cuando visitas /baja.

@using Proyecto-Taqueria.Data.Modelos

Importa la clase Producto para usarla en la pagina.

@using Proyecto-Taqueria.Data

Permite usar el ApplicationDbContext que conecta con la base de datos.

@using Microsoft.EntityFrameworkCore

Importa funcionalidades de Entity Framework Core que se utiliza para interactuar con la base de datos para guardar, buscar o eliminar registros.

@inject ApplicationDbContext Contexto

permite realizar operaciones de base de datos como agregar o consultar productos.

@rendermode InteractiveServer

Es para manejar interacciones del usuario y actualizar la UI en el navegador, lo que permite interactividad en tiempo real.

```
<h3>Baja de Producto</h3>

<div class="mb-3">
  <label for="buscarProducto" class="form-label">Buscar Producto por ID o Nombre:</label>
  <div class="input-group">
    <input type="text" id="buscarProducto" @bind-Value="terminoBusqueda" class="form-control" placeholder="ID o Nombre del producto" />
    <button class="btn btn-outline-secondary" type="button" @onclick="BuscarProductos">Buscar</button>
  </div>
</div>
```

<label for="buscarProducto">

Etiqueta visible asociada al campo respectivo.

<input type="text" id="buscarProducto" @bind-Value="terminoBusqueda" class="form-control" placeholder="ID o Nombre del producto" />

Campo de entrada de texto para que el usuario escriba el ID o nombre.

@bind-Value="terminoBusqueda"

enlaza el texto con la variable terminoBusqueda que se actualiza en tiempo real.

Placeholder

texto guía dentro del campo en donde dice "ID o Nombre del producto".

```
@if (!string.IsNullOrEmpty(mensajeFeedback))
{
  <div class="@mensajeCssClass mt-3" role="alert">
    @mensajeFeedback
  </div>
}
```

@if (!string.IsNullOrEmpty(mensajeFeedback))

Se ejecuta solo si mensajeFeedback tiene texto.

MensajeFeedback

Contiene el mensaje que se quiere mostrar por si se guarda un producto con éxito, si ocurre un error o si no se encuentran productos.

```

@if (productosEncontrados != null && productosEncontrados.Any())
{
    <h4 class="mt-4">Resultados de la Búsqueda</h4>
    <table class="table table-striped table-bordered mt-2">
        <thead class="thead-dark">
            <tr>
                <th>ID</th>
                <th>Nombre</th>
                <th>Categoría</th>
                <th>Stock</th>
                <th>Fecha Caducidad</th>
                <th>Fecha Llegada</th>
                <th>Acciones</th>
            </tr>
        </thead>
        <tbody>
            @foreach (var producto in productosEncontrados)
            {
                <tr>
                    <td>@producto.Id</td>
                    <td>@producto.Nombre</td>
                    <td>@producto.Categoría</td>
                    <td>@producto.CantidadStock</td>
                    <td>@(producto.FechaCaducidad.HasValue ? producto.FechaCaducidad.Value.ToShortDateString() : "N/A")</td>
                    <td>@(producto.FechaLlegada.HasValue ? producto.FechaLlegada.Value.ToShortDateString() : "N/A")</td>
                    <td>
                        <button class="btn btn-danger btn-sm" @onClick="() => ConfirmarEliminacion(producto)">Eliminar</button>
                    </td>
                </tr>
            }
        </tbody>
    </table>
}
else if (!string.IsNullOrEmpty(terminoBusqueda) && productosEncontrados != null && !productosEncontrados.Any())
{
    <div class="alert alert-info mt-3" role="alert">
        No se encontraron productos que coincidan con "@terminoBusqueda".
    </div>
}

```

@if (productosEncontrados != null && productosEncontrados.Any())

verifica si la variable productosEncontrados (la lista de productos hallados por la búsqueda) no es nula y contiene al menos un elemento.

<table class="table table-striped table-bordered mt-2">

Crea una tabla

```

<tr>
    <th>ID</th>
    <th>Nombre</th>
    <th>Categoría</th>
    <th>Stock</th>
    <th>Fecha Llegada</th>
    <th>Fecha Caducidad</th>
    <th>Acciones</th>
</tr>

```

Tr crea filas y th columnas donde está cada campo

@foreach (var producto in productosEncontrados)

Este es un bucle que recorre la lista productosEncotrados uno por uno en donde imprimirá los siguientes valores

```

<tr>
    <td>@producto.Id</td>
    <td>@producto.Nombre</td>
    <td>@producto.Categoría</td>

```

```

<td>@producto.CantidadStock</td>
<td>@producto.MinimoStock</td>
<td>@producto.MaximoStock</td>
<td>@(producto.FechaCaducidad.HasValue ?
producto.FechaCaducidad.Value.ToShortDateString() : "N/A")</td>
<td>@(producto.FechaLlegada.HasValue ?
producto.FechaLlegada.Value.ToShortDateString() : "N/A")</td>
<td>
<button class="btn btn-danger btn-sm" @onclick="()" =>
ConfirmarEliminacion(producto)">Eliminar</button>
<td>
</tr>

```

HasValue

indica si ese campo contiene una fecha real.

ToShortDateString

se convierte a formato corto (dd/mm/yyyy)

@onclick="()" => ConfirmarEliminacion(producto)"

Dentro de la columna "Acciones", se crea un botón "Eliminar" Al hacer clic se llama al método ConfirmarEliminacion() pasándole el producto específico de esa fila para su posible eliminación.

else if (!string.IsNullOrEmpty(terminoBusqueda) && productosEncontrados != null && !productosEncontrados.Any())

se ejecuta si el usuario realizó una búsqueda (terminoBusqueda no está vacío), pero no se encontraron productos (productosEncontrados es una lista vacía).

<div class="alert alert-info mt-3" role="alert">

Muestra un mensaje que no hubo resultados para su búsqueda.

```

@if (mostrarConfirmacion && productoAEliminar != null)
{
    <div class="alert alert-warning mt-3">
        ¿Estás seguro de que deseas eliminar el producto **@productoAEliminar.Nombre (ID: @productoAEliminar.Id)**? Esta acción no se puede deshacer.
        <button class="btn btn-danger mx-2" @onclick="EliminarProducto">Sí, Eliminar</button>
        <button class="btn btn-secondary" @onclick="CancelarEliminacion">Cancelar</button>
    </div>
}

```

@if (mostrarConfirmacion && productoAEliminar != null)

Solo entra al bloque si hay un producto seleccionado y se activó la confirmación

<button class="btn btn-danger mx-2" @onclick="EliminarProducto">Sí, Eliminar</button>

Botón "Sí, Eliminar" que ejecuta EliminarProducto() para realizar la acción de eliminar.

<button class="btn btn-secondary"

@onclick="CancelarEliminacion">Cancelar</button>

Botón "Cancelar" que ejecuta CancelarEliminacion() lo permite al usuario cancelar la operación de eliminación,

```

@code {
    private string terminoBusqueda = string.Empty;
    private List<Producto>? productosEncontrados;
    private Producto? productoAEliminar;
    private string mensajeFeedback = string.Empty;
    private string mensajeCssClass = string.Empty;
    private bool mostrarConfirmacion = false;

    private async Task BuscarProductos()
    {
        mensajeFeedback = string.Empty;
        mensajeCssClass = string.Empty;
        productosEncontrados = null;
        productoAEliminar = null;
        mostrarConfirmacion = false;

        try
        {
            List<Producto> resultados = new List<Producto>();

            if (int.TryParse(terminoBusqueda, out int idBusqueda))
            {
                var productoPorId = await Contexto.Productos.FirstOrDefaultAsync(p => p.Id == idBusqueda);
                if (productoPorId != null)
                {
                    resultados.Add(productoPorId);
                }
            }

            if (!string.IsNullOrEmpty(terminoBusqueda))
            {
                var productosPorNombre = await Contexto.Productos
                    .Where(p => p.Nombre.ToLower().Contains(terminoBusqueda.ToLower()))
                    .ToListAsync();

                foreach (var prod in productosPorNombre)
                {
                    if (!resultados.Any(r => r.Id == prod.Id))
                    {
                        resultados.Add(prod);
                    }
                }
            }
        }
    }
}

```

```

        if (!resultados.Any())
        {
            mensajeFeedback = "No se encontraron productos que coincidan con su búsqueda.";
            mensajeCssClass = "alert alert-warning";
        }
        else
        {
            productosEncontrados = resultados.OrderByDescending(p => p.Id).ToList();
            mensajeFeedback = $"Se encontraron {productosEncontrados.Count} producto(s).";
            mensajeCssClass = "alert alert-info";
        }
    }
    catch (Exception ex)
    {
        mensajeFeedback = $"Error al buscar productos: {ex.Message}";
        mensajeCssClass = "alert alert-danger";
        Console.WriteLine($"Error al buscar: {ex.Message}");
    }
}

private void ConfirmarEliminacion(Producto producto)
{
    productoAEliminar = producto;
    mostrarConfirmacion = true;
    mensajeFeedback = string.Empty;
}

private async Task EliminarProducto()
{
    if (productoAEliminar == null)
    {
        mensajeFeedback = "No hay producto seleccionado para eliminar.";
        mensajeCssClass = "alert alert-danger";
        mostrarConfirmacion = false;
        return;
    }

    try
    {
        Contexto.Productos.Remove(productoAEliminar);
        await Contexto.SaveChangesAsync();

        mensajeFeedback = $"Producto '{productoAEliminar.Nombre}' (ID: {productoAEliminar.Id}) eliminado correctamente.";
        mensajeCssClass = "alert alert-success";
    }
    catch
    {
        productoAEliminar = null;
    }
}

```

```

        mostrarConfirmacion = false;
        await BuscarProductos();

        if (productosEncontrados == null || !productosEncontrados.Any())
        {
            terminoBusqueda = string.Empty;
        }
    }
    catch (Exception ex)
    {
        mensajeFeedback = $"Error al eliminar el producto: {ex.Message}";
        mensajeCssClass = "alert alert-danger";
        Console.WriteLine($"Error al eliminar: {ex.Message}");
    }
}

private void CancelarEliminacion()
{
    mostrarConfirmacion = false;
    productoAEliminar = null;
    mensajeFeedback = "Eliminación cancelada.";
    mensajeCssClass = "alert alert-info";
}
}
}

```

private string terminoBusqueda = string.Empty;

Espacio en memoria que guarda lo que el usuario escribe en el campo de búsqueda (por ID o nombre).

private List<Producto>? productosEncontrados;

Espacio en memoria con la lista filtrada de productos que coinciden con el término de búsqueda.

private Producto? productoAEliminar;

Espacio en memoria que guarda temporalmente el producto que el usuario desea eliminar.

private string mensajeFeedback = string.Empty;

Texto dinámico mostrado como retroalimentación después de buscar, eliminar o no encontrar resultados.

private string mensajeCssClass = string.Empty;

Clase CSS que define el estilo visual del mensaje

private bool mostrarConfirmacion = false;

indica si se debe mostrar el cuadro de confirmación para eliminar un producto.

private async Task BuscarProductos()

es encontrar productos en la base de datos que coincidan con la búsqueda del usuario

mensajeFeedback = string.Empty;

mensajeCssClass = string.Empty;

productosEncontrados = null;

productoAEliminar = null;

mostrarConfirmacion = false;

Reinicia el estado de los mensajes, la lista de resultados y la confirmación, limpiando cualquier estado anterior antes de una nueva búsqueda.

List<Producto> resultados = new List<Producto>();

Inicializa una lista temporal para almacenar los productos encontrados antes de asignarlos a productosEncontrados.

if (int.TryParse(terminoBusqueda, out int idBusqueda))

Intenta convertir el terminoBusqueda a un número entero si es un número, significa que el usuario intentó buscar por ID.

var productoPorId = await Contexto.Productos.FirstOrDefaultAsync(p => p.Id == idBusqueda);

Si se buscó por ID, este comando busca el primer producto en la base de datos cuyo Id coincida con idBusqueda.

```
if (productoPorId != null)
```

```
{  
    resultados.Add(productoPorId);  
}
```

Si se encuentra un producto por ID lo añade a la lista resultados

```
if (!string.IsNullOrEmpty(terminoBusqueda))
```

Verifica si el terminoBusqueda no está vacío o solo contiene espacios esto permite buscar también por nombre incluso si ya se buscó por ID.

```
var productosPorNombre = await Contexto.Productos
```

```
.Where(p => p.Nombre.ToLower().Contains(terminoBusqueda.ToLower()))  
.ToListAsync();
```

Busca productos cuyo Nombre contenga el terminoBusqueda (ignorando mayúsculas/minúsculas) y los obtiene de la base de datos.

```
foreach (var prod in productosPorNombre)
```

```
{  
    if (!resultados.Any(r => r.Id == prod.Id))  
    {  
        resultados.Add(prod);  
    }  
}
```

Recorre los productos encontrados por nombre si un producto de esta lista no ha sido ya añadido (evitando duplicados si se encontró por ID) lo añade a la lista resultados.

```
if (!resultados.Any())
```

```
{  
    mensajeFeedback = "No se encontraron productos que coincidan con su  
búsqueda.";   
    mensajeCssClass = "alert alert-warning";  
}
```


Si no se encontraron (!resultados.Any()) actualiza mensajeFeedback a "No se encontraron productos..." y mensajeCssClass a "alert alert-warning".

else

```
{  
    productosEncontrados = resultados.OrderByDescending(p => p.Id).ToList();  
    mensajeFeedback = $"Se encontraron {productosEncontrados.Count}  
producto(s).";  
    mensajeCssClass = "alert alert-info";  
}
```

Si se encontraron, asigna resultados (ordenados descendientemente por ID) a productosEncontrados actualiza mensajeFeedback con el número de productos y mensajeCssClass a "alert alert-info".

catch (Exception ex)

Captura cualquier error durante la búsqueda, establece un mensaje de error

private void ConfirmarEliminacion(Producto producto)

Este método se ejecuta cuando el usuario hace clic en el botón "Eliminar" de un producto específico en la tabla.

productoAEliminar = producto;

Asigna el producto que se pasó como argumento a la variable productoAEliminar guardando cuál producto se quiere borrar.

mostrarConfirmacion = true;

Establece esta variable a true para que el mensaje de confirmación de eliminación se muestre en la interfaz de usuario.

mensajeFeedback = string.Empty;

Borra cualquier mensaje anterior.

private async Task EliminarProducto()

Este método se ejecuta cuando el usuario hace clic en el botón "Sí, Eliminar" en el cuadro de confirmación.

if (productoAEliminar == null)

```
{  
    mensajeFeedback = "No hay producto seleccionado para eliminar.";
```

```
mensajeCssClass = "alert alert-danger";  
mostrarConfirmacion = false;  
return;  
}
```

Una verificación de seguridad para asegurar que hay un producto seleccionado para eliminar antes de proceder si no lo hay muestra un error y sale del método.

Contexto.Productos.Remove(productoAEliminar);

Marca el productoAEliminar para ser removido del contexto de la base de datos.

await Contexto.SaveChangesAsync();

Ejecuta la eliminación del producto en la base de datos.

```
mensajeFeedback = $"Producto '{productoAEliminar.Nombre}' (ID:  
{productoAEliminar.Id}) eliminado correctamente.";
```

Establece un mensaje de éxito para el usuario confirmando que el producto ha sido eliminado.

```
productoAEliminar = null; y mostrarConfirmacion = false;
```

Restablece las variables ocultando el cuadro de confirmación y limpiando el producto que se había seleccionado.

await BuscarProductos();

Vuelve a ejecutar la búsqueda (con el término actual) para actualizar la tabla de resultados mostrando la lista sin el producto recién eliminado.

```
if (productosEncontrados == null || !productosEncontrados.Any())
```

```
{  
    terminoBusqueda = string.Empty;  
}
```

Si después de la eliminación no quedan productos en la tabla de resultados de la búsqueda limpia el campo de búsqueda para una nueva búsqueda.

private void CancelarEliminacion()

Este método se ejecuta cuando el usuario hace clic en el botón "Cancelar" en el cuadro de confirmación.

```
mostrarConfirmacion = false;
```

Oculto el cuadro de confirmación.

productoAEliminar = null;

Limpia la variable del producto que iba a ser eliminado.

mensajeFeedback = "Eliminación cancelada.";(mensaje de eliminación cancelada)

Documentación Técnica: Actualizaciones

1. Objetivo

Permitir al usuario del sistema buscar productos existentes en la base de datos mediante un identificador único (ID) o por nombre parcial, y luego visualizar y editar sus atributos principales a través de un formulario interactivo validado.

2. Estructura general del componente

El archivo ``razor`` define el comportamiento y la interfaz de usuario del componente de actualización:

- ``@page "/actualizacion"``: Define la ruta asociada a esta vista.
- ``@inject ApplicationDbContext Contexto``: Permite el acceso al contexto de la base de datos.
- ``@using`` directivas: Importa modelos, clases auxiliares y validaciones.

3. Variables y propiedades clave

- ``terminoBusqueda``: contiene el texto ingresado por el usuario para buscar.
- ``productosEncontrados``: almacena los resultados de la búsqueda.
- ``productoAEditar``: contiene una instancia del producto seleccionado para editar.
- ``mensajeFeedback`` / ``mensajeCssClass``: gestionan los mensajes contextuales y su estilo visual.

4. Lógica del componente

4.1 Método ``BuscarProductos()``

Realiza la búsqueda de productos de la siguiente manera:

- Elimina los estados visuales y lógicos previos, incluyendo resultados y formularios abiertos.
- Si el término ingresado es numérico, intenta hacer búsqueda directa por ID.
- Ejecuta siempre una búsqueda parcial y sensible a mayúsculas/minúsculas por nombre.
- Combina ambos resultados sin duplicados.
- Ordena los resultados por ID descendente.
- Genera un mensaje para indicar si se encontraron coincidencias, o una advertencia si no hubo resultados.

Este método contempla el uso de excepciones para manejar fallos y errores del sistema.

> Comentario clave: se oculta el formulario de edición al iniciar una nueva búsqueda para garantizar consistencia visual.

4.2 Método `CargarProductoParaEdicion()`

Este método permite preparar el formulario de edición creando una ****copia local del producto****, evitando que las modificaciones interfieran directamente con la lista visualizada. Esto mantiene un flujo limpio y seguro para el usuario.

4.3 Método `ActualizarProducto()`

Procedimiento encargado de guardar los cambios:

- Valida si existe un producto cargado para edición.
- Busca el registro original en la base de datos.

- Actualiza sus atributos modificables.
- Guarda los cambios con `SaveChangesAsync()`.
- Maneja casos donde otro usuario haya modificado el mismo registro (`DbUpdateConcurrencyException`).
- Captura errores genéricos y proporciona retroalimentación clara al usuario.

> Comentario clave: es importante mantener actualizadas todas las propiedades editables del modelo.

4.4 Método `CancelarEdicion()`

Este método cancela la edición en curso. Oculta el formulario y presenta un mensaje de cancelación.

5. Interfaz de usuario

- **Campo de búsqueda**: permite ingresar ID o nombre del producto.
- **Botón de búsqueda**: ejecuta el método `BuscarProductos()`.
- **Mensajes contextuales**: brindan retroalimentación en función de los resultados o errores.
- **Tabla de resultados**: se muestra si hay coincidencias, con botón para editar por fila.
- **Formulario de edición**: aparece únicamente cuando se selecciona un producto. Emplea validaciones y campos obligatorios.

6. Validaciones y experiencia de usuario

- Las validaciones se implementan con `DataAnnotationsValidator` y `ValidationMessage`.
- No se permite enviar el formulario si hay errores.
- El campo ID es de solo lectura.

- El diseño es responsivo gracias a Bootstrap.
- El sistema es capaz de recuperar el estado tras actualizaciones.

7. Consideraciones adicionales

- La búsqueda puede realizarse solo por nombre, solo por ID, o por ambos.
- Los resultados ordenados por ID en orden descendente favorecen la edición de productos recientes.
- El componente es tolerante a errores y ofrece una experiencia clara e intuitiva.

Documentación Técnica: Consultas

1.- Directivas e Inyecciones

```
@page "/Consultas"  
@using Proyecto-Taqueria.Data.Modelos  
@using Proyecto-Taqueria.Data  
@using Microsoft.EntityFrameworkCore  
@inject ApplicationDbContext Contexto  
@rendermode InteractiveServer
```

- **@page "/Consultas":**
Propósito: Define la ruta URL para acceder a este componente Razor. Cuando la aplicación se ejecuta y un usuario navega a /consulta, este componente es el que se renderiza.
Detalles: Es una directiva fundamental en Razor Pages y Blazor que mapea una URL a un componente específico.
- **@using Proyecto-Taqueria.Data.Modelos:**
Propósito: Importa el espacio de nombres donde se define la clase Producto (el modelo de datos para los productos).

Detalles: Permite que el código Razor y C# en esta página acceda directamente a la clase `Producto` sin necesidad de calificarla completamente (ej., `Producto` en lugar de `Proyecto_Taqueria.Data.Modelos.Producto`).

- **@using Proyecto_Taqueria.Data:**

Propósito: Importa el espacio de nombres donde se define `AppDbContext`, la clase que representa el contexto de la base de datos de la aplicación.

Detalles: Facilita el acceso a `AppDbContext` y sus propiedades (como `Productos`) para interactuar con la base de datos.

- **@using Microsoft.EntityFrameworkCore:**

Propósito: Importa el espacio de nombres principal de Entity Framework Core.

Detalles: Proporciona acceso a métodos de extensión de LINQ, como `ToListAsync()` y `OrderBy()`, que son utilizados para consultar la base de datos de forma asíncrona.

- **@inject AppDbContext Contexto:**

Propósito: Inyecta una instancia de `AppDbContext` en el componente.

Detalles: Utiliza la inyección de dependencias para obtener una referencia al contexto de la base de datos. Esta instancia (`Contexto`) se utiliza para realizar operaciones de consulta sobre la tabla de productos.

- **@rendermode InteractiveServer:**

Propósito: Especifica el modo de renderizado para este componente Blazor.

Detalles: `InteractiveServer` indica que el componente se renderizará en el servidor y mantendrá una conexión SignalR con el navegador para manejar interacciones del usuario y actualizaciones de la UI en tiempo real.

2.- Estructura de la Interfaz de Usuario.

```

<div class="contenedor-principal">
  <div class="contenedor-formulario">
    <h3 class="titulo-pagina">Consulta de Productos</h3>

    @if (productos == null)
    {
      <div class="alert alert-info">Cargando productos...</div>
    }
    else if (!productos.Any())
    {
      <div class="alert alert-info">
        No hay productos registrados en el inventario.
      </div>
    }
    else
    {
      <div class="form-group row mb-4">
        <label for="filtroProducto" class="col-md-2 col-form-label">Filtrar:</label>
        <div class="col-md-10">
          <div class="input-group">
            <input type="text" id="filtroProducto" @bind-Value="terminoFiltro"
              class="form-control" placeholder="ID, Nombre o Categoría..." />
            <button class="btn btn-primary" @onclick="AplicarFiltroConBoton">
              <i class="bi bi-search"></i> Buscar
            </button>
          </div>
          <small class="text-muted">Busque por ID, nombre o categoría del producto</small>
        </div>
      </div>
    }
  </div>

```

```

    @if (!string.IsNullOrEmpty(mensajeFeedback))
    {
      <div class="alert @mensajeCssClass mt-3" role="alert">
        @mensajeFeedback
      </div>
    }

    <div class="table-responsive">
      <table class="table table-hover table-bordered">
        <thead class="thead-dark">
          <tr>
            <th>ID</th>
            <th>Nombre</th>
            <th>Categoría</th>
            <th>Stock</th>
            <th>Mínimo</th>
            <th>Máximo</th>
            <th>Caducidad</th>
            <th>Llegada</th>
          </tr>
        </thead>
        <tbody>
          @foreach (var producto in productosFiltrados)
          {
            <tr>
              <td>@producto.Id</td>
              <td>@producto.Nombre</td>
              <td>@producto.Categoría</td>
              <td>@producto.CantidadStock</td>
              <td>@producto.MinimoStock</td>
              <td>@producto.MaximoStock</td>
              <td>@(producto.FechaCaducidad?.ToShortDateString() ?? "N/A")</td>
              <td>@(producto.FechaLlegada?.ToShortDateString() ?? "N/A")</td>
            </tr>
          }
        </tbody>
      </table>
    </div>
  </div>
</div>

```


- **Título Principal (<h3>):**
Propósito: Proporciona un encabezado claro para la página.
- **Renderizado Condicional (@if, @else if, @else):**
Propósito: Controla qué contenido se muestra al usuario en función del estado de la carga de productos y la existencia de datos.
- **productos == null:** Muestra un mensaje "Cargando productos..." mientras se esperan los datos de la base de datos.
- **!productos.Any():** Si productos está cargado pero está vacío, muestra un mensaje informativo indicando que no hay productos registrados.
- **else:** Una vez que los productos se han cargado y existen, muestra el formulario de filtro y la tabla de productos.
- **Sección de Filtrado (div.mb-3.d-flex.justify-content-between.align-items-center):**
Propósito: Agrupa el campo de búsqueda y el botón de filtro. Las clases de Bootstrap (d-flex, justify-content-between, align-items-center, mb-3) se utilizan para un diseño responsivo y una alineación ordenada.
- **Campo de Entrada (<InputText id="filtroProducto" ... />):**
 - **@bind-Value="terminoFiltro":** Enlaza el valor del campo de texto a la variable C# terminoFiltro. Cualquier cambio en el campo actualiza la variable, y viceversa.
 - **placeholder="ID, Nombre o Categoría...":** Proporciona una pista al usuario sobre qué tipos de valores puede introducir para el filtro.
- **Botón de Búsqueda (<button class="btn btn-primary" @onclick="AplicarFiltroConBoton">Buscar</button>):**
 - **@onclick="AplicarFiltroConBoton":** Asocia el evento de clic del botón con el método C# AplicarFiltroConBoton(), que ejecuta la lógica de filtrado.
- **Mensajes de Feedback (@if (!string.IsNullOrEmpty(mensajeFeedback))):**
Propósito: Muestra mensajes al usuario (éxito, error, advertencia) basándose en las variables mensajeFeedback y mensajeCssClass.
- **@mensajeCssClass:** Determina la apariencia del cuadro de mensaje (ej., alert alert-success, alert alert-danger).

- **Tabla de Productos** (`<table class="table table-striped table-bordered mt-3">`):
Propósito: Presenta los productos del inventario en un formato tabular, permitiendo una fácil visualización de sus atributos.
- **thead:** Define la fila de encabezados de la tabla, con columnas para ID, Nombre, Categoría, Cantidad en Stock, Stock Mínimo, Stock Máximo, Fecha de Caducidad y Fecha de Llegada.
- **tbody:** Contiene las filas de datos de los productos.
- **@foreach (var producto in productosFiltrados):** Itera sobre la lista `productosFiltrados` (que contiene los productos después de aplicar el filtro) para generar una fila `<tr>` para cada producto.
- **Celda de Fecha** (`@(producto.FechaCaducidad.HasValue ? ... : "N/A")`): Utiliza un operador ternario para mostrar la fecha formateada si existe (`HasValue`) o "N/A" si es nula. `ToShortDateString()` formatea la fecha a un formato corto legible.

3.- Bloque de Código C#

```

@code {
    private List<Producto>? productos; // Guarda todos los productos
    private List<Producto>? productosFiltrados; // Guarda los productos después de aplicar el filtro
    private string terminoFiltro = string.Empty;
    private string mensajeFeedback = string.Empty;
    private string mensajeCssClass = string.Empty;

    protected override async Task OnInitializedAsync()
    {
        await CargarProductos();
    }

    private async Task CargarProductos()
    {
        try
        {
            productos = await Contexto.Productos.OrderBy(p => p.Nombre).ToListAsync();
            // Al inicio, la lista filtrada es igual a la lista completa
            productosFiltrados = new List<Producto>(productos);
            mensajeFeedback = string.Empty;
        }
        catch (Exception ex)
        {
            mensajeFeedback = $"Error al cargar productos: {ex.Message}";
            mensajeCssClass = "alert alert-danger";
            Console.WriteLine($"Error al cargar productos: {ex.Message}");
        }
    }

    private void AplicarFiltroConBoton()
    {
        if (string.IsNullOrEmpty(terminoFiltro))
        {
            // Si el campo de filtro está vacío, mostramos todos los productos
            productosFiltrados = new List<Producto>(productos ?? new List<Producto>());
        }
        else
        {
            // Filtramos por ID, Nombre o Categoría, ignorando mayúsculas/minúsculas en texto
            productosFiltrados = productos?
                .Where(p => p.Id.ToString().Contains(terminoFiltro, StringComparison.OrdinalIgnoreCase) ||
                    p.Nombre.Contains(terminoFiltro, StringComparison.OrdinalIgnoreCase) ||
                    p.Categoría.Contains(terminoFiltro, StringComparison.OrdinalIgnoreCase))
                .ToList() ?? new List<Producto>();
        }

        // Mostrar mensaje si no hay resultados para el filtro
        if (!productosFiltrados.Any() && !string.IsNullOrEmpty(terminoFiltro))
        {
            mensajeFeedback = "No se encontraron productos que coincidan con el filtro.";
            mensajeCssClass = "alert alert-warning";
        }
        else
        {
            mensajeFeedback = string.Empty; // Limpiar mensaje si hay resultados o el filtro está vacío
        }
    }
}

```

- **Variables de Estado Privadas:**
- ***private List<Producto>? productos;***
 - **Propósito:** Almacena la lista **completa** de todos los productos recuperados de la base de datos. Se inicializa una vez al cargar el componente.
 - **Tipo:** List<Producto>? indica que puede ser una lista de objetos Producto o null.
- ***private List<Producto>? productosFiltrados;***

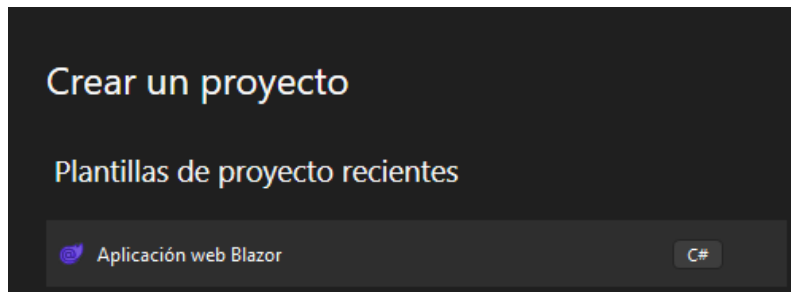
- **Propósito:** Almacena la lista de productos que se muestra actualmente en la tabla. Esta lista se actualiza cada vez que se aplica un filtro.
- **Tipo:** Similar a productos, puede ser una lista de Producto o null.
- ***private string terminoFiltro = string.Empty;;***
 - **Propósito:** Contiene el texto que el usuario introduce en el campo de filtro.
 - **Inicialización:** Se inicializa como una cadena vacía.
- ***private string mensajeFeedback = string.Empty;;***
 - **Propósito:** Almacena el mensaje de retroalimentación que se muestra al usuario (ej., "Error al cargar", "No encontrado").
 - **Inicialización:** Se inicializa como una cadena vacía.
- ***private string mensajeCssClass = string.Empty;;***
 - **Propósito:** Almacena la clase CSS que determina el estilo del cuadro de mensaje de retroalimentación (ej., "alert alert-success", "alert alert-danger").
 - **Inicialización:** Se inicializa como una cadena vacía.
- ***protected override async Task OnInitializedAsync():***
 - **Propósito:** Es un método del ciclo de vida de los componentes Blazor. Se invoca automáticamente cuando el componente ha terminado de inicializarse, antes de que se renderice por primera vez.
 - **Funcionamiento:** Llama al método CargarProductos() para obtener los datos del inventario. async y await se utilizan porque CargarProductos() realiza operaciones asíncronas de base de datos.
- ***private async Task CargarProductos():***
 - **Propósito:** Recupera todos los productos de la base de datos.
 - **Funcionamiento:**
 - Utiliza Contexto.Productos.OrderBy(p => p.Nombre).ToListAsync() para consultar la tabla Productos, ordenarlos por nombre y convertirlos en una lista.
 - Asigna la lista recuperada a la variable productos.
 - Inicializa productosFiltrados con una copia de productos para que todos los elementos sean visibles al cargar la página por primera vez.
 - Maneja cualquier excepción que pueda ocurrir durante la operación de la base de datos, mostrando un mensaje de error y registrándolo en la consola.

- ***private void AplicarFiltroConBoton():***
 - **Propósito:** Filtra la lista de productos basándose en el terminoFiltro ingresado por el usuario y actualiza productosFiltrados. Este método es llamado cuando el usuario hace clic en el botón "Buscar".
 - **Funcionamiento:**
 - **Verificación de terminoFiltro:** Si terminoFiltro está vacío o solo contiene espacios en blanco, productosFiltrados se restablece para mostrar todos los productos cargados inicialmente.
 - **Filtrado de Datos:** Si hay un terminoFiltro:
 - Usa LINQ (.Where()) para filtrar la lista productos.
 - La condición de filtro busca coincidencias en:
 - El ID del producto (convertido a cadena).
 - El Nombre del producto.
 - La Categoría del producto.
 - StringComparison.OrdinalIgnoreCase se utiliza para realizar comparaciones de texto **insensibles a mayúsculas y minúsculas**, lo que mejora la experiencia de búsqueda.
 - .ToList() convierte el resultado del filtro de nuevo a una lista.
 - **Mensajes de Retroalimentación:** Comprueba si la lista productosFiltrados está vacía después del filtro y si el terminoFiltro no estaba vacío. Si es así, muestra un mensaje indicando que no se encontraron coincidencias. De lo contrario, limpia cualquier mensaje de feedback previo.

Creación e Instalación del Proyecto

- Configuración de Proyecto en “Blazor”

Realizamos la creación del proyecto en **Aplicación web Blazor**



Creamos un Index, donde contiene un menú

```
@page "/"
<div class="contenedor-principal">
  <div class="contenedor-botones">
    <!-- Botón Alta -->
    <a href="/alta" class="boton-extra-ancho verde">
      <span class="icono">➕</span>
      <div class="contenido">
        <span class="titulo-boton">Registrar Producto</span>
        <span class="subtitulo">Agregar nuevos items al inventario</span>
      </div>
    </a>

    <!-- Botón Baja -->
    <a href="/baja" class="boton-extra-ancho rojo">
      <span class="icono">🗑️</span>
      <div class="contenido">
        <span class="titulo-boton">Eliminar Producto</span>
        <span class="subtitulo">Dar de baja items obsoletos</span>
      </div>
    </a>

    <!-- Botón Actualización -->
    <a href="/actualizacion" class="boton-extra-ancho naranja">
      <span class="icono">✏️</span>
      <div class="contenido">
        <span class="titulo-boton">Actualizar Existencias</span>
        <span class="subtitulo">Modificar cantidades o datos</span>
      </div>
    </a>

    <!-- Botón Consulta -->
    <a href="/consultas" class="boton-extra-ancho azul">
      <span class="icono">📄</span>
      <div class="contenido">
        <span class="titulo-boton">Consultar Inventario</span>
        <span class="subtitulo">Ver stock actual y reportes</span>
      </div>
    </a>
  </div>
</div>
```

De igual manera, se creó un **NavMenu.razor**, donde contiene el diseño general que contendrá el Nav en las páginas.

```
<div class="navbar-superior">
  <div class="contenido-navbar">
    <span class="logo">🍔</span>
    <h1>Gestión de Inventario - Taquería</h1>
  </div>
</div>

<style>
  .navbar-superior {
    background: #2d3748;
    color: white;
    padding: 1rem 2rem;
    box-shadow: 0 2px 10px rgba(0,0,0,0.1);
  }

  .contenido-navbar {
    max-width: 1200px;
    margin: 0 auto;
    display: flex;
    align-items: center;
    gap: 15px;
  }

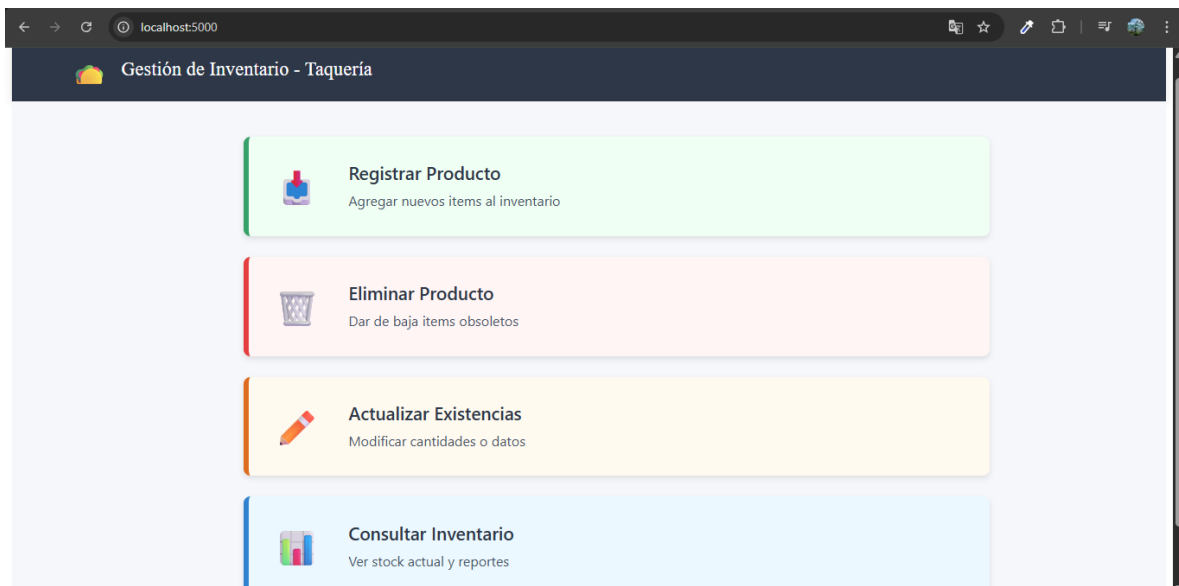
  .logo {
    font-size: 2rem;
  }

  h1 {
    font-size: 1.4rem;
    font-weight: 500;
    margin: 0;
  }

  @@media (max-width: 768px) {
    .contenido-navbar {
      justify-content: center;
    }

    h1 {
      font-size: 1.2rem;
    }
  }
</style>
```

Con lo anterior, al ejecutarlo, se muestra lo siguiente:

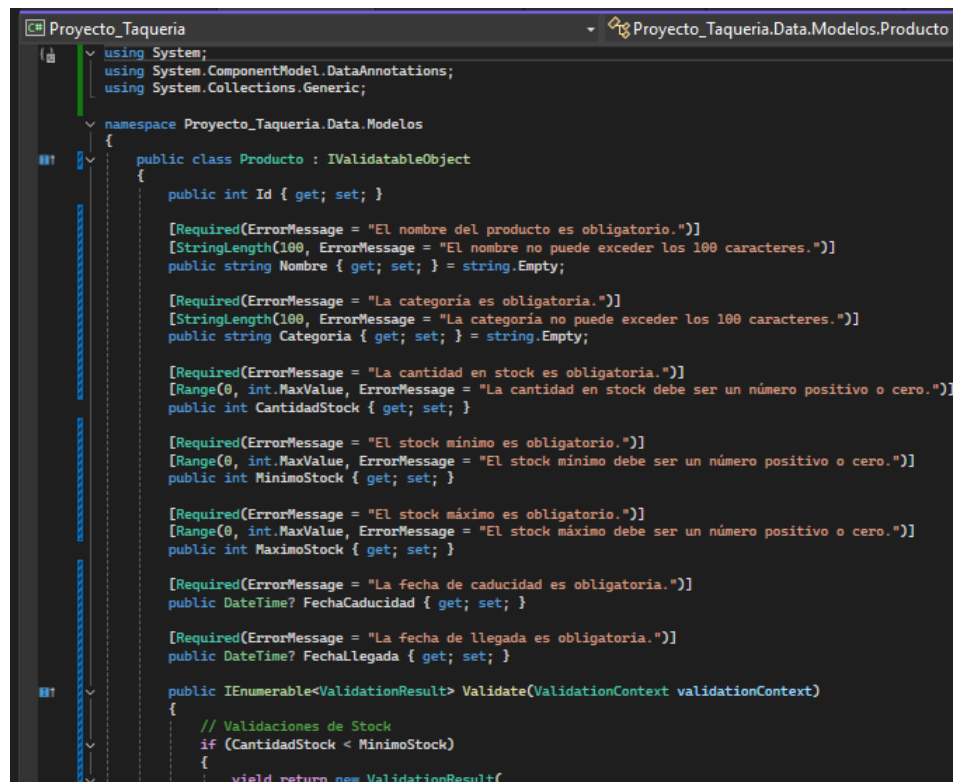


Para este proyecto, hacemos uso del modelo **Productos.cs**

Donde será el modelo principal y se guardará dentro de la BD, de esta manera, estaremos usando este modelo para hacer que el sistema funcione

Enlistando lo siguiente son:

- Nombre
- Categoría
- Cantidad en Stock
- Mínimo de Stock
- Máximo de Stock
- Fecha de Caducidad
- Fecha de Llegada



```
using System;
using System.ComponentModel.DataAnnotations;
using System.Collections.Generic;

namespace Proyecto_Taqueria.Data.Modelos
{
    public class Producto : IValidatableObject
    {
        public int Id { get; set; }

        [Required(ErrorMessage = "El nombre del producto es obligatorio.")]
        [StringLength(100, ErrorMessage = "El nombre no puede exceder los 100 caracteres.")]
        public string Nombre { get; set; } = string.Empty;

        [Required(ErrorMessage = "La categoría es obligatoria.")]
        [StringLength(100, ErrorMessage = "La categoría no puede exceder los 100 caracteres.")]
        public string Categoria { get; set; } = string.Empty;

        [Required(ErrorMessage = "La cantidad en stock es obligatoria.")]
        [Range(0, int.MaxValue, ErrorMessage = "La cantidad en stock debe ser un número positivo o cero.")]
        public int CantidadStock { get; set; }

        [Required(ErrorMessage = "El stock mínimo es obligatorio.")]
        [Range(0, int.MaxValue, ErrorMessage = "El stock mínimo debe ser un número positivo o cero.")]
        public int MinimoStock { get; set; }

        [Required(ErrorMessage = "El stock máximo es obligatorio.")]
        [Range(0, int.MaxValue, ErrorMessage = "El stock máximo debe ser un número positivo o cero.")]
        public int MaximoStock { get; set; }

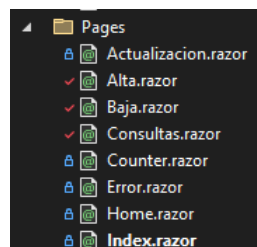
        [Required(ErrorMessage = "La fecha de caducidad es obligatoria.")]
        public DateTime? FechaCaducidad { get; set; }

        [Required(ErrorMessage = "La fecha de llegada es obligatoria.")]
        public DateTime? FechaLlegada { get; set; }

        public IEnumerable<ValidationResult> Validate(ValidationContext validationContext)
        {
            // Validaciones de Stock
            if (CantidadStock < MinimoStock)
            {
                yield return new ValidationResult(

```

Y se creó la carpeta de **Pages** (Donde contienen las páginas que usaremos en el proyecto).



- Configuración del Proyecto en SQL Server

Creamos un modelo llamado **AppDbContext**

```
using System.Collections.Generic;
using System.Reflection.Emit;
using Microsoft.EntityFrameworkCore;
using Proyecto_Taqueria.Data.Modelos;

namespace Proyecto_Taqueria.Data;

public class AppDbContext : DbContext
{
    public AppDbContext(DbContextOptions<AppDbContext> options) : base(options) { }

    public DbSet<Producto> Productos { get; set; } // Tabla "Productos"

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        //
    }
}
```

El anterior modelo, nos sirve para interactuar con la base de datos usando C#.

Seguido de eso, creamos una base de datos, donde se llamó **TaqueriaDB**



Con eso, realizamos la conexión de la Base de Datos al proyecto en el archivo **Appsettings.json**

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "BarConnection": "Server=LAPTOP-C8D85QK2\\SQLEXPRESS;Database=TaqueriaDB;Trusted_Connection=True;MultipleActiveResultSets=true;TrustServerCertificate=True"
  }
}
```

Nota: Es importante que el nombre del server sea el mismo que en SQLServer

Despues se crea una migracion para que los cambios en la BD sean posibles con el comando, esta solo crea una vez por proyecto:

- **dotnet ef migration add InitialCreate**

Para actualizar cambios en la base de datos es necesario ejecutar el siguiente comando:

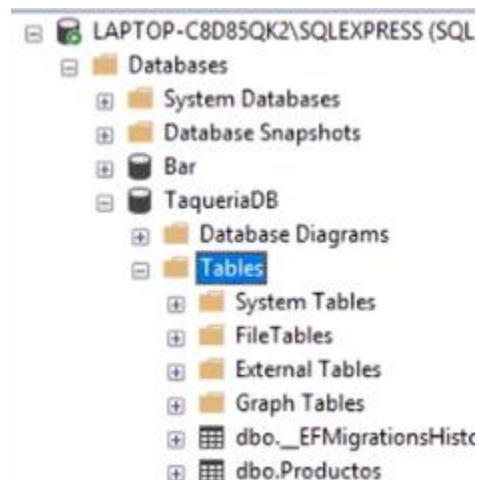
- **dotnet ef database update**

Este comando se utiliza para aplicar las migraciones de Entity Framework Core a la base de datos

```
PM> dotnet ef database update
Build started...
Build succeeded.

The Entity Framework tools version '8.0.10' is older than that of the runtime '9.0.5'. Update the tools for the latest features and bug fixes. See https://aka.ms/AAC1fbw for more information.
Info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (33ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      SELECT 1
Info: Microsoft.EntityFrameworkCore.Migrations[20411]
      Acquiring an exclusive lock for migration application. See https://aka.ms/efcore-docs-migrations-lock for more information if this takes too long.
      Acquiring an exclusive lock for migration application. See https://aka.ms/efcore-docs-migrations-lock for more information if this takes too long.
Info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (80ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
```

Una vez ejecutado todo lo anterior, nos aparece las siguientes tablas en nuestro SQL Server



De esa manera, teniendo configurado de manera correcta la Base de Datos

Nota: Para poder hacer migraciones del proyecto en una nueva máquina es necesario configurar los datos de la cadena de conexión, tener la base de datos creada y ejecutar únicamente el comando database update mencionado antes

- Configuración del Diseño

El diseño del proyecto general se decidió poner en app.css para que fuera el mismo en todas las páginas web.

