

Instituto



Obligatorio

Programación III

Santiago Moroy

Federico Goldaracena

Nicolás Olivera

Analista Programador

Semestre III

2024

Resumen

En esta propuesta de aplicación web para la cadena de restaurantes "Al mal tiempo, buena cara", se ha diseñado una simulación detallada que emula el funcionamiento de sus establecimientos temáticos. El desarrollo se lleva a cabo mediante el uso de ASP.NET MVC, asegurando una implementación robusta y eficiente. La gestión integral de los restaurantes abarca diversas áreas clave, entre las que se encuentran reservas, clientes, mesas y promociones basadas en las condiciones climáticas. La información correspondiente a estas áreas se organiza y gestiona mediante el empleo de listas, proporcionando una estructura eficaz y dinámica.

Cuando un usuario accede a la aplicación, se encuentra con una interfaz de inicio de sesión que actúa como el punto de entrada al sistema. La asignación de permisos se lleva a cabo de manera inteligente, ajustándose a las necesidades y responsabilidades específicas de cada tipo de usuario. Un ejemplo claro de esta diferenciación se observa entre el administrador y el cliente registrado.

El administrador, como figura clave en la gestión, goza de amplias facultades que abarcan la visualización y administración de clientes, reservas, mesas y promociones. Su capacidad de acción incluye la realización de altas, bajas, modificaciones, actualizaciones y eliminaciones de registros en todas estas categorías. Este nivel de control y autoridad permite al administrador desempeñar un papel central en la supervisión y dirección de los restaurantes simulados.

En contraste, el cliente registrado cuenta con permisos más específicos y orientados a su función principal. Su acceso se centra en la consulta y realización de reservas, así como en la gestión de sus propias reservas existentes. Además, el sistema aplica descuentos basados tanto en el tipo de cliente como en las condiciones climáticas, optimizando su experiencia dentro de la aplicación y garantizando un flujo de trabajo eficiente y enfocado en sus necesidades directas.

En resumen, esta aplicación web no solo simula la operación de una cadena de restaurantes temáticos, sino que también demuestra una estructura de gestión versátil y adaptativa, garantizando que cada usuario tenga acceso solo a las funciones necesarias para su rol específico. La implementación de tecnologías como ASP.NET MVC contribuye a la solidez y eficacia del sistema en su conjunto.

Contenido

1. INTRODUCCIÓN	4
1.1 .NET	4
1.2 ADO.NET	4
1.3 Visual Studio.....	5
1.4 C#	5
1.5 SQL Server Management Studio	5
1.6 DATABASE FIRST	6
1.7 MVC	6
1.8 GIT	6
1.9 GITHUB	7
1.10 EntityFramework	7
1.11 API	8
2. DESARROLLO.....	9
2.1 Base de Datos en SQL SMS.....	9
2.2 CREAR EL MVC	10
2.2.1 INICIALIZANDO EL PROYECTO	10
2.2.2 EXPORTAR LA BASE DE DATOS	11
2.2.3 CREAR LOS CONTROLADORES	12
2.2.4 CREAR LAS VISTAS	16
2.2.5 LAYOUT	17
2.2.6 RESTAURANTES Y CLIENTES.....	18
2.2.7 PERMISOS.....	18
2.2.8 API CLIMA	18
2.2.8 PAGOS.....	20
2.2.9 DESCUENTOS.....	20
3. RESULTADOS	21
3.1 PRUEBAS CRUD	21
3.1.1 CREATE	21
3.1.2 READ	22
3.1.3 UPDATE	23
3.1.4 DELETE.....	24
3.2 PRUEBAS DE DESCUENTOS	24
3.3 PRUEBAS DE MANEJO DE PESTAÑAS.....	25
4. CONCLUSIONES	27
5. REFERENCIAS	28

1. INTRODUCCIÓN

1.1 .NET

.NET es una plataforma de código abierto que facilita la creación de aplicaciones para escritorio, web y móviles, con capacidad de ejecutarse en diversos sistemas operativos. La plataforma abarca herramientas, bibliotecas y lenguajes que respaldan el desarrollo de software moderno, escalable y de alto rendimiento. Respaldada por una comunidad de desarrolladores activa, .NET se mantiene y evoluciona constantemente. En resumen, .NET realiza varias funciones esenciales: **Compilación de Código:** Traduce el código del lenguaje de programación .NET en instrucciones comprensibles para los dispositivos de computación. **Utilidades para Desarrollo Eficiente:** Ofrece utilidades que facilitan el desarrollo de software eficiente, como la obtención de la hora actual o la impresión de texto en pantalla. **Gestión de Tipos de Datos:** Define un conjunto de tipos de datos para almacenar información, como texto, números y fechas en los equipos. Esta funcionalidad es crucial para la manipulación y almacenamiento de datos en las aplicaciones desarrolladas con .NET. [1]

1.2 ADO.NET

ADO.NET proporciona acceso coherente a orígenes de datos como SQL Server y XML, así como a orígenes de datos expuestos mediante OLE DB y ODBC. Las aplicaciones de consumidor que comparten datos pueden utilizar ADO.NET para conectar a estos orígenes de datos y recuperar, controlar y actualizar los datos contenidos.

ADO.NET separa el acceso a datos de la manipulación de datos y crea componentes discretos que se pueden utilizar por separado o conjuntamente. ADO.NET incluye proveedores de datos .NET Framework para conectarse a una base de datos, ejecutar comandos y recuperar resultados. Los resultados se procesan directamente o se colocan en un objeto DataSet de ADO.NET con el fin de exponerlos al usuario para un propósito específico, combinados con datos de varios orígenes, o de pasarlos entre niveles. El objeto DataSet de ADO.NET también puede utilizarse independientemente de un proveedor de datos .NET Framework para administrar datos que son locales de la aplicación o que proceden de un origen XML.

ADO.NET proporciona funcionalidad a los desarrolladores que escriben código administrado similar a la funcionalidad que los objetos ADO (ActiveX Data Objects) proporcionan a los desarrolladores de modelo de objetos componentes (COM) nativo. Se recomienda utilizar ADO.NET, y no ADO, para obtener acceso a datos de aplicaciones .NET. [2]

1.3 Visual Studio

Visual Studio destaca como una poderosa herramienta de desarrollo que facilita la gestión integral del ciclo de desarrollo en un único entorno. Este entorno de desarrollo integrado (IDE) ofrece capacidades completas para escribir, editar, depurar y compilar código, además de permitir la implementación de la aplicación resultante. Más allá de las funciones básicas de edición y depuración de código, Visual Studio proporciona una gama de herramientas y características adicionales.

Entre ellas se incluyen compiladores, utilidades de finalización de código, control de código fuente, extensiones y diversas funcionalidades diseñadas para optimizar cada etapa del proceso de desarrollo de software. En síntesis, Visual Studio se presenta como una solución integral que va más allá de la simple edición de código, brindando a los desarrolladores un conjunto completo de recursos para mejorar la eficiencia y la calidad en todas las fases del desarrollo de aplicaciones. [3]

1.4 C#

C# (pronunciado C Sharp) representa una evolución significativa realizada por Microsoft, amalgamando lo más destacado de los lenguajes C y C++. A lo largo de su desarrollo continuo, se le han incorporado funcionalidades provenientes de otros lenguajes, como Java, aprovechando aspectos de su sintaxis evolucionada. Este lenguaje, orientado a objetos en toda la plataforma NET (tanto Framework como Core), ha ido adquiriendo las facilidades de creación de código presentes en Visual Basic, otro de los lenguajes eminentemente utilizado por Microsoft. Esta amalgama le confiere a C# una versatilidad excepcional, convirtiéndolo en un lenguaje accesible y fácil de aprender, sin sacrificar la potencia inherente a C. Con la llegada de la versión .NET Core, se ha llevado a cabo una reconstrucción completa del compilador de C#, resultando en una mejora impresionante: las aplicaciones ahora se ejecutan hasta un 600% más rápido que en versiones anteriores. Este avance sustancial refuerza la posición de C# como un lenguaje moderno, potente y eficiente dentro del ecosistema de desarrollo de Microsoft. [4]

1.5 SQL Server Management Studio

SQL Server Management Studio (SSMS) es un entorno integrado para administrar cualquier infraestructura de SQL, desde SQL Server a Azure SQL Database. SSMS proporciona herramientas para configurar, supervisar y administrar instancias de SQL Server y bases de datos. Use SSMS para implementar, supervisar y actualizar los componentes de nivel de datos que usan las aplicaciones, además de compilar consultas y scripts.

Use SSMS para consultar, diseñar y administrar bases de datos y almacenes de datos, estén donde estén, en el equipo local o en la nube. [5]

1.6 DATABASE FIRST

Con la metodología Database First o Base de Datos Primero se parte desde una base de datos existente, es decir que ya tendríamos la base de datos diseñada con todo: tablas, campos, restricciones, incluso también datos y realizaríamos el proceso inverso al Code First que consiste en crear los modelos (clases) para cada una de estas tablas. Database First permite aplicar ingeniería inversa a un modelo es decir a partir de una base de datos existente.

Ventajas de Trabajar con Database First:

- Database First permite aplicar ingeniería inversa.
- El trabajo es muy sencillo con Visual Studio y control de código.
- Se puede seguir actualizando la base de datos a partir del modelo usando las migraciones, por ejemplo, si necesitamos cambiar o eliminar una tabla en la base de datos se podrá hacer con una migración a pesar de que hayamos partido inicialmente desde una database first.
- Usando Entity Framework Core existen comandos para hacer ingeniería inversa y así lograr pasar la base de datos, tables y demás a clases de modelo usando sencillos comandos y personalizando nombres y organización de carpetas y código. [6]

1.7 MVC

En líneas generales, MVC es una propuesta de arquitectura del software utilizada para separar el código por sus distintas responsabilidades, manteniendo distintas capas que se encargan de hacer una tarea muy concreta, lo que ofrece beneficios diversos.

MVC se usa inicialmente en sistemas donde se requiere el uso de interfaces de usuario, aunque en la práctica el mismo patrón de arquitectura se puede utilizar para distintos tipos de aplicaciones. Surge de la necesidad de crear software más robusto con un ciclo de vida más adecuado, donde se potencie la facilidad de mantenimiento, reutilización del código y la separación de conceptos.

Su fundamento es la separación del código en tres capas diferentes, acotadas por su responsabilidad, en lo que se llaman Modelos, Vistas y Controladores, o lo que es lo mismo, Model, Views & Controllers, si lo prefieres en inglés. En este artículo estudiaremos con detalle estos conceptos, así como las ventajas de ponerlos en marcha cuando desarrollamos.

MVC es un "invento" que ya tiene varias décadas y fue presentado incluso antes de la aparición de la Web. No obstante, en los últimos años ha ganado mucha fuerza y seguidores gracias a la aparición de numerosos frameworks de desarrollo web que utilizan el patrón MVC como modelo para la arquitectura de las aplicaciones web. [7]

1.8 GIT

Git es un sistema de control de versiones distribuido, lo que significa que un clon local del proyecto es un repositorio de control de versiones completo. Estos repositorios locales plenamente funcionales permiten trabajar sin conexión o de forma remota con facilidad. Los desarrolladores confirman su trabajo localmente y, a continuación, sincronizan la copia del repositorio con la del servidor. Este paradigma es distinto del control de versiones centralizado, donde los clientes deben sincronizar el código con un servidor antes de crear nuevas versiones.

Git es uno de estos sistemas de control, que permite comparar el código de un archivo para ver las diferencias entre las versiones, restaurar versiones antiguas si algo sale mal, y fusionar los cambios de distintas versiones. También permite trabajar con distintas ramas de un proyecto, como la de desarrollo para meter nuevas funciones al programa o la de producción para depurar los bugs. [8]

1.9 GITHUB

Como su nombre indica, la web utiliza el sistema de control de versiones Git. Un sistema de gestión de versiones es ese con el que los desarrolladores pueden administrar su proyecto, ordenando el código de cada una de las nuevas versiones que sacan de sus aplicaciones para evitar confusiones. Así, al tener copias de cada una de las versiones de su aplicación, no se perderán los estados anteriores cuando se va a actualizar.

Las principales características de la plataforma es que ofrece las mejores características de este tipo de servicios sin perder la simplicidad, y es una de las más utilizadas del mundo por los desarrolladores. Es multiplataforma, y tiene multitud de interfaces de usuario.

Github es un portal para gestionar las aplicaciones que utilizan el sistema Git. Además de permitirte mirar el código y descargar las diferentes versiones de una aplicación, la plataforma también hace las veces de red social conectando desarrolladores con usuarios para que estos puedan colaborar mejorando la aplicación. [9]

1.10 EntityFramework

Entity Framework es el ORM oficial de Microsoft para el desarrollo en la plataforma .NET, con C# y otros lenguajes para la plataforma.

Entity Framework es una herramienta de las catalogadas como ORM (Object Relational Mapping o mapeo a objetos de las bases de datos relacionales) que permite trabajar con las bases de datos relacionales a alto nivel, evitando las complejidades y particularidades del manejo de las tablas de bases de datos, sus relaciones y el uso de SQL.

Gracias a Entity Framework podemos realizar las conexiones con las bases de datos de manera que no tengamos que hacer consultas SQL para extraer los datos, sino simples métodos de acceso a las entidades. Gracias a esos métodos el framework se conectará con la base de datos y realizará las consultas de manera transparente para el desarrollador, atendiendo al modelo de datos que tengamos definidos en el código

fuente de las aplicaciones. Esta herramienta nos permite por tanto dos capacidades principales:

Abstraernos del modelo de sistema gestor de la base de datos utilizado, ya que el sistema de base de datos que tengamos por debajo es independiente del código de nuestra aplicación. En otras palabras, será independiente de si trabajamos con PostgreSQL, SQL Server, MariaDB, etc.

Abstraerse del sistema relacional de los datos, de modo que no tenemos que lidiar con consultas SQL, acceder a los datos relacionados, etc. En cambio, el modelo de datos de nuestra aplicación estará especificado por medio de código en C#. Por tanto, nosotros no vamos a tener que lidiar con ADO.NET para recorrer las filas de las tablas, sino que Entity Framework nos entregará directamente colecciones de objetos para poder tratarlos a alto nivel.

Otra de las características de Entity Framework es que funciona por convenciones y no configuraciones. Eso quiere decir que si trabajamos de la manera particular esperada por Entity Framework no tendremos que realizar ninguna configuración. Por contra, si no seguimos las convenciones tenemos que añadir código en tus clases C# para que Entity Framework sepa cómo deseamos trabajar. [10]

1.11 API

Una API, o interfaz de programación de aplicaciones, es un conjunto de reglas o protocolos que permiten que las aplicaciones de software se comuniquen entre sí para intercambiar datos, características y funcionalidades.

Las API simplifican y aceleran el desarrollo de software y aplicaciones permitiendo a los desarrolladores integrar datos, servicios y capacidades de otras aplicaciones, en lugar de desarrollarlas desde cero. Las API también ofrecen a los propietarios de aplicaciones una forma sencilla y segura de poner los datos y las funciones de sus aplicaciones a disposición de los departamentos de su organización. Los propietarios de aplicaciones también pueden compartir o comercializar datos y funciones con asociados de negocios o terceros.

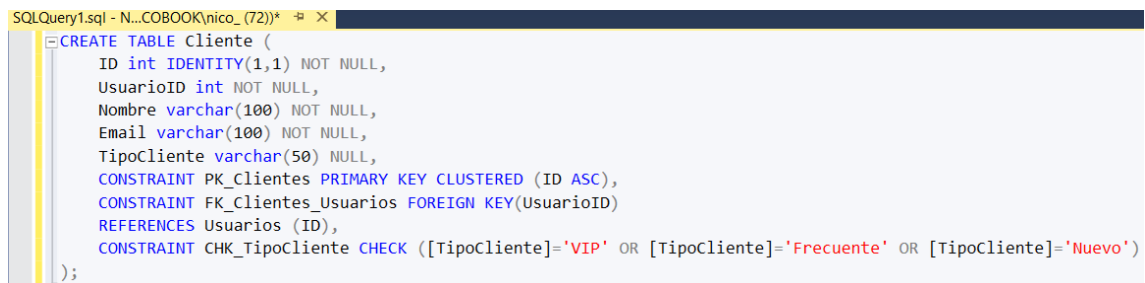
Las API permiten compartir solo la información necesaria, manteniendo ocultos otros detalles internos del sistema, lo que ayuda a la seguridad del sistema. Los servidores o dispositivos no tienen que exponer completamente los datos: las API permiten compartir pequeños paquetes de datos, relevantes para la solicitud específica. [11]

2. DESARROLLO

LINK DE GITHUB

2.1 Base de Datos en SQL SMS.

El proyecto comienza creando la base de datos en SQL SMS (Server Management Studio), a partir del diagrama realizado. Atribuyéndole a cada Tabla sus referencias y valores correspondientes. Con sentencias SQL, se crearon diferentes las clases, como Cliente, con referencias y restricciones correspondientes. (Figura 2.1).



```
SQLQuery1.sql - N...COBOOK\nico_(72))* X
CREATE TABLE Cliente (
    ID int IDENTITY(1,1) NOT NULL,
    UsuarioID int NOT NULL,
    Nombre varchar(100) NOT NULL,
    Email varchar(100) NOT NULL,
    TipoCliente varchar(50) NULL,
    CONSTRAINT PK_Clientes PRIMARY KEY CLUSTERED (ID ASC),
    CONSTRAINT FK_Clientes_Usuarios FOREIGN KEY(UsuarioID)
    REFERENCES Usuarios (ID),
    CONSTRAINT CHK_TipoCliente CHECK ([TipoCliente]='VIP' OR [TipoCliente]='Frecuente' OR [TipoCliente]='Nuevo')
);
```

Figura 2.1

En el Explorador de Objetos, se puede ver la base de datos, con las diferentes carpetas generadas, con sus tablas, sus restricciones, y referencias a otras tablas, permitiendo desde ahí, ingresar datos si así se desea. (Figura 2.2)

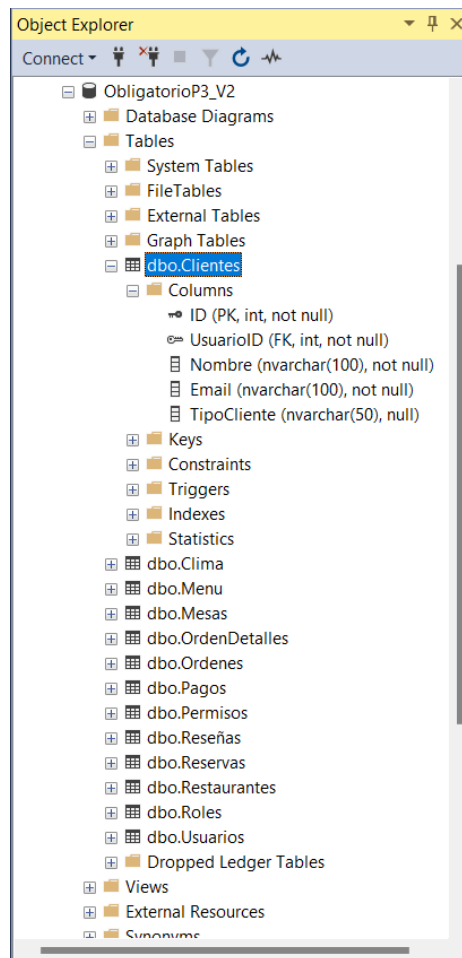


Figura 2.2

2.2 CREAR EL MVC

2.2.1 INICIALIZANDO EL PROYECTO

Luego se crea un proyecto en Visual Studio del tipo “Aplicación web de ASP.NET Core (Modelo-Vista-Controlador)”. Figura 2.3

Luego, en la sección de Herramientas/Administrador de Paquetes NuGet, se debe abrir Administrar paquetes NuGet para la solución.

Ahí, habrá que buscar en la pestaña Examinar, los paquetes correspondientes, los cuales son: (Ver Figura 2.4)

Microsoft.EntityFrameworkCore

Microsoft.EntityFrameworkCore.Design

Microsoft.EntityFrameworkCore.SqlServer

Microsoft.EntityFrameworkCore.Tools

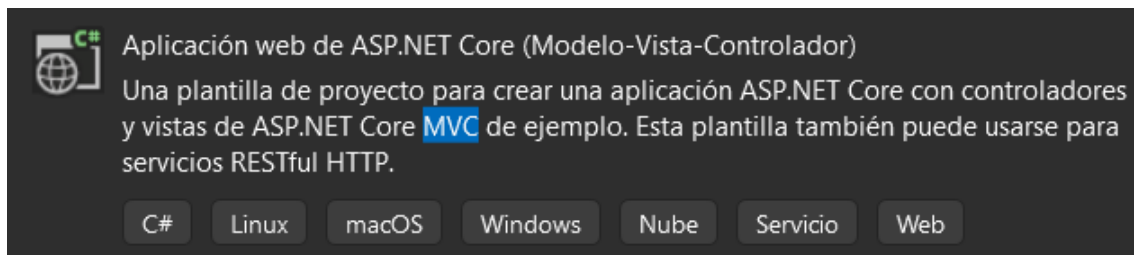


Figura 2.3



Figura 2.4

2.2.2 EXPORTAR LA BASE DE DATOS

Luego de que los paquetes se encuentren instalados, en “Consola del Administrador de paquetes” se ejecuta la siguiente línea de conexión: Scaffold-DbContext "Data Source= Ruta donde se encuentra la base de datos ; Initial Catalog= Nombre de la base de datos creada;Integrated Security=True; TrustServerCertificate=True" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models. (Ejemplo en Figura 2.5)

La cual tiene la dirección y nombre de base de datos establecida, para exportar las tablas, relaciones y atributos correspondientes.

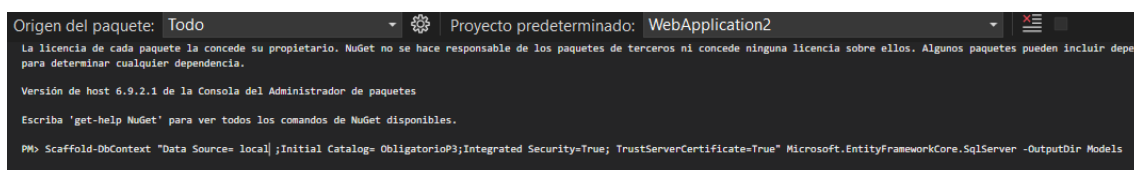


Figura 2.5

Para finalizar la exportación, se verifica en “Explorador de Soluciones” que se encuentren las clases con los mismos nombres que las tablas creadas en SQL SMS.

Las clases se ubicarán en la carpeta Models. Mientras que Views contiene la interfaz para Home y Shared, la cual contiene el Layout que compartirán las pestañas. Y la carpeta Controllers, solo contendrá HomeController hasta ese punto.

2.2.3 CREAR LOS CONTROLADORES

Dentro de la carpeta Controllers, con click derecho en la carpeta, se seleccionará Agregar/Controlador, y se elige la opción “Controlador MVC con vistas que usan Entity Framework”. Figura 2.6

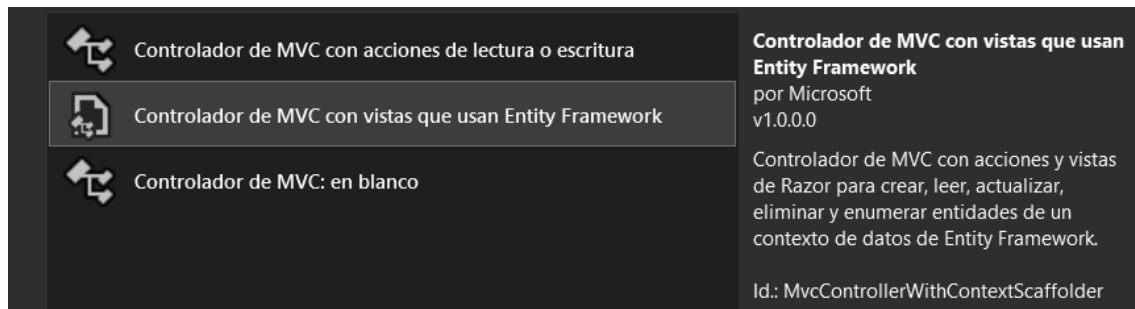


Figura 2.6

Se selecciona la Clase de modelo al cual le crearemos el Controlador, paso a repetir con todos los modelos. Ejemplo en Figura 2.7

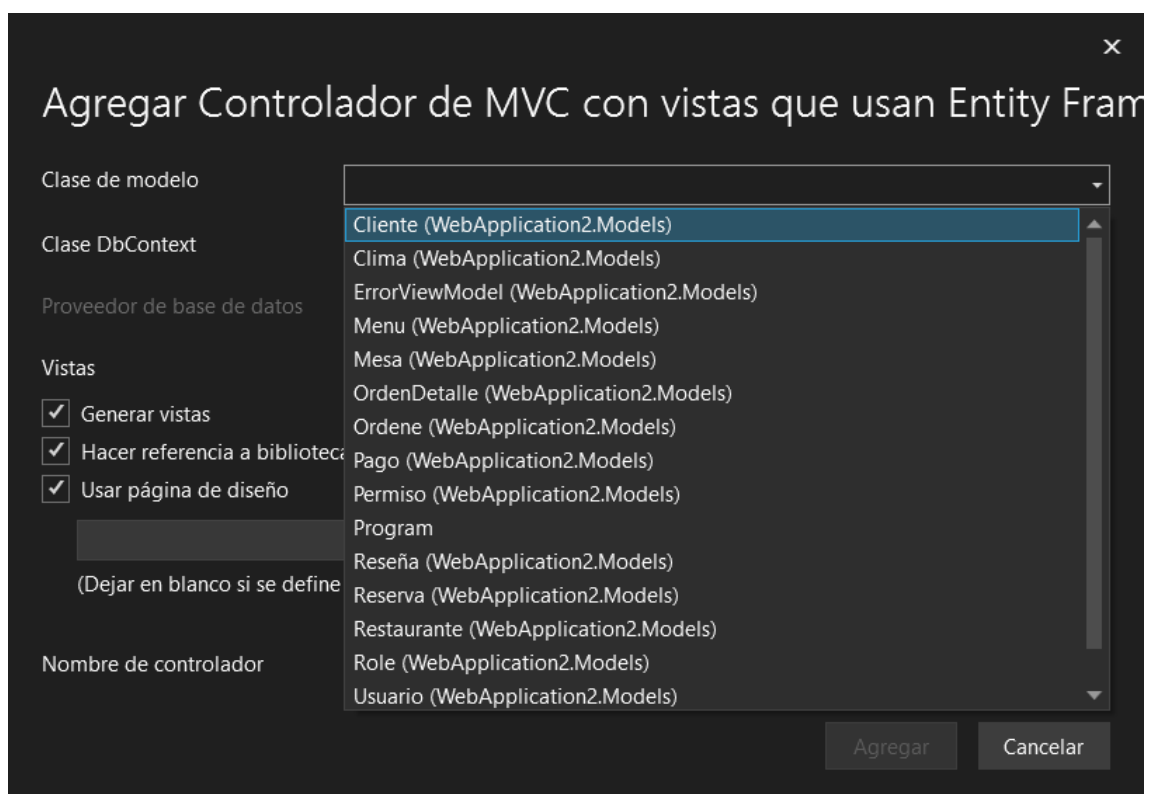
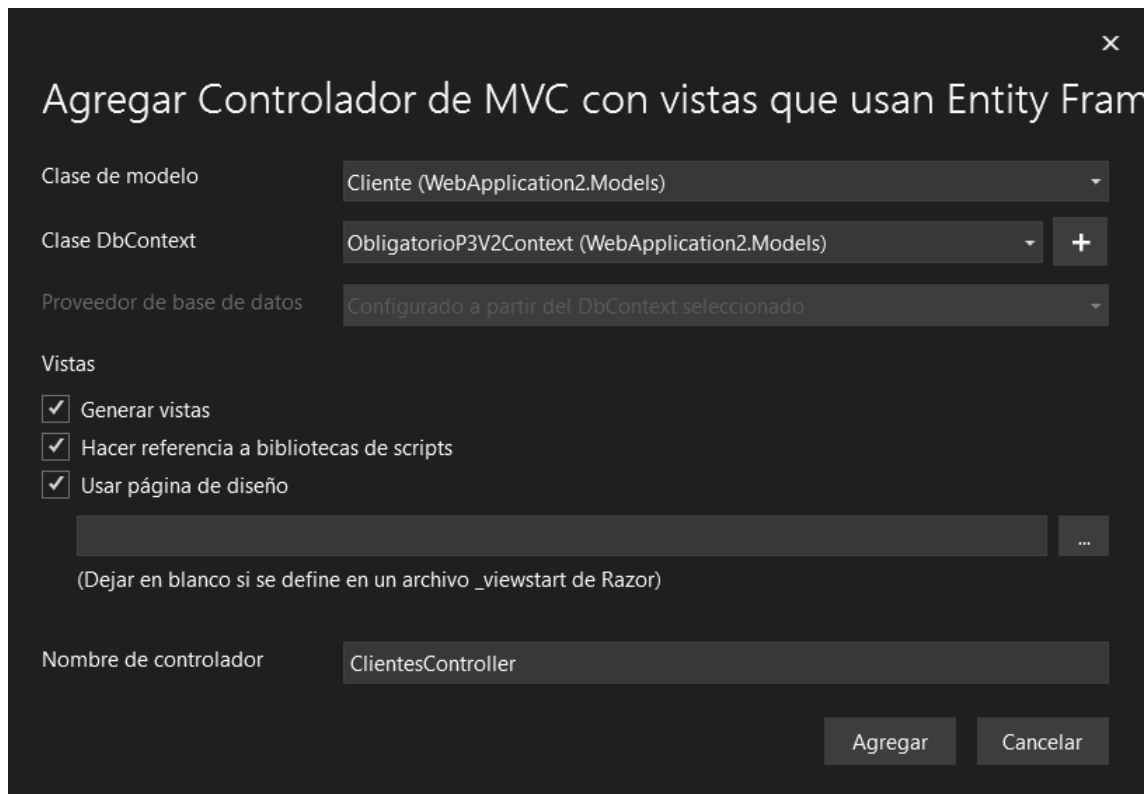


Figura 2.7

La clase DbContext te permite extraerlo desde la carpeta Models, dentro del mismo proyecto, y al tener el check en la casilla “Generar vistas”, las generará automáticamente a la par del controlador. El nombre te lo brinda Visual Studio en base

al Modelo, agregando la palabra Controller y ubicándolo dentro de los controladores. Ver Figura 2.8



Agregar Controlador de MVC con vistas que usan Entity Framework

Clase de modelo: Cliente (WebApplication2.Models)

Clase DbContext: ObligatorioP3V2Context (WebApplication2.Models) +

Proveedor de base de datos: Configurado a partir del DbContext seleccionado

Vistas

- ☒ Generar vistas
- ☒ Hacer referencia a bibliotecas de scripts
- ☒ Usar página de diseño

(Dejar en blanco si se define en un archivo _viewstart de Razor)

Nombre de controlador: ClientesController

Agregar Cancelar

Figura 2.8

Si se explora más detalladamente el controlador recién agregado de ClientesController, se verá la solución importando clases y métodos como por ejemplo de Linq, y luego se ve más detalladamente una clase ClientesController, que hereda de Controller.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using WebApplication2.Models;

namespace WebApplication2.Controllers
{
    1 referencia
    public class ClientesController : Controller
    {
        private readonly ObligatorioP3V2Context _context;

        0 referencias
        public ClientesController(ObligatorioP3V2Context context)
        {
            _context = context;
        }
    }
}
```

Se ven funciones para las diferentes acciones que luego formarán parte del sistema, como las habituales para el manejo de datos CRUD (Create, Read, Update, Delete).

A continuación, se muestran las diferentes funciones (en orden CRUD) en código:

```
// GET: Clientes/Create
0 referencias
public IActionResult Create()
{
    ViewData["UsuarioId"] = new SelectList(_context.Usuarios, "Id", "Id");
    return View();
}

// POST: Clientes/Create
// To protect from overposting attacks, enable the specific properties you want to bind to.
// For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
0 referencias
public async Task<IActionResult> Create([Bind("Id,UsuarioId,Nombre,Email,TipoCliente")] Cliente cliente)
{
    if (ModelState.IsValid)
    {
        _context.Add(cliente);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    ViewData["UsuarioId"] = new SelectList(_context.Usuarios, "Id", "Id", cliente.UsuarioId);
    return View(cliente);
}
```

Figura 2.10: Método Create dentro del Controlador de Clientes

```
// GET: Usuarios/Details/5
0 referencias
public async Task<IActionResult> Details(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var usuario = await _context.Usuarios
        .Include(u => u.Rol)
        .FirstOrDefaultAsync(m => m.Id == id);
    if (usuario == null)
    {
        return NotFound();
    }

    return View(usuario);
}
```

Figura 2.11: Método Read, el cual permite leer o obtener datos.

```

// GET: Usuarios/Edit/5
0 referencias
public async Task<IActionResult> Edit(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var usuario = await _context.Usuarios.FindAsync(id);
    if (usuario == null)
    {
        return NotFound();
    }
    ViewData["RolId"] = new SelectList(_context.Roles, "Id", "Nombre", usuario.RolId);
    return View(usuario);
}

// POST: Usuarios/Edit/5
[HttpPost]
[ValidateAntiForgeryToken]
0 referencias
public async Task<IActionResult> Edit(int id, [Bind("Id,Nombre,Email,Contraseña,RolId")] Usuario usuario)
{
    if (id != usuario.Id)
    {
        return NotFound();
    }

    if (ModelState.IsValid)
    {
        try
        {
            _context.Update(usuario);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!UsuarioExists(usuario.Id))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }
        catch (Exception ex)
        {
            ModelState.AddModelError("", $"Error al editar el usuario: {ex.Message}");
        }
    }
    ViewData["RolId"] = new SelectList(_context.Roles, "Id", "Nombre", usuario.RolId);
    return View(usuario);
}

```

Figura 2.12: Método Update, es de código más largo por las palabras clave async y await, que utilizan para manejar datos de manera asíncrona.

“async” y “await” no son métodos; son palabras clave en varios lenguajes de programación que se utilizan para manejar la programación asíncrona. Permiten que el programa continúe ejecutándose mientras espera a que se completen las operaciones asíncronas, en lugar de bloquear la ejecución.

```

// GET: Usuarios/Delete/5
0 referencias
public async Task<IActionResult> Delete(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var usuario = await _context.Usuarios
        .Include(u => u.Rol)
        .FirstOrDefaultAsync(m => m.Id == id);
    if (usuario == null)
    {
        return NotFound();
    }

    return View(usuario);
}

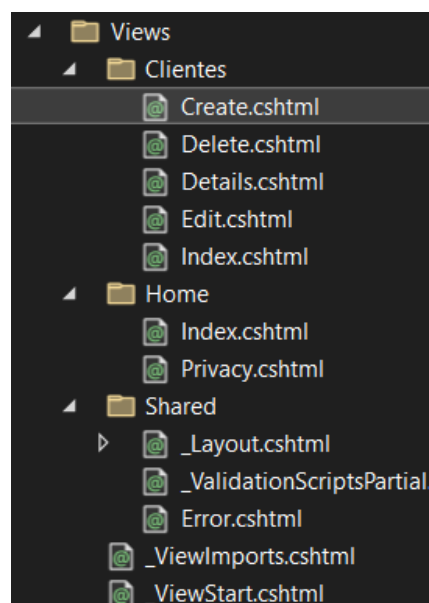
// POST: Usuarios/Delete/5
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
0 referencias
public async Task<IActionResult> DeleteConfirmed(int id)
{
    var usuario = await _context.Usuarios.FindAsync(id);
    if (usuario != null)
    {
        try
        {
            _context.Usuarios.Remove(usuario);
            await _context.SaveChangesAsync();
        }
        catch (Exception ex)
        {
            ModelState.AddModelError("", $"Error al eliminar el usuario: {ex.Message}");
        }
    }
    return RedirectToAction(nameof(Index));
}

```

Figura 2.13: Método Delete, para eliminar al Cliente.

2.2.4 CREAR LAS VISTAS

Si bien las Vistas, en el momento que se crea un controlador, tenemos la opción de crearla, lo que aparece es una estructura básica para el manejo necesario del proyecto.



Dentro de cada archivo. cshtml, tendremos una combinación de etiquetas html con css (Realmente manejadas con Bootstrap), con el fin de generar la interfaz con la que interactúa el cliente. Los inputs tomando el control para enviar los datos, que luego con el Controlador y Modelo, generarán la acción de Crear un Cliente, guardando sus datos directamente en la base de datos, y sin necesidad de interactuar directamente con ella mediante SQL SMS. Figura 2.14

```
<div class="row">
  <div class="col-md-4">
    <form asp-action="Create">
      <div asp-validation-summary="ModelOnly" class="text-danger"></div>
      <div class="form-group">
        <label asp-for="UsuarioId" class="control-label"></label>
        <select asp-for="UsuarioId" class="form-control" asp-items="ViewBag.UsuarioId"></select>
      </div>
      <div class="form-group">
        <label asp-for="Nombre" class="control-label"></label>
        <input asp-for="Nombre" class="form-control" />
        <span asp-validation-for="Nombre" class="text-danger"></span>
      </div>
      <div class="form-group">
        <label asp-for="Email" class="control-label"></label>
        <input asp-for="Email" class="form-control" />
        <span asp-validation-for="Email" class="text-danger"></span>
      </div>
      <div class="form-group">
        <label asp-for="TipoCliente" class="control-label"></label>
        <input asp-for="TipoCliente" class="form-control" />
        <span asp-validation-for="TipoCliente" class="text-danger"></span>
      </div>
      <div class="form-group">
        <input type="submit" value="Create" class="btn btn-primary" />
      </div>
    </form>
  </div>
</div>
```

Figura 2.14

De esa misma manera, maneja el sistema las secciones de Editar, o Eliminar a un Cliente.

2.2.5 LAYOUT

Una vez finalizadas las observaciones a detalle, se procede a generar la primera interfaz o vista que tendrá el Usuario, agregando un componente a la navbar si así lo desea, y una pestaña nueva para ver, la cual se encontrará con o sin datos, dependiendo de la base de datos en ese momento. La etiqueta asp-action, indica la acción a realizar, que en caso de Index, devolverá una vista dependiendo del controlador que se utilice. Ejemplo en Figura 2.15

```

<button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target=".navbar-collapse" aria-controls="navbarSupportedContent"
aria-expanded="false" aria-label="Toggle navigation">
  <span class="navbar-toggler-icon"></span>
</button>
<div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">
  <ul class="navbar-nav flex-grow-1">
    <li class="nav-item">
      <a class="nav-link text-dark" asp-area="" asp-controller="Usuarios" asp-action="Index">Usuarios</a>
    </li>
    <li class="nav-item">
      <a class="nav-link text-dark" asp-area="" asp-controller="Clientes" asp-action="Index">Clientes</a>
    </li>
    <li class="nav-item">
      <a class="nav-link text-dark" asp-area="" asp-controller="Roles" asp-action="Index">Roles</a>
    </li>
    <li class="nav-item">
      <a class="nav-link text-dark" asp-area="" asp-controller="Permisos" asp-action="Index">Permisos</a>
    </li>
    <li class="nav-item">
      <a class="nav-link text-dark" asp-area="" asp-controller="Menus" asp-action="Index">Menus</a>
    </li>
  </ul>
</div>

```

Figura 2.15

2.2.6 RESTAURANTES Y CLIENTES

Dentro del sistema, en las clases se encuentran relacionados mediante foreign keys, tanto las entidades de Restaurantes, con Menú, Mesas, Reservas, Reseñas, etc. Se busca que el sistema, mantenga conexión directa entre entidades y una comunicación entre las mismas que mantenga flexibilidad y contenga la capacidad de crecimiento.

2.2.7 PERMISOS

Los permisos fueron creados en una Tabla particular, para conectar de mejor manera, los roles, con los usuarios y sus permisos concedidos.

2.2.8 API CLIMA

Para implementar la API del Clima en el proyecto, se debe instalar otros paquetes de NuGet llamados Newtonsoft.json y Restsharp.

Luego, en la documentación de la API, se buscan los llamados por hacer, que son los siguientes:

-Para obtener la latitud y longitud de Maldonado. Ver Figura 2.16

GET ▼ Send ▼

Params • Authorization Headers (6) Body Scripts Settings Cookies

Query Params

<input checked="" type="checkbox"/>	Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/>	q	Maldonado,UY-MA,Uruguay			
<input checked="" type="checkbox"/>	limit	5			
<input checked="" type="checkbox"/>	appid	9cccd6e24e3754cff44dd4c96aed2041			
<input checked="" type="checkbox"/>	country	UY			
	Key	Value	Description		

Figura 2.16

-Luego para obtener la temperatura en grados centígrados, en la segunda llamada pasarle los parámetros de latitud y longitud. Figura 2.17

GET ▼ Send ▼

Params • Authorization Headers (6) Body Scripts Settings Cookies

Query Params

<input checked="" type="checkbox"/>	Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/>	lat	-34.9087162			
<input checked="" type="checkbox"/>	lon	-54.9582718			
<input checked="" type="checkbox"/>	appid	9cccd6e24e3754cff44dd4c96aed2041			
<input checked="" type="checkbox"/>	units	metric			
	Key	Value	Description		

Response ▼

Figura 2.17

Por último, en quicktype.io se coloca el JSON que se obtuvo como resultado, y te lo traduce a un Modelo. Dentro del proyecto en Visual Studio, se crea una clase en Models, se coloca el resultado ahí. Luego se crea un Controller, para que efectúe la llamada, quedando de la siguiente manera:

```
public class ApiClimaController : Controller
{
    0 referencias
    public IActionResult GetClima()
    {
        var client = new RestClient("https://api.openweathermap.org/data");
        var request = new RestRequest("/2.5/weather?lat=-34.9087162&lon=-54.9582718&appid=9cccd6e24e3754cff44dd4c96aed2041&units=metric");
        RestResponse response = client.ExecuteGet(request);
        ApiClimaModel clima = JsonConvert.DeserializeObject<ApiClimaModel>(response.Content);

        ViewBag.ElClima = clima;

        return View("Index", clima); // Asegúrate de que "Index" sea el nombre correcto de tu vista
    }
}
```

2.2.8 PAGOS

Los pagos se realizan como una clase, la cual adquiere sus atributos mediante sus gets y sets con su relación con otras clases, como Reserva, Cliente, Restaurante, etc, manteniendo también un registro de la fecha que se efectuó, diferentes métodos de pago, permitiendo una tasa de cambio en caso del uso de otra moneda y relacionándose con el Clima, del cual también tendrá que calcular descuentos según el mismo.

```
public partial class Pago
{
    13 referencias
    public int Id { get; set; }

    10 referencias
    public int ReservaId { get; set; }

    12 referencias
    public int ClienteId { get; set; }

    10 referencias
    public int ClimaId { get; set; }

    14 referencias
    public double Monto { get; set; }

    13 referencias
    public DateTime FechaPago { get; set; }

    13 referencias
    public string? MetodoPago { get; set; }

    12 referencias
    public double? TasaCambio { get; set; }

    13 referencias
    public string? Moneda { get; set; }

    12 referencias
    public double? MontoConvertido { get; set; }

    14 referencias
    public double? PrecioTotal { get; set; }

    10 referencias
    public virtual Cliente Cliente { get; set; } = null!;

    10 referencias
    public virtual Clima Clima { get; set; } = null!;

    10 referencias
    public virtual Reserva Reserva { get; set; } = null!;
}
```

Figura 2.16

2.2.9 DESCUENTOS

Los descuentos tienen dos maneras de calcularlos en el sistema, uno de ellos es mediante una función en la clase Cliente, que indica según el tipo de cliente, el descuento que se le efectúa en el pago. Ver ejemplo Figura 2.17

```

2 Referencias
public double CalcularDescuento(double montoOriginal)
{
    double descuento = 0;

    switch (TipoCliente)
    {
        case "VIP":
            descuento = 0.20; // 20% de descuento
            break;
        case "Frecuente":
            descuento = 0.10; // 10% de descuento
            break;
        case "Nuevo":
        default:
            descuento = 0.0; // Sin descuento
            break;
    }

    double montoDescontado = montoOriginal * descuento;
    return montoOriginal - montoDescontado;
}

```

Figura 2.17

3. RESULTADOS

A continuación, se colocan diferentes pruebas para demostrar el funcionamiento, y analizar y estudiar posibles mejoras en el servicio.

3.1 PRUEBAS CRUD

3.1.1 CREATE

Se pone a prueba la función de crear una reserva, la cual solicita un ID de cliente, la mesa que elige y el Restaurante, si no selecciona una fecha, se le indica que es un campo requerido para la reserva.

Create
Reserva

ClientId
4

MesaId
1

RestaurantId
3

FechaReserva
dd/mm/aaaa --:--

The FechaReserva field is required.

Estado

Create

Prueba 3.1

Queda registrada la reserva y el estado cambia a Confirmada cuando se verifica su disponibilidad.

3.1.2 READ

Esta prueba se realizó en la pestaña Menu, y eligiendo uno de los platos agregados previamente. La función devuelve su información detallada en la pantalla.

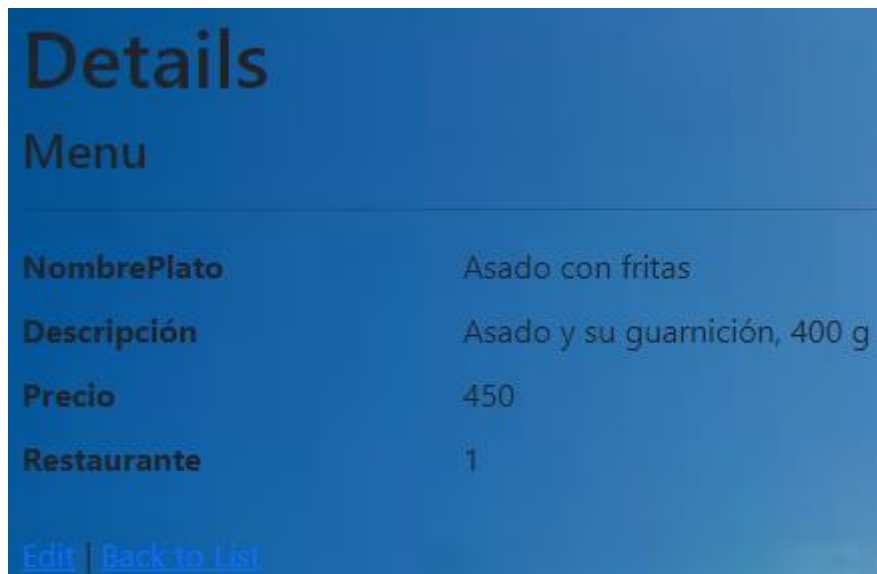


Figura 3.2

3.1.3 UPDATE

Edita a Pedro, para que pase de cliente Nuevo, a cliente VIP.



Se abre el modo edición haciendo click en Edit, y se modifica en la plantilla el campo TipoCliente a "VIP", luego se pulsa Save para ejecutar los cambios.

Pedro pasa a ser cliente VIP. Figura 3.3

The screenshot shows the 'Edit Cliente' form with the following fields:

- Usuarioid**: 12
- Nombre**: Pedro
- Email**: pedro@pepe.com
- TipoCliente**: VIP

At the bottom of the form, there is a **Save** button.

Pedro	pedro@pepe.com	VIP
-------	----------------	-----

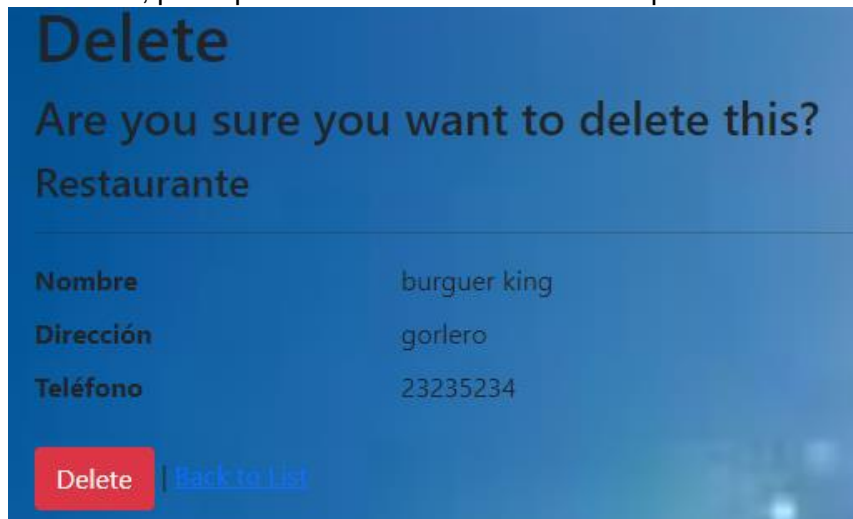
Figura 3.3

3.1.4 DELETE

Se pone a prueba el eliminar un Restaurante.

Nombre	Dirección	Teléfono	
Mc Donalds	roman guerra	25324542	Edit Details Delete
la barra	santa teresa	2354523	Edit Details Delete
burguer king	gorlero	23235234	Edit Details Delete

Aparece una alerta, para que se confirme la eliminación o puede volver hacia atrás.



Se confirma la eliminación, y borra los datos de la tabla y la base de datos. Figura 3.4

Nombre	Dirección	Teléfono	
Mc Donalds	roman guerra	25324542	Edit Details Delete
la barra	santa teresa	2354523	Edit Details Delete

Figura 3.4

3.2 PRUEBAS DE DESCUENTOS

En estas pruebas, se agregó un pago de un cliente VIP (4), para chequear que hiciera el descuento correspondiente. Luego un cliente Frecuente (5) y un cliente nuevo (6).

En los clientes VIP el servicio hace un 20% de descuento del monto, en los clientes Frecuentes un 10% y los nuevos no tienen descuento. Se puede ver el correcto funcionamiento en el Precio Total. Figura 3.5

Index									
Create New									
Monto	FechaPago	MetodoPago	TasaCambio	Moneda	MontoConvertido	PrecioTotal	Cliente	Clima	Reserva
10000	21/6/2024 15:35:00	Efectivo	2	USD	30	8000	3	1	4
4566	8/7/2024 20:15:00	Efectivo		USD	4555	3652,8	4	1	4
200000	8/7/2024 20:53:00	Efectivo		USD	0	160000	4	1	4
10000	8/7/2024 21:50:00	Efectivo				9000	5	1	4
10000	8/7/2024 21:51:00	Efectivo				10000	6	1	4

Figura 3.5

3.3 PRUEBAS DE MANEJO DE PESTAÑAS

Primera prueba que se realizó, fue intentar acceder sin logearse, a la pestaña Restaurantes, y aunque cambiaba la dirección de búsqueda, no permitía ver la pestaña.

Figura 3.6

Pestaña Usuarios

En la pestaña usuarios, aparece una tabla, con los nombres, email, contraseña y rol, lo cual definirá que permisos tiene, y también botones para editar, eliminar o crear un usuario nuevo. Figura 3.7

Index				
Create New				
Nombre	Email	Contraseña	Rol	
admin	admin@admin.com	admin	3	Edit Details Delete
Cliente	cliente@cliente.com	cliente	4	Edit Details Delete

Figura 3.7

-Se intenta desde admin, agregar un nuevo usuario.

Create

Usuario

Nombre

nuevo

Email

nuevo@nuevo.com

Contraseña

nuevq

Roll

Administrador

Create

[Back to List](#)

-Se logra añadir un usuario Nuevo con éxito. Figura 3.8

Index

[Create New](#)

Nombre	Email	Contraseña	Rol	
admin	admin@admin.com	admin	3	Edit Details Delete
Cliente	cliente@cliente.com	cliente	4	Edit Details Delete
nuevo	nuevo@nuevo.com	nuevo	3	Edit Details Delete

Figura 3.8

Pestaña Pagos

En la pestaña Pagos, aparece una tabla con los registros previos, que contiene el monto original, la fecha de pago, el método, etc. También muestra el precio total, implementando el descuento si es necesario. Figura 3.9

Index

[Create New](#)

Monto	FechaPago	MetodoPago	TasaCambio	Moneda	MontoConvertido	PrecioTotal	Cliente	Clima	Reserva	
1000	21/6/2024 15:35:00	Efectivo	2	USD	30	13123	3	1	4	Edit Details Delete
4566	8/7/2024 20:15:00	Efectivo		USD	4555	3652,8	4	1	4	Edit Details Delete

Figura 3.9

Si se intenta editar, la página muestra otro cuadro con inputs editables para las modificaciones, ofreciendo la posibilidad de confirmar las modificaciones, o volver hacia atrás y dejarlas sin efecto. Tanto el monto como el pago total pueden ser modificados.



Edit

Pago

Reservaid

4

Clienteld

3

ClímaId

1

Monto

10000

FechaPago

21/06/2024 15:35

MetodoPago

Efectivo

Figura 3.10

4. CONCLUSIONES

Durante el desarrollo de este sistema de gestión para un restaurante, se pudo aplicar de manera efectiva diferentes puntos para el logro del mismo. Se mencionan a continuación:

Complejidad y Funcionalidad Integral:

El proyecto aborda diversas funcionalidades, desde la gestión de restaurantes, como diferentes tipos de clientes, menús, hasta métodos de pago. Esto indica una implementación integral que puede ser útil para una cadena de restaurantes.

Interfaz de Usuario Amigable:

Dado que hay varias páginas para diferentes aspectos de la aplicación (restaurantes, login, clientes, reservas, reseñas), es esencial que la interfaz de usuario sea intuitiva y fácil de navegar. Una interfaz de usuario amigable contribuirá a una mejor experiencia del usuario.

Seguridad con Página de Login:

La página de login indica que se ha tenido en cuenta la seguridad, diferenciando entre usuarios y administradores. La implementación de roles y permisos adicionales para los administradores es crucial para garantizar la seguridad y la privacidad de la información.

Registro de Clientes y Detalles de la orden:

La página de clientes recopila información valiosa, y la página de la orden solicitan datos específicos relacionados con estas transacciones. Un seguimiento detallado de la información de los clientes y las transacciones es esencial para una gestión eficiente.

Listado de Mesas Disponibles:

La página de listado de alquileres, clasificando entre en Disponible, Ocupada o Confirmada, brinda una visión clara del estado actual de las transacciones. Esto facilita la gestión y el seguimiento de las reservas de mesa.

Posibles Mejoras:

Considera implementar funcionalidades adicionales como recordatorios automáticos para mesas con prioridad por atraso, generación de informes sobre balances en cierto periodo de tiempo, y una interfaz administrativa más robusta.

Optimización de Rendimiento:

A medida que la base de datos crece, la optimización de consultas y la gestión eficiente de la base de datos son críticas para mantener un rendimiento óptimo de la aplicación.

5. REFERENCIAS

- [1] <https://aws.amazon.com/es/what-is/net/>
- [2] <https://learn.microsoft.com/es-es/dotnet/framework/data/adonet/ado-net-overview>
- [3] <https://learn.microsoft.com/es-es/visualstudio/get-started/visual-studio-ide?view=vs-2022>
- [4] <https://bsw.es/que-es-c/>
- [5] <https://learn.microsoft.com/es-es/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver16>
- [6] <https://render2web.com/net-6/que-es-database-first/>
- [7] <https://desarrolloweb.com/articulos/que-es-mvc.html>
- [8] <https://learn.microsoft.com/es-es/devops/develop/git/what-is-git>
- [9] <https://www.xataka.com/basics/que-github-que-que-le-ofrece-a-desarrolladores>
- [10] <https://desarrolloweb.com/home/entity-framework>
- [11] <https://www.ibm.com/mx-es/topics/api#>