Arquitectura de la Solución

Arquitectura en Capas

La solución se estructurará en varias capas para promover la separación de responsabilidades y facilitar el mantenimiento:

- **Capa de Presentación (Front-end):**
  Se desarrollará una aplicación web ASP.NET Core Web App MVC que permita a los usuarios interactuar con el sistema a través de formularios y vistas. Esta capa se encargará de mostrar la información de los restaurantes, procesar formularios para crear, editar y eliminar.
- **Capa de Negocio:**
  Se implementará la lógica de validación y la persistencia de datos. Aquí se aplicarán validaciones de datos y manejo de errores.
- **Capa de Datos:**
  Se utilizará Entity Framework Core SQL Server para interactuar con una base de datos SQL Server. Se diseñará la tabla Restaurante con campos como Id, Nombre, Dueño, Provincia, Cantón, Distrito, Dirección exacta.

Exposición del API

Se implementará un controlador para las API que expondrán endpoints para realizar operaciones CRUD sobre la entidad Restaurante. Estos endpoints responderán en formato JSON, permitiendo que aplicaciones externas consuman la información o incluso que el propio front-end utilice llamadas AJAX para refrescar datos sin recargar la página completa.

Tecnologías y Herramientas

- **Framework:** ASP.NET Core Web App (Modelo-Vista-Controlador).
- **Motor de Base de Datos:** SQL Server Management Studio.
- **IDE:** Microsoft Visual Studio 2022.
- **Frontend:** HTML, JavaScript.
- **Nugets:** EntityFrameworkCore v8.0.13, EntityFrameworkCore.Desig v8.0.13, EntityFrameworkCore.Tools v8.0.13, EntityFrameworkCore.SqlServer v8.0.13.

Procesos y Procedimientos

Análisis y Diseño

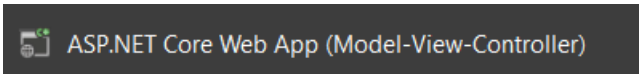
1. **Recolección de Requisitos:**

- Se definen los campos obligatorios para cada restaurante (nombre, dueño, provincia, cantón, distrito y dirección exacta.).
- Se establecen los procesos de crear, editar y eliminar.
- Se determinar los escenarios de consumo del API.

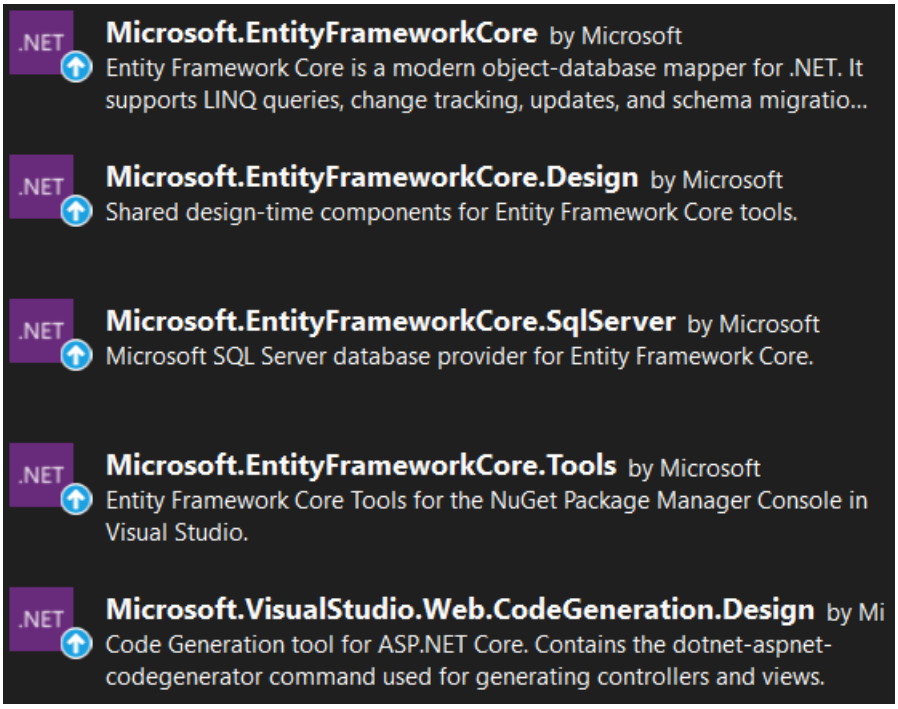
2. **Diseño de la Base de Datos:**
   - Se crea el modelo de datos con la entidad Restaurante.
   - Se genera el script SQLpara crear la tabla correspondiente.
3. **Diseño de la Arquitectura:**
   - Se utilizará ASP.NET Core MVC para crear páginas web con vistas.
   - Se utilizará el controlador MVC las vistas y el common Api para los endpoints.

Desarrollo

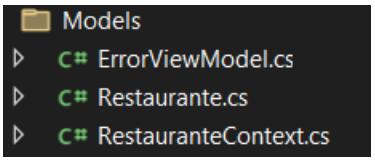
1. **Configuración del Proyecto:**
   - Se crea un nuevo proyecto ASP.NET Core Web App (Modelo-Vista-Controlador).



   - Se instalan los Nugets: EntityFrameworkCore, EntityFrameworkCore.Desig, EntityFrameworkCore.Tools, EntityFrameworkCore.SqlServer .



   - Se define el modelo Restaurante y el RestauranteContext.

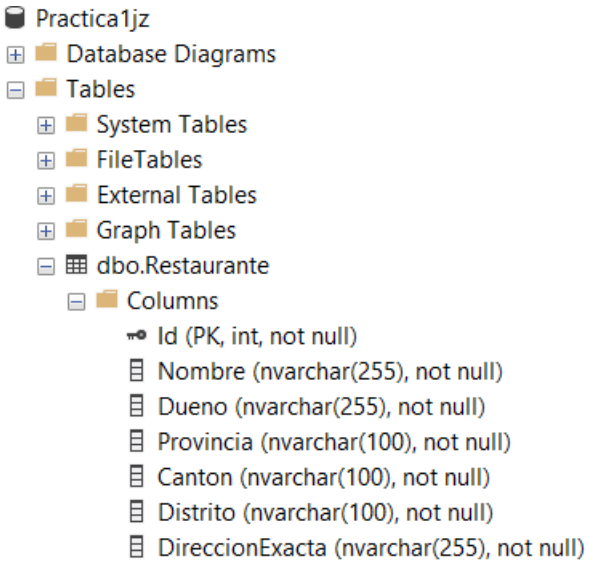


- Se configurar el acceso a la base de datos en appsettings.json y registrar el RestauranteContext en Program.cs.

```
"ConnectionStrings": {
  "Practica1jz": "Data Source=MSI\\SQLEXPRESS; Initial Catalog=Practica1jz;Integrated Security=True;Connect Timeout=30
```

```
builder.Services.AddDbContext<RestauranteContext>(op =>
{
    op.UseSqlServer(builder.Configuration.GetConnectionString("Practica1jz"));
});
```

- Se crea la Database y tabla Restaurante en SQL Server.



- Se crea el controlador con los métodos (GET, POST, DELETE) para el manejo de las vistas Index, Crear, Editar y Eliminar.

```
public class RestaurantesController : Controller
{
    private readonly RestauranteContext _context;
    0 references
    public RestaurantesController(RestauranteContext context)...
    [HttpGet]
    0 references
    public async Task<IActionResult> Index()...
    [HttpGet]
    0 references
    public IActionResult Create()...
    0 references
    public IActionResult Create(Restaurante restaurantes)...
    0 references
    public IActionResult Edit(int? id)...
    [HttpPost]
    0 references
    public IActionResult Edit(Restaurante restaurantes)...
    0 references
    public ActionResult Delete(int? id)...
    [HttpPost, ActionName("Delete")]
    [ValidateAntiForgeryToken]
    0 references
    public ActionResult DeleteConfirmado(int id)...
}
```

- Se crear un controlador API que exponga métodos HTTP (GET, POST, PUT, DELETE) para el consumo de datos y obtener la información.

```
    [Route("api/[controller]")]
    [ApiController]
    1 reference
    public class RestaurantesApiController ...
0 references
public static class RestauranteEndpoints
{
    0 references
    public static void MapRestauranteEndpoints (this IEndpointRouteBuilder routes)
    {
        var group = routes.MapGroup("/api/Restaurante").WithTags(nameof(Restaurante));
        group.MapGet("/", () =>...)
        .WithName("GetAllRestaurantes")
        .WithOpenApi();
        group.MapGet("/{id}", (int id) =>...)
        .WithName("GetRestauranteById")
        .WithOpenApi();
        group.MapPut("/{id}", (int id, Restaurante input) =>...)
        .WithName("UpdateRestaurante")
        .WithOpenApi();
        group.MapPost("/", (Restaurante model) =>...)
        .WithName("CreateRestaurante")
        .WithOpenApi();
        group.MapDelete("/{id}", (int id) =>...)
        .WithName("DeleteRestaurante")
        .WithOpenApi();
    }
}}
```

- Se diseñan las vistas con MVC View para mostrar la información y formularios para la entrada de datos.

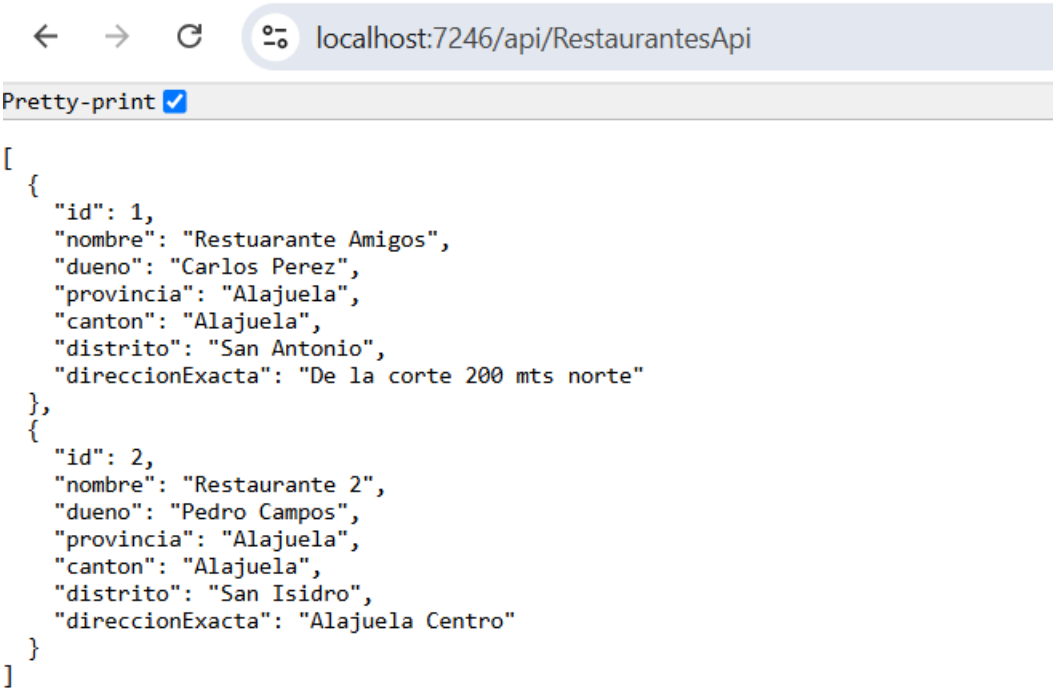
- Se utiliza JavaScript para consumir el API y actualizar la vista de forma dinámica.

```
<script>

    const apiUrl = 'https://localhost:7246/api/RestaurantesApi';


    async function obtenerRestaurantes() {
        try {
            const response = await fetch(apiUrl);
            if (!response.ok) {
                throw new Error('Error al obtener los datos');
            }
            const data = await response.json();

            const lista = document.getElementById('restaurantesList');
            data.forEach(restaurante => {
                const li = document.createElement('li');
                li.textContent = `Nombre: ${restaurante.nombre}, Dueño: ${restaurante.dueno},
                Provincia: ${restaurante.provincia}, Cantón: ${restaurante.canton},
                Distrito: ${restaurante.distrito}`;
                lista.appendChild(li);
            });
        } catch (error) {
            console.error('Error:', error);
        }
    }
    obtenerRestaurantes();
</script>
```

Resultados

Vista de los restaurantes MVC y API.

Resultado de GetAllRestaurantes.



Resultado de GetRestauranteById.