

# RETO PROGRAMACIÓN OS

## Caso

Outsourcing SA tiene una campaña telefónica nueva para cliente “SALONES EMPRESARIALES XYZ”; el servicio consiste en atender llamadas de clientes potenciales en las que desean información para reservar salones para eventos de sus empresas.

Se requiere hacer una solución web que utiliza un agente de Call Center para capturar la solicitud de información la aplicación debe capturar datos del cliente, información general del evento.

## Ejercicio

1. Construir una base de datos que soporte la aplicación.
2. Construir una solución web con los siguientes requerimientos:
  - 2.1. Formulario para la creación, actualización y borrado de clientes con los siguientes datos:
    - Identificación.
    - Nombre y apellidos.
    - Teléfono.
    - Correo.
    - Departamento (Lista Desplegable).
    - Ciudad (Lista Desplegable Dependiente).
    - Edad.
  - 2.2. Formulario para la creación, actualización y borrado de reservas con los siguientes datos:
    - Cliente.
    - Fecha de evento.
    - Cantidad de personas.
    - Motivo (lista desplegable de: Evento empresarial, despedida empresa, desayuno comercial, almuerzo).
    - Observaciones.
    - Estado (Confirmado, No confirmado).
  - 2.3. Construir un módulo de reportes con filtros propuestos por el desarrollador de las reservas con exportación a archivos.

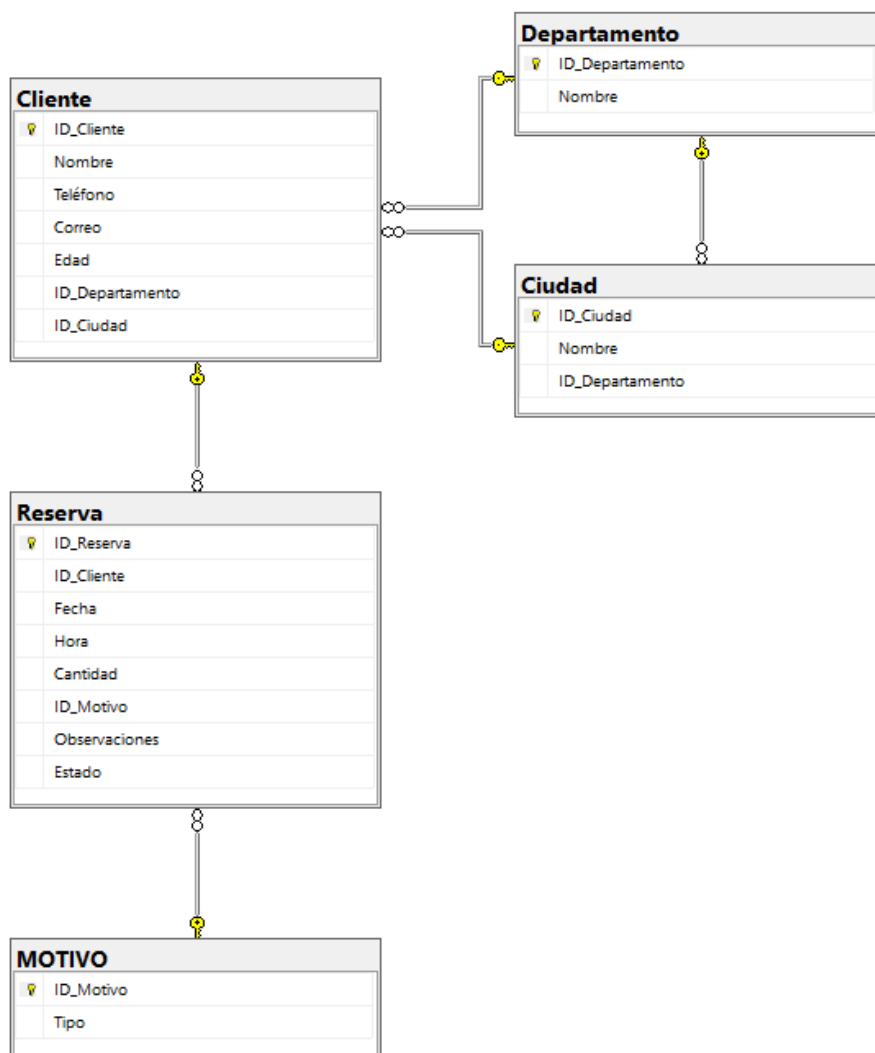
## Documentación

### 1. Desarrollo base de datos.

La empresa SALONES EMPRESARIALES XYZ opera en tres departamentos de Colombia, los cuales son Antioquia, Boyacá y Cundinamarca, donde posee salones de eventos en las ciudades más importantes de dichos departamentos, por lo tanto, se habilita la posibilidad de reservar en tales ciudades.

Revisar ANEXO Script base de datos.

La estructura de la base de datos queda de la siguiente forma:



Se anexan 3 Departamentos:

- Antioquia, Boyacá y Cundinamarca

Y se anexan las ciudades con mayor densidad poblacional de cada departamento y donde actúa la empresa.

- Apartadó, Bello, Caldas, Cauca, Chigorodó, Envigado, Itagüí, Medellín, Rionegro y Turbo.
- Chiquinquirá, Garagoa, Duitama, Moniquirá, Paipa, Puerto Boyacá, Samacá, Sogamoso, Tunja y Villa de Leyva.
- Bogotá, Chía, Facatativá, Funza, Fusagasugá, Girardot, Madrid, Mosquera, Soacha y Zipaquirá.

## 2. Creación Proyecto MVC.

(Database first)

Se crea el proyecto y se vincula la base de datos de la siguiente forma:

- 
- Scaffold-DbContext "Server=SEBASTIAN\WINCC; Database=Reto; Trusted\_Connection=True; TrustServerCertificate=True;" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models
- 

Se agregan las siguientes dependencias:

- 
- Microsoft.EntityFrameworkCore.SqlServer (7.0.1)
  - Microsoft.EntityFrameworkCore.Tools (7.0.1)
  - Rotativa.AspNetCore (1.2.0)
-

Se realiza la inyección de dependencias de la base de datos, primero generando la conexión en el archivo “appsettings.json”.

```
"ConnectionStrings": {  
  "Conection": " Server=SEBASTIAN\\WINCC; Database=Reto;  
Trusted_Connection=True;  
TrustServerCertificate=True;"  
}
```

Luego, en el archivo “Program.cs” se realiza la inyección de dependencias.

```
builder.Services.AddDbContext<RetoContext>(options =>  
{  
  options.UseSqlServer(builder.Configuration.GetConnectionString("Conection"));  
});
```

### 3. CRUD Cliente.

#### 3.1. Create.

Para realizar el CRUD de la tabla cliente, primero se genera el controlador, que se ubica en el archivo “ClienteController.cs”. Una vez en el archivo, se crea el constructor del objeto Cliente y el contexto del mismo, que le relaciona con la base de datos de su tabla respectiva.

```
private readonly RetoContext _context;

//Constructor
public ClienteController(RetoContext context)
{
    _context = context;
}

});
```

Luego, se tiene el método Index asíncrono, que permite listar los clientes y los datos de los mismos.

```
//Manipulacion de la base de datos de forma asincrona
public async Task<IActionResult> Index()
{
    var clidept = _context.Clientes
        .Include(c => c.Ciudad)
        .Include(c => c.Departamento);

    return View(await clidept.ToListAsync());
}
```

El método Create permite registrar un nuevo cliente, este método cuenta con dos tipos, uno de tipo GET y otro de tipo POST, el primero configura los datos y configuración que se debe presentar en la página; el segundo método, consiste en la transferencia de información y datos, estos dos métodos están presentes para las tres acciones que se realizan sobre el cliente, que son la creación, edición y eliminación.

Este método trae información de las tablas “Departamento” y “Cliente”, donde lista la información que tiene cada tabla de acuerdo a su identificador. Sin embargo, toma el identificador y el objeto se refiere al campo “Nombre” de cada tabla, lo que permite mostrar la información y no sólo un número.

```

public IActionResult Create()
{
    ViewData["Departamentos"] = new SelectList(_context.Departamentos, "DepartamentoId", "Nombre");
    ViewData["Ciudades"] = new SelectList(_context.Ciudades, "CiudadId", "Nombre");
    return View();
}

```

Ahora, para el envío de información por medio del método de tipo POST, se crea un 'ViewModel' de tipo Cliente, llamado "ClienteViewModel.cs", donde se establecen todos los campos y requerimientos de cada uno. Luego, el método valida cada campo y envía la información a la base de datos, de ahí, se plasman y listan en la vista indice.

```

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create(ClienteViewModel model)
{
    if (ModelState.IsValid)
    {
        var cli = new Cliente()
        {
            ClienteId = model.ClienteId,
            Nombre = model.Nombre,
            Teléfono = model.Telefono,
            Correo = model.Correo,
            Edad = model.Edad,
            DepartamentoId = Int32.Parse(model.DepartamentoId.ToString()),
            CiudadId = Int32.Parse(model.CiudadId.ToString()),
        };
        _context.Add(cli);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    ViewData["Departamentos"] = new SelectList(_context.Departamentos, "DepartamentoId", "Nombre");
    ViewData["Ciudades"] = new SelectList(_context.Ciudades, "CiudadId", "Nombre");
    return View();
}

```

Finalmente, se crea la vista para realizar el registro de clientes, esta vista se llama "Create.cshtml", en esta vista se utiliza nuevamente el ViewModel de cliente, se crean el formulario donde se recibirá la información pertinente para cada cliente y se modifica su estética por medio de Bootstrap 5.

### 3.2. Edit.

Para realizar la edición de cada cliente, se genera un botón de editar en la vista “Index.cshtml”, que redirige a la vista “Edit.cshtml”, donde se aplican dos métodos, uno de tipo GET y otro de tipo POST. El primero, coloca la información del cliente en un formulario a partir de su identificador y presenta la configuración realizada en un comienzo, así permite realizar las modificaciones pertinentes.

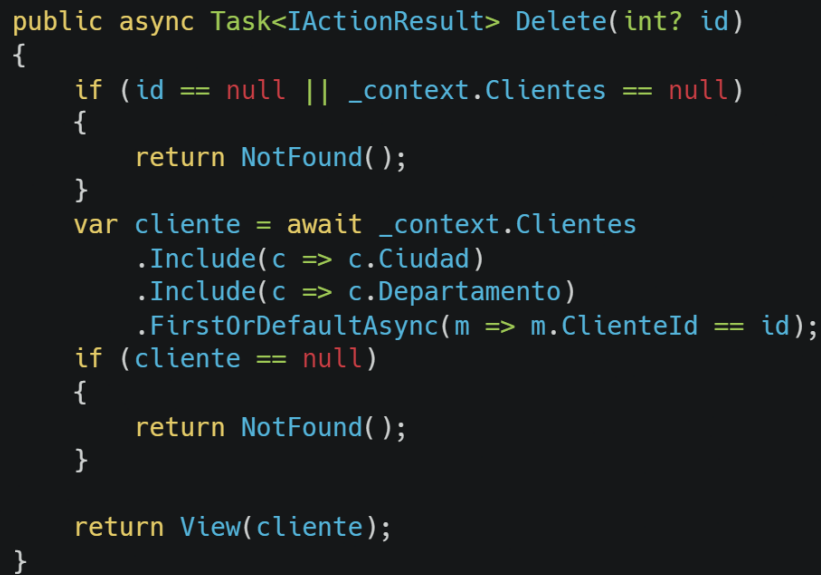
```
public async Task<IActionResult> Edit(int id)
{
    var cliente = await _context.Clientes.FindAsync(id);
    if (cliente == null)
    {
        return NotFound();
    }
    ViewData["Ciudades"] = new SelectList(_context.Ciudades, "CiudadId", "Nombre", cliente.CiudadId);
    ViewData["Departamentos"] = new SelectList(_context.Departamentos, "DepartamentoId", "Nombre", cliente.DepartamentoId);
    return View(cliente);
}
```

Luego, el método post, permite la conexión con la base de datos y modifica los registros que se adecuaron.

```
[HttpPost]
public async Task<IActionResult> Edit(int id, [Bind("ClienteId,Nombre,Teléfono,Correo,Edad,DepartamentoId,CiudadId")] Cliente cliente)
{
    _context.Update(cliente);
    await _context.SaveChangesAsync();
    ViewData["Ciudades"] = new SelectList(_context.Ciudades, "CiudadId", "Nombre", cliente.CiudadId);
    ViewData["Departamentos"] = new SelectList(_context.Departamentos, "DepartamentoId", "Nombre", cliente.DepartamentoId);
    return RedirectToAction(nameof(Index));
}
```

### 3.3. Delete.

Finalmente, se tienen los métodos de eliminación de registros respecto a la tabla cliente, por lo tanto, se genera un botón de eliminación junto a cada registro en la tabla índice del apartado de cliente, de esta manera, se generan dos métodos para realizar la eliminación de registros. Nuevamente se utilizan los métodos de tipo GET y POST. El primer método realiza una búsqueda en la base de datos para confirmar que el registro existe, de esta manera, genera la información de dicho registro en una nueva vista, esta vista se llama “Delete.cshtml”.



```
public async Task<IActionResult> Delete(int? id)
{
    if (id == null || _context.Clientes == null)
    {
        return NotFound();
    }
    var cliente = await _context.Clientes
        .Include(c => c.Ciudad)
        .Include(c => c.Departamento)
        .FirstOrDefaultAsync(m => m.ClienteId == id);
    if (cliente == null)
    {
        return NotFound();
    }

    return View(cliente);
}
```

Y el segundo método se encarga de confirmar la eliminación del registro seleccionado, primero confirmando que el registro existe y luego eliminándolo de la base de datos.



```

[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DeleteConfirmed(int id)
{
    if (_context.Cientes == null)
    {
        return Problem("Entity set 'RetoContext.Cientes' is null.");
    }
    var cliente = await _context.Cientes.FindAsync(id);
    if (cliente != null)
    {
        _context.Cientes.Remove(cliente);
    }

    await _context.SaveChangesAsync();
    return RedirectToAction(nameof(Index));
}

private bool ClienteExists(int id)
{
    return _context.Cientes.Any(e => e.ClienteId == id)
}

```

Finalmente, se muestra la vista del resultado del CRUD del apartado Cliente.

## Cientes

Registrar

Buscar identificación:

Identificación	Nombre	Teléfono	Correo	Edad	Departamento	Ciudad	Opciones
2	Bolivar	3209178579	si@gmail.com	1			<input type="button" value="Editar"/> <input type="button" value="Delete"/>
1234567	Angie	3209178579	si@gmail.com	5			<input type="button" value="Editar"/> <input type="button" value="Delete"/>
123351176	Sebastian	3209	si@gmail.com	23			<input type="button" value="Editar"/> <input type="button" value="Delete"/>
123456789	Angie	3209178579	si@gmail.com	50			<input type="button" value="Editar"/> <input type="button" value="Delete"/>
147852369	a	3209178579	si@gmail.com	16			<input type="button" value="Editar"/> <input type="button" value="Delete"/>

### 4. CRUD Reserva.

Para realizar este apartado, se utilizan funciones similares, para la creación, edición y eliminación de registros. En este caso se genera un ViewModel llamado "ReservaViewModel" donde se plasman todas las propiedades que necesita el modelo.

Y la vista índice, en la cual se listan las reservas realizadas es la siguiente.

## Reservas

Registrar

Buscar número de reserva:

# Reserva	Cliente	Fecha	Cantidad de personas	Motivo	Observaciones	Estado	Opciones	Informe
0	1234567	1/5/2023 12:00:00 AM	1		no	Activo	<input type="button" value="Editar"/>   <input type="button" value="Borrar"/>	<input type="button" value="Imprimir"/>
1	123351176	1/5/2023 12:00:00 AM	1		no	Activo	<input type="button" value="Editar"/>   <input type="button" value="Borrar"/>	<input type="button" value="Imprimir"/>
2	123351176	1/5/2023 12:00:00 AM	1		no	Activo	<input type="button" value="Editar"/>   <input type="button" value="Borrar"/>	<input type="button" value="Imprimir"/>
3	1234567	1/6/2023 12:00:00 AM	1		no 1	Activo	<input type="button" value="Editar"/>   <input type="button" value="Borrar"/>	<input type="button" value="Imprimir"/>

### 5. Reportes.

Para realizar los reportes del apartado de reservas, se utiliza la extensión “Rotativa”, la cual permite la creación de archivos pdf, por lo tanto, en la vista índice se genera un botón llamado “Imprimir”, el cual cuenta con un método llamado imprimir.

En este caso, se crea un nuevo ViewModel, llamado “JoinViewModel”, en el cual se agregan propiedades de la tabla Cliente y Reserva, de esta manera, plasmar los datos necesarios teniendo en cuenta el identificador del cliente y así generar su respectivo reporte de la reserva en formato PDF.


```
public IActionResult Imprimir(int idreserva,
[Bind("ReservaId,ClienteId,Fecha,Cantidad,MotivoId,Observaciones,Estados")] Reserva resv)
{
    //LÓGICA HACIA BASE DE DATOS
    JoinViewModel modelo = _context.Reservas.Include(dv => dv.Cliente).Where(v => v.ReservaId == idreserva)
        .Select(v => new JoinViewModel()
        {
            ReservaId = Int32.Parse(v.ReservaId.ToString()),
            ClienteId = Int32.Parse(v.ClienteId.ToString()),
            ClienteNombre = v.Cliente.Nombre,
            Ciudad = v.Cliente.Ciudad.Nombre,
            Fecha = v.Fecha,
            Cantidad = v.Cantidad,
            MotivoId = v.Motivo.Tipo,
            Observaciones = v.Observaciones,
            Estado = (bool)v.Estado,
        }).FirstOrDefault();

    ViewData["Clientes"] = new SelectList(_context.Clientes, "ClienteId", "Nombre", resv.ClienteId);
    ViewData["Motivos"] = new SelectList(_context.Motivos, "MotivoId", "Tipo", resv.MotivoId);

    return new ViewAsPdf("Imprimir", modelo)
    {
        FileName = $"Reserva.pdf",
        PageOrientation = Rotativa.AspNetCore.Options.Orientation.Portrait,
        PageSize = Rotativa.AspNetCore.Options.Size.A4
    };
}
```

De esta manera, se crea un objeto de tipo JoinViewModel, el cual se utilizan sus propiedades para asignar a los campos de las tablas Reserva y Cliente, así plasmar los datos en el informe.

Finalmente, al dar click sobre el botón “Imprimir” se genera el archivo PDF, llamado “Reserva.PDF”. Se puede observar un ejemplo del archivo generado a continuación.



**NÚMERO RESERVA**  
2

**SALONES EMPRESARIALES XYZ**  
Direccion: Av. Amarilis Park 123  
Correo: salonesempresariales@example.com

**CLIENTE**  
Sebastian  
123351176

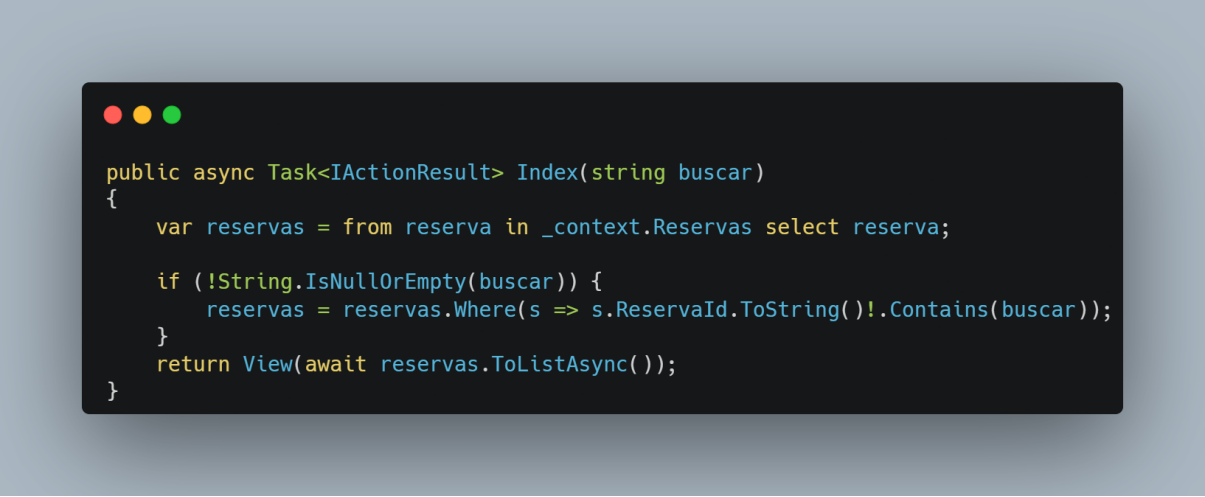
Fecha del evento:	1/5/2023 12:00:00 AM
Ciudad:	Apartadó
Cantidad de Personas:	1
Motivo:	Desayuno Comercial
Observaciones:	no
Estado:	Activo

## 6. Filtros.

Se realizaron filtros, uno para cada apartado, donde se utiliza un buscador por identificador. Entonces, se genera un buscador en la parte superior de cada vista

índice de los apartados, donde se escribe el identificador solicitado, de esta forma, arroja la información de dicho identificador.

El método utilizado para realizar la búsqueda o filtro es el siguiente, se ubica en el método Index de cada apartado.

A screenshot of a code editor with a dark background and light-colored text. The code is written in C# and represents the Index method of a controller. It starts with a public async Task<IActionResult> Index(string buscar) method signature. Inside the method, it declares a variable reservas = from reserva in \_context.Reservas select reserva;. Then, it has an if (!String.IsNullOrEmpty(buscar)) block where it filters the reservas using Where(s => s.ReservaId.ToString().Contains(buscar)). Finally, it returns View(await reservas.ToListAsync());. The code is enclosed in curly braces.

```
public async Task<IActionResult> Index(string buscar)
{
    var reservas = from reserva in _context.Reservas select reserva;

    if (!String.IsNullOrEmpty(buscar)) {
        reservas = reservas.Where(s => s.ReservaId.ToString().Contains(buscar));
    }
    return View(await reservas.ToListAsync());
}
```

A partir del valor dado por el cliente, el servidor realiza la búsqueda en la base de datos y así plasma el cliente solicitado o todas las reservas hechas por un cliente.