

1. Base de Datos


1.1. Descripción General

- **Nombre de la Base de Datos:** EmployeeDB
- **Propósito:** Almacenar información relacionada con los empleados, incluyendo sus datos personales, posición, oficina y salario.

1.2. Tablas

1.2.1. Tabla: Employee

- **Columnas:**
 - **Id** (int, Primary Key): Identificador único para cada empleado.
 - **Name** (string): Nombre del empleado.
 - **Position** (string): Posición o cargo del empleado.
 - **Office** (string): Ubicación u oficina del empleado.
 - **Salary** (decimal, Nullable): Salario del empleado.

	Column Name	Data Type	Allow Nulls
	Id	int	<input type="checkbox"/>
	Name	nvarchar(50)	<input checked="" type="checkbox"/>
	Position	nvarchar(50)	<input checked="" type="checkbox"/>
	Office	nvarchar(50)	<input checked="" type="checkbox"/>
	Salary	decimal(18, 0)	<input checked="" type="checkbox"/>
			<input type="checkbox"/>

1.3. Relaciones

- No aplica (solo una tabla).

2. Modelo

2.1. Descripción General

- **Nombre del Modelo:** *Employee*
- **Propósito:** Representar la entidad de un empleado en la aplicación y mapearla a la tabla *Employee* en la base de datos.

```
9 referencias
public partial class Employee
{
    5 referencias
    public int Id { get; set; }
    2 referencias
    public string Name { get; set; }
    2 referencias
    public string Position { get; set; }
    2 referencias
    public string Office { get; set; }
    2 referencias
    public Nullable<decimal> Salary { get; set; }
}
```

Propiedades:

- *Id*: Identificador único.
- *Name*: Nombre del empleado.
- *Position*: Cargo del empleado.
- *Office*: Oficina donde trabaja el empleado.
- *Salary*: Salario del empleado, es un valor opcional.

2.3. Relación con la Base de Datos

- El modelo *Employee* está directamente relacionado con la tabla *Employee* en la base de datos. Cada propiedad del modelo corresponde a una columna en la tabla.

3. Controlador

3.1. Descripción General

- **Nombre del Controlador:** `EmployeeController`
- **Propósito:** Gestionar las solicitudes HTTP relacionadas con las operaciones CRUD de los empleados.

3.2. Métodos Clave

- **`Index()`**: Muestra la vista principal de gestión de empleados.
- **`GetEmployees()`**: Retorna todos los empleados en formato JSON para su uso en el DataTable.

```
public JsonResult GetEmployees() //Traer todos los Employees
{
    List<Employee> myEmployees = new List<Employee>(); //Creacion de lista del tipo de
    modelo a usar.

    //Modelo de entidades creado por entity framework que accede a la base de datos.
    using (EmployeeDBEntities db = new EmployeeDBEntities())
    {
        myEmployees = (from Employee in db.Employee
                        select Employee).ToList(); //Consulta LINQ para llamar los datos y guardarlos
        en la lista.

        return Json(new {data= myEmployees }, JsonRequestBehavior.AllowGet); //Para el
        dataTable es necesario pasarle un atributo llamado data
    }
}
```

- **GetEmployee(int id):** Retorna un empleado específico en formato JSON basado en su ID.
-

```
public JsonResult GetEmployee(int id) { //Traer 1 solo Employee
```

```
Employee myEmployee = new Employee();
```

```
using (EmployeeDBEntities db = new EmployeeDBEntities())
```

```
{
```

```
    myEmployee = (from Employee in db.Employee
```

```
        where Employee.Id == id //Consulta LINQ para obtener un Employee segun  
                                el Id
```

```
        select Employee).FirstOrDefault(); //Seleccione el primero que encuentre.
```

```
}
```

```
    return Json( myEmployee , JsonRequestBehavior.AllowGet); //se retorna lo obtenido de  
la consulta. aqui ya no es necesario mandar el atributo, sino el objeto.
```

```
}
```

- **UpdateEmployees(Employee oEmployee):** Actualiza o crea un empleado en la base de datos.
-

```
public JsonResult UpdateEmployees(Employee oEmployee) //oEmployee es el objeto que  
recibe desde el metodo ajax.
```

```
{
```

```
    bool respuesta = true; //Controla si fue satisfactoria la operación o no.
```

```

try
{
    if (oEmployee.Id == 0) //Si el id es ==0 Es un nuevo Employee
    {

        using (EmployeeDBEntities db = new EmployeeDBEntities())
        {
            db.Employee.Add(oEmployee); //Se añade el objeto a la DDBB
            db.SaveChanges(); //Guardar los cambios efectuados.
        }
    }
    else
    {
        using (EmployeeDBEntities db = new EmployeeDBEntities())
        {
            Employee tempEmployee = (from Employee in db.Employee
                                     where Employee.Id == oEmployee.Id
                                     select Employee).FirstOrDefault(); //Consulta LINQ para obtener
un Employee según el Id

            tempEmployee.Name = oEmployee.Name;
            tempEmployee.Position = oEmployee.Position; //Se añaden los valores a la DB.
            tempEmployee.Office = oEmployee.Office;
            tempEmployee.Salary = oEmployee.Salary;
            db.SaveChanges(); //Es IMPORTANTE tener bien configurada la PK.

        }
    }
}

```

```

    }
}
catch
{
    respuesta = false;
}

return Json(new { resultado = respuesta }, JsonRequestBehavior.AllowGet); //Nos
devuelve true si todo sale bien o false si existe alguna excepción
}

```

- **DeleteEmployee(int id)**: Elimina un empleado de la base de datos basado en su ID.
-

```

public JsonResult DeleteEmployee(int id) {

    bool respuesta = true; //Nos muestra si es satisfactoria o no la operación.

    try
    {
        using (EmployeeDBEntities db= new EmployeeDBEntities())
        {
            Employee oEmployee = new Employee();

            oEmployee= (from Employee in db.Employee.Where(e => e.Id == id)
//Manera distinta de obtener el Employee por el Id (Funcion flecha o labda)
            select Employee).FirstOrDefault();
            db.Employee.Remove(oEmployee); //Eliminacion del objeto
            db.SaveChanges(); //Guardar cambios.
        }
    }
    catch (Exception)
    {
        respuesta=false;
        throw;
    }
    return Json(new { resultado = respuesta }, JsonRequestBehavior.AllowGet);
}

```

4. jQuery

4.1. Descripción General

- **Propósito:** Manejar la lógica de la interfaz de usuario, incluyendo la carga dinámica de datos, manejo de eventos y operaciones CRUD (Crear, Leer, Actualizar, Eliminar).

4.2. Estructura del Script Principal

- **Archivo:** `EmployeeManagement.js`
- **Descripción:** Este script se ejecuta en el lado del cliente y es responsable de interactuar con la interfaz de usuario y el backend para gestionar la información de los empleados.

4.3. Funciones Clave

- `$(document).ready(function () {...});` :: Inicializa el DataTable y configura los botones y eventos.

```
var table_Employee //Donde se guarda la tabla.

$(document).ready(function () {

    table_Employee = $("#myTable").DataTable({

        "ajax": {

            "url": '@Url.Action("GetEmployees", "Employee")', // La URL a la que se enviará la
solicitud para poder recibir los datos.

            "type": 'GET', // El tipo de solicitud: 'GET', 'POST', etc...En este caso GET

            "dataType": 'json', // El tipo de datos que retorna la función en este caso Json

        },

        "columns": [

            { "data": "Name" },

            { "data": "Position" },

            { "data": "Office" }, //Columnas a mostrar

            { "data": "Salary" },
```

```

{
  "data": "Id",

  "render": function (data) {

    return "<button class='btn btn-primary btn-sm' onClick='abrirModulo(" + data +
    ")'><i class='fas fa-pen'></i></button>" + //La variable data es el Id.

    "<button class='btn btn-danger btn-sm' onClick='Eliminar(" + data + ")'><i
    class='fas fa-trash'></i></button>";

  },

  "orderable": false,

  "searchable": false,

  "width": "150px"

}

],

dom: 'Bftrtip',

buttons: [

  {

    text: 'Add new',

    attr: { class: "btn btn-success btn-md" },// estilos del btn actualizar.

    action: function (e, dt, node, config) {

      abrirModulo(0); //abrir el modulo al dar click se le pasa por defecto 0 para saber
      que es uno nuevo.

    }

  }

],

});

});

```

- `abrirModulo(id)`: Abre el modal para agregar o editar un empleado.

```
function abrirModulo(id) {  
  
    $('#txtId').val(id)  
  
    $('#formModal').modal('show'); //Abriendo el formulario  
  
    if (id != 0) {  
        jQuery.ajax({  
            url: '@Url.Action("GetEmployee", "Employee")' + "?Id=" + id, //Se obtiene los datos  
            por el Id  
            type: "GET",  
            dataType: "json",  
            contentType: "application/json; charset=utf-8",  
            success: function (data) {  
  
                console.log(data) //Revisar si efectivamente estan llegando los datos.  
  
                if (data != null) {  
                    $('#txtId').val(data.Id);  
                    $('#txtName').val(data.Name);  
                    $('#txtPosition').val(data.Position); //Luego de que se obtienen se pintan en los  
campos correspondientes.  
                    $('#txtOffice').val(data.Office);  
                    $('#txtSalary').val(data.Salary);  
                }  
            }  
        })  
    }  
}
```

```

    } else
    {
        $("#txtName").val("");

        $("#txtPosition").val(""); // si es 0 es porque se va a agregar uno nuevo, por lo que se
colocan los campos en blanco.

        $("#txtOffice").val("");

        $("#txtSalary").val("");
    }
}

```

-
- **Eliminar(id)**: Maneja la eliminación de un empleado seleccionado.
-

```

function Eliminar(id) {

    if (confirm("Realmente desea eliminar?")) { //Pregunta al usuario si en realidad desea
eliminar

        console.log("Si")

        jQuery.ajax({

            url: '@Url.Action("DeleteEmployee", "Employee")' + "?Id=" + id, //Se le pasa al
controlador el Id a eliminar.

            type: "GET",

            dataType: "json",

            contentType: "application/json; charset=utf-8",

            success: function (data) {

                if (data.resultado) {

                    table_Employee.ajax.reload();

                }

            }

        })
    }
}

```

```

    })

    } else {

        console.log("No")

    }

}

```

-
- **Save()**: Guarda o actualiza la información del empleado.
-

```

function Save() {

    var $data = {
        oEmployee: {
            Id: parseInt($("#txtId").val()), // Se realiza un casting o parcerizacion para convertir
            el texto en int.
            Name: $("#txtName").val(),
            Position: $("#txtPosition").val(),
            Office: $("#txtOffice").val(),
            Salary: $("#txtSalary").val(),
        }
    }

    console.log($data); //Saber si en realidad se estan guardando los datos en el json

    jQuery.ajax({
        url: '@Url.Action("UpdateEmployees", "Employee")', //llamada al metodo actualizar del
        controlador.
        type: 'POST',
        data: JSON.stringify($data), //Convierte un json en texto.
        dataType: "json",
        contentType: "application/json; charset=utf-8",

        success: function (data) {
            console.log(data.resultado);
            if (data.resultado) { //Valor obtenido de la llamada al controlador.
                table_Employee.ajax.reload(); //Recargar luego de guardar la informacion.
                $('#formModal').modal('hide'); //Oculte el modal luego de guardar.
            } else {
                alert("No se pudo guardar los cambios."); //Si algo sale mal.
            }
        },
    },

```

```
error: function (error) {  
    console.log(error)  
  
    },  
    beforeSend: function () {  
  
    }  
    });  
}
```

4.5. Integración con Bootstrap

- **Uso de Bootstrap:** Los botones y modales en el DataTable utilizan clases de Bootstrap para un estilo consistente y responsivo.