educación IT
CODE YOUR FUTURE

Programación Web .NET Core

Módulo 3

educación IT
CODE YOUR FUTURE

ADO.NET:
Plantillas para Vistas 1

Acceso a datos con ADO .NET: Plantillas para vistas

En esta etapa, se abordará el mismo escenario, pero con las herramientas de *Visual Studio .NET* para generar las vistas automáticamente.

1. Crearemos una nueva tabla en la misma base de datos, como se muestra a continuación:


Results Messages

	Codigo	Nombre	Apellido	Email
1	1	Marina	Togneri	mtogneri@hotmail.com
2	2	Gustavo	Herrera	gustavo.herrera@yahoo.com
3	3	Pedro	Martinez	pmartinez@gmail.com

```
use Comercio
go

createtableClientes(
Codigointidentityprimarykey,
Nombre varchar(25)notnull,
Apellido varchar(25)notnull,
Email varchar(50)notnull
)
go

insertinto Clientes values
('Marina','Togneri','mtogneri@hotmail.com'),
('Gustavo','Herrera','gustavo.herrera@yahoo.com'),
('Pedro','Martinez','pmartinez@gmail.com')
go

select*from Clientes
go
```

2. En la carpeta ***Models*** agregamos dos clases:
    - La clase **`Cliente`** representará el modelo de datos (coincidente con la estructura de la tabla ***Clientes*** creada en SQL Server).

¡Veámoslo en las siguientes diapositivas!

```
usingSystem;
usingSystem.Collections.Generic;
usingSystem.Linq;
usingSystem.Web;

namespaceAccesoADatos.Models
{
publicclassCliente
    {
publicintCodigo{ get; set; }
publicstring Nombre { get; set; }
publicstring Apellido { get; set; }
publicstring Email { get; set; }
    }
}
```

...

```
usingSystem;
usingSystem.Collections.Generic;
usingSystem.Configuration;
usingSystem.Data;
usingSystem.Data.SqlClient;
usingSystem.Linq;
usingSystem.Web;

namespaceAccesoADatos.Models
{
publicclassAdmCliente
    {
privateSqlConnectionconexion;

privatevoidConectar()
        {
stringstringConexion = ConfigurationManager.ConnectionStrings["Conexion"].ToString();
conexion = newSqlConnection(stringConexion);
        }
```

...

```
publicintAlta(Cliente pCliente)
        {
Conectar();
SqlCommand sentencia = newSqlCommand("insertinto Clientes(Nombre, Apellido, Email)
values (@nombre, @apellido, @email)", conexion);

sentencia.Parameters.Add("@nombre", SqlDbType.VarChar);
sentencia.Parameters.Add("@apellido", SqlDbType.VarChar);
sentencia.Parameters.Add("@email", SqlDbType.VarChar);

sentencia.Parameters["@nombre"].Value = pCliente.Nombre;
sentencia.Parameters["@apellido"].Value = pCliente.Apellido;
sentencia.Parameters["@email"].Value = pCliente.Email;

conexion.Open();

int i = sentencia.ExecuteNonQuery();

conexion.Close();

return i;
        }
```

...

```
publicList<Cliente>TraerTodos()
        {
Conectar();
List<Cliente> clientes = newList<Cliente>();

SqlCommand sentencia = newSqlCommand("selectcodigo, nombre, apellido, email from
clientes", conexion);

conexion.Open();

SqlDataReader registros = sentencia.ExecuteReader();

while (registros.Read())
            {
                Cliente cliente = new Cliente
                {
Codigo = int.Parse(registros["codigo"].ToString()),
                    Nombre = registros["nombre"].ToString(),
                    Apellido = registros["apellido"].ToString(),
                    Email = registros["email"].ToString()
                };

clientes.Add(cliente);
            }
```

...

...

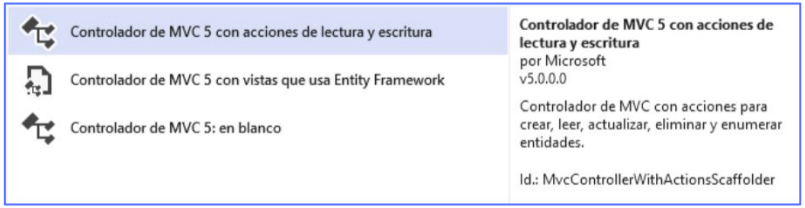
```
conexion.Close();

return clientes;
        }

public Cliente TraerCliente(intpCodigo)
        {
Conectar();

SqlCommand sentencia = newSqlCommand("selectcodigo, nombre, apellido, email from
clientes wherecodigo=@codigo", conexion);

sentencia.Parameters.Add("@codigo", SqlDbType.Int);
sentencia.Parameters["@codigo"].Value = pCodigo;

conexion.Open();

SqlDataReader registros = sentencia.ExecuteReader();

            Cliente cliente = newCliente();

if (registros.Read())
            {
cliente.Codigo = int.Parse(registros["codigo"].ToString());
cliente.Nombre = registros["nombre"].ToString();
cliente.Apellido = registros["apellido"].ToString();
cliente.Email = registros["email"].ToString();
            }
```

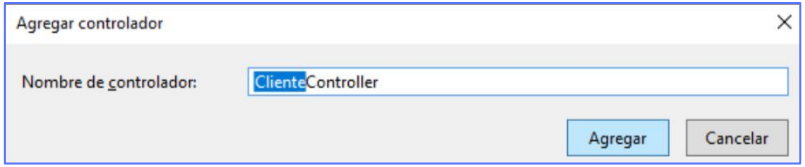
...

```
...
conexion.Close();

return cliente;
        }

publicintModificar(Cliente pCliente)
        {
Conectar();

SqlCommand sentencia = newSqlCommand("update clientes set nombre=@nombre,
apellido=@apellido, email=@email wherecodigo=@codigo", conexion);

sentencia.Parameters.Add("@nombre", SqlDbType.VarChar);
sentencia.Parameters.Add("@apellido", SqlDbType.VarChar);
sentencia.Parameters.Add("@email", SqlDbType.VarChar);

sentencia.Parameters["@nombre"].Value = pCliente.Nombre;
sentencia.Parameters["@apellido"].Value = pCliente.Apellido;
sentencia.Parameters["@email"].Value = pCliente.Email;

conexion.Open();

int i = sentencia.ExecuteNonQuery();

conexion.Close();

return i;
        }
...
```

```
publicintBorrar(intpCodigo)
        {
Conectar();

SqlCommand sentencia = newSqlCommand("deletefrom clientes wherecodigo=@codigo",
conexion);

sentencia.Parameters.Add("@codigo", SqlDbType.Int);
sentencia.Parameters["@codigo"].Value = pCodigo;

conexion.Open();

int i = sentencia.ExecuteNonQuery();

conexion.Close();

return i;
        }
    }
}
```

3. A continuación, crearemos el ***Controlador*** como hicimos en los ejemplos anteriores.
4. Al controlador principal lo llamaremos ***Home***, para esto presionamos el botón derecho del mouse sobre la carpeta ***Controllers*** y seleccionamos ***Agregar > Controlador...***
5. Seleccionamos del diálogo la opción denominada ***Controlador de MVC 5 con acciones de lectura y escritura***, como podemos ver en la imagen de la derecha.





Visual Studio .NET generará la estructura básica de nuestro controlador con todas las acciones que deberemos implementar:

```
usingSystem;
usingSystem.Collections.Generic;
usingSystem.Linq;
usingSystem.Web;
usingSystem.Web.Mvc;

namespaceAccesoADatos.Controllers
{
publicclassClienteController :Controller
    {
// GET: Cliente
publicActionResultIndex()
        {
returnView();
        }
```

...

...

```
// GET: Cliente/Details/5
publicActionResultDetails(int id)
        {
returnView();
        }

// GET: Cliente/Create
publicActionResultCreate()
        {
returnView();
        }

// POST: Cliente/Create
        [HttpPost]
publicActionResultCreate(FormCollectioncollection)
        {
try
            {
// TODO: Addinsertlogichere

returnRedirectToAction("Index");
            }
```

...

```
...

catch
            {
returnView();
            }
        }

// GET: Cliente/Edit/5
publicActionResultEdit(int id)
        {
returnView();
        }

// POST: Cliente/Edit/5
        [HttpPost]
publicActionResultEdit(int id, FormCollectioncollection)
        {
try
            {
// TODO: Addupdatelogichere

returnRedirectToAction("Index");
            }

...
```

...

```
catch
            {
returnView();
            }
        }

// GET: Cliente/Delete/5
publicActionResultDelete(int id)
        {
returnView();
        }

// POST: Cliente/Delete/5
        [HttpPost]
publicActionResultDelete(int id, FormCollectioncollection)
        {
try
            {
// TODO: Adddeletelogichere

returnRedirectToAction("Index");
            }
```

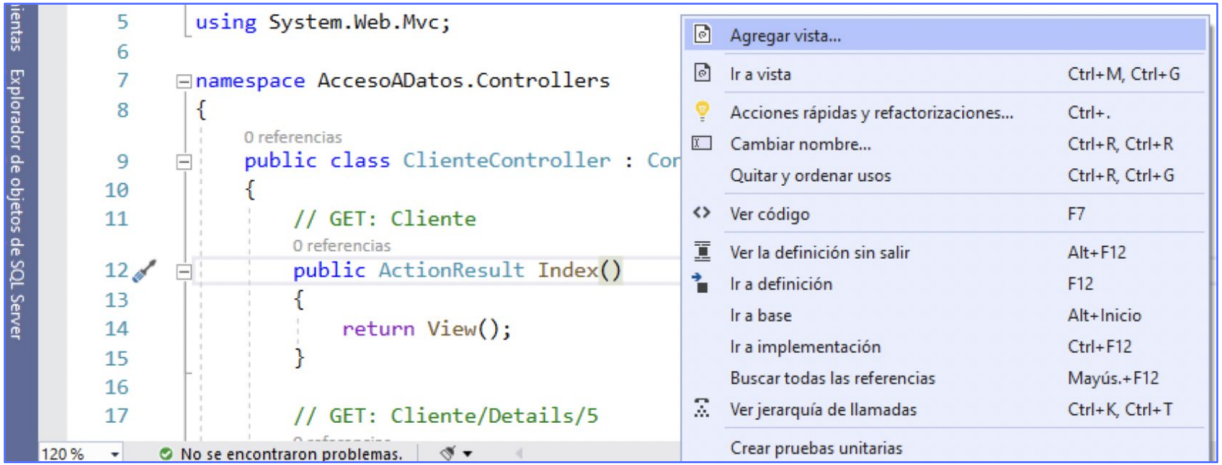
...

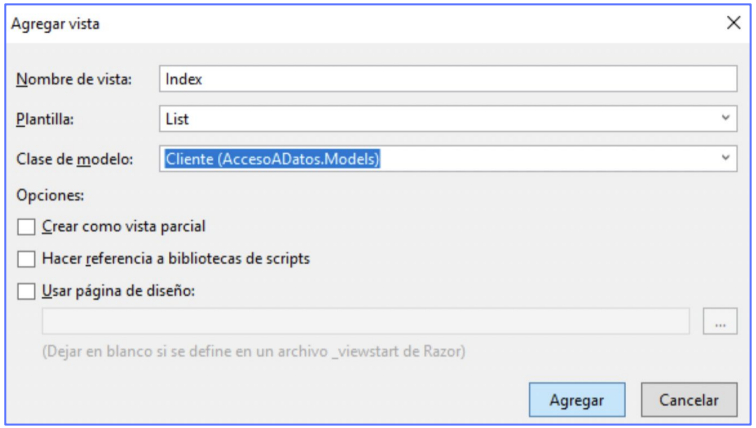
```
...
catch
            {
returnView();
            }
        }
    }
}
```

Se han generado las 8 acciones que se deben relacionar con el código implementado en el ***Modelo*** y la generación de las vistas.

Listado de clientes

1. Ahora deberemos implementar la acción **`Index`** del controlador **`HomeController`**:






```
usingAccesoADatos.Models;
usingSystem;
usingSystem.Collections.Generic;
usingSystem.Linq;
usingSystem.Web;
usingSystem.Web.Mvc;

namespaceAccesoADatos.Controllers
{
publicclassClienteController :Controller
    {
// GET: Cliente
publicActionResultIndex()
        {
AdmClienteobjAdmCli = newAdmCliente();
returnView(objAdmCli.TraerTodos());
        }
```

2. Crearemos un objeto de la clase **`AdmCliente`** y llamamos al método **`TraerTodos`** y se lo pasaremos como argumento a la vista.

3. La vista todavía no existe, por lo que haremos clic con el botón derecho del mouse sobre **`Index()`** y seleccionaremos ***Agregar Vista***. En la siguiente ventana, seleccionaremos como plantilla ***List*** y como clase de Modelo **`Cliente`**.

4. Ahora se ha generado el archivo ***Index.cshtml*** que tiene todo lo necesario para mostrar los datos recuperados del modelo. Pasemos a la siguiente pantalla para ver el detalle.

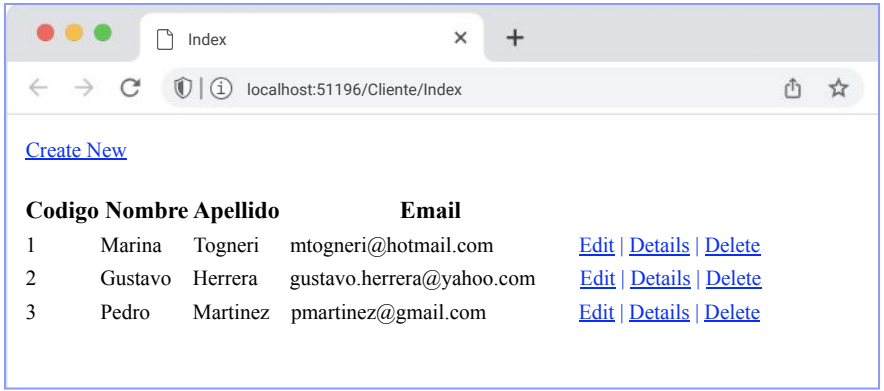
```
@model IEnumerable<AccesoADatos.Models.Cliente>

@{
Layout = null;
}

<!DOCTYPEhtml>

<html>
<head>
<metaname="viewport"content="width=device-width"/>
<title>Index</title>
</head>
<body>
<p>
@Html.ActionLink("Create New", "Create")
</p>
<tableclass="table">
<tr>
<th>
@Html.DisplayNameFor(model =>model.Codigo)
</th>
```

...

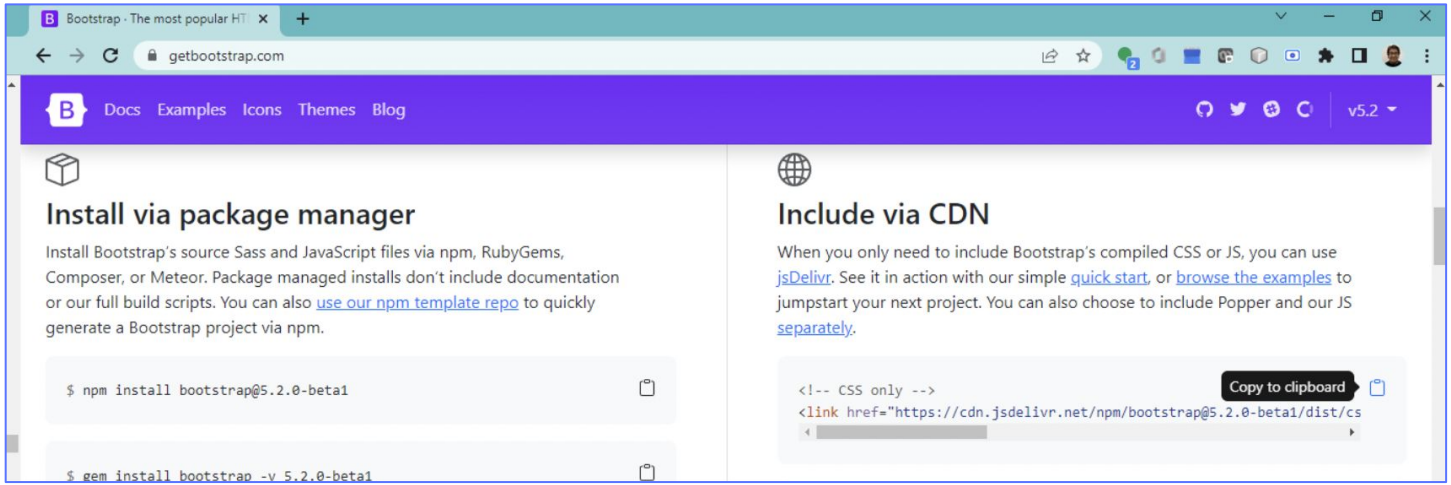
```
...
<th>
@Html.DisplayNameFor(model =>model.Nombre)
</th>
<th>
@Html.DisplayNameFor(model =>model.Apellido)
</th>
<th>
@Html.DisplayNameFor(model =>model.Email)
</th>
<th></th>
</tr>

@foreach (variteminModel) {
<tr>
<td>
@Html.DisplayFor(modelItem =>item.Codigo)
</td>
<td>
@Html.DisplayFor(modelItem =>item.Nombre)
</td>
<td>
@Html.DisplayFor(modelItem =>item.Apellido)
</td>
...
```

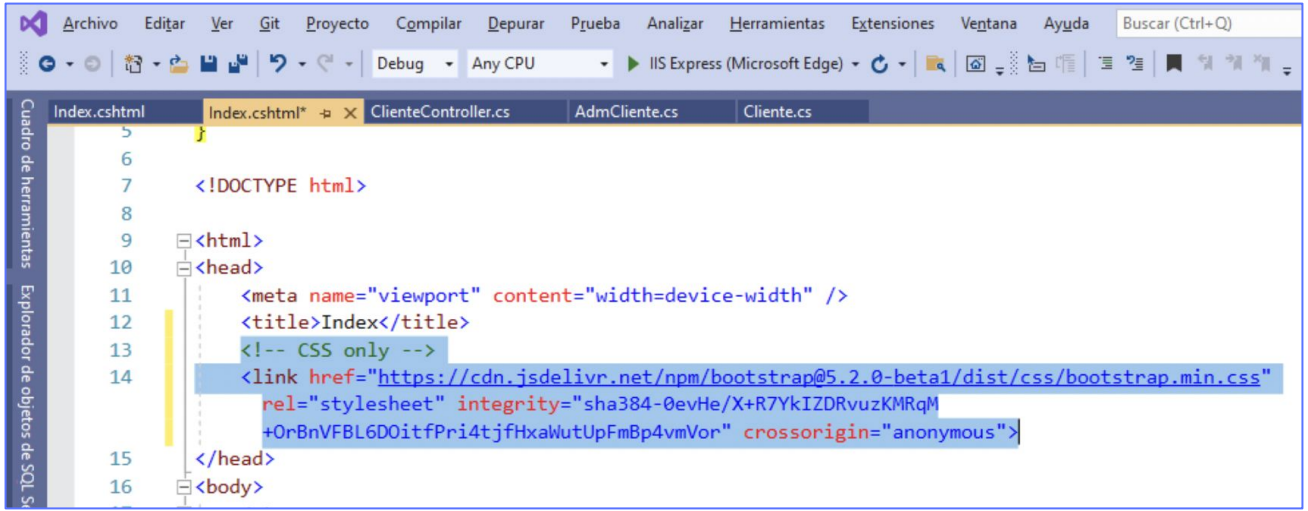
```
<td>
@Html.DisplayFor(modelItem =>item.Email)
</td>
<td>
@Html.ActionLink("Edit", "Edit", new{ /* id=item.PrimaryKey */ }) |
@Html.ActionLink("Details", "Details", new{ /* id=item.PrimaryKey */ }) |
@Html.ActionLink("Delete", "Delete", new{ /* id=item.PrimaryKey */ })
</td>
</tr>
    }

</table>
</body>
</html>
```

5. Si ejecutamos la aplicación podemos ver el listado de clientes:



Nota: Deberemos hacer algunas modificaciones en el código, para que los vínculos se muestren en español, se importe la librería *Bootstrap* y se pasen los parámetros **`id`** para la edición, borrado y detalle. Vemos lo siguiente:



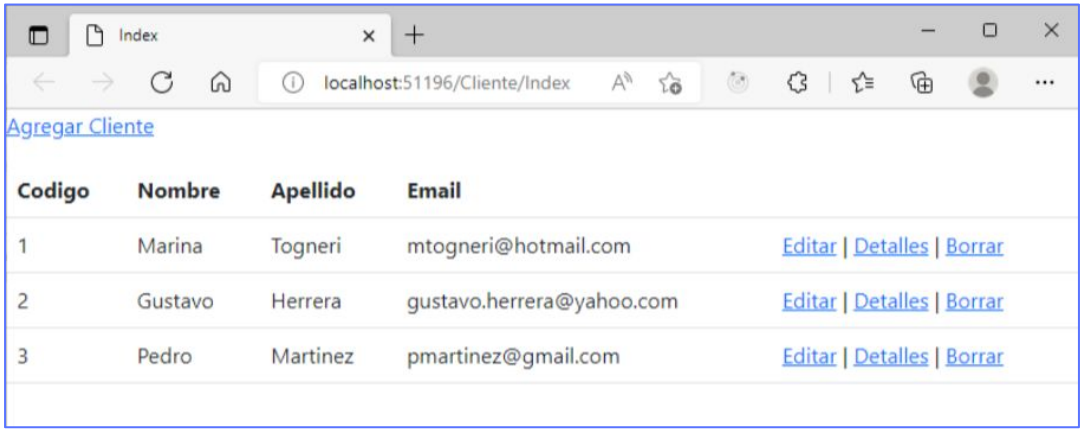

Archivo Editar Ver Git Proyecto Compilar Depurar Prueba Analizar Herramientas Extensiones Ventana Ayuda
Buscar (Ctrl+Q)
Debug
Any CPU
IIS Express (Microsoft Edge)
Index.cshtml
Index.cshtml*
ClienteController.cs
AdmCliente.cs
Cliente.cs
Cuadro de herramientas
Explorador de objetos de SQL
}
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width" />
<title>Index</title>
<!-- CSS only -->
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0-beta1/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-0evHe/X+R7YkIZDRvuzKMRqM
+OrBnVFBL6DOitfPri4tjfHxaWutUpFmBp4vmVor" crossorigin="anonymous">
</head>
<body>

```
<body>
<p>
@Html.ActionLink("Agregar Cliente", "Create")
</p>
<tableclass="table">
<tr>
<th>
@Html.DisplayNameFor(model =>model.Codigo)
</th>
<th>
@Html.DisplayNameFor(model =>model.Nombre)
</th>
<th>
@Html.DisplayNameFor(model =>model.Apellido)
</th>
<th>
@Html.DisplayNameFor(model =>model.Email)
</th>
<th></th>
</tr>
```

...

```
@foreach (variteminModel) {
<tr>
<td>
@Html.DisplayFor(modelItem =>item.Codigo)
</td>
<td>
@Html.DisplayFor(modelItem =>item.Nombre)
</td>
<td>
@Html.DisplayFor(modelItem =>item.Apellido)
</td>
<td>
@Html.DisplayFor(modelItem =>item.Email)
</td>
<td>
@Html.ActionLink("Editar", "Edit", new{ id=item.Codigo }) |
@Html.ActionLink("Detalles", "Details", new{ id=item.Codigo }) |
@Html.ActionLink("Borrar", "Delete", new{ id=item.Codigo })
</td>
</tr>
    }
</table>
</body>
```

Si se ejecuta nuevamente, vemos lo siguiente:



Como podemos ver, el generador de vistas del Visual Studio .NET utiliza elementos, llamados ***HTML helpers***. Los ***HTML helpers*** son métodos que se pueden invocar en la vista, a través de la propiedad **`HTML`**, por ejemplo:

```
@Html.ActionLink("Agregar Cliente", "Create")
```

Después, cuando se ejecuta el código en el servidor, se generará el HTML correspondiente:

```
<a href="/Home/Create">Agregar Cliente</a>
```

Nota: La utilización de los ***helpers*** que provee ASP.NET MVC evita muchos de los problemas que se pueden generar, cuando se escribe directamente el código HTML. En los módulos más avanzados del curso, se abordarán los ***HTML Helpers*** principales y cómo utilizarlos.

educación IT
CODE YOUR FUTURE

¡Sigamos trabajando!