

# Programación Web .NET Core

Módulo 3

# **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!



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace AccesoADatos.Models
{
    public class Cliente
    {
        public intCodigo { get; set; }
        public string Nombre { get; set; }
        public string Apellido { get; set; }
        public string Email { get; set; }
    }
}
```

...

```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Data;
using System.Data.SqlClient;
using System.Linq;
using System.Web;

namespace AccesoADatos.Models
{
    public class AdmCliente
    {
        private SqlConnection conexion;

        private void Conectar()
        {
            string stringConexion = ConfigurationManager.ConnectionStrings["Conexion"].ToString();
            conexion = new SqlConnection(stringConexion);
        }
    }
}
```

...

```
public int Alta(Cliente pCliente)
{
    Conectar();
    SqlCommand sentencia = new SqlCommand("insert into 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;
}
```

...

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

    SqlCommand sentencia = new SqlCommand("select codigo, 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 = new SqlCommand("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 = new Cliente();

    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;
}

public int Modificar(Cliente pCliente)
{
    Conectar();

    SqlCommand sentencia = new SqlCommand("update clientes set nombre=@nombre,
    apellido=@apellido, email=@email where codigo=@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;
}
```

...



```
public int Borrar(int pCodigo)
{
    Conectar();

    SqlCommand sentencia = new SqlCommand("delete from clientes where codigo=@codigo",
        conexion);

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

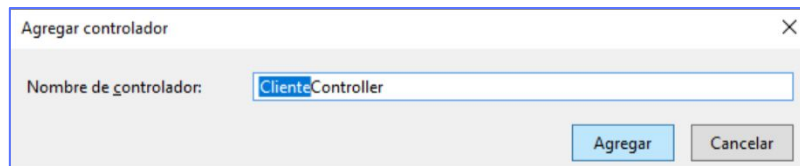
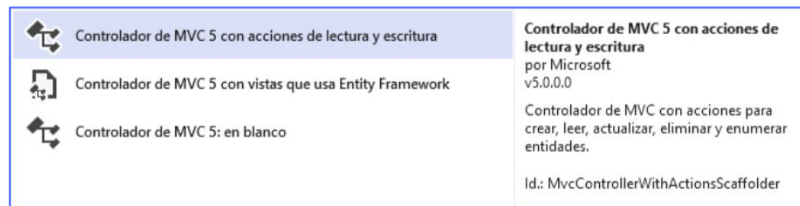
    conexion.Open();

    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:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace AccesoADatos.Controllers
{
    public class ClienteController : Controller
    {
        // GET: Cliente
        public ActionResult Index()
        {
            return View();
        }
    }
}
```



```
...  
  
// GET: Cliente/Details/5  
public ActionResult Details(int id)  
{  
    return View();  
}  
  
// GET: Cliente/Create  
public ActionResult Create()  
{  
    return View();  
}  
  
// POST: Cliente/Create  
[HttpPost]  
public ActionResult Create(FormCollection collection)  
{  
    try  
    {  
        // TODO: Add insert logic here  
  
        return RedirectToAction("Index");  
    }  
}
```

...

```
...  
  
catch  
{  
    returnView();  
}  
  
// GET: Cliente/Edit/5  
public ActionResult Edit(int id)  
{  
    returnView();  
}  
  
// POST: Cliente/Edit/5  
[HttpPost]  
public ActionResult Edit(int id, FormCollection collection)  
{  
    try  
    {  
        // TODO: Add update logic here  
  
        return RedirectToAction("Index");  
    }  
}
```

...

```
...  
  
catch  
{  
    returnView();  
}  
  
// GET: Cliente/Delete/5  
public ActionResult Delete(int id)  
{  
    returnView();  
}  
  
// POST: Cliente/Delete/5  
[HttpPost]  
public ActionResult Delete(int id, FormCollection collection)  
{  
    try  
    {  
        // TODO: Add delete logic here  
        return RedirectToAction("Index");  
    }  
}
```

...





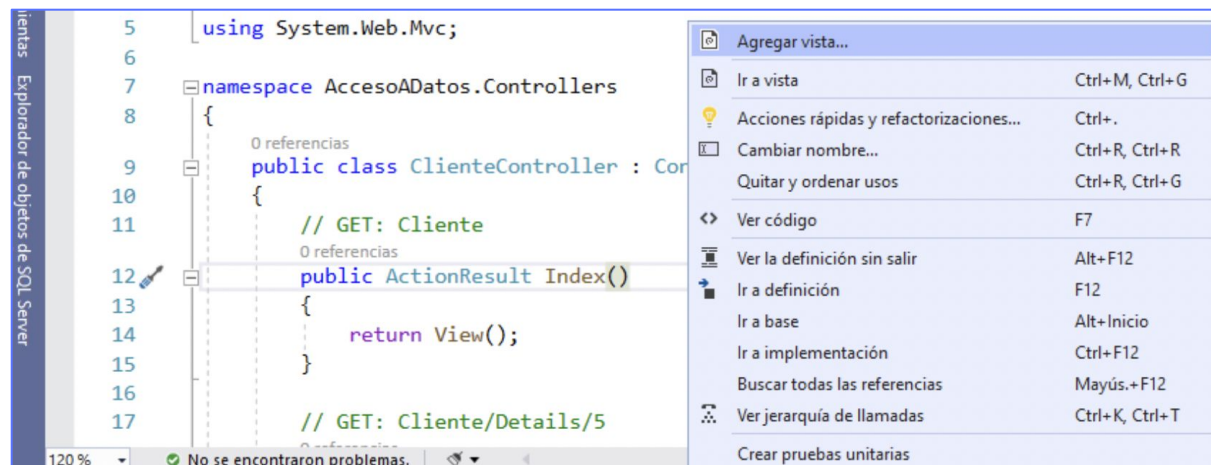
```
catch
{
    returnView();
}
```

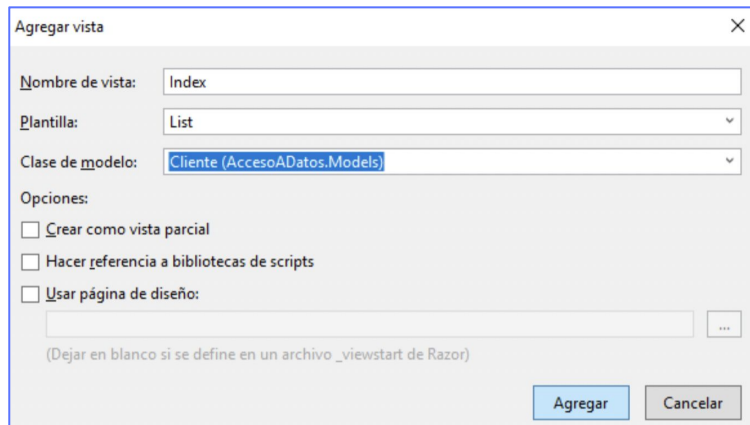


Se han generado las 8 acciones que se deben relacionar con el código implementado en el **Modelo** y la generación de las vistas.

## Listado de clientes

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





Agregar vista

Nombre de vista: Index

Plantilla: List

Clase de modelo: Cliente (AccesoADatos.Models)

Opciones:

☐ Crear como vista parcial

☐ Hacer referencia a bibliotecas de scripts

☐ Usar página de diseño:

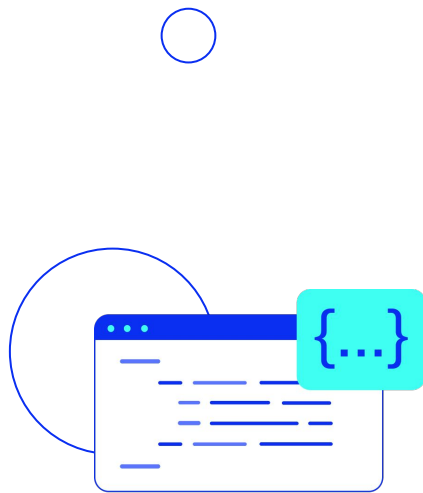
(Dejar en blanco si se define en un archivo \_viewstart de Razor)

Agregar Cancelar

```
using AccesoADatos.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace AccesoADatos.Controllers
{
    public class ClienteController : Controller
    {
        // GET: Cliente
        public ActionResult Index()
        {
            AdmCliente objAdmCli = new AdmCliente();
            return View(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>
```

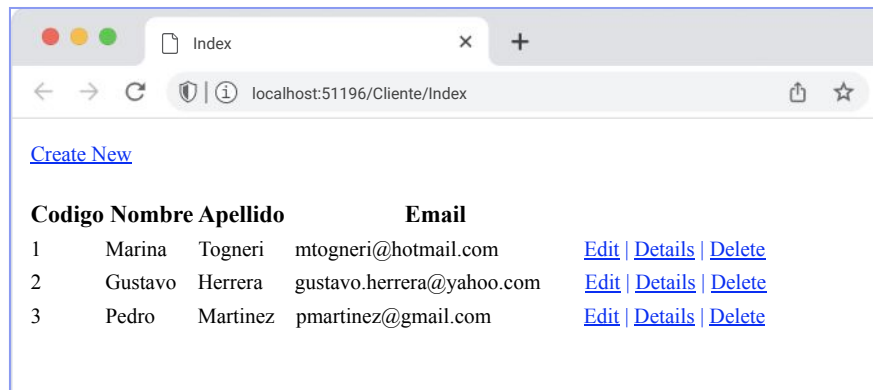
```
...  
  
<th>  
@Html.DisplayNameFor(model =>model.Nombre)  
</th>  
<th>  
@Html.DisplayNameFor(model =>model.Apellido)  
</th>  
<th>  
@Html.DisplayNameFor(model =>model.Email)  
</th>  
<th></th>  
</tr>  
  
@foreach (var item in Model) {  
    <tr>  
        <td>  
            @Html.DisplayFor(modelItem => item.Codigo)  
        </td>  
        <td>  
            @Html.DisplayFor(modelItem => item.Nombre)  
        </td>  
        <td>  
            @Html.DisplayFor(modelItem => item.Apellido)  
        </td>  
    </tr>  
}
```

...

```
<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>
    }

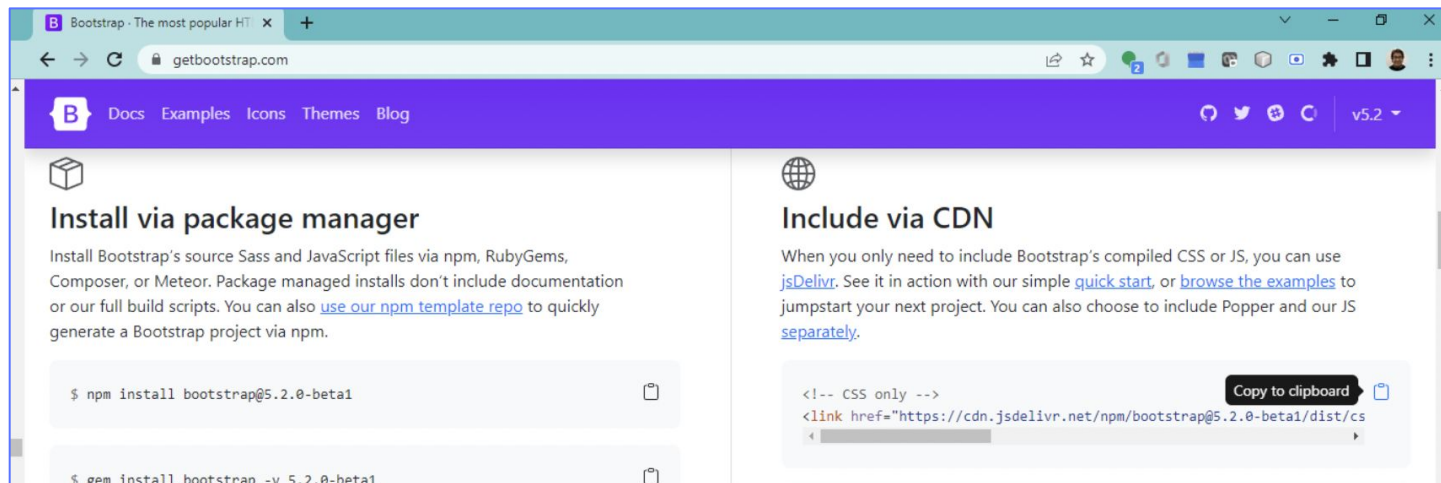
</table>
</body>
</html>
```

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:



The screenshot shows the Bootstrap 5.2.0-beta1 documentation page. The page has a purple header with the Bootstrap logo and navigation links: Docs, Examples, Icons, Themes, Blog. The main content is divided into two columns. The left column is titled 'Install via package manager' and includes instructions for installing Bootstrap via npm, RubyGems, Composer, or Meteor. It also provides a link to the npm template repo. The right column is titled 'Include via CDN' and includes instructions for including Bootstrap's compiled CSS or JS via jsDelivr. It also provides a link to the quick start and a link to browse the examples. The page is version 5.2.0-beta1.

**Install via package manager**

Install Bootstrap's source Sass and JavaScript files via npm, RubyGems, Composer, or Meteor. Package managed installs don't include documentation or our full build scripts. You can also [use our npm template repo](#) to quickly generate a Bootstrap project via npm.

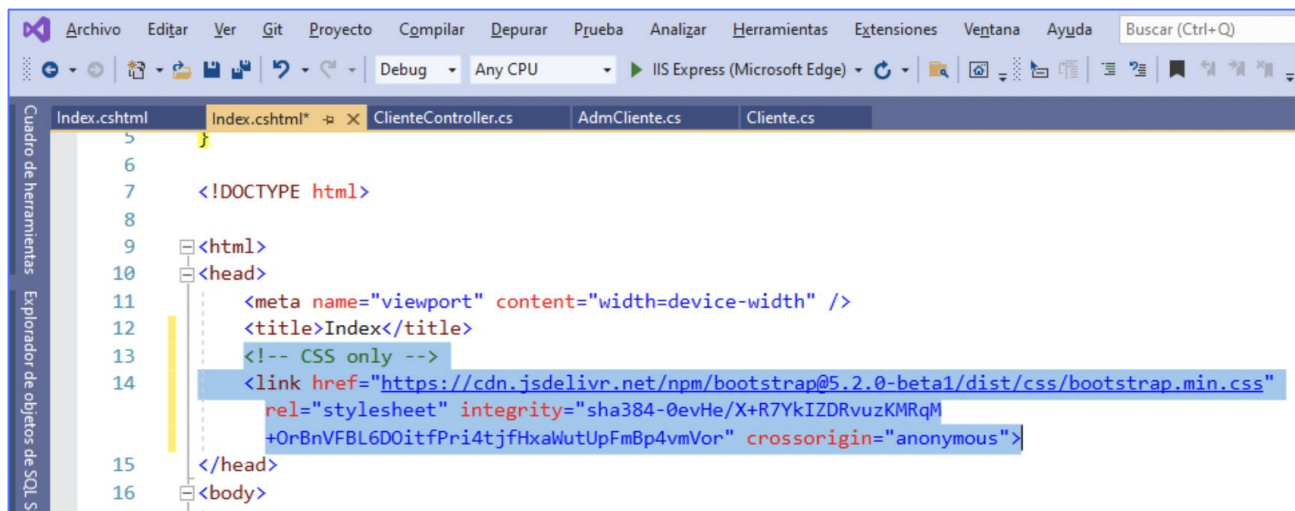
```
$ npm install bootstrap@5.2.0-beta1
```

```
$ gem install bootstrap -v 5.2.0-beta1
```

**Include via CDN**

When you only need to include Bootstrap's compiled CSS or JS, you can use [jsDelivr](#). See it in action with our simple [quick start](#), or [browse the examples](#) to jumpstart your next project. You can also choose to include Popper and our JS [separately](#).

```
<!-- CSS only -->
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0-beta1/dist/cs
```



```
5 }
6
7 <!DOCTYPE html>
8
9 <html>
10 <head>
11     <meta name="viewport" content="width=device-width" />
12     <title>Index</title>
13     <!-- CSS only -->
14     <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0-beta1/dist/css/bootstrap.min.css"
15           rel="stylesheet" integrity="sha384-0evHe/X+R7YkIZDRvuzKMRqM
16           +OrBnVFBL6D0itfPri4tjfhXaWutUpFmBp4vmVor" crossorigin="anonymous">
```

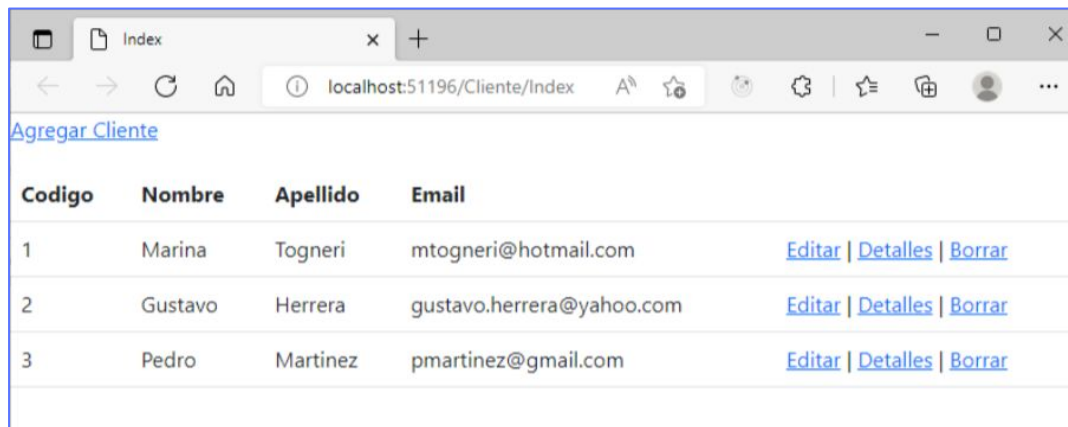
```
<body>
<p>
@Html.ActionLink("Agregar Cliente", "Create")
</p>
<tableclass="table">
<tr>
<th>
@Html.DisplayNameFor(model =>model.Codigo)
</th>
<th>
@Html.DisplayNameFor(model =>model.Nombre)
</th>
<th>
@Html.DisplayNameFor(model =>model.Apellido)
</th>
<th>
@Html.DisplayNameFor(model =>model.Email)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



```
@foreach (var item in Model) {  
    <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:



The screenshot shows a web browser window with the address bar displaying 'localhost:51196/Cliente/Index'. The page has a title 'Index' and a link 'Agregar Cliente' at the top left. Below the link is a table with four columns: 'Codigo', 'Nombre', 'Apellido', and 'Email'. The table contains three rows of data, each with a 'Codigo' and an 'Email' field, and a set of links ('Editar', 'Detalles', 'Borrar') at the end of each row.

Codigo	Nombre	Apellido	Email	
1	Marina	Togneri	mtogneri@hotmail.com	<a href="#">Editar</a>   <a href="#">Detalles</a>   <a href="#">Borrar</a>
2	Gustavo	Herrera	gustavo.herrera@yahoo.com	<a href="#">Editar</a>   <a href="#">Detalles</a>   <a href="#">Borrar</a>
3	Pedro	Martinez	pmartinez@gmail.com	<a href="#">Editar</a>   <a href="#">Detalles</a>   <a href="#">Borrar</a>

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.



**¡Sigamos  
trabajando!**