Tipo : Guía de Enunciado
Capítulo : Programación del lado del cliente
Duración : 180 minutos

I. OBJETIVO

Implementa paginación de Bootstrap en la vista de customer.

II. REQUISITOS

Los siguientes elementos de software son necesarios para la realización del laboratorio:

- Windows 10 (como mínimo Windows 8)
- Visual Studio 2017 (como mínimo Visual Studio 2015)

III. EJECUCIÓN DEL LABORATORIO

1. Abrir la solución del módulo anterior.

2. La mejor manera de hacer paginación es desde SQL Server, para no obtener todos los registros y administrarlos siempre en cada request. Por ende procedemos a modificar nuestro ICustomerRepository con el fin de obtener la lista ya paginada.
   (1) Agregar los siguientes lineas a la interface:
   "**Cibertec.Repositories\Northwind\ICustomerRepository**"

```
IEnumerable<Customer> PagedList(int startRow, int endRow);
int Count();
```

   Ver imagen:

```
using Cibertec.Models;
using System.Collections.Generic;

namespace Cibertec.Repositories.Northwind
{
    3 references | Cesar Velarde, 6 hours ago | 2 authors, 3 changes
    public interface ICustomerRepository : IRepository<Customer>
    {
        2 references | Cesar Velarde, 6 hours ago | 1 author, 1 change | 0 exceptions
        IEnumerable<Customer> PagedList(int startRow, int endRow);
        2 references | Cesar Velarde, 6 hours ago | 1 author, 1 change | 0 exceptions
        int Count();
    }
}
```

   (2) Proceder con la implementación, en el fichero "**Cibertec.Repositories.Dapper\Northwind\CustomerRepository**", con el siguiente código:

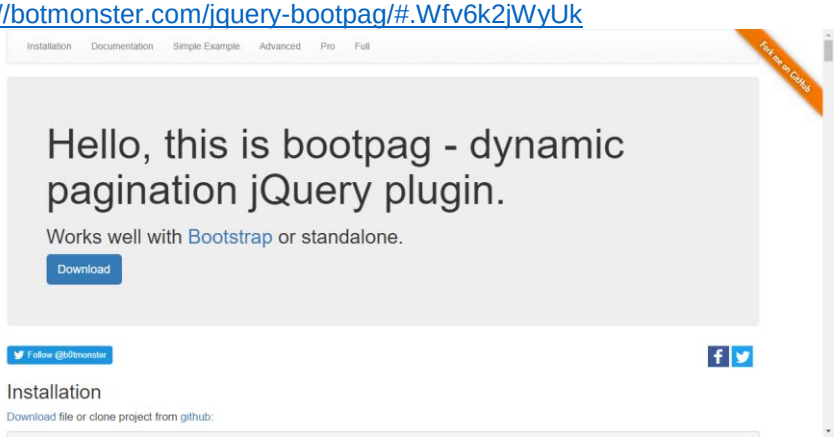
```
public int Count()
{
    using (var connection = new SqlConnection(_connectionString))
    {
        return connection.ExecuteScalar<int>("SELECT COUNT(Id) FROM
dbo.Customer");
    }
}

public IEnumerable<Customer> PagedList(int startRow, int endRow)
{
    if (startRow >= endRow) return new List<Customer>();
    using (var connection = new SqlConnection(_connectionString))
    {
```

```
            var parameters = new DynamicParameters();
            parameters.Add("@startRow", startRow);
            parameters.Add("@endRow", endRow);
            return connection.Query<Customer>("dbo.CustomerPagedList",
                parameters,
                commandType: System.Data.CommandType.StoredProcedure);
        }
    }
```

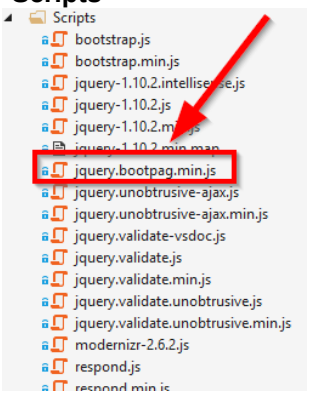
(3) Como parte final del repositorio ejecutamos el siguiente procedimiento almacenado en nuestra base de datos:

```
CREATE PROCEDURE [dbo].[CustomerPagedList]
    @startRow int,
    @endRow int
AS
BEGIN
    WITH CustomerResult AS
    (
           SELECT [Id]
             ,[FirstName]
             ,[LastName]
             ,[City]
             ,[Country]
             ,[Phone]
             ,ROW_NUMBER() OVER (ORDER BY Id) AS RowNum
      FROM .[dbo].[Customer]
    )

    SELECT [Id]
             ,[FirstName]
             ,[LastName]
             ,[City]
             ,[Country]
             ,[Phone]
    FROM CustomerResult
    WHERE RowNum BETWEEN @startRow AND @endRow;
END
```

3. Para la paginación con Bootstrap usaremos el plugin Jquery llamado "**bootpag**", mismo que lo descargaremos de esta url:
   http://botmonster.com/jquery-bootpag/#.Wfv6k2jWyUk



4. Una vez descargado lo agregamos a nuestro proyecto "**Cibertec.Mvc**" en la carpeta "**Scripts**"



5. Procedemos a modificar el archivo javascrip ubicado en "**App\customer.js**", con el siguiente código:

```
(function (customer) {
    customer.success = successReload;
    customer.pages = 1;
    customer.rowSize = 25;

    init();


    return customer;

    function successReload(option) {
        cibertec.closeModal(option);
    }

    function init() {
        $.get('/Customer/Count/' + customer.rowSize,
            function (data) {
                customer.pages = data;
                $('.pagination').bootpag({
                    total: customer.pages,
                    page: 1,
                    maxVisible: 5,
                    leaps: true,
                    firstLastUse: true,
                    first: '←',
                    last: '→',
                    wrapClass: 'pagination',
                    activeClass: 'active',
                    disabledClass: 'disabled',
                    nextClass: 'next',
                    prevClass: 'prev',
                    lastClass: 'last',
                    firstClass: 'first'
                }).on('page', function (event, num) {
                    getCustomers(num);
                    });
                getCustomers(1);
            });
    }

    function getCustomers(num) {
        var url = '/customer/List/' + num + '/' + customer.rowSize;
        $.get(url, function (data) {
            $('.content').html(data);
```

```
        });
    }

})(window.customer = window.customer || {});
```

6. En el archivo "**App_Start\RouteConfig**" agregamos la siguiente línea de código:

```
public class RouteConfig
{
    1 reference | Cesar Velarde, 6 hours ago | 2 authors, 2 changes | 0 exceptions
    public static void RegisterRoutes(RouteCollection routes)
    {
        routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
        routes.MapMvcAttributeRoutes();
        routes.MapRoute(
            name: "Default",
            url: "{controller}/{action}/{id}",
            defaults: new { controller = "Home", action = "Index", id = Ur
        );
    }
}
```

7. En el controlador **Customer** agregamos los siguientes métodos:

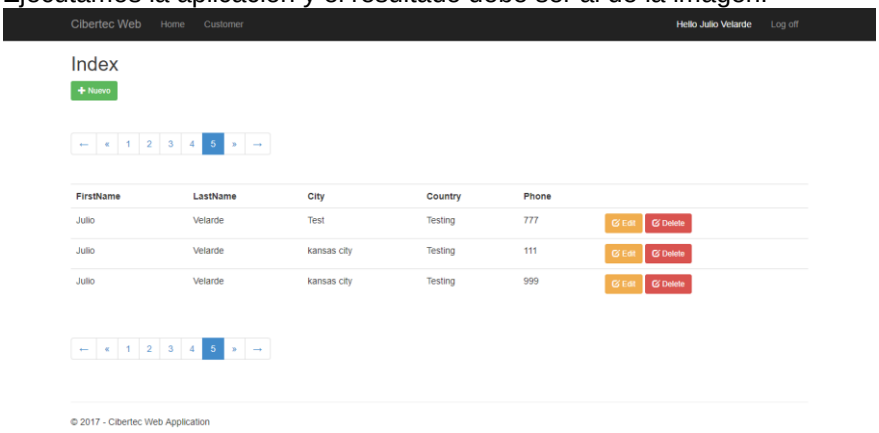
```
[Route("List/{page:int}/{rows:int}")]
public PartialViewResult List(int page, int rows)
{
       if (page <= 0 || rows <= 0) return PartialView(new
List<Customer>());
       var startRecord = ((page - 1) * rows) + 1;
       var endRecord = page * rows;
       return PartialView("_List",_unit.Customers.PagedList(startRecord,
endRecord));
}

[HttpGet]
[Route("Count/{rows:int}")]
public int Count(int rows)
{
       var totalRecords = _unit.Customers.Count();
       return totalRecords % rows != 0 ? (totalRecords / rows) + 1 :
totalRecords / rows;
}
```

8. En la vista "**Views\Customer\Index**" agregamos la llamada a nuestra nueva librería descargada y el nuevo markup a usar:

```
 9      @Messages.AlertMessages()
10
11      <p>
12          <a href="#" onclick="cibertec.getModal('@Url.Action("Create")'
13              <span class="glyphicon glyphicon-plus"></span> Nuevo
14          </a>
15      </p>
16
17      <div class="pagination"></div>
18      <div class="content"></div>
19      <div class="pagination"></div>
20
21      @Modal.GetModal("Customer")
22
23      <script src="~/Scripts/jquery.bootpag.min.js"></script>
24      <script src="~/App/cibertec.js" type="text/javascript"></script>
25      <script src="~/App/customer.js" type="text/javascript"></script>
```

(Considerar en texto resaltado)

9. Ejecutamos la aplicación y el resultado debe ser al de la imagen:



IV. EVALUACIÓN

1. ¿Cuáles son las carpetas principales en una aplicación MVC?

   Las carpetas principales son: Controllers, Views y Models

2. ¿Qué pasa si al nombre del controlador no se le coloca el prefijo Controller?

   Por convención todos los archivos que van a ser controladores deben tener el prefijo Controller al final del nombre para que puedan ser reconocidos por Asp.NET como tales.

3. ¿Cuál es el tipo de datos de los métodos del controller?

   El tipo base de todos los controladores es ActionResult.

4. ¿Qué es una vista Layout?

   Es la vista maestra, en ella se establece la estructura base (html) de todas las vistas que tendrá la aplicación web. Algunas vistas pueden no utilizar la vista base.

5. Los modelos, ¿pueden ser librería de clases?

   Sí, los modelos son un concepto que involucra entidades, acceso a datos, reglas de negocio y pueden ser implementadas en librería de clases (dll).

6. ¿Qué son las vistas parciales?

   Al igual que las vistas normales, estas permiten ser reutilizadas en ciertas partes de la aplicación.