UNIVERSIDAD DE COSTA RICA

ECCI Escuela de Ciencias de la Computación e Informática

Ingeniería de Software – CI0126

Laboratorio #8 JavaScript y .NET

Profesor: Dr. Gustavo López

Asistente: Edwin Brenes

1. Índice

2. Introducción

El objetivo de este laboratorio es aprender a utilizar scripts de JavaScript dentro de *Views* generadas con *Razor*, es decir, incorporar elementos de *js* dentro de ficheros *.cshtml*. Además, usted va aprender a utilizar algunas nuevas funcionalidades presentes .NET y va a explorar cómo crear tablas con carga de contenido dinámico de forma asíncrona, cuyos datos provienen de su propia aplicación construida en *.NET*.

3. Entregables

1. Siga paso a paso las secciones de Proyecto en .NET Framework y complete lo que se le solicita. Al finalizar las secciones, debe entregar una versión funcional del laboratorio. Adjunte un enlace a un repositorio en GitHub.

2. Complete los ejercicios propuestos en la sección Ejercicios, el código fuente debe de estar en el mismo repositorio del punto anterior.

Nota: se tomará en consideración, como parte de la evaluación, las buenas prácticas de programación (clean code).

4. Proyecto en .NET Framework

Utilizando Visual Studio, cree un nuevo proyecto de *.NET Framework* y utilice la plantilla *MVC*, misma que ha utilizado en laboratorios anteriores. O bien, puede reutilizar la base de los proyectos construidos en alguno de los laboratorios elaborados anteriormente.

Al crear el proyecto, usted deberá tener un directorio de archivos similar al que se muestra en la Imagen 1. A este directorio de archivos, agregue una nueva carpeta (el nombre debe ser ***js***) de modo que su nuevo directorio de archivos se vea similar al mostrado en la Imagen 2.

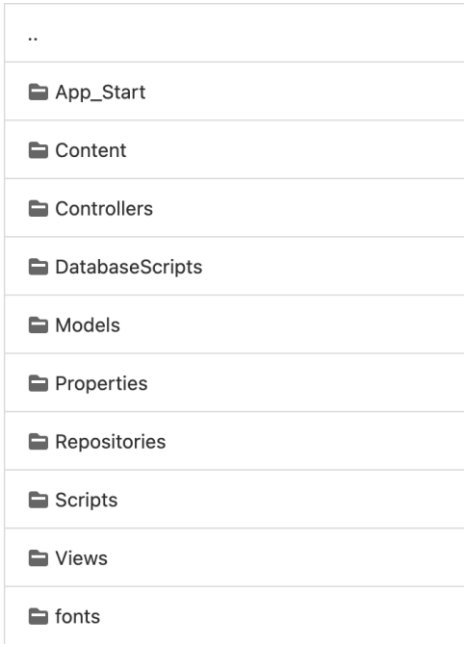

Imagen 1

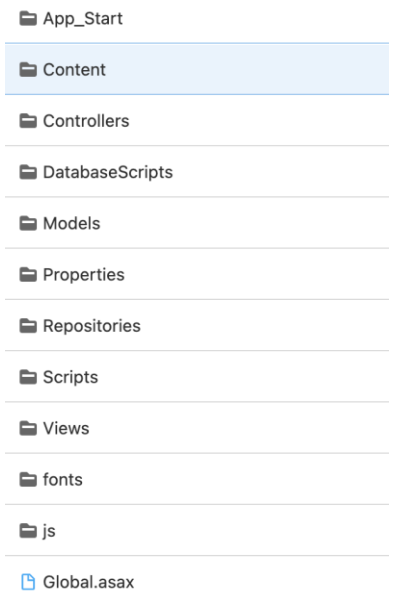

Imagen 2

Nota: no es necesario que usted tenga los directorios llamados *DatabaseScripts* ni *Repositories.*

Ahora, de manera similar a lo realizado en los laboratorios previos, agregue un *connection string* (de alguna base de datos) a su proyecto. Puede utilizar la base de datos brindada en el curso de Bases de Datos o bien utilizar alguna de las bases de datos que usted ya creó en laboratorios anteriores.

a. Scripts de Bases de datos

En la base de datos correspondiente, realice la creación de una tabla llamada *Products* e inserte algunos datos. Puede utilizar el siguiente script:

```
CREATE TABLE Products (
   id INTEGER IDENTITY NOT NULL,
   quantity INTEGER NOT NULL,
   name VARCHAR(MAX) NOT NULL,
   price DECIMAL NOT NULL DEFAULT 0.0

   Constraint PK_Products Primary Key (id)
)
INSERT INTO Products (name,price, quantity) values  ('Nintendo', 250.0,200)
INSERT INTO Products (name,price, quantity) values  ('Laptop', 600.50,100)
```

b. Modelo, Vista, Controlador

Una vez creada la tabla, usted debe crear un nuevo modelo que le permita almacenar información de productos. Cree un nuevo modelo llamado Products y agregue dicho modelo en la carpeta Models. Además, cree un *ProductsHandler* (similar a los estudiados en laboratorios previos) y en su código fuente agregue la siguiente estructura:

```
// Para importar su modelo Products
using Laboratorio8.Models;

public class ProductsHandler
{
   public IEnumerable<Products> GetAll()
   {
       /*
       * IEnumerable es una estructura similar a List, la cual se utilizó en
laboratorios previos
       * Llene esta lista con el resultado de su consulta.
       */
       IEnumerable<Products> productsList;

       /*
       * Agregue la lógica y elementos necesarios para que este método retorne
todos los productos
```

```
        * Es decir, debe realizar un SELECT * FROM Products
        */

        return productsList;
    }

}
```

Agregue un nuevo *Controller* a los controladores ya existentes. En el controlador creado, agregue el siguiente código:

```
using Laboratorio8.Models;
using Laboratorio8.Handlers;
using System.Web.Mvc;

namespace shop_demo_app.Controllers
{
    public class ProductsController : Controller
    {
        private ProductsHandler ProductsHandler;

        public ProductsController()
        {
            ProductsHandler = new ProductsHandler();
        }

        public ActionResult Index()
        {
            return View();
        }

        public JsonResult GetAllProducts()
        {
            var products = ProductsHandler.GetAll();
            return Json(products, JsonRequestBehavior.AllowGet);
        }
    }
}
```

Finalmente, agregue una vista *Index.cshtml* asociada al método Index del controlador de productos.

Breve explicación del método GetAllProdcuts

Este método invoca al handler que usted creó previamente, con lo cual la variable *products* debe contener una lista de productos. *JsonResult,* lo que permite es retornar a través de de este *endpoint* (o ruta de controlador) datos en formato JSON, de hecho en el return se realiza lo siguiente:

1. Serializa productos en un JSON
2. Indica que los datos retornados serán accesibles a través de un *request GET* de *HTTP*. Es decir, se permite que estos datos sean consumidos mediante un *request* de tipo *GET.*

Inicie la aplicación y en su navegador consulte la ruta */Products/GetAllProducts*, deberá observar que el resultado obtenido es un un JSON que básicamente es la lista de productos existentes en la base de datos.

c. Integrar código js en una página de Razor

En la vista *Index.cshtml* asociada al método Index del controlador de productos, agregue el siguiente código:

```
@{
    string dataOrigin = Url.Content("~/Products/GetAllProducts");
}
<body>
     <h1> WELCOME </h1>
    <input type="button" value="Show Products" onclick="getProducts()" />
    <div id="ProductsContext"></div>
</body>
<script src="~/js/DataTable.js" type="text/javascript"></script>
<script type="text/javascript">

    let productsContext = document.getElementById("ProductsContext");
    const dataTable = new DataTable("@dataOrigin", productsContext);

    function getProducts() {
        dataTable.fillContext();
    }
</script>
```

Explicación del código

La variable dataOrigin contiene la dirección del dominio en el cual corre la aplicación y se concatena la ruta que se creó para obtener los productos. Es decir, mientras la aplicación corre en localhost, data origin contiene un string que luce así: https://localhost:port/Products/GetAllProducts. No obstante, si posteriormente se hace un despliegue de la aplicación en un servidor y a esta se le asigna un dominio (por ejemplo myapp.com) dicha ruta se ajusta sin tener que realizar cambios en el código, de modo que la ruta generada luce, por ejemplo, así: https://myapp.com/Products/GetAllProducts

La idea de esta vista, es la siguiente:

5. Tener una vista vacía, con un único botón *Show products*
6. Al presionar el botón, se debe cargar los datos de los productos y esto se hará de modo asíncrono (sin tener que refrescar la vista).

7. El código de data table se agregara posteriormente, pero será un objeto que es capaz de hacer lo siguiente:
    a. Dado un contexto y una fuente de datos, una instancia de data table será capaz de convertir los los datos recuperados de la fuente datos a un contenido "presentable" en html
    b. El contenido se construye dinámicamente con los datos recuperados y lo coloca en el contexto proporcionado

Cargar datos en pantalla

En el directorio llamado ***js*** que usted creó al inicio del laboratorio, agregue un archivo llamado dataTable.js y en el contenido del código fuente, agregue lo siguiente:

```
class DataTable {
   constructor(dataOrigin, context) {
       this.dataOrigin = dataOrigin;
       this.context = context;
   }

   fillContext() {
       fetch(this.dataOrigin)
       .then(response => {
           return response.json();
        })
       .then(data => {
           let htmlContent = '';
           let row = '';
           for (const object of data) {
               for (const key in object) {
                   row += `${object[key]} `
               }
               htmlContent += `<p> ${row} </p> <br/>`;
               row = '';
           }
           this.context.innerHTML = htmlContent;
       })
       .catch(error => {
           console.log(error);
```

```
            })
    }
}
```

8. Ejercicios

a. Construir tabla dinámica

Instrucciones

1. Modifique el código de dataTable.js para que los productos obtenidos con fetch, se presenten en una Tabla HTML. En esencia lo que debe cambiar es lo que se realiza sobre la variable ***htmlContent***.
2. Utilice las funciones createElement, appendChild, entre otras que le pueden servir.

b. Tabla dinámica con js plugins

Instrucciones

1. Modifique el código de dataTable.js para que los productos obtenidos con fetch, se presenten en una tabla, pero esta vez utilice un plugin de js existente.
2. Seleccione alguna de las opciones y consulte la documentación para poder lograr el objetivo:
    a. Tabulator
        i. Ejemplos
        ii. Instalación
    b. Clusterize.js
    c. DataTables | Table plug-in for jQuery
    d. Otras opciones
    e. 20 Useful Javascript Data Table Libraries – Bashooka