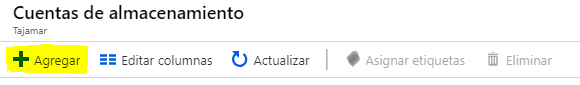
En este tutorial vamos a aprender a crear una cuenta de almacenamiento con Portal de Azure (también podremos usarlo en entorno local con la máquina virtual de Azure) en el cual vamos a crear una tabla y vamos a manejar sus datos.

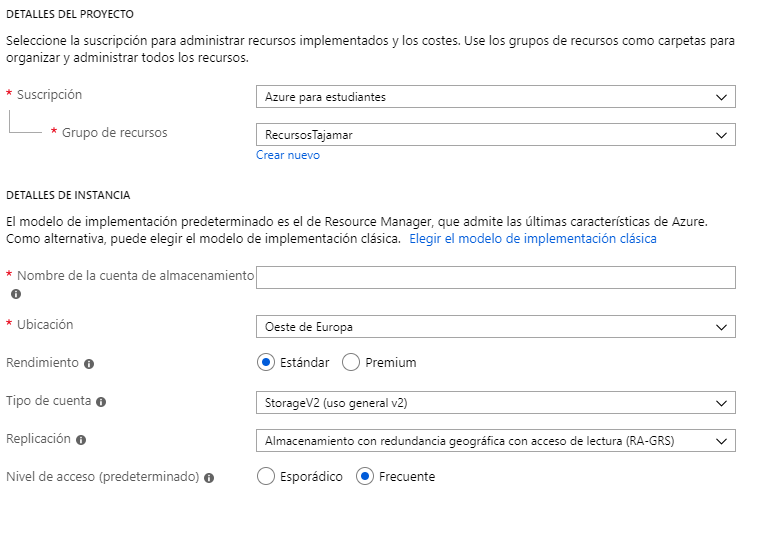
**Con Cuenta Azure:**

Primero Accedemos a “Cuentas de Almacenamiento”.



Una vez dentro añadimos una nueva cuenta.

Luego rellenamos los datos necesarios y pulsamos en crear.



Una vez creada la cuenta de almacenamiento, en el apartado “Claves de acceso” podremos visualizar la clave de acceso que necesitaremos para más adelante.

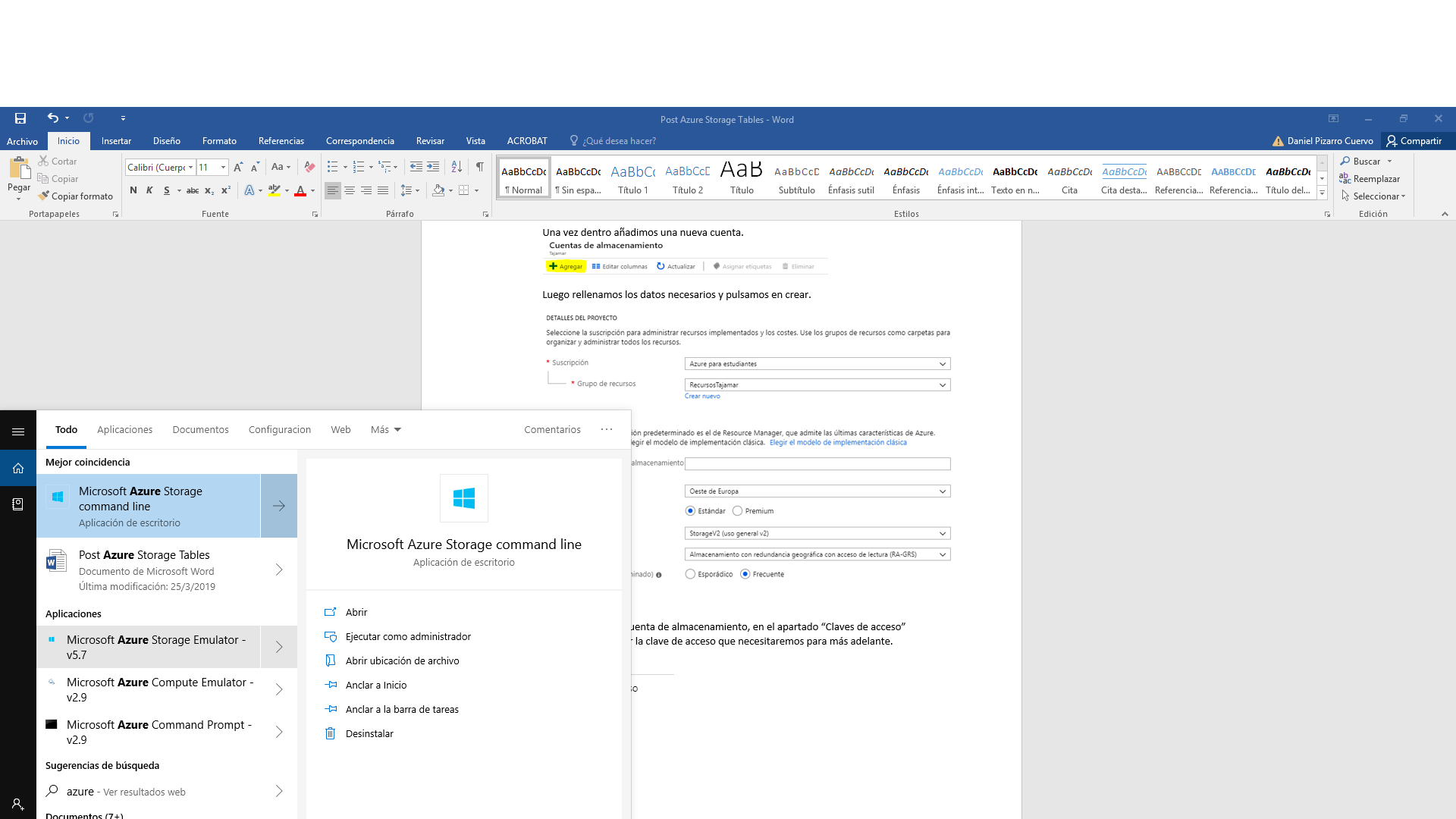


<add key="StorageConnectionString" value="DefaultEndpointsProtocol=https;AccountName=account-name;AccountKey=account-key" />

Sustituiremos los parámetros por nuestra clave correspondiente

**Sin cuenta Azure:**

Abriremos el emulador de Azure.



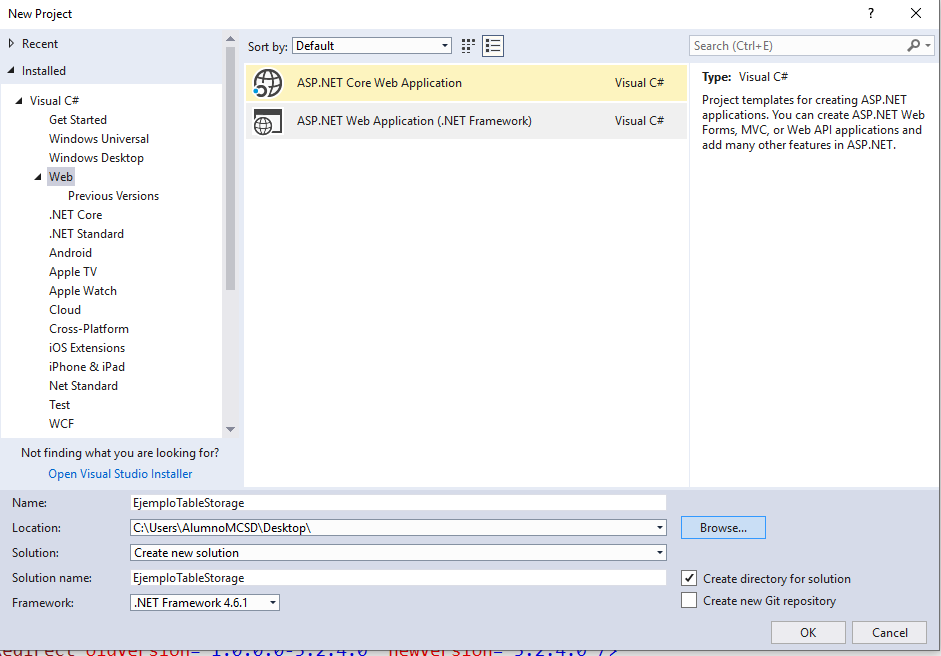
Luego deberemos añadir a nuestro proyecto el siguiente código:

<add key="StorageConnectionString" value="UseDevelopmentStorage=true" />

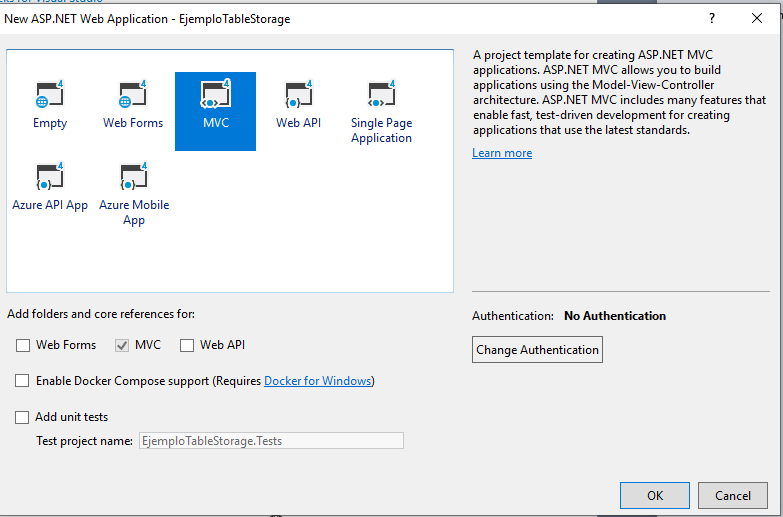
Ya hemos creado todo lo necesario para empezar a crear la aplicación.

Abrimos Visual Studios y creamos un nuevo proyecto **ASP.NET MVC**, al cual llamaremos

“**EjemploTableStorage”.**

****

Y seleccionaremos MVC para que nos cree un proyecto predeterminado.

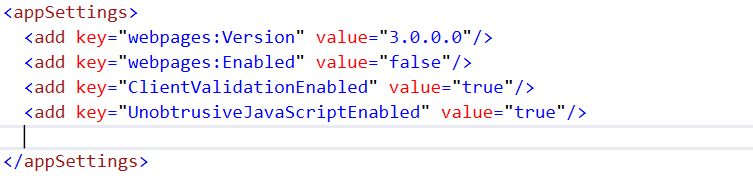


Lo primero que vamos a cambiar es el archivo “**Web.config**” para añadir en **<appSettings>**

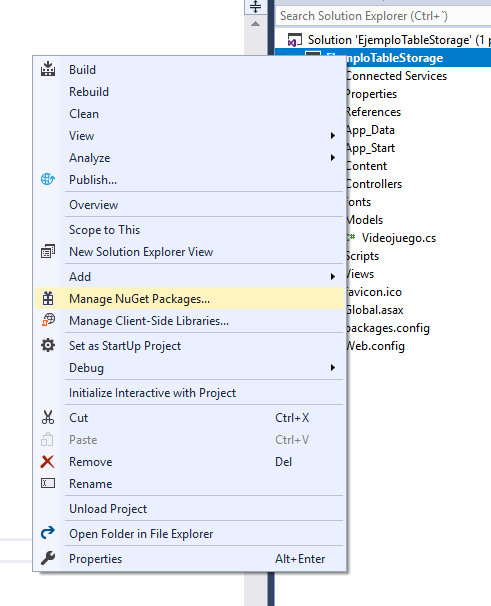
Nuestra cadena de conexión que nos proporciona Azure (O en su defecto la conexión predeterminada del emulador).



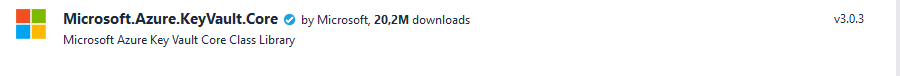
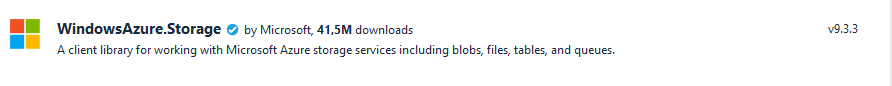
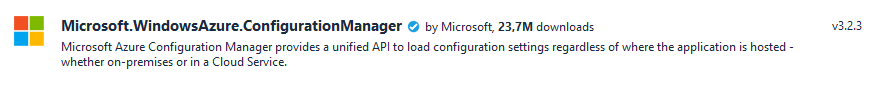
Y añadimos el código que hemos obtenido antes (Recuerda que es distinto si usas Portal Azure o el emulador de Azure).

****

También deberemos incluir los paquetes necesarios para el correcto funcionamiento de la aplicación.



E instalamos los tres paquetes necesarios:



Ahora en la carpeta “**Models**” añadiremos una clase llamada “**Videojuego.cs**” (Click derecho sobre la carpeta y pulsamos en “**añadir**” y pulsamos en “**clase**”).

Una vez creada lo primero que haremos es indicar que la clase herede de “**TableEntity**”.

Añadimos el using:

using Microsoft.WindowsAzure.Storage.Table;

y la herencia:

public class Videojuego : TableEntity

Una vez hecho esto, podremos crear nuestra clase con los atributos que necesitemos, nos tendrá que quedar algo parecido al siguiente código:

public Videojuego() { }

private String \_Estudio { get; set; }

public String Estudio

{

get

{

return this.\_Estudio;

}

set

{

this.PartitionKey = value;

this.\_Estudio = value;

}

}

private String \_IdVideojuego { get; set; }

public String IdVideojuego

{

get

{

return this.\_IdVideojuego;

}

set

{

this.RowKey = value;

this.\_IdVideojuego = value;

}

}

public String Nombre { get; set; }

public String Creador { get; set; }

public String Descripcion { get; set; }

public DateTime FechaLanzamiento { get; set; }

public int Precio { get; set; }

public String Categorias { get; set; }

Ahora tendremos que crear a nivel de proyecto una carpeta llamada “**Repositories**” y añadiremos una clase llamada “**RepositoryVideojuegos.cs**”.

Añadiremos los usings necesarios:

using Microsoft.Azure;

using Microsoft.WindowsAzure.Storage;

using Microsoft.WindowsAzure.Storage.Table;

y crearemos el controlador en el cual indicaremos que cadena de conexión deseamos usar (La misma que hemos añadido en el **Web.config**).

CloudTable tabla;

public RepositoryVideojuegos()

{

String keys = CloudConfigurationManager.GetSetting("StorageConnectionString");//Nombre de tu conexión.

CloudStorageAccount account = CloudStorageAccount.Parse(keys);

CloudTableClient client = account.CreateCloudTableClient();

this.tabla = client.GetTableReference("videojuegos");

this.tabla.CreateIfNotExists();

}

public void CrearVideojuego(String estudio, String idvideojuego

, String nombre, String creador,String descripcion,DateTime fecha,int precio,String categorias)

{

Videojuego videojuego = new Videojuego();

videojuego.Estudio = estudio;

videojuego.IdVideojuego = idvideojuego;

videojuego.Nombre = nombre;

videojuego.Creador = creador;

videojuego.Descripcion = descripcion;

videojuego.FechaLanzamiento = fecha;

videojuego.Precio = precio;

videojuego.Categorias = categorias;

TableOperation operation = TableOperation.Insert(videojuego);

this.tabla.Execute(operation);

}

public Videojuego BuscarVideojuego(String partitionkey

, String rowkey)

{

TableOperation operation = TableOperation.Retrieve<Videojuego>(partitionkey, rowkey);

TableResult result = this.tabla.Execute(operation);

if (result.Result == null)

{

return null;

}

else

{

Videojuego videojuego = result.Result as Videojuego;

return videojuego;

}

}

public void ModificarVideojuego(String estudio, String idvideojuego

, String nombre, String creador, String descripcion, DateTime fecha, int precio, String categorias)

{

Videojuego videojuego = this.BuscarVideojuego(estudio, idvideojuego);

if (videojuego != null)

{

videojuego.Nombre = nombre;

videojuego.Creador = creador;

videojuego.Descripcion = descripcion;

videojuego.FechaLanzamiento = fecha;

videojuego.Precio = precio;

videojuego.Categorias = categorias;

TableOperation operation = TableOperation.Replace(videojuego);

this.tabla.Execute(operation);

}

}

public void EliminarVideojuego(String partitionkey, String rowkey)

{

Videojuego videojuego = this.BuscarVideojuego(partitionkey, rowkey);

if (videojuego != null)

{

TableOperation operation = TableOperation.Delete(videojuego);

this.tabla.Execute(operation);

}

}

public List<Videojuego> MostrarVideojuegos()

{

TableQuery<Videojuego> query = new TableQuery<Videojuego>();

List<Videojuego> videojuegos = this.tabla.ExecuteQuery(query).ToList();

return videojuegos;

}

Con esto ya tendríamos las acciones que necesitamos para poder probar la aplicación.

Necesitaremos un controlador para que la aplicación pueda acceder al repositorio y se pueda crear y mostrar los videojuegos. Dentro de la carpeta “**Controllers**” añadiremos un controlador **MVC Empty (Vacío)** al cual hemos llamado “**VideojuegosController**”.

Tendremos que crear un método “**Index**” encargado de mostrar los datos y otros métodos llamados “**Create,Delete,Edit,Details**” que deberá dividirse en dos uno por **Get** y otro por **Post,** a excepción de **Delete** y **Details:**

public class VideojuegosController : Controller

{

RepositoryVideojuegos repo;

public VideojuegosController()

{

this.repo = new RepositoryVideojuegos();

}

public ActionResult Index()

{

List<Videojuego> videojuegos = repo.MostrarVideojuegos();

return View(videojuegos);

}

public ActionResult Create()

{

return View();

}

[HttpPost]

public ActionResult Create(Videojuego videojuego)

{

repo.CrearVideojuego(videojuego.Estudio, videojuego.IdVideojuego

,videojuego.Nombre, videojuego.Creador, videojuego.Descripcion,

videojuego.FechaLanzamiento, videojuego.Precio, videojuego.Categorias);

return RedirectToAction("Index");

}

public ActionResult Details(String partition, String row)

{

Videojuego videojuego = repo.BuscarVideojuego(partition, row);

return View(videojuego);

}

public ActionResult Edit(String partition, String row)

{

Videojuego videojuego = repo.BuscarVideojuego(partition, row);

return View(videojuego);

}

[HttpPost]

public ActionResult Edit(Videojuego videojuego)

{

repo.ModificarVideojuego(videojuego.Estudio, videojuego.IdVideojuego

, videojuego.Nombre, videojuego.Creador, videojuego.Descripcion,

videojuego.FechaLanzamiento, videojuego.Precio, videojuego.Categorias);

return RedirectToAction("Index");

}

public ActionResult Delete(String partition, String row)

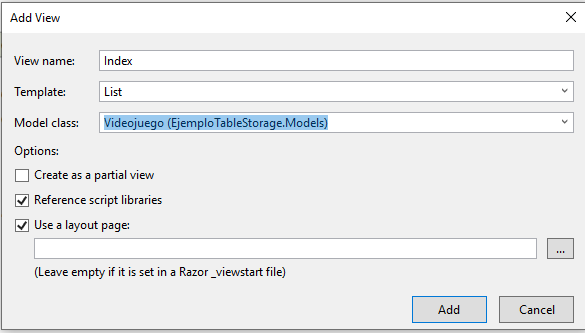
{

repo.EliminarVideojuego(partition, row);

return RedirectToAction("Index");

}

Ahora solo nos quedan las vistas, las cuales las crearemos con “**scaffolding**” (click derecho sobre el nombre del método y pulsando en “**crear vista**” y marcaremos lo siguiente).



**--NOTA IMPORTANTE--**

El campo “**Template**” deberá ser :

**Create** para crear, **Details** para mostrar detalles, **Delete** para eliminar, **Edit** para modificar y **List** para mostrar.

Eliminaremos los datos que no queremos mostrar para que se vea más atractiva la tabla.

Nos tienen que quedar las vistas así:

**Index:**

@model IEnumerable<EjemploTableStorage.Models.Videojuego>

@{

ViewBag.Title = "Index";

}

<h2>Index</h2>

<p>

@Html.ActionLink("Añadir Videojuego", "Create")

</p>

@if (Model != null || Model.Count() > 0)

{

<table class="table">

<tr>

<th>

@Html.DisplayNameFor(model => model.Estudio)

</th>

<th>

@Html.DisplayNameFor(model => model.IdVideojuego)

</th>

<th>

@Html.DisplayNameFor(model => model.Nombre)

</th>

<th></th>

</tr>

@foreach (var item in Model)

{

<tr>

<td>

@Html.DisplayFor(modelItem => item.Estudio)

</td>

<td>

@Html.DisplayFor(modelItem => item.IdVideojuego)

</td>

<td>

@Html.DisplayFor(modelItem => item.Nombre)

</td>

<td>

@Html.ActionLink("Editar", "Edit", new { partition = item.PartitionKey, row = item.RowKey }) |

@Html.ActionLink("Detalles", "Details", new { partition = item.PartitionKey, row = item.RowKey }) |

@Html.ActionLink("Eliminar", "Delete", new { partition = item.PartitionKey, row = item.RowKey })

</td>

</tr>

}

</table>

}

else{

<h4>No hay Videojuegos...</h4>

}

**Create:**

@model EjemploTableStorage.Models.Videojuego

@{

ViewBag.Title = "Create";

}

<h2>Create</h2>

@using (Html.BeginForm())

{

@Html.AntiForgeryToken()

<div class="form-horizontal">

<h4>Videojuego</h4>

<hr />

@Html.ValidationSummary(true, "", new { @class = "text-danger" })

<div class="form-group">

@Html.LabelFor(model => model.Estudio, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.Estudio, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.Estudio, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.IdVideojuego, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.IdVideojuego, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.IdVideojuego, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.Nombre, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.Nombre, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.Nombre, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.Creador, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.Creador, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.Creador, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.Descripcion, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.Descripcion, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.Descripcion, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.FechaLanzamiento, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.FechaLanzamiento, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.FechaLanzamiento, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.Precio, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.Precio, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.Precio, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.Categorias, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.Categorias, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.Categorias, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

<div class="col-md-offset-2 col-md-10">

<input type="submit" value="Create" class="btn btn-default" />

</div>

</div>

</div>

}

<div>

@Html.ActionLink("Volver", "Index")

</div>

@section Scripts {

@Scripts.Render("~/bundles/jqueryval")

}

**Details:**

@model EjemploTableStorage.Models.Videojuego

@{

ViewBag.Title = "Details";

}

<h2>Details</h2>

<div>

<h4>Videojuego</h4>

<hr />

<dl class="dl-horizontal">

<dt>

@Html.DisplayNameFor(model => model.Estudio)

</dt>

<dd>

@Html.DisplayFor(model => model.Estudio)

</dd>

<dt>

@Html.DisplayNameFor(model => model.IdVideojuego)

</dt>

<dd>

@Html.DisplayFor(model => model.IdVideojuego)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Nombre)

</dt>

<dd>

@Html.DisplayFor(model => model.Nombre)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Creador)

</dt>

<dd>

@Html.DisplayFor(model => model.Creador)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Descripcion)

</dt>

<dd>

@Html.DisplayFor(model => model.Descripcion)

</dd>

<dt>

@Html.DisplayNameFor(model => model.FechaLanzamiento)

</dt>

<dd>

@Model.FechaLanzamiento.ToString("dd/MM/yyy")

</dd>

<dt>

@Html.DisplayNameFor(model => model.Precio)

</dt>

<dd>

@Html.DisplayFor(model => model.Precio)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Categorias)

</dt>

<dd>

@Html.DisplayFor(model => model.Categorias)

</dd>

</dl>

</div>

<p>

@Html.ActionLink("Editar", "Edit", new { partition = Model.PartitionKey,row = Model.RowKey }) |

@Html.ActionLink("Volver", "Index")

</p>

**Edit:**

@model EjemploTableStorage.Models.Videojuego

@{

ViewBag.Title = "Edit";

}

<h2>Edit</h2>

@using (Html.BeginForm())

{

@Html.AntiForgeryToken()

<div class="form-horizontal">

<h4>Videojuego</h4>

<hr />

@Html.ValidationSummary(true, "", new { @class = "text-danger" })

<div class="form-group">

@Html.LabelFor(model => model.Estudio, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.Estudio, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.Estudio, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.IdVideojuego, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.IdVideojuego, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.IdVideojuego, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.Nombre, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.Nombre, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.Nombre, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.Creador, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.Creador, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.Creador, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.Descripcion, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.Descripcion, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.Descripcion, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.FechaLanzamiento, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.FechaLanzamiento, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.FechaLanzamiento, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.Precio, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.Precio, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.Precio, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.Categorias, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.Categorias, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.Categorias, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

<div class="col-md-offset-2 col-md-10">

<input type="submit" value="Save" class="btn btn-default" />

</div>

</div>

</div>

}

<div>

@Html.ActionLink("Volver", "Index")

</div>

Ya tendríamos todo preparado para probar nuestra aplicación.

**Puntos Importantes:**

Creo que los puntos más importantes para el desarrollo de la aplicación son

* Entender para que puede utilizarse este tipo de almacenamiento

(**NoSQL**).

* Si no se está muy familiarizado con Portal de Azure, pueden surgir problemas a la hora de la creación de la cuenta de almacenamiento.
* La sintaxis de creación de la tabla.

Espero que os haya servido de ayuda y no olvidéis dejad un comentario para cualquier pregunta.