Tutorial for Class number 11

This exercise intends develop a web application to use **Ajax** (*Asynchronous Javascript and XML)* using **jQuery Unobtrusive AJAX**

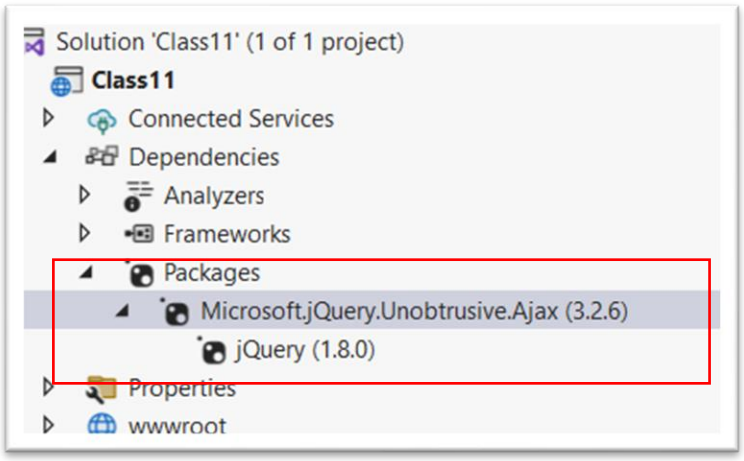
This technology is used to make partial updates in the web page, make the sites more fluid and responsive.

First step – Configure and test Ajax

- Create a new "ASP.NET Core Web App (Model-View-Controller)" project.
- Install the package

```
PM> Install-Package Microsoft.jQuery.Unobtrusive.Ajax
```

Confirm the installation in the Solution Explorer



This package allows to use **forms** and **links** to generate asynchronous requests (AJAX).

Using Ajax in a Form

Alter the view **Index** of the associated with the **Home** controller:

```
Index.cshtml
@{
    ViewData["Title"] = "Home Page";
}
Date: @DateTime.Now

<form asp-action="TestAjaxForm"
    data-ajax="true" data-ajax-method="post"
    data-ajax-mode="replace" data-ajax-update="#box">

    <label>Text to Send to server:</label>
    <input type="text" name="Text" />
    <input type="submit" value="Send" />
</form>

<div id="box">Original text</div>
<br />
Date: @DateTime.Now
```

This form will send the data to the action "TestAjaxForm" using Ajax. The **data-ajax** attributes are used to configure the behaviour.

The next list, explain all the options:

Custom Attributes

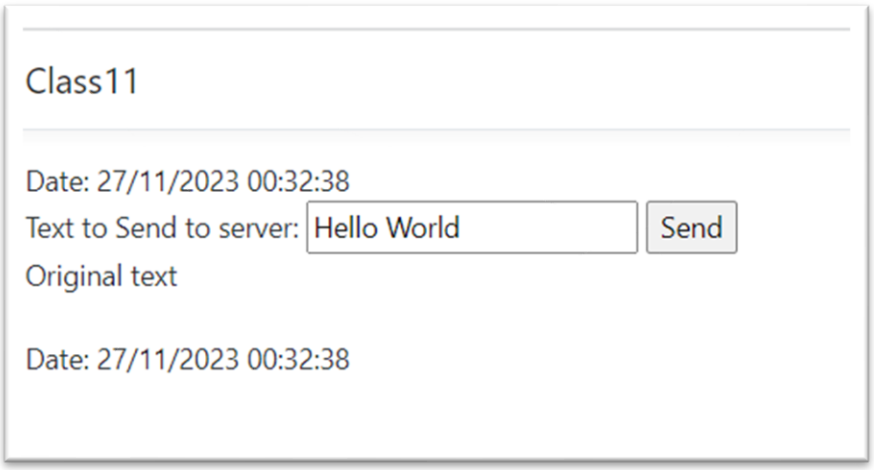
The following table details the custom attributes that control the behaviour of jQuery Unobtrusive AJAX:

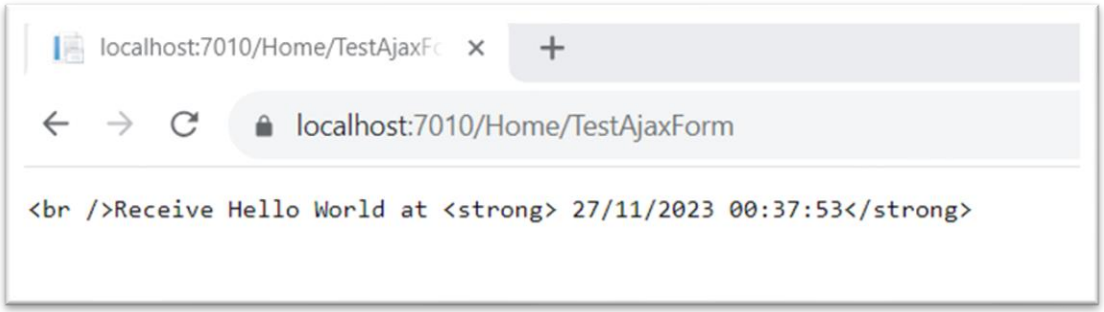
Attribute	Description
`data-ajax`	Must be set to `true` to activate unobtrusive Ajax on the target element.
`data-ajax-confirm`	Gets or sets the message to display in a confirmation window before a request is submitted.
`data-ajax-method`	Gets or sets the HTTP request method ("Get" or "Post").
`data-ajax-mode`	Gets or sets the mode that specifies how to insert the response into the target DOM element. Valid values are `before`, `after` and `replace`. Default is `replace`
`data-ajax-loading-duration`	Gets or sets a value, in milliseconds, that controls the duration of the animation when showing or hiding the loading element.
`data-ajax-loading`	Gets or sets the id attribute of an HTML element that is displayed while the Ajax function is loading.
`data-ajax-begin`	Gets or sets the name of the JavaScript function to call immediately before the page is updated.
`data-ajax-complete`	Gets or sets the JavaScript function to call when response data has been instantiated but before the page is updated.
`data-ajax-failure`	Gets or sets the JavaScript function to call if the page update fails.
`data-ajax-success`	Gets or sets the JavaScript function to call after the page is successfully updated.
`data-ajax-update`	Gets or sets the ID of the DOM element to update by using the response from the server.
`data-ajax-url`	Gets or sets the URL to make the request to.

Create a **TestAjaxForm** action, in the **Home** controller, that will receive the data in the form, and process it, in this case return a string to the client browser.

```
public string TestAjaxForm(string Text)
{
    return "<br>Receive " + Text + " at <strong> " + DateTime.Now + "</strong>";
}
```

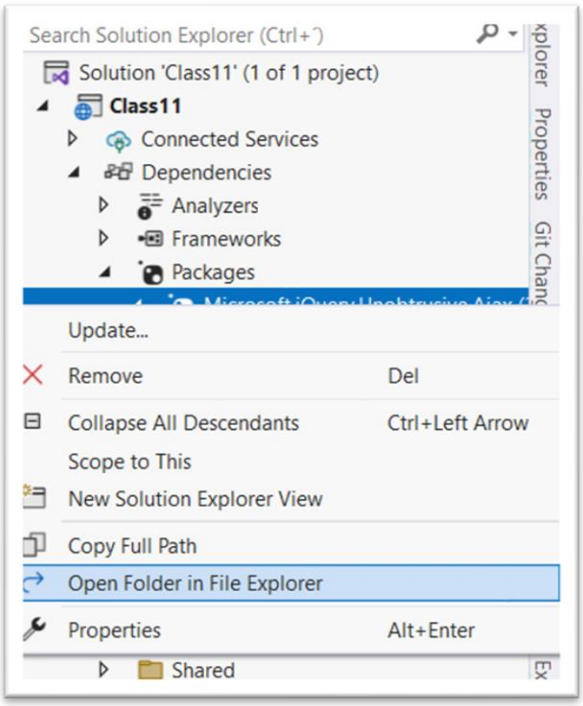
Now you can test the application.



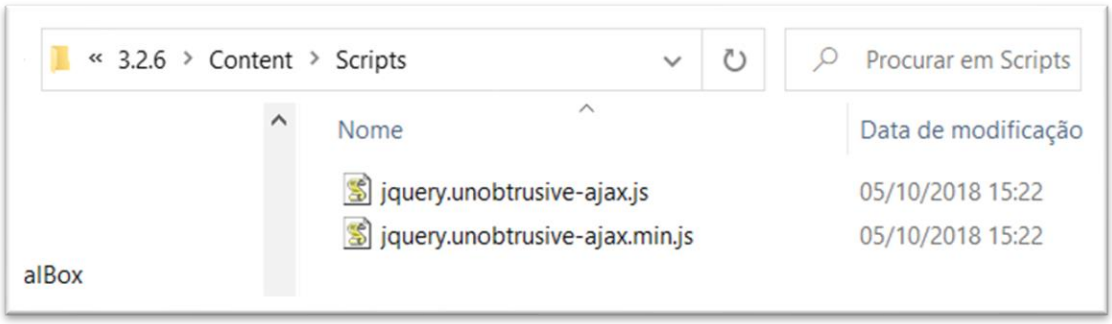


Oops. Did not work!!!!
The correct result was supposed to do the replacement of text "Original text".

To fix this we need include in the site a reference to the JavaScript file that process the Ajax.
Right Click in the Package previously installed and select “Open Folder in File Explorer”.



Open to the scripts folder.

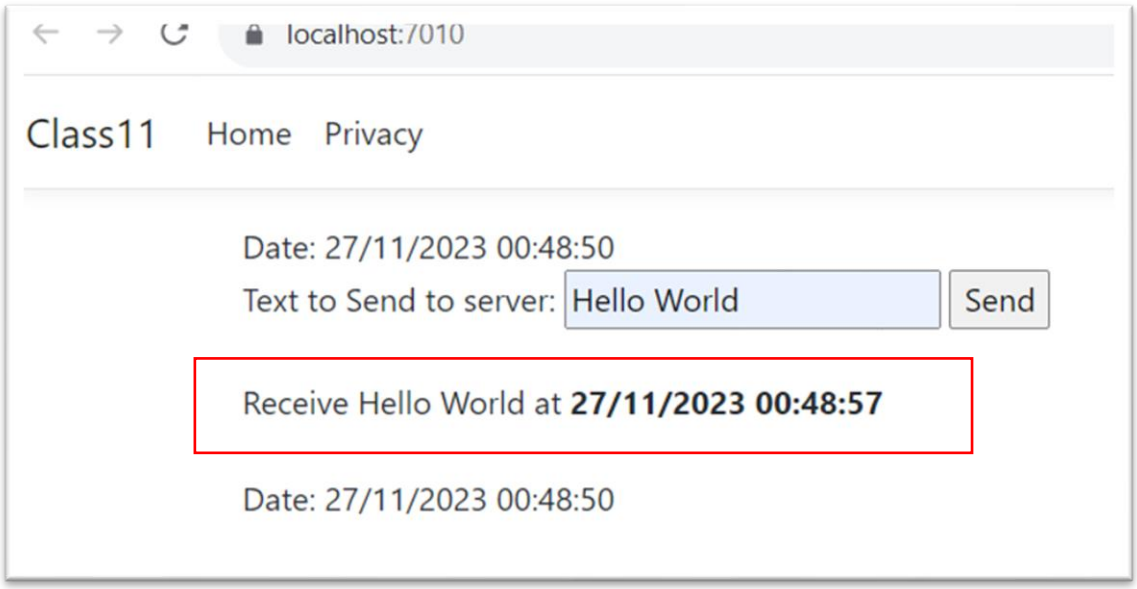


In the Soluction Explorer create a folder “jquery-ajax” in the lib folder.
Put the previous files in that folder

Now we need call that script file in each view that use Ajax, or in the **_Layout.cshtml** file, that is used in all views.

```
</footer>
<script src="~/lib/jquery/dist/jquery.min.js"></script>
<script src="~/lib/jquery-ajax/jquery.unobtrusive-ajax.js"></script>
<script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
 <script src="~/js/site.js" asp-append-version="true"></script>
@await RenderSectionAsync("Scripts", required: false)
```

Test again.



Now it works!! Only the content of the element with id=box is updated, the rest of the page are unaltered.

You can test the application using the other **data-ajax-mode** options (after, before).

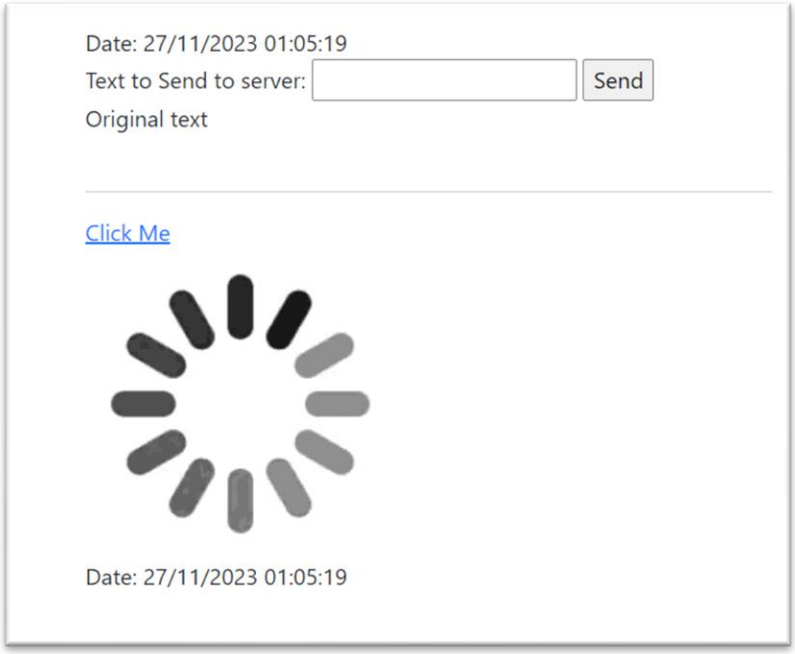
Using Ajax in a Link

Add next code to the view **Index** of the associated with the **Home** controller:

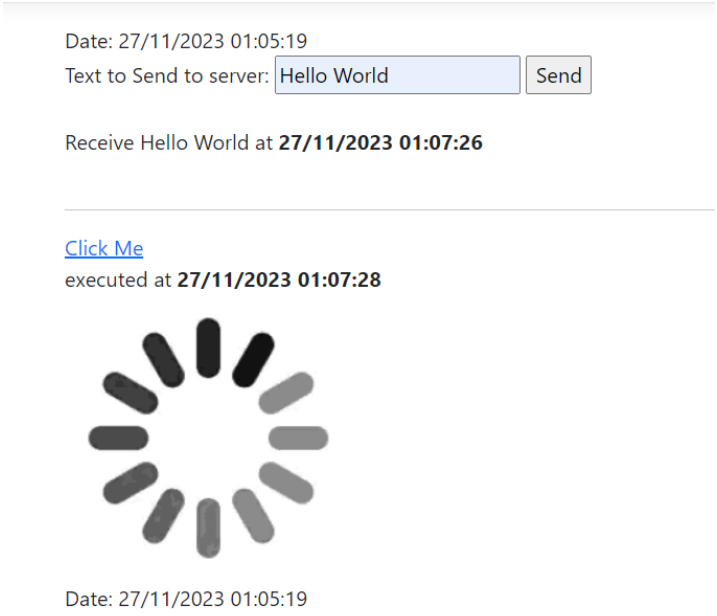
Create a **TestAjaxLink** action, in the Home controller.

```
public string TestAjaxLink()
{
    System.Threading.Thread.Sleep(5000); //Simulate time-consuming processing
    return "executed at <strong> " + DateTime.Now + "</strong>";
}
```

Test the application and click the link



While the page waits for the response from the server, we may do other things, for example, we can use the send button.



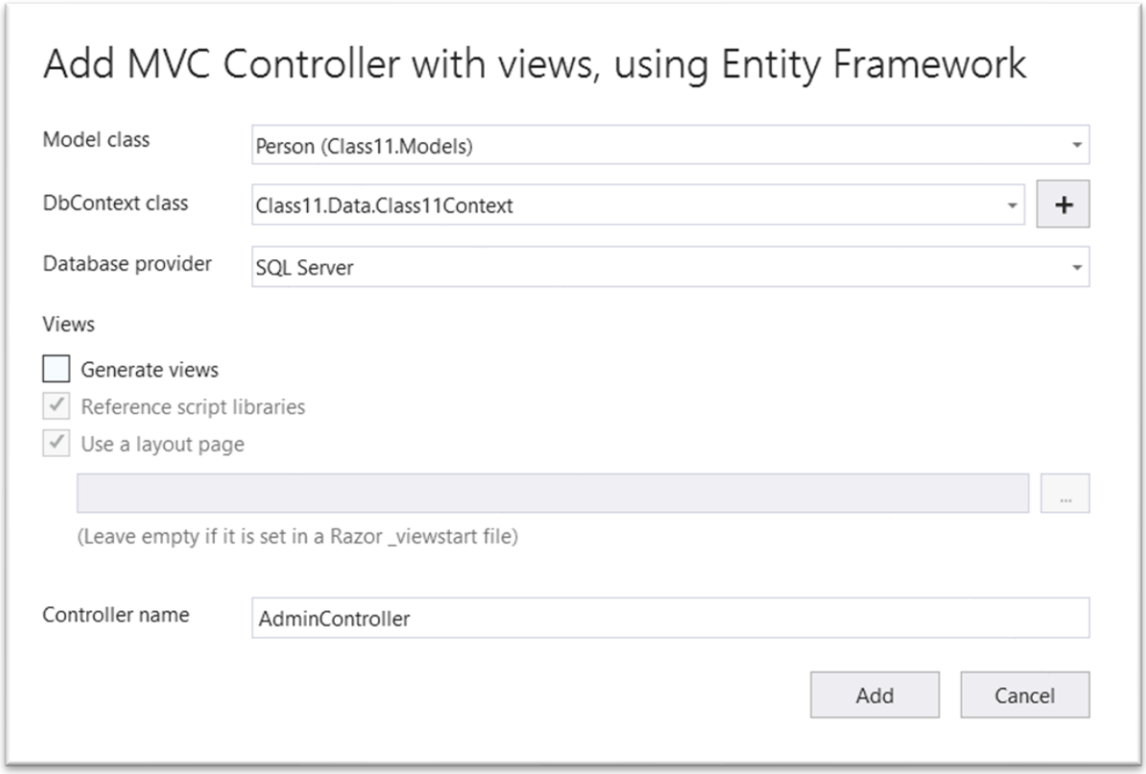
Second step – Using Ajax in CRUD operations

- Create a new class in the Model folder.

```
namespace Class11.Models
{
    0 references
    public class Person
    {
        0 references
        public int Id { get; set; }

        [Required]
        0 references
        public string? Name { get; set; }
    }
}
```

Create a new controller **AdminController** using Entity Framework. This operation will prepare the project with the DbContext and add the connection string to the database.

Delete all actions and views except the Index



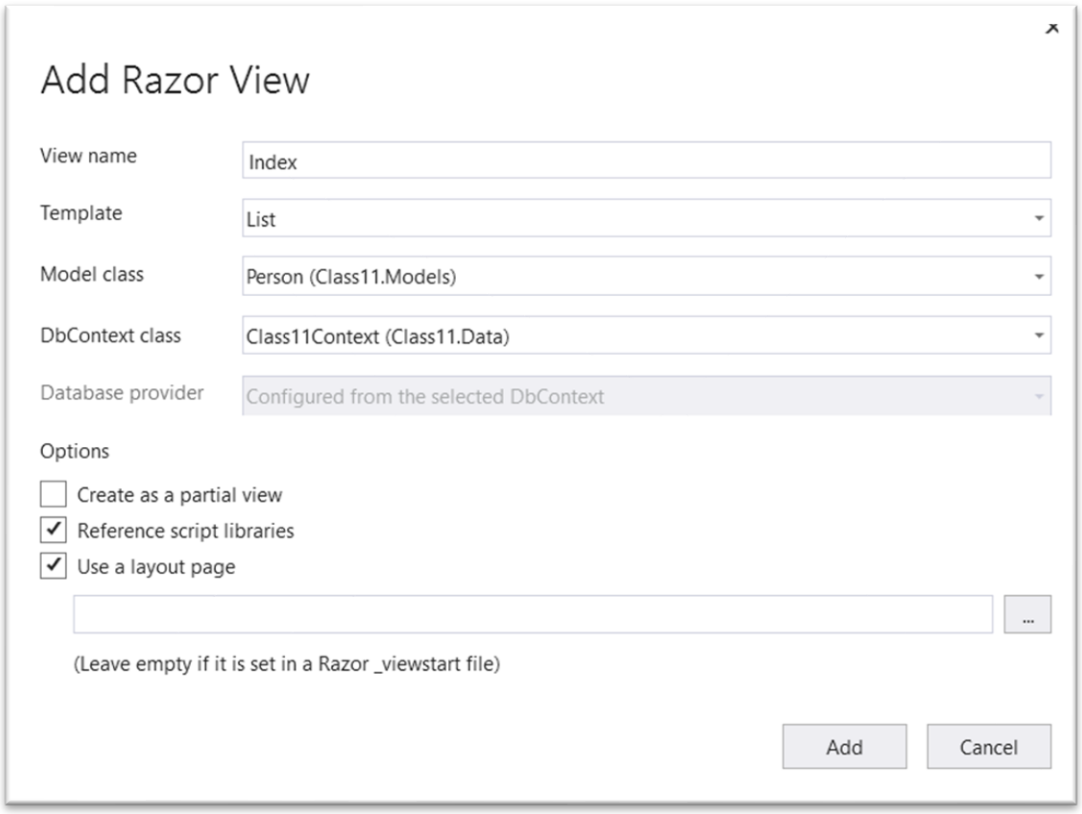
Add a migration

```
PM> add-migration first
```

Update database

```
PM> update-database
```

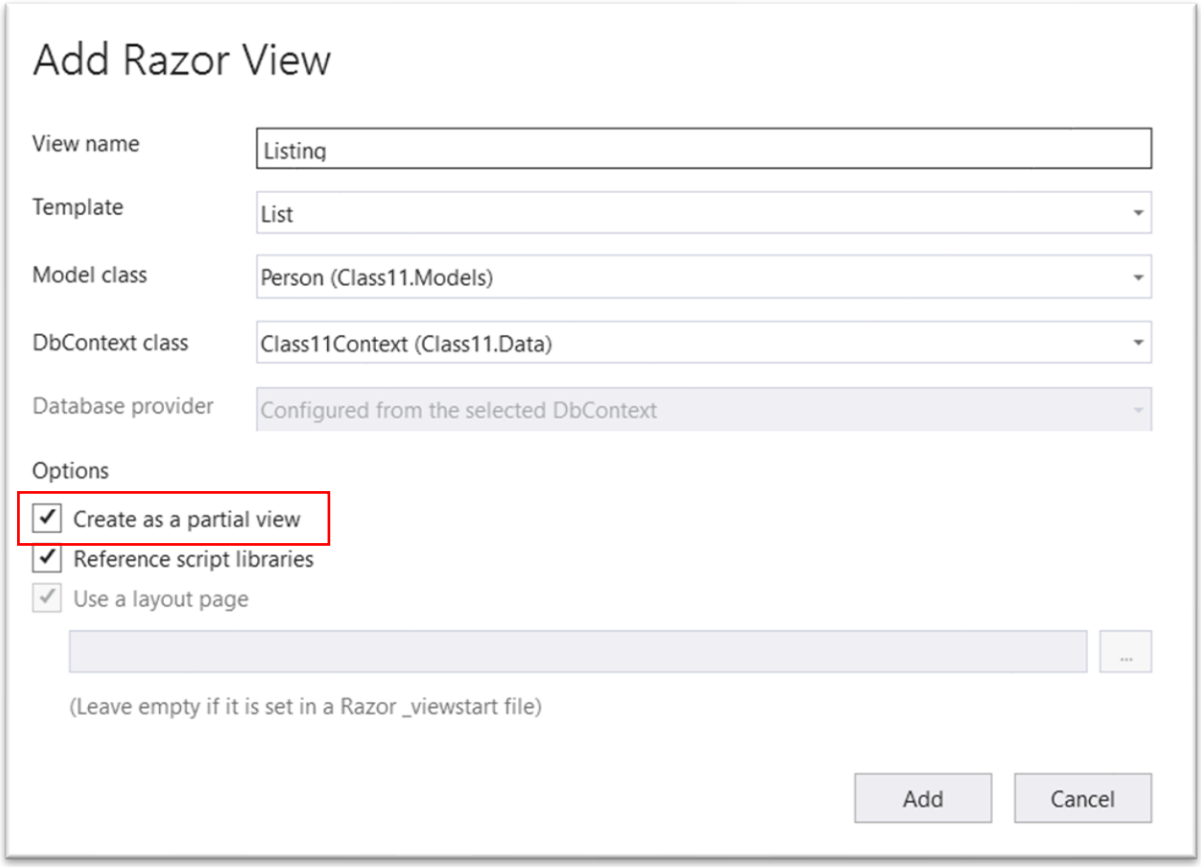
Create the View Index



Alter the code:

```
@model IEnumerable<Class11.Models.Person>

@{
    ViewData["Title"] = "Index";
}

<h1>Index</h1>

<form asp-action="Create">
    <input type="text" name="NewName" />
    <input type="submit" value="Create" class="btn btn-primary" />
</form>

Date: @DateTime.Now
<hr />

<partial name="Listing" model="Model" />

<hr />
Date: @DateTime.Now
```

Create a partial view **Listing.**



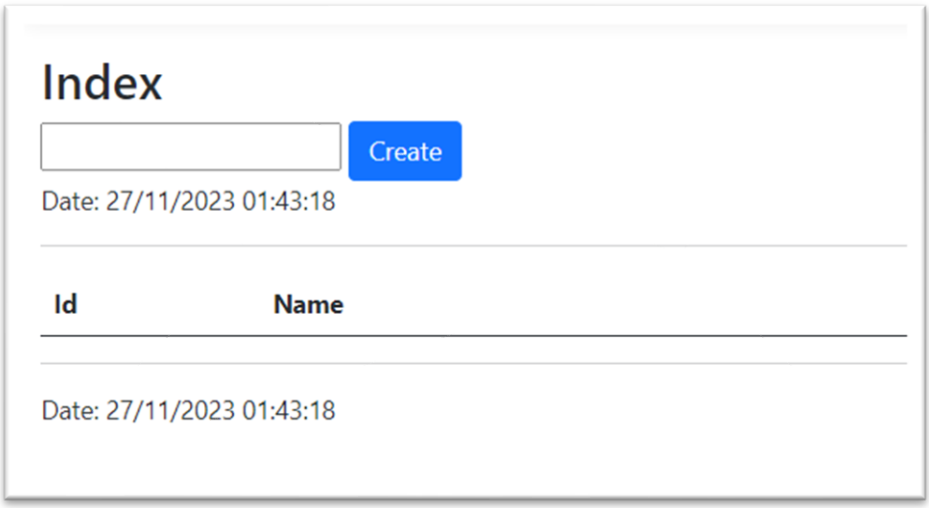
Alter the code to this:

```
@model IEnumerable<Class11.Models.Person>

<table class="table">
    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.Id)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Name)
            </th>
            <th></th>
            <th></th>
        </tr>
    </thead>
    <tbody>
        @foreach (var item in Model)
        {
            <tr>
                <td>
                    @Html.DisplayFor(modelItem => item.Id)
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.Name)
                </td>
                <td>
                    <a asp-action="Edit" asp-route-id="@item.Id">Edit</a>
                </td>
                <td>
                    <a asp-action="Delete" asp-route-id="@item.Id">Delete</a>
                </td>
            </tr>
        }
    </tbody>
</table>
```

Add in **AdminController** the action **Create**

```
0 references
public IActionResult Create(string NewName)
{
    Person newP=new Person();
    newP.Name = NewName;
    _context.Person.Add(newP);
    _context.SaveChanges();

    return PartialView("Listing", _context.Person);
}
```

Run the application at https://localhost:xxxx/Admin



Insert a new Person Name

Did not work well. Let's put Ajax working…

Configure the form to create a new person, in the Ajax form.
Edit the Index view

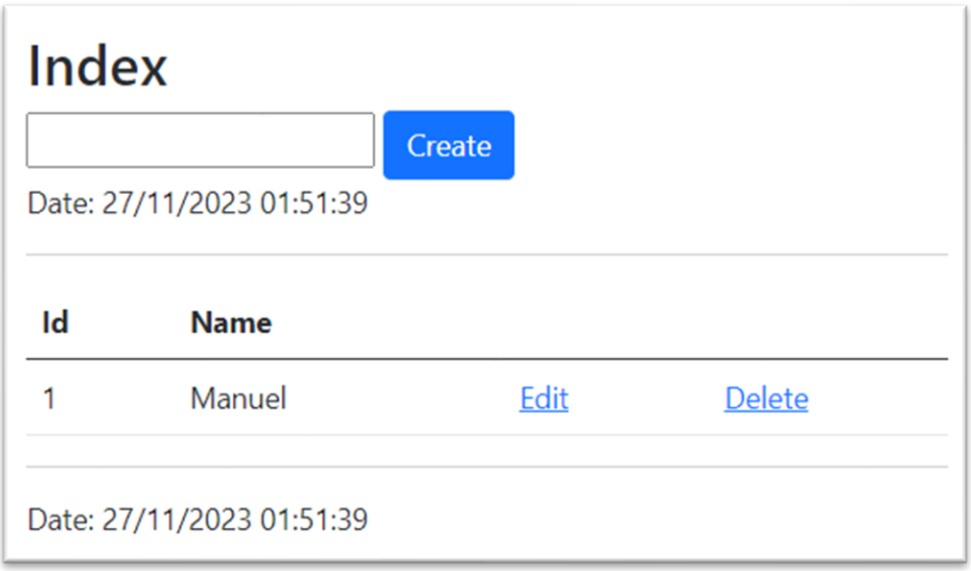
```
<form asp-action="Create" data-ajax="true"
    data-ajax-method="post" data-ajax.mode="replace"
    data-ajax-update="#Tabela" >
    <input type="text" name="NewName" />
    <input type="submit" value="Create" class="btn btn-primary" />
</form>

Date: @DateTime.Now
```

The **data-ajax-update="#Tabela"**, indicate the local where to put the response from server (a partialview **Listing**)

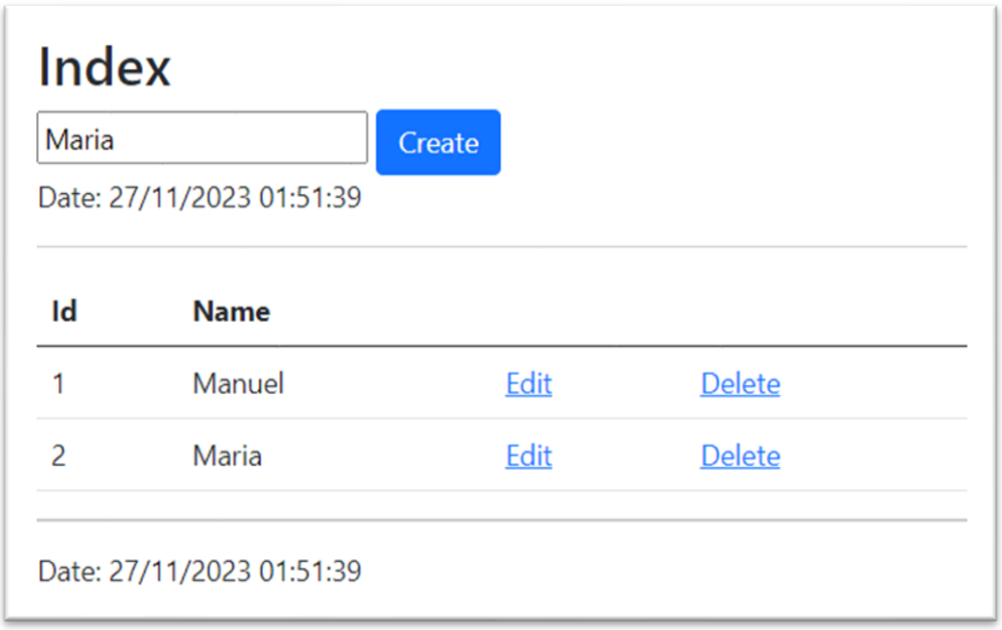
Edit the Listing view and put an **Id="Tabela"** in the *<table>*

```
<table id="Tabela" class="table">
    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.Id)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Name)
            </th>
            <th></th>
            <th></th>
        </tr>
    </thead>
```

Run the application again



Create a new person with name *Maria*.



It works. Only the table are updated.

Alter the link **Edit** in the **Listing** view, to use Ajax

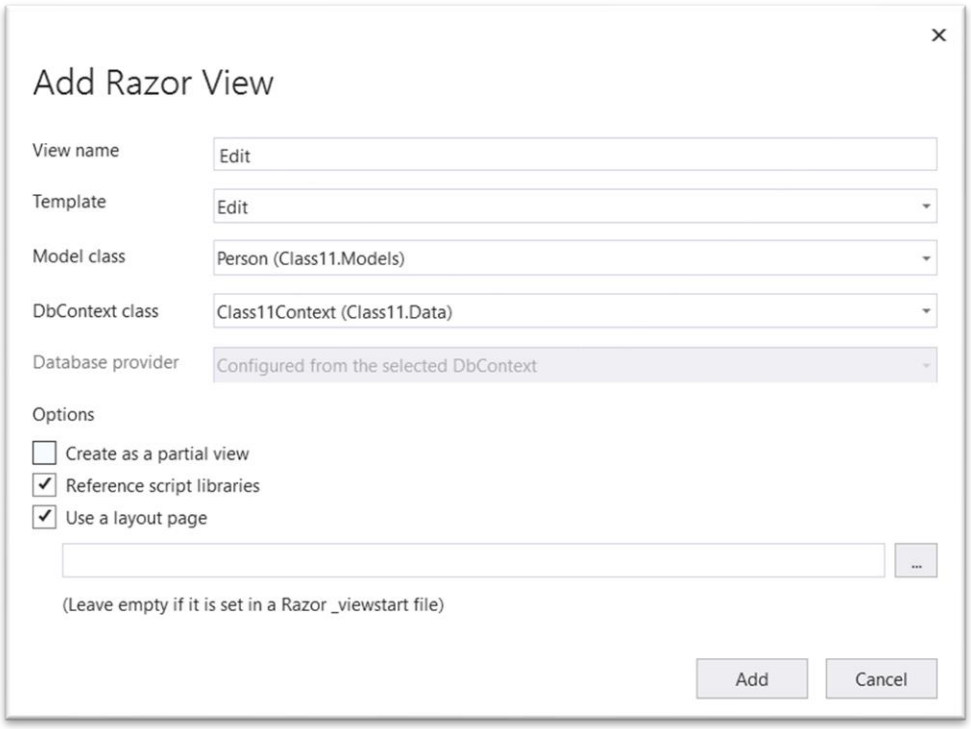
```
@foreach (var item in Model)
{
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.Id)
        </td>
        <td id="Col_@item.Id">
            @Html.DisplayFor(modelItem => item.Name)
        </td>
        <td>
            <a asp-action="Edit" asp-route-id="@item.Id"
            data-ajax="true" data-ajax-method="get"
            data-ajax-mode="replace" data-ajax-update="#Col_@item.Id" >Edit</a>
        </td>
        <td>
```

Each row of a table has an id in the column where the name of person appears. When we click in the link Edit, the result returned by server will be inserted in that position, replace the text of that cell.

Add an **Edit** Action

```
public IActionResult Edit(int Id)
{
    Person a = _context.Person.SingleOrDefault(x => x.Id == Id);
    return PartialView("Edit", a);
}
```

Create a partial view Edit with Template Edit



Alter the code:

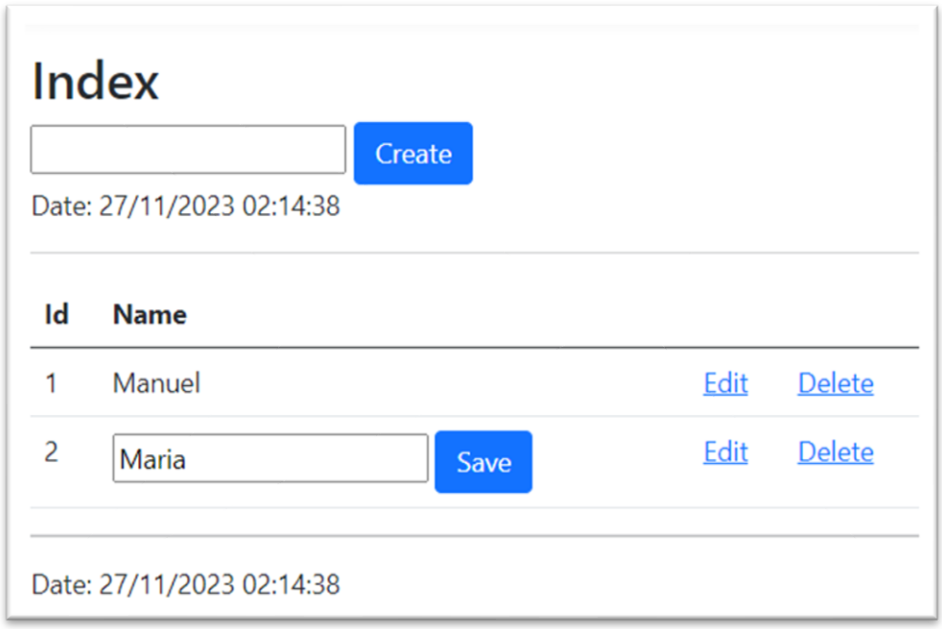
```
@model Class11.Models.Person

@* Transform the form in AJAX *@

<form asp-action="Edit" data-ajax="true" data-ajax-method="post"
      data-ajax-update="#Col_@Model.Id" data-ajax-mode="replace">

    <div asp-validation-summary="ModelOnly" class="text-danger"></div>
    <input type="hidden" asp-for="Id" />
    <div>
        <input asp-for="Name" />
        <span asp-validation-for="Name" class="text-danger"></span>

        <input type="submit" value="Save" class="btn btn-primary" />
    </div>
</form>


@section Scripts {
    @{
        await Html.RenderPartialAsync("_ValidationScriptsPartial");
    }
}
```

Create the action Edit that process the previous form

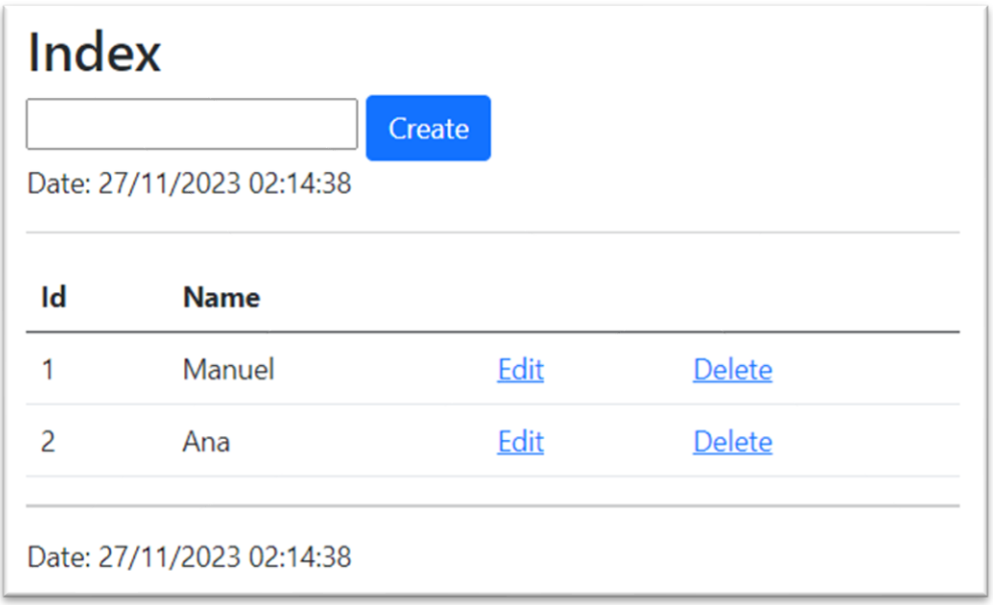
```
[HttpPost]
0 references
public string Edit(int id, Person p)
{
    _context.Update(p);
    _context.SaveChanges();
    return p.Name;
}
```

This action returns a string with the name of the person.

Run the application



The edit form appears in the row where we click edit link.
Change the name and Save.



The name changes and only the cell are updated.

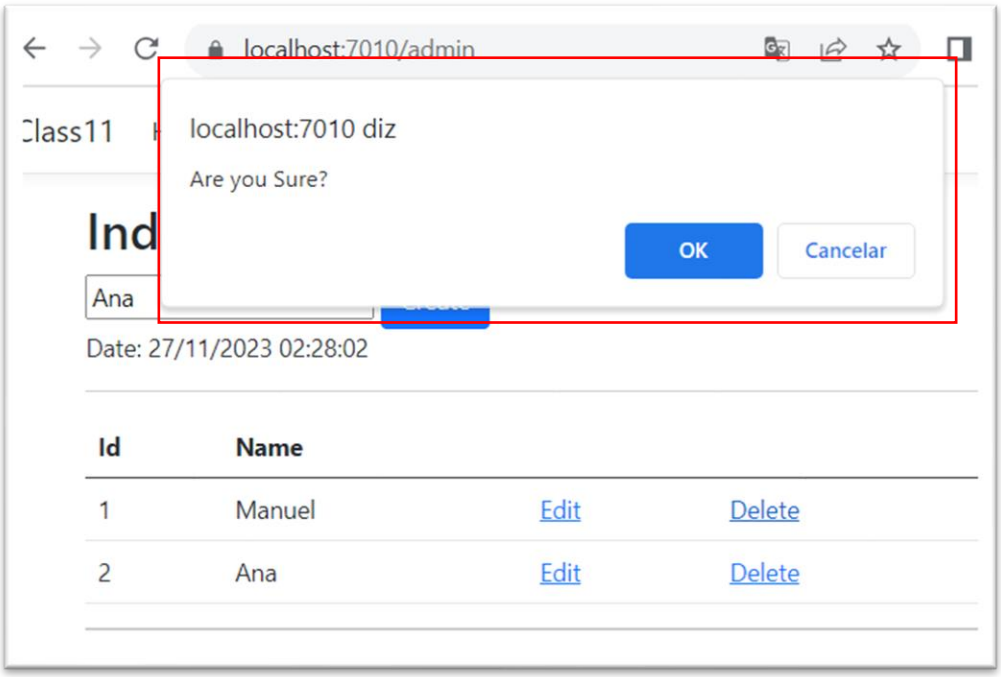
Alter the link Delete in the Listing view, to use Ajax

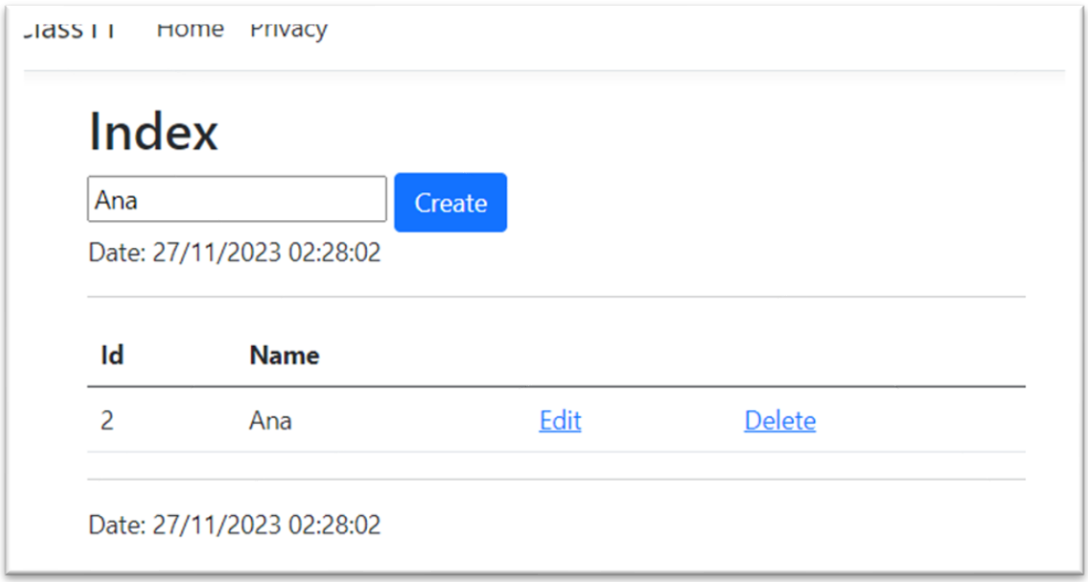
```
<td>
    <a asp-action="Delete" asp-route-id="@item.Id"
    data-ajax="true" data-ajax-method="get"
    data-ajax-mode="replace" data-ajax-update="#Tabela"
    data-ajax-confirm="Are you Sure?" >Delete</a>
</td>
```

Create the action Delete

```
0 references
public IActionResult Delete(int id)
{
    Person p=_context.Person.SingleOrDefault(x=>x.Id == id);
    _context.Person.Remove(p);
    _context.SaveChanges();

    return PartialView("Listeng", _context.Person);
}
```

Run the application and test the delete link





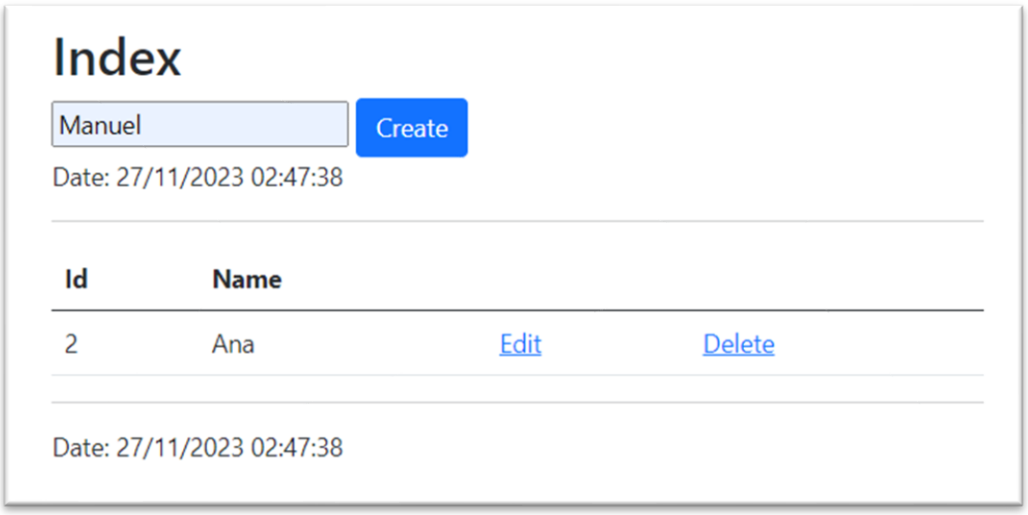
Put in the menu an option to the action **Index** of **AdminController**

Extra:
When we create a new Person, the Name of person remains in the text box. If we can erase that, we may use javascript to do that.

Edit the Index View

```
<h1>Index</h1>

<form asp-action="Create" data-ajax="true"
    data-ajax-method="post" data-ajax-mode="replace"
    data-ajax-update="#Tabela" data-ajax-success="sucesso">
    <input id="inp" type="text" name="NewName" />
    <input type="submit" value="Create" class="btn btn-primary" />
</form>

Date: @DateTime.Now
<hr />

<partial name="Listing" model="Model" />

<hr />
Date: @DateTime.Now

@* javascript with function that are executed when the AJAX operatio have success *@
<script>
    function sucesso() {
        $("#inp").val("");
    }
</script>
```

Test again to view the effect



After creating Manuel person the form is cleared.