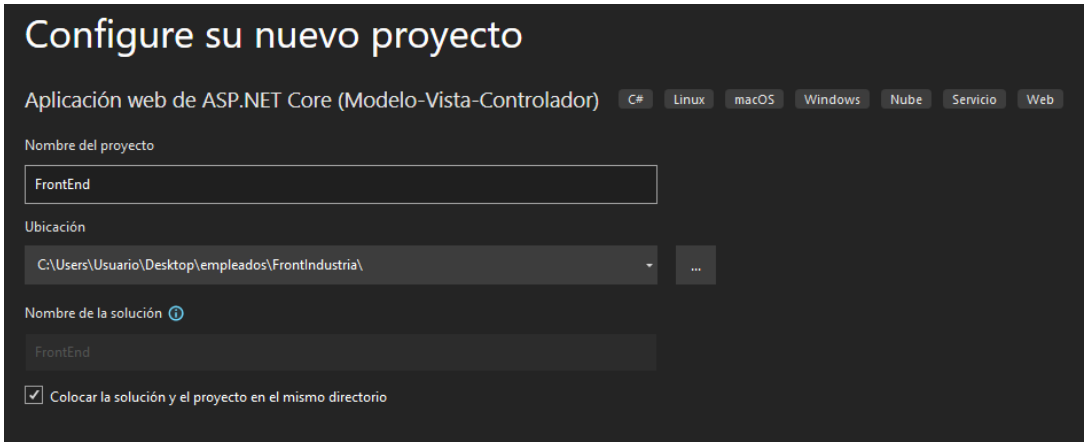
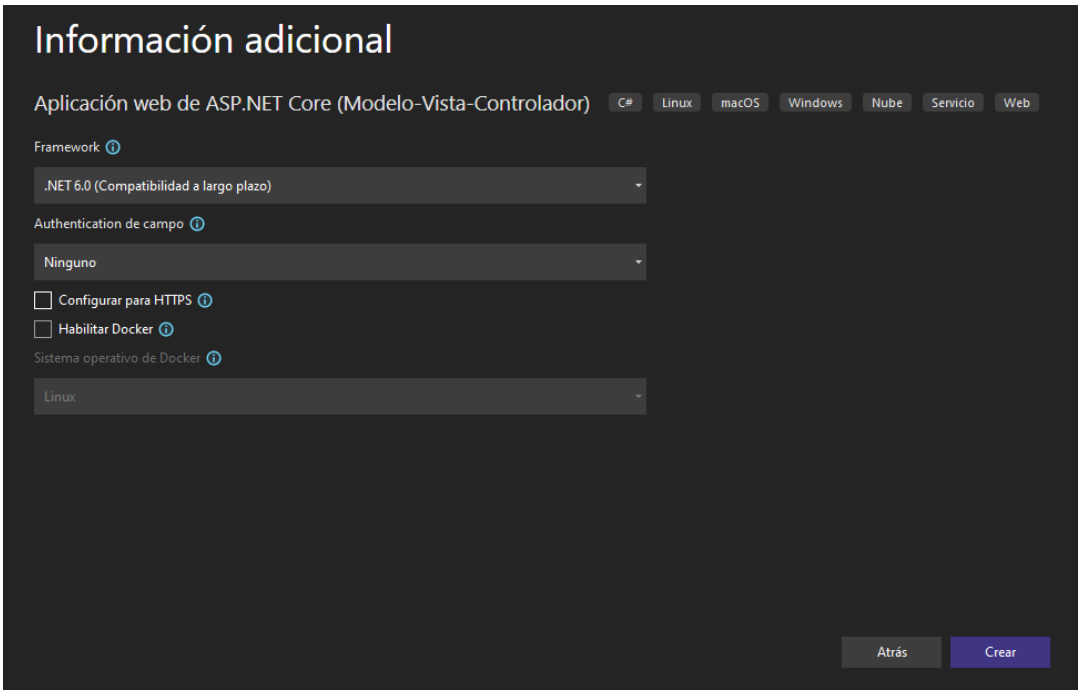
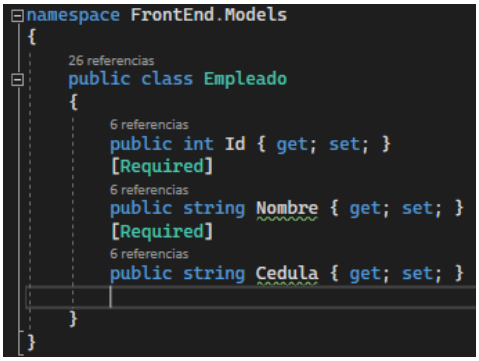
CONSUMIR WEB API DESDE .NET CORE



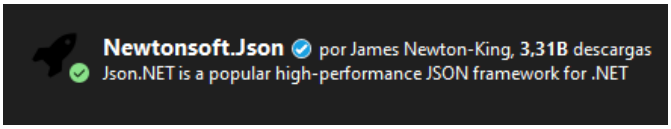


1. Se crea el modelo Carpeta Models para consumir la entidad que viene de la petición http.



```
namespace FrontEnd.Models
{
    26 referencias
    public class Empleado
    {
        6 referencias
        public int Id { get; set; }
        [Required]
        6 referencias
        public string Nombre { get; set; }
        [Required]
        6 referencias
        public string Cedula { get; set; }

    }
}
```

2. Instalar Newtonsoft.Json para deserializar los objetos que vienen como string en el llamado http



3. Se crea las variables de conexión en el archivo appsettings.json

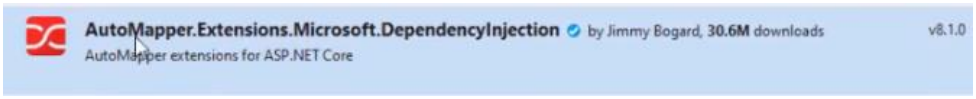
```
"ApiSetting": {
  //"usuario": "",
  //"clave": "",
  "baseUrl": "https://localhost:7153/api/empleados"

}
```

4. Se crean los DTOs para Empleado

```
using System.ComponentModel.DataAnnotations;
namespace FrontEnd.DTOs
{
    0 referencias
    public class EmpleadoCreacionDTO
    {
        [Required]
        0 referencias
        public string Nombre { get; set; }
        [Required]
        0 referencias
        public string Cedula { get; set; }
    }
}
```

```
namespace FrontEnd.DTOs
{
    2 referencias
    public class EmpleadoDTO
    {
        0 referencias
        public int Id { get; set; }
        0 referencias
        public string Nombre { get; set; }
        0 referencias
        public string Cedula { get; set; }
    }
}
```

5. Instalar AutoMapper para mappear Empleado.



6. En el archivo program.cs o startup.cs agregamos el servicio.

```
builder.Services.AddAutoMapper(typeof(Program));
```

7. Creamos el archivo AutoMapperProfiles.cs y lo heredamos de " :Profile " y agregamos el mapeo de estudiante y lo hacemos de doble vía para consulta, también hacemos uno para creación.

```
using AutoMapper;
using FrontEnd.Models;
using FrontEnd.DTOs;

namespace FrontEnd.Utilidades
{
    1 referencia
    public class AutoMapperProfiles: Profile
    {
        0 referencias
        public AutoMapperProfiles()
        {
            CreateMap<Empleado, EmpleadoDTO>().ReverseMap();
            CreateMap<EmpleadoCreacionDTO, Empleado>().ReverseMap();
        }
    }
}
```

8. Se crea la interfaz IServicioApi con todos los métodos http CRUD

```
using FrontEnd.Models;

namespace FrontEnd.Servicios
{
    4 referencias
    public interface IServicioApi
    {

        3 referencias
        Task<List<Empleado>> Lista();
        2 referencias
        Task<Empleado> Obtener(int id);
        1 referencia
        Task<bool> Guardar(Empleado objetoEmpleado);
        1 referencia
        Task<bool> Editar(Empleado objetoEmpleado);

        1 referencia
        Task<bool> Eliminar(int id);
    }
}
```

9. Se agrega la siguiente línea en program.cs para poder usar la interfaz IServicioApi en todas las instancias. Debe estar creado (ServicioApi y IServicioApi)

```
builder.Services.AddScoped<IServicioApi, ServicioApi>();
```

10. Se crea la clase ServicioApi que hereda de IServicioApi y se implementan los métodos de la imagen anterior.

```
using FrontEnd.Models;
using FrontEnd.DTOs;
using Newtonsoft.Json;
using System.Text;
using AutoMapper;

namespace FrontEnd.Servicios
{
    2 referencias
    public class ServicioApi : IServicioApi
    {
        private static string _baseUrl;
        private readonly IMapper mapper;

        0 referencias
        public ServicioApi(IMapper mapper)
        {
            var builder = new ConfigurationBuilder().SetBasePath(Directory.GetCurrentDirectory()).AddJsonFile("appsettings.json").Build();
            _baseUrl = builder.GetSection("ApiSetting:baseUrl").Value;
            this.mapper = mapper;
        }

        4 referencias
        public async Task<List<Empleado>> Lista()
        {
            List<Empleado> lista = new List<Empleado>();
            using (var httpClient = new HttpClient())
            {
                var respuesta = await httpClient.GetAsync(_baseUrl);
                var respuestaString = await respuesta.Content.ReadAsStringAsync();
                var alista = JsonConvert.DeserializeObject<List<EmpleadoDTO>>(respuestaString);
                lista = mapper.Map<List<Empleado>>(alista);
            }
            return lista;
        }

        3 referencias
        public async Task<Empleado> Obtener(int id)
        {
            Empleado empleado = new Empleado();
            using (var httpClient = new HttpClient())
            {
                var respuesta = await httpClient.GetAsync(_baseUrl + "/" + id);
                var respuestaString = await respuesta.Content.ReadAsStringAsync();
                var alista = JsonConvert.DeserializeObject<EmpleadoDTO>(respuestaString);
                empleado = mapper.Map<Empleado>(alista);
            }
            return empleado;
        }
```

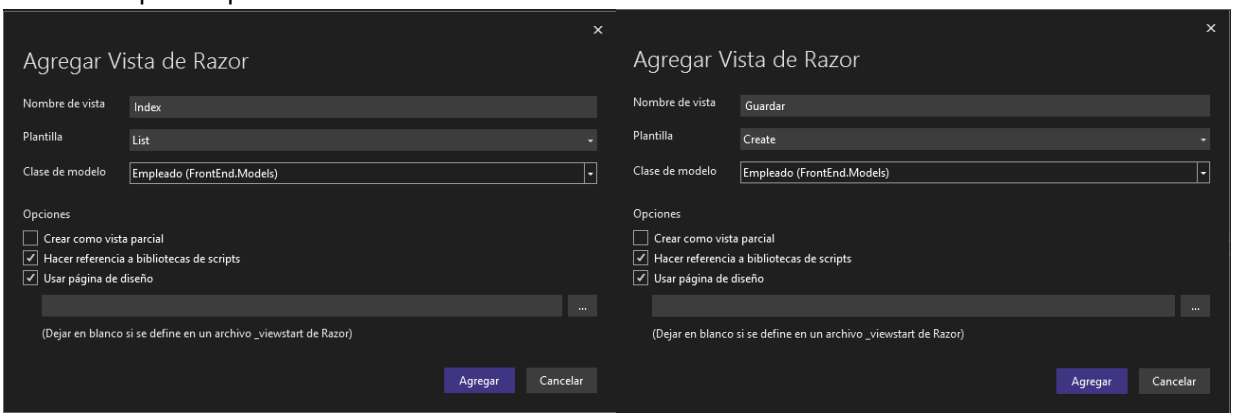
```
2 referencias
public async Task<bool> Guardar(Empleado objetoEmpleado)
{
    EmpleadoCreacionDTO empleadoDTO = new EmpleadoCreacionDTO();

    using (var httpClient = new HttpClient())
    {
        empleadoDTO = mapper.Map<EmpleadoCreacionDTO>(objetoEmpleado);
        var respuesta = await httpClient.PostAsJsonAsync(_baseUrl, empleadoDTO);
        if(respuesta.IsSuccessStatusCode)
        {
            return true;
        }
    }
    return false;
}

2 referencias
public async Task<bool> Editar(Empleado objetoEmpleado)
{
    EmpleadoCreacionDTO empleadoDTO = new EmpleadoCreacionDTO();
    using (var httpClient = new HttpClient())
    {
        empleadoDTO = mapper.Map<EmpleadoCreacionDTO>(objetoEmpleado);
        var respuesta = await httpClient.PutAsJsonAsync($"{_baseUrl}/{objetoEmpleado.Id}", empleadoDTO);
        if (respuesta.IsSuccessStatusCode)
        {
            return true;
        }
    }
    return false;
}

2 referencias
public async Task<bool> Eliminar(int id)
{
    using (var httpClient = new HttpClient())
    {
        var respuesta = await httpClient.DeleteAsync($"{_baseUrl}/{id}");
        if (respuesta.IsSuccessStatusCode)
        {
            return true;
        }
    }
    return false;
}
```

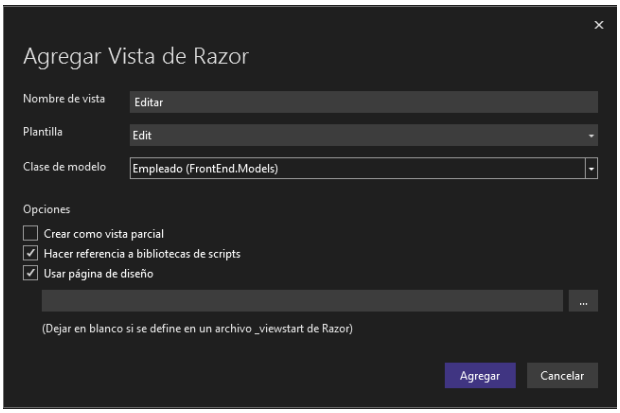
11. Se crea el controlador para la entidad. En la carpeta Controller, click derecho agregar->Controlador-> Controlador de MVC en blanco

```
using Microsoft.AspNetCore.Mvc;
using FrontEnd.Models;
using FrontEnd.Servicios;
using System.Diagnostics;

namespace FrontEnd.Controllers
{
    1 referencia
    public class EmpleadoController : Controller
    {
        private readonly IServicioApi servicioApi;

        0 referencias
        public EmpleadoController(IServicioApi servicioApi)
        {
            this.servicioApi = servicioApi;
        }

        0 referencias
        public async Task<IActionResult> Index()
        {
            List<Empleado> list = await servicioApi.Lista();
            return View(list);
        }

        0 referencias
        public IActionResult Guardar()
        {
            return View();
        }

        [HttpPost]
        0 referencias
        public async Task<IActionResult> Guardar(Empleado empleado)
        {
            var respuesta = await servicioApi.Guardar(empleado);
            if (respuesta)
                return RedirectToAction("Index");
            else
                return View();
        }


        0 referencias
        public async Task<IActionResult> Editar(int id)
        {
            Empleado empleado = await servicioApi.Obtener(id);
            return View(empleado);
        }
```

```
        [HttpPost]
        0 referencias
        public async Task<IActionResult> Editar(Empleado empleado)
        {
            var respuesta = await servicioApi.Editar(empleado);
            if (respuesta)
                return RedirectToAction("Index");
            else
                return View();
        }

        [HttpGet]
        0 referencias
        public async Task<IActionResult> Eliminar(int id)
        {
            var respuesta = await servicioApi.Eliminar(id);
            if (respuesta)
                return RedirectToAction("Index");
            else
                return View();
        }
    }
}
```

12. Creamos las vistas Para los métodos http, tener en cuenta que para el método editar y eliminar solo se crea la vista en el que viene acompañado de [HttpPost]. Agregar Vista -> Vista de Razor y se escoge la plantilla y el modelo empleado para mostrar.





13. En el archivo program.cs escogemos la página de inicio, escogemos el controlador (Controller) y la vista (action).

```
app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Empleado}/{action=Index}/{id?}");
```

14. Instalar el siguiente paquete para poder actualizar nuestras vistas sin necesidad de parar la ejecución.

15. Nos crea la vista con lo requerido y le hacemos los cambios que necesitemos. Para este ejemplo modificamos los espacios para editar y eliminar filas de la lista.

```
@model IEnumerable<FrontEnd.Models.Empleado>
@{
    ViewData["Title"] = "Index";
}
<h1>Lista Empleados</h1>
<p>
    <a asp-action="Guardar" asp-controller="Empleado">Create New</a>
</p>
<table class="table">
    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.Id)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Nombre)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Cedula)
            </th>
            <th></th>
        </tr>
    </thead>
    <tbody>
@foreach (var item in Model) {
        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.Id)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Nombre)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Cedula)
            </td>
            <td>
                <a asp-action="Editar" asp-controller="Empleado" class="btn btn-primary" asp-route-id="@item.Id">Editar</a>
                <a asp-action="Eliminar" asp-controller="Empleado" class="btn btn-danger" asp-route-id="@item.Id">Eliminar</a>
            </td>
        </tr>}
    </tbody>
</table>
```

16. En este ejemplo ocultamos label e input del Id porque no es necesario y modificamos los botones.

```
@model FrontEnd.Models.Empleado
@{
    ViewData["Title"] = "Guardar";
}
<h1>Guardar</h1>
<h4>Empleado</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Guardar" method="post">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="Id" class="control-label" hidden></label>
                <input asp-for="Id" class="form-control" type="hidden" />
                <span asp-validation-for="Id" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Nombre" class="control-label"></label>
                <input asp-for="Nombre" class="form-control" />
                <span asp-validation-for="Nombre" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Cedula" class="control-label"></label>
                <input asp-for="Cedula" class="form-control" />
                <span asp-validation-for="Cedula" class="text-danger"></span>
            </div>
            <div class="form-group">
                <input type="submit" value="Create" class="btn btn-primary" />
                <a asp-action="Index" class="btn btn-warning">Back to List</a>
            </div>
        </form>
    </div>
</div>
@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}
```

17. En este ejemplo ocultamos label e input del Id porque no es necesario y modificamos los botones.

```
@model FrontEnd.Models.Empleado
@{
    ViewData["Title"] = "Editar";
}
<h1>Editar</h1>
<h4>Empleado</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Editar" method="post">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="Id" class="control-label" hidden></label>
                <input asp-for="Id" class="form-control" type="hidden" />
                <span asp-validation-for="Id" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Nombre" class="control-label"></label>
                <input asp-for="Nombre" class="form-control" />
                <span asp-validation-for="Nombre" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Cedula" class="control-label"></label>
                <input asp-for="Cedula" class="form-control" />
                <span asp-validation-for="Cedula" class="text-danger"></span>
            </div>
            <div class="form-group">
                <input type="submit" value="Save" class="btn btn-primary" />
                <a asp-action="Index" class="btn btn-warning">Back to List</a>
            </div>
        </form>
    </div>
</div>
@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}
```

18. Con toda la configuración anterior podemos ya consumir nuestro Web Api.