

Reglas de organización de APIs en ASP.Net Core

Código del documento

06/08/2021

Dawid Lizurej

Departamento IT

Contenido

[1. Introducción 4](#_Toc97300072)

[2. Convención de nombres de la solución 4](#_Toc97300073)

[3. Estructura de carpetas 4](#_Toc97300074)

[3.1 Controllers 4](#_Toc97300075)

[1.3.1. Filtros 4](#_Toc97300076)

[2.3.1. Atributos 4](#_Toc97300077)

[3.3.1. DTO 4](#_Toc97300078)

[4.3.1. Interfaces 4](#_Toc97300079)

[3.2 Servicios 5](#_Toc97300080)

[1.3.2. Helpers 5](#_Toc97300081)

[2.3.2. Modelos 5](#_Toc97300082)

[3.3.2. Interfaces 5](#_Toc97300083)

[3.3 ServiciosCompartidos 5](#_Toc97300084)

[3.4 BBDD 5](#_Toc97300085)

[1.3.4. Modelos 6](#_Toc97300086)

[4. Convención de nombres de fichero 6](#_Toc97300087)

[5. Convención de nombres de clases 6](#_Toc97300088)

[6. Convención de nombres de métodos 6](#_Toc97300089)

[7. Ejemplo de API segmentada por negocio / modulo 7](#_Toc97300090)

[8. Ejemplo de API segmentada por tipo de objeto 8](#_Toc97300091)

Control Documental

|  |  |  |
| --- | --- | --- |
| Redactado por: |  |  |
| Revisado por: |  |  |
| Aprobado por: |  |  |

Histórico de modificaciones

|  |  |  |  |
| --- | --- | --- | --- |
| CÓDIGO | VERSIÓN | FECHA | COMENTARIO |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Introducción

En este documento explicará cual es la convección de estructural por carpetas de un proyecto de ICP en ASP.Net Core.

# Convención de nombres de la solución

Los nombres de los proyectos deberán de llamarse **Icp.NombreCliente.NombreProyecto**

En el caso de que se trate de proyectos internos deberán de llamarse **Icp.NombreProyecto**

# Estructura de carpetas

Aquí explicaremos como colocar y organizar los ficheros del proyecto.

## Controllers

Los controllers deberán de guardarse todos dentro de un directorio llamado “Controllers” y cada Controller deberá de tener su propio directorio. Dentro de este, mínimo deberemos de encontrar al propio Controller.

Cada Controller tiene que nominarse en función de lo que vaya a hacer o representar. Por ejemplo: **PedidosController**.

A su vez, el directorio PedidosController debe de tener los siguientes subdirectorios:

### Filtros

En el caso de que un controlador necesitase tener algún filtro propio, este debería de ir dentro de una carpeta “Filtros” dentro del directorio del controlador en cuestión. En cambio, si este filtro aplica a mas de un controller, este deberá de almacenarse dentro del directorio “Controllers”

### Atributos

En el caso de que un controlador necesitase tener algún atributo propio, este debería de ir dentro de una carpeta “Atributos” dentro del directorio del controller en cuestión. En cambio, si este filtro aplica a más de un controller, este deberá de almacenarse dentro del directorio “Controllers”

### DTO

Todos los archivos DTO deben de guardarse en el directorio del controller que los va a utilizar. Si un DTO fuese a ser utilizado por mas de un controller deberemos de crear una un nivel por encima.

### Interfaces

Todos los archivos de las interfaces deben de guardarse en el directorio del controller que los va a utilizar. Si una interfaz fuese a ser utilizado por más de un controller deberemos de crear una un nivel por encima.

## Servicios

Los servicios son los encargados de manejar toda la lógica de negocio. Existen dos tipos de servicio: los que son y no son compartidos. La idea de los servicios compartidos es que estos manejen aquella lógica que puede ser utilizada en cualquier parte de la aplicación. Por otro lado, los servicios no compartidos son aquellos que su función solo es una. Por ejemplo: **PedidosService.**

Otra de las funciones es la de interactuar con el contexto de base de datos y es por ello por lo que son los únicos que van a ejecutar cualquier instrucción que posteriormente tenga algún tipo de efecto en la base de datos.

Los servicios deberán de ir guardados en el directorio “Servicios” y cada servicio a su vez deberá de tener su propio directorio. Por ejemplo: **PedidosService**

Este directorio como mínimo deberá de estar compuesto por el fichero **PedidosSerice.cs** y, además, podrá tener los siguientes directorios:

### Helpers

Aquellas funciones que se repiten mucho dentro del propio servicio. En el caso de que alguna función se use en más de un servicio, esta deberá de ir en un directorio dentro de “Servicios”

### Modelos

Todas las clases que se necesiten utilizar dentro de algún servicio deberán de guardarse en este directorio. Es importante diferenciar los modelos de los DTO puesto que un modelo puede ser diferente a un DTO y no contener las mismas propiedades. Por lo general, el modelo de un servicio será igual que un DTO, pero en los casos en los que queramos filtrar la información que se devuelve desde el Controller debemos de únicamente retornar las propiedades estrictamente necesarias a través de una clase DTO.

### Interfaces

Todos los archivos de las interfaces deben de guardarse en el directorio del servicio que los va a utilizar. Si una interfaz fuese a ser utilizado por más de un controller deberemos de crear una un nivel por encima.

## ServiciosCompartidos

Un servicio compartido es un servicio normal pero que se puede llegar a utilizar en más de un Controller.

Además de tener su propio directorio padre “ServiciosCompartidos” y por lo demás, se deben de gestionar igual que un servicio convencional.

## BBDD

En este directorio guardaremos todos los contextos y modelos que necesitemos para gestionar el acceso y manipulación de datos datos. Cada servicio deberá tener su pareja que gestione las configuraciones de acceso a datos.

Dentro del directorio “BBDD” crearemos un directorio por cada conexión que demos de alta. Por ejemplo: **PedidosContext**.

Las cadenas de conexión de requeridas por los contextos se guardarán en la configuración appSettings.json de momento.

Si se desea, se puede generar una clase parcial por cada vista, procedimiento almacenado o función que se quiera añadir al contexto. Para evitar así, tener una clase con demasiadas líneas de código.

### Modelos

Del mismo modo que en los Controller y los Servicios, crearemos una carpeta con los modelos que requiera el contexto.

# Convención de nombres de fichero

Todos los nombres de los ficheros tienen que estar nominados con nomenclatura Pascal. Y el nombre tiene que ser el mismo que el de la clase a la que representan.

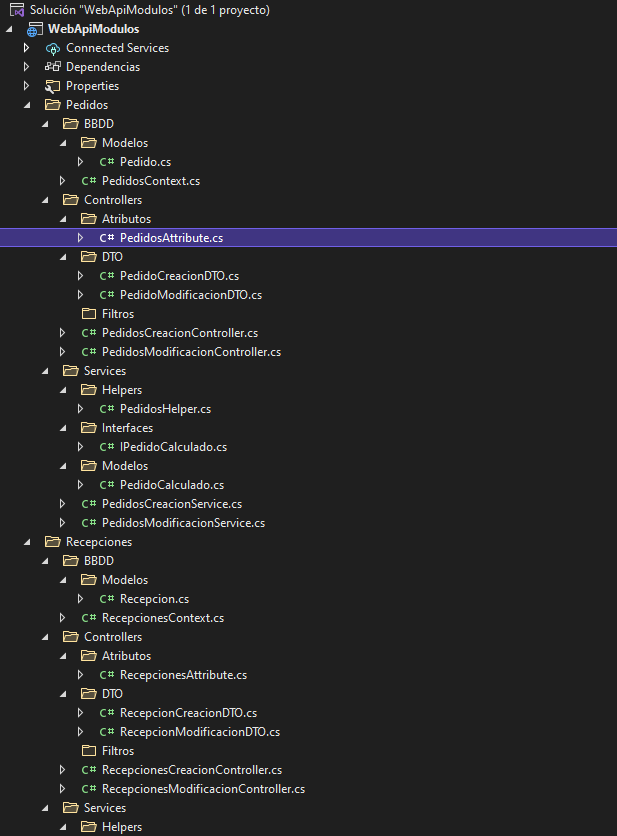
# Convención de nombres de clases

Todos los nombres de las clases tienen que estar nominados con nomenclatura Pascal.

# Convención de nombres de métodos

Todos los nombres de los métodos tienen que estar nominados con nomenclatura Pascal. Tienen que contener al menos dos palabras en la que una de ellas tiene que describir la acción que va a realizar.

# Ejemplo de API segmentada por negocio / modulo



# Ejemplo de API segmentada por tipo de objeto

