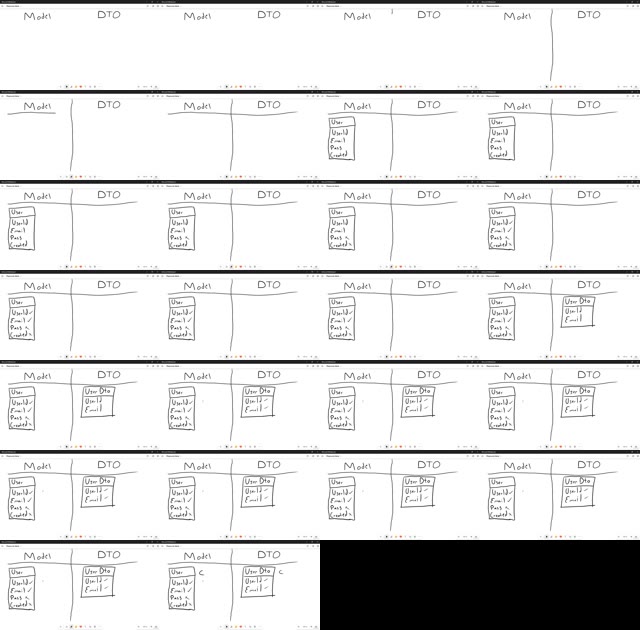
35. DTOs y Models



**2:02 / 2:05**

En esta sección nos vamos a centrar ya a un flujo de trabajo más realista en lo que es las aplicaciones

backend y vamos a ver ciertos conceptos que son necesarios en este punto.

Vamos a ver el concepto de que es un modelo y un DTO.

Bueno, en el caso del modelo va a ser la representación de una entidad en la base de datos.

Por ejemplo, aquí vamos a tener un modelo llamado usuario y el usuario va a ser la representación de

lo que tienes en tu base de datos, en este caso el user ID y en tu tabla de la base de datos sería

una representación que sería un espejo de ella tendrías el ID y tendrías un correo, tendrás el password,

tendrías también la fecha de creación.

Pero que es el problema que involucra trabajar solamente con los modelos para toda la interacción de

tu sistema cuando involucra que algunas pantallas no van a ocupar el password.

Para qué quieres el password?

De hecho, para qué mostrarías un password o para que mostrarías cuando se creó?

O a lo mejor solamente necesitas el correo y el user id.

En ciertos esquemas no?

Bueno, el DTO viene a resolver esa parte, el Dto.

Que sus siglas significan Datatransfer Object es un objeto para transferencia de data.

Es un objeto que se va a utilizar para transferirse entre capas y este dto.

Bueno, en el caso que tendríamos a lo mejor un dto llamado User dto, podríamos manejarlo con una entidad

distinta, la cual solamente tenga lo necesario para la transferencia de data.

Es decir, a lo mejor en unos grids que vamos a mostrar o un JSON que vamos a retornar en nuestro backend,

solamente necesitamos estos dos campos.

Entonces hacemos una entidad que tenga solamente esos dos campos y es la que vamos a utilizar.

La ventaja de los DTO es que podemos reducir también la transferencia de cuánta información estás transmitiendo

entre cada capa.

O cuánta información vas a ir a la base de datos por ella.

Es decir, optimizas tu aplicación porque hay muchas veces que la entidad tiene muchísimos campos.

Puede ser el caso que tenga 20 campos y no necesita esos 20 campos.

Creas un DTO con la información que necesitas solamente transferir entre tus capas y de esa manera ya

cumples con ese objetivo de estar solamente mostrando lo que necesitas.

Y si te preguntas qué son estas dos cosas que estamos mostrando, pues son dos clases.

Esta es una clase y esta también es una clase.

Y ahorita vamos a ver lo que es el DTO ya con código.

36. Creación de DTO

En esta clase vamos a hacer un dto.

Pero vamos a hacerlo a partir de lo que vamos a hacer en toda esta sección.

Esta sección lo que vamos a hacer es conectarnos a un servicio externo.

Hay una página llamada Jsonplaceholder, Jsonplaceholder y Typekit y esta página tiene es un fake service,

es decir, es un servicio que solamente sirve para hacer pruebas, para estar haciendo, simulando un

servicio.

Tercero.

Entonces, lo que vamos a ver en este aprendizaje es cómo conectarnos a un servicio tercero y que involucre

hacer uso de ciertas entidades que vamos a ciertas clases que vamos a hacer uso en esta.

En esta sección, para que veas cómo te puedes conectar, obtener información, poderla enlazar en este

caso un DTO y que puedas retornarlo en tu controller de manera asíncrona.

Bueno en este tenemos aquí uno llamado post.

Si te vas hasta abajo aquí tenemos este post y te fijas que es un array, es un JSON, es un array que

tiene entidades que tiene un userid, un ID, un title y un body.

Bueno, vamos a hacer un DTO que represente esta información.

Aquí vamos a crear una carpeta en nuestro proyecto llamada Datos.

Vamos a ponerle Details, agregar carpeta y vamos a ponerle Dto mayúscula y una S chiquita, Ahí está.

Y aquí vamos a crear una nueva clase llamada en este caso vamos al nuevo clase Agregar clase, vamos

a llamarla Post dto post dto.

Y aquí lo que vamos a hacer es representar la información que está en ese servicio Fake service.

Tenemos un user ID que es un entero, tenemos un ID que es un entero y tenemos dos string que se llama

title y body.

Bueno, vamos a poner esos campos, vamos a poner primero va a ser una propiedad que se llame ID y aquí

vamos a poner la otra propiedad que se llame User ID, que es entera.

Vamos a poner la entidad título title, que es un string y vamos a poner el último que es el body.

Aquí el punto, si te fijas se pone verde.

Esto que significa?

Que es verde?

Que este string podría ser null, es decir este string.

Bueno este string no permite nulos, pero el punto es que el csharp en este punto no sabe si tu title

va a permitir nulos o no va a permitir nulos.

Entonces si tú quieres que permita nulos y esto funciona también para tipos int double float agrega

un símbolo de interrogación aquí y de esta manera tu propiedad va a permitir que sea nula.

Así, de esa manera, agregando el símbolo aquí, va a permitir que sea nula.

Si tú no le agregas el símbolo, no permite que sea nulo.

Es por eso que se pone verde.

Verde es una alerta, es un warning, simplemente una advertencia.

Y bueno, ya tenemos nuestro DTO que representa lo que vamos a obtener de la información en forma de

array de un servicio externo.

37. HttpClient para solicitudes externas

Lo que vamos a hacer es obtener la información del servicio Jsonplaceholder, es decir, nos vamos a

conectar a esta URL externa y vamos a obtener esta información.

La vamos a retornar en un en una lista de elementos.

Para ello vamos a crear un servicio el cual haga esa conexión y el servicio lo vamos a llamar desde

un controlador.

El servicio lo vamos a inyectar.

Entonces primero vamos a crear aquí en servicios un servicio, dándole clic derecho vamos a agregar

primero su interface y vamos a darle aquí nuevo elemento, vamos a agregar una interfaz y vamos a ponerle

que sea del tipo y vamos a llamarle y post service.

Recuerda que la ventaja de tener interfaz a la par interfaz e implementación con clase es que tú podrías

cambiar la clase por otra clase que implementa la misma interface, o podrías tener varias clases que

implementen la misma interface y poder utilizarla en tu controlador con los.

Como vimos la inyección por medio de aquí.

Pero bueno, esa es la ventaja.

Por eso siempre creo la interface cuando creo un servicio bueno.

Aquí tenemos este servicio y vamos a poner aquí que solo contenga todo lo que implemente este servicio,

tenga un un método público que sea asíncrono con Task y que regrese un IEnumerable que sea del tipo

post dto el que hicimos en la clase pasada y aquí este le vamos a llamar GET así simplemente.

Bueno, aquí la ventaja de tener a lo mejor lo primero que te que te confunde es porque un innumerable

y por qué no, un list list es una clase, por lo cual List tiene muchísimas cosas más allá de solamente

ser un listado list es para manipulación de información, es decir, List no solamente es una lista

de elementos, sino que también tiene muchos métodos que al final de cuentas está en memoria.

La ventaja que tenemos al regresar Enumerable es que innumerable es te fijas?

Una list implementa innumerable, innumerable tiene solamente lo necesario para que nosotros necesitamos

un enumerable enumerable Es que se puede iterar, Es decir, es que es una colección de información.

De qué tipo?

Bueno, el tipo que le indicaste en este caso va a ser nuestro dto.

Post y numerable es más rápido y es más eficiente que regresar una lista.

La lista aparte innumerable, no solamente es de lectura.

Es por ello que aquí, en lugar de regresar una lista, voy a regresar innumerable.

Una lista se puede meter innumerable, ya que una lista implementa innumerable.

Por eso es más eficaz utilizar esto.

Bueno, ahora teniendo esto, vamos a crear ahora sí la clase que implemente esto.

Vamos a agregar a la clase, vamos a llamarla Post Service y vamos a hacer que implemente esta interface

de la siguiente manera agregando los dos puntos y post y post service.

Bueno, aquí vamos a crear una entidad privada que es un tipo HttpClient, este http HttpClient.

Lo que sirve es una clase que ya tiene punto net que nos sirve para conectarnos a solicitud por solicitudes

HTTP, get, http, post put, delete.

Es decir, seríamos un cliente, seríamos el front end, seríamos como un front end, seríamos como

alguien que necesita un recurso.

Obviamente aquí nosotros también nos podemos conectar como backend a otros backend, en este caso al

servicio de JSON placeholder, es decir, este servicio que si te fijas esto es un get cuando tú puedes

verlo en el navegador es porque es algo que es por GET.

Vamos a comenzar a hacer esto de manera muy burda manera No, no correcta.

Vamos a ir mejorándolo, vamos a crear un constructor y en el constructor vamos a inicializar el objeto

y tú me dirás Oye, pero tú me dijiste que los objetos no tienen que estar inicializados dentro de otra

clase.

Sí.

Ahorita vamos a ver cómo se maneja con punto net este tipo de entidades.

HttpClient pero ahorita vamos a hacer que por lo pronto vaya por la información y ahorita vamos a ir

mejorando las cosas para que vayas aprendiendo.

Por qué se hacen de la de una manera, cómo se optimiza el código a partir de las mejoras que tiene

tu framework Este.

La implementación de esta interfaz me obliga a tener.

Esto me obliga a tener un método que sea del tipo asíncrono.

Aquí falta poner la palabra async async solo vale la implementación async no va en la interfaz.

Aquí si pones async te va a mandar al diablo porque te va a decir que no se puede tener un easing en

algo que no tenga código.

Si te fijas ahí dice modificador.

Solo se puede usar en métodos que tengan un cuerpo, es decir, que tengan código.

Bueno, ahí pues no lo podemos tener.

Aquí sí, porque vamos a tener un cuerpo que es la implementación de la interface.

Lo que vamos a hacer primero es ir a llamar a la URL.

Cuál URL?

Bueno, aquí la tenemos, aquí tenemos la URL, no la voy a copiar, está aquí arriba JSON placeholder

punto com post me lo va a copiar igual en esta sección la voy a dejar en la descripción para que la

puedas copiar aquí.

Esta URL me la voy a guardar aquí y voy a guardarla toda completa.

Ahí está nuestra URL.

Ahora lo que vamos a hacer es una variable que obtenga el resultado de lo que vamos a ejecutar de forma

asíncrona con await.

HttpClient, de hecho, nos lo está rellenando.

Si te fijas, HttpClient tiene ya métodos asíncronos que regresan un task, eso ya lo vimos en las clases

pasadas.

Entonces de esta manera nosotros ya podemos hacer que esto sea va a ser asíncrono y podemos aquí asegurarnos

que termine la ejecución, porque ir a un servicio que es tercero, un servicio externo es tiempo.

Y ese tiempo pues bueno, tenemos aquí que cerciorarnos que termine para poder obtener su contenido.

Para obtener su contenido vamos a leer su body con el body.

Aquí estamos creando un body del tipo vamos a hacer también asíncronamente y este body lo vamos a leer

a partir del result content punto read as string async.

También es de forma asíncrona.

Ahora lo que sigue.

Vamos a hacer que se lean los post.

Vamos a crear una variable post que estaba a deserializar que vimos en la introducción.

Vimos cómo deserializar un JSON.

Si vemos el body es un JSON, por lo cual lo podemos meter en una lista de objetos.

Está utilizando var porque lo de la derecha me indica ya el tipo.

Aquí me está indicando que si te fijas aquí me dice que es un tipo httpresponsemessage, aquí me dice

que es un tipo string y por lo cual el string lo puedo deserializar.

Para serializar vamos a utilizar una clase llamada JSON serializador punto de Serialice y entre el entre

las entre los generics vamos a poner lo que necesitamos, un enumerable del tipo post dto y el body

que es un string lo va a leer Y ya listo, ya tenemos deserializado el post y aquí lo único que vamos

a hacer es un return post.

Bien, ahora lo que sigue es vamos a inyectar solamente voy a cambiar esto a plural.

Aquí le puse plural, voy a cambiar el plural para que se entienda que es un conjunto de post para cambiar.

Le voy a cambiar el nombre acá en el servicio desde el desde la interface voy a cambiarle el post con

una S y acá voy a decir que sí, que me que me cambie todos donde, donde aparece y aquí en la clase.

Aquí le vamos a cambiar por si no nos cambian el apartado este y post service.

Y aquí vamos a ponerle post y aquí en nuestra clase, en el caso de que te pase a ti, bueno, aquí

vamos a ponerle también una una S.

Ahí está listo para que se entienda que es algo que viene en post.

Igual tú puedes dejarlo como post en singular.

Aquí la interface lo voy a poner una S, ahí está listo.

De esta manera ya se se se ve de forma, se forma plural, ya está.

Y bueno, ahora voy a ir a inyectar esto, inyectarlo donde?

Pues en program vamos a Program y en Program tenemos un apartado aquí para inyectar.

Recordemos que aquí está para nuestras inyecciones que hicimos aquí.

Vamos a poner aquí un apartado, vamos a poner Builder Services, vamos a ponerla con tipo Scope, que

ya explicamos que es Scope, que solamente es única.

Por cada solicitud vamos a poner que va a ser el tipo y post service y post service.

Esa es la implementación.

Bueno, ya teniendo esto vamos a ya está inyectado, por lo cual lo podemos obtener en un controlador.

Vamos a ir a un controlador aquí a crearlo, agregar controlador, vamos a llamarlo post en API, ojo

aquí API API en blanco agregar.

Vamos a ponerle post con S al final.

Bueno, igual tú puedes dejarlo como tú gustes, yo lo voy a poner una vez al final y vamos a darle

que se cree Vamos a hacer una propiedad, una un campo que sea privado, el cual sea del tipo de nuestros

post service y de esta manera nosotros ya podemos obtenerlo en el constructor de manera inyectada.

El constructor se llama igual post controller y podemos obtenerlo inyectado aquí y podemos obtenerlo

y para hacerle la asignación.

Ahora vamos a hacer un método que sea el tipo Httpget y este método que sea el tipo Httpget va a ser

un público, va a ser asíncrono, async, va a ser un Task, va a retornar un innumerable enumerable

y va a tener del tipo post dto.

De esta manera te fijas ya el objetivo del DTO.

Que esta es un.

Es un objeto que está transfiriéndose entre capas.

Aquí vamos a poner get y esto solamente lo podemos hacer una línea de código.

Vamos a poner la fecha, vamos a poner aquí un enter y vamos a poner que sea asíncrono.

Con el await vamos a poner service y que vaya por GET y ahí está.

De esta manera ya tenemos nuestro servicio que va a ir a una ejecución, a una un recurso tercero y

va a obtener la información, la vamos a serializar y aquí simplemente lo ejecutamos con API POST,

lo ejecutamos y vamos a ver que nuestros objetos no se están llenando y eso tiene una explicación,

pero ya vemos que se está conectándose al servicio.

Bueno, el servicio está haciendo algo porque tarda un tiempo, que es lo que está pasando aquí.

Bueno, lo que pasa es que los objetos que se retornan tienen están en minúscula, fíjate user id,

title body.

Bueno, y nuestra clase?

Nuestra clase que está de dto.

Por convención de csharp escrito todo aquí vamos a ir a la clase DTO.

La que tenemos aquí nuestra.

Nuestra clase está en mayúscula.

La primera letra entonces no está haciendo el enlazamiento.

Lo que pasa es que las convenciones de Csharp nos indican que escribamos de esta manera.

Es una sugerencia.

Y yo te sugiero que también lo hagas.

Entonces aquí el punto es cómo lo resuelvo en el servicio?

Aquí no está haciéndose.

La serialización se está haciendo, pero no está encontrando su equivalente porque no coinciden con

la mayúscula en minúscula.

Lo que tenemos que hacer aquí es mandar una opción para que ignore mayúsculas y minúsculas, y para

eso vamos a agregar aquí.

Antes del post vamos a agregar var options.

Vamos a declarar es igual a new JSON, Serializer, options, json, Serializer, options y aquí podemos

agregar opciones entre ellas.

Una de ellas es una que se llama Property name Propertyname case Insensitive y esta la ponemos en true

y de esa manera nosotros la mandamos aquí después del body.

Después de este body vamos a darle enter para que se alcance.

A ver, falta un punto y coma aquí arriba.

Después del body vamos a darle options.

Ahí está y ya está.

De esta manera ya va a ignorar las mayúsculas minúsculas.

Y si aunque tengas una mayúscula en minúscula, con que corresponda la letra va a poder hacer el lanzamiento,

entonces me voy aquí a mi Postman, ejecuto y esto ya debería tener la información.

Te fijas?

Ahora sí tiene la información porque ha ignorado los casos y ya tenemos nuestro controlador que se ejecuta

un servicio tercero de esta manera, pero nuestro servicio no está correctamente implementado.

Todavía faltan hacer ciertas configuraciones para que esto quede de una manera más.

38. IHttpClientFactory para fabricar objetos HttpClient

Imagina una situación donde necesites esta misma solicitud en más controladores o en más servicios.

Y bueno, esta URL pues la tengas que estar escribiendo y escribiendo y tengas que estar creando tu

HttpClient.

De hecho, esto está mal porque estamos creando un objeto dentro del constructor cuando deberíamos obtenerlo

inyectado.

Bueno, en.

NET tenemos algo llamado y HttpClient Factory que si no conoce un patrón de diseño llamado Factory Factory,

es un patrón de diseño que sirve para crear objetos.

En este patrón de diseño simplemente se especifica que tengas una fábrica para crear objetos en esta.

Esta fábrica va a ser una creadora de objetos HttpClient con una configuración inicial, los cuales

van a poder estar inyectados.

Por lo cual por lo cual tú vas a poder accederlos en tus constructores de tanto de servicios como de

controladores, siempre y cuando este servicio esté inyectado.

Bueno, ahora que tenemos aquí en Program vamos a hacer eso.

Ojo Nota con esto porque he visto que hay muchas personas que tienen problema con esto.

Tú tienes que inyectar el servicio antes, antes de tu HttpClient, que va a ser inyectado por medio

de HttpClient Factory.

Bueno, cómo se inyecta?

Vamos a hacerlo debajo.

Si lo pones arriba vas a tener un problema porque no va a tener la configuración de tu de tu HttpClient.

Entonces aquí vamos a ponerlo debajo de tus servicios.

Vamos a poner Add HttpClient y aquí vamos a ponerle cuál es la interfaz que la va a utilizar.

Bueno, va a utilizar la E-post Service.

Y cuál es la implementación que la va a utilizar?

Post service.

Y de esta manera tú puedes estar poniendo una configuración de HttpClient inicial en los servicios que

tú desees.

Ya tenemos esta parte.

Ahora lo que vamos a poner es la configuración.

Esto recibe una función de primera clase, por lo cual aquí vamos a ponerle una C de.

Es el parámetro que nos va a mandar, que es en sí el cliente.

Vamos a ponerle aquí C punto base address y de esta manera nosotros podemos agregarle un newry newry

en la URL y este newry vamos a agregarle la URL que teníamos acá en post service.

Esta URL la vamos a quitar de aquí.

Quítese por allá, vamos a quitarle ahorita vamos a regresar aquí.

Quítese por ahí y aquí vamos a regresar a Program y vamos a agregarla aquí.

Bien, y de esta manera ya esta URL ya está puesta en la configuración que está insertada en el HttpClient

que vas a poder obtener en tu constructor en esta en esta implementación de post service, de esta de

esta interface.

Ahora vamos aquí a Post Service y vamos a quitar esto.

Quite este por allá y en el constructor vamos a poder obtener nuestro HttpClient ya con esa URL ya utilizándola

para que ya esté lista para que nosotros podamos utilizarla sin que conozcamos acá la URL.

Vamos a a manejar aquí simplemente hacemos la asignación y aquí en esta URL vamos a obtenerla a partir

de nuestro objeto HttpClient.

Punto.

Se llama base Address.

Ahí va a estar la URL, es decir, yo aquí no conozco cuál es la URL y no debería conocerla, pero si

la tenemos configurada en nuestra inyección en este apartado en program, aquí la tenemos y esa y la

deseamos utilizar ese HttpClient con esa URL en otros servicios.

Pues simplemente volvemos a repetir esto y lo ponemos la interface con el con la clase que implemente

donde hacemos esa HttpClient y la URL.

Ya no necesitamos conocerla, simplemente la utilizamos y ya está.

Podemos reutilizar el código.

Es la manera más correcta de hacer esto.

Si yo ejecuto esto, vamos al apartado de Postman.

Vamos a mandar la misma solicitud, la que ya teníamos, y esto debería funcionar la misma manera,

es decir, refactorizamos el código sin que cambiara su funcionamiento y está más organizado, porque

si en un futuro necesitáramos cambiar la URL, pues simplemente la cambiamos aquí y ya está.

Y ya no tenemos que ir a todos los servicios que utilizaran esta URL e ir a cambiarla una por una.

Entonces, de una manera más práctica.

39. Obtener información de appsettings.json