Sección 6: Programación asíncrona

31. ¿Qué es programación asíncrona?

En esta sección vamos a ver lo que es la programación asíncrona y esto para que lo entiendas te puedo

dar una analogía para que quede muy claro.

Imagina que eres un cocinero, el cual tiene que hacer tres platillos, tiene que hacer un espagueti,

un pavo y un postre.

Tú no puedes hacer el pavo hasta que termines el espagueti y no puede hacer el postre hasta que termines

el espagueti y el pavo.

Solamente puedes trabajar en una tarea a la vez.

En este escenario, tú.

Cuánto tardarías?

Imagina que el espagueti dura diez minutos.

El pavo dura otros 20 minutos y el postre dura cinco minutos.

Tardaría 35 minutos para terminar los tres platillos, ya que no puedes comenzar el uno platillo hasta

que termines el primero.

A pesar de que muchas veces el platillo por sí solo ya está, está en la estufa y está cociéndose o

está haciendo preparándose sin que tengas que intervenir mucho.

Esto es algo que se llama programación síncrona, un evento que no puede comenzar hasta que el otro

evento termine.

Este trae ciertos problemas cuando estás programando.

El problema radica que tú estés trabajando a veces en conexiones con con terceros, por ejemplo una

base de datos con un servicio.

Tercero con la lectura de un archivo, la escritura de un archivo.

Y estos procesos tienen un tiempo y tú no puedes continuar a hacer el siguiente proceso hasta que termine

este.

Bueno, C-sharp es un lenguaje de programación que tiene gran potencial con la programación asíncrona.

La programación asíncrona es que puedas hacer un sistema, un programa que pueda estar respondiendo

a otros eventos mientras estás haciendo ciertas tareas en segundo plano.

Esto es la programación asíncrona.

Imagínate, tú eres el cocinero, puedes estar preparando el espagueti y comenzar el pavo mientras el

espagueti está haciéndose en la estufa sin que tengas que intervenir mucho.

Tú ya puedes comenzar con el pavo sin ni siquiera que termine la tarea del espagueti.

Y puedes también comenzar con el postre.

Entonces ya estás adelantando tiempo que tú a lo mejor estás detenido y puedes aprovechar ese tiempo

detenido porque está haciéndose algo fuera de tu alcance y puedes estar haciendo los tres procesos a

la vez y al final solamente estar, estar verificando si ya terminaron los procesos para decir y entregar,

decir ya terminamos el espagueti.

Ok, entreguemos, ya terminamos el pavo, entreguemos, ya terminamos el postre, entreguemos y bueno,

la programación asíncrona es eso, tú puedes estar haciendo una tarea, pero estar aprovechando el tiempo

mientras esa tarea está fuera de tu alcance y se está haciendo un cálculo fuerte, se está haciendo

una conexión a una base de datos.

Tú puedes estar aprovechando ese tiempo para hacer algo a continuación y de esta manera, cuando tú,

en lugar de estar haciendo programación síncrona, la cual está esperando una tarea a la otra, a la

otra, a la otra, puedes estar trabajando de manera paralela varias tareas y también si estás haciendo

varias tareas a la vez, verificar cuando ya terminaron todas, regresar una respuesta.

Por eso la programación asíncrona es muy poderosa.

Teniendo esta línea de tiempo, una raya para abajo que va a indicar el tiempo que va pasando.

Imagina lo siguiente.

Vamos a conectarnos a una base de datos, pero ese proceso de ir a una base de datos tiene cierto tiempo.

La base de datos está mejor a la mejor en otro servidor ese tiempo, pues tienes que ir a conectarte,

a que te responda y hay un tiempo por ahí y en ese tiempo tú puedes estar haciendo otras tareas, por

ejemplo mandando un correo que también tiene otro proceso que va a tardar tiempo con programación asíncrona.

Tú podrías estar mandando la señal de que se conecte a la base de datos y tenga que hacer todo lo que

tengas que hacer con la base de datos, insertar, agarrar información, lo que sea, llamándolo P11

proceso uno.

Y mientras antes de que termine tú ya puedes estar comenzando a mandar el correo y eso lo llamamos proceso

dos, pero llegando a un punto donde tú tienes que esperar a que terminen las tareas y e indicar la

respuesta a quien te está solicitando seas un controlador en este caso, pues mandarle la respuesta

a quien te solicito una vez terminen los dos procesos.

Entonces aquí estamos en espera del proceso uno y ya cuando hasta cuando termine el proceso uno, aquí

ya podemos pasar a la espera del proceso dos.

Pero esos procesos no necesariamente es que no hayan comenzado, comenzaron desde este punto.

Esos procesos comenzaron a ejecutarse, por lo cual se estaban haciendo de forma paralela y al final

solamente estábamos esperando.

Estamos verificando si esos procesos ya terminaron en estos dos puntos y cuando ya terminaron, pues

nosotros ya podemos responder a quien nos haya solicitado.

De esta manera es como funciona la programación asíncrona.

Y ahora vamos a ver unos ejemplos de por qué es importante conocer esto, porque cuando estás trabajando

con backend vas a trabajar con muchísimos procesos que son asíncronos por necesidad, porque estás conectándote

a base de datos, porque estás haciendo escritura con un servicio de nube, porque estás escribiendo

archivos, porque estás leyendo archivos, etcétera Y estos procesos tienen un tiempo que tú puedes

ir aprovechando haciendo otras tareas.

Así que esta sección vamos a ver programación asíncrona.

32. Creación de método síncrono

Para entender las ventajas de la programación asíncrona vamos a ver qué es lo que involucra un método

síncrono.

Lo contrario, vamos a crear un controlador aquí que se llame vamos a darle clic, vamos a darle Agregar

controlador, vamos a llamarle Controller aquí.

Recuerda que tienes que darle en API y API en blanco.

Vamos a darle Agregar, vamos a ponerle nombre Sum Controller y una vez así vamos a agregarlo aquí.

Bien, lo primero que vamos a hacer va a ser un método que vamos a acceder directamente con la API diagonal,

el nombre del controlador SOM y vamos a agregarle un método GET que se le agregue diagonal sync de forma

síncrona.

Bueno, aquí vamos a poner lo siguiente.

Vamos a hacer que este método regrese.

Nada, un Actionresult, una interface y actionresult y el método.

Vamos a mandar a llamar getString y aquí al final voy a poner simplemente un return ok?

Y ahí está bien, vamos a hacer lo siguiente.

Imagina que tenemos que hacer una tarea que dura un segundo.

Para simular este segundo vamos a utilizar una clase llamada Thread, que lo que hace Thread es tiene

un método estático llamado sleep que simula hace un detenido del sistema por un segundo.

Bien, esto va a simular solamente la conexión de una base de datos.

Aquí vamos a imprimir un mensaje de conexión a base de datos terminada con un Console.WriteLine.

Ahí está.

Y lo que sigue.

Vamos a hacer otra simulación de otra cosa.

Un envío a un correo.

Esto es como regularmente se trabaja la programación síncrona, no puedes comenzar lo siguiente hasta

que termine lo primero.

Y si no termina lo primero no puedes continuar.

Por lo cual aquí va a pasar un segundo hasta que comience todo esto.

Estas siguientes líneas no vas a poder pasar hasta que pase ese segundo.

Bueno, para poder medir eso vamos a utilizar una clase llamada Stopwatch que lo que hace es Stopwatch,

es ser un cronómetro.

Stopwatch es una clase que se encuentra en System Diagnostic y Stopwatch.

Vamos a iniciar inicializarlo como un tal cual.

Como un cronómetro tiene un método que se llama Start y eso hace que se comience a contar el tiempo

y vamos a ponerle aquí debajo vamos a poner stop Stopwatch stop y eso hace que termine el tiempo ya

cuando lo detienes.

Y aquí vamos a poner una línea de console.readline que diga todo ha terminado para ver que todos los

procesos hayan terminado.

Esto solamente es una simulación para que entiendas qué es lo que involucra hacer que sea síncrono,

síncrono, Es decir, no puedes hacer algo dos veces, no puedes hacer dos tareas a la vez, no puedes

hacer una tarea hasta que termine la primera.

Eso es de forma asíncrona.

Haces esto, luego haces esto otro, por lo cual hasta que termine esas líneas, pues como van, la

secuencia de líneas se va a ejecutar.

Y bueno, aquí vamos a poner nosotros podemos obtener el tiempo que transcurre con Stopwatch, podemos

obtener el tiempo que transcurre con la propiedad a laps y esta lapse del tiempo transcurrido nos va

a indicar cuánto tiempo se ejecutó en sí.

Bueno, vamos a correr esto, vamos a ejecutar nuestro método llamado Sum y dentro tenemos Sync y este

sync lo pusimos en el http get para que pudiéramos diferenciarlo de otro método que vamos a hacer más

adelante.

Vamos a darle Enviar y vamos a ver aquí que nos va a dar el tiempo que ha transcurrido.

Vamos a ver.

Y ahí está.

Nos ha dado que ha transcurrido dos segundos con algunas milésimas de segundos.

Y acá vemos en la consola que se han ejecutado asíncronamente las cosas.

Es decir, terminó primero el proceso, lo terminó este, luego todo ha terminado.

De hecho, si yo lo vuelvo a ejecutar, podemos ver ese proceso.

Vamos a verlo.

Ahí está, Va a empezar a imprimirse aquí.

Parece que transcurrió muy rápido.

Vamos a hacerlo de nuevo.

Lo voy a poner aquí a la derecha para que se pueda ver.

En el momento que le doy a esta conexión a base de datos, envió el mail terminado.

Es decir, no sé, comienza la siguiente tarea hasta que termine la primera.

Te fijas?

Aquí vamos a ver que hay conexión a base de datos.

Lo envío de mail.

Ha terminado, Todo ha terminado y dura dos segundos.

Bueno, cuando estás trabajando de manera asíncrona podemos ir ahorrando tiempo mientras una tarea se

está haciendo, Es decir, mientras la tarea que dura este segundo que está yendo a una base de datos

que la mejor tiene, que está haciendo un procesamiento en un servidor externo y que nosotros no dependemos,

no tenemos el control de eso.

Podemos ir aprovechando para ir mandando el correo electrónico, el cual va a ir a ejecutar un servicio

también y pues podemos ir aprovechando el tiempo para hacer otras tareas mientras la otra está haciéndose

ya en un plano que no está a nuestro alcance.

Simplemente nosotros vamos a necesitar esperar, verificar si ha terminado y una vez que he terminado

los dos procesos le regresamos la respuesta.

Eso es como vamos a trabajar en la siguiente clase con las tareas asíncronas.

Pero bueno, antes tenemos que ver un concepto llamado Task.

33. Task

En C Sharp tenemos una entidad llamada Task.

Task representa una operación asíncrona, es decir, nos va a resolver la vida con la sincronía C Sharp

es potente con este tipo de tareas.

Para comenzar, vamos a poner un nuevo método que sea.

Vamos a entrar por él por http get y vamos a ponerle async y vamos a ver qué es lo que involucra trabajar

con Task.

Para trabajar con Task, nosotros podemos hacer que un método del controlador regrese por sí mismo.

Una Un método trabaje con por sí mismo, con asincronía y para eso vamos a ponerle que en lugar de que

retorne un Actionresult así regrese un Task, el cual va a ser del tipo y actionresult.

Y bien, tenemos que especificar async cuando vamos a trabajar con Task y este.

Esto lo que significa es que vas a trabajar con una operación asíncrona que se va a ejecutar en un en

un subproceso y ese es su proceso.

Involucra que estés trabajando con 1,1 tiempo que esté separado del proceso principal, por lo cual

puedes aprovechar para hacer otro subproceso con otros Task.

Bueno, el punto es que necesitas async y Task y aquí puedes ponerle que regrese lo que tú quieras.

En este caso, como yo, no va a retornar más que una respuesta de http pongo y actionresult, pero

tú podrás hacer que retorne un int que retorne un objeto.

Bueno, ya dependiendo la situación aquí vamos a llamarle get a este método async y vamos a poner que

esto trabaje con una tarea.

Para trabajar con tareas basta con que hagamos un objeto del tipo Task.

Vamos a poner aquí esto solamente simulación.

Ya más adelante vamos a ver que muchas cosas de punto net ya trabajan con Task y ya están implementadas.

Bueno, aquí vamos a crear una tarea con el con la clase Task y vamos a especificar que esta tarea tenga

algo adentro.

Te fijas?

Task recibe una función de primera clase, recibe una función primera clase que puedo representar con

una expresión lambda.

Ahí está.

Y de esta manera yo puedo darle un funcionamiento aquí interno y ese funcionamiento interno.

Vamos a ponerle la línea de arriba que tenemos en el método asíncrono.

Aquí ya está listo.

Esto es como trabaja Task, pero tenemos que iniciar el Task, es decir, aquí tenemos Task, pero tengo

que iniciarlo.

Entonces aquí vamos a poner Task uno punto Start y de esa manera el Task inicia.

Pero eso no involucra, eso no indica que ha terminado, Es decir, aquí inicia y esto se está haciendo

en un subproceso.

Esta parte se está haciendo en un subproceso, la cual tú puedes estar continuando aquí haciendo cosas.

Por ejemplo, yo puedo poner aquí un console.readline que indique que esa es otra cosa.

Igual esto no necesariamente ha terminado y tú puedes estar haciendo otra cosa.

Y cuando tú deseas que esto ya termine o estés comprobando que termine, necesitas el resultado.

Pones simplemente await y la task.

De esta manera tú ya te cercioras que ok, ya no pasa al siguiente punto hasta que termines.

Es decir, esto se está ejecutando en un segundo plano.

Esto se ejecuta en el proceso principal y esto va a verificar que hasta que termine va a continuar a

la siguiente línea, la cual vamos a poner un console.readline que indique que todo ha terminado.

Vamos a poner aquí todo ha terminado y bueno, de esta manera tú puedes tener mayor control con la sincronía.

Y aquí simplemente vamos a hacer un return Ok?

Por qué es importante entender este await?

Porque si tú no pones el await, esta tarea podría estar todavía ejecutándose y tú ya llegaste el return

Ok?

Es decir, si quitamos esta wait, esta tarea podría estar ahí haciéndose todavía.

Vamos a ejecutarlo, vamos a ejecutar y ver qué es lo que pasa cuando quito el await.

Y recordemos que aquí tenemos el método async.

Vamos a ponerle la aquí y vamos a ejecutar Send y vemos en la consola que hace primero la línea, hago

otra cosa.

Todo ha terminado, lo cual no es cierto y conexión a base de datos terminada.

Esto está dentro del subproceso.

Si te fijas este ya sería un flujo.

Estamos aprovechando hacer otra cosa, pero también es necesario enterarnos cuando eso ya ha terminado

y decir ok, espérate, todo ha terminado, todavía no ha terminado.

Esto todavía está ejecutándose porque lo ha impreso después, por lo cual es importante la wait ponerlo

cuando necesitemos que termine para poder responderle al cliente.

Y es por eso que es importante poner el await aquí, este task y así de esta manera ya ok, no termines,

no pases esta línea hasta que el proceso haya terminado.

Entonces ahora sí vamos a ejecutarlo, vamos a ejecutar esto, vamos a darle send y vamos a ver que

en este punto ahora el proceso va a tener un comportamiento y ahora si realista hago otra cosa o que

el proceso ya se inició, hago otra cosa mientras y ahora sí, conexión a base de datos terminada y

ahora sí, todo ha terminado.

Eso es lo que hace la wait.

Entonces el objetivo del Task es eso, poder aprovechar el tiempo mientras está haciéndose algo en otro

plano, en segundo plano o a lo mejor está haciendo.

Imagina cuando te conectas a una base de datos.

La base de datos está tiene su propio entorno y tú pues no sabes cuánto va a durar, pero puedes ir

aprovechando ese tiempo para hacer una cosa.

Pero antes de terminar, tienes que cerciorarte que ya haya terminado el proceso que hiciste externo

o el proceso de entrada salida.

El proceso de.

Y hasta que esto haya terminado.

Ahora, si indicamos la respuesta, todo ha terminado.

Los Task también pueden retornar un valor.

Por ejemplo, aquí yo puedo ponerle un Generic aquí y puedo decir que esto va a retornar un entero y

es donde comienza a tener más sentido esto, porque imagina que esa conexión a la base de datos te regresa

un número, en este caso un ocho y ese ocho lo necesitas para responderlo aquí.

Entonces aquí vamos a poner nosotros en esta wait, vamos a poner el resultado, vamos a poner el resultado

de ese lo que nos va a obtener y este resultado puedo imprimir aquí.

Entonces aquí también la importancia de tener una wait, porque no puedes estar respondiéndole el OK

hasta que haya terminado el proceso secundario.

Es decir, a lo mejor necesitas la lista de usuarios, necesitas ir a la base de datos, pero puedes

ir haciendo otras cosas que mientras, aprovechando el tiempo, pero al final necesitas el resultado,

por lo cual ya aprovechaste el tiempo, hiciste otras cosas y aquí el resultado ya lo obtienes al final

para ahora sí, responderle al servicio.

Vamos a ejecutar esto y ahora sí, ya que está corriendo esto, vamos a darle send y vamos a ver que

ahora sí tenemos el resultado de un proceso que se hizo asíncrono.

Tenemos ese ocho, ese ocho lo obtuvimos el proceso que está en el Task y bueno, vemos en la consola

que tiene algo de coherencia y ahora sí, las líneas que están ejecutando en C Sharp tenemos la ventaja

de tener esa entidad Task que representa una tarea asíncrona y vamos a estar haciendo uso bastante de ella en este curso.

34. Síncrono vs Asíncrono