DESARROLLO DE SOFTWARE

Fundamentos de la Programación

AGENDA

Conceptos fundamentales de la programación

Paradigmas de programación

Elementos básicos de un programa

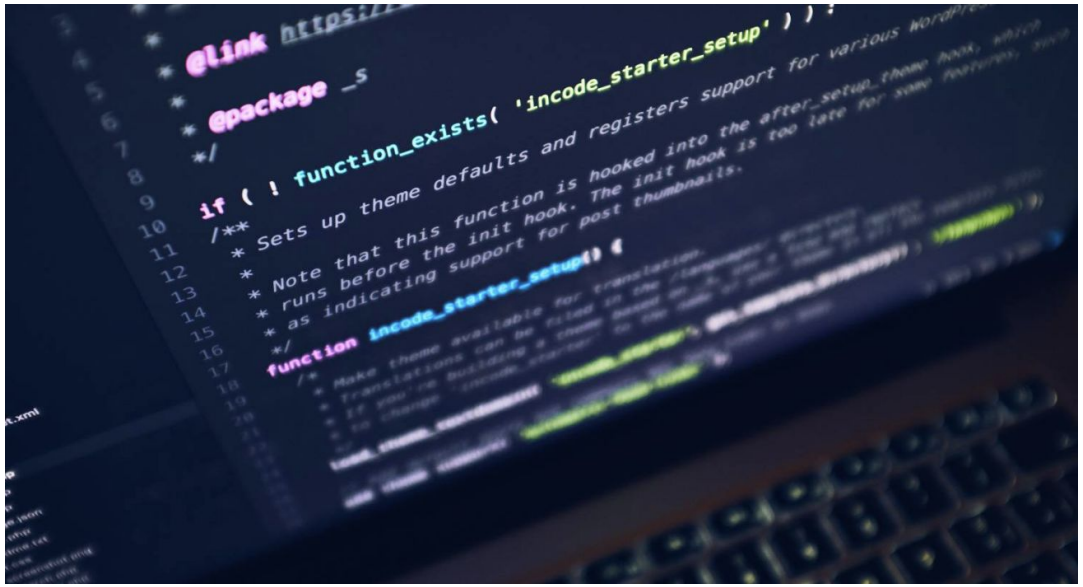
Lenguajes de programación

Pensamiento algorítmico

Resolución de problemas

Herramientas de programación

Buenas prácticas de programación



CONCEPTO FUNDAMENTALES DE LA PROGRAMACIÓN

Es el proceso de crear instrucciones para que una computadora las ejecute.

Permite crear software que resuelva problemas, automatice tareas y controle sistemas.

Es una herramienta fundamental en la era digital.

PARADIGMAS DE PROGRAMACIÓN

Existen diferentes maneras de organizar el código y estructurar los programas.

Los paradigmas de programación más comunes son:

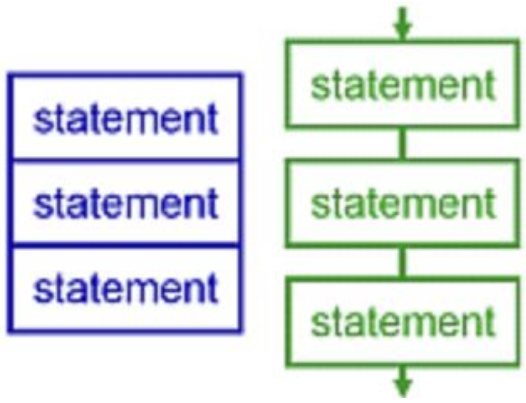
Programación estructurada: se basa en la descomposición de problemas en subproblemas más pequeños y manejables.

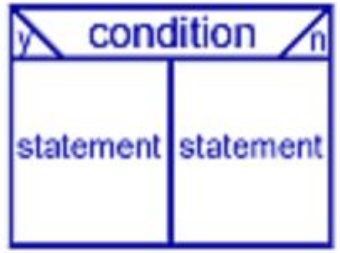
Programación orientada a objetos (OOP): se basa en la creación de objetos que encapsulan datos y comportamiento.

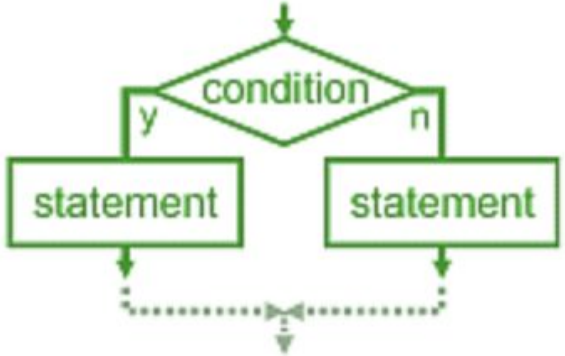
Programación funcional: se basa en la definición de funciones como bloques de código reutilizables.

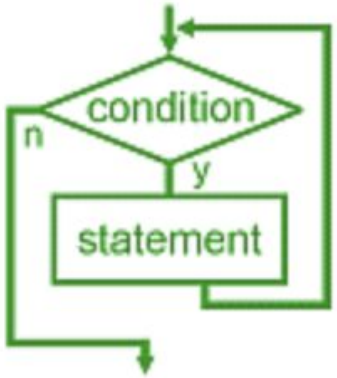
PROGRAMACIÓN ESTRUCTURADA

- SECUENCIA
- SELECCIÓN
- ITERACIÓN
- EVITAR EL GOTO
- SUBRUTINAS (FUNCIONES)









PROGRAMACIÓN ORIENTADA A OBJETOS (OOP)

Encapsulamiento

Agrupa datos (atributos) y el comportamiento asociado a ellos (métodos) dentro de una unidad llamada objeto.
Oculta la implementación interna de los objetos, exponiendo solo una interfaz pública (métodos) para interactuar con ellos.
Promueve la modularidad y la reutilización del código.

Abstracción

Representa las características esenciales de un objeto, ignorando detalles irrelevantes.
Permite crear modelos conceptuales más simples y fáciles de entender.
Facilita la creación de jerarquías de clases y la herencia.

PROGRAMACIÓN ORIENTADA A OBJETOS (OOP)

POO es un paradigma de programación que se basa en la creación de objetos que encapsulan datos y comportamiento. Las principales características de la POO son encapsulamiento, abstracción, herencia, polimorfismo, clases y objetos, y relaciones entre objetos. La POO ofrece numerosas ventajas, como modularidad, reutilización del código, mantenimiento, escalabilidad y flexibilidad, lo que la convierte en un paradigma **popular** para el desarrollo de software de gran tamaño y complejidad.

- librerías
- frameworks

PROGRAMACIÓN ORIENTADA A OBJETOS (OOP)

Herencia

Permite a una clase heredar atributos y métodos de otra clase padre.
Promueve la reutilización del código y la especialización.
Facilita la creación de jerarquías de clases y la organización del código.

Polimorfismo

Permite que objetos de diferentes clases respondan al mismo mensaje de manera diferente.
Se implementa a través de:
Sobrecarga de métodos: permite definir métodos con el mismo nombre pero diferentes parámetros o tipos de retorno.
Sobrescritura de métodos: permite redefinir un método heredado de una clase padre en una clase hija.
Proporciona flexibilidad y potencia a la programación orientada a objetos.

PROGRAMACIÓN ORIENTADA A OBJETOS (OOP)

Clases y objetos:

Las clases son los planos o plantillas a partir de los cuales se crean los objetos.
Los objetos son instancias individuales de una clase.
Las clases definen las características y el comportamiento de los objetos.
Los objetos encapsulan datos y métodos específicos.

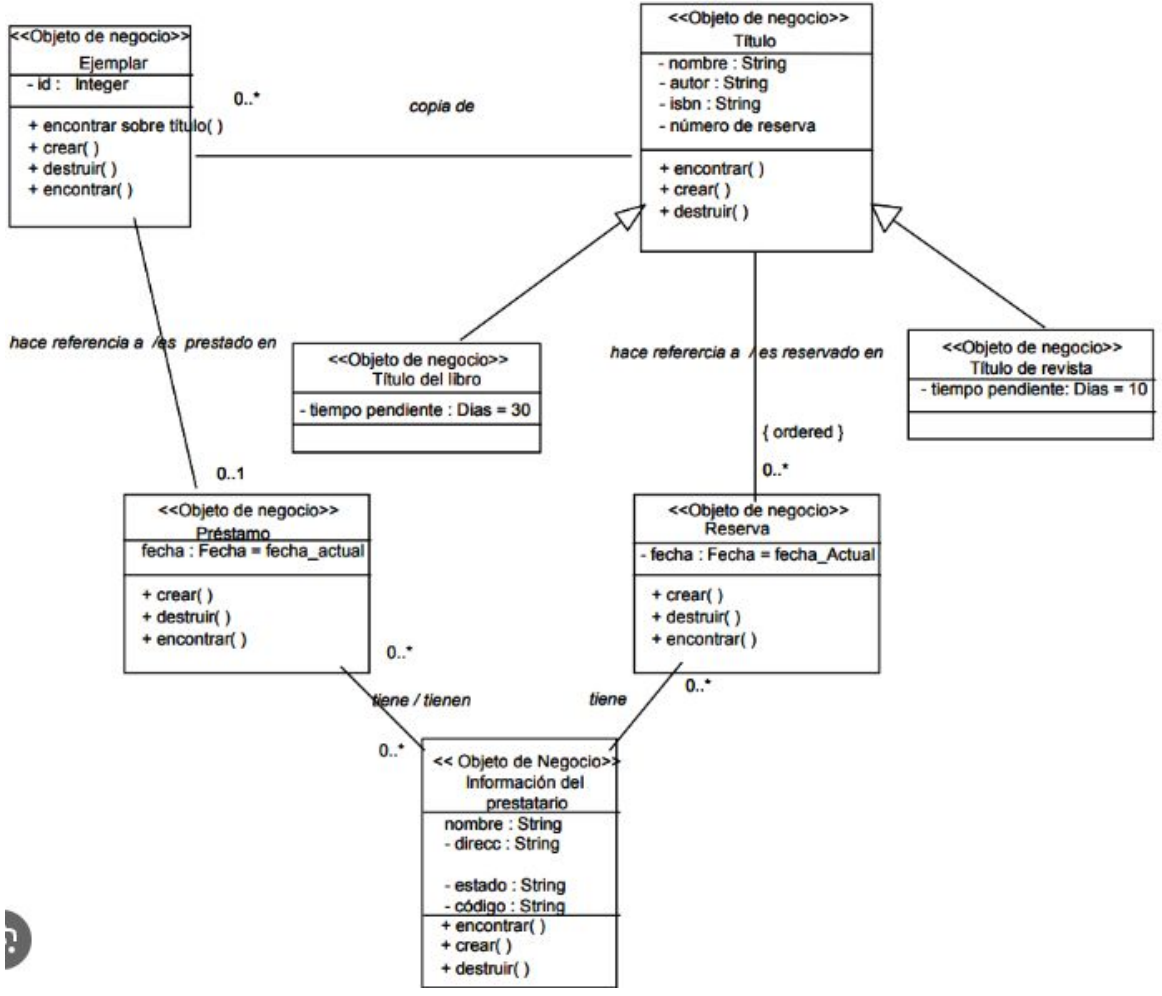
Relaciones entre objetos:

Los objetos pueden relacionarse entre sí de diferentes maneras:
Asociación: un objeto utiliza o contiene otro objeto.
Agregación: un objeto es parte de otro objeto, pero mantiene su propia identidad.
Composición: un objeto es parte de otro objeto y no puede existir de forma independiente.

Smalltalk
Ruby

UML - DIAGRAMA DE CLASES

- Estos diagramas muestran las clases que componen un sistema y las relaciones entre ellas. Las clases se representan como rectángulos y las relaciones se representan como líneas entre los rectángulos.



PROGRAMACIÓN FUNCIONAL

Dividir la mayor cantidad posible de tareas en funciones, de esta forma estas tareas pueden ser usadas por otras funciones con diferentes objetivos.

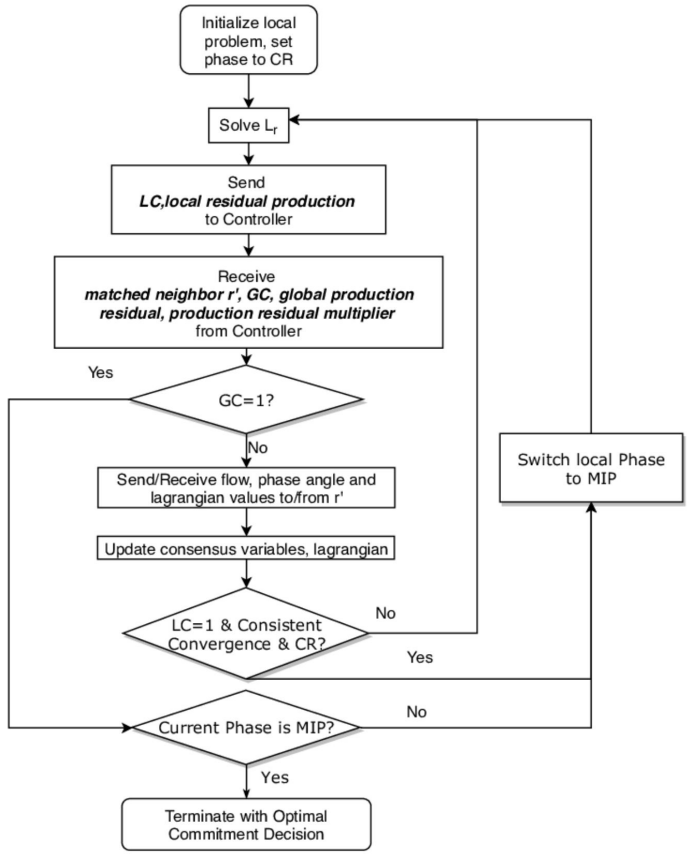
El objetivo es conseguir lenguajes expresivos y matemáticamente elegantes, en los que no sea necesario bajar al nivel de la máquina para describir el proceso llevado a cabo por el programa, y evitar el concepto de estado del cómputo. La secuencia de computaciones llevadas a cabo por el programa se rige única y exclusivamente por la reescritura de definiciones más amplias a otras cada vez más concretas y definidas.

Este paradigma, **por no contener datos mutables**, se caracteriza por ser usado para el manejo de información y no para la creación o modificación de la misma.

```haskell
--Función recursiva que calcula el factorial de un número natural
factorial :: Integer -> Integer
factorial n
   | n < 0  = error "no existe el factorial para enteros negativos"
   | n == 0 = 1
   | otherwise = n * factorial (n-1)
```

PENSAMIENTO ALGORITMICO

Un algoritmo es una secuencia de pasos ordenados para resolver un problema.
El pensamiento algorítmico es la capacidad de pensar en términos de algoritmos.
Es una habilidad esencial para cualquier programador.



MODELO PRODUCTOR CONSUMIDOR

1 - Qué es el patrón productor consumidor?

El patrón productor consumidor (`producers/consumers`) en un patrón de diseño que designa los procesos para ser productores de recursos, en nuestro caso, mensajes. O consumidores de los mismos.

Cuando hablamos de software utilizamos productor consumidor en **comunicación asíncrona**, y normalmente será a través de un `bus` o `colas` para almacenar esos recursos **de forma temporal**.

Utilizamos el patrón productor consumidor por ejemplo cuando **nos suscribimos a una web**, newsletter, etc, donde dicho servicio nos manda un email.

Por ejemplo, indicas el email y así como le das a “suscribirte” el servidor te responde “Gracias por la subscripción” pero, a los dos minutos te llega el email indicando que estás suscrito.

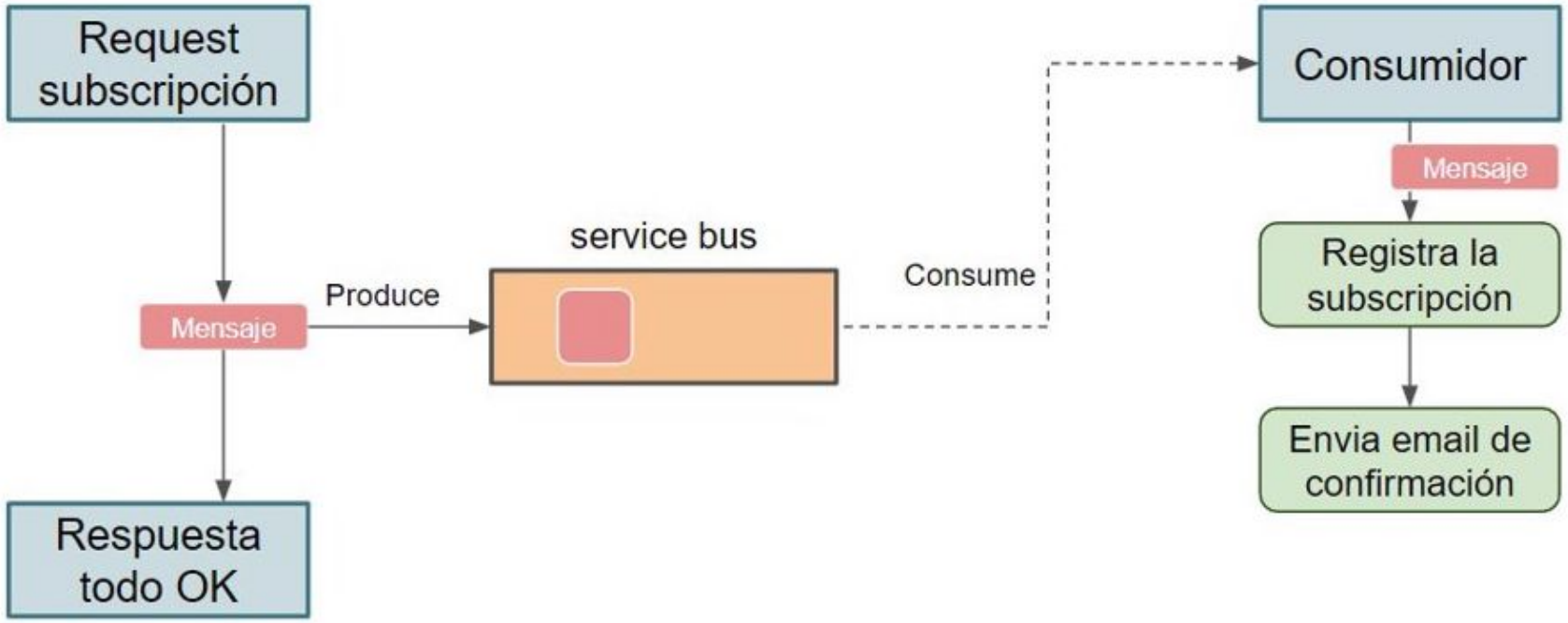
Esto es porque se utiliza el patrón productor consumidor.

A la vista del usuario, al hacer click se ha registrado su email y ha llegado el mensaje a su email. Pero el proceso no es del todo así.

Al hacer click, el software ha creado un mensaje con el email y lo ha publicado en nuestro message broker, e instantáneamente le ha dado la respuesta al usuario diciendo que todo está bien.

MODELO PRODUCTOR CONSUMIDOR

Pero aún queda procesar el mensaje, para ello hay otro sistema que va a consumir el mensaje y enviar el email con la confirmación.



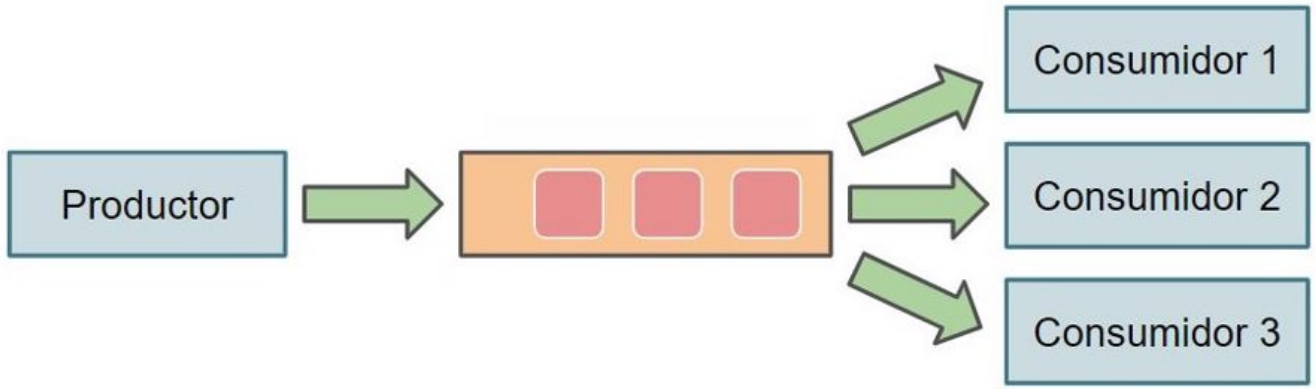
Este es solo uno de los muchos ejemplos que puedes encontrar en la web. Cuando compras productos en muchas páginas web (como por ejemplo Amazon) funciona de la misma manera, te da un Id de la compra y al rato te confirma que está todo bien, pero si por lo que fuera algo falla, también te lo comunica.

MODELO PRODUCTOR CONSUMIDOR

1.1 - Message Bus

Para implementar la lógica productor consumidor normalmente utilizaremos un `message bus`, también llamado *service bus*.

Esto es debido a que podemos tener **múltiples consumidores para el mismo bus** y ninguna de las aplicaciones es consciente de cuál es la aplicación que produce los mensajes o que otras aplicaciones están escuchando dicho bus.

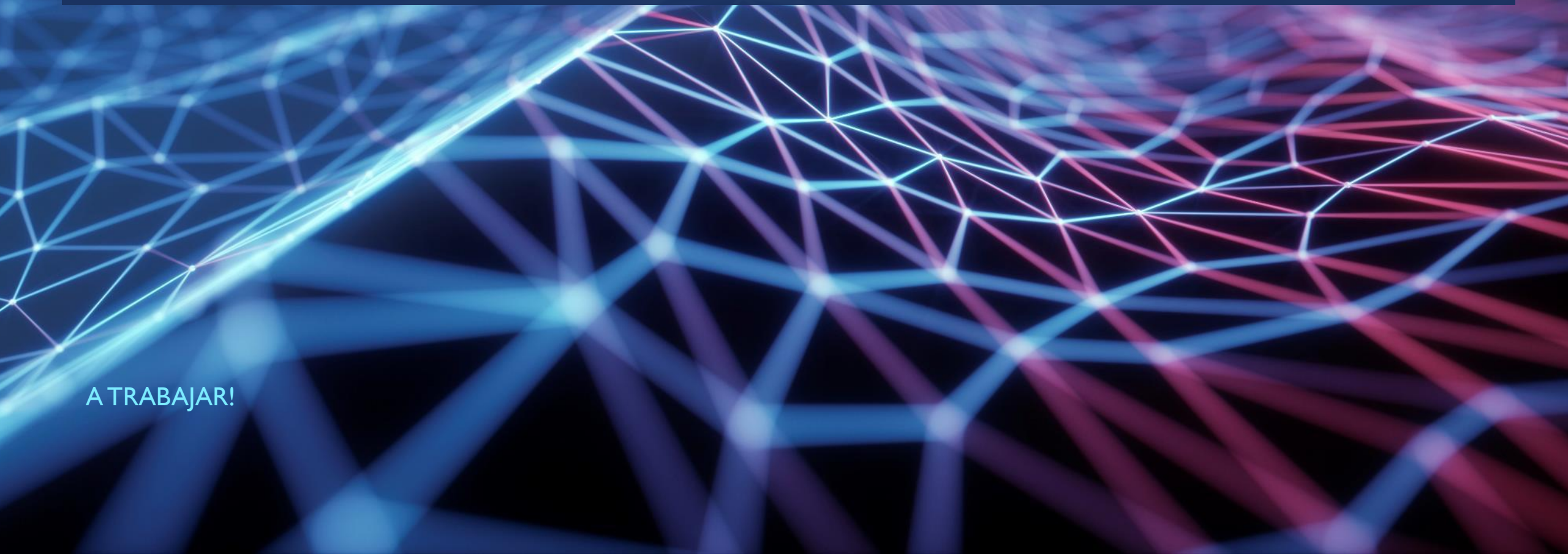


Un punto importante a tener en cuenta es que **no hay garantía de que los mensajes sean consumidos en orden**, esto quiere decir que si dos mensajes son generados puede ser que tengamos varios consumidores y no los reciban en el mismo orden.

Dependiendo de qué software utilizas para el service bus puede darse el caso de que si no hay ningún consumidor el mensaje se pierda (RabbitMQ) mientras que otros softwares almacenan dichos mensajes (Kafka).

Nota: en RabbitMQ puedes almacenar los mensajes si haces el binding a una cola directamente dentro de RabbitMQ.

PROYECTOS - PROGRAMAR



A TRABAJAR!