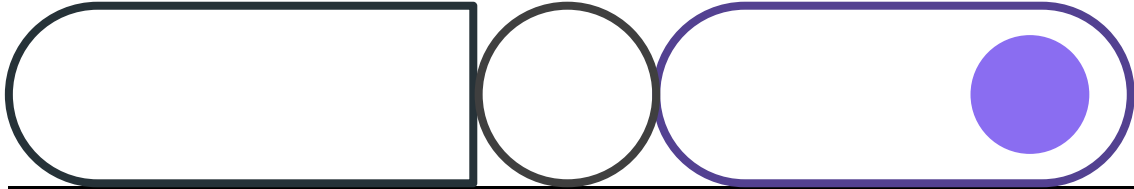
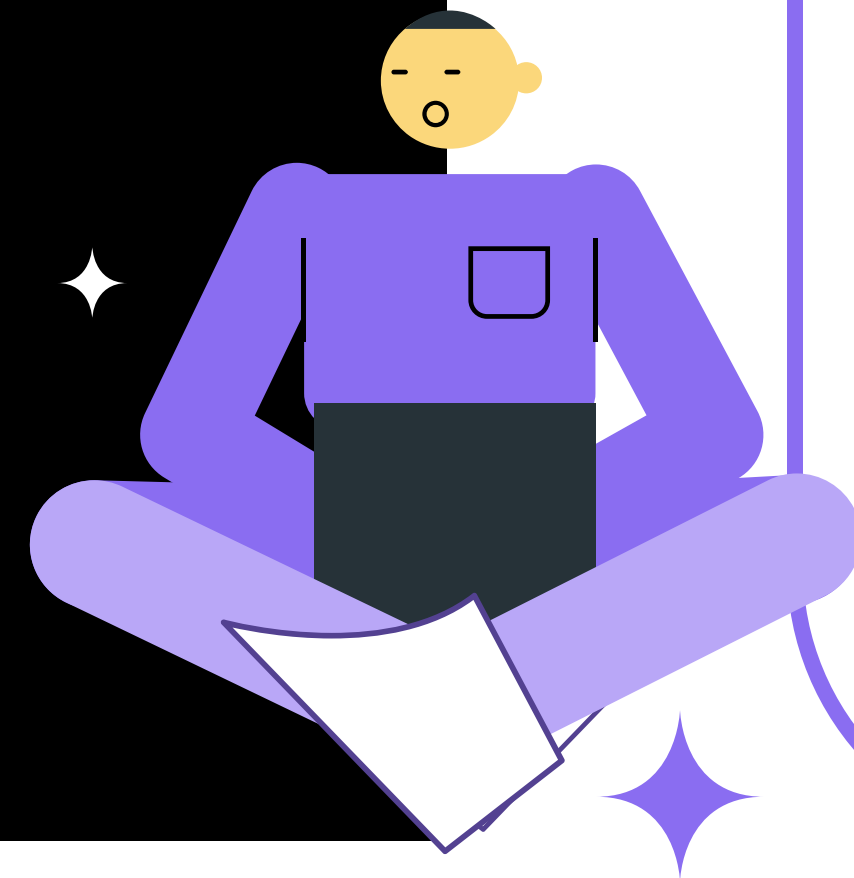


Web context



Las aplicaciones en Go usan **contextos** para controlar y administrar aspectos muy importantes de aplicaciones, como la cancelación y el intercambio de datos en la programación concurrente.



Índice

- 01** [¿Qué es web context?](#)
- 02** [Contexto con valor](#)
- 03** [Contexto con tiempo de espera](#)
- 04** [Cancelación de contexto en Go](#)



01

¿Qué es web context?

¿Qué es web context?

Es un paquete de información que se envía entre las distintas capas de nuestra aplicación. Se crea generalmente cuando llega un nuevo **request** a la API. Este paquete se transmitirá a la capa de servicio y a la capa de almacenamiento.

El paquete comienza con algunas piezas importantes de funcionalidad. Veamos:

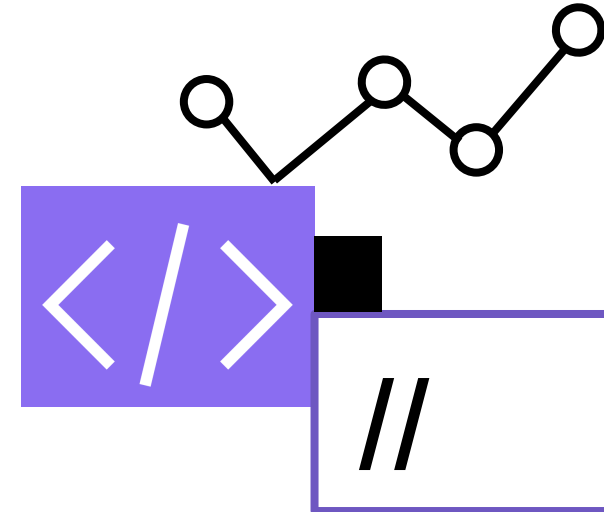
- **La capacidad de almacenar información adicional** que se puede transmitir a lo largo de la propagación.
- **La capacidad de controlar la cancelación:** puede crear paquetes que detienen la ejecución de su código si superan un plazo específico o un valor de tiempo de espera.

02

Contexto con valor

Uno de los usos más comunes del context es compartir datos o utilizar valores de ámbito de solicitud. Cuando tenemos varias funciones y deseamos compartir datos entre ellas, podemos hacerlo usando contextos. La forma más fácil de hacerlo es usar la función **context.WithValue**. Esto crea un nuevo contexto basado en un contexto principal y agrega un valor a una clave dada.





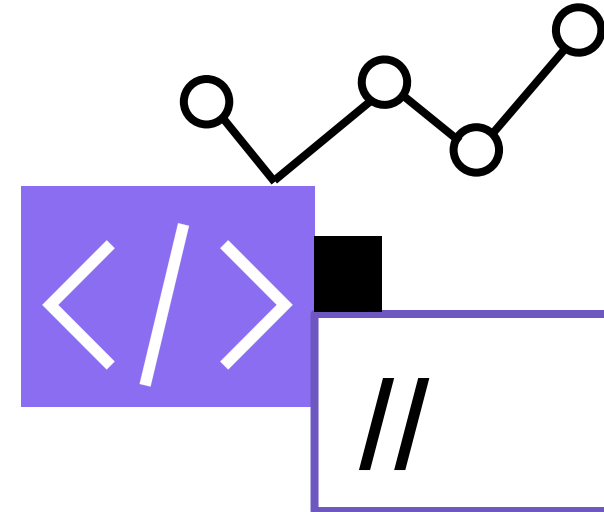
Agregando valores a nuestro contexto

Podemos pensar el **context** como si fuera un mapa, por lo que podemos agregar y recuperar valores por clave. Esto es muy poderoso, ya que nos permite almacenar cualquier tipo de datos dentro del contexto.

```
ctx := context.Background()  
context.WithValue(ctx, "api-key", "super-secret-api-key")
```

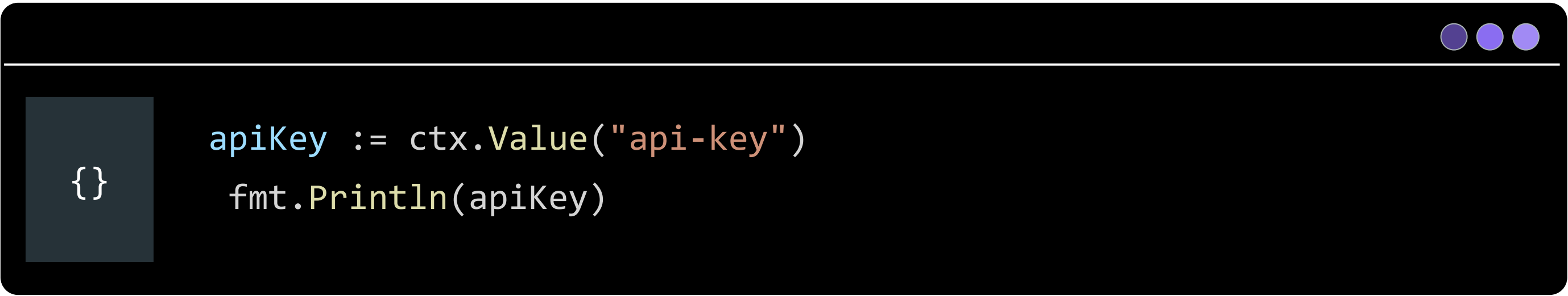


Es importante tener en cuenta que la función **WithValue()** devuelve una copia del contexto existente y no modifica el contexto original.



Agregando valores a nuestro contexto

Con la función **Value("key")** podemos leer el valor de la llave que solicitamos. Hay que tener en cuenta que cuando intentemos acceder a un par clave/valor del objeto **ctx** que no existe, nos devolverá **"nil"**.



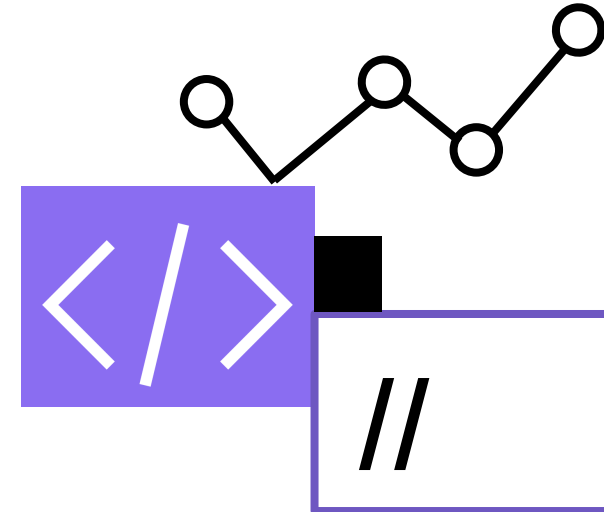
```
apiKey := ctx.Value("api-key")
fmt.Println(apiKey)
```

03

Contexto con tiempo de espera

Los **tiempos de espera** son un patrón muy común para realizar solicitudes externas, como consultar una base de datos u obtener datos de otro servicio a través de HTTP.





Crear un contexto con tiempo de espera

La función **WithTimeout(parent Context, timeout time.Duration) (Context, CancelFunc)** retorna un context y una función de cancelación en caso de que queramos activarla manualmente. Funciona de la misma manera que una cancelación de contexto normal. El comportamiento de esta función es sencillo: en caso de que se alcance el tiempo de espera, el contexto se cancela.

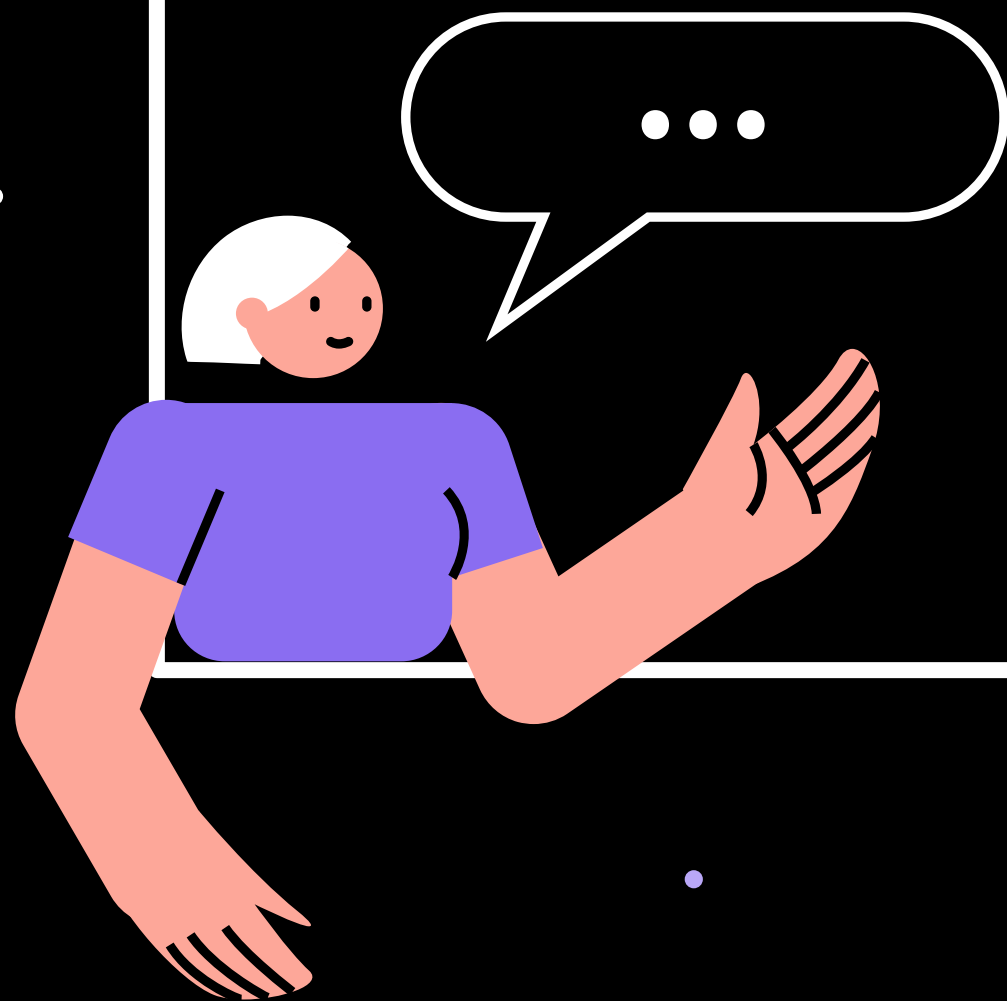
```
{ } ctx, cancel := context.WithTimeout(context.Background(),  
2*time.Second)
```

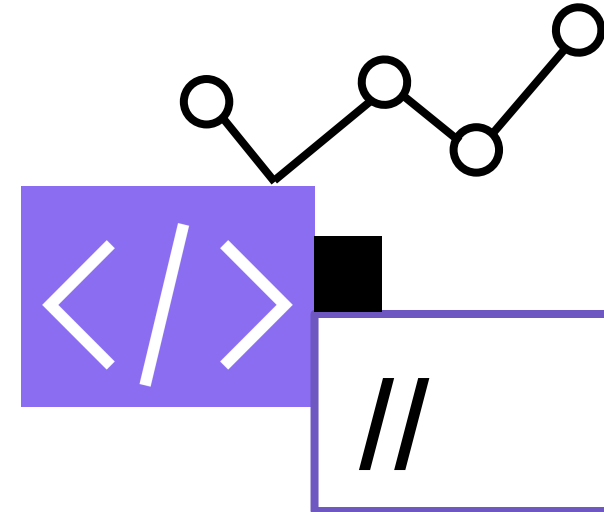
04

Cancelación de contexto en Go

Otra característica muy útil del contexto en Go es cancelar cosas que están relacionadas. Es una buena práctica propagar la señal de cancelación cuando se recibe.

Digamos que tenemos una función en la que se inician decenas de rutinas. Esa función principal espera a que terminen todas las rutinas o una señal de cancelación antes de continuar.





Cancelación del context

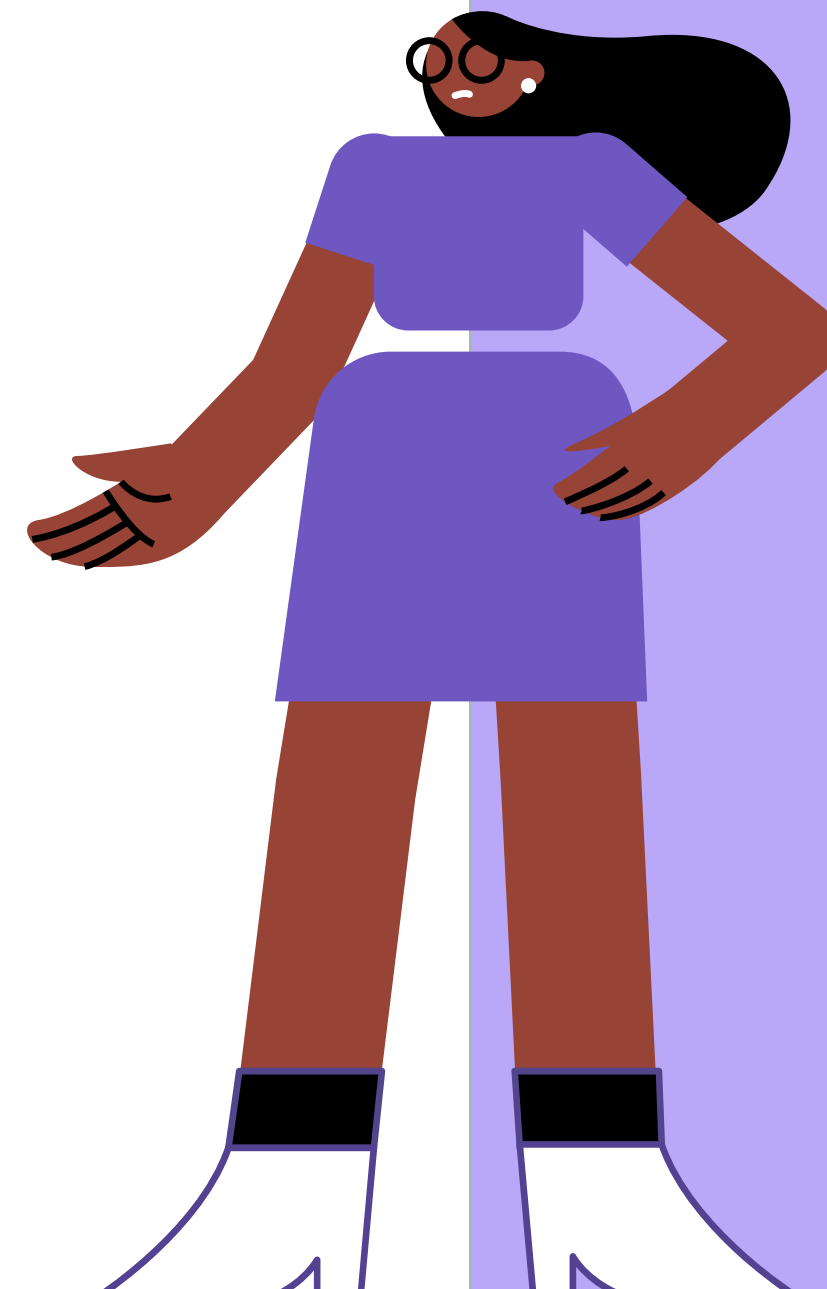
Para crear un contexto con cancelación solo tenemos que llamar a la función **context.WithCancel(ctx)** pasando su contexto como parámetro. Esto devolverá un nuevo contexto y una función de cancelación. Para cancelar ese contexto, solo necesitamos llamar a la función de cancelación.

```
{ }
```

```
ctx, cancel := context.WithCancel(context.Background())  
defer cancel()
```

Conclusiones

En esta clase, aprendimos qué son los contextos y cómo podemos usarlos para la propagación de valor a través de nuestra aplicación. También vimos cómo podemos usar contextos para establecer tiempos de espera dentro de nuestro código y asegurarnos de que **no estamos desperdiciando recursos informáticos**. Si bien podemos usar contextos para pasar información entre las capas de la aplicación, es absolutamente necesario que los usemos **solo para las cosas que realmente necesitan propagarse**.



¡Muchas gracias!