



GO

BOOTCAMP

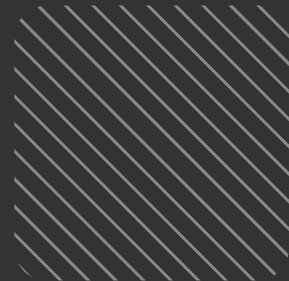


Clase en vivo

//Go Web

IT BOARDING

BOOTCAMP



Objetivos de la clase:

- Comprender qué son los middlewares.
- Conocer las herramientas que nos ofrece Gin para implementarlos.
- Entenderemos la importancia de la documentación en nuestros proyectos
- Implementaremos el uso de Swagger para la creación de la documentación.

Índice



01 [Repaso](#)

02 [Middlewares en GO](#)

03 [Documentación en GO](#)

IT BOARDING

BOOTCAMP



1

Repaso

IT BOARDING

BOOTCAMP

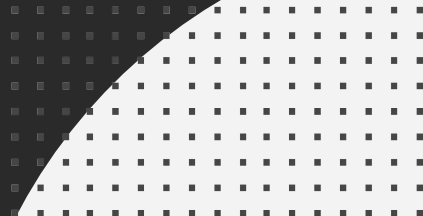


Middleware

IT BOARDING

BOOTCAMP





El middleware es una entidad que intercepta el ciclo de vida de una solicitud o respuesta del servidor. En palabras simples, **es un fragmento de código que se ejecuta antes o después de que el servidor atienda una solicitud con una respuesta.**

IT BOARDING

BOOTCAMP

// ¿Qué son?

Para explicar qué son los middlewares, vamos a comenzar con un ejemplo, después juntos hallaremos la razón.

Tenemos tres funciones que tienen la misma firma: **sumar**, **restar** y **multiplicar**. Ahora supongamos que por alguna razón, a esas tres funciones haya que agregarles comportamientos. Necesitamos mostrar la hora cada vez que una de esas funciones es invocada.

Además, necesitamos registrar en un archivo el resultado que generó dicha función (a modo de ejemplo, solo mostraremos un mensaje).

¿Qué deberíamos hacer?

La primera opción que se nos ocurre es... modificar todas las funciones:

```
func sumar(a, b int) int {  
    //Mostrar la fecha y la hora  
    fmt.Println(time.Now())  
    resultado := a + b  
    // Hacer algo después  
    return resultado  
}
```


Middleware

Si seguimos esta lógica, deberíamos escribir en cada función las mismas líneas de código.

¿Qué pasaría si en vez de tener tres funciones, tenemos cientos de funciones?

Imagínense cambiar la misma lógica en cientos de funciones una y otra vez...

¡enloqueceremos!

▲ ¿Podemos hacer algo más inteligente?

▲ ¿Qué tal si separamos dicho comportamiento, lo abstraemos a otra función?

▲ ¿Y si intentamos hacer que la función principal (sumar, restar y multiplicar) no se entere de comportamientos ajenos a ellas?

▲ **¡Qué interesante sería crear comportamientos y funcionalidad para poder aplicarlos a las funciones que deseemos!**

Un middleware es...

- Una técnica que se utiliza para resolver este tipo de cuestiones.
- Una función.
- Una función que recibe una función, y retorna otra función (con la misma firma) pero en la nueva función creada al vuelo y le agrega funcionalidad.



De esta manera, la función destino (*sumar, restar y multiplicar*), jamás se enterará de los comportamientos que les estamos aplicando; ya que estas son envueltas y ejecutadas por otra función (el Middleware).

Middleware

- Cada función hace solo una cosa, es responsable de realizar lo que se tiene que hacer. Además, el comportamiento de cada uno de ellos puede ser utilizado para todas aquellas funciones que posean la misma firma (que sean del mismo tipo).

Por ejemplo:

Si deseamos agregarle a cada una de las funciones (*sumar*, *restar* y *multiplicar*) el comportamiento de mostrar la hora cada vez que una de ellas sea invocada, podemos hacerlo creando el siguiente middleware:

```
func mostrarTiempo(f func(int, int)
int) func(int, int) int {

    return func(a, b int) int {
        fmt.Println(time.Now())
        return f(a, b)
    }
}
```

¿Cómo lo invocamos?

```
func main() {  
    fmt.Println("Resultado: ", mostrarTiempo(sumar)(10, 5))  
}
```

Si lo hacemos paso a paso, sería así:

```
f := mostrarTiempo(sumar)  
resultado := f(10, 20)  
fmt.Println(resultado)
```

Documentación

IT BOARDING

BOOTCAMP



// ¿Para qué sirve documentar una API?

Documentar correctamente una API permite a las personas desarrolladoras (internas o externas a nuestro equipo) un acercamiento accesible al proceso de aprendizaje de nuestra API / servicio.

Herramientas de documentación API

Desarrollar una API requiere un duro esfuerzo, tenemos que dedicarnos al diseño y la implementación, a la documentación y a su mantenimiento con el objetivo de que las personas desarrolladoras que la utilicen tengan claro su funcionamiento.

Por este motivo existen herramientas de documentación API que nos facilitarán el trabajo. Las **más conocidas son RAML (lenguaje de modelado API RESTful), Slate y Swagger**. Esta última, es uno de los estándares más utilizados y fue elegida por **Open API Initiative** como modelo de especificación API.



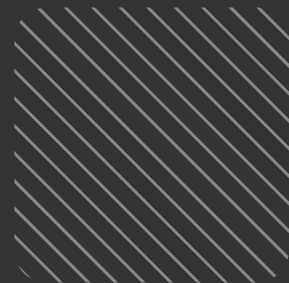


2

Middleware en GO

IT BOARDING

BOOTCAMP



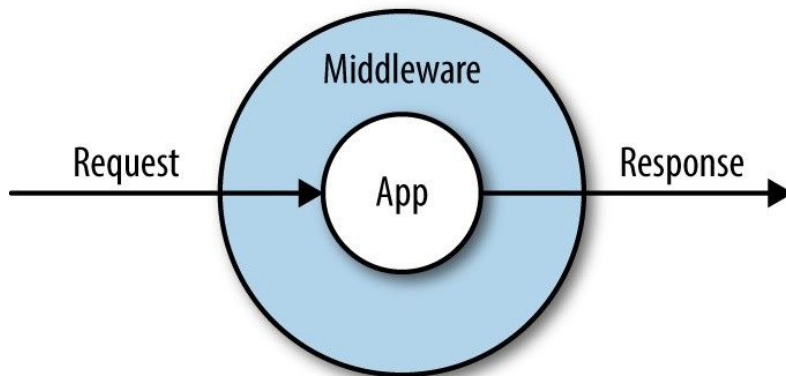
En términos de arquitectura del software, un middleware es un componente en nuestras aplicaciones que nos va a permitir **centrarnos en la lógica de negocio**, resolviendo algunas complejidades de bajo nivel.

IT BOARDING

BOOTCAMP

// Middlewares en un contexto HTTP

Las funciones *middleware* en un contexto HTTP pueden tener propósitos muy amplios y diferentes.



Middlewares en GO

```
func GetDummyEndpoint(c *gin.Context) {  
    resp := map[string]string{"hello": "world"}  
    c.JSON(200, resp)  
}  
  
{}  
func main() {  
    api := gin.Default()  
    api.GET("/dummy", GetDummyEndpoint)  
    api.Run(":8080")  
}
```



Middlewares en GO

{}

```
func DummyMiddleware(c *gin.Context) {  
    fmt.Println("Im a dummy!")  
  
    // Pass on to the next-in-chain  
    c.Next()  
}  
  
func main() {  
    // Insert this middleware definition before any routes  
    api.Use(DummyMiddleware)  
    // ... more code  
}
```



Middlewares en GO

{}

```
func DummyMiddleware() gin.HandlerFunc {  
    // Do some initialization logic here  
    // Foo()  
    return func(c *gin.Context) {  
        c.Next()  
    }  
}  
  
func main() {  
    // ...  
    api.Use(DummyMiddleware())  
    // ...  
}
```



Middleware de autenticación de API

Si están creando una **API con Gin**, probablemente quieran agregar algún tipo de mecanismo de autenticación. La solución más sencilla es comprobar si el cliente nos envía un parámetro de URL, como `api_token`, para luego validarlo y dejarlo realizar el request.



Middleware de autenticación de API

{}

```
func TokenAuthMiddleware() gin.HandlerFunc {  
    requiredToken := os.Getenv("API_TOKEN")  
  
    return func(c *gin.Context) {  
        token := c.GetHeader("api_token")  
  
        if token != requiredToken {  
            c.AbortWithStatusJSON(401, gin.H{"error": "Invalid API token"})  
            return  
        }  
  
        c.Next()  
    }  
}
```



A codear...





3

Documentación en GO

IT BOARDING

BOOTCAMP



// ¿Qué es Swagger?

Swagger es una herramienta extremadamente útil para describir, producir, consumir y visualizar APIs RESTful.



Swagger en Go

Para implementar Swagger en Go se utilizará **swaggo**, que nos permite generar la documentación del proyecto utilizando comentarios dentro del código.

```
$ go get -u github.com/swaggo/swag/cmd/swag
go get -u github.com/swaggo/files
go get -u github.com/swaggo/gin-swagger

export PATH=$PATH:$HOME/go/bin
```



Anotaciones en el main del programa

Arriba de la función **main** se agregan las anotaciones para especificar y documentar a nivel global nuestra aplicación.

{ }

```
// @title MELI Bootcamp API
// @version 1.0
// @description This API Handle MELI Products.
// @termsOfService https://developers.mercadolibre.com.ar/es_ar/terminos-y-condiciones

// @contact.name API Support
// @contact.url https://developers.mercadolibre.com.ar/support

// @license.name Apache 2.0
// @license.url http://www.apache.org/licenses/LICENSE-2.0.html
func main() {
```



Anotaciones en el controlador GetAll

Arriba del método **GetAll** del controlador, se agregan las anotaciones para documentar el endpoint.

{ }

```
// ListProducts godoc
// @Summary List products
// @Tags Products
// @Description get products
// @Accept json
// @Produce json
// @Param token header string true "token"
// @Success 200 {object} web.Response
// @Router /products [get]
func (c *Product) GetAll() gin.HandlerFunc {
```



Anotaciones en el controlador Store

Arriba del método **Store** del controlador, se agregan las anotaciones para documentar el endpoint.

{}

```
// StoreProducts godoc
// @Summary Store products
// @Tags Products
// @Description store products
// @Accept json
// @Produce json
// @Param token header string true "token"
// @Param product body request true "Product to store"
// @Success 200 {object} web.Response
// @Router /products [post]
func (c *Product) Store() gin.HandlerFunc {
```

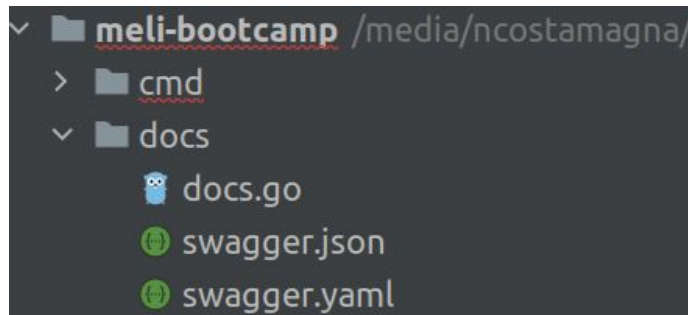


Generar documentación

Para generar la documentación se utilizará el comando que nos proporciona **swaggo**, especificando dónde se encuentra el archivo **main.go**

```
$ swag init -g cmd/server/main.go
```

Una vez ejecutado el comando, swaggo nos generará un paquete **docs** el cual contendrá toda la documentación del proyecto en base a las anotaciones generadas.



// ¿Como se podrá visualizar la documentación?

Para visualizar la documentación, **swaggo** nos proporciona el paquete **gin-swagger** que nos ayudará a visualizar la documentación desde gin.

Importación de paquetes

Se importan los paquetes **files** y **gin-swagger** que permitirán visualizar la documentación desde un endpoint.

Se debe importar el paquete **docs** que generó swaggo.

```
{}
```

```
"github.com/ncostamagna/meli-bootcamp/docs"  
"github.com/swaggo/files"  
ginSwagger "github.com/swaggo/gin-swagger"
```

Dentro de los **env** se debe agregar la variable **HOST** con la URL base de la aplicación.

```
.env
```

```
HOST=localhost:8080
```



Main del programa

En el *main* del programa se debe agregar el endpoint correspondiente a la documentación generada.

Debemos asignarle el HOST de la variable de entorno a la documentación

{ }

```
func main() {
    _ = godotenv.Load()
    db := store.New(store.FileType, "./products.json")
    repo := products.NewRepository(db)
    service := products.NewService(repo)
    p := handler.NewProduct(service)
    r := gin.Default()

    docs.SwaggerInfo.Host = os.Getenv("HOST")
    r.GET("/docs/*any", ginSwagger.WrapHandler(swaggerFiles.Handler))

    pr := r.Group("/products")
    pr.POST("/", p.Store())
    pr.GET("/", p.GetAll())
    pr.PUT("/:id", p.Update())
    pr.PATCH("/:id", p.UpdateName())
    pr.DELETE("/:id", p.Delete())
    r.Run()
}
```



Documentación

Al ingresar a <http://localhost:8080/docs/index.html> se podrá visualizar la documentación.

MELI Bootcamp API ^{1.0}

[Base URL: localhost:8080]
doc.json

This API Handle MELI Products.
[Terms of service](#)
[API Support - Website](#)
[Apache 2.0](#)

Products

GET

/products

List products

POST

/products

Store products

PUT

/products/{id}

Update products

DELETE

/products/{id}

Delete products

PATCH

/products/{id}

Update products name





Conclusiones

En esta clase aprendimos cómo documentar nuestros sistemas para facilitar la integración de externos.

Además aprendimos a implementar middlewares para poner capas intermedias en nuestras aplicaciones, como un sistema de validación.

En MeLi, no vas a realizar la documentación directamente con Swagger mediante la utilización de swaggo, pero es bueno que tengas presente que siempre es bueno documentar nuestras API, y ésta es una forma.



Actividad





Gracias.

IT BOARDING

BOOTCAMP

