



INTRODUCCIÓN A LA CLASE

GO WEB

Objetivos de esta clase

- Comprender por qué documentar una API.
- Conocer las buenas prácticas para el momento de documentar un API.
- Conocer y aplicar Swagger en GO.
- Comprender qué son los middlewares.
- Aplicar distintos middlewares en GO.





DOCUMENTACIÓN DE NUESTRA API

GO WEB

// ¿Para qué sirve documentar una API?

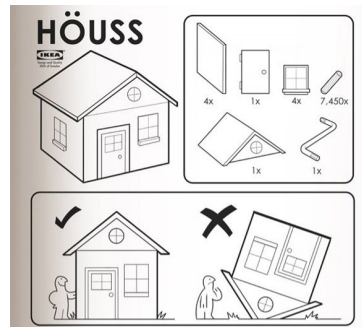
“Documentar correctamente una API permite que a los desarrolladores (internos o externos a tu equipo) les sea accesible el proceso de aprendizaje de tu API / servicio.”

IT BOARDING

BOOTCAMP

¿Por qué es importante documentar una API?

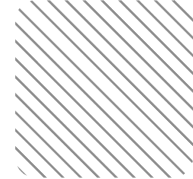
Es una de las piezas básicas que determinan la experiencia de un desarrollador al hacer uso de tus servicios API



Herramientas de documentación API

Existen diversas herramientas, aunque algunas de las **más conocidas sean RAML (lenguaje de modelado API RESTful), Slate y Swagger**. Éste último es uno de los estándares más utilizados y fue elegido por **Open API Initiative** como modelo de especificación API.





Buenas prácticas para una buena documentación API

Descripción endpoint detallada

Amplía la descripción de los endpoints con texto explicativo de la forma más detallada posible, utilizando un lenguaje comprensible, intentando no enfocarse en lo técnico, no todos tienen la misma experiencia o están familiarizados en el entorno de tu API.

Aunque las API actúan como una capa de abstracción que sirve para ocultar ciertos detalles de las operaciones que se ejecutan, lo mejor para no confundir a los desarrolladores es precisamente evitar abstracciones en la documentación.

Detalla los parámetros y las respuestas esperadas

Lista todos los parámetros de entrada y salida que proporcione cada endpoint, además de los tipos de respuesta HTTP esperados; ésto evitará sorpresas a los desarrolladores que consuman tu API teniendo en cuenta las diferentes casuísticas que deberán valorar en el momento de desarrollar un frontend apropiado.





Swagger

// ¿Qué es? ¿Cómo utilizarlo 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 {
```

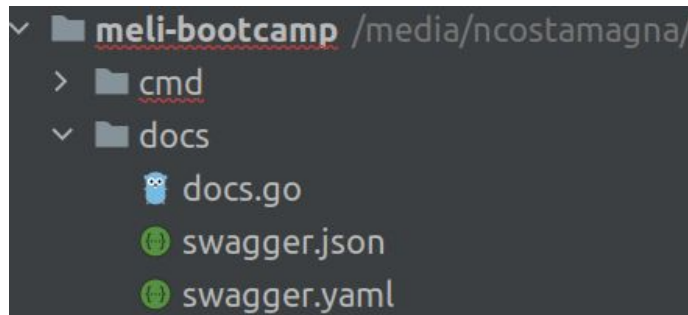


Generar documentación

Para generar la documentación se utilizará el comando que nos proporciona **swaggo**, especificando donde 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, **swagger** 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



// Para concluir

Hemos visto cómo utilizar swagger para documentar los proyectos, puedes ver la documentación oficial de [swaggo](#) para funcionalidades más complejas

¡A seguir aprendiendo!

IT BOARDING

BOOTCAMP



MIDDLEWARES

GO WEB

// ¿Qué son los middlewares?

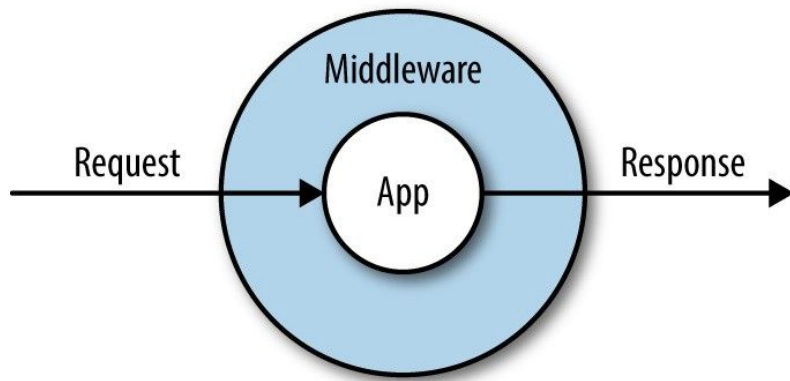
En términos de arquitectura del software, un middleware es un componente en nuestras aplicaciones que nos va a permitir centrarnos en la lógica de negocio, resolviendo algunas complejidades de bajo nivel.

IT BOARDING

BOOTCAMP

Middleware en un contexto HTTP

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



Middlewares en GO

{}

```
package main

import(
    "github.com/gin-gonic/gin"
)

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



Middleware de autenticación de API

{ }

```
func TokenAuthMiddleware () gin.HandlerFunc {
    requiredToken := os.Getenv("TOKEN")

    if requiredToken == "" {
        log.Fatal("no se encontró el token en variable de entorno" )
    }

    return func(c *gin.Context) {
        token := c.GetHeader("token")

        if token == "" {
            c.AbortWithStatusJSON (401, web.NewResponse (401, nil, "falta token en cabecera" ))
            return
        }

        if token != requiredToken {
            c.AbortWithStatusJSON (401, web.NewResponse (401, nil, "token incorrecto" ))
            return
        }

        c.Next ()
    }
}
```



Request ID middleware

```
package main

import (
    "github.com/gin-gonic/gin"
    "github.com/google/uuid"
)

func RequestIdMiddleware() gin.HandlerFunc {
    return func(c *gin.Context) {
        c.Writer.Header().Set("X-Request-Id", uuid.New().String())
        c.Next()
    }
}
```

Después de realizar una solicitud a su servicio, verá un nuevo header en la respuesta, similar a este: **X-Request-Id: 865d5aab-a5ba-492d-a6fb-d06d94720527**




Middleware no son solo de HTTP y REST



Tal y como se expuso desde el principio, el concepto de middleware puede ser tan genérico o agnóstico como nosotros queramos. Es por eso que dentro de otras comunicaciones también se pueden encontrar con middlewares.





Los invitamos a completar la
siguiente encuesta sobre el
módulo GO WEB del Bootcamp.

**¡Es muy muy importante para
nosotros contar con su
feedback!**

Solamente les tomará unos
minutos completarla :)

[Link a la encuesta](#)





Gracias.

IT BOARDING

BOOTCAMP





Autor: Benjamin Berger

Email: benjamin@digitalhouse.com

Última fecha de actualización: 12-05-21

IT BOARDING

BOOTCAMP

