INTRODUCCIÓN A LA CLASE

GO WEB

Objetivos de esta clase

- Entender cómo implementar las validaciones en el request.
- Entender, generar y aplicar manejo genérico de respuestas.

¡Aprendamos más de esto!

VALIDACIÓN DEL REQUEST

GO WEB

// ¿Cómo validamos el request?

Validar que los datos que envío el usuario sean correctos, como por ejemplo, que un dato sea requerido o que un valor sea numérico.

Validación del request

Nos encargaremos de agregar validaciones en el controlador para determinar valores requeridos.

En caso de no pasar las validaciones, el Controlador no enviará la tarea al Servicio sino que la rechazara directamente, informando al cliente.

Las validaciones del request siempre debe hacerlas en Controlador

Validación del request

{}

```
if req.Name == "" {
    ctx.JSON(400, gin.H{ "error": "El nombre del producto es requerido"})
    return
}

if req.Type == "" {
    ctx.JSON(400, gin.H{ "error": "El tipo del producto es requerido"})
    return
}

if req.Count == 0 {
    ctx.JSON(400, gin.H{ "error": "La cantidad es requerida"})
    return
}

if req.Price == 0 {
    ctx.JSON(400, gin.H{ "error": "El precio es requerido"})
    return
}
```

// Tener en cuenta

Si necesitamos validar que un valor exista en la base de datos, debemos pasar por las capas de Servicio y Repositorio.

// Tener en cuenta

No podemos ir a la Base de Datos desde el controlador. Debemos pasar primero por el service, y luego del service comunicarse con nuestro repositorio. Para luego hacer el camino inverso con la respuesta de dicha petición.

MANEJO DE RESPUESTAS

GO WEB

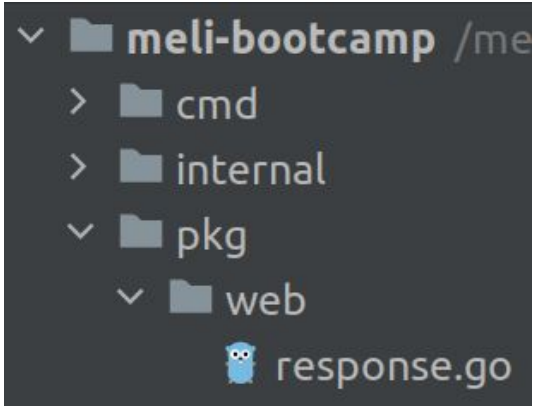
// ¿Cómo realizar un manejo de respuestas standard?

“Implementaremos un paquete que maneje las respuestas que retornaremos al cliente, para que las respuestas tengan la misma estructura (ya sea una respuesta correcta como un error)”

Paquete web

Generaremos un directorio pkg y dentro de él implementaremos el paquete web.

En este paquete implementaremos una lógica para enviar las respuestas con un determinado formato.



Formato de respuestas

Para los errores nos devolverá un campo con el código y un campo con el mensaje de error.

```json
{
   "code": 401,
   "error": "Token invalido"
}
```

Para las respuestas correctas nos devolverá un campo con el código y un campo data con la respuesta esperada.

```json
{
   "code": 200,
   "data": { ... }
}
```

Estructura de respuestas

Definiremos nuestra estructura de respuestas con los campos **Code**, **Data** y **Error**

- **Code**: código de respuesta.
- **Error**: mensaje de error.
- **Data**: entidad en caso que la respuesta sea correcta.

{}

```
type Response struct {
    Code  string      `json:"code"`
    Data  interface{} `json:"data,omitempty"`
    Error string      `json:"error,omitempty"`
}
```

Retorno de Respuestas

Implementaremos una función que reciba el code, data y error, y nos devuelva la estructura **Respuesta** que definimos.

{}

```
func NewResponse(code int, data interface{}, err string) Response {
    // Codigo ...
}
```

Respuesta correcta

En caso que el código sea menor a 300, retornamos una respuesta correcta con el dato y el error vacío.

```
{}
func NewResponse(code int, data interface{}, err string) Response {

    if code < 300{
        return Response{strconv.FormatInt(int64(code), 10), data, ""}
    }
}
```

El código lo recibimos como entero, hacemos la conversión a texto con el paquete **strconv.**

Respuesta incorrecta

Caso contrario (siendo mayor o igual a 300), retornamos una respuesta incorrecta, con el dato en **nil** y el mensaje de error.

{}

```
func NewResponse(code int, data interface{}, err string) Response {

   if code < 300{
      return Response{strconv.FormatInt(int64(code), 10), data, ""}
   }
    return Response{strconv.FormatInt(int64(code), 10), nil, err}
 }
```

Importación de paquete web

Dentro de nuestro Controlador, importamos el paquete web que utilizaremos para manejar las respuestas.

```
{}
import (
   "github.com/gin-gonic/gin"
   "github.com/meli-bootcamp/internal/products"
   "github.com/meli-bootcamp/pkg/web"
 )
```

Retorno de respuestas

Para retornar las respuestas utilizando nuestra funcionalidad, lo hacemos de la siguiente manera.

```
{}  ctx.JSON(401, web.NewResponse(401, nil, "Mensaje de error"))
```

Retorno mensaje error

Recibimos la respuesta con el siguiente formato para todos los errores que puedan ocurrir en nuestra aplicación.

```json
{
   "code": 401,
   "error": "Mi mensaje de error"
}
```

Retorno respuesta correcta

Recibimos la siguiente respuesta en caso que no haya ocurrido ningún error, de esta forma el cliente siempre espera el campo **code** y el campo **data** (en el caso que petición haya sido correcta).

json

```json
{
 "code": "200",
 "data": {
   "id": 1,
   "nombre": "Televisor LCD",
   "tipo": "electrodomésticos",
   "cantidad": 5,
   "precio": 20000
 }
}
```

Respuestas en el Controlador

{}

```
func (c *Product) GetAll() gin.HandlerFunc {
  return func(ctx *gin.Context) {
     token := ctx.Request.Header.Get("token")
     if token != "123456" {
        ctx.JSON(401, web.NewResponse(401, nil, "Token inválido"))
        return
     }
     p, err := c.service.GetAll()
     if err != nil {
        ctx.JSON(500, web.NewResponse(500, nil, err.Error()))
        return
     }
     if len(p) == 0 {
        ctx.JSON(404, web.NewResponse(404, nil, "No hay productos almacenados"))
        return
     }
     ctx.JSON(200, web.NewResponse(200, p, ""))
  }
}
```

{}

```
func (c *Product) Store() gin.HandlerFunc {
   return func(ctx *gin.Context) {
      token := ctx.GetHeader("token")
      if token != "123456" {
         ctx.JSON(401,  web.NewResponse(401, nil, "Token inválido"))
         return
      }
      var req request
      if err := ctx.ShouldBindJSON(&req); err != nil {
         ctx.JSON(400,  web.NewResponse(400, nil, err.Error()))
         return
      }
      if req.Name == "" {
         ctx.JSON(400, web.NewResponse(400, nil, "El nombre del producto es requerido"))
         return
      }
      if req.Type == "" {
         ctx.JSON(400, web.NewResponse(400, nil, "El tipo del producto es requerido"))
         return
      }
      if req.Count == 0 {
         ctx.JSON(400, web.NewResponse(400, nil, "La cantidad es requerida"))
         return
      }
      if req.Price == 0 {
         ctx.JSON(400, web.NewResponse(400, nil, "El precio es requerido"))
         return
      }
      p, err := c.service.Store(req.Name, req.Type, req.Count, req.Price)
      if err != nil {
         ctx.JSON(400,  web.NewResponse(400, nil, err.Error()))
         return
      }
      ctx.JSON(200, web.NewResponse(200,p, ""))
   }
 }
```

// Para concluir

Es importante siempre tener un manejo de respuestas definido, para que el cliente sepa siempre qué esperar

Gracias.

IT BOARDING

BOOTCAMP

Historial de Cambios

Fecha	Version	Autor	Comentarios
27/06/2021	**0.0.1**	**Nahuel Costamagna**	**Creación de documento**

IT BOARDING

BOOTCAMP