INTRODUCCIÓN A LA CLASE

GO WEB

Objetivos de esta clase

- Aprender sobre configuración de entorno (“*.env*”) en GO.
- Aprender a guardar la configuración de entorno de un proyecto en un “*.env*” e implementarlo.
- Manipular un “*.env*” desde GO.
- Comprender cómo hacer el traspaso de guardar datos en memoria a guardarlos en un archivo.
- Comprender cómo leer y modificar información de un archivo.

GODOTENV PACKAGE

GO WEB

// ¿Qué es y para qué sirve?

“Es un package de GO que sirve para poder cargar variables de *environment* (entorno) desde un archivo *.env*.”

INSTALACIÓN:

Para instalar el pkg “go*dotenv*”, desde la consola de comandos, ejecutamos la siguiente instrucción:

```
$ go get -u github.com/joho/godotenv
```

Para utilizarlo, en la raíz de nuestro proyecto se debe crear un archivo con nombre “*.env*”. Aquí un ejemplo:

.env
```
MY_USER=MELI
MY_PASS=BOOTCAMPGO
```

IMPORTACIÓN Y USO:

Ya instalado el go*dotenv* y creado el archivo *.env*, sólo tenemos que importarlo desde nuestra aplicación GO y utilizarlo.

{}

```go
package main

import (
   "github.com/joho/godotenv"
   "log"
   "os"
)

func main() {
 err := godotenv.Load()
 if err != nil {
   log.Fatal("error al intentar cargar archivo .env")
 }
 usuario := os.Getenv("MY_USER")
 password := os.Getenv("MY_PASS")
}
```

Token en Variable Entorno

IT BOARDING

BOOTCAMP

Token en Variable de Entorno

Implementar **godotenv** al proyecto para poder acceder al archivo **.env** que se utilizará al correr el programa de manera local.

```
$ go get -u github.com/joho/godotenv
```

Crear el archivo **.env** juanto al main del proyecto y agregar el Token.

```
$ TOKEN=123456
```

Implementar Dotenv en Main

Se debe implementar la carga del archivo **.env** al inicio de la función main, el método **Load** carga el contenido (del archivo .env) en la variable de entorno.

{}

```
func main() {
    _ = godotenv.Load()
    repo := products.NewRepository()
    service := products.NewService(repo)
    p := handler.NewProduct(service)

    r := gin.Default()
    pr := r.Group("/products")
    pr.POST("/", p.Store())
    pr.GET("/", p.GetAll())
    r.Run()
}
```

Validar Token

{}

```
func (c *Product) GetAll() gin.HandlerFunc {
   return func(ctx *gin.Context) {

        token := ctx.GetHeader("token")

        if token != os.Getenv("TOKEN") {
            ctx.JSON(401, gin.H{
                "error": "token inválido",
            })
            return
        }

        p, err := c.service.GetAll()
 ...
```

IMPLEMENTACIÓN DE NUEVO STORAGE

GO WEB

// ¿Qué se hará en esta sección?

Vamos a trabajar sobre nuestro proyecto para que guarde la información de productos en un archivo json.

IT BOARDING

BOOTCAMP

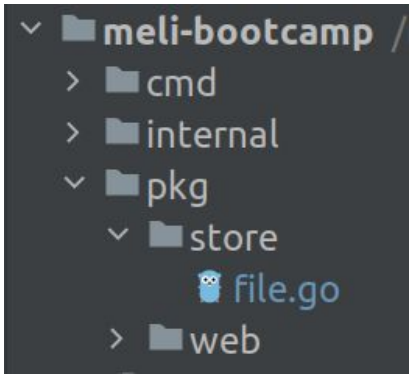
Paquete Store

IT BOARDING

BOOTCAMP

Implementar paquete store

Dentro del directorio **pkg** se genera el paquete **store** y dentro el archivo **file.go**.



En el archivo **file.go** se realizan las importaciones que se utilizarán:

- **os**: Para manipular el archivo json.
- **encoding/json**: Para transformar los datos del archivo en un objeto JSON.

```
{}
import (
    "encoding/json"
    "os"
)
```

Implementar interface Store

Se debe implementar la interface Store con los métodos **Read** y **Write**, ambos métodos reciben una interface y devolverán un error.

```
{}
type Store interface {
    Read(data interface{}) error
    Write(data interface{}) error
}
```

Se debe implementar una constante de tipo **Type** para definir el tipo de **store** que se utilizará, en este caso solo será por archivo (**FileType**).

```
{}
type Type string

const (
    FileType Type = "file"
)
```

Factory de Store

Se debe implementar la función **Factory** que se encarga de generar la estructura que deseamos y recibe el tipo de **store** que queremos implementar y el nombre del archivo.

Se declara la estructura FileStore con el campo que guarde el nombre del archivo.

{}

```
func New(store Type, fileName string) Store {
   switch store {
   case FileType:
       return &FileStore{fileName}
   }
   return nil
}
type FileStore struct {
   FileName string
}
```

Método Write

Se utiliza para escribir datos de la estructura en el archivo. Simplemente recibe una interface y lo convertirá a una representación JSON en bytes para guardarlo en el archivo que especificamos al momento de instanciar la función Factory.

{}

```
func (fs *FileStore) Write(data interface{}) error {
    fileData, err := json.MarshalIndent(data, "", "   ")
    if err != nil {
        return err
    }
    return os.WriteFile(fs.FileName, fileData, 0644)
}
```

Método Read

Sirve para leer el archivo y guardar su contenido empleando la interface que recibirá como parámetro.

```
{}
func (fs *FileStore) Read(data interface{}) error {
    file, err := os.ReadFile(fs.FileName)
    if err != nil {
        return err
    }
    return json.Unmarshal(file, &data)
}
```

Implementar Store en el Proyecto

IT BOARDING

BOOTCAMP

Definir Base de Datos en Repositorio

Dentro de la estructura **repository** se declara el campo de tipo **Store** que se importará del paquete que se generó previamente.

```
type repository struct {
    db store.Store
}
```

Agregar store en la función NewRepository

Dentro de la función que permite inicializar el repository, agregamos cómo argumento el store.

{}

```
func NewRepository(db store.Store) Repository {
    return &repository{
        db: db,
    }
}
```

Guardar Producto

IT BOARDING

BOOTCAMP

Obtener productos del archivo

{}

```
func (r *repository) Store(id int, name, productType string, count
int, price float64) (Product, error) {

    var ps []Product
    r.db.Read(&ps)
    ps = append(ps, p)
}
```

Guardar Producto

{}

```
func (r *repository) Store(id int, name, productType string, count int, price
float64) (Product, error) {

    var ps []Product
    r.db.Read(&ps)
    p := Product{id, name, productType, count, price}
    ps = append(ps, p)
    if err := r.db.Write(ps); err != nil {
        return Product{}, err
    }
    return p, nil
}
```

// Aclaración

Cuando ejecutamos el método Read no recibimos ni controlamos el error, en caso que no pueda obtener productos porque el archivo no existe, el método Write se encargará de crearlo.

Obtener Productos

IT BOARDING

BOOTCAMP

Obtener productos del archivo

Para obtener productos se declara una slice de Productos dentro del scope del método, se le pasará esa variable al método Read que es responsable de cargarla con la información del archivo, la cual finalmente será retornada por el método.

```
{}
func (r *repository) GetAll() ([]Product, error) {
    var ps []Product
    r.db.Read(&ps)
    return ps, nil
}
```

// Aclaración

Al método Read le pasamos la referencia de la variable Productos, para que al momento de ser modificada dentro del método Read también esa modificación se vea reflejada fuera de él.

Obtener último ID

IT BOARDING

BOOTCAMP

Obtener información del archivo

Ahora al no tener más las variables globales de último ID y productos, se debe obtener el último ID del archivo.

Se obtendrá la información de productos guardada, en caso de no existir el archivo, retornará como último ID cero.

{}

```
func (r *repository) LastID() (int, error) {
    var ps []Product
    if err := r.db.Read(&ps); err != nil {
        return 0, err
    }
}
```

Verificar slice de productos

{}

```
func (r *repository) LastID() (int, error) {
    var ps []Product
    if err := r.db.Read(&ps); err != nil {
        return 0, err
    }

    if len(ps) == 0 {
        return 0, nil
    }
}
```

Retornar el último ID

{}

```go
func (r *repository) LastID() (int, error) {
    var ps []Product
    if err := r.db.Read(&ps); err != nil {
        return 0, err
    }
    if len(ps) == 0 {
        return 0, nil
    }

    return ps[len(ps)-1].ID, nil
}
```

Main del programa

IT BOARDING

BOOTCAMP

Enviar Base de datos al repositorio

Instanciamos desde el **Factory de store**, indicando el tipo archivo (**FileType**) y donde deseamos guardar el **json**, y le pasamos la base de datos al **repositorio**.

{}

```
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()
    pr := r.Group("/products")
    pr.POST("/", p.Store())
    pr.GET("/", p.GetAll())
    r.Run()
}
```

Archivo products.json

{}

```
[
    {
        "id": 1,
        "nombre": "Televisor LCD",
        "tipo": "electrodomesticos",
        "cantidad": 5,
        "precio": 20000
    },
    {
        "id": 2,
        "nombre": "Heladera",
        "tipo": "electrodomesticos",
        "cantidad": 1,
        "precio": 2000
    }
]
```

// Para concluir

De esta forma no se perderá la información al bajar el servidor, los productos quedarán guardados en el archivo.

¡Continuemos aprendiendo!

IT BOARDING

BOOTCAMP

Gracias.

IT BOARDING

BOOTCAMP