Bootcamp Go

Práctica 5 Go Web

Objetivo

El objetivo de esta guía práctica es que podamos afianzar los conceptos sobre ***variables de entorno*** y ***manipulación de archivos***, vistos en este módulo. Para esto, vamos a plantear una serie de ejercicios simples e incrementales (trabajaremos y agregaremos complejidad a lo que tenemos que construir) a lo largo del módulo.

Problema

Un supermercado necesita un sistema para gestionar los productos frescos que tienen publicados en su página web. Por este motivo, necesitan un **servidor** que ejecute una API que les permita manipular los productos cargados de distintos clientes. Los campos que conforman un producto son:

Nombre	Tipo de dato **JSON**	Tipo de dato **GO**	Descripción | Ejemplo
id	number	int	Identificador en conjunto de datos | **15**
name	string	string	Nombre caracterizado | **Cheese - St. Andre**
quantity	number	int	Cantidad almacenada | **60**
code_value	string	string	Código alfanumérico característico | **S73191A**
is_published	boolean	bool	El producto se encuentra publicado o no | **True**
expiration	string	string	Fecha de vencimiento | **12/04/2022**
price	number	float64	Precio del producto | **50.15**

¡Atención! Al hacer click aquí verás la solución del ejercicio anterior para que puedas continuar si no lograste terminar.

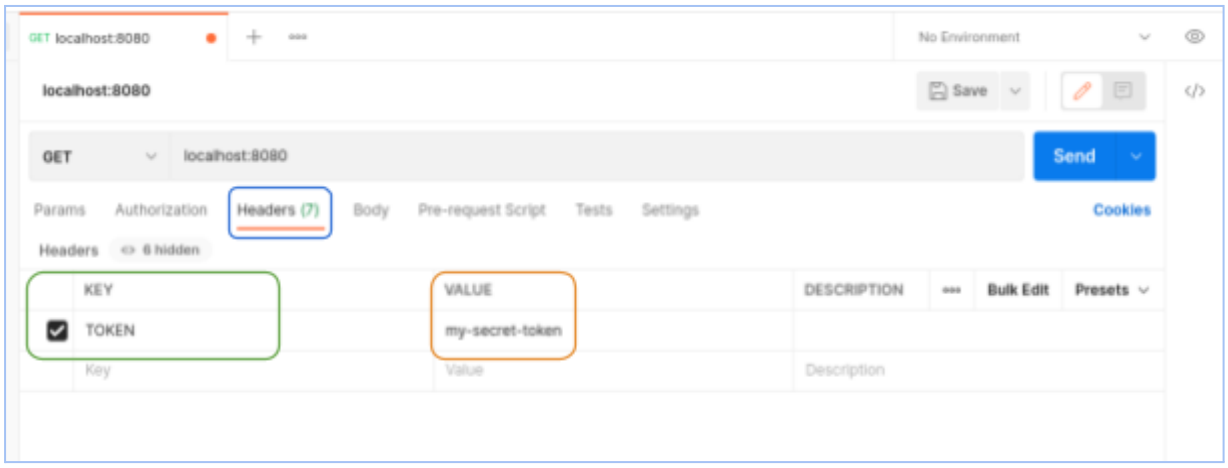
¿Are you ready?

Ejercicio 1: Token de acceso

Vamos a definir un ***token de acceso*** en nuestra API, para eso crearemos una **variable de entorno** que contenga un token secreto. Esta variable se deberá iniciar a partir de un archivo `.env` con el nombre de `TOKEN` y el valor del mismo.

Luego, vamos a implementar un control de acceso a las acciones que modifiquen nuestros datos, utilizaremos los métodos `POST`, `PUT`, `PATCH` y `DELETE`; para esto debemos **leer el *header*** de las request que recibamos en estos métodos y validar que se encuentre el token con el valor que definimos.

En **Postman** podemos agregar contenido al header, veamos cómo: dentro de nuestra consulta debemos hacer clic en la pestaña de **Headers**. Luego, veremos una lista que nos permite agregar campos a la cabecera de nuestra consulta, el campo **KEY** contiene el nombre de la variable y **VALUE** contiene el valor de dicha variable.

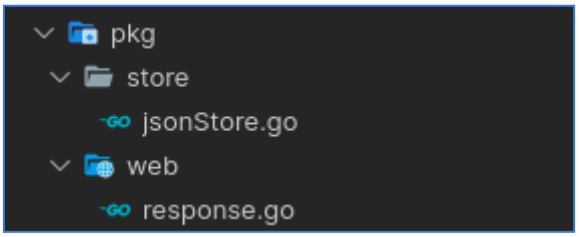


Ejercicio 2: Paquete store

En lugar de trabajar sobre una lista cargada en memoria, ahora tenemos que definir un paquete **store** que nos servirá como interfaz para modificar el archivo **.json** de productos. Este paquete debe estar dentro de una carpeta **pkg** (la misma contiene paquetes de uso general y utilidades para el consumo de la aplicación).

Debe implementar funciones de **inicialización** (verificar que es posible leer el archivo y modificarlo), **búsqueda** (buscar un producto concreto *por id*), **modificación** (actualizar campos de un producto *por id*) y **borrado** (eliminar un producto *por id*).

Este paquete se debe iniciar en el `main.go` de nuestra API y ser utilizado por **Repository**, implementando los respectivos métodos en las interfaces que definimos en el ejercicio anterior.



Ejercicio 3: Manejo de responses

Finalmente definimos un paquete `web` cuya función será definir un estándar para las `response` de nuestra API. Dentro de él, tendremos funciones para las condiciones de éxito y otro para las de fallo —*a estas funciones es buena idea pasarles por argumento un* `c *gin.Context`, con esto podemos acceder a los métodos como `c.Json()` y además trasladar el contexto a la ejecución de las funciones—. Las estructuras son:

```
type errorResponse struct {
   Status  int    `json:"status"`
   Code    string `json:"code"`
   Message string `json:"message"`
}
type response struct {
   Data interface{} `json:"data"`
}
```